UNIT-I

Introduction of IOT:

Definition:

A dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual "things" have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network, often communicate data associated with users and their environments.

1. IOT comprises things that have unique identities and are connected to the internet.

2. while many existing devices such as networked computers are 4G enable mobile phones already have sum form of unique identities and also connected to the internet. The focus on IOT is in the configuration, control and networking via the internet of devices are things that are traditionally not associated with the internet.

3. These include devices such as thermostat, utility meters, a bluetooth connected headset, irrigation pumps and sensors and control. circuits for an electric car enginee.

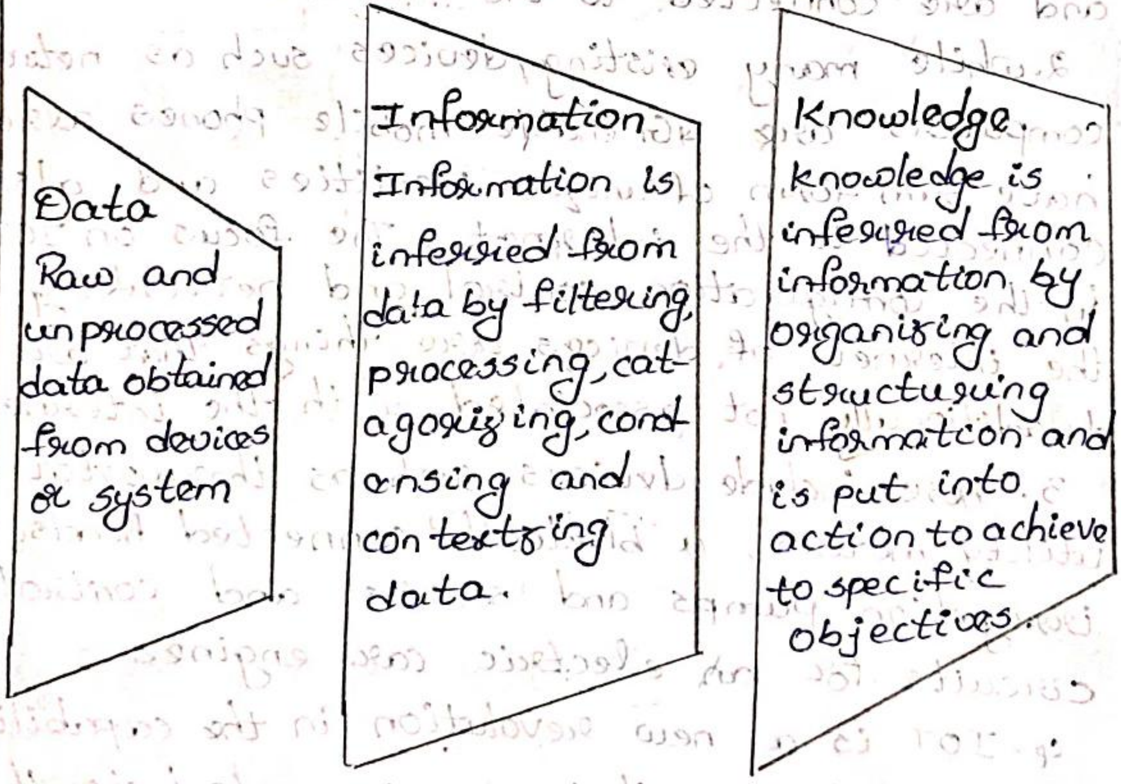
4. IOT is a new revolution in the capabilities of the end point that are connected to the internet and is being driven by advancement in capability [In combination with lower cost]. In sensor network, mobile devices, wireless

communication networking & cloud technologies.

5. The product include hardware and software components for IOT end points, hubs or control centers of the IOT universe.

6. The scope of IOT is not limited to just connecting things [devices, appliances, machines] to the internet.

7. IOT allows these things to communicate and exchange data [control and information that put include data associated with users]. while executing meaningful application towards a common user or machine goal.



Data
Raw and unprocessed data obtained from devices or system
Information
Information is inferried from data by filtering, processing, catagorizing, condensing and contextzing data.
Knowledge
knowledge is inferred from information by organising and structuring information and is put into action to achieve to specific objectives.

Fig: Inferring information and knowledge from data.

Applications of IOT :

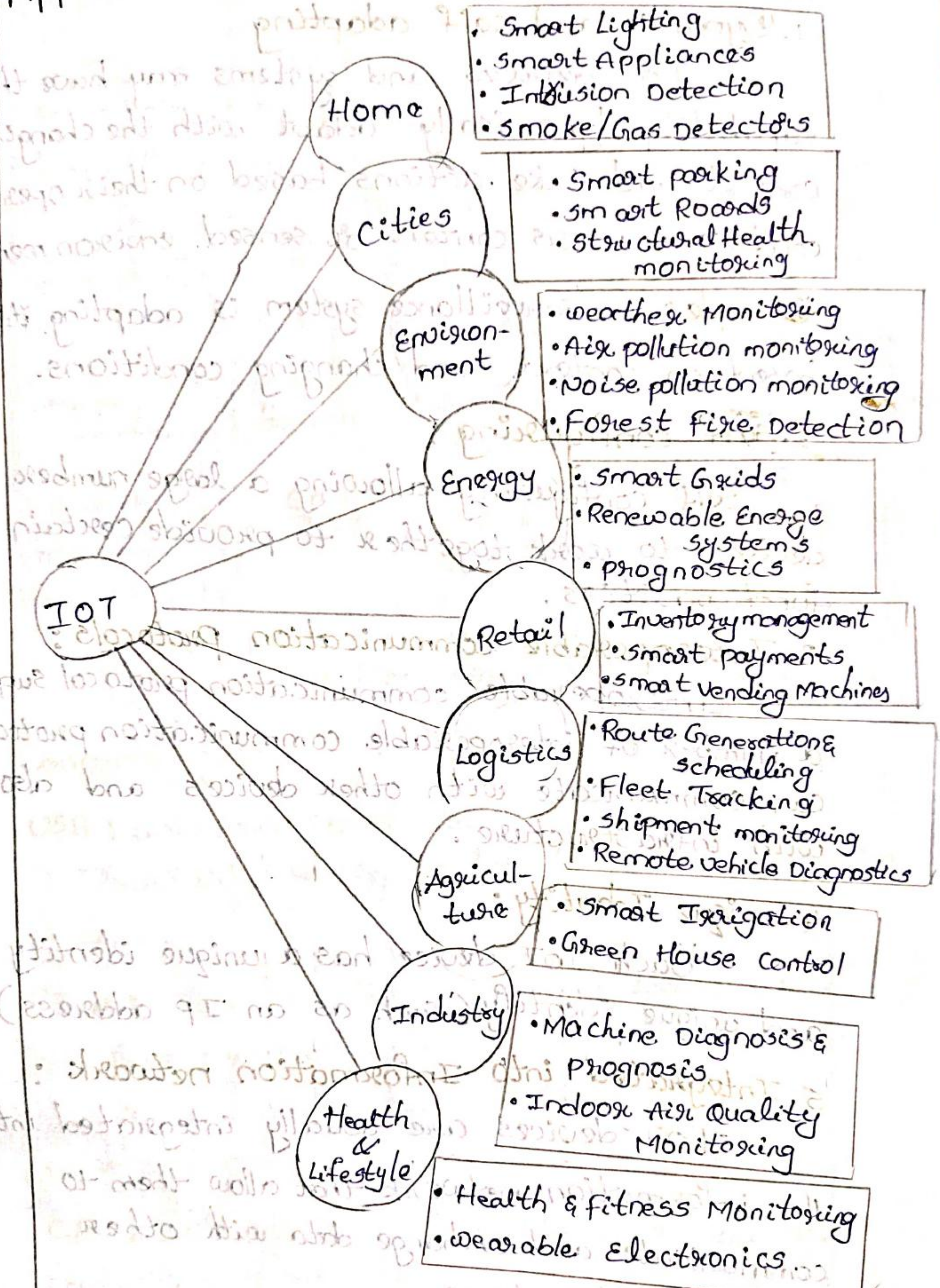


Fig: Applications of IOT

Characteristics of IOT:

1. Dynamic and self adapting

IOT devices and systems may have the capability dynamically adapt with the changing context and take actions based on their operating conditions, users context or sensed environment.

Example: The surveillance system is adapting itself based on context and changing conditions.

2. Self configuring:

Self configuring allowing a large number of devices to work together to provide certain functionalities.

3. Interoperable communication protocols:

Interoperable communication protocol support a number of interoperable communication protocols can communicate with other devices and also with infrastructure.

4. unique Identity:

Each IOT device has a unique identity and unique identify (such as an IP address).

5. Integrated into Information network:

IOT devices are usually integrated into the information network that allow them to communicate and exchange data with other devices and systems.

Physical design of IOT :

1. Things in IOT

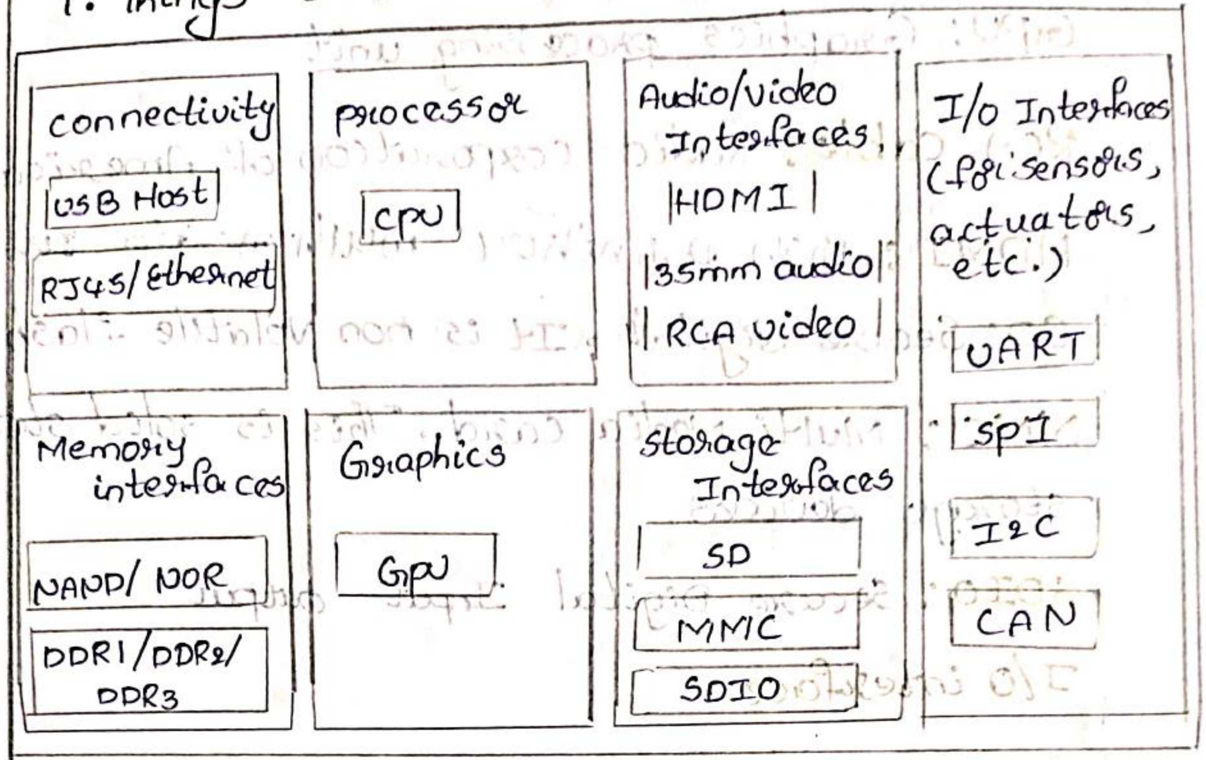


Figure: Generic block diagram of an IOT Device

Connectivity:

USB : universal Serial Bus
These is primarily used to connect a USB device to Host.

RJ45: This is refer to a cable termination specifies physical male and female connectors and the pin assignments of wires in telephone cable and other networks that use RJ45.

CPU: It's performs the Arithematic and logic controlling, input output actions.

NAND/NOR: NOR flash memory is one of two types of non volatile storage technologies.

NAND is the other. non volatile memory doesn't require power to retain data. NOR and NAND different logic gates.

GPU: Graphics processing unit

RCA cable: Radio corporation of America.

HDMI: High Definition multi media interface

SD: Secure digital. It is non volatile flash.

MMC: Multi Media card. This is solid state storage devices.

SDIO: Secure Digital Input output

I/O interface

UART: Universal asynchronus Receiver Transmitor

SPI: Serial pheripheral Interfaces.

I_2C: Inter Integrated circuit.

CAN: Controller Area Networks.

The things in IOT refers to IOT devices which have unique identities and perform remote sensing, activating and monitoring capabilities.

IOT devices can exchange data with other connected devices applications. It collects data from other devices process the data either locally or remotely.

IOT devices may consists of several interfaces for communication to other devices both wired and wireless. This includes I/O interface for sensors, interfaces for internet connectivity. memory and storage interfaces. audio, video interfaces.

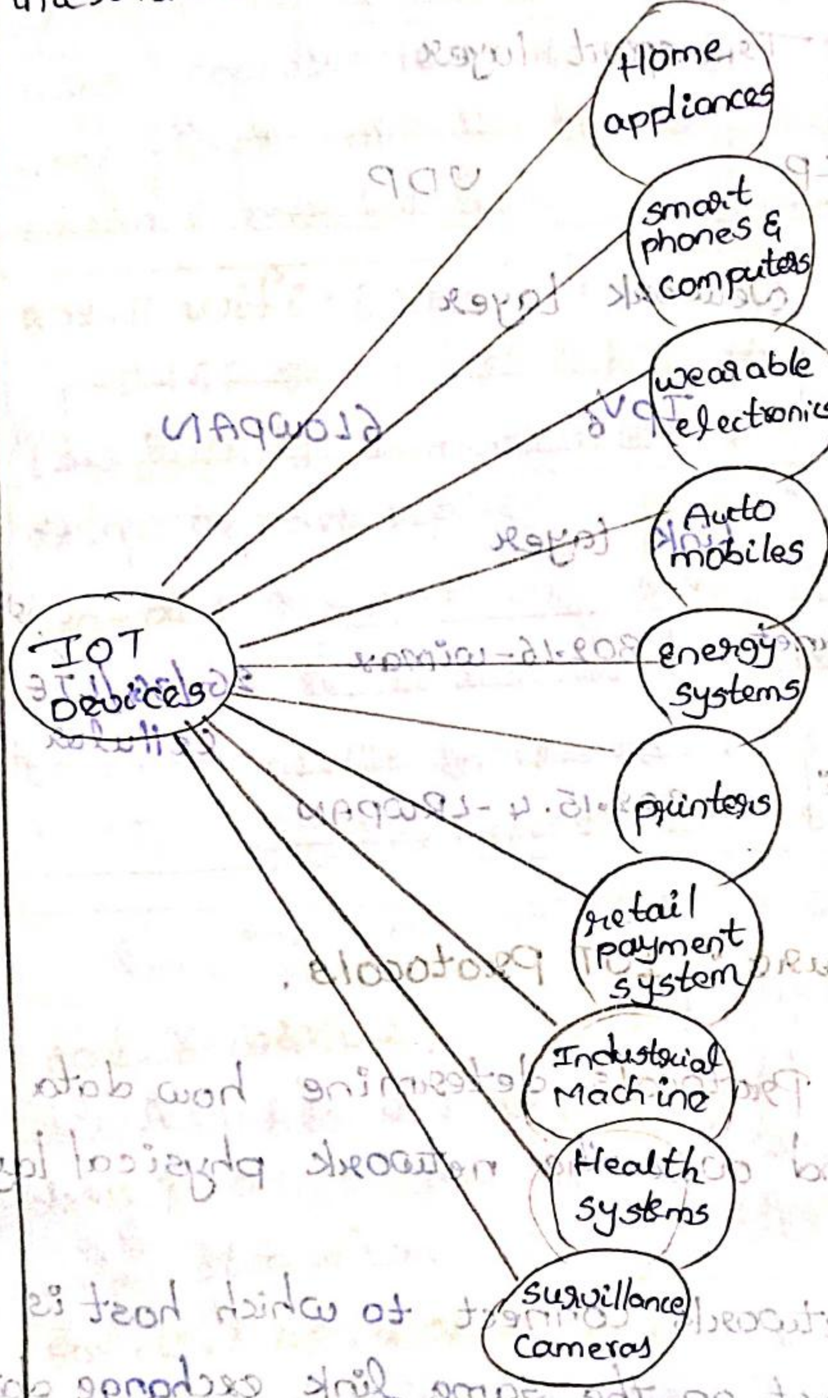


Figure: IOT Devices

IOT protocols:

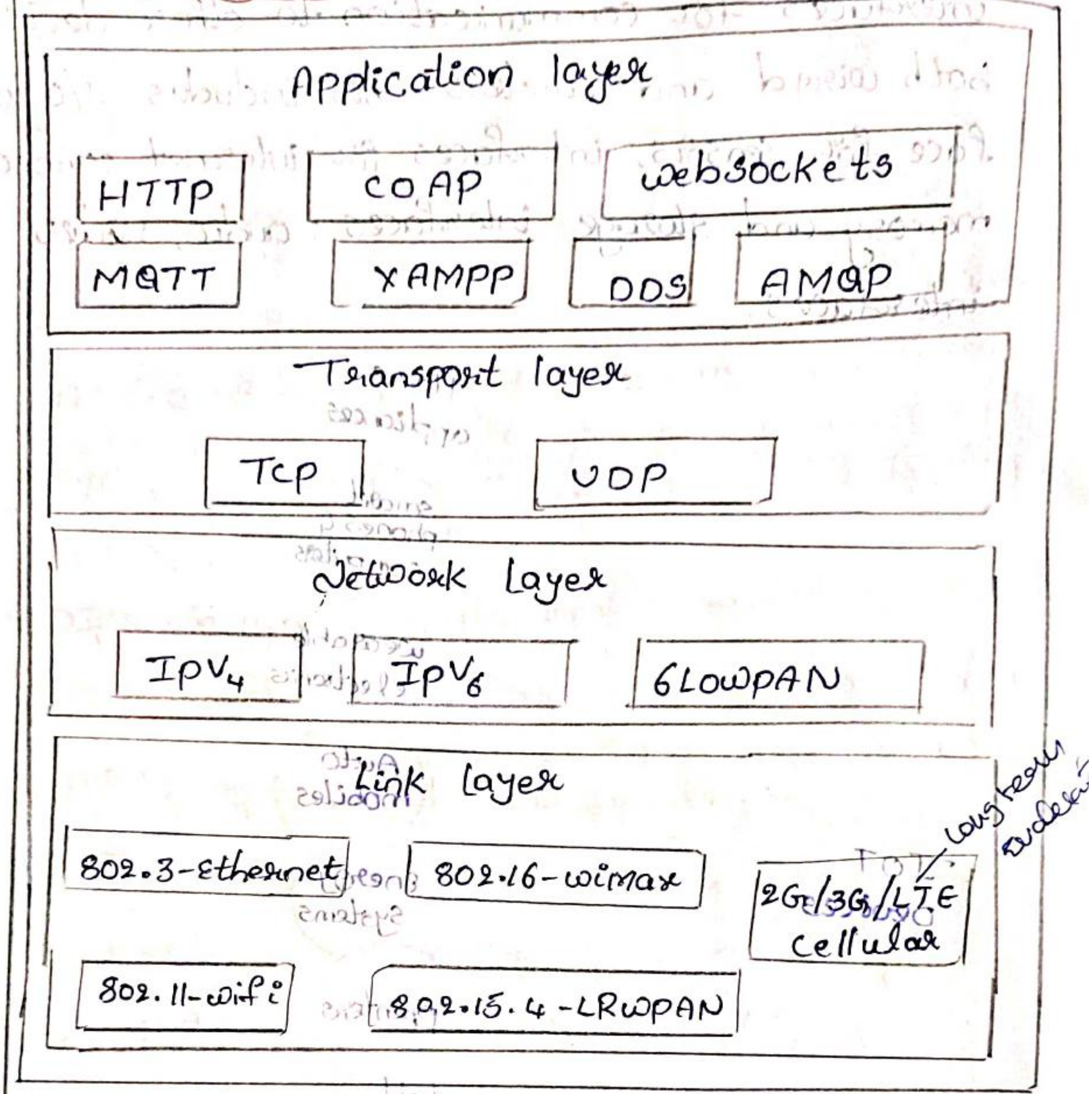


Figure: IOT Protocols.

Link layer: Protocols determine how data is physically send over the network physical layer or medium.

Local network connect to which host is attached, host on the same link exchange data packets over the link layer using link layer protocols.

Link layer determines how packets are coded and signaled by the hardware device over the medium to which the host is attached.

Link layer protocol:

802.3-Ethernet: IEEE 802.3- is collection of wired ethernet standards for link layer example 802.3 uses coaxial cable, 802.3I uses copper twisted pair connection, 802.3J uses fiber optic connection, 802.3 AI uses ethernet over fiber.

802.11 wifi: [wireless fedility]
IEEE 802.11 is a collection of wireless LAN [WLAN] communication standards include extensive description of link layer.
Example: 802.11 A in 5GHz band,
802.11 B and 802.11G operates in 2.4GHz,
802.11 n operates in 2.4/5GHz,
802.11AC operates in 5GHz band,
802.11 AD operates in 60Hz band.

802.16-wiMax: IEEE 802.16 is a collection of wireless broad band standards including exclusive description of link layer. wiMax provides data rated from 1.5 mb/s to 1Gb/s.

802.15.4 LR-WPAN: IEEE 802.15.4 is a collection of standard for low rate wireless personal area network [LR-WPAN] basis for high level communication protocol such as zigbee provide data rate from 40 kbps to 250 kbps.

2G/3G/4G mobile communication :
Data rates from 9.6 kbps [2G] upto 100 Mb/s [4G]

Network layer :
Responsible for sending IP data gram from source network to destination network performs the post addressing and packet routing data grams contains source and destination addressess.

Network layer protocols :
IPV4 : Internet Protocol version - 4 is used to identify the devices on a network using a hierarical addressing scheme. This is 32 bit address allows total of 2**32 bit addressess.

IPV6 : Internet protocol version - 6 uses 128 bit addressess scheme and allows 2**128 bit addressess.

6LOWPAN : IPV6 over low power wireless personal area network. It's operates in 2.4GHz frequency range and data transfer 250 kbps.

Transport layer :
Provides end to end mesage transfer capability independent of the underlined network setup on connection with Ack as in TCP and without ACK as in UDP.
provides functions such as error control, segmentation, flow control and congestion control protocols.

Protocols :

TCP : TCP used by web browsers [alongwith http and https], email [along with SMTP, ftp] connection oriented and stateless protocol.

IP protocols deals with sending packets, TCP ensures reliable transmission of protocols in order. Avoids network congestion and congestion collapse.

UDP : user datagram protocol is connectionless protocol useful in type sensitive application and very small data units to exchange transaction and stateless protocol. does not provide oriented and guarenty delivery.

Application layer :

Defines how the application interface with lower layer protocols to send over the network enables process to process communication using ports.

Http : Hyper Text Transfer protocol that forms foundation of www. Follow request and response model stateless protocol.

COAP : Constrainted Application protocol for machine to machine applications with constrained devices, constrained environment and constrained network using client server architecture.

web socket : It allows full duplex communication over a single socket connection.

MQTT : Message Queue Telementary Transport is light weight messaging protocol based on publish - subscribe model uses client server architecture well suited for constrained environment.

XMPP : Extensable message and presence protocal for real time communication and streaming XML data between network entities. support client server communication.

DDS : Data Distribution Service is data-centric middleware standards for device to device or machine to machine communication uses publish - subscribe model.

AMQP : Advanced Message queuing protocol is open application layer protocol for business messaging supports both point to point and publish subscriber model.

Logical design of IOT :

It refers to an abstract represents of entities and processes without going into the low level specifies of implementation.

(i) IOT Fundamental blocks

(ii) IOT communication model

(iii) IOT communication API's

i) IOT Fundamental blocks:

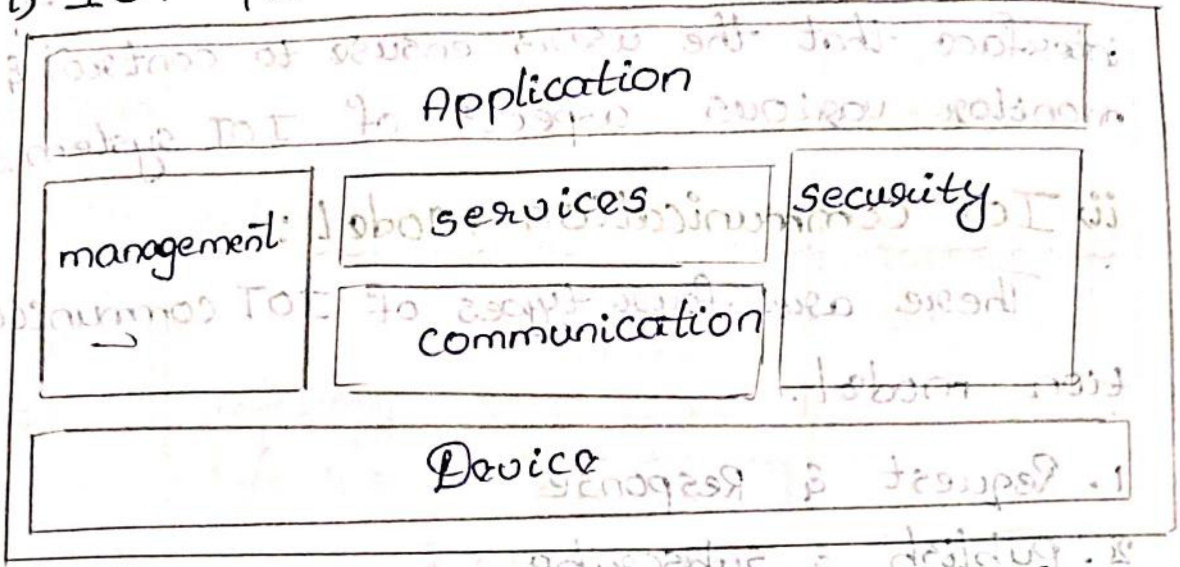


Figure : Functional Blocks of IOT

It provides the system the capabilites for identification, sensing, activation, communication & management.

Device : An IOT system comprises of devices that provide sensing, activation, monitoring control functions.

Communication : communication handles the communication for IOT system.

Services : Services for device monitoring, device control services, data publishing services & services for device discovery.

Management : It provides various functions to govern the IOT system.

Security : Secures IOT system & priority functions such as authentication, authorization, message and context integrity and data security.

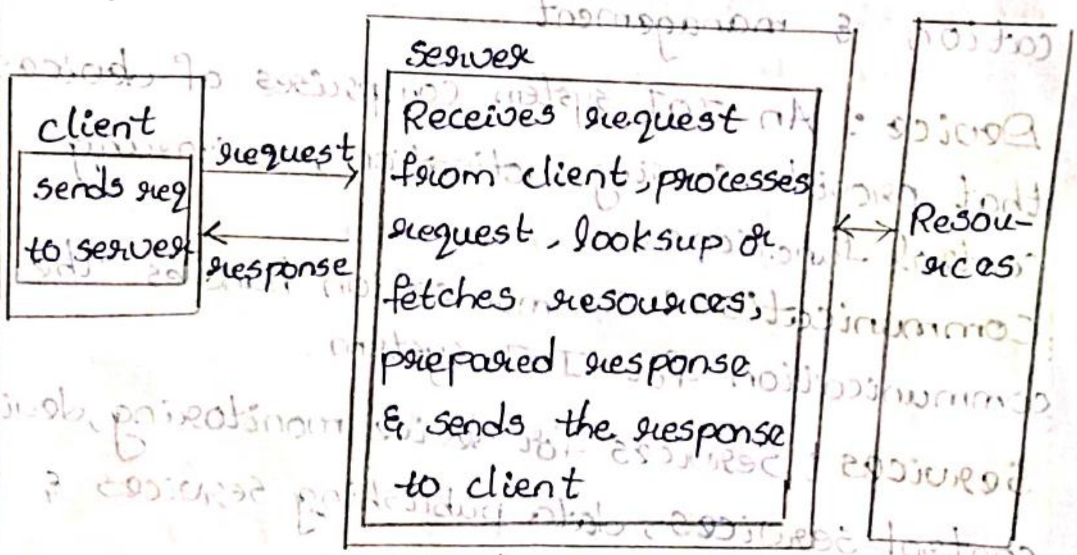
Application : IOT application provide an interface that the users ensuse to control & monitor various aspects of IOT system.

ii) IOT Communication Model :

There are four types of IOT communication model.

1. Request & Response
2. publish & subscribe
3. push & pull
4. exclusive pair

1. Request & Response model :



server
client
sends req to server
request
response
Receives request from client, processes request, looksup or fetches resources; prepared response & sends the response to client
Resources

Figure : Request - Response communication model

In which the client sends request to the server & the server replies to request is a stateless communication model & each request response pair is independent of others.

(ii) publish & subscribe model:

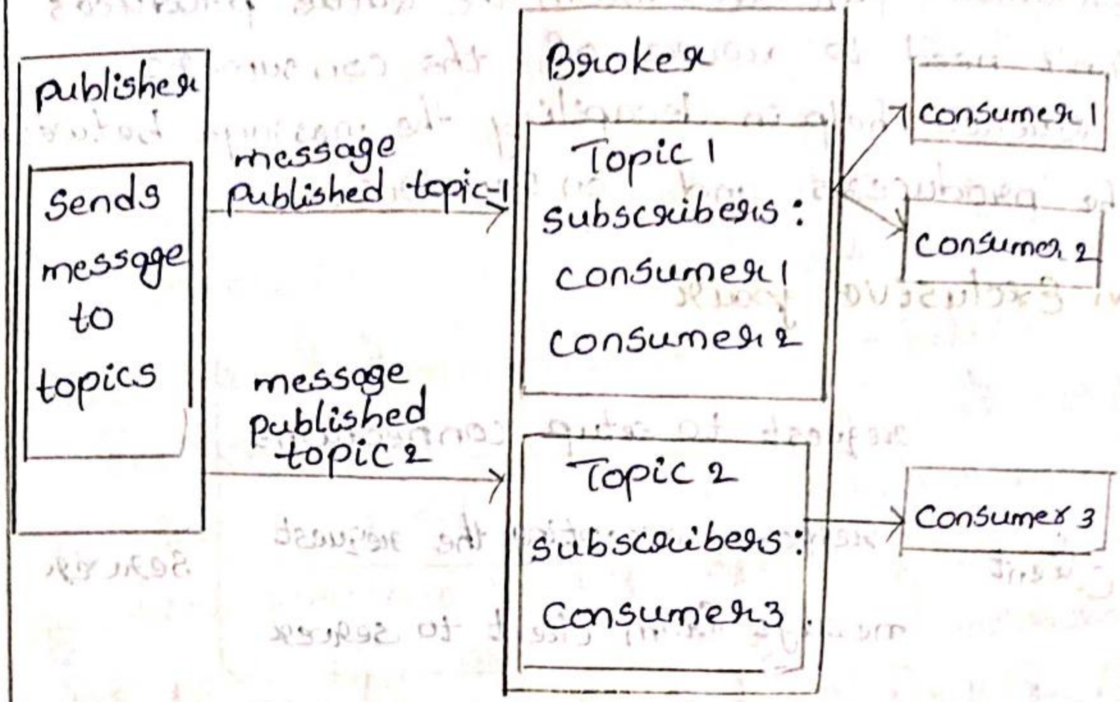

publisher
sends message to topics
message published topic-1
message published topic 2
Broker
Topic 1 subscribers: consumer 1 consumer 2
Topic 2 subscribers: Consumer 3
consumer 1
Consumer 2
Consumer 3

Figure: publish-subscribe communication model

publish and subscribe model involves publishers, brokers & consumers. publishers are source of data. publishers send data to the topics which are managed by the broker. publishers are not aware of the consumers. Consumers subscribe to the topics which are managed by the broker. when the broker receives data for a topic from the publishers. It sends the data to all the subscriber consumers.

(iii) push and pull model:

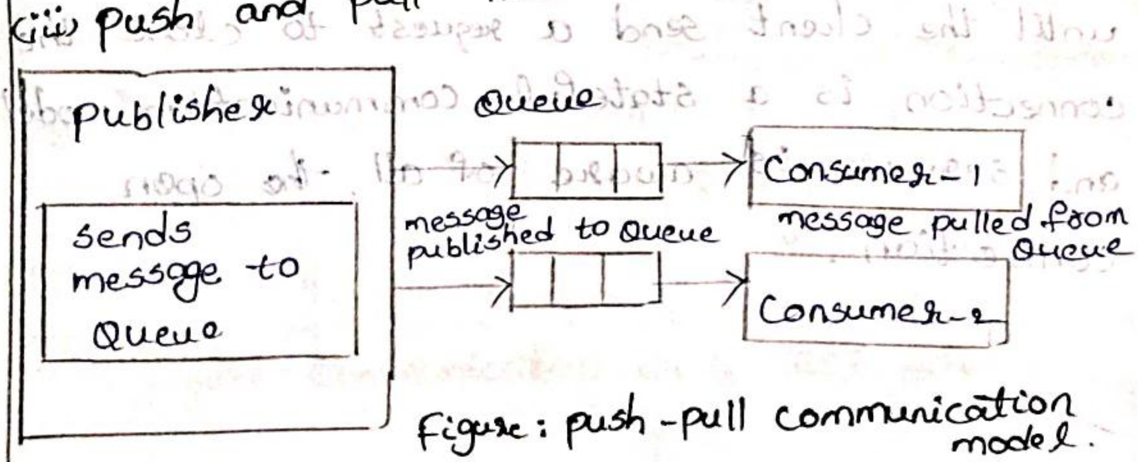
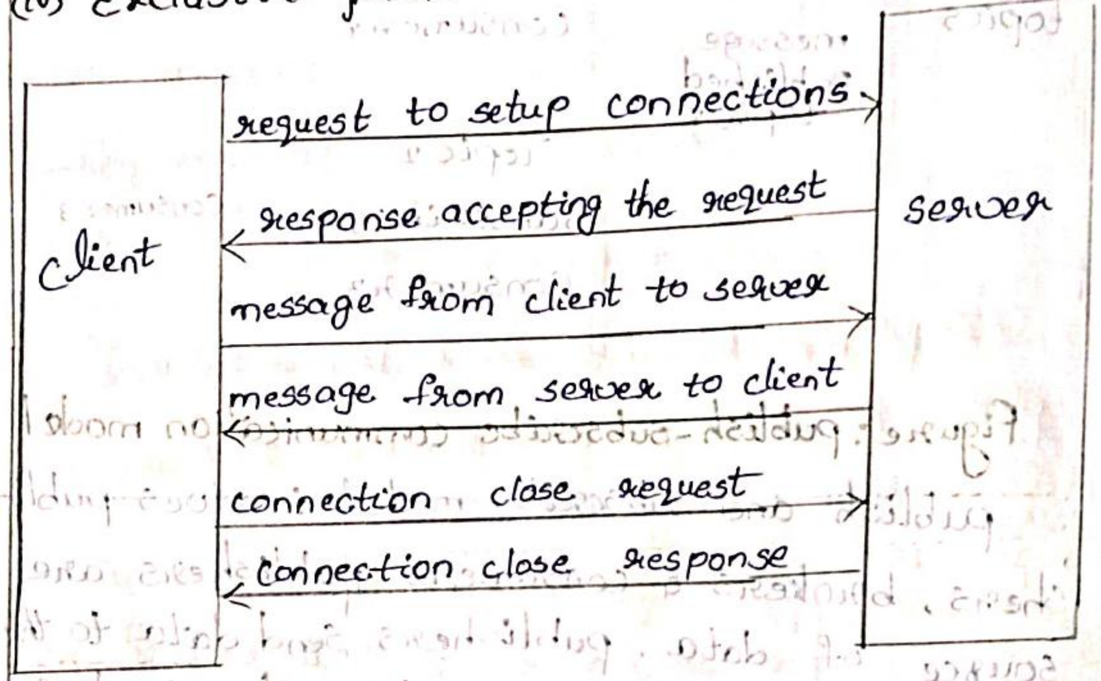


Figure: push-pull communication model.

In which data produces data to Queue and consumers pull data from the queue produces don't need to aware of the consumers.

Queues help in decapiling the message between the producers and consumers.

(iv) Exclusive pair



Client
request to setup connections
response accepting the request
message from client to server
message from server to client
connection close request
connection close response
Server

Figure: Exclusive pair communication model.

Exclusive pair is bidirectional, full duplex communication model that use persisting connection between client and server.

Once connection is setup it remains open until the client send a request to close the connection is a stateful communication model and server is aware of all the open connection.

3. IOT Communication API's :

There are two-types of models

1. Rest Based communication API's model (Rest response based model)
2. web socket communication API's model (exclusive pair based model)

1. Rest based Communication API model:

Rest standards for representational state transfer is a set of architectural principles by which we can design web services and web API's that focus on a systems resources and have resource state or addressed and transfered.

