Unit 1:

Assignment Topics:

1. Define Internet of things and characteristics of IOT and protocols used in Link Layer

A. Internet of Things (loT) comprises things that have unique identities and are connected to the Internet.

The characteristics of IoT
1.   Self-Configuring
2.   Dynamic and Self-Adapting
3.   Interoperable Communication Protocols
4.   Unique Identify

Integrated into information Network

IoT protocols used in Link Layer are
1.802.3-Ethernet
2. 802.11-WiFi
3. 802.16-WiMax
  4. 802.15.4-LR_WPAN 2G/3G/4G

 2.   Various enabling technologies and applications of IOT

A. IoT is enabled by several technologies including Wireless Sensor Networks, Cloud Computing, Big Data Analytics, Embedded Systems, Security Protocols and architectures, Communication Protocols, Web Services, Mobile internet and semantic search engines.

1)   Wireless Sensor Network(WSN): Comprises of distributed devices with sensors which are used to monitor the environmental and physical conditions. Zig Bee is one of the most popular wireless technologies used by WSNs.

WSNs used in IoT systems are described as follows:

- Weather Monitoring System: in which nodes collect temp, humidity and other data, which is aggregated and analyzed.
- Indoor air quality monitoring systems: to collect data on the indoor air quality and concentration of various gases.
- Soil Moisture Monitoring Systems: to monitor soil moisture at various locations.
- Surveillance Systems: use WSNs for collecting surveillance data(motion data detection).
- Smart Grids : use WSNs for monitoring grids at various points.

2) **Cloud Computing:** Services are offered to users in different forms.
- Infrastructure-as-a-service(IaaS):provides users the ability to provision computing and storage resources. These resources are provided to the users as a virtual machine instances and virtual storage.
- Platform-as-a-Service(PaaS): provides users the ability to develop and deploy application in cloud using the development tools, APIs, software libraries and services provided by the cloud service provider.
- Software-as-a-Service(SaaS): provides the user a complete software application or the user interface to the application itself.

3) **Big Data Analytics:** Some examples of big data generated by IoT are
- Sensor data generated by IoT systems.
- Machine sensor data collected from sensors established in industrial and energy systems.
- Health and fitness data generated IoT devices.
- Data generated by IoT systems for location and tracking vehicles.
- Data generated by retail inventory monitoring systems.

4) **Communication Protocols:** form the back-bone of IoT systems and enable network connectivity and coupling to applications.
- Allow devices to exchange data over network.
- Define the exchange formats, data encoding addressing schemes for device and routing of packets from source to destination.
- It includes sequence control, flow control and retransmission of lost packets.

5) **Embedded Systems:** is a computer system that has computer hardware and software embedded to perform specific tasks. Embedded System range from low cost miniaturized

devices such as digital watches to devices such as digital cameras, POS terminals, vending machines, appliances etc.,

3. **Communication APIs of IOT**

A. **IoT Communication APIs:**
   a) **REST based communication APIs(Request-Response Based Model)**
   b) **WebSocket based Communication APIs(Exclusive Pair BasedModel)**
   a) **REST based communication APIs:** Representational State Transfer(REST) is a set of architectural principles by which we can design web services and web APIs that focus on a system's resources and have resource states are addressed and transferred.

The REST architectural constraints: Fig. shows communication between client server with REST APIs.
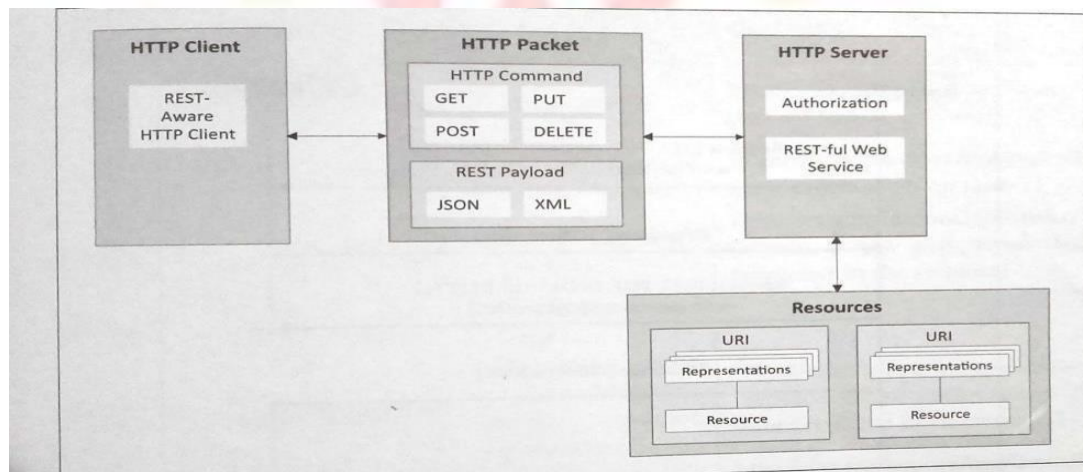
Client-Server:
The principle behind client-server constraint is the separation of concerns. Separation allows client and server to be independently developed and updated.
Stateless:
Each request from client to server must contain all the info. Necessary to understand the request, and cannot take advantage of any stored context on the server.
Cache-able:
Cache constraint requires that the data within a response to a request be implicitly or explicitly labeled as cache-able or non-cacheable. If a response is cache-able, then a client cache is given the right to reuse that response data for later, equivalent requests.
**Layered System:**
constraints the behavior of components such that each component cannot see beyond the immediate layer with which they are interacting.

User Interface:
constraint requires that the method of communication between a client and a server must be uniform.

Code on Demand:
Servers can provide executable code or scripts for clients to execute in their context. This constraint is the only one that is optional.

Request-Response model used by REST:
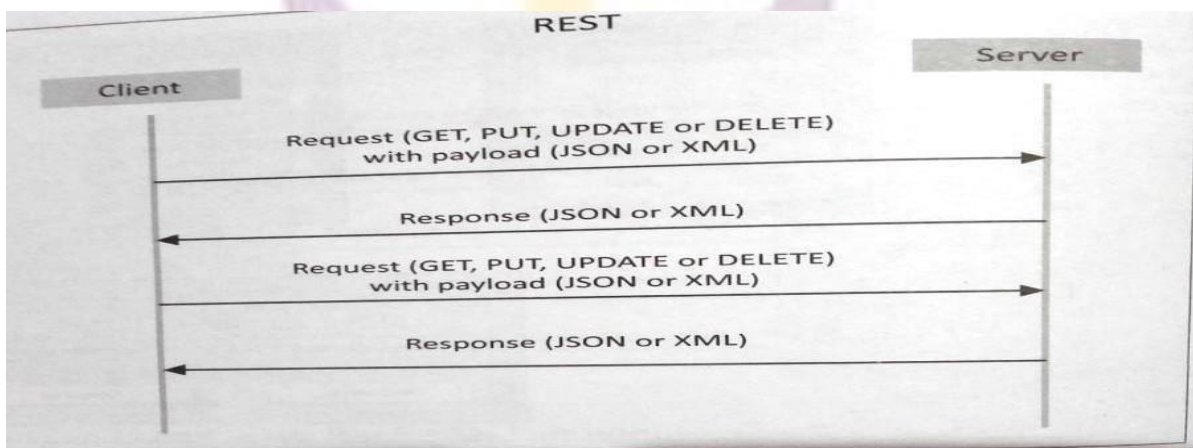
RESTful webservice is a collection of resources which are represented by URIs. RESTful web API has a base URI(e.g: http://example.com/api/tasks/). The clients and requests to these URIs using the methods defined by the HTTP protocol(e.g: GET, PUT, POST or DELETE). A RESTful web service can support various internet media types.