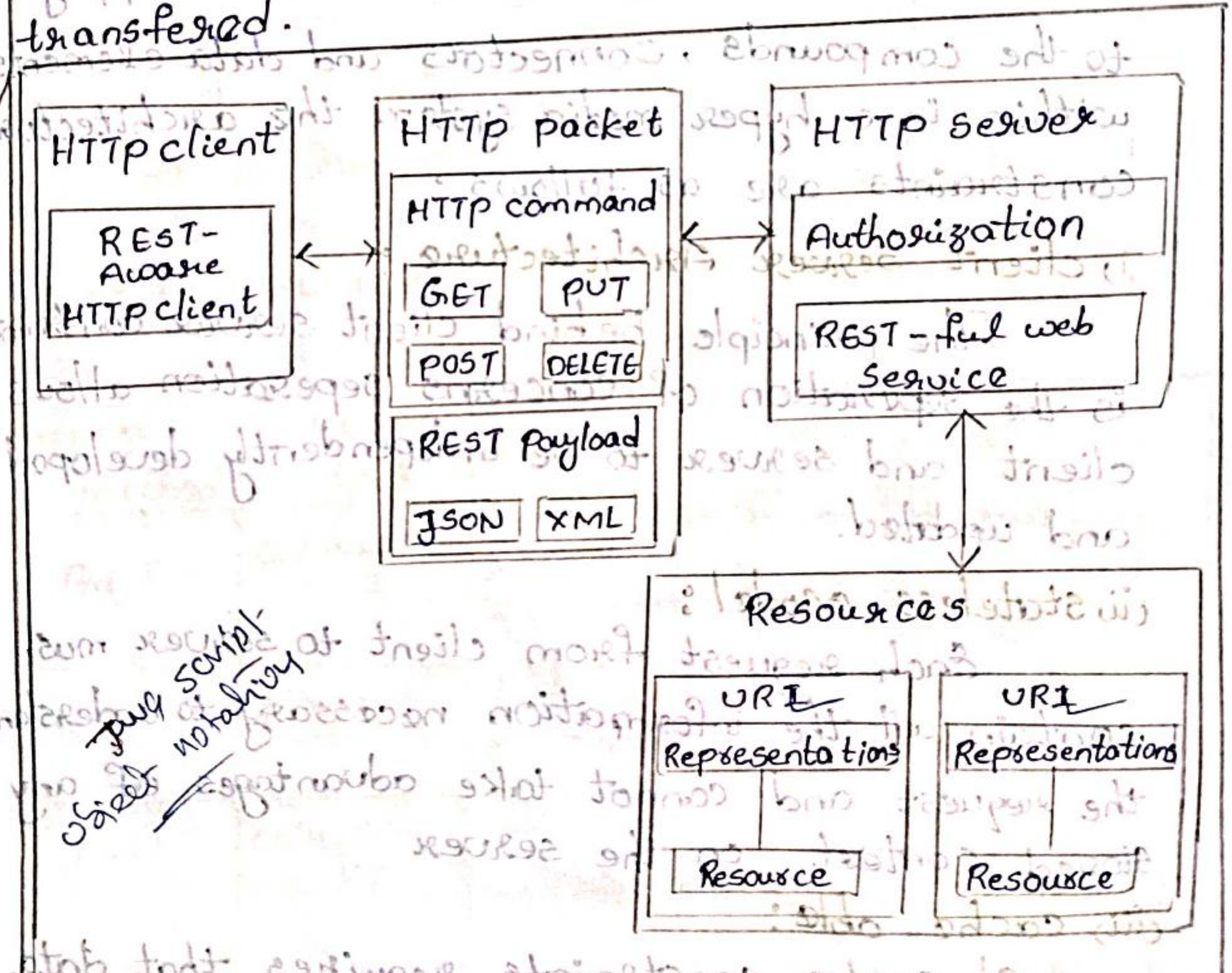


Figure: Communication with REST APIs

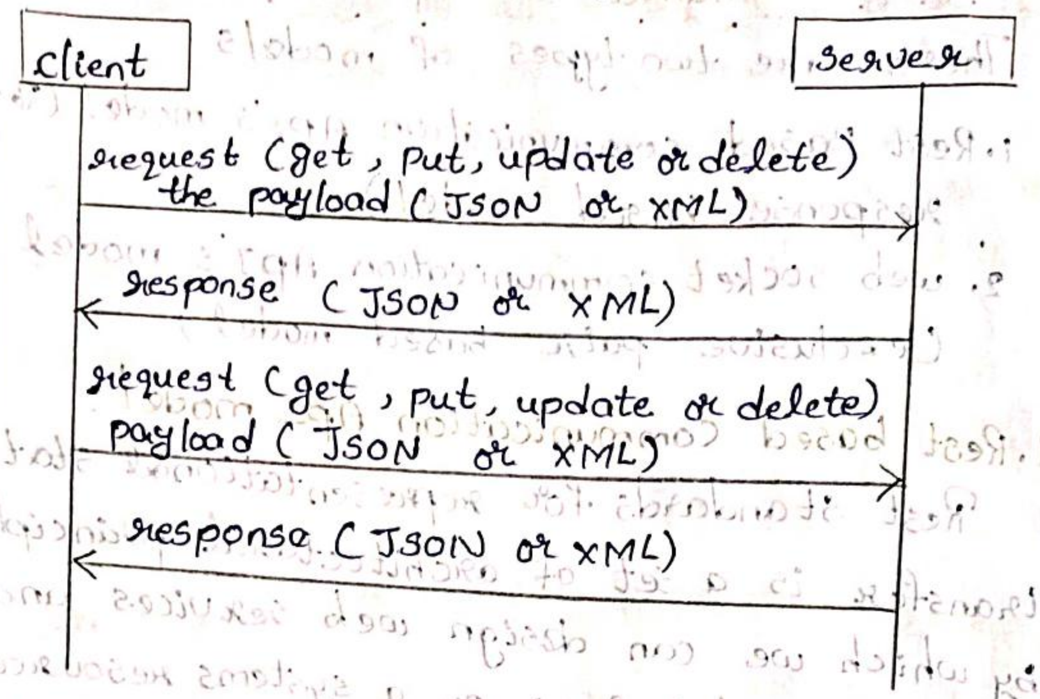

Client
Server
request (get, put, update or delete) the payload (JSON or XML)
response (JSON or XML)
request (get, put, update or delete) payload (JSON or XML)
response (JSON or XML)

Figure : Request -response model used by REST

The rest architectural constraints apply to the compounds, connectors and data elements within in a hyper media system the architectural constraints are as follows :

i, client Server Architecture :

The principle behind client server constraints is the seperation of concerns seperation allows client and server to be independently developed and updated.

ii, stateless model :

Each request from client to server must contain all the information necessary to understand the request and cannot take advantages of any stored context on the server

iii, cache - able :

~~ch~~ cache constraints requires that data within a response to a request be implicitly or explicitly labelled as cache-able or non cache-able.

If a response is cache-able then a client is given the write to reuse that response data for later equivalent request.

iv) layered system:

Constraints the behaviour of components such the each component cannot see beyound the immediate layer with each they are inter-acting.

v) user interface:

Constraints request that the method communication between client and a server must be the uniform.

vi) Code on demand:

server can provide executable code or scripts for client to execute in their context. These constraints is the only that is optional.

HTTP method	Resource type	Action	Example
GET	collection URI	List all the resources in a collection	Http://Example.com /api/tasks/ (list all tasks)
GET	element URI	Get about a resource	Http://Example.com /api/tasks/7/
POST	collection URI	Create a new resource	http://Example.com/ api/tasks (create a new task from data provided in the request)

POST	Element URI	Generally not used	
PUT	collection URI	Replace the entire collection with another collection	http://example.com/api/tasks (replace the entire the collection with data provided in the resource)
PUT	Element URI	update a resource	http://example.com/api/tasks/1 (update task 1)
Delete	collection URI	Delete the entire collection	http://example.com/api/tasks/ (delete the all tasks)
Delete	element URI	delete a resource	http://example.com/api/tasks/1 (delete task 1)

Table: HTTP Request methods and actions

Restfull web services is a collection of resources which are represented by URI's. web API has a base URI(http://Example.com/.api/.tasks/) the client and request to this URI using the methods defined by the http protocol.

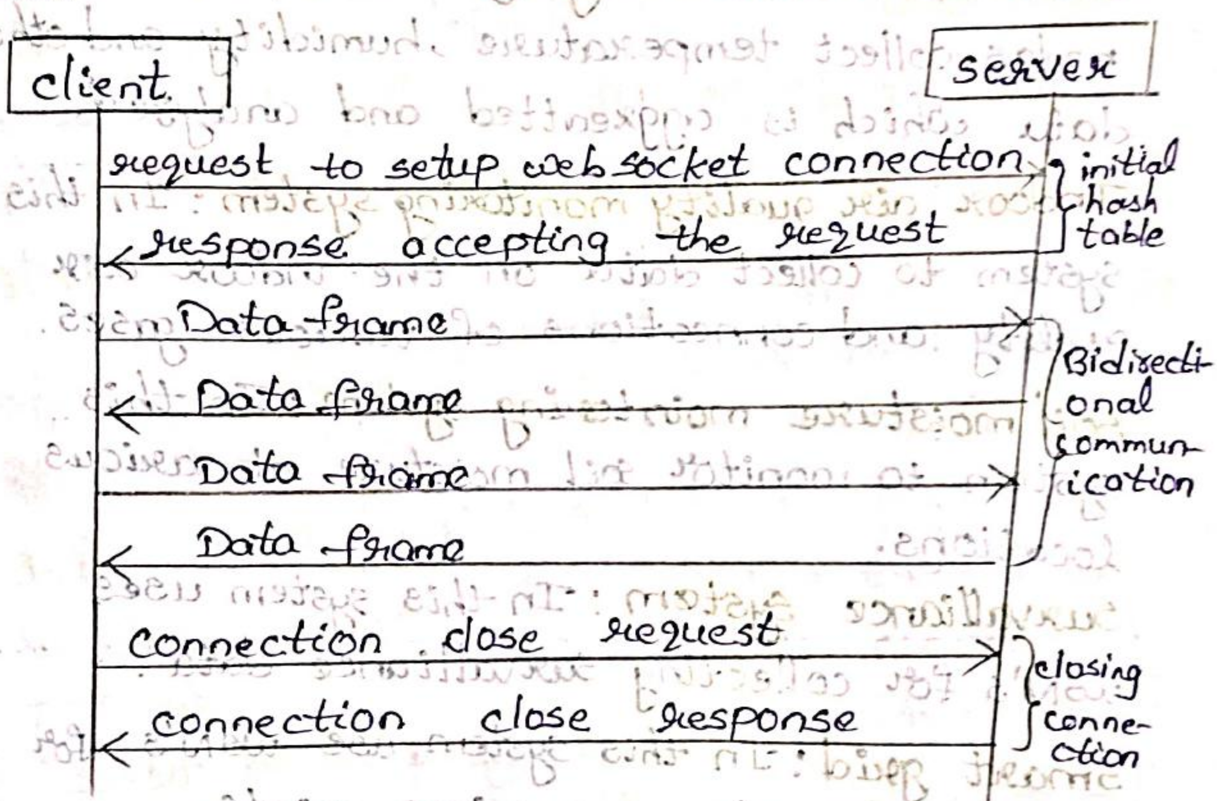
Example : Get, put, post or delete

A restfull web services can support various internet media types.

Web socket based communication API's :

web socket API's allow bidirectional, full duplex communication between client and servers.

web socket API's follow the exclusive pair communication model



client
server
request to setup web socket connection
response accepting the request
initial hash table
Data frame
Data frame
Data frame
Data frame
Bidirectional communication
connection close request
connection close response
closing connection

figure: Exclusive pair model used by websocket API's

IOT enabling Technologies :

IOT is enabled by several Technologies including wireless sensors networks, cloud computing, big data analytics, embedded system, security protocol and architectures, communication protocol, web services, mobile internet and semantic search engines.

1. WSN : wireless sensor networks

WSN compresses of distributed devices with sensors which are used to monitor the environmental and physical condition.

ZigBee is one of the most popular wireless technology used by WSN. WSN's used in IOT system are described as follows

weather monitoring system : In which nodes collect temperature, humidity and other data which is aggreatted and analysed.

Indoor air quality monitoring system : In this system to collect data on the indoor air quality and connections of various gases.

Soil moisture mointoring system : In this system to monitor soil moisture at various locations.

Survalliance system : In this system uses WSN's for collecting survalliance data.

Smart grid : In this system use WSN's for monitoring grids at various points.

structural health monitoring systems : In this system use WSN's to monitor the health of structures [building, bridges] by collecting vibrations from sensors nodes deployed at various points in the structure.

2. Cloud computing :

Services are offered to users in different forms

1. IAAS : Infrastructure as a service

It provides user the ability to provision computing and storage resources. These resources are provided to the users as a virtual machine instances and virtual storage.

2. PAAS :: plat-form as a service

It provides user the ability to develop and deploy application cloud using the development tools, API's, software libraries and services provided by the cloud service provider.

3. SAAS : software as a service

. It provides the user a complete software application are the user interface to the application itself.

3. Big data analytics :

Some examples of big data generated by IOT are

1. sensor data generated IOT system.
2. Machine sensor data collected from sensor established in industrial and energy system.
3. Health and fitness data generated IOT devices.
4. Data generated by IOT systems for location and tracking vehicles.
5. Data generated by retail inventory monitoring system.

4. communication protocol:

Communication protocol from the backbone of IOT systems and enable network connectivity and compiling two application.

* Allow devices to exchange data over network.
* Define the exchange format data encoding addressing schemes for device and rooting of packets from source to destination.
* It includes sequence control, flow control and retransmission of loss packets.

5. Embedded system:

Embedded system is a computer system that has computer hardware and software embedded to perform specific task.

Embedded system range from low cost miniaturized devices such as digital watches to device such as digital cameras, POS terminals, winding machines, appliances and soon etc.

IOT levels and deployment templates:

IOT level-1: system has a single node that perform sensing and or activation, stores data, performs analysis and host the applications as shown in figure below. suitable for modeling low cost and low complexity solutions were the data involved is not big and analysis requirement are not computationally intensive an example of IOT level-1 is Home automation.

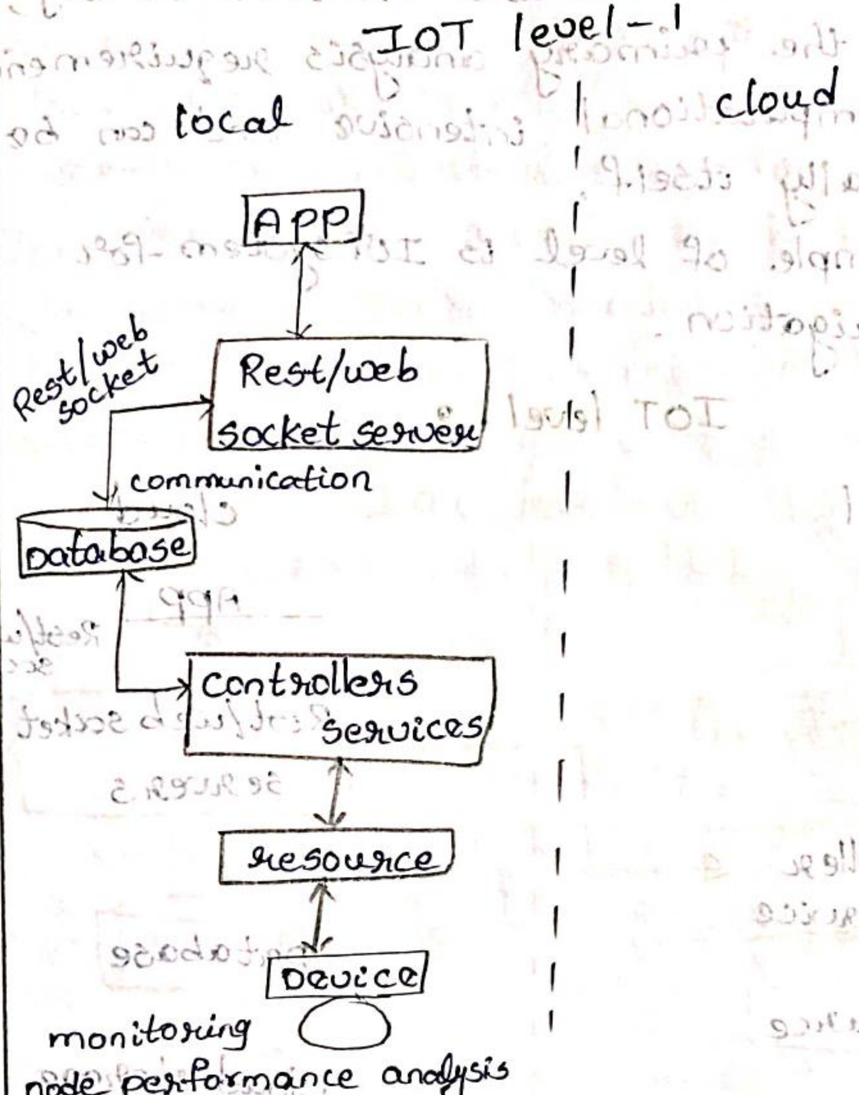


Figure: IOT level-1

Nodes are fixing different locations in house that nodes will be sensing or activating stores the data and analysis the data. The IOT level-1 used in homes in different senario's smart light, home remote services, appliances.

IOT Level-2 :

IOT level-2 has a single node that performs sensing and or activating and local analysis as shown in figure below. Data is stored in cloud and application is usually cloud based.

Level-2 IOT systems are suitable for solutions where data are involved as big; however the primary analysis requirement is not computational intensive and can be done locally itself.

An example of level is IOT system for smart irrigation.

IOT level-2

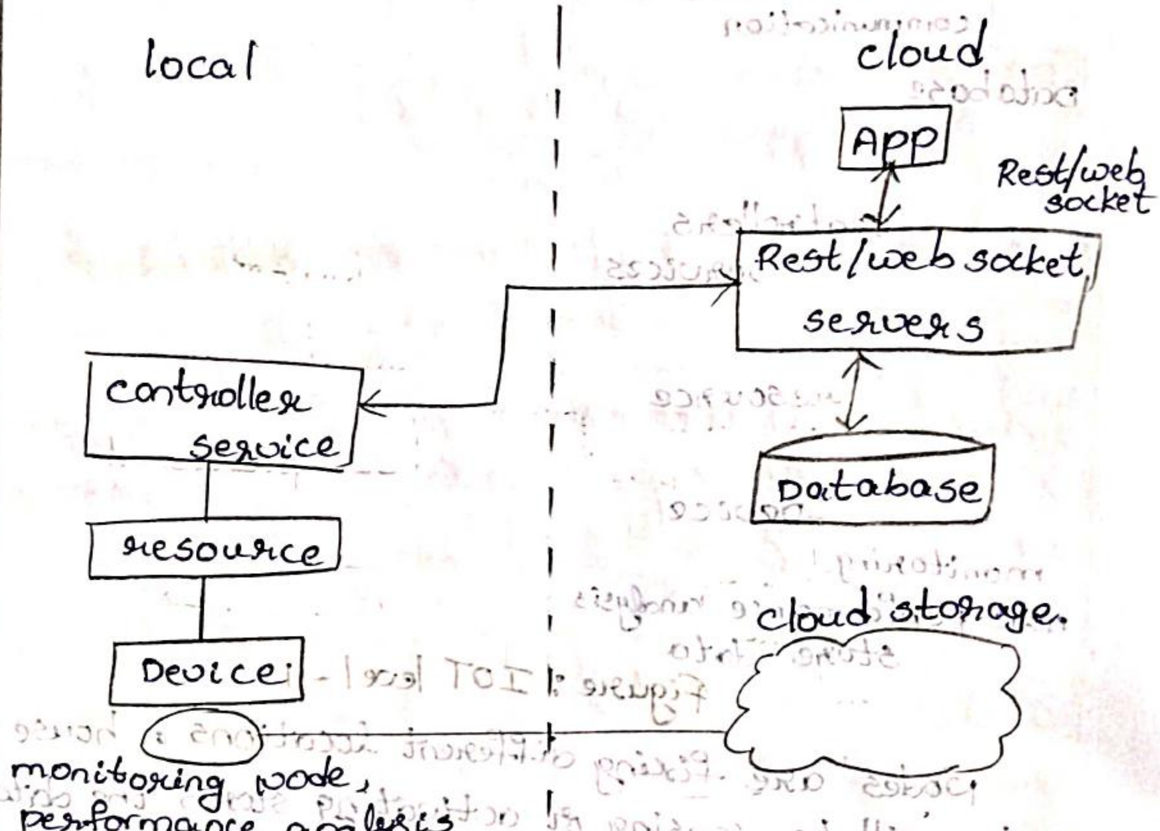


Figure : IOT level-2

nodes are inserted or fixing the different locations in different areas like dams and also lands. The node will be sensing and activating the large storage data and analysis of the data also, the IOT level-2 used in smart irrigation example of soil testing and protecting the dams.

IOT level-3 :

IOT level-3 system has a single node data is stored and analyzed in the cloud application is cloud based as shown fig, below. The IOT level-3 systems are suitable for solutions where the data involved is big and analysis requirements are computationally intensive.

An example of IOT level-3 system for tracking package handling.

IOT level-3

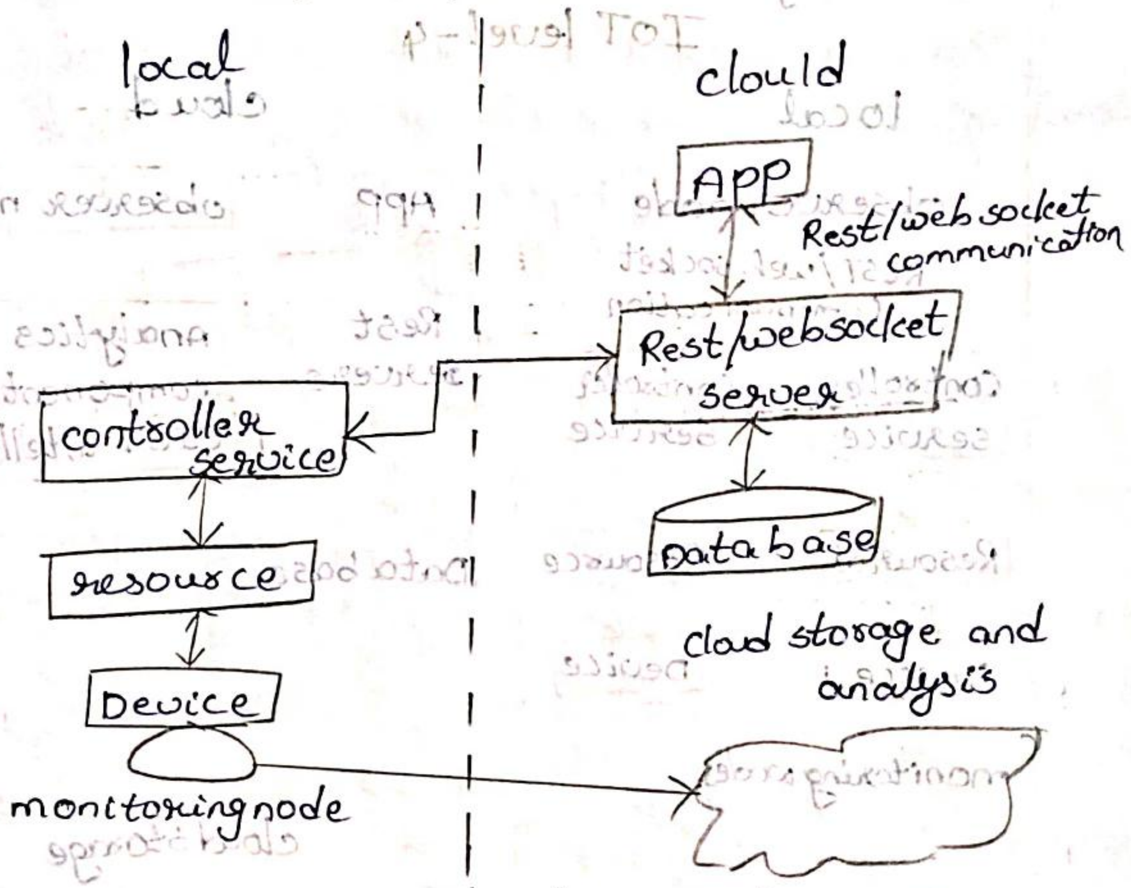


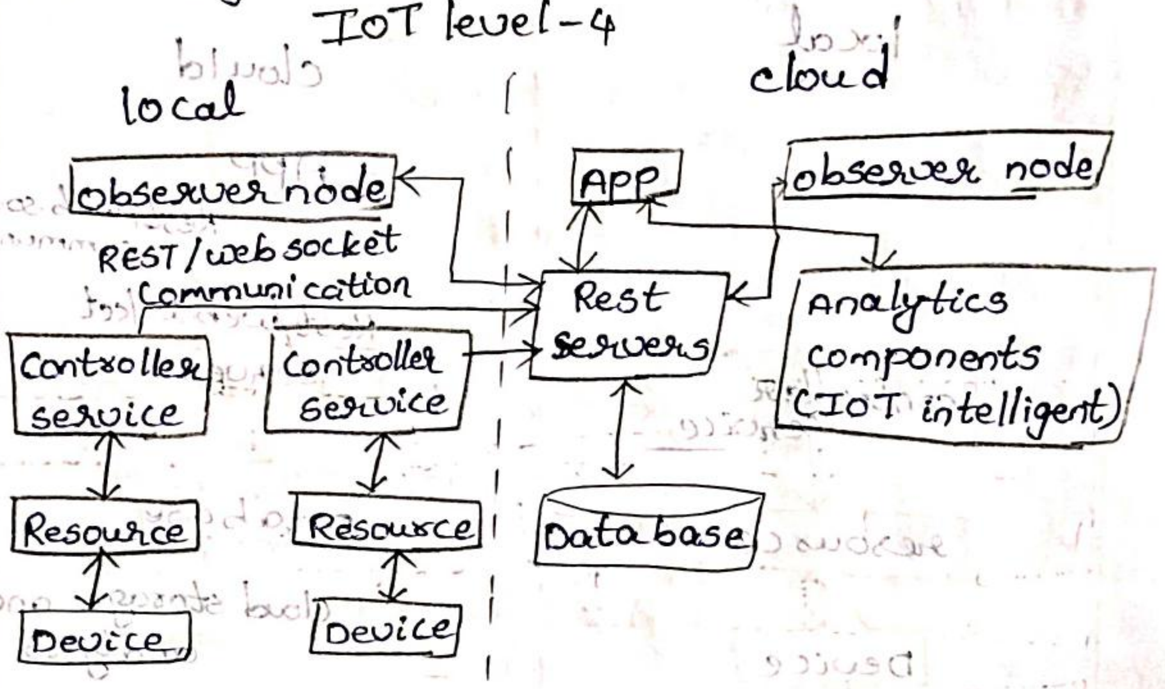
Figure: IOT level-3

IOT level-3 system has a single node that node data will be collected that data will be send to the cloud storage device that cloud storage device analysis of that data. The node data collected vibration levels. The finding, tracking the location of objects or vehicles. Here use the devices accelarometer and gyroscope and sensors.

IOT - level-4 :

IOT level-4 system has multiple nodes that perform local analysis data is stored in the cloud and application is cloud based as shown figure below. IOT level-4 contain local and cloud based observer nodes. which can be subscribe to and recive information collected in the cloud from IOT devices.

An example of a leve-4 IOT system for noise monitoring.



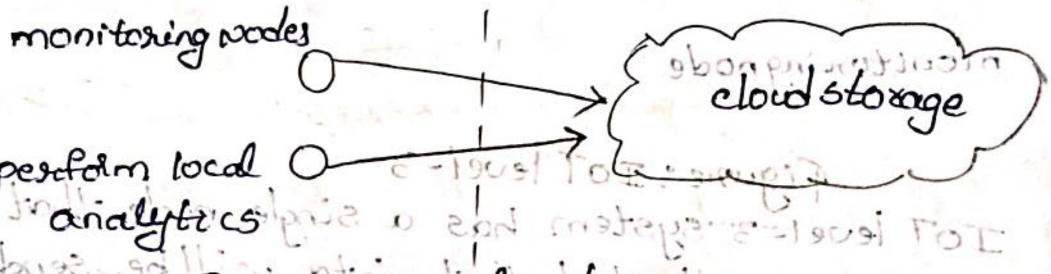


Figure : IOT level-4

IOT level-4 system for using noise mointoring system this system uses the multiple nodes that nodes are fixing in various locations the nodes will be collected the information and also perform the analysis then send to the cloud storage.

IOT level-5 :

IOT level-5 system has multiple end nodes and one coordinator node has shown in fig below. The end nodes that perform sensing and or acti-vation. The coordinator node collects the data from the end nodes and sense to the cloud.

Data is stored and analyzed in the cloud and application is cloud based. IOT level-5 system are suitable for solution based on wireless sensor network in which data is big and analysis requirements are computitionally intensive. An example of level-5 system for forest fire detection :

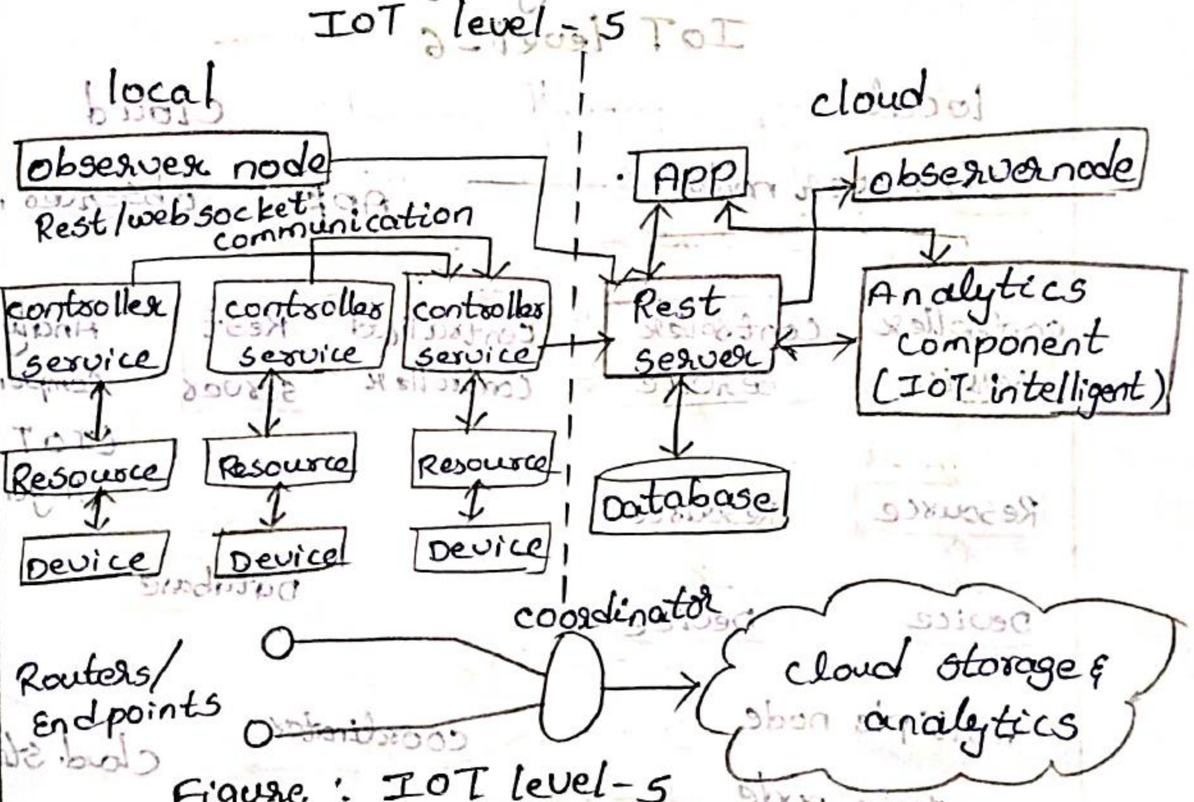


Figure : IOT level-5

IOT level-5 system used in forest fire detection that system monitoring the temperature, humidity and carbon dioxide levels.

IOT Level-6:

The IOT level-6 has multiple independent end nodes that perform sensing and or activation and sensed data to the cloud data is stored in the cloud and application is cloud based as shown figure. The analytics components analysis the data and stores the result in the cloud data base. The results are visualized with cloud based application the centralized controller is aware of the status of all the end nodes and sense control command to nodes an example of a IOT level-6 system is weather monitoring system:

IOT level-6

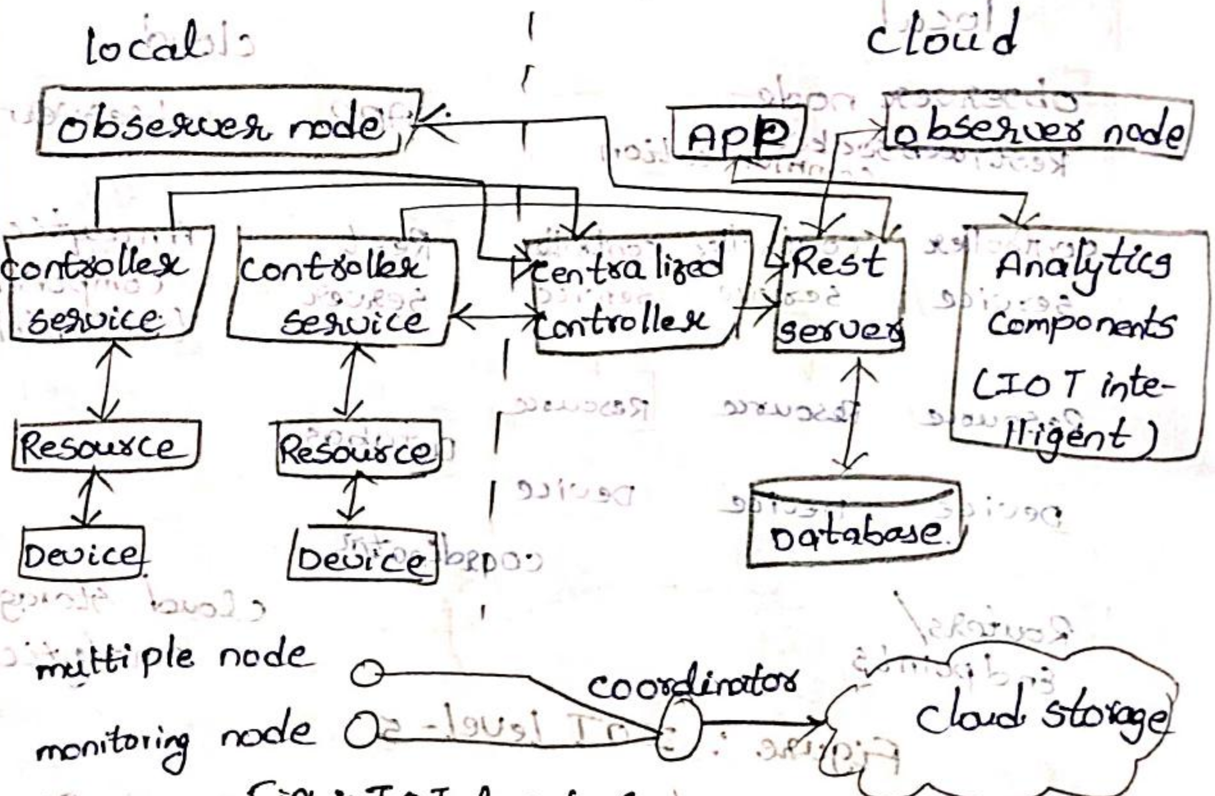


Fig: IOT level-6

IOT level-5 and IOT level-6 both systems are same using ~~the~~ whether monitoring systems and IOT level-6 is same the centralized controller these is mediator between cloud and local.

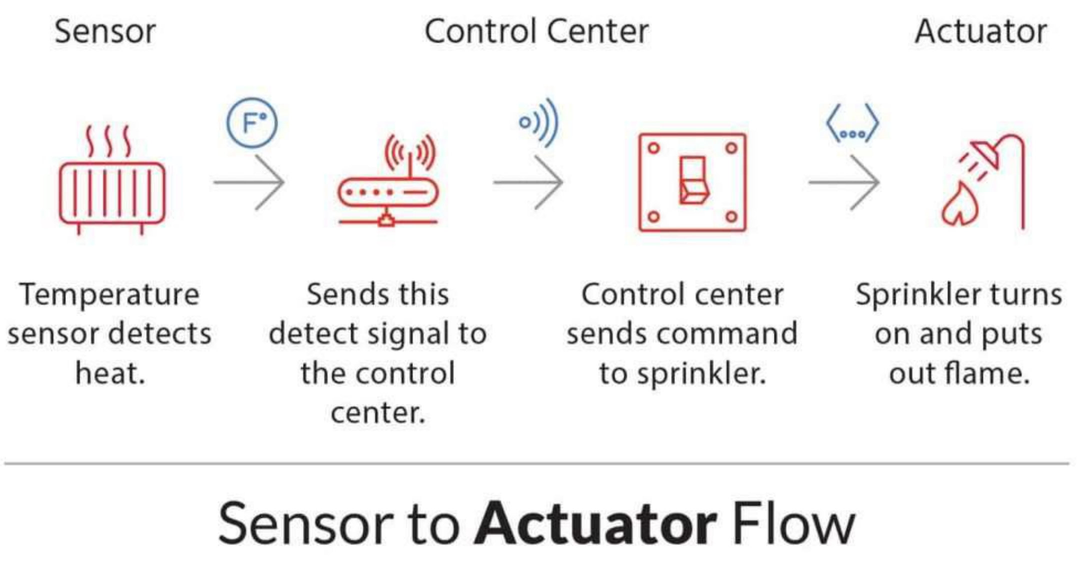
The IOT level-6 system used in weather monitoring system. This is monitoring by temperature, humidity and pressure (pressure of air).

C VENKATA SUBBAIAH
HOD, DPET. OF CSE,
AITS-KADAPA

UNIT II

Prototyping IoT Objects using Microprocessor/Microcontroller

Working principles of sensors and actuators:


Sensor
Control Center
Actuator
F°
Temperature sensor detects heat.
Sends this detect signal to the control center.
Control center sends command to sprinkler.
Sprinkler turns on and puts out flame.
Sensor to **Actuator** Flow

Sensors:
A better term for a sensor is a transducer. A transducer is any physical device that converts one form of energy into another. So, in the case of a sensor, the transducer converts some physical phenomenon into an electrical impulse that determines the reading. A microphone is a sensor that takes vibrational energy (sound waves), and converts it to electrical energy in a useful way for other components in the system to correlate back to the original sound.

Types of sensors –

1. Electrical sensor :
2. Light sensor:
3. Touch sensor:
4. Range sensing:
5. Mechanical sensor:
6. Pneumatic sensor:
7. Optical sensor:
8. Flow sensor
9. Temperature Sensors
10. Voltage sensors
11. Humidity sensor

Electrical sensor :
Electrical proximity sensors may be contact or non contact. Simple contact sensors operate by making the sensor and the component complete an electrical circuit. Non- contact electrical proximity sensors rely on the electrical principles of either induction for detecting metals or capacitance for detecting non metals as well.

Light sensor:
Light sensor is also known as photo sensors and one of the important sensor. Light dependent resistor or LDR is a simple light sensor available today.The property of LDR is that its resistance is inversely proportional to the intensity of the ambient light i.e when the intensity of light increases, it’s resistance decreases and vise versa.

Touch sensor:
Detection of something like a touch of finger or a stylus is known as touch sensor.It’s name suggests that detection of something.

They are classified into two types:

Resistive type
Capacitive type

Today almost all modern touch sensors are of capacitive types.Because they are more accurate and have better signal to noise ratio.

Range sensing:

Range sensing concerns detecting how near or far a component is from the sensing position, although they can also be used as proximity sensors. Distance or range sensors use non-contact analog techniques. Short range sensing, between a few millimetres and a few hundred millimetres is carried out using electrical capacitance, inductance and magnetic technique. Longer range sensing is carried out using transmitted energy waves of various types eg radio waves, sound waves and lasers.

Mechanical sensor:

Any suitable mechanical / electrical switch may be adopted but because a certain amount of force is required to operate a mechanical switch it is common to use micro-switches.

Pneumatic sensor:

These proximity sensors operate by breaking or disturbing an air flow. The pneumatic proximity sensor is an example of a contact type sensor. These cannot be used where light components may be blown away.

Optical sensor:

In there simplest form, optical proximity sensors operate by breaking a light beam which falls onto a light sensitive device such as a photocell. These are examples of non contact sensors. Care must be exercised with the lighting environment of these sensors for example optical sensors can be blinded by flashes from arc welding processes, airborne dust and smoke clouds may impede light transmission etc.

Flow sensor is **a component that measures the flow of a fluid such as a gas or liquid**. Flow sensors utilize both mechanical and electrical subsystems to measure changes in the fluid's physical attributes and calculate its flow. Measuring these physical attributes depends on the fluid's physical attribute.

Temperature Sensors. Temperature sensors measure the amount of heat energy in a source, allowing them to detect temperature changes and convert these changes to data. Machinery used in manufacturing often requires environmental and device temperatures to be at specific levels.

Voltage sensors are wireless tools that can be attached to any number of assets, machinery or equipment. They provide 24/7 monitoring, constantly watching for voltage data that could indicate a problem. Low voltage may signal a potential issue, while other assets may be in danger when voltage is too high.

A humidity sensor is an electronic device that measures the humidity in its environment and converts its findings into a corresponding electrical signal.

The Importance of Accurate Sensors:

Imagine that you are a bar owner and you want to measure the amount of beer coming out of one of your taps. One way you might do this is to install a sensor in line with the line that runs from the keg of beer to the tap. This sensor would most likely have a small impeller inside of it. When the beer ran through the sensor, it would cause the impeller to spin, just like the propeller on a weather station.

When the impeller spins, it will send a stream of electrical impulses to a computer. The computer will interpret the impulses to determine how much beer is flowing through. Sounds simple, right?

This is where sensors get interesting. If you look back at our description, you’ll see that we never directly measured the amount of beer flowing through the sensor; we interpreted it from a stream of electrical impulses. That means that we must first figure out how to interpret it.

Actuators:

Another type of transducer that you will encounter in many IoT systems is an actuator. In simple terms, an actuator operates in the reverse direction of a sensor. It takes an electrical input and turns it into physical action. For instance, an electric motor, a hydraulic system, and a pneumatic system are all different types of actuators.

Types of actuators:

The actuator requires energy. The main types of energy sources are the following:

- electric
- hydraulic
- pneumatic
- thermal/magnetic

Electric actuators, a common option for IoT devices, convert energy into mechanical torque. Electric energy is less noisy in operation than other actuator types. These actuators don't require fluid to run. Additionally, electric actuators offer high-control precision positioning due to programmability. But these actuators can be expensive. They also may not be suitable for extreme operating environments found in some manufacturing, aerospace and military use cases.

Hydraulic actuators can exert a large amount of force and move at a high speed. These characteristics suit use in construction and manufacturing equipment. But they have high maintenance requirements. For example, they can need noise mitigation, and fluid leaks can reduce their performance.

Pneumatic actuators, which use compressed air or gas for energy, have lower maintenance requirements and a longer life span than other types of actuators. They are durable and capable of working in extreme temperatures. This category can also quickly start and stop motion. But they still require some maintenance. For example, they demand a constant air supply, and their efficiency is affected by changes in air and gas pressure.

Thermal or magnetic actuators use energy gained from heating up a shape-memory alloy. They have a compact form factor, are lightweight and have high power density. It also removes the need for a temperature sensor when used in a thermal valve that integrates fluid control, actuation and temperature-sensing functions. Because this actuator uses heat to move, the actuator's piston can shift positions and cause a lag if the device is heating up or cooling down, which causes hysteresis. The actuator's metal can also suffer from structural and functional fatigue.

Controller:

In a typical IoT system, a sensor may collect information and route to a control center. There, previously defined logic dictates the decision. As a result, a corresponding command controls an actuator in response to that sensed input. Thus, sensors and actuators in IoT work together from opposite ends. Later, we will discuss where the control center resides in the greater IoT system.

Setting up the Board:

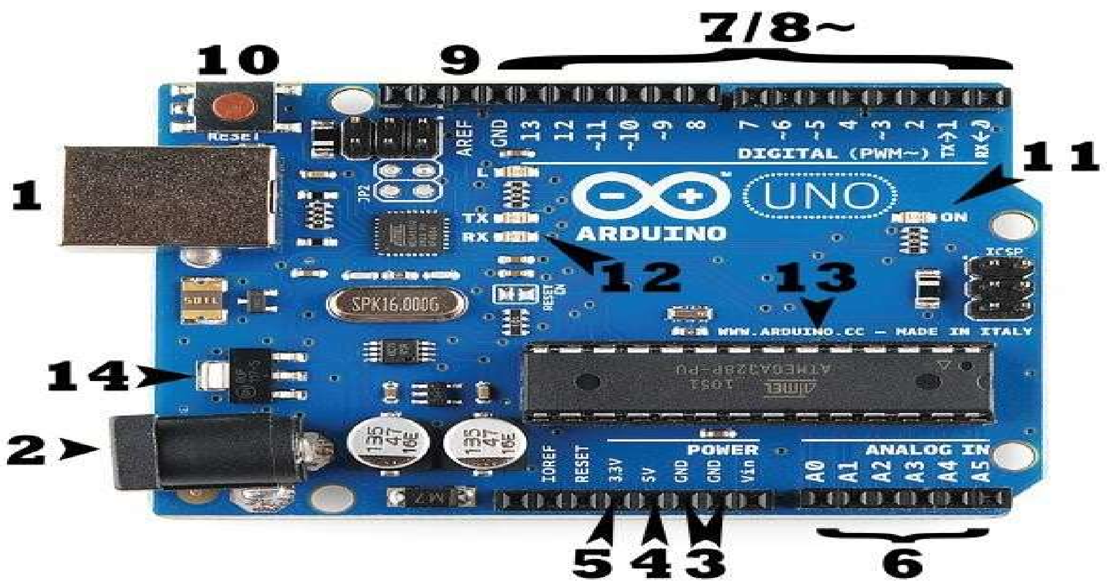
Arduino is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller)and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board. Accepts analog and digital signals as input and gives desired output.

BOARD DETAILS:

- USB or power barrel jack
- LED Power Indicator
- Output power,
- Analog Input Pins
- Power Supply:
- Voltage Regulator
- Tx-Rx LED Indicator
- Ground
- Digital I/O Pin

ARDUIN0 UN0	
Feature	Value
OperatingVoltage	5V
ClockSpeed	16MHz
Digital I/O	14
AnalogInput	6
PWM	6
UART	1
Interface	USB via ATMega16U2

SET UP:



Power (USB / Barrel Jack):
Every Arduino board needs a way to be connected to a power source. The Arduino UNO can be powered from a USB cable coming from your computer or a wall power supply (like this) that is terminated in a barrel jack. In the picture above the USB connection is labeled (1) and the barrel jack is labeled (2). The USB connection is also how you will load code onto your Arduino board.

NOTE: Do NOT use a power supply greater than 20 Volts as you will overpower (and thereby destroy) Arduino. The recommended voltage for most Arduino models is between 6 and 12 Volts.

Pins (5V, 3.3V, GND, Analog, Digital, PWM, AREF):

The pins on your Arduino are the places where you connect wires to construct a circuit (probably in conjunction with a breadboard and some wire. They usually have black plastic 'headers' that allow you to just plug a wire right into the board. The Arduino has several different kinds of pins, each of which is labeled on the board and used for different functions.

GND (3): Short for 'Ground'. There are several GND pins on the Arduino, any of which can be used to ground your circuit.

5V (4) & 3.3V (5): As you might guess, the 5V pin supplies 5 volts of power, and the 3.3V pin supplies 3.3 volts of power. Most of the simple components used with the Arduino run happily off of 5 or 3.3 volts.

Analog (6): The area of pins under the 'Analog In' label (A0 through A5 on the UNO) are Analog In pins. These pins can read the signal from an analog sensor (like a temperature sensor) and convert it into a digital value that we can read.

Digital (7): Across from the analog pins are the digital pins (0 through 13 on the UNO). These pins can be used for both digital input (like telling if a button is pushed) and digital output (like powering an LED.

PWM (8): You may have noticed the tilde (~) next to some of the digital pins (3, 5, 6, 9, 10, and 11 on the UNO). These pins act as normal digital pins, but can also be used for something called Pulse-Width Modulation (PWM). We have a tutorial on PWM, but for now, think of these pins as being able to simulate analog output (like fading an LED in and out).

AREF (9): Stands for Analog Reference. Most of the time you can leave this pin alone. It is sometimes used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.

Reset Button
Just like the original Nintendo, the Arduino has a reset button (10). Pushing it will temporarily connect the reset pin to ground and restart any code that is loaded on the Arduino. This can be very useful if your code doesn't repeat, but you want to test it multiple times. Unlike the original Nintendo however, blowing on the Arduino doesn't usually fix any problems.

Power LED Indicator

Just beneath and to the right of the word "UNO" on your circuit board, there's a tiny LED next to the word 'ON' (11). This LED should light up whenever you plug your Arduino into a power source. If this light doesn't turn on, there's a good chance something is wrong. Time to re-check your circuit!

TX RX LEDs
TX is short for transmit, RX is short for receive. These markings appear quite a bit in electronics to indicate the pins responsible for serial communication. In our case, there are two places on the Arduino UNO where TX and RX appear – once by digital pins 0 and 1, and a second time next to the TX and RX indicator LEDs (12). These LEDs will give us some nice visual indications whenever our Arduino is receiving or transmitting data (like when we're loading a new program onto the board).

Main IC
The black thing with all the metal legs is an IC, or Integrated Circuit (13). Think of it as the brains of our Arduino. The main IC on the Arduino is slightly different from board type to board type, but is usually from the ATmega line of IC's from the ATMEL company. This can be important, as you may need to know the IC type (along with your board type) before loading up a new program from the Arduino software. This information can usually be found in writing on the top side of the IC. If you want to know more about the difference between various IC's, reading the datasheets is often a good idea.p
Voltage Regulator
The voltage regulator (14) is not actually something you can (or should) interact with on the Arduino. But it is potentially useful to know that it is there and what it's for. The voltage regulator does exactly what it says – it controls the amount of voltage that is let into the Arduino board. Think of it as a kind of gatekeeper; it will turn away an extra voltage that might harm the circuit. Of course, it has its limits, so don't hook up your Arduino to anything greater than 20 volts.
ARDUINO IDE OVERVIEW:
Program coded in Arduino IDE is called a SKETCH

1. To create a new sketchFile -> New To open an existing sketch File -> open -> There are some basic ready-to-use sketches available in the EXAMPLES section File -> Examples -> select any program
2. Verify: Checks the code for compilation errors
3. Upload: Uploads the final code to the controller board
4. New: Creates a new blank sketch with basic structure
5. Open: Opens an existing sketch
6. Save: Saves the current sketch

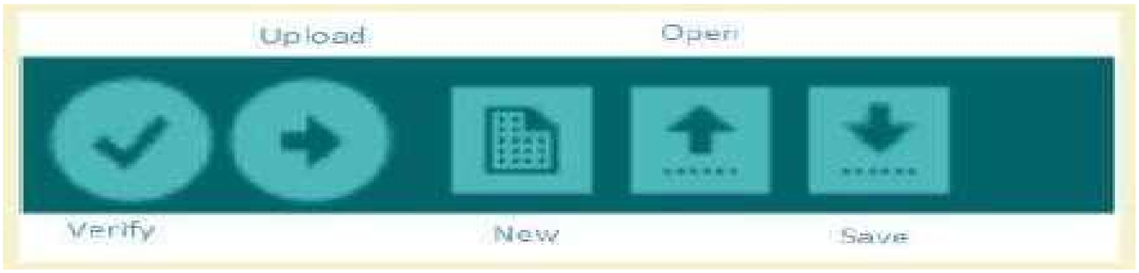
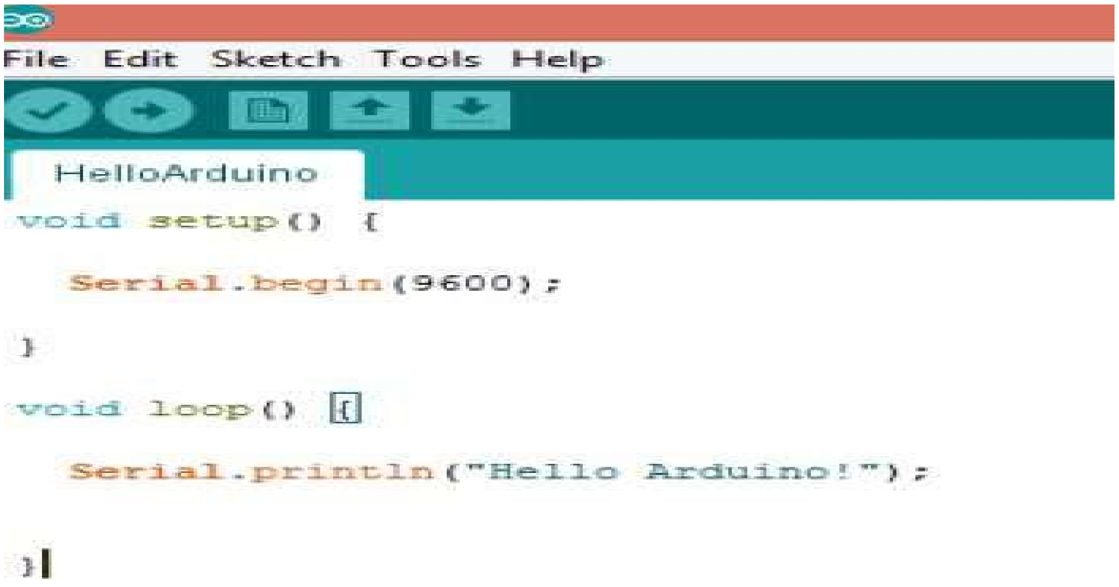


Fig. 2 Compilation and Execution

Serial Monitor: Opens the serial console All the data printed to the console are displayed here.



```
void setup() {
  Serial.begin(9600);
}
void loop() {
  Serial.println("Hello Arduino!");

}
```

Fig : Structure of SKETCH

A sketch can be divided into two parts:

- Setup ()
- Loop()
- The function setup() is the point where the code starts, just like the main() function in C and C++
- I/O Variables, pin modes are initialized in the Setup() function Loop() function, as the name suggests, iterates the specified task in the program.

Programming for IoT using Aurdino / ESP32:

DATA TYPES:

Void, Long, Int, Char, Boolean, Unsigned char, Byte, Unsigned int, Word, Unsigned long

, Float, Double, Array

Arduino Function libraries:

Input/Output Functions:

The arduino pins can be configured to act as input or output pins using the pinMode() function Void setup ()

```
{
pinMode (pin , mode);
}
```

Pin- pin number on the Arduino board Mode- INPUT/OUTPUT digitalWrite() : Writes a HIGH or LOW value to a digital pin

analogRead() : Reads from the analog input pin i.e., voltage applied across the pin

Character functions such as

isdigit(), isalpha(), isalnum(), isxdigit(), islower(), isupper(), isspace() return 1(true) or 0(false)

Delay() function is one of the most common time manipulation function used to provide a delay of specified time. It accepts integer value (time in miliseconds)

EXAMPLE BLINKING LED:

Requirement:

Arduino controller board, USB connector, Bread board, LED, 1.4Kohm resistor, connecting wires, Arduino IDE Connect the LED to the Arduino using the Bread board and the connecting wires Connect the Arduino board to the PC using the USB connector Select the board type and port Write the sketch in the editor, verify and upload Connect the positive terminal of the LED to digital pin 12 and the negative terminal to the ground pin (GND) of Arduino Board.

```
void setup()

{

pinMode(12, OUTPUT); // set the pin mode

} void loop()

{

digitalWrite(12, HIGH); // Turn on the LED delay(1000); digitalWrite(12, LOW); //Turn of
the LED delay(1000);

}
```

Set the pin mode as output which is connected to the led, pin 12 in this case. Use digitalWrite() function to set the output as HIGH and LOW Delay() function is used to specify the delay between HIGH-LOW transition of the output.

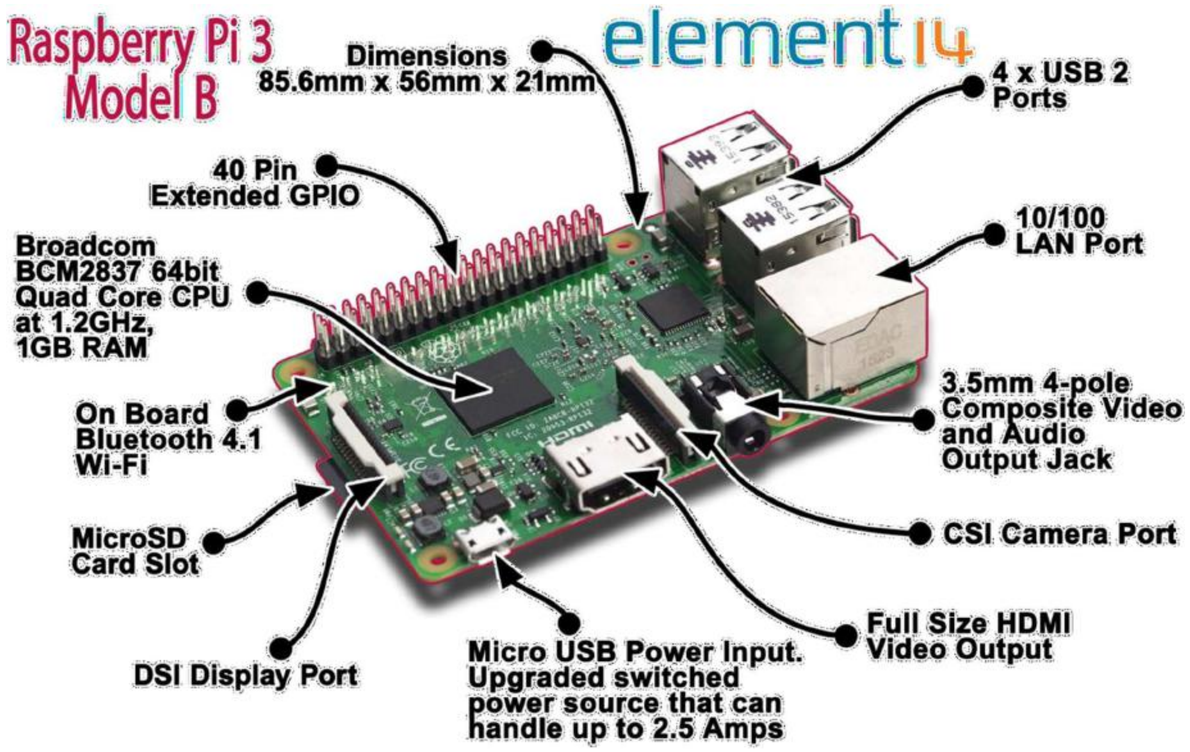
Connect he board to the PC Set the port and board type Verify the code and upload,

notice the TX – RX led in the board starts flashing as the code is uploaded.

Programming for IoT using Raspberry Pi:

RASPBERRY PI:

Raspberry Pi is a credit card sized micro processor available in different models with different processing speed starting from 700 MHz. Whether you have a model B or model B+, or the very old version, the installation process remains the same. People who have checked out the official Raspberry Pi website, But using the Pi is very easy and from being a beginner, one will turn pro in no time. So, it's better to go with the more powerful and more efficient OS, the Raspbian. The main reason why Raspbian is extremely popular is that it has thousands of pre built libraries to perform many tasks and optimize the OS. This forms a huge advantage while building applications.


Raspberry Pi 3 Model B
element14
Dimensions 85.6mm x 56mm x 21mm
4 x USB 2 Ports
40 Pin Extended GPIO
10/100 LAN Port
Broadcom BCM2837 64bit Quad Core CPU at 1.2GHz, 1GB RAM
3.5mm 4-pole Composite Video and Audio Output Jack
On Board Bluetooth 4.1 Wi-Fi
MicroSD Card Slot
CSI Camera Port
Full Size HDMI Video Output
DSI Display Port
Micro USB Power Input. Upgraded switched power source that can handle up to 2.5 Amps

Raspberry Pi Elements

As for the specifications, the Raspberry Pi is a credit card-sized computer powered by the Broadcom BCM2835 system-on-a-chip (SoC). This SoC includes a 32-bit ARM1176JZFS processor, clocked at 700MHz, and a Videocore IV GPU. It also has 256MB of RAM in a POP package above the SoC. The Raspberry Pi is powered by a 5V micro USB AC charger or at least 4 AA batteries (with a bit of hacking). While the ARM CPU delivers real-world performance similar to that of a 300MHz Pentium 2, the Broadcom GPU is a very capable graphics core capable of hardware decoding several high definition video formats. The Raspberry Pi model available for purchase at the time of writing — the Model B — features HDMI and composite video outputs, two USB 2.0 ports, a 10/100 Ethernet port, SD card slot, GPIO (General Purpose I/O Expansion Board) connector, and analog audio output (3.5mm headphone jack). The less expensive Model A strips out the Ethernet port and one of the USB ports but otherwise has the same hardware. Raspberry Pi Basics: installing Raspbian and getting it up and running.

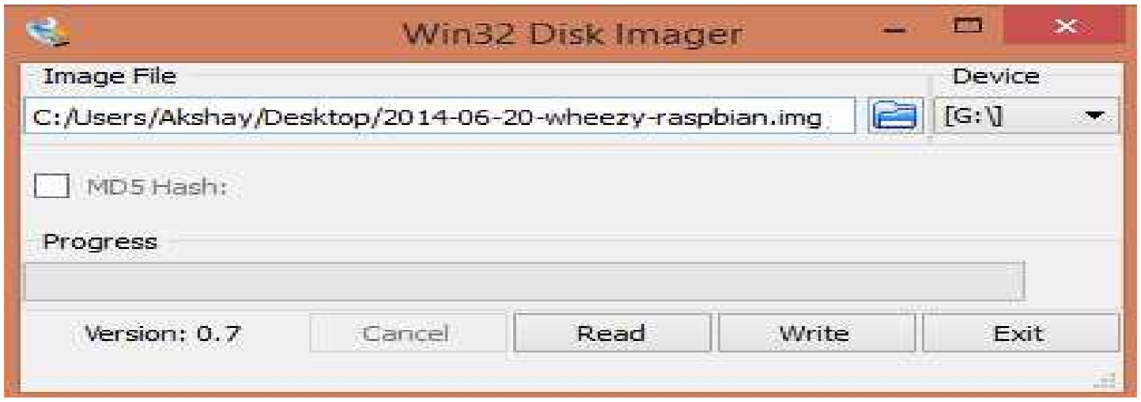
Downloading Raspbian and Image writer:

You will be needing an image writer to write the downloaded OS into the SD card (micro SD card in case of Raspberry Pi B+ model). So download the "win32 disk imager" from the website.

Writing the image

Insert the SD card into the laptop/pc and run the image writer. Once open, browse and select the downloaded Raspbian image file. Select the correct device, that is the drive representing the SD card. If the drive (or device) selected is different from the SD card then the other selected drive will become corrupted. SO be careful.

After that, click on the "Write" button in the bottom. As an example, see the image below, where the SD card (or micro SD) drive is represented by the letter "G:\"



OS Installation

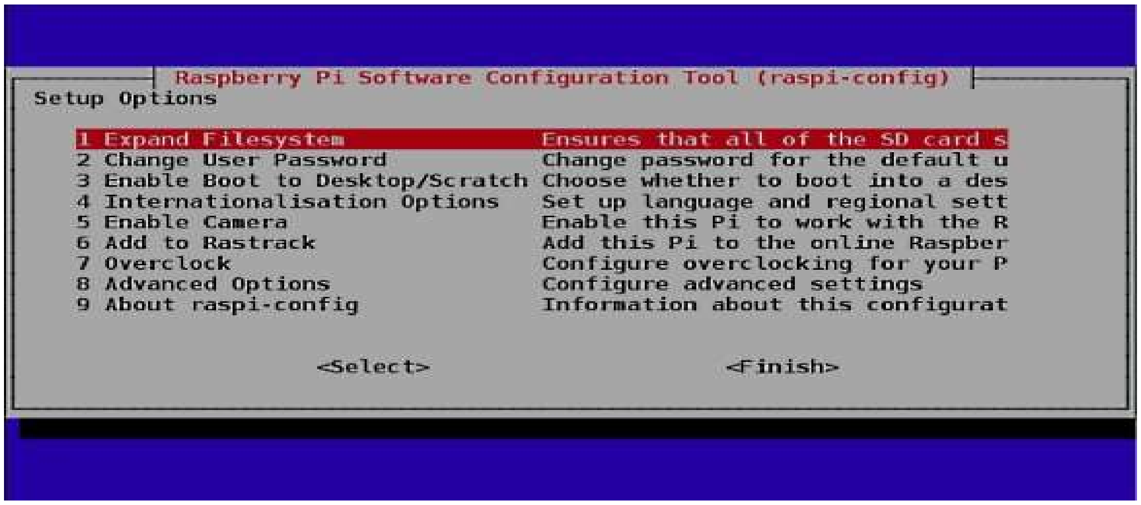
Once the write is complete, eject the SD card and insert it into the Raspberry Pi and turn it on. It should start booting up.

Setting up the Pi

Please remember that after booting the Pi, there might be situations when the user credentials like the "username" and password will be asked. Raspberry Pi comes with a default user name and password and so always use it whenever it is being asked. The credentials are:

login: pi
password: raspberry

When the Pi has been booted for the first time, a configuration screen called the "Setup Options" should appear and it will look like the image below.



```
Raspberry Pi Software Configuration Tool (raspi-config)
Setup Options

  1 Expand Filesystem              Ensures that all of the SD card s
  2 Change User Password           Change password for the default u
  3 Enable Boot to Desktop/Scratch Choose whether to boot into a des
  4 Internationalisation Options   Set up language and regional sett
  5 Enable Camera                  Enable this Pi to work with the R
  6 Add to Rastrack                Add this Pi to the online Raspber
  7 Overclock                      Configure overclocking for your P
  8 Advanced Options               Configure advanced settings
  9 About raspi-config             Information about this configurat

              <Select>                     <Finish>
```

Raspberry Configuration

If you have missed the "Setup Options" screen, its not a problem, you can always get it by typing the following command in the terminal.

sudo raspi-config

Once you execute this command the "Setup Options" screen will come up as shown in the image above.

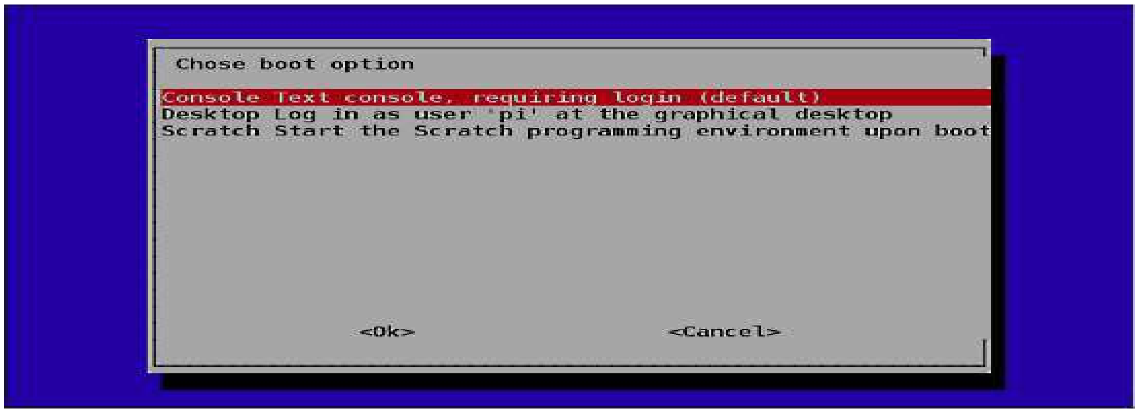
Now that the Setup Options window is up, we will have to set a few things. After completing each of the steps below, if it asks to reboot the Pi, please do so. After the reboot, if you don't get the "Setup Options" screen, then follow the command given above to get the screen/window.

The first thing to do:

select the first option in the list of the setup options window, that is select the "Expand Filesystem" option and hit the enter key. We do this to make use of all the space present on the SD card as a full partition. All this does is, expand the OS to fit the whole space on the SD card which can then be used as the storage memory for the Pi

The second thing to do

Select the third option in the list of the setup options window, that is select the "Enable Boot To Desktop/Scratch" option and hit the enter key. It will take you to another window called the "choose boot option" window that looks like the image below.



```
Chose boot option

Console Text console, requiring login (default)
Desktop Log in as user 'pi' at the graphical desktop
Scratch Start the Scratch programming environment upon boot

<Ok>                    <Cancel>
```

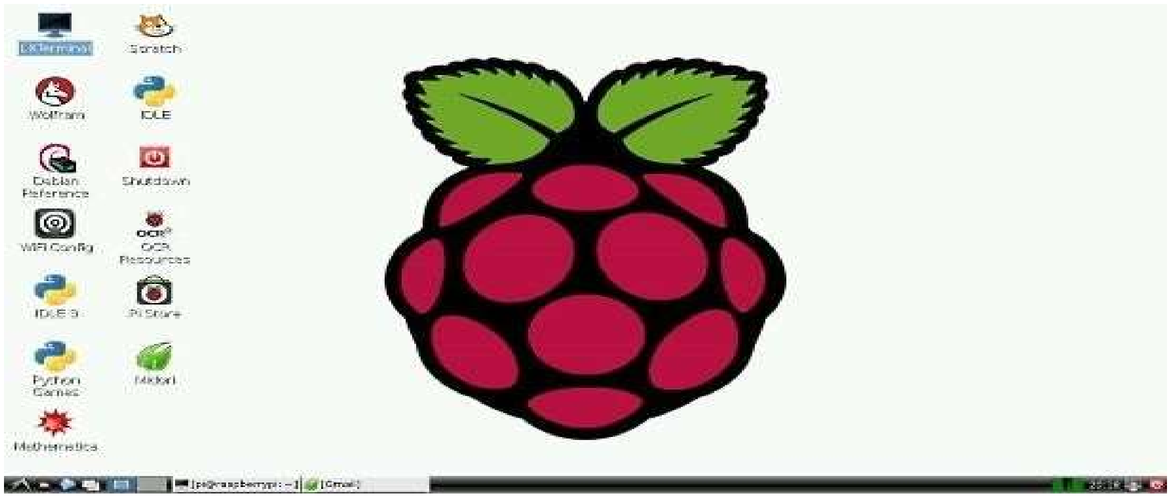
7 Boot Options

In the "choose boot option window", select the second option, that is, "Desktop Log in as user 'pi' at the graphical desktop" and hit the enter button. Once done you will be taken back to the "Setup Options" page, if not select the "OK" button at the bottom of this window and you will be taken back to the previous window. We do this because we want to boot into the desktop environment which we are familiar with. If we don't do this step then the Raspberry Pi boots into a terminal each time with no GUI options. Once, both the steps are done, select the "finish" button at the bottom of the page and it should reboot automatically. If it doesn't, then use the following command in the terminal to reboot**.**

Sudo Reboot

Updating the firmware After the reboot from the previous step, if everything went right, then you will end up on the desktop which looks like the image below.

Once you are on the desktop, open a terminal and enter the following command to update the firmware of the Pi.



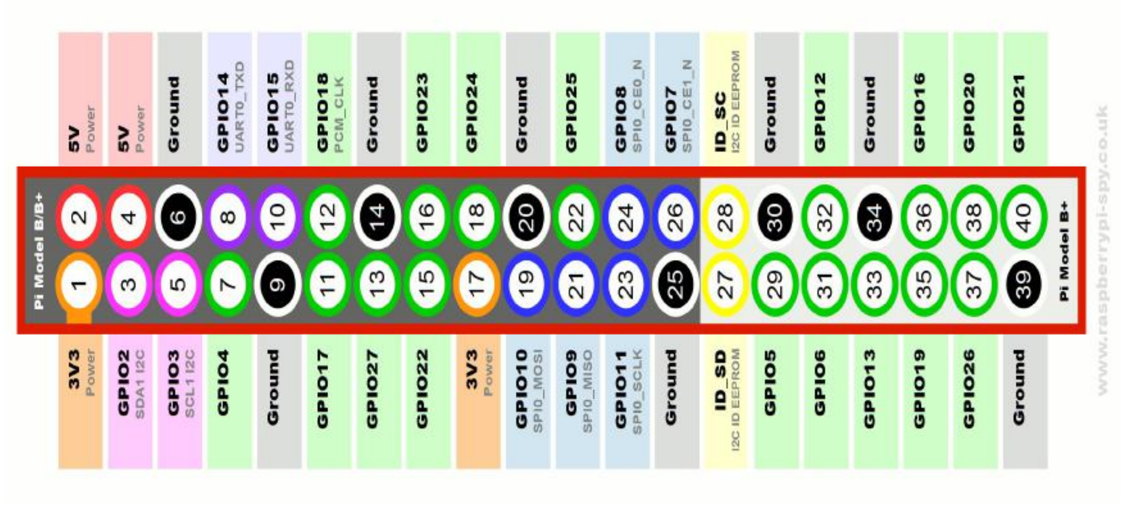
Raspberry Desktop

Sudo Rpi-Update:

Updating the firmware is necessary because certain models of the Pi might not have all the required dependencies to run smoothly or it may have some bug. The latest firmware might have the fix to those bugs, thus its very important to update it in the beginning itself.

Conclusion

So, we have covered the steps to get the Pi up and running. This method works on all the different models of Raspberry Pi (model A, B, B+ and also RPi 2) as Raspbain was made to be supported on all models. However, while installing other software or libraries, the procedure might change a bit while installing depending on the model of the Pi or the version of Raspbian itself. The concept of Raspberry is to keep trying till you get the result or build that you want. This might involve a lot of trial and error but spending the time will be worth it. The actual usage doesn't end here. This is just the beginning. It is up to you to go ahead to build something amazing out of it.



GPIO:

Act as both digital output and digital input.

Output: turn a GPIO pin high or low.

Input: detect a GPIO pin high or low

Installing GPIO library:

Open terminal

Enter the command “sudoapt-get install python-dev” to install python development Enter the command “sudoapt-get install python-rpi.gpio” to install GPIO library. Basic python coding:

Open terminal enters the command

sudo nano filename.py

This will open the nano editor where you can write your code Ctrl+O : Writes the code to the file

Ctrl+X : Exits the editor

Blinking LED:

Code:

import RPi.GPIO as GPIO #GPIO library import time

GPIO.setmode(GPIO.BOARD) # Set the type of board for pin numbering GPIO.setup(11, GPIO.OUT) # Set GPIO pin 11as output pin

for i in range (0,5): GPIO.output(11,True) # Turn on GPIO pin 11

time.sleep(1)

GPIO.output(11,False)

time.sleep(2)

GPIO.output(11,True)

GPIO.cleanup()

Power Pins

The header provides 5V on Pin 2 and 3.3V on Pin 1. The 3.3V supply is limited to 50mA. The 5V supply draws current directly from your microUSB supply so can use whatever is left over after the board has taken its share. A 1A power supply could supply up to 300mA once the Board has drawn 700mA

Basic GPIO

The header provides 17 Pins that can be configured as inputs and outputs. By default they are all configured as inputs except GPIO 14 & 15.

In order to use these pins you must tell the system whether they are inputs or outputs. This can be achieved a number of ways and it depends on how you intend to control them. I intend on using Python.

SDA & SCL: The 'DA' in SDA stands for data, the 'CL' in SCL stands for clock; the S stands for serial. You can do more reading about the significance of the clock line for various types of computer bus, You will probably find I2C devices that come with their own userspace drivers and the linux kernel includes some as well. Most computers have an I2C bus, presumably for some of the purposes listed by wikipedia, such as interfacing with the RTC (real time clock) and configuring memory. However, it is not exposed, meaning you can't attach anything else to it, and there are a lot of interesting things that could be attached -- pretty much any kind of common sensor (barometers, accelerometers, gyroscopes, luminometers, etc.) as well as output devices and displays. You can buy a USB to I2C adapter for a normal computer, but they cost a few hundred dollars. You can attach multiple devices to the exposed bus on the pi.

UART, TXD & RXD: This is a traditional serial line; for decades most computers have had a port for this and a port for parallel.1 Some pi oriented OS distros such as Raspbian by default boot with this serial line active as a console, and you can plug the other end into another computer and use some appropriate software to communicate with it. Note this interface does not have a clock line; the two pins may be used for full duplex communication (simultaneous transmit and receive).

PCM, CLK/DIN/DOUT/FS: PCM is is how uncompressed digital audio is encoded. The data stream is serial, but interpreting this correctly is best done with a separate clock line (more lowest level stuff).

SPI, MOSI/MISO/CE0/CE1: SPI is a serial bus protocol serving many of the same purposes as I2C, but because there are more wires, it can operate in full duplex which makes it faster and more flexible.

Raspberry Pi Terminal Commands

[sudo apt-get update] - Update Package Lists

[sudo apt-get upgrade] - Download and Install Updated Packages

[sudo raspi-config] - The Raspberry Pi Configuration Tool

[sudo apt-get clean] - Clean Old Package Files

[sudo reboot] - Restart your Raspberry Pi

[sudo halt] - Shut Down your Raspberry Pi

WRITING A PYTHON PROGRAM

To demonstrate creating and executing a Python program, we'll make a simple "hello world" program. To begin, open the Nano text editor and create a new file named hello-world.py by entering this at the command prompt: sudo nano hello-world.py

Enter this code into Nano, then press Ctrl-X and Y to exit and save the file:

```
#!/usr/bin/python


print "Hello, World!";
```

All Python program files will need to be saved with a ".py" extension. You can write the program in any text editor such as Notepad or Notepad++, just be sure to save the file with a ".py" extension. To run the program without making it executable, navigate to the location where you saved your file, and enter this at the command prompt: python hello-world.py

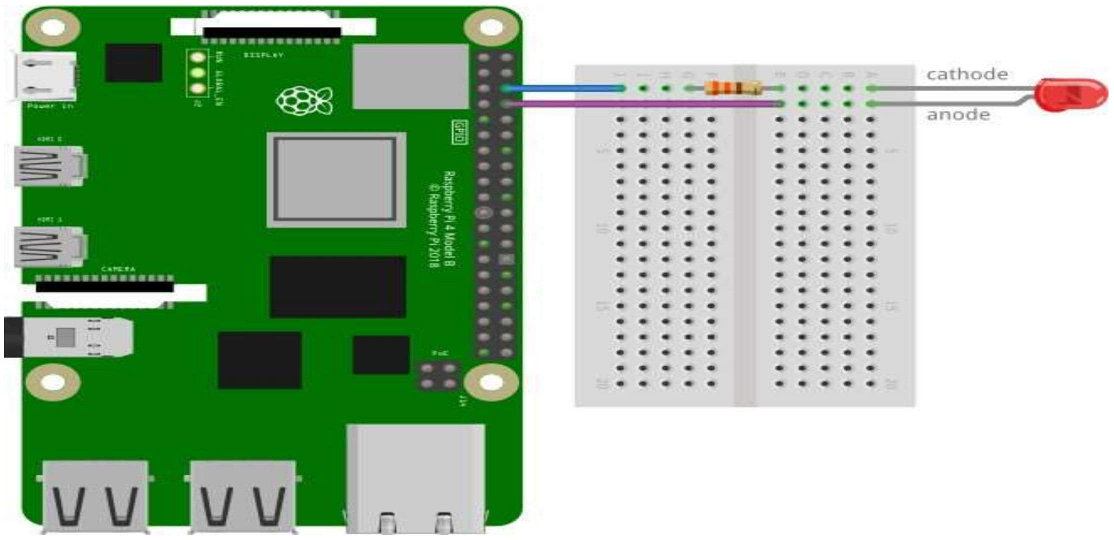
CONNECT THE LED TO THE RASPBERRY PI

Components:

- Raspberry Pi
- One LED
- One 330 Ohm resistor
- Jumper wires
- Breadboard

Connect the components as shown in the wiring diagram below.

The 330 Ohm resistor is a current limiting resistor. Current limiting resistors should always be used when connecting LEDs to the GPIO pins. If an LED is connected to a GPIO pin without a resistor, the LED will draw too much current, which can damage the Raspberry Pi or burn out the LED. Here is a nice calculator that will give you the value of a current



Raspberry pi with LED

limiting resistor to use for different LEDs. After connecting the hardware components, the next step is to create a Python program to switch on and off the LED. This program will make the LED turn on and off once every second and output the status of the LED to the terminal. The first step is to create a Python file. To do this, open the Raspberry Pi terminal and type nano LED.py. Then press Enter. This will create a file named LED.py and open it in the Nano text editor. Copy and paste the Python code below into Nano and save and close the file.

import RPi.GPIO as GPIO import time GPIO.setmode(GPIO.BCM) GPIO.setwarnings(False) GPIO.setup(14,GPIO.OUT)

# While loop while True:

# set GPIO14 pin to HIGH GPIO.output(14,GPIO.HIGH)

# show message to Terminal print "LED is ON"

# pause for one second time.sleep(1)

# set GPIO14 pin to HIGH

GPIO.output(14,GPIO.LOW)

# show message to Terminal print "LED is OFF"

# pause for one second time.sleep(1)

At the top of the program we import the RPi.GPIO and time libraries. The RPi.GPIO library will allow us to control the GPIO pins. The time library contains the sleep() function that we will use to make the LED pause for one second.