b) WebSocket Based Communication APIs: WebSocket APIs allow bi-directional, full duplex communication between clients and servers. WebSocket APIs follow the exclusive pair communication model.
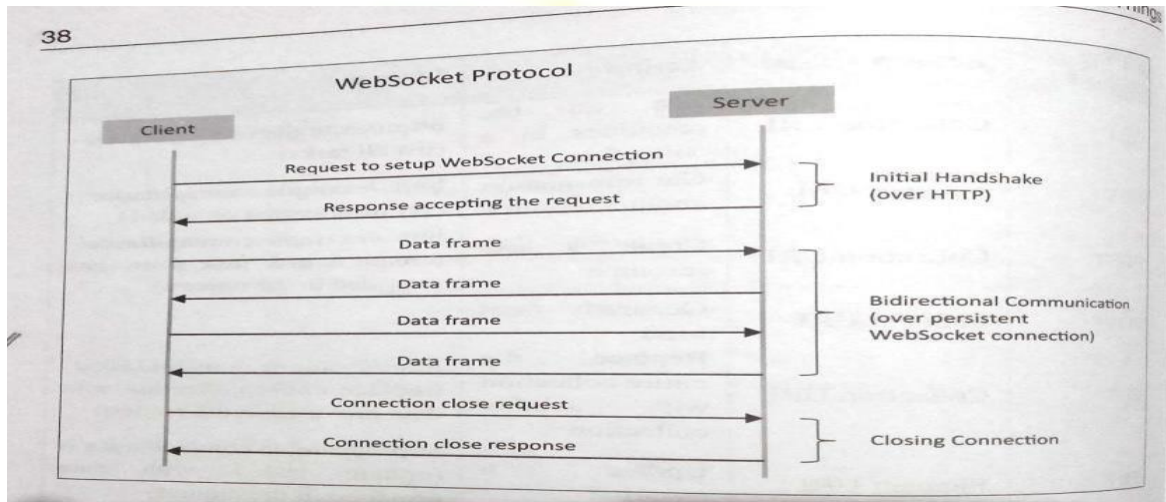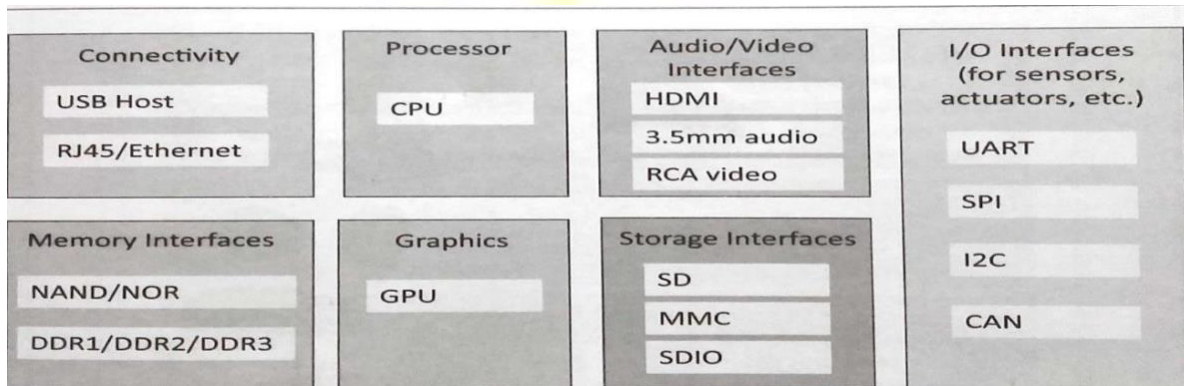
4. Physical design of IOT with neat diagram

A. Physical Design Of IoT

1) Things in IoT:

Connectivity	Processor	Audio/Video Interfaces	I/O Interfaces (for sensors, actuators, etc.)
USB Host	CPU	HDMI	UART
RJ45/Ethernet		3.5mm audio	SPI
		RCA video	I2C
Memory Interfaces	Graphics	Storage Interfaces	
NAND/NOR	GPU	SD	
DDR1/DDR2/DDR3		MMC	CAN
		SDIO	

The things in IoT refers to IoT devices which have unique identities and perform remote sensing, actuating and monitoring capabilities. IoT devices can exchange dat with other connected devices applications. It collects data from other devices and process data either locally or remotely.

An IoT device may consist of several interfaces for communication to other devices both wired and wireless. These includes

1. I/O interfaces for sensors,
2. Interfaces for internet connectivity
3. memory and storage interfaces
4. audio/video interfaces.

1) IoT Protocols:

a) Link Layer : Protocols determine how data is physically sent over the network's physical layer or medium. Local network connect to which host is attached. Hosts 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 h/w device over the medium to which the host is attached.

5. IOT levels with neat diagram Ans:

A. IoT Level 1: System has a single node that performs sensing and/or actuation, stores data, performs analysis and host the application as shown in fig. Suitable for modeling low cost and low complexity solutions where the data involved is not big and analysis requirement are not computationally intensive. An e.g., of IoT Level1 is Home automation.
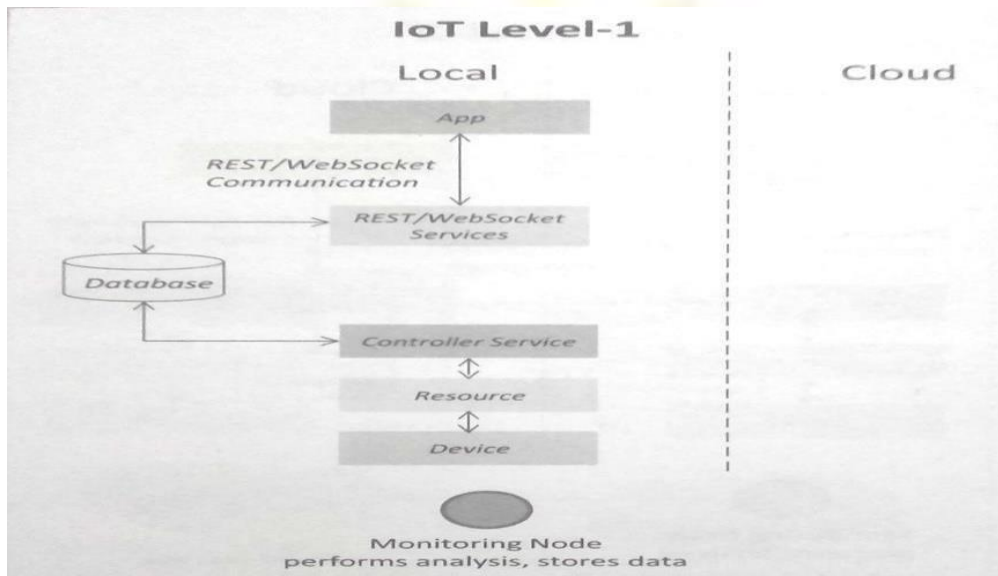