Next we initialize the GPIO object with GPIO.setmode(GPIO.BCM). We are using the BCM pin numbering system in this program. We use .GPIO.setwarnings(False) to disable the warnings and GPIO.setup(14,GPIO.OUT) is used to set GPIO14 as an output.

Now we need to change the on/off state of GPIO14 once every second. We do this with the GPIO.output() function. The first parameter of this function is the GPIO pin that will be switched high or low. We have the LED connected to GPIO14 in this circuit, so the first argument is 14.

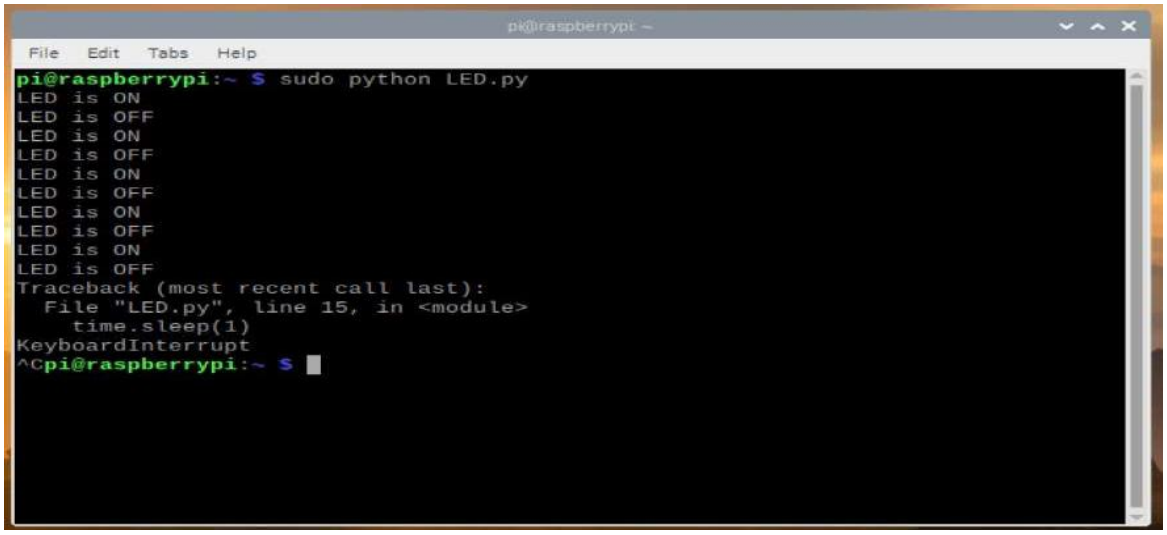
The second parameter of the GPIO.output() function is the voltage state of the GPIO pin. We can use either GPIO.HIGH or GPIO.LOW as an argument to turn the pin on or off.

Each GPIO.output() function in the code above is followed by a sleep() function that causes the pin to hold its voltage state for the time (in seconds) defined in the parameter of the function. In this program we are switching the LED on and off once every second so the argument is 1. You can change this value to make the LED blink on and off faster or slower. Run the Python program above by entering the following into the Raspberry Pi's terminal:

sudo python LED.py

You should see the LED blinking on and off once every second.

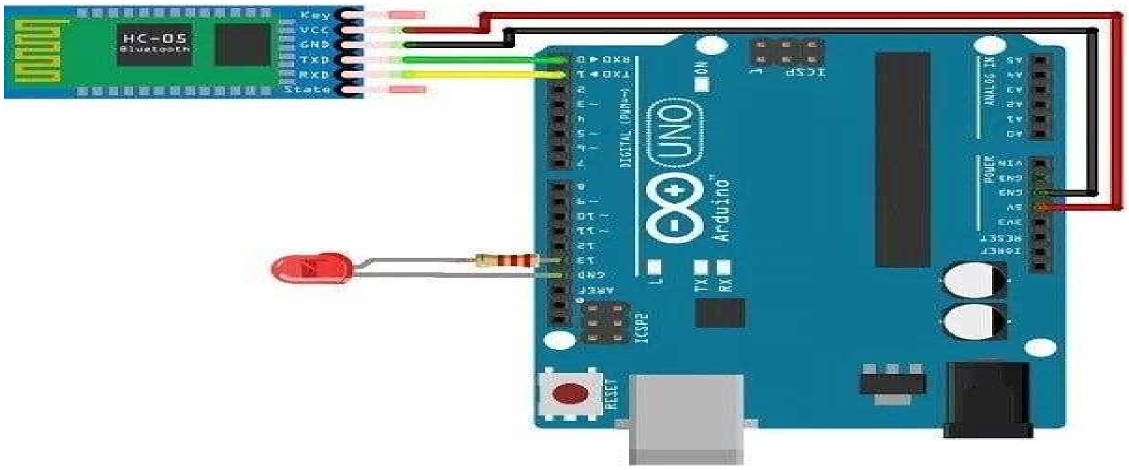
You should also see a message in the terminal with “LED is ON“ when the LED is turned on, and “LED is OFF” when the LED is turned off.



```
pi@raspberrypi:~ $ sudo python LED.py
LED is ON
LED is OFF
LED is ON
LED is OFF
LED is ON
LED is OFF
LED is ON
LED is OFF
LED is ON
LED is OFF
Traceback (most recent call last):
  File "LED.py", line 15, in <module>
    time.sleep(1)
KeyboardInterrupt
^Cpi@raspberrypi:~ $ 
```

Communication through BLUETOOTH MODULES

Bluetooth Low Energy Modules available at a reasonable cost, most of these modules are not compatible with existing devices that support the classic Bluetooth. The HC-05 is an expensive module that is compatible with wide range of devices including smartphone, laptops and tablets. Adding a Bluetooth to Arduino can take your project to the next level. It opens up lots of possibilities for user interface (UI) and communication.



There are three main parts to this module. An Android smartphone, a Bluetooth transceiver, and an Arduino. HC 05/06 works on serial communication. The Android app is designed to send serial data to the Arduino Bluetooth module when a button is pressed on the app. The Arduino Bluetooth module at the other end receives the data and sends it to the Arduino through the TX pin of the Bluetooth module (connected to RX pin of Arduino). The code uploaded to the Arduino checks the received data and compares it. If the received data is 1, the LED turns ON. The LED turns OFF when the received data is 0. You can open the serial monitor and watch the received data while connecting.

```
char data = 0;            //Variable for storing received data void setup()

{

Serial.begin(9600);     //Sets the data rate in bits per second (baud) for serial data transmission
pinMode(13, OUTPUT);        //Sets digital pin 13 as output pin

}

void loop()

{

if(Serial.available() > 0) // Send data only when you receive data:

{

data = Serial.read();             //Read the incoming data and store it into variable data
Serial.print(data);       //Print Value inside data in Serial monitor Serial.print("\n");//New line

if(data == '1')  //Checks whether value of data is equal to 1 digitalWrite(13, HIGH);

else if(data == '0') digitalWrite(13, LOW);

}
```

}

WiFi MODULES

ESP8266WiFi library

ESP8266 is all about Wi-Fi. If you are eager to connect your new ESP8266 module to a Wi-Fi network to start sending and receiving data, this is a good place to start. If you are looking for more in depth details of how to program specific Wi-Fi networking functionality, you are also in the right place. The Wi-Fi library for ESP8266 has been developed based on ESP8266 SDK, using the naming conventions and overall functionality philosophy of the Arduino WiFi library.

In order to get our ESP8266 to work properly with our Arduino, we need to do some initial programming. Specifically, we will be changing the ESP8266 to work as an access point and a client and changing the baud rate. Since most code samples out there are communicating with the ESP module with a baud rate of 9600, that's what we will use. We will also verify that the ESP8266 module can connect to our router.

With your Arduino Uno connected to your computer, open the serial monitor via the Arduino IDE (ctrl + shift + m). On the bottom of the serial monitor there are dropdowns for line endings and baud rate. Set line endings to "Both NL & CR" and change the baud rate to "115200". Then send the following commands:

1. **Verify that the ESP8266 is connected properly.**

   Command to send: AT Expected response: OK

2. **Change the mode.**

   Command to send: AT+CWMODE=3 Expected response: OK

3. **Connect to your router (Make sure to replace YOUR_SSID and YOUR_WIFI_PASSWORD).**
   Command to send: AT+CWJAP="YOUR_SSID","YOUR_WIFI_PASSWORD"
   Expected response:
   WIFI CONNECTED WIFI GOT IP
   OK
4. **Set baud rate to 9600.**
   Command to send: AT+UART=9600,8,1,0,0
   Expected response: OK
5. **Verify that the ESP8266 is communicating with baud rate of 9600**

   Command to send: AT Expected response: OK

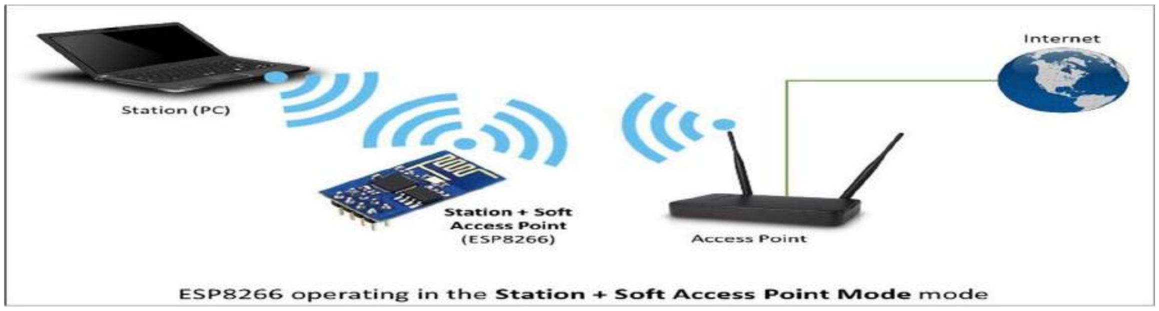
```
#include "WiFiEsp.h"

#include <ArduinoJson.h>

#ifndef HAVE_HWSERIAL1

 #include "SoftwareSerial.h"

// set up software serial to allow serial communication to our TX and RX pins
SoftwareSerial Serial1(10, 11);

#endif

// Set baud rate of so we can monitor output from esp.

 #define ESP8266_BAUD 9600
```

```
        // CHANGE THIS TO MATCH YOUR SETTINGS
                char ssid[] = "MY_SSID";
                char pass[] = "MY_WIFI_PASSWORD"; int status = WL_IDLE_STATUS;


        // Define an esp server that will listen on port 80
        WiFiEspServer server(80);
        void setup()
                {
                // Open up communications for arduino serial and esp serial at same rate
                Serial.begin(9600);
                Serial1.begin(9600);
                // Initialize the esp module WiFi.init(&Serial1);
                // Start connecting to wifi network and wait for connection to complete
                 while (status != WL_CONNECTED)
                        {
                                Serial.print("Conecting to wifi network: ");
                                Serial.println(ssid);
                                status = WiFi.begin(ssid, pass);
                }
        // Once we are connected log the IP address of the ESP module
        Serial.print("IP Address of ESP8266 Module is: ");
        Serial.println(WiFi.localIP());
        Serial.println("You're connected to the network");
        // Start the server
         server.begin();
}
// Continually check for new clients
 void loop()
        {
        WiFiEspClient client = server.available();
        // If a client has connected...
        if (client)
                {
                String json = "";
                Serial.println("A client has connected");

                while (client.connected())

                        {

                        // Read in json from connected client

                         if (client.available())

                                {

                                // ignore headers and read to first json bracket
                                client.readStringUntil('{');

                                // get json body (everything inside of the main  brackets) String
                                jsonStrWithoutBrackets = client.readStringUntil('}');

                                // Append brackets to make the string parseable as json String
                                jsonStr = "{" + jsonStrWithoutBrackets + "}";


                        // if we managed to properly  form jsonStr...

                         if      (jsonStr.indexOf('{', 0) >= 0)

                        {

                                //       parse   string  into    json,   bufferSize     calculated
                                by https://arduinojson.org/v5/assistant/

                                const size_t bufferSize = JSON_OBJECT_SIZE(1) + 20;
                                DynamicJsonBuffer jsonBuffer(bufferSize);

                                JsonObject &root = jsonBuffer.parseObject(jsonStr);

                                // get and print the value of the action key in our json object
                                const char *value = root["action"];

                                Serial.println(value);

                                if (strcmp(value, "on") == 0)

                                {

                                // Do something when we receive the on command
                                Serial.println("Received on command from client");

                        }

                else if (strcmp(value, "off") == 0)

                        {

                        // Do something when we receive the off command
                        Serial.println("Received off command from client");

        }

}
```

```
 // send response and close connection client.print( "HTTP/1.1 200 OK\r\n" "Connection:
close\r\n"

// the connection will be closed after completion of the Response        \r\n");

client.stop();

else

          {

          // we were unable to parse json, send http error status and close connection
          client.print("HTTP/1.1 500 ERROR\r\n" "Connection: close\r\n" "\r\n");

          Serial.println("Error, bad or missing json"); client.stop();

          }

     }

}

delay(100);

client.stop();

Serial.println("Client disconnected");

}

}.
```

Devices that connect to Wi-Fi networks are called stations (STA). Connection to Wi-Fi is provided by an access point (AP), that acts as a hub for one or more stations. The access point on the other end is connected to a wired network. An access point is usually integrated with a router to provide access from a Wi-Fi network to the internet. Each access point is recognized by a SSID (Service Set IDentifier), that essentially is the name of network you select when connecting a device (station) to the Wi-Fi.

ESP8266 modules can operate as a station, so we can connect it to the Wi-Fi network. It can also operate as a soft access point (soft-AP), to establish its own Wi-Fi network. When the ESP8266 module is operating as a soft access point, we can connect other stations to the ESP module. ESP8266 is also able to operate as both a station and a soft access point mode. This provides the possibility of building e.g. mesh networks.


Internet
Station (PC)
Station + Soft Access Point (ESP8266)
Access Point
ESP8266 operating in the Station + Soft Access Point Mode mode

ESP8266 Module

C VENKATA SUBBAIAH
HOD, DPET. OF CSE,
AITS-KADAPA

UNIT III

IoT Architecture and Protocols

Architecture Reference Model:

Introduction:

The Internet of Things (IoT) has seen an increasing interest in adaptive frameworks and architectural designs to promote the correlation between IoT devices and IoT systems. This is because IoT systems are designed to be categorized across diverse application domains and geographical locations. It, therefore, creates extensive dependencies across domains, platforms and services. Considering this interdependency between IoT devices and IoT systems, an intelligent, connection-aware framework has become a necessity, this is where IoT architecture comes into play.

In essence, an IoT architecture is the system of numerous elements that range from sensors, protocols, actuators, to cloud services, and layers. Besides, devices and sensors the Internet of Things (IoT) architecture layers are distinguished to track the consistency of a system through protocols and gateways. Different architectures have been proposed by researchers and we can all agree that there is no single consensus on architecture for IoT.

State of the art

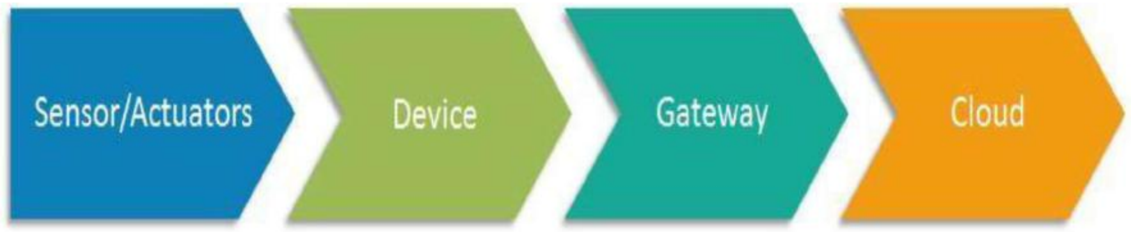
IoT architecture varies from solution to solution, based on the type of solution which we intend to build. IoT as a technology majorly consists of four main components, over which an architecture is framed.

1) Sensors

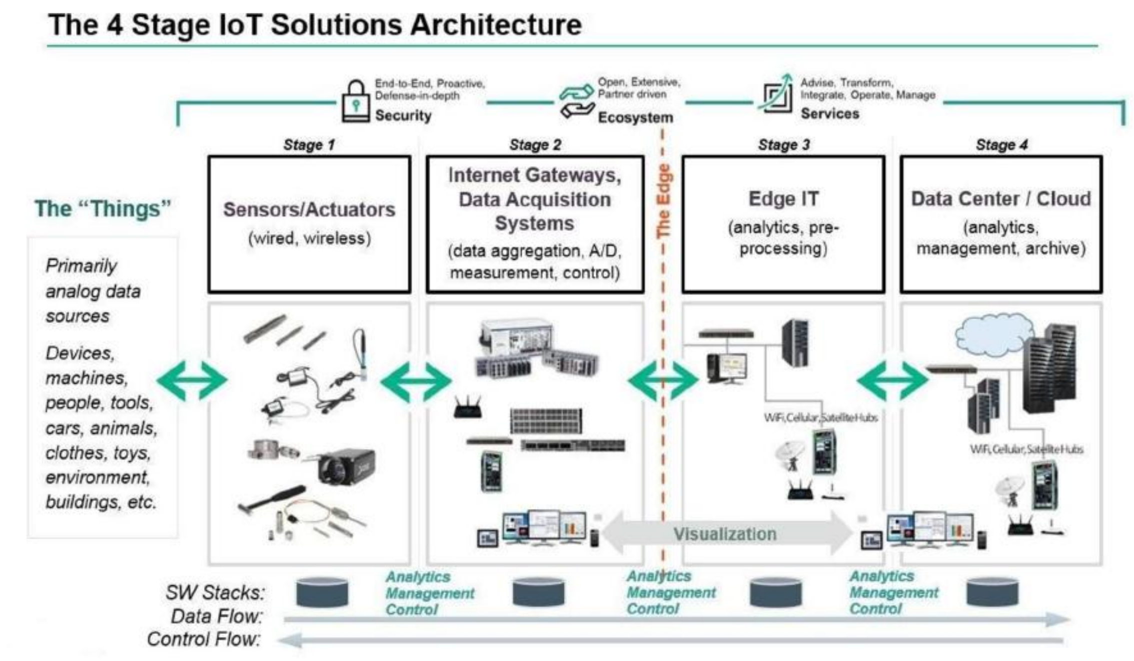
2) Devices

3) Gateway

4) Cloud



Stages of IoT Architecture:


The 4 Stage IoT Solutions Architecture
End-to-End, Proactive, Defense-in-depth
Security
Open, Extensive, Partner driven
Ecosystem
Advise, Transform, Integrate, Operate, Manage
Services
Stage 1
Stage 2
Stage 3
Stage 4
The "Things"
Primarily analog data sources
Devices, machines, people, tools, cars, animals, clothes, toys, environment, buildings, etc.
Sensors/Actuators
(wired, wireless)
Internet Gateways, Data Acquisition Systems
(data aggregation, A/D, measurement, control)
The Edge
Edge IT
(analytics, pre-processing)
Data Center / Cloud
(analytics, management, archive)
WiFi, Cellular, Satellite Hubs
Visualization
SW Stacks:
Data Flow:
Control Flow:
Analytics
Management
Control

1. Sensors/actuators

Sensors collect data from the environment or object under measurement and turn it into useful data. Think of the specialized structures in your cell phone that detect the directional pull of gravity and the phone's relative position to the ―thing‖ we call the earth and convert it into data that your phone can use to orient the device.

Actuators can also intervene to change the physical conditions that generate the data. An actuator might, for example, shut off a power supply, adjust an air flow valve, or move a robotic gripper in an assembly process.

The sensing/actuating stage covers everything from legacy industrial devices to robotic camera systems, water level detectors, air quality sensors, accelerometers, and heart rate monitors. And the scope of the IoT is expanding rapidly, thanks in part to low-power wireless sensor network technologies and Power over Ethernet, which enable devices on a wired LAN to operate without the need for an A/C power source.

2. The Internet gateways

The data from the sensors starts in analog form. That data needs to be aggregated and converted into digital streams for further processing downstream. Data acquisition systems (DAS) perform these data aggregation and conversion functions. The DAS connects to the sensor network, aggregates outputs, and performs the analog-to-digital conversion. The Internet gateway receives the aggregated and digitized data and routes it over Wi-Fi, wired LANs, or the Internet, to Stage 3 systems for further processing. Stage 2 systems often sit in close proximity to the sensors and actuators.

For example, a pump might contain a half-dozen sensors and actuators that feed data into a data aggregation device that also digitizes the data. This device might be physically attached to the pump. An adjacent gateway device or server would then process the data and forward it to the Stage 3 or Stage 4 systems. Intelligent gateways can build on additional, basic gateway functionality by adding such capabilities as analytics, malware protection, and data management services. These systems enable the analysis of data streams in real time.

3. Edge IT

Once IoT data has been digitized and aggregated, it's ready to cross into the realm of IT. However, the data may require further processing before it enters the data centre. This is where edge IT systems, which perform more analysis, come into play. Edge IT processing systems may be located in remote offices or other edge locations, but generally these sit in the facility or location where the sensors reside closer to the sensors, such as in a wiring closet. Because IoT data can easily eat up network bandwidth and swamp your data centre resources, it's best to have systems at the edge capable of performing analytics as a way to lessen the burden on core IT infrastructure. You'd also face security concerns, storage issues, and delays processing the data. With a staged approach, you can pre-process the data, generate meaningful results, and pass only those on. For example, rather than passing on raw vibration data for the pumps, you could aggregate and convert the data, analyse it, and send only projections as to when each device will fail or need service.

4.The data centre and cloud

Data that needs more in-depth processing, and where feedback doesn't have to be immediate, gets forwarded to physical data centre or cloud-based systems, where more powerful IT systems can analyse, manage, and securely store the data. It takes longer to get results when you wait until data reaches Stage 4, but you can execute a more in-depth analysis, as well as combine your sensor data with data from other sources for deeper insights. Stage 4 processing may take place on-premises, in the cloud, or in a hybrid cloud system, but the type of processing executed in this stage remains the same, regardless of the platform.

Reference Model and architecture:

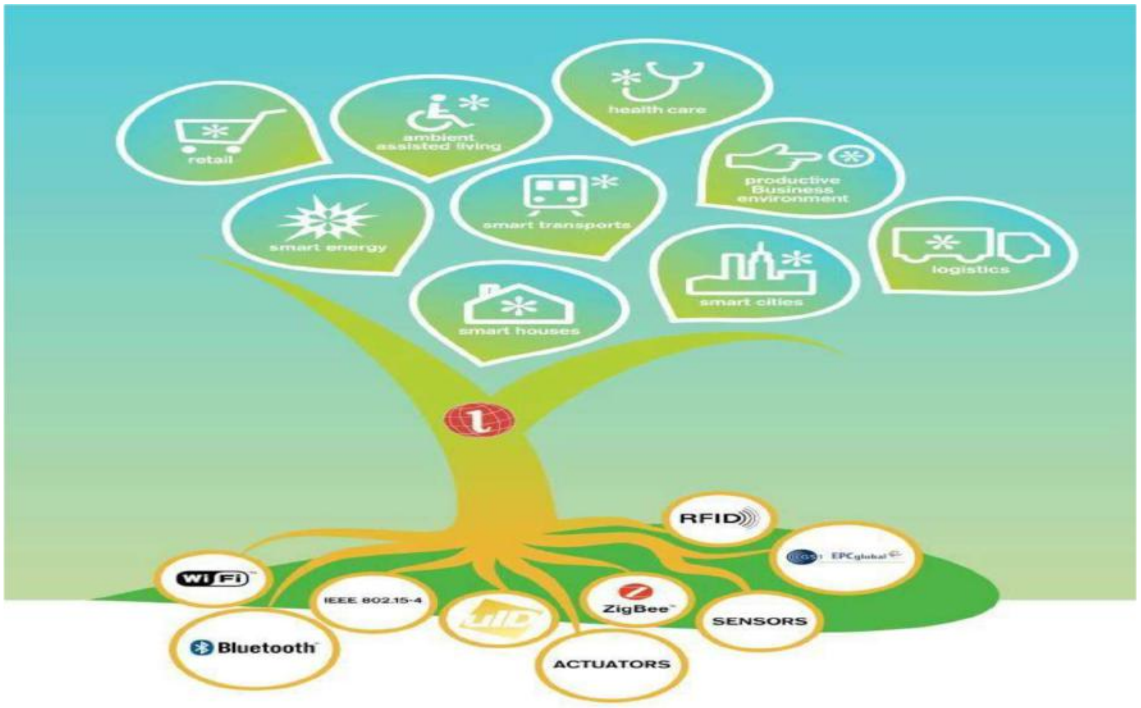
Reference Architecture that describes essential building blocks as well as design choices to deal with conflicting requirements regarding functionality, performance, deployment and security. Interfaces should be standardised, best practices in terms of functionality and information usage need to be provided.

The central choice of the IoT-A project was to base its work on the current state of the art, rather than using a clean-slate approach. Due to this choice, common traits are derived to form the base line of the Architectural Reference Model (ARM). This has the major advantage of ensuring backward compatibility of the model and also the adoption of established, working solutions to various aspects of the IoT. With the help of end users, organised into a stakeholder's group, new requirements for IoT have been collected and introduced in the main model building process. This work was conducted according to established architecture methodology.

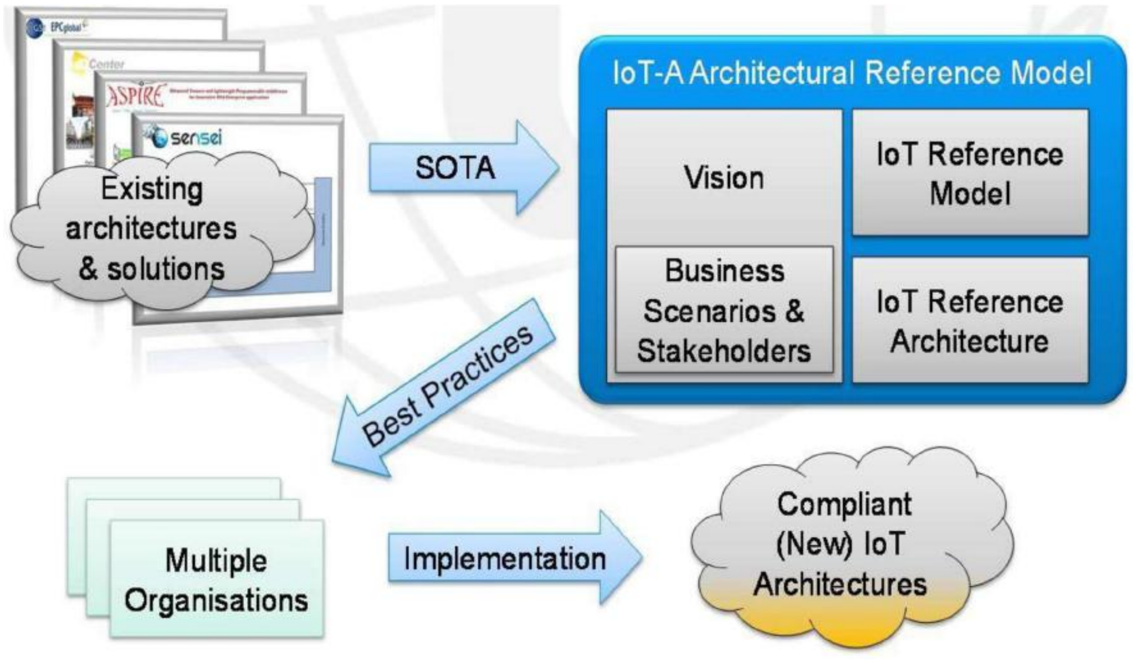
A Reference Architecture (RA) can be visualised as the ―Matrix that eventually gives birth ideally to all concrete architectures. For establishing such a Matrix, based on a strong and exhaustive analysis of the State of the Art, we need to envisage the superset of all possible functionalities, mechanisms and protocols that can be used for building such concrete architecture and to show how interconnections could take place between selected ones (as no concrete system is likely to use all of the functional possibilities). Giving such a foundation along with a set of design-choices, based on the characterisation of the targeted system w.r.t. various dimensions (like distribution, security, real-time, semantics) it becomes possible for a system architect to select the protocols, functional components, architectural options, needed to build their IoT systems.

As any metaphoric representation, this tree does not claim to be fully consistent in its depiction; it should therefore not be interpreted too strictly. On the one hand, the roots of this tree are spanning across a selected set of communication protocols (6LoWPAN, Zigbee, IPv6,) and device technologies (sensors, actuators, tags,) while on the other hand the blossoms / leaves of the tree represent the whole set of IoT applications that can be built from the sap (i.e., data and information) coming from the roots. The trunk of the tree is of utmost importance here, as it represents the Architectural Reference Model (ARM). The ARM is the combination of the Reference Model and the Reference Architecture, the set of models, guidelines, best practices, views and perspectives that can be used for building fully

interoperable concrete IoT architectures and systems. In this tree, we aim at selecting a minimal set of interoperable technologies (the roots) and proposing the potentially necessary set of enablers or building blocks (the trunk) that enable the creation of a maximal set of interoperable IoT systems (the leaves).



IoT-A architectural reference model building blocks


EPCglobal
ASPIRE
sensei
Existing architectures & solutions
SOTA
IoT-A Architectural Reference Model
Vision
IoT Reference Model
Business Scenarios & Stakeholders
IoT Reference Architecture
Best Practices
Multiple Organisations
Implementation
Compliant (New) IoT Architectures

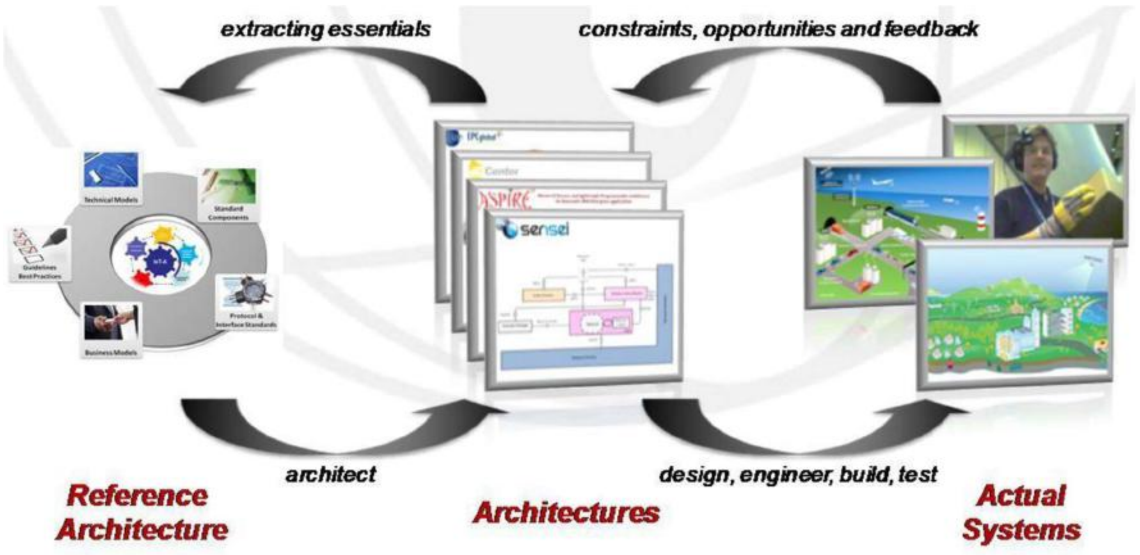
Starting with existing architectures and solutions, generic baseline requirements can be extracted and used as an input to the design. The IoT-A ARM consists of four parts:

The vision summarises the rationale for providing an architectural reference model for the IoT. At the same time, it discusses underlying assumptions, such as motivations. It also discusses how the architectural reference model can be used, the methodology applied to the architecture modelling, and the business scenarios and stakeholders addressed.

Business scenarios defined as requirements by stakeholders are the drivers of the architecture work. With the knowledge of businesses aspirations, a holistic view of IoT architectures can be derived.

The IoT Reference Model provides the highest abstraction level for the definition of the IoT-A Architectural Reference Model. It promotes a common understanding of the IoT domain. The description of the IoT Reference Model includes a general discourse on the IoT domain, an IoT Domain Model as a top-level description, an IoT Information Model explaining how IoT information is going to be modelled, and an IoT Communication Model in order to understand specifics about communication between many heterogeneous IoT devices and the Internet as a whole.

The IoT Reference Architecture is the reference for building compliant IoT architectures. As such, it provides views and perspectives on different architectural aspects that are of concern to stakeholders of the IoT. The terms' view and perspectives are used according to the general literature and standards the creation of the IoT Reference Architecture focuses on abstract sets of mechanisms rather than concrete application architectures. To organisations, an important aspect is the compliance of their technologies with standards and best practices, so that interoperability across organisations is ensured.

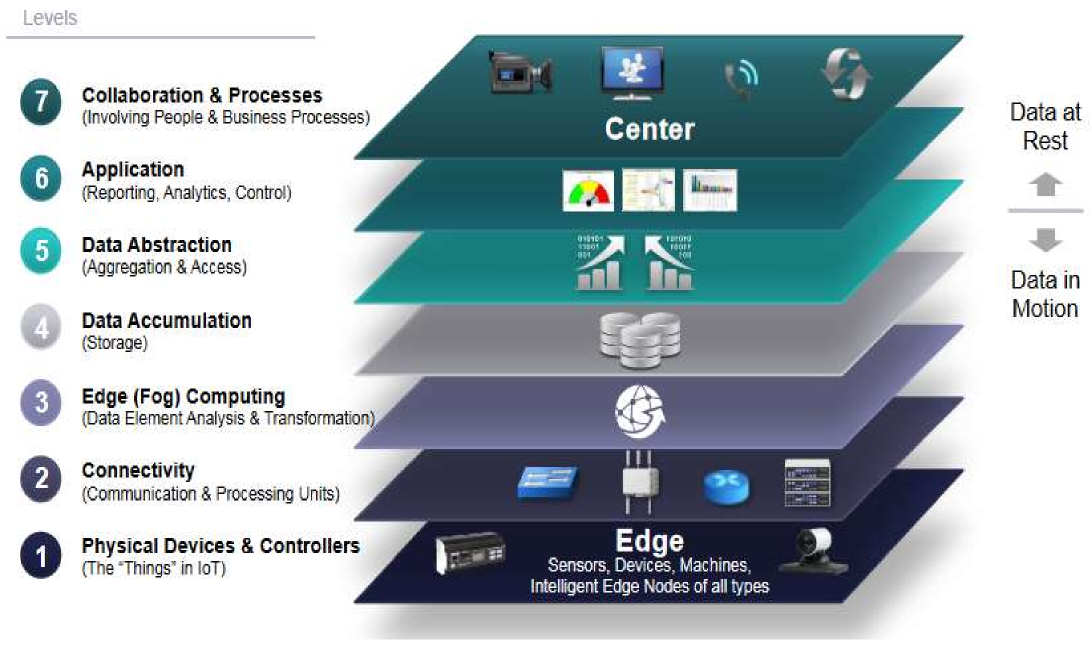


In an IoT system, data is generated by multiple kinds of devices, processed in different ways, transmitted to different locations, and acted upon by applications. The proposed IoT reference model is comprised of seven levels. Each level is defined with terminology that can be standardized to create a globally accepted frame of reference.

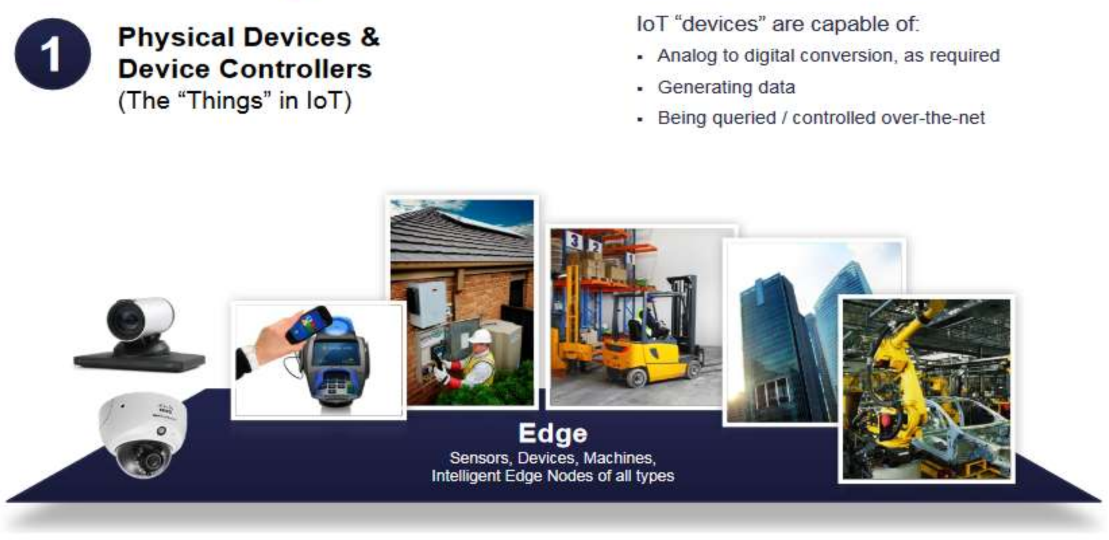
- ➢ Simplifies: It helps break down complex systems so that each part is more understandable. Clarifies: It provides additional information to precisely identify levels of the IoT and to establish common terminology.
- ➢ Identifies: It identifies where specific types of processing is optimized across different parts of the system.

- Standardizes: It provides a first step in enabling vendors to create IoT products that work with each other.
- Organizes: It makes the IoT real and approachable, instead of simply conceptual.

IoT reference Model:


Levels
7 Collaboration & Processes
(Involving People & Business Processes)
6 Application
(Reporting, Analytics, Control)
5 Data Abstraction
(Aggregation & Access)
4 Data Accumulation
(Storage)
3 Edge (Fog) Computing
(Data Element Analysis & Transformation)
2 Connectivity
(Communication & Processing Units)
1 Physical Devices & Controllers
(The "Things" in IoT)
Center
Edge
Sensors, Devices, Machines,
Intelligent Edge Nodes of all types
Data at Rest
Data in Motion

1. **Physical Devices and Controllers:**
   The IoT Reference Model starts with Level 1: physical devices and controllers that might control multiple devices. These are the "things" in the IoT, and they include a wide range of endpoint devices that send and receive information. Today, the list of devices is already extensive. It will become almost unlimited as more equipment is added to the IoT over time. Devices are diverse, and there are no rules about size, location, form factor, or origin. Some devices will be the size of a silicon chip. Some will be as large as vehicles. The IoT must support the entire range. Dozens or hundreds of equipment manufacturers will produce IoT devices. To simplify compatibility and support manufacturability, the IoT Reference Model generally describes the level of processing needed from Level 1 devices. Figure 2 describes basic capabilities for a device


1 Physical Devices & Device Controllers
(The "Things" in IoT)
IoT "devices" are capable of:
Analog to digital conversion, as required
Generating data
Being queried / controlled over-the-net
Edge
Sensors, Devices, Machines,
Intelligent Edge Nodes of all types

2. **Connectivity:**
   Communications and connectivity are concentrated in one level—Level 2. The most important function of Level 2 is reliable, timely information transmission. This includes transmissions:
   - ➢ Between devices (Level 1) and the network
   - ➢ Across networks (east-west)
   - ➢ Between the network (Level 2) and low-level information processing occurring at Level 3

   Traditional data communication networks have multiple functions, as evidenced by the International Organization for Standardization (ISO) 7-layer reference model. However, a complete IoT system contains many levels in addition to the communications network. One objective of the IoT Reference Model is for communications and processing to be executed by existing networks. The IoT Reference Model does not require or indicate creation of a different network—it relies on existing networks. However, some legacy devices aren't IP-enabled, which will require introducing communication gateways. Other devices will require proprietary controllers to serve the communication function. However, over time, standardization will increase. As Level 1 devices proliferate, the ways in which they interact with Level 2 connectivity equipment may change. Regardless of the details, Level 1 devices communicate through the IoT system by interacting with Level 2 connectivity equipment.
   - **Connectivity includes:**
     - o Communicating with and between the Level devices
     - o Reliable delivery across the network(s)
     - o Implementation of various protocols
     - o Switching and routing
     - o Translation between protocols
     - o Security at the network level(Self Learning) Networking Analytics

Level 2 functionality focuses on East-West communications

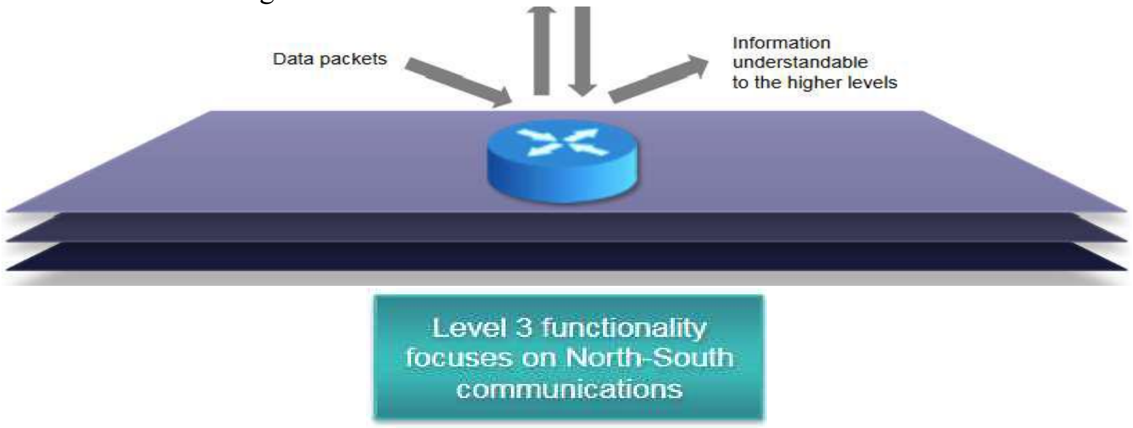
3. **Edge (Fog) Computing**
   The functions of Level 3 are driven by the need to convert network data flows into information that is suitable for storage and higher-level processing at Level 4 (data accumulation). This means that Level 3 activities focus on high-volume data analysis and transformation. For example, a Level 1 sensor device might generate data samples multiple times per second, 24 hours a day, 365 days a year. A basic tenet of the IoT Reference Model is that the most intelligent system initiates information processing as early and as close to the edge of the network as possible. This is sometimes referred to as fog computing. Level 3 is where this occurs.
   Given that data is usually submitted to the connectivity level (Level 2) networking equipment by devices in small units, Level 3 processing is performed on a packet-by-packet basis. This processing is limited, because there is only awareness of data units—

not “sessions” or “transactions.” Level 3 processing can encompass many examples, such as
Evaluation: Evaluating data for criteria as to whether it should be processed at a higher level

- Formatting: Reformatting data for consistent higher-level processing
- Expanding/decoding: Handling cryptic data with additional context (such as the origin)
- Distillation/reduction: Reducing and/or summarizing data to minimize the impact of data and traffic on the network and higher-level processing systems
- Assessment: Determining whether data represents a threshold or alert; this could include redirecting data to additional destinations.

Include;

- Data filtering, clean up, aggregation
- Packet content inspection
- Combination of network and data level analytics
- Thresholding
- Event generation


Data packets
Information understandable to the higher levels
Level 3 functionality focuses on North-South communications

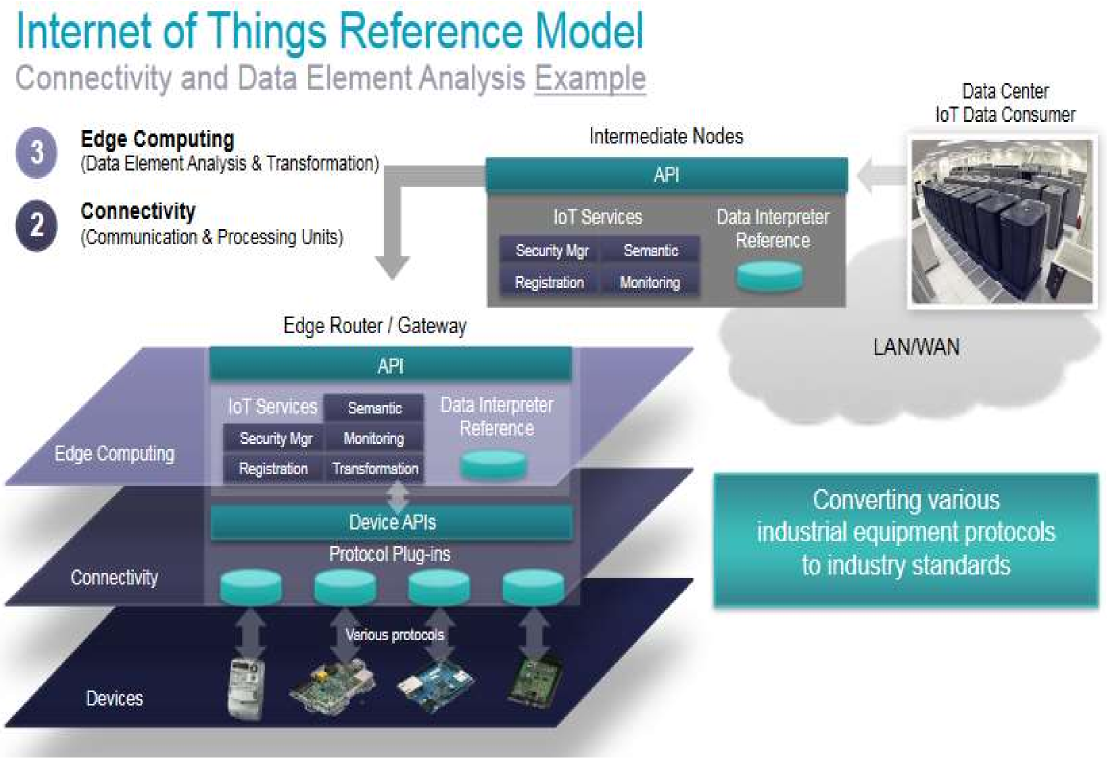


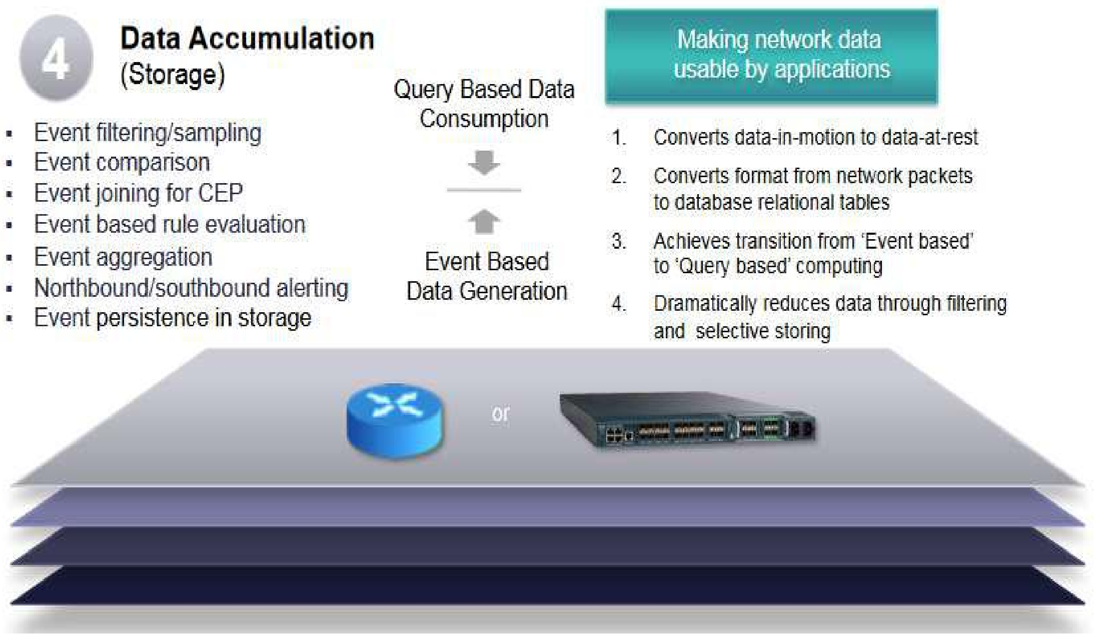
Fig : Level 2 and 3 Connectivity and Data Element Analysis Example

4. Data Accumulation

Networking systems are built to reliably move data. The data is "in motion." Prior to Level 4, data is moving through the network at the rate and organization determined by the devices generating the data. The model is event driven. As defined earlier, Level 1 devices do not include computing capabilities themselves. However, some computational activities could occur at Level 2, such as protocol translation or application of network security policy. Additional compute tasks can be performed at Level 3, such as packet inspection. Driving computational tasks as close to the edge of the IoT as possible, with heterogeneous systems distributed across multiple management domains represents an example of fog computing. Fog computing and fog services will be a distinguishing characteristic of the IoT. Most applications cannot, or do not need to, process data at network wire speed. Applications typically assume that data is "at rest"—or unchanging—in memory or on disk. At Level 4, Data Accumulation, data in motion is converted to data at rest. Level 4 determines:

- If data is of interest to higher levels: If so, Level 4 processing is the first level that is configured to serve the specific needs of a higher level.
- If data must be persisted: Should data be kept on disk in a non-volatile state or accumulated in memory for short-term use?
- The type of storage needed: Does persistency require a file system, big data system, or relational database?
- If data is organized properly: Is the data appropriately organized for the required storage system?
- If data must be recombined or recomputed: Data might be combined, recomputed, or aggregated with previously stored information, some of which may have come from non-IoT sources.

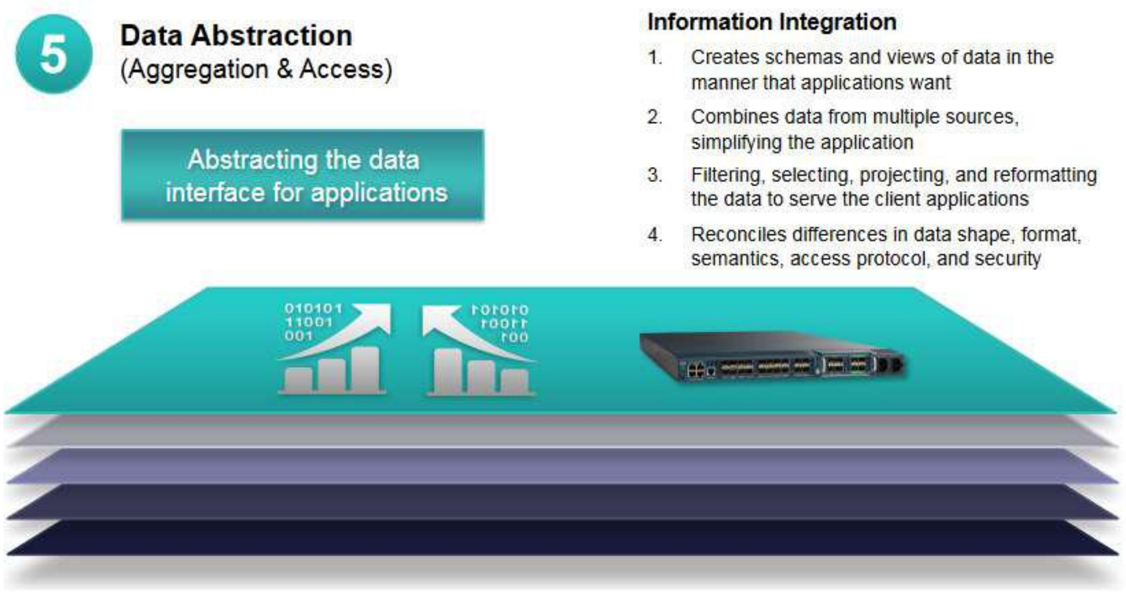
As Level 4 captures data and puts it at rest, it is now usable by applications on a non-real-time basis. Applications access the data when necessary. In short, Level 4 converts event-based data to query-based processing. This is a crucial step in bridging the differences between the real-time networking world and the non-real-time application world. Figure 6 summarizes the activities that occur at Level 4.


4
Data Accumulation
(Storage)
Event filtering/sampling
Event comparison
Event joining for CEP
Event based rule evaluation
Event aggregation
Northbound/southbound alerting
Event persistence in storage
Query Based Data Consumption
Event Based Data Generation
Making network data usable by applications
1. Converts data-in-motion to data-at-rest
2. Converts format from network packets to database relational tables
3. Achieves transition from 'Event based' to 'Query based' computing
4. Dramatically reduces data through filtering and selective storing
or

5. **Data Abstraction:**
IoT systems will need to scale to a corporate—or even global—level and will require multiple storage systems to accommodate IoT device data and data from traditional enterprise ERP, HRMS, CRM, and other systems. The data abstraction functions of Level 5 are focused on rendering data and its storage in ways that enable developing simpler, performance-enhanced applications. With multiple devices generating data, there are many reasons why this data may not land in the same data storage:
There might be too much data to put in one place.
    - Moving data into a database might consume too much processing power, so that retrieving it must be separated from the data generation process. This is done today with online transaction processing (OLTP) databases and data warehouses.
    - Devices might be geographically separated, and processing is optimized locally.
    - Levels 3 and 4 might separate "continuous streams of raw data" from "data that represents an event." Data storage for streaming data may be a big data system, such as Hadoop. Storage for event data may be a relational database management system (RDBMS) with faster query times.
    - Different kinds of data processing might be required. For example, in-store processing will focus on different things than across-all-stores summary processing

For these reasons, the data abstraction level must process many different things. These include:

- Reconciling multiple data formats from different sources
- Assuring consistent semantics of data across sources
- Confirming that data is complete to the higher-level application
- Consolidating data into one place (with ETL, ELT, or data replication) or providing access to multiple data
- stores through data virtualization
- Protecting data with appropriate authentication and authorization
- Normalizing or de-normalizing and indexing data to provide fast application access


5
Data Abstraction
(Aggregation & Access)
Abstracting the data interface for applications
Information Integration
1. Creates schemas and views of data in the manner that applications want
2. Combines data from multiple sources, simplifying the application
3. Filtering, selecting, projecting, and reformatting the data to serve the client applications
4. Reconciles differences in data shape, format, semantics, access protocol, and security
010101 11001 001

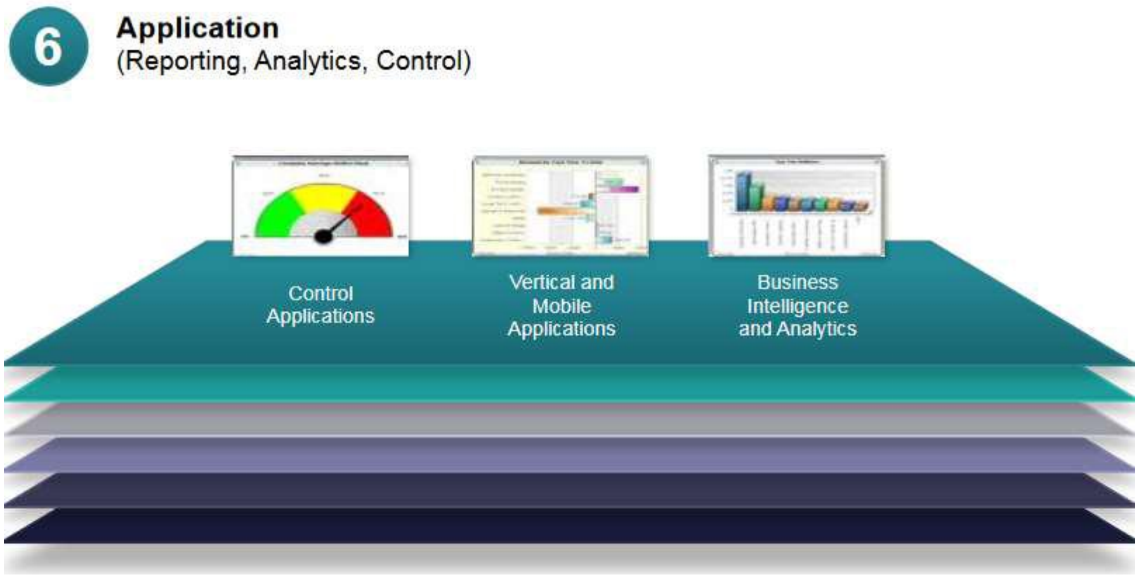
6. Application

Level 6 is the application level, where information interpretation occurs. Software at this level interacts with Level 5 and data at rest, so it does not have to operate at network speeds. The IoT Reference Model does not strictly define an application. Applications vary based on vertical markets, the nature of device data, and business needs. For example, some applications will focus on monitoring device data. Some will focus on controlling devices. Some will combine device and non-device data. Monitoring and control applications represent many different application models, programming patterns, and software stacks, leading to discussions of operating systems, mobility, application servers, hypervisors, multi-threading, multi-tenancy, etc. These topics are beyond the scope of the IoT Reference Model discussion. Suffice it to say that application complexity will vary widely.

Examples include:

- Mission-critical business applications, such as generalized ERP or specialized industry solutions
- Mobile applications that handle simple interactions
- Business intelligence reports, where the application is the BI server
- Analytic applications that interpret data for business decisions
- System management/control center applications that control the IoT system itself and don't act on the data produced by it

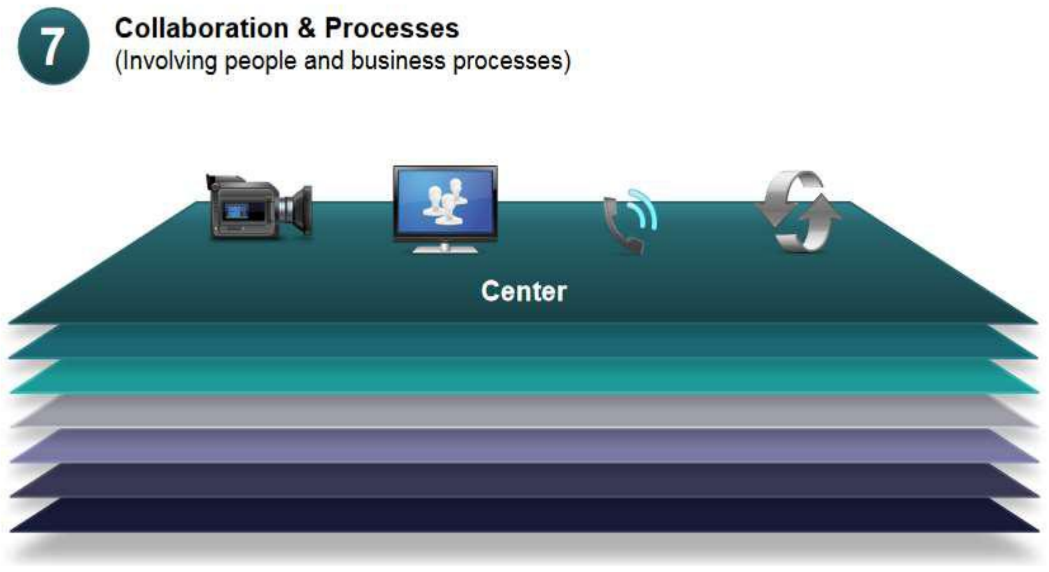
If Levels 1-5 are architected properly, the amount of work required by Level 6 will be reduced. If Level 6 is designed properly, users will be able to do their jobs better. Figure 8 depicts Level6


6
Application
(Reporting, Analytics, Control)
Control Applications
Vertical and Mobile Applications
Business Intelligence and Analytics

7. Collaboration and Processes

One of the main distinctions between the Internet of Things (IoT) and IoT is that IoT includes people and processes. This difference becomes particularly clear at Level 7: Collaboration and Processes. The IoT system, and the information it creates, is of little value unless it yields action, which often requires people and processes. Applications execute business logic to empower people. People use applications and associated data for their specific needs. Often, multiple people use the same application for a range of different purposes. So the objective is not the application—it is to empower people to do their work better. Applications (Level 6) give business people the right data, at the right time, so they can do the right thing.

But frequently, the action needed requires more than one person. People must be able to communicate and collaborate, sometimes using the traditional Internet, to make the IoT useful. Communication and collaboration often require multiple steps. And it usually transcends multiple applications. This is why Level 7, as shown in Figure 9, represents a higher level than a single application.

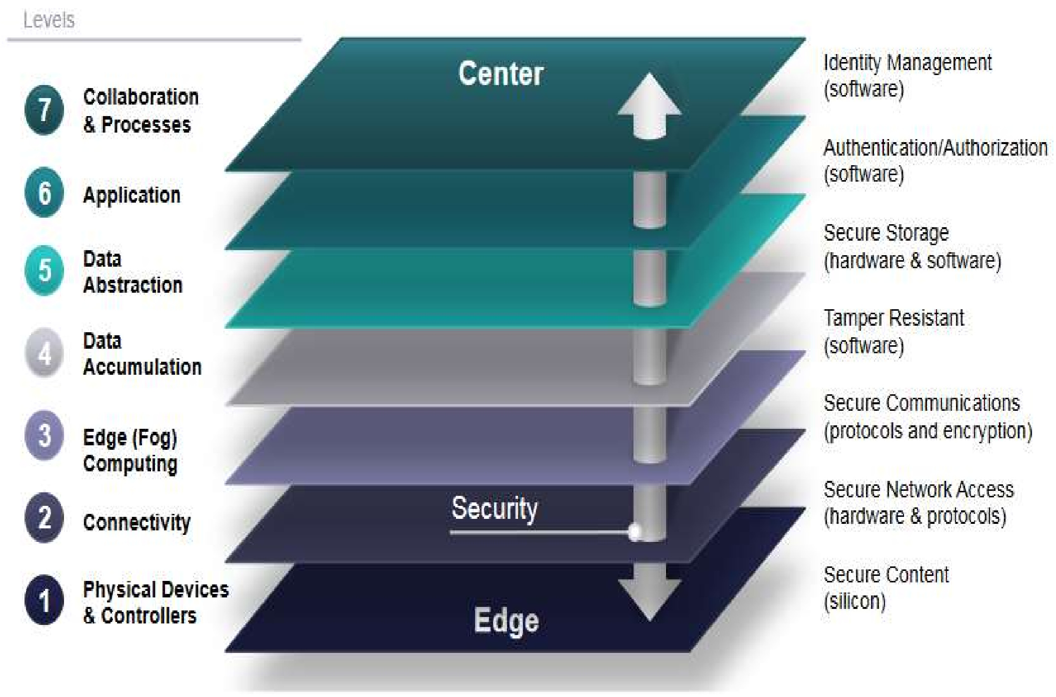

7
Collaboration & Processes
(Involving people and business processes)
Center

Security in the IoT:

Discussions of security for each level and for the movement of data between levels could fill a multitude of papers. For the purpose of the IoT Reference Model, security measures must:

- Secure each device or system
- Provide security for all processes at each level
- Secure movement and communication between each level, whether north- or south-bound

As shown in Figure 10, security must pervade the entire mode


Levels
7 Collaboration & Processes
6 Application
5 Data Abstraction
4 Data Accumulation
3 Edge (Fog) Computing
2 Connectivity
1 Physical Devices & Controllers
Center
Security
Edge
Identity Management (software)
Authentication/Authorization (software)
Secure Storage (hardware & software)
Tamper Resistant (software)
Secure Communications (protocols and encryption)
Secure Network Access (hardware & protocols)
Secure Content (silicon)

Protocols:

Internet protocol (IP) is a set of rules that dictates how data gets sent to the internet. IoT protocols ensure that information from one device or sensor gets read and understood by another device, a gateway, a service. Protocols they are :

1. 6LowPAN
2. RPL
3. CoAP
4. MQTT

6LoWPAN:(Internet Protocol version 6 (IPv6) over low-power wireless networks):

While the Internet Protocol is key for a successful Internet of Things, constrained nodes and constrained networks mandate optimization at various layers and on multiple protocols of the IP architecture. Some optimizations are already available from the market or under development by the IETF. Figure below highlights the TCP/IP layers where optimization is applied**.**

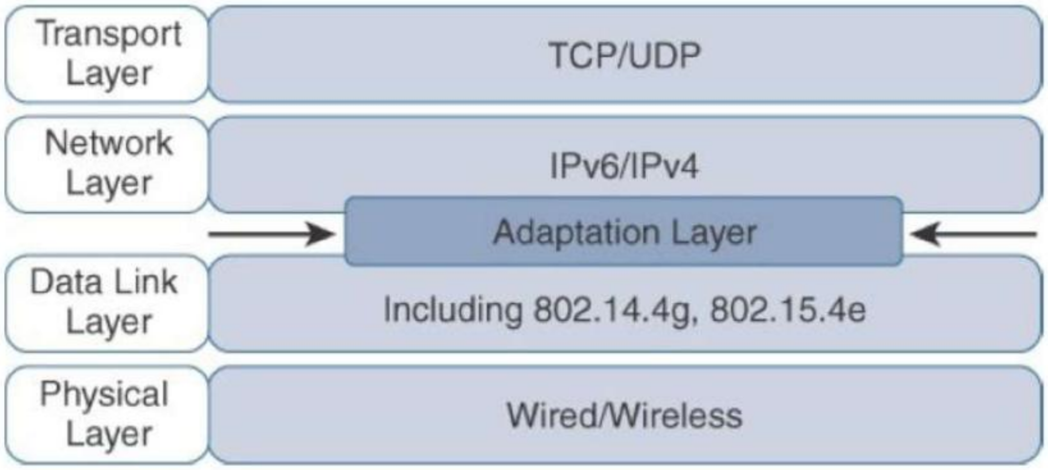


Figure : Optimizing IP for IoT Using an Adaptation Layer

In the IP architecture, the transport of IP packets over any given Layer 1 (PHY) and Layer 2 (MAC) protocol must be defined and documented. The model for packaging IP into lower-layer protocols is often referred to as an adaptation layer.

Unless the technology is proprietary, IP adaptation layers are typically defined by an IETF working group and released as a Request for Comments (RFC). An RFC is a publication from the IETF that officially documents Internet standards, specifications, protocols, procedures, and events. For example, RFC 864 describes how an IPv4 packet gets encapsulated over an Ethernet frame, and RFC 2464 describes how the same function is performed for an IPv6 packet.

IoT-related protocols follow a similar process. The main difference is that an adaptation layer designed for IoT may include some optimizations to deal with constrained nodes and networks. The main examples of adaptation layers optimized for constrained nodes or "things" are the ones under the 6LoWPAN working group and its successor, the 6Lo working group.

The initial focus of the 6LoWPAN working group was to optimize the transmission of IPv6 packets over constrained networks such as IEEE 802.15.4. Figure below shows an example of an IoT protocol stack using the 6LoWPAN adaptation layer beside the well-known IP protocol stack for reference.

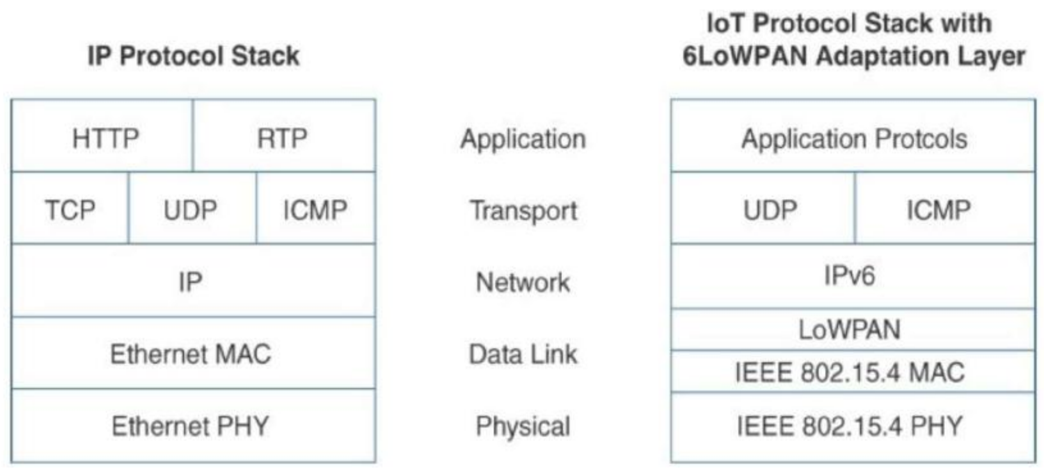


Fig: Comparison of an IoT Protocol Stack Utilizing 6LoWPAN and an IP Protocol Stack

The 6LoWPAN working group published several RFCs, but RFC 4994 is foundational because it defines frame headers for the capabilities of header compression, fragmentation, and mesh addressing. These headers can be stacked in the adaptation layer to keep these concepts separate while enforcing a structured method for expressing each capability. Depending on the implementation, all, none, or any combination of these capabilities and their corresponding headers can be enabled. Figure below shows some examples of typical 6LoWPAN header stacks.

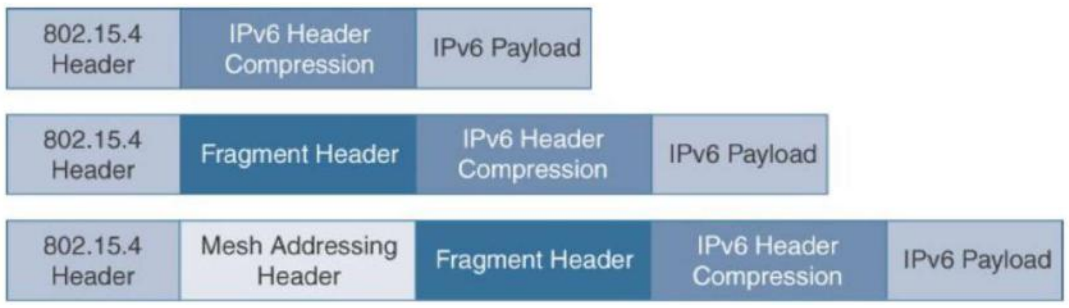


Figure :6LoWPAN Header Stack

Header Compression

IPv6 header compression for 6LoWPAN was defined initially in RFC 4944 and subsequently updated by RFC 6282. This capability shrinks the size of IPv6's 40-byte headers and User Datagram Protocol's (UDP's) 8-byte headers down as low as 6 bytes combined in some cases. Note that header compression for 6LoWPAN is only defined for an IPv6 header and not IPv4.

The 6LoWPAN protocol does not support IPv4, and, in fact, there is no standardized IPv4 adaptation layer for IEEE 802.15.4. 6LoWPAN header compression is stateless, and conceptually it is not too complicated. However, a number of factors affect the amount of compression, such as implementation of RFC 4944 versus RFC 6922, whether UDP is included, and various IPv6 addressing scenarios.

At a high level, 6LoWPAN works by taking advantage of shared information known by all nodes from their participation in the local network. In addition, it omits some standard header fields by assuming commonly used values. Figure below highlights an example that shows the amount of reduction that is possible with 6LoWPAN header compression.

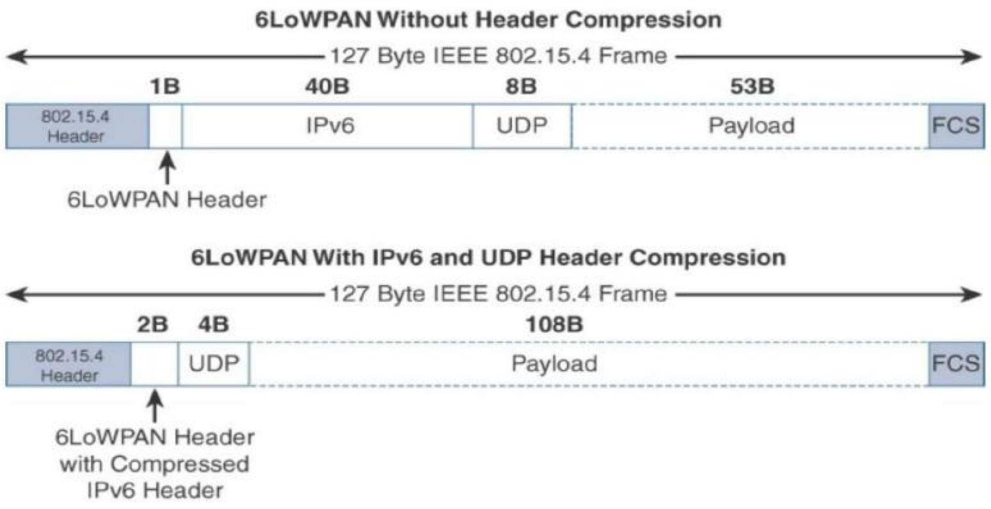

6LoWPAN Without Header Compression
127 Byte IEEE 802.15.4 Frame
1B
40B
8B
53B
802.15.4 Header
IPv6
UDP
Payload
FCS
6LoWPAN Header
6LoWPAN With IPv6 and UDP Header Compression
127 Byte IEEE 802.15.4 Frame
2B
4B
108B
802.15.4 Header
UDP
Payload
FCS
6LoWPAN Header with Compressed IPv6 Header

Figure : 6LoWPAN Header Compression

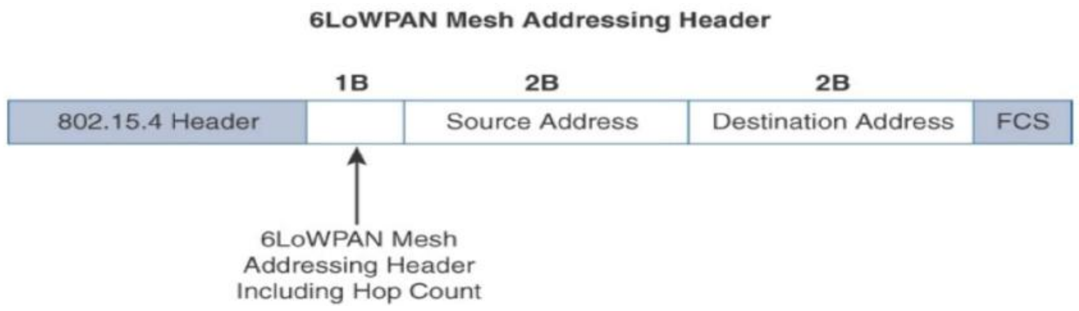
At the top of Figure above, you see a 6LoWPAN frame without any header compression enabled: The full 40- byte IPv6 header and 8-byte UDP header are visible. The 6LoWPAN header is only a single byte in this case. Notice that uncompressed IPv6 and UDP headers leave only 53 bytes of data payload out of the 127- byte maximum frame size in the case of IEEE 802.15.4.

The bottom half of Figure above shows a frame where header compression has been enabled for a best-case scenario. The 6LoWPAN header increases to 2 bytes to accommodate the compressed IPv6 header, and UDP has been reduced in half, to 4 bytes from 8. Most importantly, the header compression has allowed the payload to more than double, from 53 bytes to 108 bytes, which is obviously much more efficient. Note that the 2-byte header compression applies to intra-cell communications, while communications external to the cell may require some field of the header to not be compressed.

Mesh Addressing

The purpose of the 6LoWPAN mesh addressing function is to forward packets over multiple hops. Three fields are defined for this header: Hop Limit, Source Address, and Destination Address. Analogous to the IPv6 hop limit field, the hop limit for mesh addressing also provides an upper limit on how many times the frame can be forwarded. Each hop decrements this value by 1 as it is forwarded. Once the value hits 0, it is dropped and no longer forwarded.

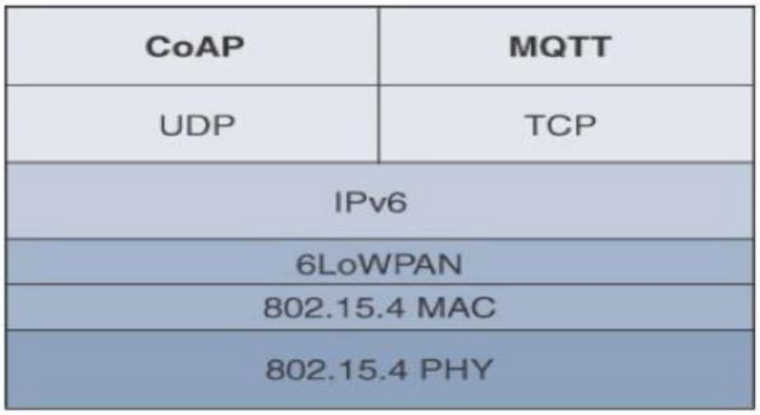
The Source Address and Destination Address fields for mesh addressing are IEEE 802.15.4 addresses indicating the endpoints of an IP hop. Figure below details the 6LoWPAN mesh addressing header fields.


6LoWPAN Mesh Addressing Header
1B
2B
2B
802.15.4 Header
Source Address
Destination Address
FCS
6LoWPAN Mesh Addressing Header Including Hop Count

Note that the mesh addressing header is used in a single IP subnet and is a Layer 2 type of routing known as mesh-under. RFC 4944 only provisions the function in this case as the definition of Layer 2 mesh routing specifications was outside the scope of the 6LoWPAN working group, and the IETF doesn't define "Layer 2 routing." An implementation performing Layer 3 IP routing does not need to implement a mesh addressing header unless required by a given technology profile.

IoT Application Layer Protocols (COAP ANS MQTT)

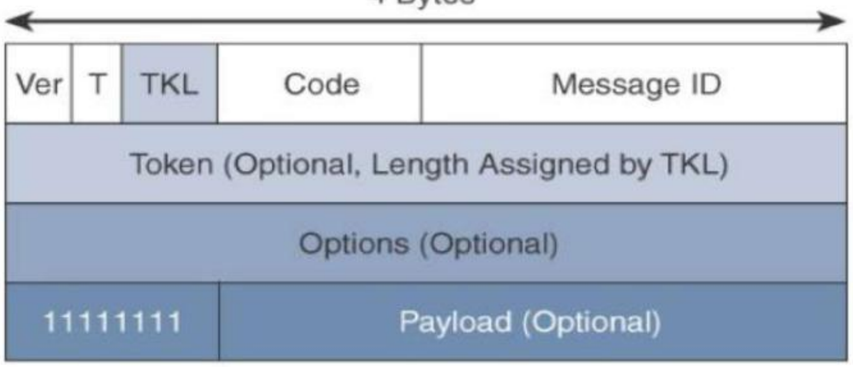
When considering constrained networks and/or a large-scale deployment of constrained nodes, verbose web-based and data model protocols, may be too heavy for IoT applications. To address this problem, the IoT industry is working on new lightweight protocols that are better suited to large numbers of constrained nodes and networks. Two of the most popular protocols are CoAP and MQTT. Figure below highlights their position in a common IoT protocol stack.