IoT Level2: has a single node that performs sensing and/or actuating and local analysis as shown in fig. Data is stored in cloud and application is usually cloud based. Level2 IoT systems are suitable for solutions where data are involved is big, however, the primary analysis requirement is not computationally intensive and can be done locally itself. An e,g., of Level2 IoT system for Smart Irrigation.
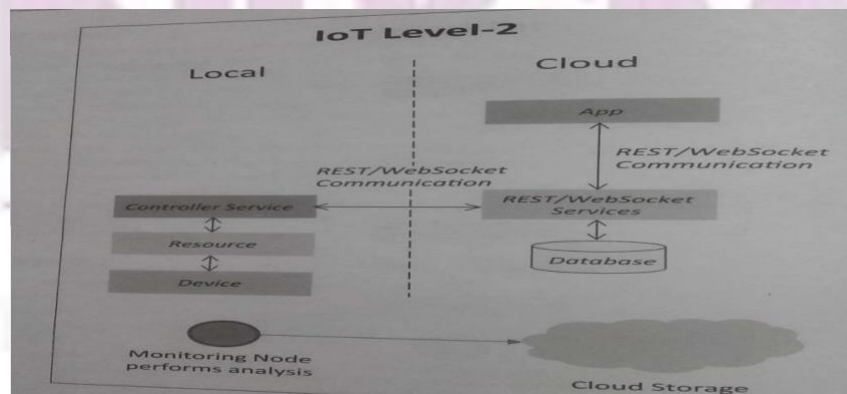
IoT Level3: system has a single node. Data is stored and analyzed in the cloud application is cloud based as shown in fig. Level3 IoT systems are suitable for solutions where the data involved is big and analysis requirements are computationally intensive. An example of IoT level3 system for tracking package handling.
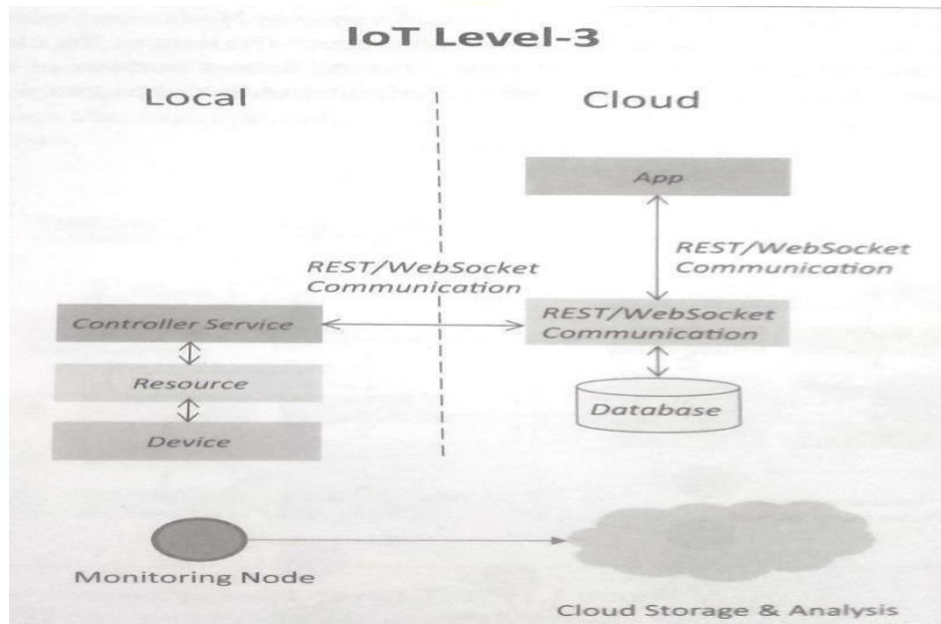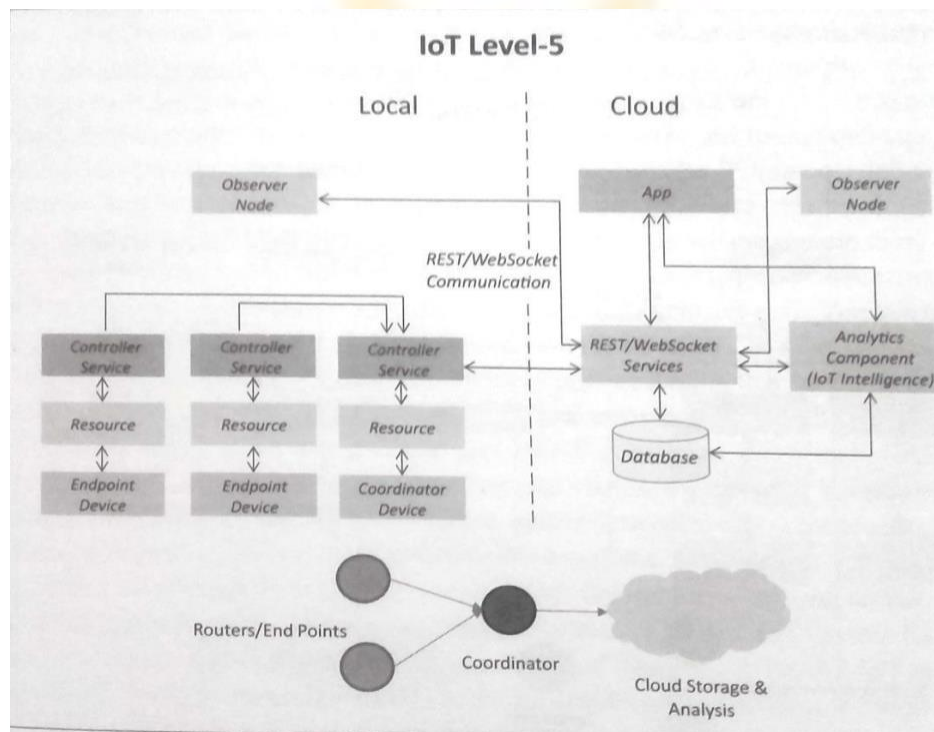