1. **CoAP (Constrained Application Protocol (CoAP)):**
   Constrained Application Protocol (CoAP) resulted from the IETF Constrained RESTful Environments (CoRE) working group's efforts to develop a generic framework for resource-oriented applications targeting constrained nodes and networks. The CoAP framework defines simple and flexible ways to manipulate sensors and actuators for data or device management.

   The CoAP messaging model is primarily designed to facilitate the exchange of messages over UDP between endpoints, including the secure transport protocol Datagram Transport Layer Security (DTLS).

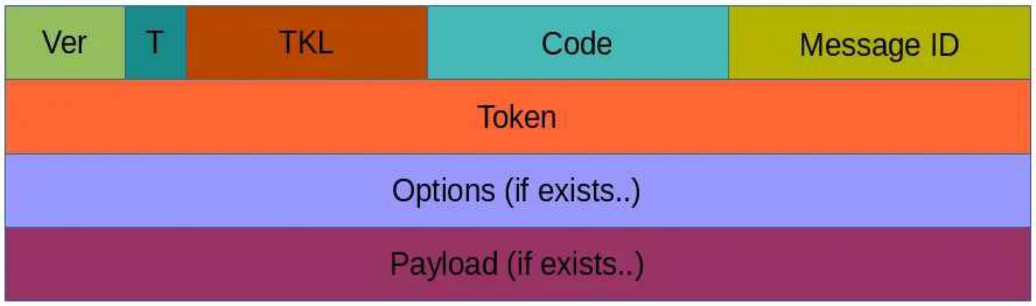
   From a formatting perspective, a CoAP message is composed of a short fixed-length Header field (4 bytes), a variable-length but mandatory Token field (0–8 bytes), Options fields if necessary, and the Payload field. Figure below details the CoAP message format, which delivers low overhead while decreasing parsing complexity.



CoAP Message Format

The CoAP message format is relatively simple and flexible. It allows CoAP to deliver low overhead, which is critical for constrained networks, while also being easy to parse and process for constrained devices.

CoAP Message Fields

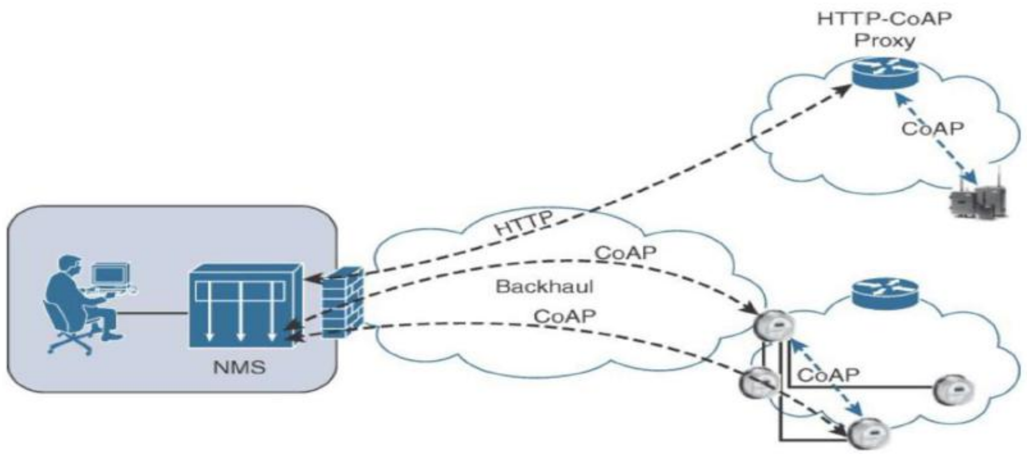


- **Ver**: It is a 2 bit unsigned integer indicating the version
- **T**: it is a 2 bit unsigned integer indicating the message type: 0 confirmable, 1 non-confirmable
- **TKL**: Token Length is the token 4 bit length
- **Code**: It is the code response (8 bit length)
- **Message ID**: It is the message ID expressed with 16 bit
- **Token** :With a length specified by TKL, correlates request and responses
- **Option** : specifies the option number, length and option value.
- **Payload** :carries the COAP application data.

CoAP can run over IPv4 or IPv6. However, it is recommended that the message fit within a single IP packet and UDP payload to avoid fragmentation. For IPv6, with the default MTU size being 1280 bytes and allowing for no fragmentation across nodes, the maximum CoAP message size could be up to 1152 bytes, including 1024 bytes for the payload. In the

case of IPv4, as IP fragmentation may exist across the network, implementations should limit themselves to more conservative values and set the IPv4 Don't Fragment (DF) bit.

CoAP communications across an IoT infrastructure can take various paths. Connections can be between devices located on the same or different constrained networks or between devices and generic Internet or cloud servers, all operating over IP. Proxy mechanisms are also defined, and RFC 7252 details a basic HTTP mapping for CoAP. As both HTTP and CoAP are IP-based protocols, the proxy function can be located practically anywhere in the network, not necessarily at the border between constrained and non-constrained networks.

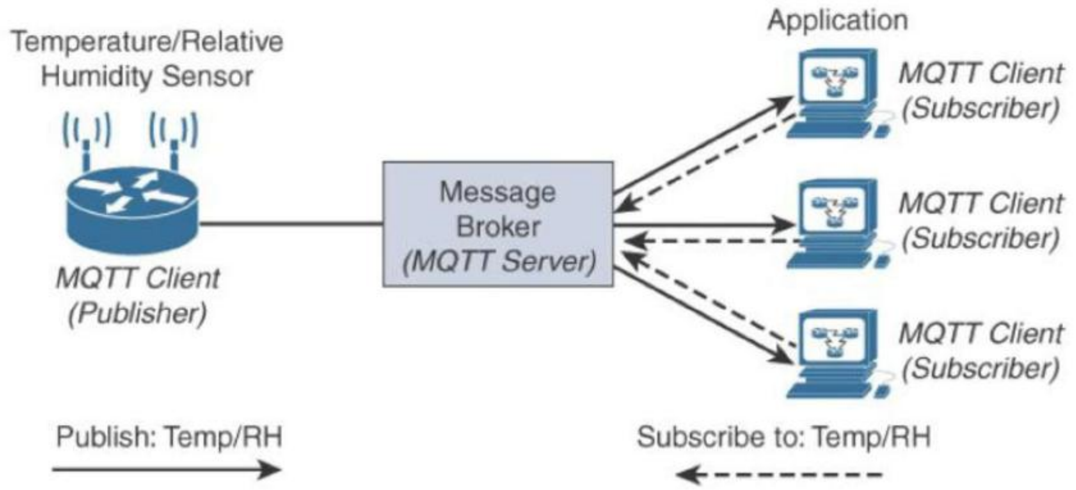


Just like HTTP, CoAP is based on the REST architecture, but with a “thing” acting as both the client and the server. Through the exchange of asynchronous messages, a client requests an action via a method code on a server resource. A uniform resource identifier (URI) localized on the server identifies this resource. The server responds with a response code that may include a resource representation. The CoAP request/response semantics include the methods GET, POST, PUT, and DELETE.

Message Queuing Telemetry Transport (MQTT):

At the end of the 1990s, engineers from IBM and Arcom (acquired in 2006 by Eurotech) were looking for a reliable, lightweight, and cost-effective protocol to monitor and control a large number of sensors and their data from a central server location, as typically used by the oil and gas industries. Their research resulted in the development and implementation of the Message Queuing Telemetry Transport (MQTT) protocol that is now standardized by the Organization for the Advancement of Structured Information Standards (OASIS).

The selection of a client/server and publish/subscribe framework based on the TCP/IP architecture, as shown in Figure below.

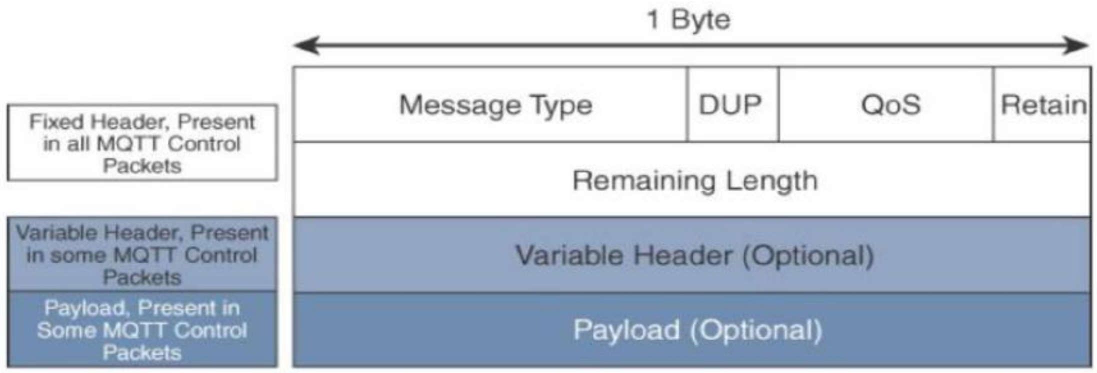


An MQTT client can act as a publisher to send data (or resource information) to an MQTT server acting as an MQTT message broker. In the example illustrated in Figure 2.22, the MQTT client on the left side is a temperature (Temp) and relative humidity (RH) sensor that publishes its Temp/RH data. The MQTT server (or message broker) accepts the network connection along with application messages, such as Temp/RH data, from the publishers. It also handles the subscription and un-subscription process and pushes the application data to MQTT clients acting as subscribers.

The application on the right side of Figure above is an MQTT client that is a subscriber to the Temp/RH data being generated by the publisher or sensor on the left. This model, where subscribers express a desire to receive information from publishers, is well known. A great example is the collaboration and social networking application Twitter.

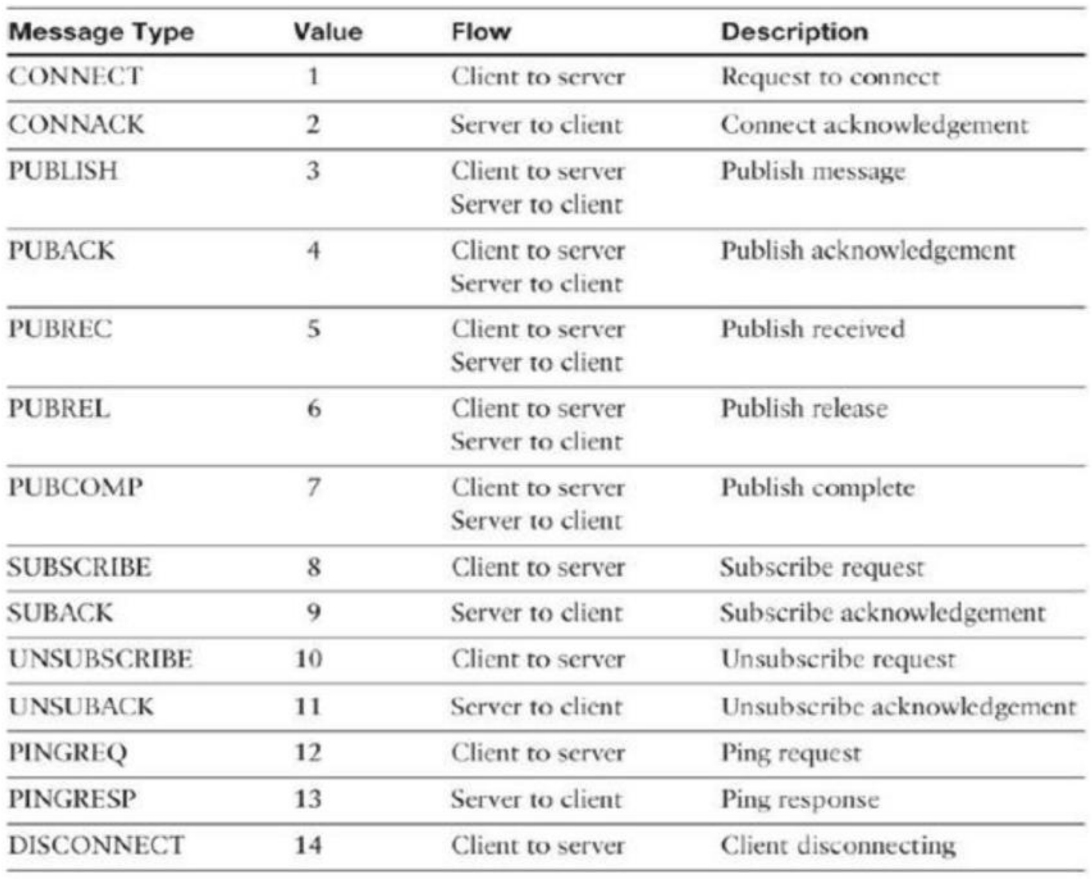
With MQTT, clients can subscribe to all data (using a wildcard character) or specific data from the information tree of a publisher. In addition, the presence of a message broker in MQTT decouples the data transmission between clients acting as publishers and subscribers. In fact, publishers and subscribers do not even know (or need to know) about each other. A benefit of having this decoupling is that the MQTT message broker ensures that information can be buffered and cached in case of network failures. This also means that publishers and subscribers do not have to be online at the same time. MQTT control packets run over a TCP transport using port 1883. TCP ensures an ordered, lossless stream of bytes between the MQTT client and the MQTT server. Optionally, MQTT can be secured using TLS on port 8883, and WebSocket (defined in RFC 6455) can also be used.

MQTT is a lightweight protocol because each control packet consists of a 2-byte fixed header with optional variable header fields and optional payload. You should note that a control packet can contain a payload up to 256 MB. Figure 2.23 provides an overview of the MQTT message format.


1 Byte
Fixed Header, Present in all MQTT Control Packets
Variable Header, Present in some MQTT Control Packets
Payload, Present in Some MQTT Control Packets
Message Type
DUP
QoS
Retain
Remaining Length
Variable Header (Optional)
Payload (Optional)

MQTT Message Format

Compared to the CoAP message format, MQTT contains a smaller header of 2 bytes compared to 4 bytes for CoAP. The first MQTT field in the header is Message Type, which identifies the kind of MQTT packet within a message. Fourteen different types of control packets are specified in MQTT version 3.1.1. Each of them has a unique value that is coded into the Message Type field. Note that values 0 and 15 are reserved. MQTT message types are summarized in Table 2.5.



Message Type	Value	Flow	Description
CONNECT	1	Client to server	Request to connect
CONNACK	2	Server to client	Connect acknowledgement
PUBLISH	3	Client to server Server to client	Publish message
PUBACK	4	Client to server Server to client	Publish acknowledgement
PUBREC	5	Client to server Server to client	Publish received
PUBREL	6	Client to server Server to client	Publish release
PUBCOMP	7	Client to server Server to client	Publish complete
SUBSCRIBE	8	Client to server	Subscribe request
SUBACK	9	Server to client	Subscribe acknowledgement
UNSUBSCRIBE	10	Client to server	Unsubscribe request
UNSUBACK	11	Server to client	Unsubscribe acknowledgement
PINGREQ	12	Client to server	Ping request
PINGRESP	13	Server to client	Ping response
DISCONNECT	14	Client to server	Client disconnecting

The next field in the MQTT header is DUP (Duplication Flag). This flag, when set, allows the client to notate that the packet has been sent previously, but an acknowledgement was not received. The QoS header field allows for the selection of three different QoS levels. The next field is the Retain flag. Only found in a PUBLISH message, the Retain flag notifies the server to hold onto the message data. This allows new subscribers to instantly receive the last known value without having to wait for the next update from the publisher.

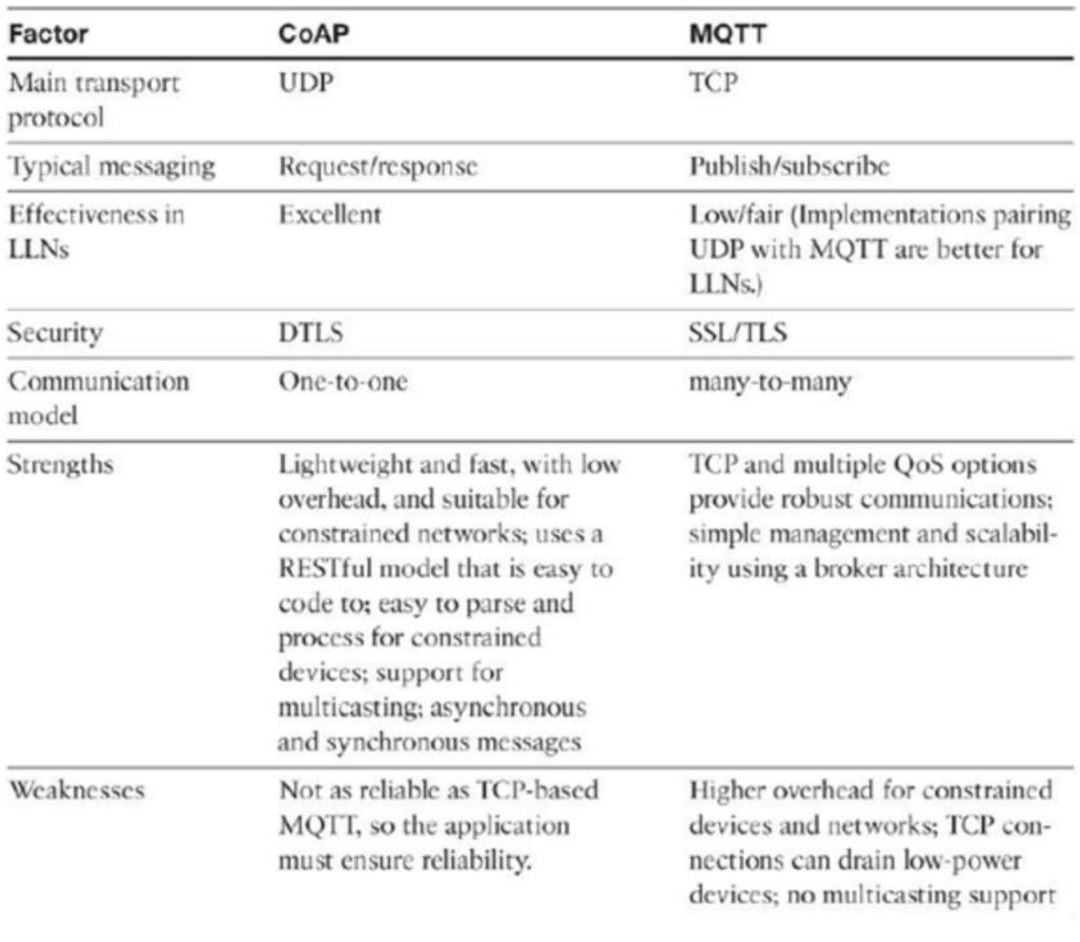
The last mandatory field in the MQTT message header is Remaining Length. This field specifies the number of bytes in the MQTT packet following this field.

MQTT sessions between each client and server consist of four phases: session establishment, authentication, data exchange, and session termination. Each client connecting to a server has a unique client ID, which allows the identification of the MQTT session between both parties. When the server is delivering an application message to more than one client, each client is treated independently.

Subscriptions to resources generate SUBSCRIBE/SUBACK control packets, while un-subscription is performed through the exchange of UNSUBSCRIBE/UNSUBACK control packets. Graceful termination of a connection is done through a DISCONNECT control packet, which also offers the capability for a client to reconnect by re-sending its client ID to resume the operations.

A message broker uses a topic string or topic name to filter messages for its subscribers. When subscribing to a resource, the subscriber indicates the one or more topic levels that are used to structure the topic name. The forward slash (/) in an MQTT topic name is used to separate each level within the topic tree and provide a hierarchical structure to the topic names.

Comparison of CoAP and MQTT

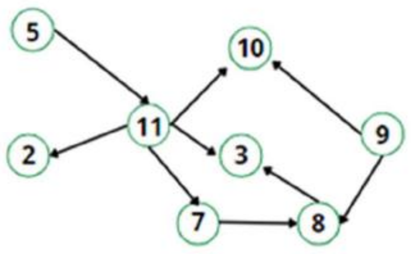


Factor	CoAP	MQTT
Main transport protocol	UDP	TCP
Typical messaging	Request/response	Publish/subscribe
Effectiveness in LLNs	Excellent	Low/fair (Implementations pairing UDP with MQTT are better for LLNs.)
Security	DTLS	SSL/TLS
Communication model	One-to-one	many-to-many
Strengths	Lightweight and fast, with low overhead, and suitable for constrained networks; uses a RESTful model that is easy to code to; easy to parse and process for constrained devices; support for multicasting; asynchronous and synchronous messages	TCP and multiple QoS options provide robust communications; simple management and scalability using a broker architecture
Weaknesses	Not as reliable as TCP-based MQTT, so the application must ensure reliability.	Higher overhead for constrained devices and networks; TCP connections can drain low-power devices; no multicasting support

RPL:(RPL (IPv6 Routing protocol):

RPL stands for Routing Protocol for Low Power and Lossy Networks for heterogeneous traffic networks. It is a routing protocol for Wireless Networks. This protocol is based on the same standard as by Zigbee and 6 Lowpan is IEEE 802.15.4 It holds both many-to-one and one-to-one communication. It is a Distance Vector Routing Protocol that creates a tree-like routing topology called the Destination Oriented Directed Acyclic Graph (DODAG), rooted towards one or more nodes called the root node or sink node.

The **Directed Acyclic Graphs** (DAGs) are created based on user-specified specific Objective Function (OF). The OF defines the method to find the best-optimized route among the number of sensor devices.



The IETF chartered the ROLL (Routing Over Low power and Lossy networks) working group to evaluate all three routing protocols and determine the needs and requirements for developing a routing solution for IP smart objects. After the study of various use cases and a survey of existing protocols, the consensus was that a new routing protocol should be developed for IP smart objects, given the characteristics and requirements of the constrained network. This new Distance Vector Routing Protocol was named the IPv6 Routing Protocol for Low power and Lossy networks(RPL). The RPL specification was published as RFC 6550 by the ROLL working group.

In an RPL Network, each node acts as a router and becomes part of a mesh network. Routing is performed at the IP Layer. Each node examines every received IPv6 packet and determines the next-hop destination based on the information contained in the IPv6 header. No information from the MAC layer header is needed to perform the next determination.

Modes of RPL:

This protocol defines two modes:

Storing mode: All modes contain the entire routing table of the RPL domain. Every node knows how to reach every other node directly.

Non-Storing mode: Only the border router(s) of the RPL domain contain(s) the full routing table. All other nodes in the domain maintain their list of parents only and use this as a list of default routes towards the border router. The abbreviated routing table saves memory space and CPU. When communicating in non-storing mode, a node always forwards its packet to the border router, which knows how to ultimately reach the final destination.

RPL is based on the concept of a Directed Acyclic Graph (DAG). A DAG is Directed Graph where no cycle exists. This means that from any vertex or point in the graph, we cannot follow an edge or a line back to this same point. All of the edges are arranged in a path oriented toward and terminating at one or more root nodes.

A basic RPL process involves building a Destination Oriented Directed Acyclic Graph (DODAG). A DODAG is a DAG rooted in one destination. In RPL this destination occurs at a border router known as the DODAG root. In a DODAG, three parents maximum are maintained by each node that provides a path to the root. Typically one of these parents is the preferred parent, which means it is the preferred next hop for upward roots towards the root. The routing graph created by the set of DODAG parents across all nodes defines the full set of upwards roots. RPL protocol information should ensure that routes are loop-free by disallowing nodes from selected DODAG parents positioned further away from a border router.

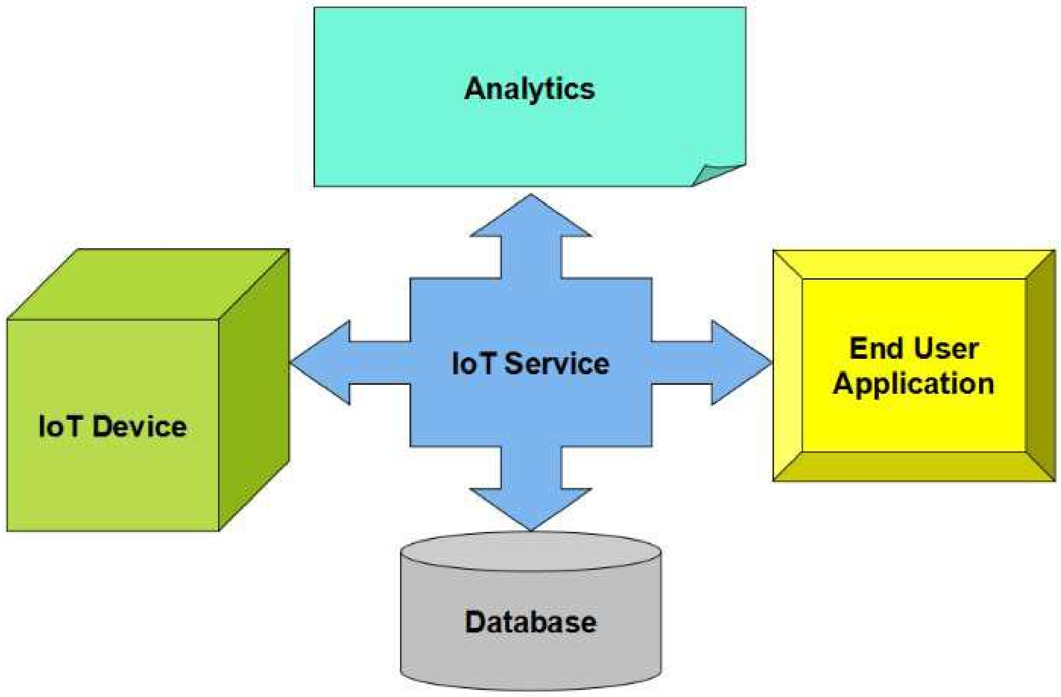
Implementation of RPL Protocol:

The RPL protocol is implemented using the Contiki Operating system. This Operating System majorly focuses on IoT devices, more specifically Low Power Wireless IoT devices. It is an Open source Model and was first bought into the picture by Adam Dunkel's.

The RPL protocol mostly occurs in wireless sensors and networks. Other similar Operating Systems include T-Kernel, EyeOS, LiteOS, etc.

IOT FRAMEWORK-THING SPEAK:

The Internet of Things(IoT) is a system of 'connected things'. The things generally comprise of an embedded operating system and an ability to communicate with the internet or with the neighbouring things. One of the key elements of a generic IoT system that bridges the various 'things' is an IoT service. An interesting implication from the 'things' comprising the IoT systems is that the things by themselves cannot do anything. At a bare minimum, they should have an ability to connect to other 'things'. But the real power of IoT is harnessed when the things connect to a 'service' either directly or via other 'things'. In such systems, the service plays the role of an invisible manager by providing capabilities ranging from simple data collection and monitoring to complex data analytics. The below diagram illustrates where an IoT service fits in an IoT ecosystem:



UNIT -IV
Device Discovery and Cloud Service for IoT

Device-management Gateway:

Device Management (DM) means provisioning for the device ID or address which is distinct from other resources, device activating, configuring (managing device parameters and settings), registering, deregistering, attaching and detaching. Device management also means accepting subscription for its resources. Device fault management means course of actions and guidelines to be followed in case if a fault
develops in the device.
Open Mobile Alliance (OMA)-DM and several standards are used for device management. OMA-DM model suggests the use of a DM server which interacts with devices through a gateway in case of IoT/M2M applications. A DM server is a server for assigning the device ID or address, activating, configuring (managing device parameters and settings), subscribing to device services or opting out of device services and configuring device modes. A device instead of a DM server, communicates to a gateway in case of low-power loss environment.
Gateway functions for device management are:

- Does forwarding function when the DM server and device can interact without reformatting or structuring
- Does protocol conversion when the device and DM server use distinct protocols
- Does proxy function in case an intermediate pre-fetch is required in a lossy environment or network environment needs.
- Data communication between personal/local area network of devices and a gateway for communicating via Internet.
- Gateway enables data enrichment and consolidation and device management.
- Data management functions at the gateway are transcoding, data privacy, data security, data enrichment, data consolidation, transformation and device management.
- Transcoding means adaptations, conversions, changes of protocol or format using software which renders the web response/messages in formats/representations as required and acceptable at the IoT device and rendering requests for messages in formats/representations as required and acceptable at the server.
- Data acquires and transfers to other end at scheduled intervals, on an event, or on polling.
- Data aggregation, compaction and fusion save energy during data dissemination.
- Data destinations may use the 48-bit MAC address, 32-bit IPv4 address, 48-bit IPv6 address or port number during communication at the data-link or network layers.
- Each device and application has an ID or address of communication source and each destination has an ID or address. Communication between the end points and between the layers is secure when using the authentication and authorisation processes.
- Device management functions are the device ID or address, activation, configuring (managing device parameters and settings), registering, deregistering, attaching, detaching and fault management.
- Gateway functions for device management are—forwarding function between DM server and device; protocol conversion when device and DM server use distinct protocols and proxy function.
- Communication gateway enables protocol conversion between two ends.

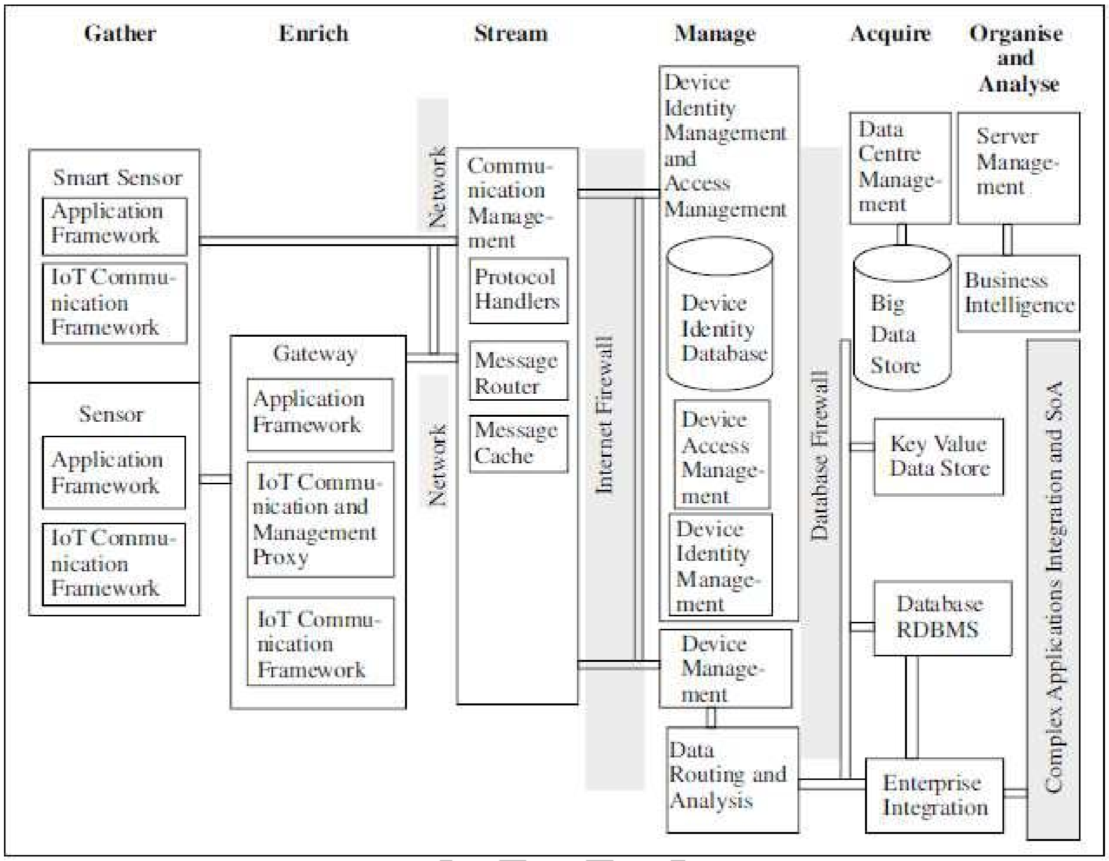


Fig: Oracle's IoT architecture (Device identity management means identifying a device, registering a device for actions after identifying, de-registering the device, assigning unique identity to the device. Device access management means enabling, disabling the device access, authenticating a device for access, authorizing a device for access to a subsystem. Chapter 2 will explain these in greater detail.)

An architecture has the following features:

- The architecture serves as a reference in applications of IoT in services and business processes.
- A set of sensors which are smart, capture the data, perform necessary data element analysis and transformation as per device application framework and connect directly to a communication manager.
- A set of sensor circuits is connected to a gateway possessing separate data capturing, gathering, computing and communication capabilities. The gateway receives the data in one form at one end and sends it in another form to the other end.
- The communication-management subsystem consists of protocol handlers, message routers and message cache.
- This management subsystem has functionalities for device identity database, device identity management and access management.
- Data routes from the gateway through the Internet and data centre to the application server or enterprise server which acquires that data.
- Organisation and analysis subsystems enable the services, business processes, enterprise integration and complex processes

REGISTERING A DEVICE:

Device management functions are assigning each device ID or address, device activation, configuring (managing device parameters and settings), registering, deregistering, attaching, detaching, subscription and fault management.

- Go to the **Registries page** in Google Cloud console.
- At the top of the page, click **Create a registry.**
- Enter **a Registry ID** and select a cloud region. For information on registry naming and size requirements, see Permitted characters and size requirements.
- Select the Protocols that devices in this registry will use to connect to Cloud IoT Core: MQTT, HTTP, or both.
- Select a Default telemetry topic or create a new one.
  The default topic is used for published telemetry events that don't have a subfolder or if the subfolder used doesn't have a matching Cloud Pub/Sub topic.
- (Optional) If you want to publish separate streams of data from a device, add more telemetry topics.
  Each topic maps to a subfolder specified in either the MQTT topic path or the HTTP request when the data is published.
  To create additional telemetry topics:

  a. Click Add more telemetry topics, then click Add topic and subfolder.

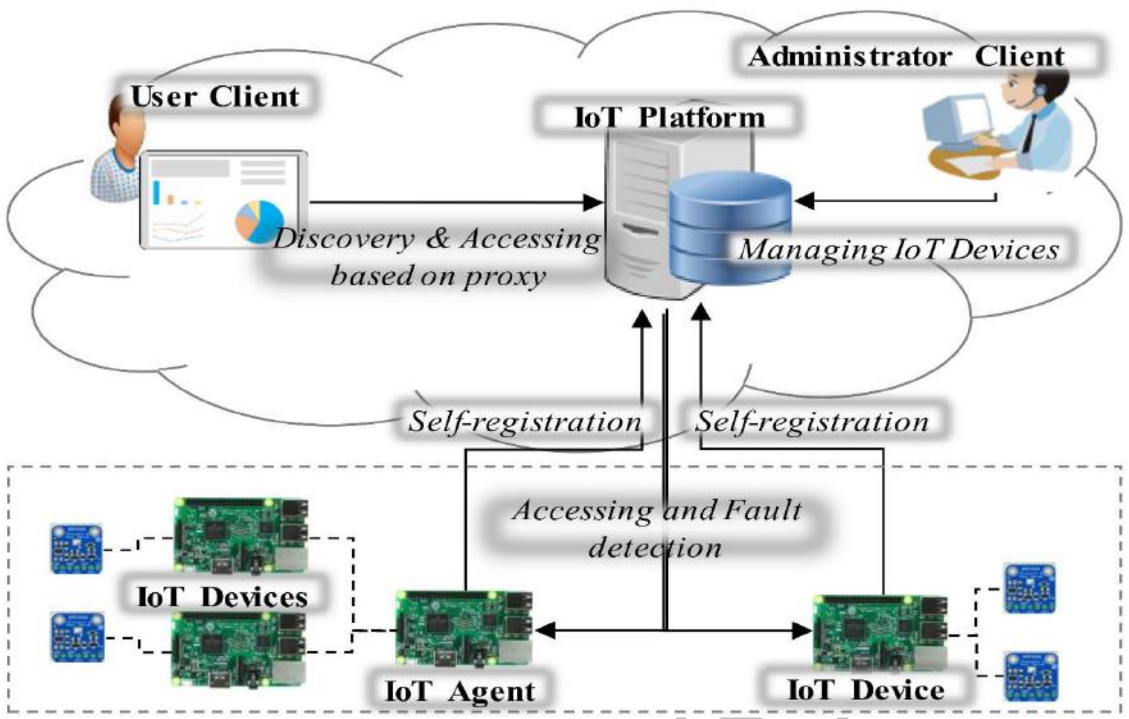
  b. Select a Pub/Sub topic or create a new one.

  c. Enter a descriptive subfolder name.

  For information on subfolder naming and size requirements, see Permitted characters and size requirements.
- (Optional) Select a Device state topic or create a new one. This topic can be the same as a telemetry topic, or can be used only for state data.
  State data is published to Cloud Pub/Sub on a best-effort basis: if publication to the topic fails, it will not be retried. If no topic is defined, device state updates are still persisted internally by Cloud IoT Core, but only the last 10 states are retained.
  For more information, see Getting device state.
- Click Create to continue.

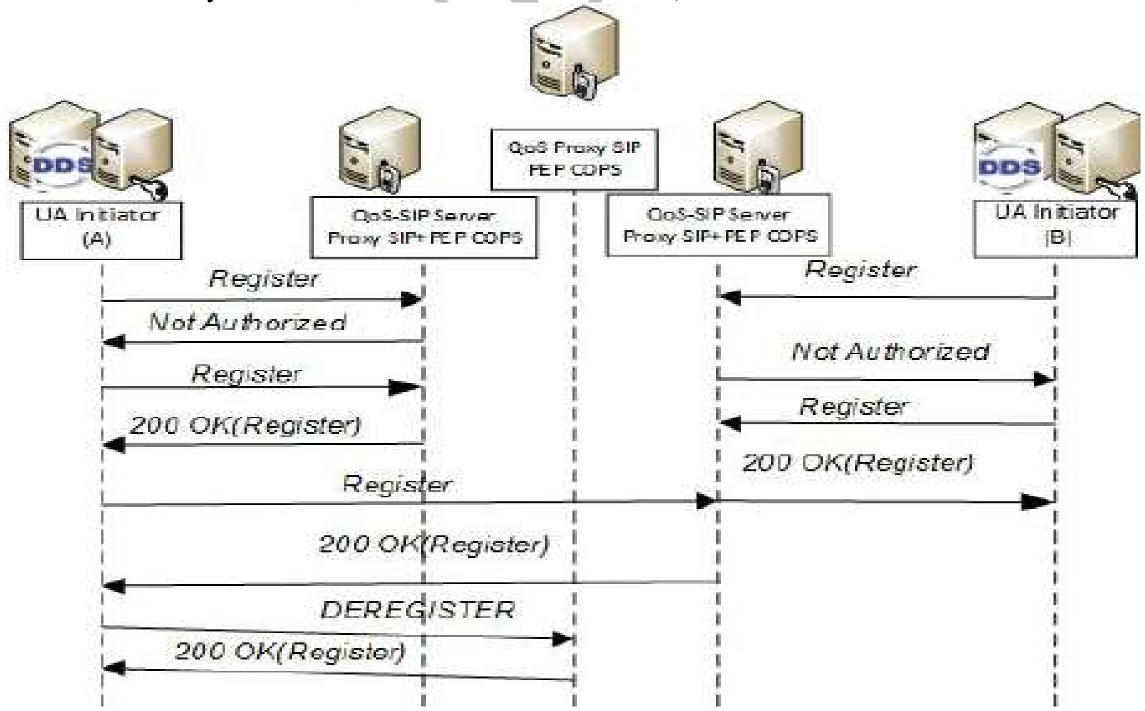
Getting device details:

1. Go to the **Registries page** in Google Cloud console.
2. Click the ID of the registry for the device.
3. In the menu on the left, click Devices.
4. Click the ID of the device to go to the Device details page. This page summarizes recent device activity, including the last time a message was published and the time of the most recent error. This page also shows the device numeric ID.
5. Click the Configuration & state history tab to see recent configuration versions and update times for the device.



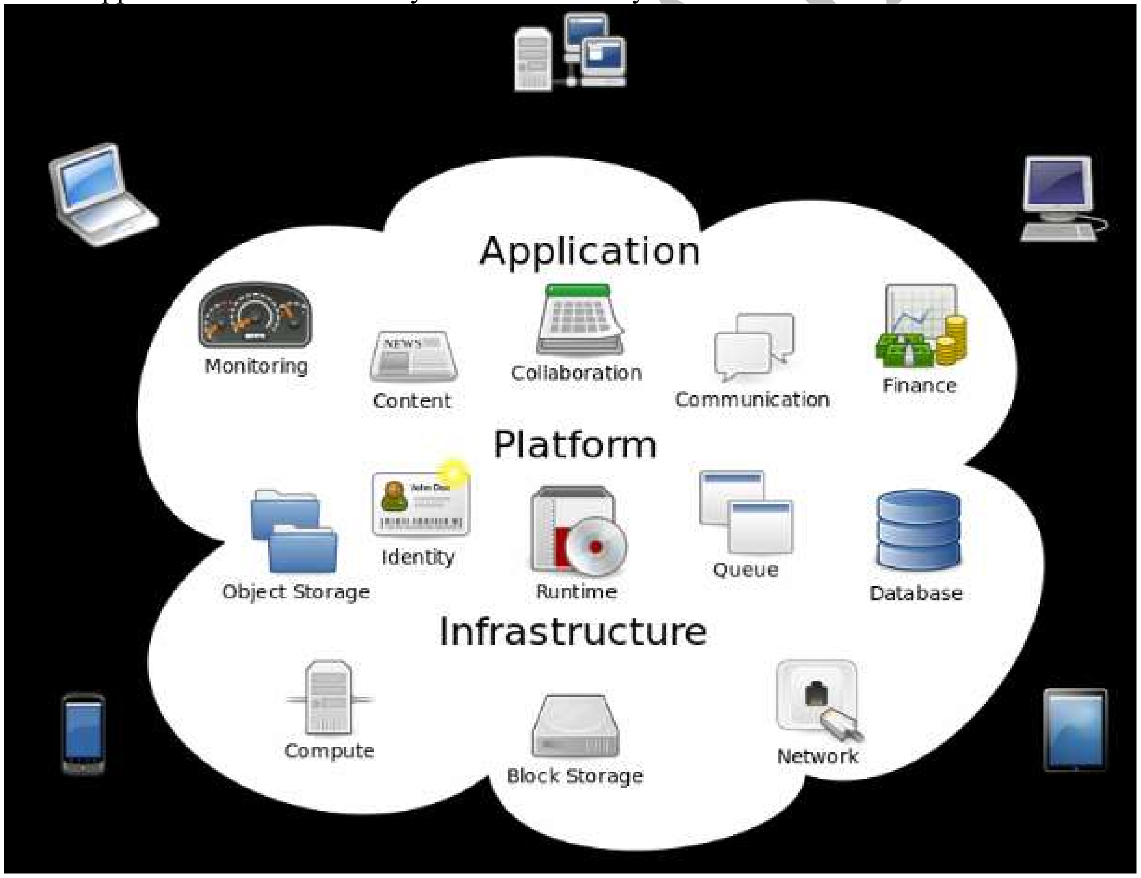
Deregistering a device:

1. Go to the Registries page in Google Cloud console.
2. Click the ID of the registry for the device.
3. In the menu on the left, click Devices.
4. Select each device you want to delete, then click Delete.
5. Confirm you want to delete the selected devices, then click Delete.



Cloud Storage models and communication APIs:

In truth, cloud computing and IoT are tightly coupled. The growth of IoT and the rapid development of associated technologies create a widespread connection of ―things. This has lead to the production of large amounts of data, which needs to be stored, processed and accessed. Cloud computing as a paradigm for big data storage and analytics. While IoT is exciting on its own, the real innovation will come from combining it with cloud computing. The combination of cloud computing and IoT will enable new monitoring services and powerful processing of sensory data streams. For example, sensory data can be uploaded and stored with cloud computing, later to be used intelligently for smart monitoring and actuation with other smart devices. Ultimately, the goal is to be able to transform data to insight and drive productive, cost-effective action from those insights. The cloud effectively serves as the brain to improved decision-making and optimized internet-based interactions. However, when IoT meets cloud, new challenges arise. There is an urgent need for novel network architectures that seamlessly integrate them. The critical concerns during integration are quality of service (QoS) and quality of experience (QoE), as well as data security, privacy and reliability. The virtual infrastructure for practical mobile computing and interfacing includes integrating applications, storage devices, monitoring devices, visualization platforms, analytics tools and client delivery. Cloud computing offers a practical utility-based model that will enable businesses and users to access applications on demand anytime and from anywhere.



Characteristics

First, the cloud computing of IoT is an on-demand self service, meaning it‘s there when you need it. Cloud computing is a web-based service that can be accessed without any special assistance or permission from other people; however, you need at minimum some sort of internet access.

Second, the cloud computing of IoT involves broad network access, meaning it offers several connectivity options. Cloud computing resources can be accessed through a wide variety of internet-connected devices such as tablets, mobile devices and laptops. This level of convenience means users can access those resources in a wide variety of manners, even from older devices. Again, though, this emphasizes the need for network access points.

Third, cloud computing allows for resource pooling, meaning information can be shared with those who know where and how (have permission) to access the resource, anytime and anywhere. This lends to broader collaboration or closer connections with other users. From an IoT perspective, just as we can easily assign an IP address to every "thing" on theplanet, we can share the "address" of the cloud-based protected and stored information with others and pool resources.

Fourth, cloud computing features rapid elasticity, meaning users can readily scale the service to their needs. You can easily and quickly edit your software setup, add or remove users, increase storage space, etc. This characteristic will further empower IoT by providing elastic computing power, storage and networking.

Finally, the cloud computing of IoT is a measured service, meaning you get what you pay for. Providers can easily measure usage statistics such as storage, processing, bandwidth and active user accounts inside your cloud instance. This pay per use (PPU) model means your costs scale with your usage. In IoT terms, it's comparable to the ever-growing network of physical objects that feature an IP address for internet connectivity, and the communication that occurs between these objects and other internet-enabled devices and systems; just like your cloud service, the service rates for that IoT infrastructure may also scale with use.

Service and Deployment

Service models

Service delivery in cloud computing comprises three different service models:

a. Software as a service (SaaS),
b. Platform as a service (PaaS),
c. Infrastructure as a service (IaaS).

Software as a service (SaaS) provides applications to the cloud‘s end user that are mainly accessed via a web portal or service-oriented architecture-based web service technology. These services can be seen as ASP (application service provider) on the application layer. Usually, a specific company that uses the service would run, maintain and give support so that it can be reliably used over a long period of time.

Platform as a service (PaaS) consists of the actual environment for developing and provisioning cloud applications. The main users of this layer are developers that want to develop and run a cloud application for a particular purpose. A proprietary language was supported and provided by the platform (a set of important basic services) to ease communication, monitoring, billing and other aspects such as startup as well as to ensure an application‘s scalability and flexibility. Limitations regarding the programming languages supported, the programming model, the ability to access resources, and the long-term persistence are possibledisadvantages.

Infrastructure as a service (IaaS) provides the necessary hardware and software upon which a customer can build a customized computing environment. Computing resources, data storage resources and the communications channel are linked together with these essential IT resources to ensure the stability of applications being used on the cloud. Those stack models can be referred to as the medium for IoT, being used and conveyed by the users in different methods for the greatest chance of interoperability. This includes connecting cars, wearables, TVs, smartphones, fitness equipment, robots, ATMs, and vending machines as well as the vertical applications, security and professional services, and analytics platforms that come withthem.

Deployment

Deployment models
Deployment in cloud computing comprises four deployment models: private cloud, public cloud, community cloud and hybrid cloud.
A private cloud has infrastructure that‘s provisioned for exclusive use by a single organization comprising multiple consumers such as business units. It may be owned, managed and operated by the organization, a third party or some combination of them, and it may exist on or off premises.
A public cloud is created for open use by the general public. Public cloud sells services to anyone on the internet. (Amazon Web Services is an example of a large public cloud provider.) This model is suitable for business requirements that require management of load spikes and the applications used by the business, activities that would otherwise require greater investment in infrastructure for the business. As such, public cloud also helps reduce capital expenditure and bring down operational ITcosts.
A community cloud is managed and used by a particular group or organizations that have shared interests, such as specific security requirements or a common mission.
Finally, a hybrid cloud combines two or more distinct private, community or public cloud infrastructures such that they remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability. Normally, information that‘s not critical is outsourced to the public cloud, while business-critical services and data are kept within the control of the organization.
CLOUD STORAGE API:
A cloud storage API is an application program interface that connects a locally-based application to a cloud-based storage system, so that a user can send data to it and access and work with data stored in it. To the application, the cloud storage system is just another target device, like tape or disk-based storage. An application program interface (API) is code that allows two software programs to communicate with each other. The API defines the correct way for a developer to write a program that requests services from an operating system (OS) or other application. APIs are implemented by function calls composed of verbs and nouns. The required syntax is described in the documentation of the application being called.
How APIs work
APIs are made up of two related elements. The first is a specification that describes how information is exchanged between programs, done in the form of a request for processing and a return of the necessary data. The second is a software interface written to that specification and published in some way for use.The software that wants to access the features and capabilities of the API is said to call it, and the software that creates the API is said to publish it.
Why APIs are important for business
The web, software designed exchange information via the internet and cloud computing have all combined to increase the interest in APIs in general and services in particular. Software that was once custom-developed for a specific purpose is now often written referencing APIs that provide broadly useful features, reducing development time and cost and mitigating the risk of errors.APIs have steadily improved software quality over the last decade, and the growing number of web services exposed through APIs by cloud providers is also encouraging the creation of cloud-specific applications, internet of things (IoT) efforts and apps to support mobile devices and users.
Three basic types of APIs
APIs take three basic forms: local, web-like and program-like:
1. **Local APIs** are the original form, from which the name came. They offer OS or middleware services to application programs. Microsoft's .NET APIs, the TAPI

(Telephony API) for voice applications, and database access APIs are examples of the local API form.

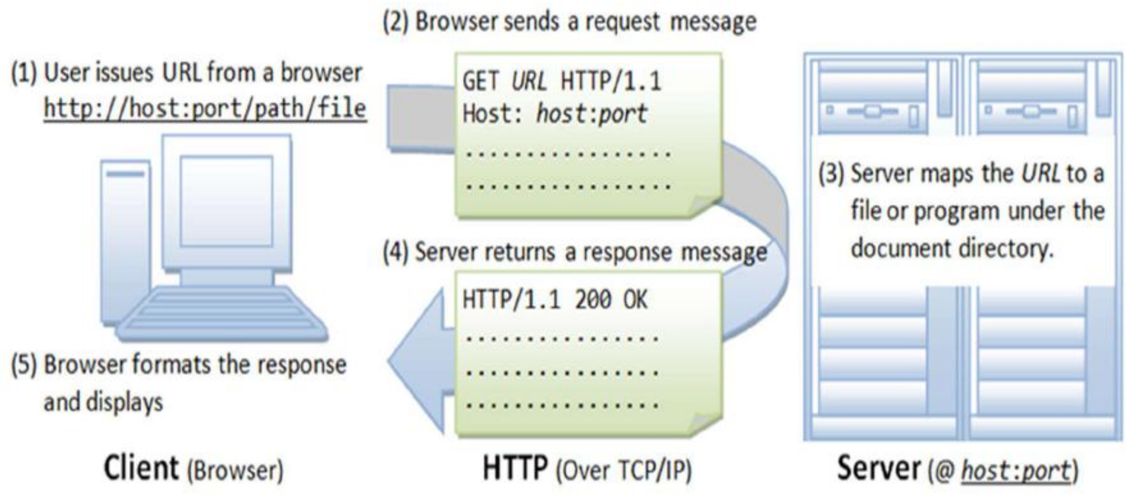
2. **Web APIs** are designed to represent widely used resources like HTML pages and are accessed using a simple HTTP protocol. Any web URL activates a web API. Web APIs are often called REST (representational state transfer) or RESTful because the publisher of REST interfaces doesn't save any data internally between requests. As such, requests from many users can be intermingled as they would be on the internet.
3. **Program APIs** are based on remote procedure call (RPC) technology that makes a remote program component appear to be local to the rest of the software. Service oriented architecture (SOA) APIs, such as Microsoft's WS-series of APIs, are program APIs.

Web server for IoT:

Any computer that can implement http or https is able to play the role of a web server. Http is a protocol, a way of communication which supplies web pages. It is pretty widely used and easy to implement. Through http you can transfer html and create simple user interfaces, it can implement Java Script and make more complicated web pages and it is available in most of the browsers. One of the great qualities of this protocol is that it replaced complicated and heavy displays with user friendly web pages.

How does it work? The browser sends a request to the server who searches the demanded page and returns it to the browser for the user. The request will consist of information about the kind of browser that is used, about the computer or about the document requested. It will have a method, a URL, a query string and the upload body in case you want data to be sent to the server.

The response will include the status, which tells the browser if the page was found or not (the errors among the 400s are about a not found page, 300 are redirections and 200s are confirmations of the page being found).


(2) Browser sends a request message
(1) User issues URL from a browser
http://host:port/path/file
GET URL HTTP/1.1
Host: host:port
(3) Server maps the URL to a file or program under the document directory.
(4) Server returns a response message
HTTP/1.1 200 OK
(5) Browser formats the response and displays
Client (Browser)
HTTP (Over TCP/IP)
Server (@ host:port)

Https has two important security roles:

- It encrypts the data. The request and the response will be both encrypted on sending and decrypted when read.

- The server is always asked for a certificate of authenticity before it is asked for a page. This prevents against stolen data through false web pages**.**

What does a query consist of? It will always look like this: http://address: [port]URL? Query string. The port can be absent, in which case it will be 80 for http and 443 for https. It has to be specified if it is not one the two. Concerning the URL, when it is not written, the default will be. The available methods in http are: get, post, put and delete. The main ones being the first two.

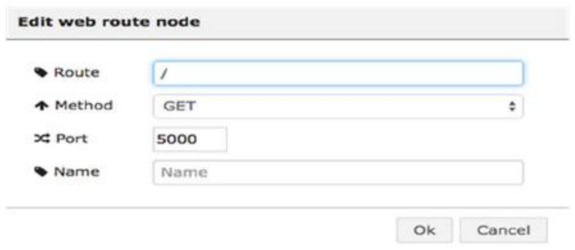
- Get method needs no upload body. It will only ask for data from the server and send only the headers, the address, the URL.
- Post sends important data to the server, which will be uploaded. Post has the role of modifying data on the server. The response of both these methods is the page and any additional information that was requested.
- Put is similar to post, only that in the semantic way, this method only creates an object on the server.
- Delete also plays a semantic part. It needs no upload body and it deletes objects on the server. The same action can be performed however using get.

On one server there can be more than one website, which means that, if the host is not specified in the request, the response may not be the one the browser expects.
Also, the response may have more than text. Any additional feature: images, JavaScript objects and so on will need a new request, so the process will be slowed down.

Webservers on gadgets using Wyliodrin:
The boards are non-powerful computers. With wyliodrin there is no need to install any software or make any configuration on the boards to run a webserver on them.

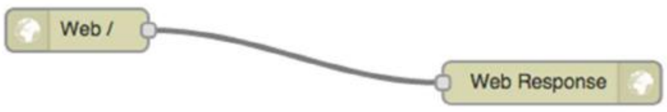
To create a webserver in wyliodrin you will need a web node. The simplest way to use a web page in this particular way is to send static files. In the project files, create a new folder static. Everything inside it will be sent back to the browser by the server, regardless of the fact that they are html, Java Script or CSS files. Images can be added as well, but they will definitely make the process slower. There are other ways of adding an image. For example, by using a storage system and including the images from there. This method will solve speed and memory issues.

The web node: The route option is actually the URL. The webserver will be active when it stumbles upon the specified route. After words you choose the method, and write the port to setup the server. This port will only be used once, in the beginning.

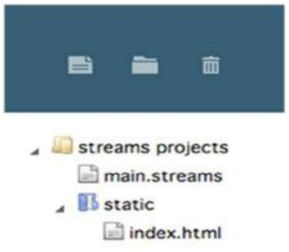

Edit web route node

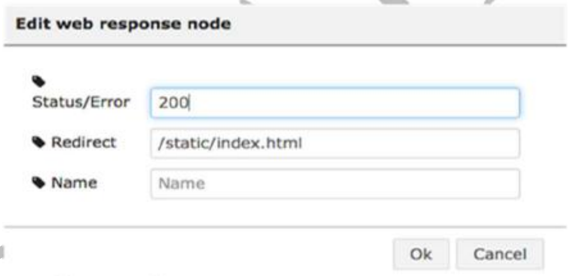
Route	/
Method	GET
Port	5000
Name	Name

The payload goes either in the query string for the get method or in the upload data for post. The message is built on this payload, on two mandatory variables: res which stands for the response and req which is the request. Without the last two, the server won't be able to provide a response.



The web response node: The message received by this node comes from one web node. For a web response the simple way is to make a redirection. Which means, in the redirect field, you can write the path to one of the static files and the browser will be sent to this page. On top of these, you will need the board's IP address which might not be public unless it is in the same network with the web server.




Edit web response node

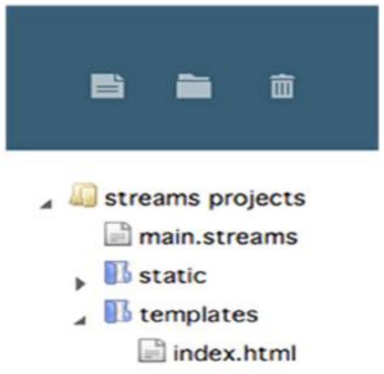
Status/Error: 200

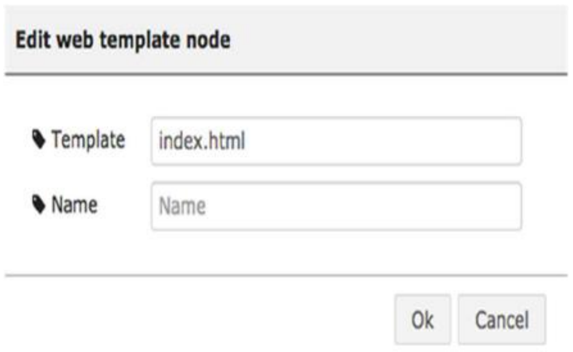
Redirect: /static/index.html

Name: Name

As a solution, IOT servers have a public adrdress. The port for these servers can be either 80 for http or 443 for https. The user accesses the public page, through the IOT server which is connected to Wyliodrin as well as the board. Now the problem with the board and the web being in the same network is solved as both can communicate with Wyliodrin.

Web templates: Just as for the static files, you will need a templates folder. This time, when you use the node, you don't need the whole path. You can only write the name of the file in the templates folder. What does the node do? It processes the response, meaning it loads the values plus the payload in it and sends it back to the browser. The values need to be in between two sets of curly brackets {{}}. Note that the values won't update unless the page is reloaded.




Edit web template node

Template: index.html

Name: Name

Web services:

A long time ago, the web services were more complicated. Now the application only requests the web server for the data, and it is the browser's job to rearrange it so that it is in the right format for the application.

How to implement it into a Wyliodrin application? Using a simple web response and web server node, you send a static page to the user and each time you make a query, instead of a template, you use a web response node and send the payload to the browser, which can be a number, an object or anything else.

JQuery:

There is a library called JQuery, based on JavScript, thus available in any browser, which can make function calls to the server.

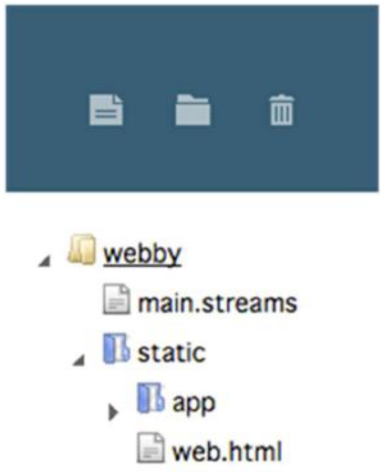
Case study : You have the following situation: you change the payload into a variable which stores values from a sensor. You want this variable to be shown in your web page. Practically, when an API gets called, what you will do, will be to make a get request to the server using the web address that you want with the URL /sensor . The web page will send values that you will store in a variable in your html file.

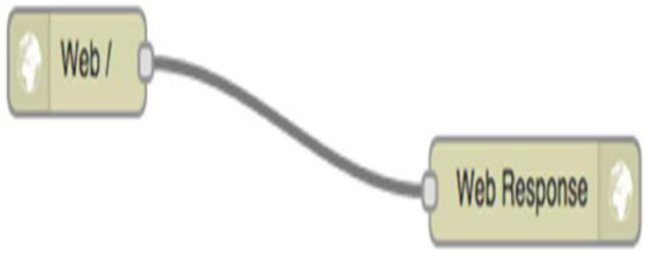
Web sockets:

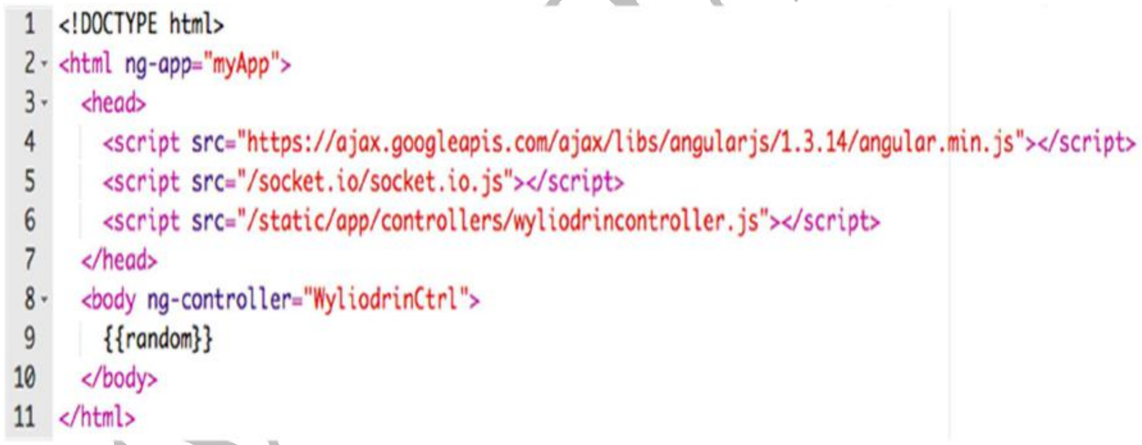
A web socket is based on the http or https protocol. It builds a connection between the browser and the server, so that either one can send data. When the browser makes a request, the server recognises the socket and doesn't close the connection. The two parties send the packages they need to send. If the server does not know how to work with sockets, the socket io will go back to querying.

AngularJS:

AngularJS is a library through which you can build browser applications. In the next example, every web node will create a new socket and serve a static web page. If you include in the response a variable, and this variable changes, wyliodrin controller will be notified every time this kind of novelty appears and AngularJS will replace the old value of the variable with the new one, creating a dynamic web page.







```
<!DOCTYPE html>
<html ng-app="myApp">
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
    <script src="/socket.io/socket.io.js"></script>
    <script src="/static/app/controllers/wyliodrincontroller.js"></script>
  </head>
  <body ng-controller="WyliodrinCtrl">
    {{random}}
  </body>
</html>
```

UNIT -V

UAV IoT

Introduction To Unmanned Aerial Vehicles/Drones:

An unmanned aerial vehicle (UAV), commonly known as a drone, is an aircraft without any human pilot, crew, or passengers on board. UAVs were originally developed through the twentieth century for military missions too "dull, dirty or dangerous for humans, and by the twenty-first, they had become essential assets to most militaries. As control technologies improved and costs fell, their use expanded to many non-military applications. These include aerial photography, precision agriculture, forest fire monitoring, river monitoring, environmental monitoring, policing and surveillance, infrastructure inspections, smuggling, product deliveries, entertainment, and drone racing.

- Any aircraft or flying machine operated without a human pilot such machines is called an unmanned aerial vehicle (UAV). It can be guided autonomously or remotely by a human operator using onboard computers and robots.
- During surveillance or military operation, UAVs can be a part of an unmanned aircraft system (UAS), Drones are separately for air and water.
- Drones have become increasingly popular in recent years. They are used for a variety of purposes, including photography, videography, surveying, inspection, and even delivery. But have you ever wondered how drones work? In this blog post, we'll take a look at the working principle of drones
- The basic components of a drone are the frame, motors, propellers, battery, flight controller, and sensors. Let's take a closer look at each of these components.

  Frame
  Battery
  Flight controllers
  Sensors
  Motors and Propellers

Subjects for Drone or UAV:

Understanding and development of drones depend on many subjects. The design of drone for a particular application comprises many factors like the aerodynamic shape of propellors, strength and weight of drone parts, electric motor, electric speed controller, radio transmitter or receiver, and software interface on mobile or computer for monitoring and data analysis.

Fluid Dynamics or Aerodynamics:

- Fluid dynamics plays an important role to decide the forces acting on the body of a drone
- The shape, size, and speed of the propeller and drone depending on the aerodynamics of propellers or blades
- Computational Fluid Dynamics (CFD) modelling helps for flow dynamics of airflow over drones
- CFD modelling of turbo-machinery) is essential to decide the amount of thrust generated by propellors

- Wind tunnel testing of the aerofoil blade of the drone is still important for testing CFD results

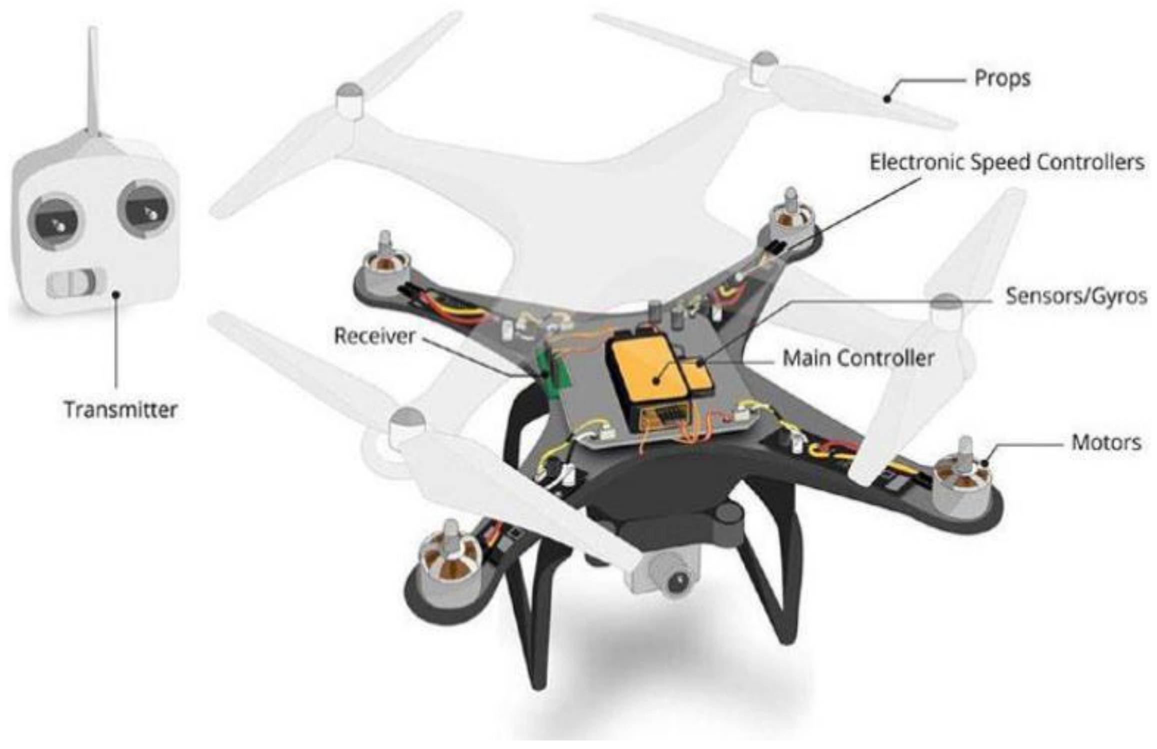
Mechanical Design

- Rigid body dynamics to study the motion and forces acting on drones
- Strength of materials
- Low weight and rigid materials are selected for drone

Electronics and Electrical Components:

- Electric motor with and without brush is required to drive the propellors
- Electronic Speed Controller
- Flight controller unit and computer processors

Radio Communication: transmitter and receiver for radio signals
Battery: Low weight and high-power wattage battery is important
Software-based interface: data collection and analysis using mobile or computer



Drone Types:

A drone is an unmanned aerial vehicle (UAV) or unmanned aircraft system. It is essentially a flying robot this is controlled remotely or can fly autonomously with software-controlled flight plans embedded in its system that work in conjunction with sensors and a global positioning system (GPS). Drones are of different types and sizes and are used for a variety of purposes.

Types of Drones

Here’s a rundown of the four main types of drones, their uses, their strengths and weaknesses:

1. Multi-Rotor Drones
2. Fixed-Wing Drones
3. Single-Rotor Drones
4. Fixed-Wing Hybrid VTOL

Multi-Rotor Drones

Multi-rotor drones are the easiest and cheapest option for getting an ‘eye in the sky.’ They also offer greater control over position and framing, and hence they are perfect for aerial photography and surveillance. They are called multi-rotor because they have more than one motor, more commonly troopers (3 rotors), quadcopters (4 rotors), hex copters (6 rotors) and octocopters (8 rotors), among others. By far, quadcopters are the most popular multi-rotor drones.

Advantages:

- It provides better control of the aircraft during the flight.
- Due to its increased manoeuvrability, it can move up and down on the same vertical line, back to front, side to side and rotate in its own axis.
- It has the ability to fly much more closely to structures and buildings.
- The ability to take multiple payloads per flight increases its operational efficiency and reduces the time taken for inspections.

Disadvantages:

- Multi-rotor drones have limited endurance and speed, making them unsuitable for large scale aerial mapping, long-endurance monitoring and long-distance inspection such as pipelines, roads and power lines.
- They are fundamentally very inefficient and require a lot of energy just to fight gravity and keep them in the air.
- With the current battery technology, they are limited to around 20-30 minutes when carrying a lightweight camera payload. However, heavy-lift multi-rotors are capable of carrying more weight, but in exchange for much shorter flight times.
- Due to the need for fast and high-precision throttle changes to keep them stabilised, it isn’t practical to use a gas engine to power multi-rotors, so they are restricted to electric motors. So until a new power source comes along, we can only expect very small gains in flight time.

Technical Uses:

- Visual inspections
- Thermal reports
- Photography & Videography
- 3D scans

Fixed-Wing Drones:

A fixed-wing drone has one rigid wing that is designed to look and work like an aeroplane, providing the lift rather than vertical lift rotors. Hence, this drone type only needs the energy to move forward and not to hold itself in the air. This makes them energy-efficient.

Advantages:

- Fixed-wing drones cover longer distances, map much larger areas, and loiter for long times monitoring their point of interest. The average flight time is a couple of hours. But with a greater energy density of fuel (gas engine powered), many fixed-wing UAVs can stay aloft for 16 hours or more.
- This drone type can fly at a high altitude, carry more weight and are more forgiving in the air than other drone types.

Disadvantages:

- Fixed-wing drones can be expensive.
- Training is usually required to fly fixed-wing drones. The first time you launch a fixed-wing drone, you need to be confident in your abilities to control through the flight and back to a soft landing. A fixed-wing drone is always moving forward, and they move a lot quicker than a multi-rotor, and hence you might not get a chance to put it into a hover. In most cases, a launcher is needed to get a fixed-wing drone into the air.
- With fixed-wing, the flight is just the beginning. The hundreds and thousands of captured images have to be processed and stitched together into one big tiled image. There is a lot more to be done after this, including performing data analysis, such as the stockpile volume calculations, tree counts, overlaying other data onto the maps, and so on.

Technical Uses:

- Aerial Mapping
- Drone Surveying – Forestry/Environmental Drone Surveys, Pipeline UAV Surveys, UAV Coastal Surveys
- Agriculture
- Inspection
- Construction
- Security

Single-Rotor Drones:

Single-rotor drone types are strong and durable. They look similar to actual helicopters in structure and design. A single-rotor has just one rotor, which is like one big spinning wing, plus a tail rotor to control direction and stability.

Advantages:

- A single-rotor helicopter has the benefit of much greater efficiency over a multi-rotor, which increases if the drone is gas-powered for even longer endurance.
- A single-rotor helicopter allows for very long blades, which are more like a spinning wing than a propeller, giving great efficiency.
- If you need to hover with a heavy payload (e.g. an aerial LIDAR laser scanner) or have a mixture of hovering with long endurance or fast forward flight, then a single-rotor helicopter is really your best bet.
- They are built to be strong and durable.

Disadvantages:

- Single-rotor drone types are complex and expensive.
- They vibrate and aren’t as stable or forgiving in the event of a bad landing.
- They also require a lot of maintenance and care due to their mechanical complexity.
- The long, heavy spinning blades of a single rotor can be dangerous.

Technical Uses:

- Aerial LIDAR laser scan
- Drone surveying
- Carrying heavy payloads

Fixed-Wing Hybrid VTOL

Hybrid VTOL drone types merge the benefits of fixed-wing and rotor-based designs. This drone type has rotors attached to the fixed wings, allowing it to hover and take off and land vertically. This new category of hybrids are only a few on the market, but as technology advances, this option can be much more popular in the coming years. One example of fixed-wing hybrid VTOL is Amazon's Prime Air delivery drone.

Advantages:

- The autopilot can do all the hard work of keeping the drone stable, leaving the human pilot the easier task of guiding it around the sky.
- Hybrid VTOL drones offer you the best of both worlds – fixed-wing & rotor-based designs.
- They are perfect at either hovering or forward flight.

Disadvantages:

- Only a handful of fixed-wing hybrid VTOLs are currently on the market
- The technology used in these drone types is still in the nascent stage.

Technical Uses:

- Drone Delivery

Other Significant Drone Types

Some of the popular drone types other than the ones mentioned above include:

1. **Small-Drones**
   These drone types are used for recreational purposes; they cannot perform commercial functions that other drone models carry out. Small drones are too light and lack the stability required for accurately capturing images.
2. **Micro-Drones**
   These are small drones, but they can still provide valuable intelligence because of their micro cameras. The British military commonly uses this drone, and it's called the Black Hornet. Black Hornets can fly up to 25 minutes (single charge) and have a range of up to one mile.

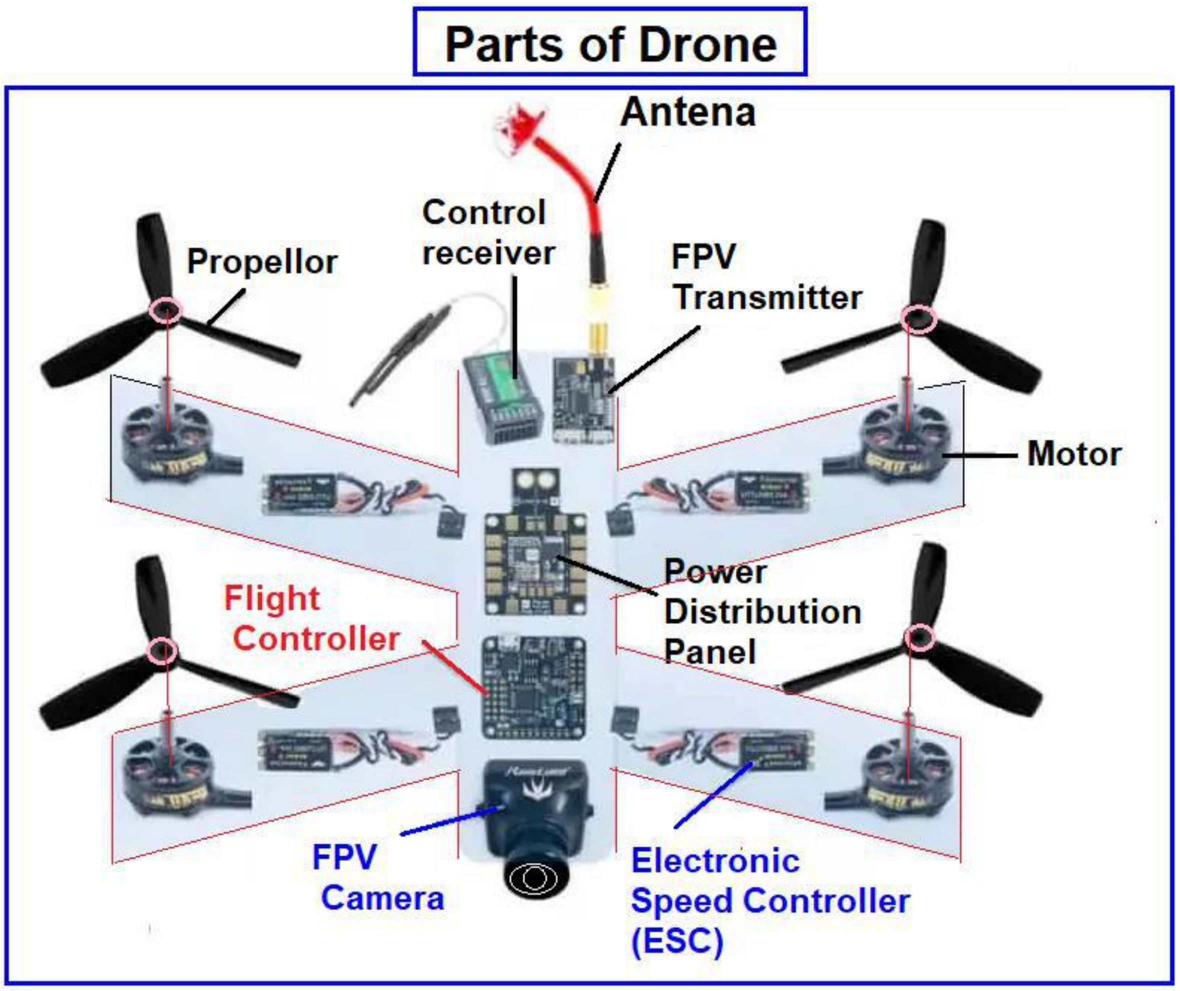
3. **Tactical-Drones**
   These drones are large without being bulky. Equipped with GPS technology and infrared cameras, they measure 4.5 feet and weigh 4.2 lbs. They are often used for surveillance work.
4. **Reconnaissance Drones**
   These drones measure approximately 16 feet in length, over 2200 pounds, and hover for 52 hours at 35,000 feet. They can be launched from the ground and are known as High Altitude Long Endurance drones (HALE) and Medium Altitude Long Endurance drones (MALE).
5. **Large-Combat-Drones**
   These drone types are approximately 36 feet long and are usually used to fire laser-guided bombs or air-to-surface missiles on targets. They have a range of over 1000 miles and can be used for up to 14 hours at a stretch.
6. **Non-Combat-Large-Drones**
   Although large, these drones are not for combat. They are more complex than Black Hornet and are used for larger-scaler recon missions.
7. **Target-and-Decoy-Drones**
   These types of drones are used for monitoring and striking targets. The look of the decoy drone usually depends on the mission.
8. **GPS-Drones**
   This drone type links to satellites via a GPS hook-up to map out the rest of their flight, collecting data that can be extracted to make informed decisions.
9. **Photography-Drones**
   Photography drones are outfitted with professional-grade cameras. 4K camera drones can take high-resolution pictures. These drone types make use of automated flight mode and precision stability to take pictures covering vast spaces.