IoT Level4: System has multiple nodes that perform local analysis. Data is stored in the cloud and application is cloud based as shown in fig. Level4 contains local and cloud based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices. An example of a Level4 IoT system for NoiseMonitoring.
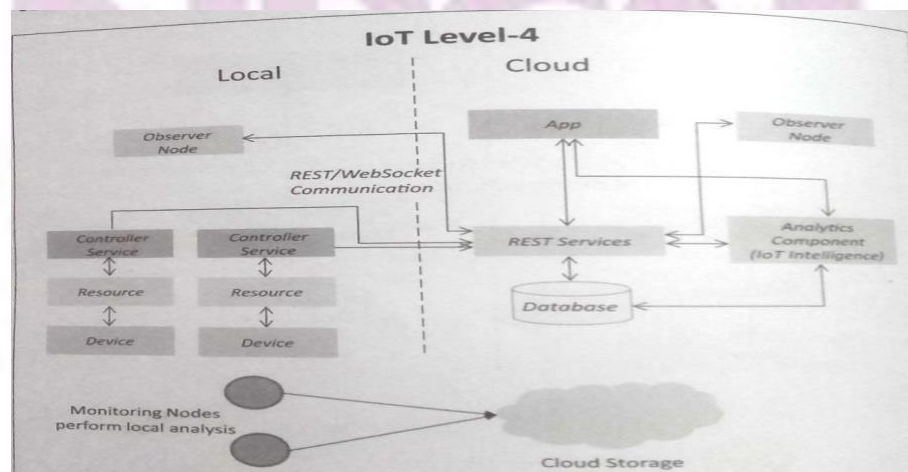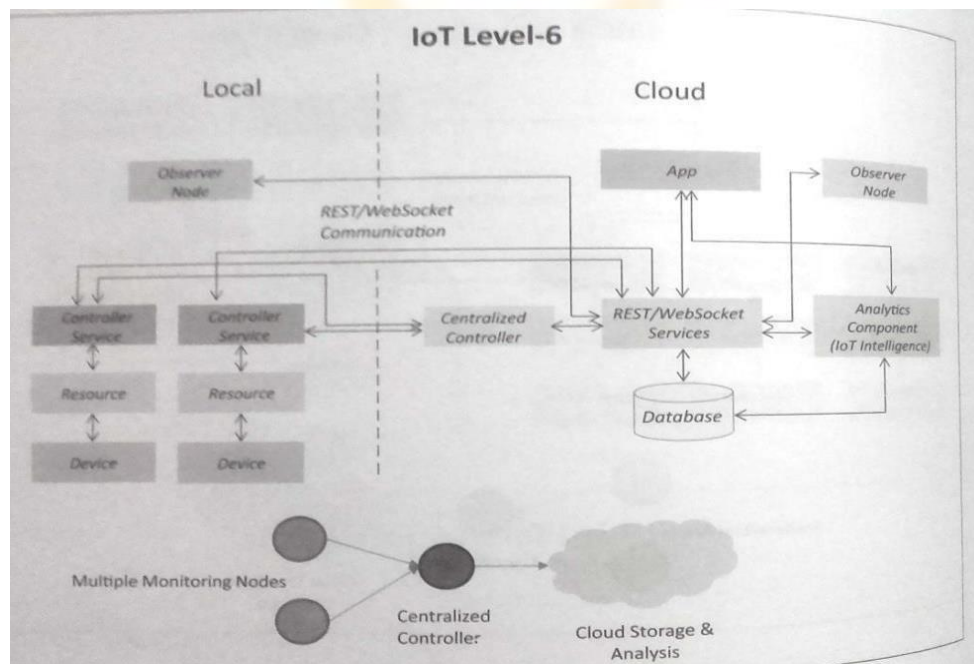
IoT Level5: System has multiple end nodes and one coordinator node as shown in fig.The end nodes that perform sensing and/or actuation. Coordinator node collects data from the end nodes and sends to the cloud. Data is stored and analyzed in the cloud and application is cloud based. Level5 IoT systems are suitable for solution based on wireless sensor network, in which data involved is big and analysis requirements are computationally intensive. An example of Level5 system for Forest Fire Detection.

IoT Level6: System has multiple independent end nodes that perform sensing and/or actuation and sensed data to the cloud. Data is stored in the cloud and application is cloud based as shown in fig. The analytics component analyses 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 sends control commands to nodes. An example of a Level6 IoT system for Weather Monitoring System.

UNIT -2

1. **M2M Architecture**

   Machine-to-Machine (M2M) refers to networking of machines(or devices) for the purpose of remote monitoring and control and data exchange.

   - Term which is often synonymous with IoT is Machine-to-Machine (M2M) .
   - IoT and M2M are often used interchangeably.

   Fig. Shows the end-to-end architecture of M2M systems comprises of M2M area networks, communication networks and application fomain.

   - An M2M area network comprises of machines(or M2M nodes) whiach have embedded network modules for sensing, actuation and communicating various communiction protocols can be used for M2M LAN such as ZigBee, Bluetooth, M-bus, Wireless M-Bus etc., These protocols provide connectivity between M2M nodes within an M2M area network.
   - The communication network provides connectivity to remote M2M area networks. The communication network provides connectivity to remote M2M area network. The communication networkcan use either wired or wireless network(IP based). While the M2M are networks use either properietorary or non-IP baed communication protocols, the communication network uses IP-based network. Since non-IP based protocols are used within M2M area network, the M2M nodes within one network cannot communicate with nodes in an external network.
   - To enable the communication between remote M2M are network, M2M gateways are used.
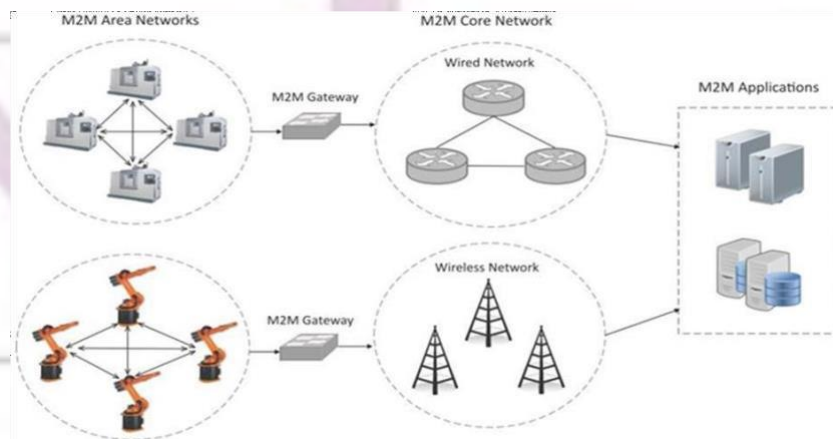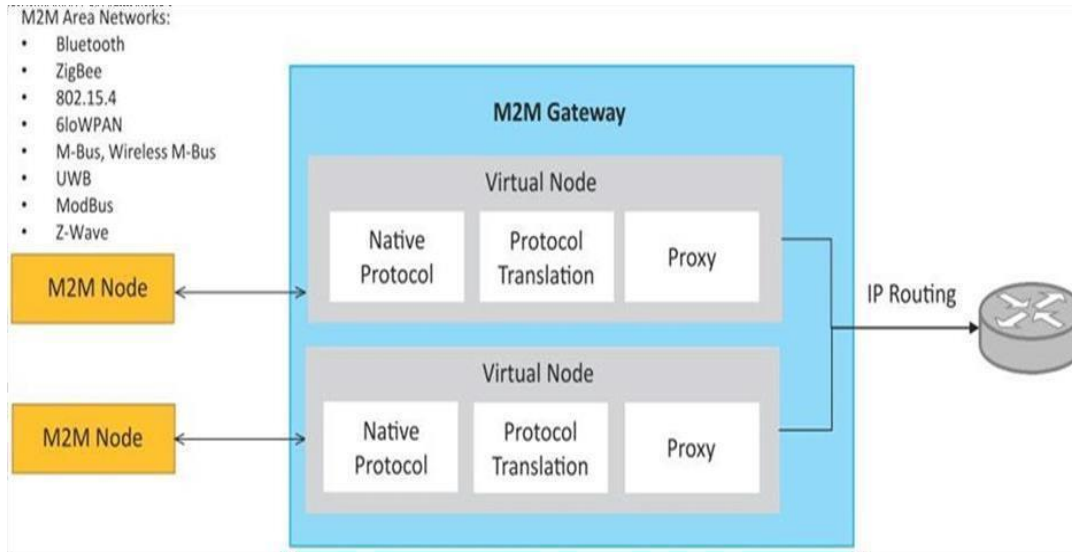
Fig. Shows a block diagram of an M2M gateway.

The communication between M2M nodes and the M2M gateway is based on the communication protocols which are naive to the M2M are network. M2M gateway performs protocol translations to enable Ip-connectivity for M2M are networks. M2M gateway acts as a proxy performing translations from/to native protocols to/from Internet Protocol(IP). With an M2M gateway, each mode in an M2M area network appears as a virtualized node for external M2M area networks.

2. **SDN architecture**

   Key elements of SDN:

   1) **Centralized Network Controller :** With decoupled control and data planes and centralized network controller, the network administrators can rapidly configure the network.

2) **Programmable Open APIs**

SDN architecture supports programmable open APIs for interface between the SDN application and control layers (Northbound interface).

3) Standard Communication Interface (OpenFlow)

SDN architecture uses a standard communication interface between the control and infrastructure layers (Southbound interface). OpenFlow, which is defined by the Open Networking Foundation (ONF) is the broadly accepted SDN protocol for the Southbound interface.

3. NFV architecture

Network Function Virtualization(NFV)

- Network Function Virtualization (NFV) is a technology that leverages virtualization to consolidate the heterogeneous network devices onto industry standard high volume servers, switches and storage.
- NFV is complementary to SDN as NFV can provide the infrastructure on which SDN can run.

Key elements of NFV: NFV Architecture
  Virtualized Network Function (VNF):

VNF is a software implementation of a network function which is capable of running over the NFV Infrastructure (NFVI).

1) NFV Infrastructure (NFVI):

NFVI includes compute, network and storage resources that are virtualized.

### 2) NFV Management and Orchestration:

NFV Management and Orchestration focuses on all virtualization-specific management tasks and covers the orchestration and life-cycle management of physical and/or software resources that support the infrastructure virtualization, and the life-cycle management of VNFs.

Need for IoT Systems Management

Managing multiple devices within a single system requires advanced management capabilities.

1) **Automating Configuration :** IoT system management capabilities can help in automating the system configuration.
2) **Monitoring Operational & Statistical Data :** Management systems can help in monitoring opeartional and statistical data of a system. This data can be used for fault diagnosis or prognosis.
3) **Improved Reliability:** A management system that allows validating the system configurations before they are put into effect can help in improving the systemreliability.
4) **System Wide Configurations :** For IoT systems that consists of multiple devices or nodes, ensuring system wide configuration can be critical for the correct functioning of the system.
5) **Multiple System Configurations :** For some systems it may be desirable to have multiple valid configurations which are applied at different times or in certainconditions.

Retrieving & Reusing Configurations : Management systems which have the capability of retrieving configurations from devices can help in reusing the configurations for other devices of the same type.
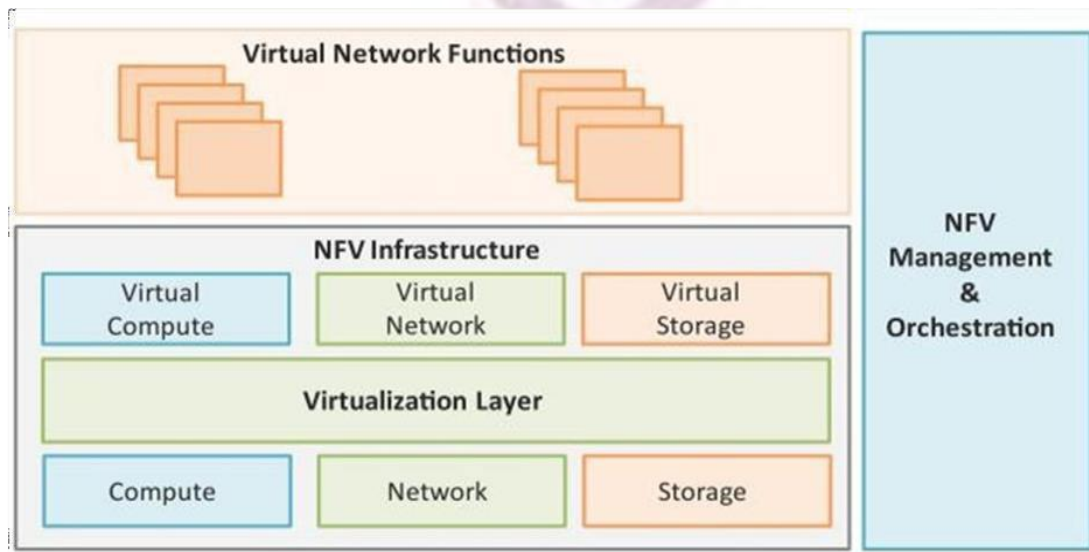
4.   IOT system Management with NETCONFIG – YANG

YANG is a data modeling language used to model configuration and state data manupulated by the NETCONF protocol.