Drones are a transformative technology. They have been and can be used in various areas such as.

Applications Of Drone Technology

- **Land mapping:** The drone technology in the SVAMITVA scheme has helped about half a million village residents to get their property cards by mapping out the areas.
- **Emergency response:** Drones are significant for the agencies such as the fire and emergency services wherever human intervention is not safe. It can perfectly save human efforts during disaster management.
- **Distant and remote delivery purposes:** Recently, the Ministry of Civil Aviation has approved a project with the Telangana government for using drone technology to deliver vaccines in remote areas.
- **Agriculture:** In the agriculture sector, micronutrients, and hazardous pesticides can be spread with the help of drones. It can also be used for performing surveys for identifying the challenges faced by the farmers.
- **E- Commerce:** Drones offer a perfect and cost-effective solution for delivery of products by e-com facilitators.
- **Monitoring:** The railways are using drones for track monitoring. Telecom companies are using drones for monitoring the tower.
- **Security and defence:** Drone system can be used as a symmetric weapon against terrorist attacks. They can be integrated into the national airspace system.

UAV elements:



Frame:

- It should have sufficient strength to hold the propeller momentum and additional weight for motors and cameras
- Sturdy and less aerodynamic resistance

Propellers:

- The speed and load lifting ability of a drone depends on shape, size, and number of propellors
- The long propellors create huge thrust to carry heavy loads at a low speed (RPM) and less sensitive to change the speed of rotation
- Short propellors carry fewer loads. They change rotation speeds quickly and require a high speed for more thrust.

Motor:

- Both motors brushless and brushed type can be used for drones
- A brushed motor is less expensive and useful for small-sized drone
- Brushless type motors are powerful and energy very efficient. But they need Electronic Speed Controller (ESC) to control their speed. These brushless motors

arewidely used for racing freestyle drones, traffic surveys and aerial photography drones.

ESC (Electronic Speed Controller):

- ESC is used to connect the battery to the electric motor for the power supply
- It converts the signal from the flight controller to the revolution per minted (RPM) of motor
- ESC is provided to each y motor of the drone

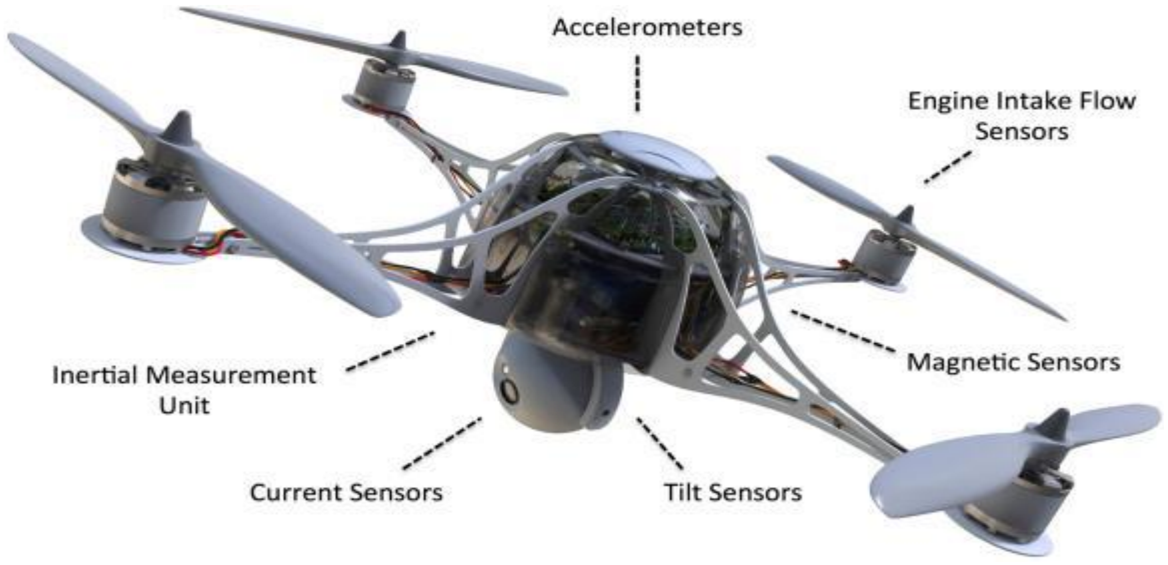
Flight Controller (FC)

- It is the computer processor which manages balance and telecommunication controls using different transmitter
- Sensors are located in this unit for the accelerometer, barometer, magnetometer, gyrometer and GPS
- The distance measurement can be carried out by an ultrasound sensor

Radio Transmitter sends the radio signal to ESC to pilot to control motor speed.
Radio Receiver: Received the signal from the pilot. This device is attached to the quadcopter
Battery: High-power capacity, Lithium Polymer (LiPo) is used for most drones. The battery can have 3S (3 cells) or 4S (4 cells).

- When the pilot or autonomous system gives the drone a command, the flight controller sends signals to the motors to spin the propellers
- The speed and direction of the motors and propellors are adjusted to achieve the desired movement. The sensors provide data to the flight controller, which uses it to stabilize the drone in the air and adjust its movement
- Drones can be controlled manually using a remote controller or programmed to fly autonomously. Autonomous drones use sensors and pre-programmed instructions to fly to a specific location, perform a task like taking photos or delivering a package, and return to their starting point.

Sensors -arms:



Accelerometers

Accelerometers are used to determine position and orientation of the drone in flight. Like your Nintendo Wii controller or your iPhone screen position, these small silicon-based sensors play a key role in maintaining flight control. MEMS accelerometers sense movement in several ways. One type of technology senses the micro movement of very small structures embedded in a small integrated circuit. The movement of these small 'diving boards' changes the amount of electrical current moving through the structure, indicating a change of position relative to gravity.

Another technology used in accelerometers is thermal sensing, which offers several distinct advantages. It does not have moving parts, but instead senses changes in the movement of gas molecules passing over a small integrated circuit. Because of the sensitivity of these sensors, they play a role in stabilizing on-board cameras that are vital for applications like filmmaking.

By controlling up and down movement, as well as removing jitter and vibration, filmmakers are able to capture extremely smooth looking video. Additionally, because these sensors are more immune to vibrations than other technologies, thermal MEMS sensors are perfect in drone applications to minimize problems from the increased vibration generated by the movement of rotating propulsion fans and propellers.

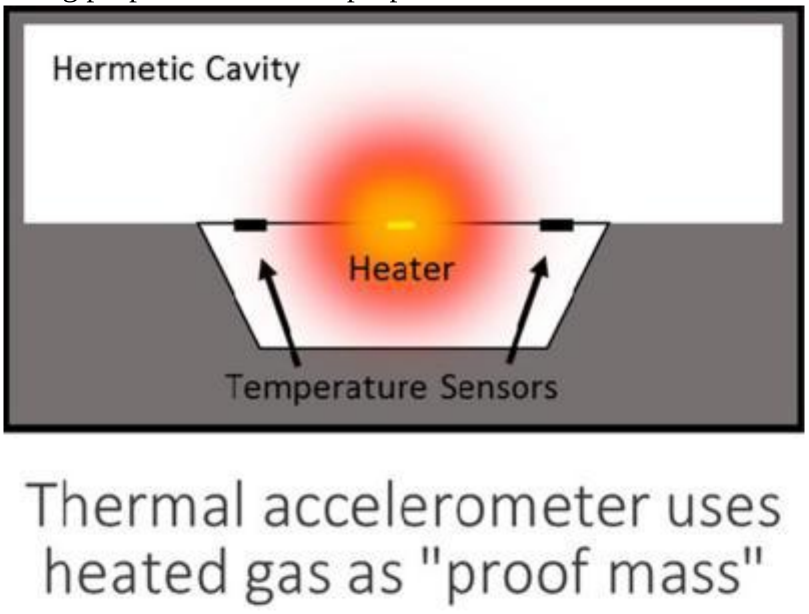


Fig: Because they have no moving parts, accelerometers based on thermal sensing technology offer much better stability and accuracy than mechanical based sensors.

Inertial Measurement Units

Inertial measurement units combined with GPS are critical for maintaining direction and flight paths. As drones become more autonomous, these are essential to maintain adherence to flight rules and air traffic control.

Inertial measurement units utilize multi-axis magnetometers that are in essence small, accurate compasses. These sense changes in direction and feed data into a central processor, which ultimately indicates direction, orientation, and speed.

Fig: The INS380SA module pictured here is a complete inertial navigation system with a built-in 48-channel GPS receiver.

Tilt Sensors

Tilt sensors, combined with gyros and accelerometers, provide input to the flight-control system in order to maintain level flight. This is extremely important for applications where stability is paramount, from surveillance to delivery of fragile goods.

These types of sensors combine accelerometers with gyroscopes, allowing the detection of small variations of movement. It is the gyroscope compensation that allows these tilt sensors to be used in moving applications like motor vehicles or drones.

Fig :Incorporating both an accelerometer and a gyroscope, Tilt sensors help drones maintain level flight.

Current Sensors

In drones, power consumption and use are important. Current sensors can be used to monitor and optimize power drain, safe charging of internal batteries, and detect fault conditions with motors or other areas of the system.

Current sensors work by measuring electrical current (bi-directional) and ideally provide electrical isolation to reduce power loss and eliminate the opportunity for electrical shock or damage to the user or systems. Sensors with fast response time and high accuracy optimize battery life and performance of drones.

Magnetic Sensors

In drones, electronic compasses provide critical directional information to inertial navigation and guidance systems. Anisotropic magneto resistive (AMR) permalloy technology sensors, which have superior accuracy and response time characteristics while consuming significantly less power than alternative technologies, are well-suited to drone applications. Turnkey solutions provide drone manufacturers with quality data sensing in a very rugged and compact package.

Engine Intake Flow Sensors

Flow sensors can be used to effectively monitor air flow into small gas engines used to power some drone varieties. These help the engine CPU determine the proper fuel-to-air ratio at a specified engine speed, which results in improved power and efficiency, and reduced emissions.

Many gas engine mass-flow sensors employ a calorimetric principal utilizing a heated element and at least one temperature sensor to quantify mass flow. MEMS thermal mass air flow sensors also utilize the calorimetric principal, but in a micro scale, making it highly suitable for applications where reduced weight is critical.

Motors and Electric Speed Controllers:

There are two types motors:

1. Brushed motor
2. Brushless motor

Brushed vs. brushless motor drones: What are the differences:

In the event that you have invested some effort and time looking for UAVs, you have, without a doubt, run over the expressions “brushless” or “brushed” engines. What is the distinction between the drones with these motors? Is it accurate to say that one is in a way that is better than the other?

Brushed motor drones:

The flying robots with brushed motors are often considered conventional and classic. These motors showed up in the market a very long while ago and have become the establishment of standard engine innovation.

Similar to other motors, a brushless one boasts of two significant components: a stator and a rotor. As the names infer, the stator is the fixed remaining part; meanwhile, the rotor is the turning part of an engine.

When it comes to brushed motors, a stator fixed with a couple of opposing polarity magnets encircles a rotor that is positioned centrally. The rotor’s rotational motions are actuated by the physical touching of its commutator’s brushes with a power supply. The power flow into the motor’s rotor generates a magnetic field responding with the stator’s magnets.

It is worth noting that such a magnetic interaction, combined with the power flow actuated by the brush contact with the power supply, brings about an eternal turn of the motor’s rotor. Expanding the power supply to the rotor has to do with how strong the magnetic field becomes, thus accelerating the rotor’s turn.

The way that the commutator's brushes have to contact the power supply is probably the greatest drawback of UAVs with brushed engine innovation. As you can envision, the commutator's fast turning creates a great deal of friction between its brushes and the power supply. Such friction's aftermath is twice as numerous: in addition to the fact that it produces a great deal of heat, it quickens the damage of the parts of a brushed engine in the course of use.

You may not like that the friction brought about by steady contact implies the life expectancy of your drone's brushed engines is restricted. Brushed ones will ask for a ton of upkeep and should be taken the place of significantly earlier when compared to the brushless ones. The brushes' steady contact with the power supply additionally makes the motors a lot louder in comparison with the brushless option.

That is not all; another downside that numerous users neglect when they put brushless and brushed motor drones in comparison is the brushed engine's inefficiency. In case you do not know, the heat created in the motor because of constant contact implies that a ton of the power given by the UAV battery is used for no purpose. As opposed to being transferred to rotational motions in the drone's propellers, the energy rather gets dispersed as heat.

Notwith standing the numerous drawbacks of these engines, they are intensely utilized owing to one significant explanation: the brushed motors are a lot less expensive. They are still generally significant, particularly for low-cost undertakings.

In case you miss it, in the UAV world, brushed engines tend to be found in inexpensive toy product units. All things considered; these flying robots are not intended to stay in the air for more than a couple of minutes all at once. Thanks to brushed engines, UAV producers have the choice to make the costs of their toy UAVs remain reasonable**.**

Brushless motor drones:

Brushless engines were first evolved about 60 years ago as a more proficient option in contrast to brushed engines. The construction changes made for these motors were pretty noteworthy. Rather than a rotor positioned centrally, brushless ones come with a centrally located stator encircled by a rotor fixed with permanent magnets.

The brushless engine's stator is equipped for producing its own magnetic field since it is constructed out of a few sets of power coils. By providing a power flow to such coils, the stator creates a magnetic field with the interaction with the encompassing rotor's magnets and leads to its turn. Through directing the power flow that the power coils of the stator get, the pace of the pivot of your drone's brushless engine can be controlled.

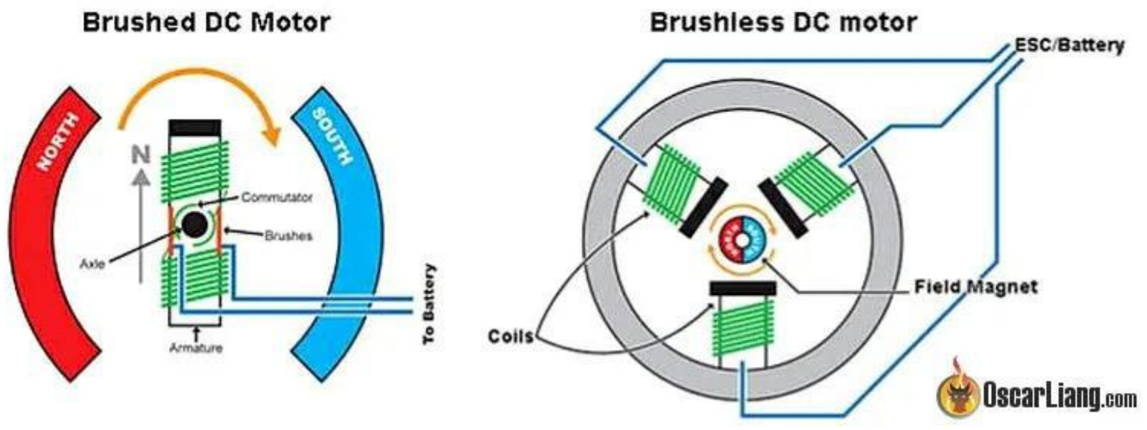
A crucial attribute of brushless choices is the way that there is not any contact between pivoting and fixed parts. Because there is no friction, there will not be any energy lost in the form of heat, which hence makes the entire cycle significantly more effective. For your information, brushed engines are able to do just about eighty percent of efficiency in the best case scenario; meanwhile, brushless motors can hit up to ninety percent of efficiency.

With no heat production, your drone's brushless engines will be able to work for a long while with no threat of overheating. That is especially significant for professional-level UAVs which are intended to remain in the air ceaselessly for more than half an hour.

The absence of contact between the brushless motor's moving parts additionally implies that damage in the course of úe is significantly decreased. Truth be told, the brushless engines of a flying robot can be utilized for quite a long while with no maintenance. Brushless options are additionally less noisy when compared to brushed UAVs, which thus absolutely wipes out the trademark shrieking sound generated by the continuous contact between the brushes of the commutator and the current supply.

These days, just about any prosumer UAVs and UAVs designed for open-air flight are intended to utilize brushless engines. They are not just more dependable and effective but more lightweight and less sizable.

Their efficiency additionally implies that your flying robots can stay in the air for an extended time with a similar battery limit. For models with a premium in weight and size, brushless engines have been a significant bit of innovation.



Electric Speed Controllers:

The electric speed controllers (ESCs) take in thecommands from the flight controller and adjust thespeed of the individual motors accordingly. The ESCscontrol how fast the motors spin as well as increaseand decrease the speed of specific motors to turn andbank in flight. For our project, the flight navigationwill be automated instead of using a physicalcontroller. The motor with 6 coils is activated usingtwo mosfets to create a rotating magnetic field thatturns the brushless motors. The motor speed can beincreased by cycling through these intervals at ahigher frequency.The ESC has to know when to activate the next phasebased on the position of the motor. The position canbe found using a Hall effect sensor to sense themagnetic field in the motor. By carefully controllingthe speed that the rotor turns, the 4 propellers arerotated allowing the drone to lift off, fly forward, turn,and land. In our project, the AR 2.0 drone willcomplete 28,000 revolutions per minute whilehovering and require 14.5 Watts with a speed of 11meters/second.

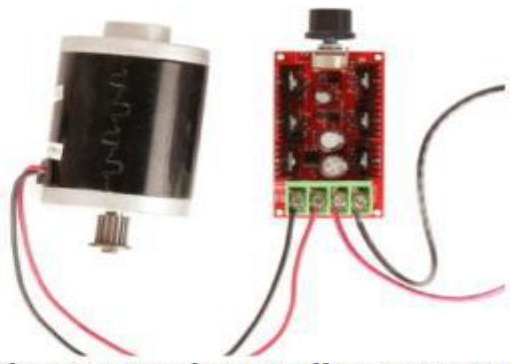


Figure 2. Electric Speed Controller connected to Motor

.

GPS (Global Positioning System):

GPS/GNSS used in Unmanned Aerial Vehicles (UAV) are increasingly being used for a wide range of applications including reconnaissance, surveillance, surveying and mapping, spatial information acquisition and geophysics exploration, among others. Often, in these situations, GPS is the key to operating the UAV safely.

Whether the vehicles are guided autonomously, or guided by ground-based pilots, GPS in UAV plays an important role. As long as sufficient satellite signals can be accessed during the entire UAV mission, GPS navigation techniques can offer consistent accuracy. Often, GPS is used in conjunction with Inertial Navigation Systems (INS), to offer more comprehensive UAV navigation solutions.

The most common use of GPS in UAV is navigation. A central component of most navigation systems on a UAV, GPS is used to determine the position of the vehicle. The relative positioning and speed of the vehicle are also usually determined by the UAV GPS. The position provided by the receiver can be used to track the UAV, or, in combination with an automated guidance system, steer the UAV.

Autonomous UAV usually rely on a GPS position signal which, combined with inertial measurement unit (IMU) data, provides highly precise information that can be implemented for control purposes. In order to avoid accidents in an area heavily populated by other UAV or manned vehicles, it is necessary to know exactly where the UAV is located at all times. Equipped with GPS, a UAV can not only provide location and altitude information, but necessary vertical and horizontal protection levels.

As stated above, UAV are often used for earth observation measurements, making use of cameras and radars installed for this very purpose. In order to accurately geographically reference collected data, it is important to know the exact position of the vehicle when a measurement or photo was taken. A UAV GPS receiver can pinpoint the exact position of the UAV, often down to the centimetre. The same concept applies to the exact time at which the photo or measurement was taken. The precise time stamps provided by UAV GPS are invaluable in collecting this information

GPS MODULE

IMU (Inertial Measurement Unit):

An Inertial Measurement Unit (IMU) is an electronic device that uses accelerometers and gyroscopes to measure acceleration and rotation, which can be used to provide position data.IMUs are essential components in unmanned aerial systems (UAVs, UAS and drones) – common applications include control and stabilization, guidance and correction, measurement and testing, and mobile mapping.

The raw measurements output by an IMU (angular rates, linear accelerations and magnetic field strengths) or AHRS (roll, pitch and yaw) can be fed into devices such as Inertial Navigation Systems (INS), which calculate relative position, orientation and velocity to aid navigation and control of UAVs.

IMUs are manufactured with a wide range of features, parameters, and specifications, so the most suitable choice will depend on the requirements for a particular UAV application. This article outlines some of the key options and considerations, such as the underlying technology, performance, and ruggedness, in selecting an IMU for drone-based applications. It also highlights some of the leading IMU manufacturers for UAS (jumps to this section).

Inertial Measurement Unit Technologies:

There are many types of IMU, some of which incorporate magnetometers to measure magnetic field strength, but the four main technological categories for UAV applications are: Silicon MEMS (Micro-Electro-Mechanical Systems), Quartz MEMS, FOG (Fibber Optic Gyro), and RLG (Ring Laser Gyro).

Silicon MEMS IMUs are based around miniaturized sensors that measure either the deflection of a mass due to movement, or the force required to hold a mass in place. They typically perform with higher noise, vibration sensitivity and instability parameters than FOG IMUs, but MEMS-based IMUs are becoming more precise as the technology continues to be developed.

Advanced Navigation's Motus MEMS IMU

MEMS IMUs are ideal for smaller UAV platforms and high-volume production units, as they can generally be manufactured with much smaller size and weight, and at lower cost.

FOG IMUs use a solid-state technology based on beams of light propagating through a coiled optical fibre. They are less sensitive to shock and vibration, and offer excellent thermal stability, but are susceptible to magnetic interference. They also provide high performance in important parameters such as angle random walk, bias offset error, and bias instability, making them ideal for mission-critical UAV applications such as extremely precise navigation.

EMCORE's EN-150 FOG IMU

Higher bandwidth also makes FOG IMUs suitable for high-speed platform stabilization. Typically, larger and more costly than MEMS-based IMUs, they are often used in larger UAV platforms.

RLG IMUs utilise a similar technological principle to FOG IMUs but with a sealed ring cavity in place of an optical fiber. They are generally considered to be the most accurate option, but are also the most expensive of the IMU technologies and typically much larger than the alternative technologies.

Quartz MEMS IMUs use a one-piece inertial sensing element, micro-machined from quartz, that is driven by an oscillator to vibrate at a precise amplitude. The vibrating quartz can then be used to sense angular rate, producing a signal that can be amplified and converted into a DC signal proportional to the rate. Quartz MEMS technology features high reliability and stability over temperature, and tactical-grade quartz MEMS IMUs rival FOG and RLG technologies for SWaP-C (size, weight, power and cost) metrics. These factors make it ideal for inertial systems designed for the space- and power-constrained environments of UAVs.

The Systron Donner Inertial (an Emcore brand) SDI500 Quartz MEMS IMU

IMU Performance and Accuracy:

The performance and accuracy of an IMU are influenced by a combination of factors, including the sensor technology, the thermal properties of the packaging, and the software used. The following parameters can be used when comparing the performance and accuracy of specific IMUs, to help determine suitability for a given UAV application:

- Bias – what does the IMU output read when the input is zero?
- Bias repeatability – how similar is the IMU bias when conditions have changed between measurements (e.g. for each powerup of the IMU)?
- Bias stability – how much does the bias change over time?
- Random Walk – how much random noise is present?
- Vibration Sensitivity – how much does the output of the angular rate change per unit of vibration present in the environment?

These factors are dependent on the technologies used in the IMU and the physical properties of the accelerometers, gyroscopes and magnetometers. If an IMU is manufactured with temperature compensation, this will improve the stability of the measurements.

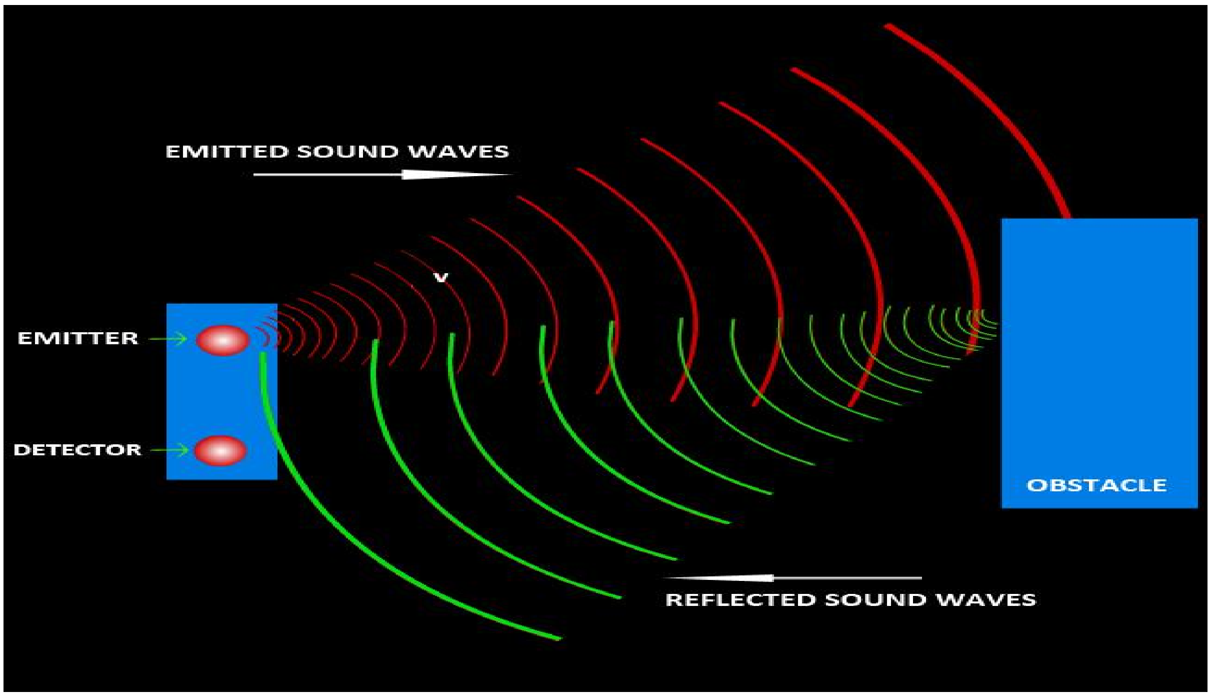
For high-accuracy applications such as UAV surveying and mapping, a high data output rate is also important as this will reduce errors due to interpolation between readings.

Ultrasonic Sensor:

An ultrasonic sensor is an electronic device that measures the distance of a target object by emitting ultrasonic sound waves, and converts the reflected sound into an electrical signal. Ultrasonic waves travel faster than the speed of audible sound (i.e. the sound that humans can hear). Ultrasonic sensors have two main components: the transmitter (which emits the sound using piezoelectric crystals) and the receiver (which encounters the sound after it has travelled to and from the target).

In order to calculate the distance between the sensor and the object, the sensor measures the time it takes between the emission of the sound by the transmitter to its contact with the receiver. The formula for this calculation is $D = \frac{1}{2} T \times C$ (where D is the distance, T is the time, and C is the speed of sound ~ 343 meters/second). For example, if a scientist set up an ultrasonic sensor aimed at a box and it took 0.025 seconds for the sound to bounce back, the distance between the ultrasonic sensor and the box would be:

D = 0.5 x 0.025 x 343



Ultrasonic sensors are used primarily as proximity sensors. They can be found in automobile self-parking technology and anti-collision safety systems. Ultrasonic sensors are also used in robotic obstacle detection systems, as well as manufacturing technology. In comparison to infrared (IR) sensors in proximity sensing applications, ultrasonic sensors are not as susceptible to interference of smoke, gas, and other airborne particles (though the physical components are still affected by variables such as heat).

Ultrasonic sensors are also used as level sensors to detect, monitor, and regulate liquid levels in closed containers (such as vats in chemical factories). Most notably, ultrasonic technology has enabled the medical industry to produce images of internal organs, identify tumours', and ensure the health of babies in the womb.

UAV Software -Arudpilot:

UAV Navigation develops software in accordance with a strict validation process to ensure the final product delivered to the customer is as reliable as possible. The company's mission-critical software products fall into the following categories:

- User Interface (UI) software e.g.Visionair.
- Embedded autopilot software (RTOS, device drivers, and low-level software).
- Flight Control software.

Steps:

1. **Specification:** The final software product (or modification to an existing product) is clearly specified. Requirements are analyzed and agreed upon between client and UAV Navigation work teams in order to ensure the final product reaches its expectations.

2. **Development*:** UAV Navigation ensures to provide the specific forms that the development engineers will need during its work. Only when this specific and strict training time has been completed before the engineer starts developing products. This way, we ensure the high quality of our products. Software is developed by the appropriate department as below:
    - UI software: Software Department.
    - Embedded software: Software Department (in consultation with Hardware Department).
    - Flight Control software: Flight Control Department.

   Software is developed in accordance with a proprietary coding standard based on industry standards, such as MISRA, JPL (NASA), Embedded C Coding Standard (Barr Group), and Google C++ Style. Part of this development process includes peer-checking.
3. **Testing & Simulation*:** All software undergoes an intensive period of testing in the office, including at least 12 hours of simulated flight missions. All flight modes are tested as well as interaction with payloads and emergency handling (comms failure, etc). If necessary, the development team will return to Step 2 to modify features.
4. **Acceptance Test Procedure (ATP):** Phase 1 - Bench Testing*: Once Step 3 is completed and there is no need to iterate through Step 2 again, a formal ATP is carried out on the test bench, mainly based on the use of HIL tools. Every component and control solution (FW, RW, Target, AHRS, etc..) has its own written ATP documents for regression testing and new feature quality control, which are under constant improvement as part of the quality process.
5. **ATP: Phase 2 - Flight Testing**: Once the software has successfully passed the company's formal ATP on the simulation bench, the software can proceed to actual flights on the company's appropriate test platform(s), fully testing all flight modes and features. The platforms used currently include:

   Fixed-wing:

   Electric, 2.5kg MTOW.

   Single piston engined, 9kg MTOW.

   Rotary wing:

   Single piston engined, 8kg MTOW.
6. **Software Release:** The software developed is uploaded to the UAV Navigation Download Center, a restricted area where the official versions of UAV Navigation software are made available to all clients. Additionally, at UAV Navigation, we provide the client with a Confidential space dedicated to each particular client. No other clients have access to this area. It is used to supply specific software and documentation.
7. **Confirmatory Testing on Customer Platform:** The final step is for the software to be validated on the actual customer’s platform. UAV Navigation is available to participate in this vital stage by deploying an engineering team on-site and an engineer to the test location if necessary. UAV Navigation agrees with the customer a FAT (Flight Acceptance Test) document describing all the flight tests required to accept the completion of the work.

Mission Planning:

UAV mission planning software for geophysical surveys. With completely customizable survey parameters, you can plan your magnetic UAV survey at the elevation and line spacing that you want. UgCS supports terrain following and more, create and store multiple missions and routes, and choose the heading of the sensor.

Take the guesswork out of UAV operations for geophysical surveys with UgCS(Licenses) Mission Planning Software.

- **Automated Drone Mission Planning:** Plan your UAV flight missions the way you want! UgCS is the most advanced UAV flight software available.
- **Custom Flight Lines:** For geophysical surveys, flight and tie lines can be customized to your survey needs and to ensure maximum sensor platfom stability.
- **Photogrammetry Tools:** Increase the data capture productivity at least 2 times with the inbuilt automatic photogrammetry planning tool
- **Geotagging Tools:** Tag landmarks and other features of interest in the software.
- **Digital Elevation Model (DEM) and KML file import:** Use pre-installed or import more precise DEM data to increase accuracy and safety for missions with terrain following
- **Map Customization:** Adjust the allowed flight range and No-Fly Zones to fly according to regulatory requirements
- **Battery Changes:** Take into account your battery hotswaps while planning your survey.
- **Offline Map Caching:** Plan and fly missions without an internet connection.

Internet Of Drones (IOD):

The internet of drones (IoD) combines drones and the internet to empower users in multiple ways. Basically, it means IoT sensors are starting to populate low-altitude airspace. In that sense, IoD is simply IoT in the sky. Drones make it possible for sensor omnipresence to blanket the planet's atmosphere, creating a highly interconnected global village.

Applications of Drones in Various Industries:

Commercial drones are now used by retailers to deliver products faster to consumers. At the start of the 2020s, the fastest drones were able to exceed 160 miles per hour but the legal limit set by the Federal Aviation Administration (FAA) is 100 mph. The average drone can travel at 45 mph. Here are ways drones can improve different industries:

- **Smart agriculture** - Smart drones are perfect tools for farmers to get aerial views of their crops to monitor growth. Using GPS, they can detect specific areas that need more attention. They can take measurements important to agriculture, such as temperature, humidity, sunlight and wind. Farmers are concerned about conserving and maximizing natural resources, but with drones, they will be able to identify and reduce waste. Drones can also be used for agricultural land mapping and spraying agricultural chemicals on crops.
- **Mining** - One of the best ways drones can help the mining industry is by improving the safety of mining operations. Mining is a dangerous job due to the explosives involved. Drone cameras can help map out safety zones for crew members. Since

mines typically take up vast amounts of land, drones can help monitor site conditions to avoid manual inspections that require labor, additional costs, and much more time.

- **Construction** - A contractor can use a drone to get aerial photos of construction projects to help make the site safer and more efficient. Drones can help crew members detect installation vulnerabilities and complications involved with building layout and roofing. As with mining and other dangerous works, the construction industry can improve with drones facilitating group coordination. Drones can also help locate and limit construction waste.
- **Emergency and Delivery Services** - The odds of saving lives in emergency situations increase with drones. They can help find a lost or missing child and victims in remote locations. In order for drones to perform this function, they require high-quality communication capabilities and enhanced sky connectivity. Drones can transport disaster relief in the form of food and medicine to victims. It's actually possible for drones to help rescue victims underwater. They certainly have many more public safety uses following an earthquake, hurricane or tornado.
- **Films and TV** - Drones are taking cinematography to new heights. They make shooting films and videos from the sky much easier and safer than using a helicopter or a crane. For film producers trying to cut labor costs, smart drones provide precision positioning and moving the camera to get desired elements within the picture frame. Drones have already been used for TV news programs, especially in live pursuits of police chasing suspects. Ultimately, drones allow for automated filmmaking integrated with machine learning software to get the best camera angles and trajectories.

Mechanical Design Requirements

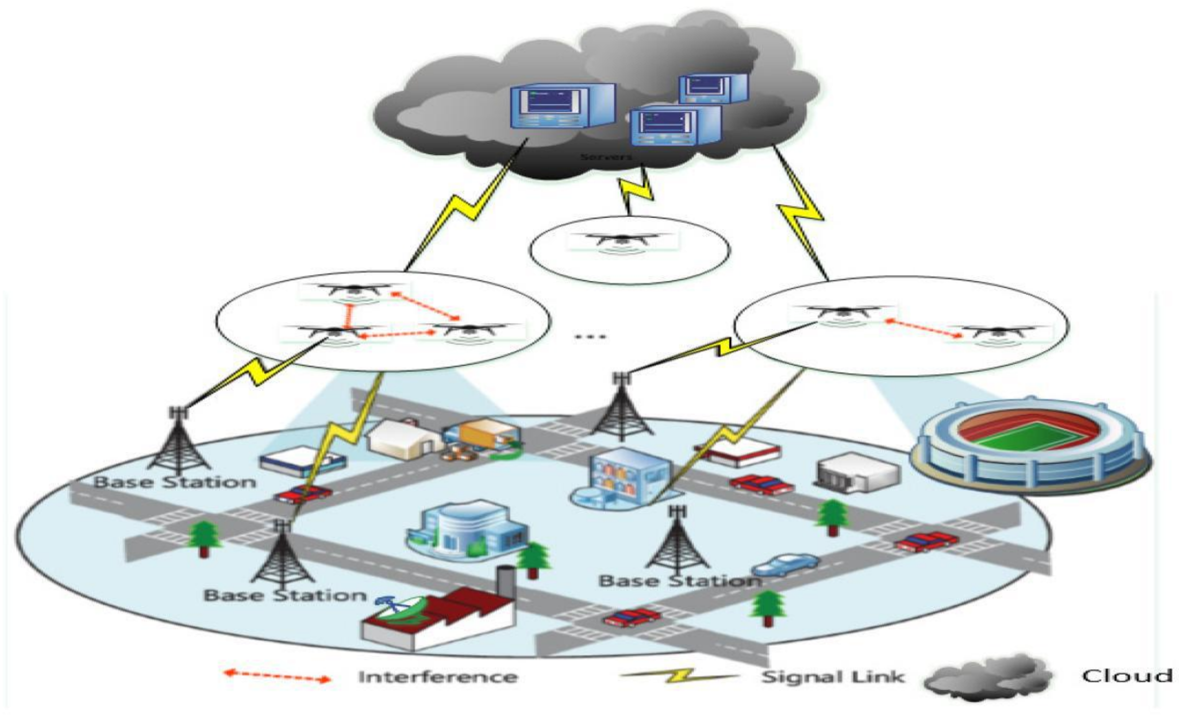
- **Drone parts** - The drone's body connects with one, two, three or four propellers for lifting and vertical motion. Drones are typically powered by a motor that draws energy from an intelligent lithium-polymer (LiPo) battery. Drones can also be powered by solar cells, hydro fuel cells and laser beams. The legs of a drone are typically used for antennas, the compass, GPS, and other sensors are embedded in the body, while the camera is typically mounted on a camera platform attached to the body.
- **Remote controller** - Users control the drone with a remote controller that includes joysticks for direction, similar to a video game. The remote controller may operate from a centralized base station.
- **Flight controller** - This tiny robot serves as an automated pilot that helps the drone achieve stability in difficult situations. The flight controller gets various signals from sensors.
- **MEMS sensors** - Micro-electro-mechanical (MEMS) sensors improve flight performance and allow the drone to be controlled accurately by users. The Inertial Measurement Unit (IMU) is the main sensor, as it measures acceleration and rotation of the drone. A drone sensor is typically the size of an ant and can be used to measure barometric pressure and other metrics.
- **Accessories** - Drones can be equipped with cameras for aerial views, integrated with AI and automation technology. A GPS module allows for satellite communication and determining geolocations.

IoD Infrastructure Requirements

In order for IoD to be effective on a mass scale for businesses and personal use, its infrastructure must be omnipresent, secure and flexible. At the core of the system should be smart technology, in which real-time data can be made available on demand. Ideally, the infrastructure allows for easy integration with new technology.

Cybersecurity should be a top priority for drone owners, as special authentication and key exchange protocols must generate a symmetric security key. Yes, drones can be hacked, much like any form of electronic communication. Remote hijacking with malware is even worse, so developing strong cybersecurity layers cannot be understated or overlooked.

Another IoD requirement is seamless coverage across suburban, urban and rural areas. At the moment, a good percentage of the earth is still not connected to the internet. But any area with at least 4G+ connectivity is sufficient for drone-to-base data sharing. As far as vertical coverage, drones get as high as 30,000 feet, but typically fly between 200 and 400 feet above the ground.



Internet of drones