The generic approach of IoT device management weith NETCONF-YANG.Roles of various components are:

1) Management System
2) Management API
3) Transaction Manager
4) Rollback Manager
5) Data Model Manager
6) Configuration Validator
7) Configuration Database
8) Configuration API
9) Data Provider API
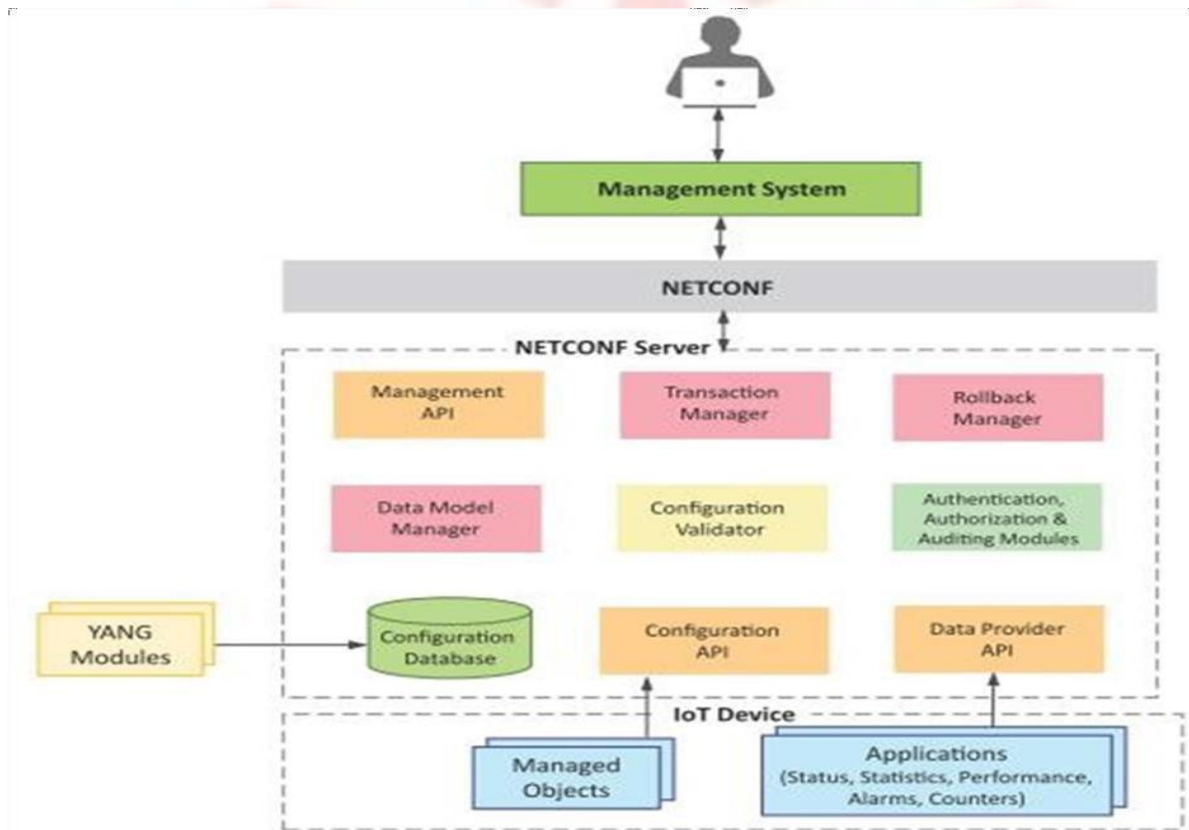
1) **Management System :** The operator uses a management system to send NETCONF messages to configure the IoT device and receives state information and notifications from the device as NETCONF messages.
2) **Management API :** allows management application to start NETCONF sessions.
3) **Transaction Manager:** executes all the NETCONF transactions and ensures that ACID properties hold true for the trasactions.
4) **Rollback Manager :** is responsible for generating all the transactions necessary to rollback a current configuration to its original state.
5) **Data Model Manager :** Keeps track of all the YANG data models and the corresponding managed objects. Also keeps track of the applications which provide data for each part of a data m,odel.
6) **Configuration Validator :** checks if the resulting configuration after applying a transaction would be a valid configuration.
7) **Configuration Database :** contains both configuration and operastional data.
8) **Configuration API :** Using the configuration API the application on the IoT device can be read configuration data from the configuration datastore and write opeartional data to the opearational datastore.
9) **Data Provider API:** Applications on the IoT device can register for callbacks for various events using the Data Provider API. Through the Data Provider API, the applications can report statistics and opeartional data.

5. Limitations of SNMP

SNMP is stateless in nature and each SNMP request contains all the information to process the request. The application needs to be intelligent to manage the device.

1. SNMP is a connectionless protocol which uses UDP as the transport protocol, making it unreliable as there was no support for acknowledgement of requests.
2. MIBs often lack writable objects without which device configuration is not possible using SNMP.
3. It is difficult to differentiate between configuration and state data in MIBs.
4. Retrieving the current configuration from a device can be difficult with SNMP. Earlier versions of SNMP did not have strong security features.
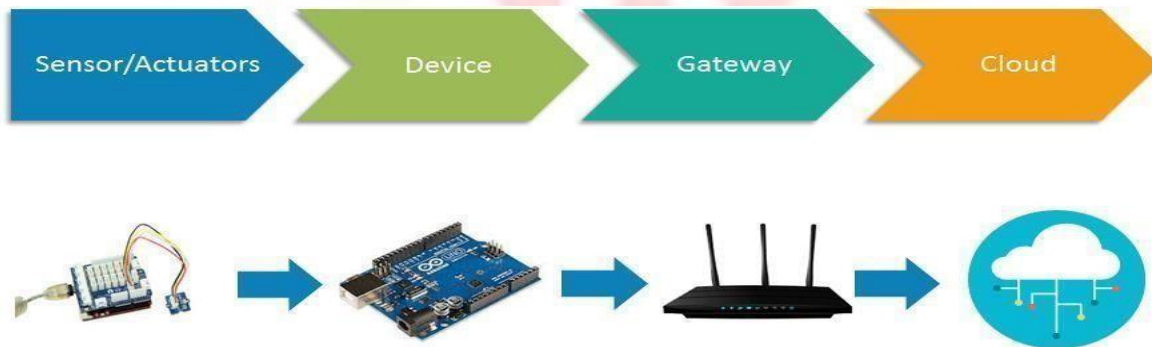
UNIT – 3

1. **IOT architecture**

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 Stage

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.
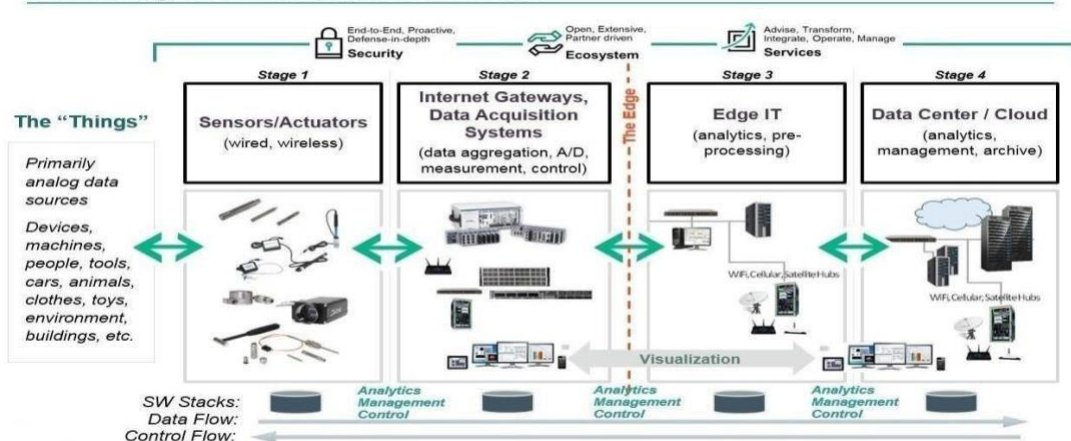
Stage 2:-

The Internet gateway

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.

Stage 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 center. 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 center 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 preprocess 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, analyze it, and send only projections as to when each device will fail or need service.

Stage 4:-

The data center and cloud

Data that needs more in-depth processing, and where feedback doesn't have to be immediate, gets forwarded to physical data center or cloud-based systems, where more powerful IT systems can analyze, 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 placeon-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.

2.   **Data types and List Datatypes**

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

There are various data types in Python. Some of the important types are listed below.

Python Numbers

Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined as int, float and complex class in Python. We can use the type() function to know which class a variable or a value belongs to and the isinstance() function to check if an object belongs to a particular class.

Script.py

1. a = 5
2. print(a, "is of type", type(a))
3. a = 2.0
4. print(a, "is of type", type(a))
5. a = 1+2j
6. print(a, "is complex number?", isinstance(1+2j,complex))

Integers can be of any length, it is only limited by the memory available. A floating point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. 1 is integer, 1.0 is floating point number. Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part. Here are some examples.

>>> a = 1234567890123456789

>>>a

1234567890

123456789

>>> b = 0.1234567890123456789

>>>b

0.12345678

901234568

>>> c = 1+2j

>>> c (1+2j)

Python List

List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type. Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ].

Course Name: INTERNET OF THINGS
Year/Sem: IV-I

```
>>> a = [1, 2.2,
    'python']
```

We can use the slicing operator [ ] to extract an item or a range of items from a list. Index starts form 0 in Python.

Script.py
```
1. a = [5,10,15,20,25,30,35,40]
2. # a[2] = 15
3. print("a[2] = ", a[2])
4. # a[0:3] = [5, 10, 15]
5. print("a[0:3] = ", a[0:3])
6. # a[5:] = [30, 35, 40]
7. print("a[5:] = ", a[5:])
```

Lists are mutable, meaning; value of elements of a list can be altered.
```
>>> a = [1,2,3]
>>> a[2]=4
```

Python Tuple

Tuple is an ordered sequences of items same as list. The only difference is that tuples are immutable. Tuples once created cannot be modified. Tuples are used to write- protect data and are usually faster than list as it cannot change dynamically. It is defined within parentheses () where items are separated by commas.

```
>>> t = (5,'program', 1+3j)
```

Python Strings

String is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """.

```
>>> s = "This is a string"
>>> s = '''a multiline
```

Like list and tuple, slicing operator [ ] can be used with string. Strings are immutable.
Script.py

```
a = {5,2,3,1,4}
#printing set variable print("a = ", a)
# data type of variable a print(type(a))
```

We can perform set operations like union, intersection on two sets. Set have unique values. They eliminate duplicates. Since, set are unordered collection, indexing has no meaning. Hence the slicing operator [] does not work. It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value. In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of anytype.

```
>>> d = {1:'value','key':2}

>>> type(d)

<class 'dict'>
```

We use key to retrieve the respective value. But not the other way around.

Script.py
```
d        =
{1:'value',' key':2} print(type( d)) print("d[1]
= ", d[1]);
print("d['key'] =
", d['key']); # Generates error print("d[2] = ", d[2]);
```

Python data structures Python if...else Statement
Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes. Decision making is required when we want to execute a code only if a certain condition is satisfied.

The if…elif…else statement is used in Python for decision making.

Python if Statement Syntax
If test
expressio n:
statement(s)

Here, the program evaluates the test expression and will execute statement(s) only if the text expression is True.
If the text expression is False, the statement(s) is not executed. In Python, the body of the if statement is indicated by the indentation. Body starts with an indentation and the first unindented line marks the end. Python interprets non-zero values as True. None and 0 are interpreted as False.
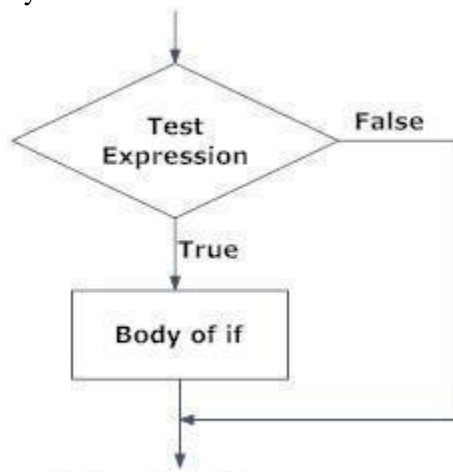
Python if Statement Flowchart

Fig: Operation of if statement

Example: Python if Statement

\# If the number is positive, we print an appropriate message num = 3
if num > 0:
print(num, "is a positive

number.")   print("This   is always printed.")
num = -1 if num > 0: print(num, "is a positive
number.") print("This is also always printed.")

When you run the program, the output will be:
3   is   a positive number This is   always printed
This is also always printed.

In the above example, num > 0 is the test expression. The body of if is executed only if this evaluates to True.
When variable num is equal to 3, test expression is true and body inside body of if is executed. If variable num
is equal to -1, test expression is false and body inside body of if is skipped. The print() statement falls outside
of the if block (unindented). Hence, it is executed regardless of the test expression.

Python if...else Statement Syntax
if   test
expressi on:
Body of if
else:
Body of else

The if..else statement evaluates test expression and will execute body of if only when test condition is True.
If the condition is False, body of else is executed. Indentation is used to separate the blocks.
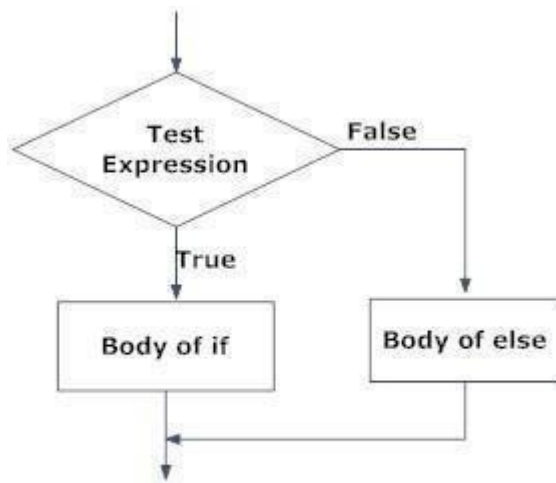
Python if..else Flowchart

Fig: Operation of if...else statement

Example of if...else

```
# Program checks if the number is positive or negative # And displays an appropriate message
num = 3
# Try these two  variations as well. # num = -5
# num = 0
if num >= 0: print("Positiv e or Zero")
else:
print("Negative number")
```

In the above example, when num is equal to 3, the test expression is true and body of if is executed and body of else is skipped.

If num is equal to -5, the test expression is false and body of else is executed and body of if is skipped.

If num is equal to 0, the test expression is true and body of if is executed and body of else is skipped.
Python if...elif...else Statement Syntax if test expression:
Body of if
elif test expression:

Body of elif else:
Body of else

The elif is short for else if. It allows us to check for multiple expressions. If the condition for if is False, it checks the condition of the next elif block and so on. If all the conditions are False, body of else is executed. Only one block among the several if...elif...else blocks is executed according to the condition. The if block can have only one else block. But it can have multiple elif blocks.

Python File Methods Files
File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk). Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data. When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed. Hence, in Python, a file operation takes place in the following order.

Open a file
Read or write (perform operation)
Close the file

How to open a file?

Python has a built-in function open() to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

```
>>> f = open("test.txt") # open file in current directory
>>> f = open("C:/Python33/README.txt") # specifying full path
```

We can specify the mode while opening a file. In mode, we specify whether we want to read 'r', write 'w' or append 'a' to the file. We also specify if we want to open the file in text mode or binary mode. The default is reading in text mode. In this mode, we get strings when reading from the file. On the other hand, binary mode returns bytes and this is the mode to be used when dealing with non-text files like image or exe files.

Python File Modes

Mode	Description
'r'	Open a file for reading. (default)
'w'	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode.
'+'	Open a file for updating (reading and writing)

```
f = open("test.txt")
# equivalent    to 'r'    or 'rt'
f=open("test.txt",'w') # write in text mode
f = open("img.bmp",'r+b') # read and write in binary mode
```

Unlike other languages, the character 'a' does not imply the number 97 until it is encoded using ASCII (or other equivalent encodings). Moreover, the default encoding is platform dependent. In windows, it is 'cp1252' but 'utf-8' in Linux. So, we must not also rely on the default encoding or else our code will behave differently in different platforms. Hence, when working with files in text mode, it is highly recommended to specify the encoding type.

```
f = open("test.txt",mode = 'r',encoding = 'utf-8')
```

Python Program with Raspberry PI with focus of interfacing external gadgets, controlling output, reading input from pins.

Light an LED through Python program.

Solution:
One of the biggest selling points of the Raspberry Pi is its GPIO, or General Purpose Input/Output ports. They are the little pins sticking out of the circuit board and allow you to plug various devices into your Raspberry Pi. With a little programming, you can then control them or detect what they are doing.
In this tutorial I am going to show you how to light an LED. In addition to your Raspberry Pi running Raspbian, what you will need is:
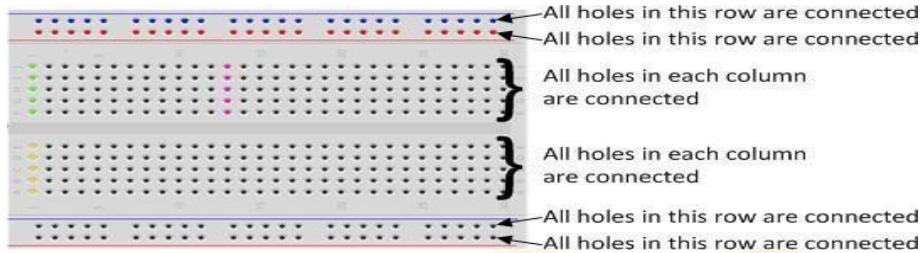
A Breadboard
An LED
A 330 ohm resistor

The Breadboard:

he breadboard is a way of connecting electronic components to each other without having to solder them together. They are often used to test a circuit design before creating a Printed Circuit Board (PCB).

The holes on the breadboard are connected in a pattern.

With the breadboard in the CamJam EduKit, the top row of holes are all connected together – marked with red dots. And so are the second row of holes – marked with blue dots. The same goes for the two rows of holes at the bottom of the breadboard.

In the middle, the columns of wires are connected together with a break in the middle. So, for example, all the green holes marked are connected together, but they are not connected to the yellow holes, nor the purple ones. Therefore, any wire you poke into the green holes will be connected to other wires poked into the other green holes.

The LED:

When you pick up the LED, you will notice that one leg is longer than the other. The longer leg (known as the 'anode'), is always connected to the positive supply of the circuit. The shorter leg (known as the 'cathode') is connected to the negative side of the power supply, known as 'ground'.

LEDs will only work if power is supplied the correct way round (i.e. if the 'polarity' is correct). You will not break the LEDs if you connect them the wrong way round – they will just not light. If you find that they do not light in your circuit, it may be because they have been connected the wrong way round.

LED stands for Light Emitting Diode, and glows when electricity is passed through it.

The Resistor:

You must ALWAYS use resistors to connect LEDs up to the GPIO pins of the Raspberry Pi. The Raspberry Pi can only supply a small current (about 60mA). The LEDs will want to draw more, and if allowed to they will burn out the Raspberry Pi. Therefore putting the resistors in the circuit will ensure that only this small current will flow and the Raspberry Pi will not be damaged.

Resistors are a way of limiting the amount of electricity going through a circuit; specifically, they limit the amount of 'current' that is allowed to flow. The measure of resistance is calledthe Ohm (Ω), and the larger the resistance, the more it limits the current. The value of a resistor is marked with coloured bands along the length of the resistor body.

You will be using a 330Ω resistor. You can identify the 330Ω resistors by the colour bands along the body. The colour coding will depend on how many bands are on the resistors supplied:

If there are four colour bands, they will be Orange, Orange, Brown, and then Gold.
If there are five bands, then the colours will be Orange, Orange, Black, Black, Brown.
It does not matter which way round you connect the resistors. Current flows in both ways through them.

Jumper Wires:

Jumper wires are used on breadboards to 'jump' from one connection to another. The ones you will be using in this circuit have different connectors on each end. The end with the 'pin' will go into the Breadboard. The end with the piece of plastic with a hole in it will go onto the Raspberry Pi's GPIO pins.

The Raspberry Pi's GPIO Pins :

GPIO stands for **General Purpose Input Output**. It is a way the Raspberry Pi can control and monitor the outside world by being connected to electronic circuits. The Raspberry Pi is able to control LEDs, turning them on or off, or motors, or many other things. It is also able to detect whether a switch has been pressed, or temperature, or light. In the CamJam EduKit you will learn to control LEDs and a buzzer, and detect when a button has been pressed. The diagram below left shows the pin layout for a Raspberry Pi Models A and B (Rev 2 - the original Rev 1 Pi is slightly different), looking at the Raspberry Pi with the pins in the top right corner. The new 40 pin Raspberry Pi's shares exactly the same layout of pins for the top 13 rows of GPIOpins.

Building the Circuit:

The circuit consists of a power supply (the Raspberry Pi), an LED that lights when the power is applied, and

Models A & B

Models A+, B+ & Pi2

a resistor to limit the current that can flow through the circuit.

You will be using one of the 'ground' (GND) pins to act like the 'negative' or 0 volt ends of a battery. The 'positive' end of the battery will be provided by a GPIO pin. Here we will be using pin 18. When they are 'taken high', which means it outputs 3.3 volts, the LED will light. Now take a look at the circuit diagram below. You should turn your Raspberry Pi off for the next bit, just in case you accidentally short something out.
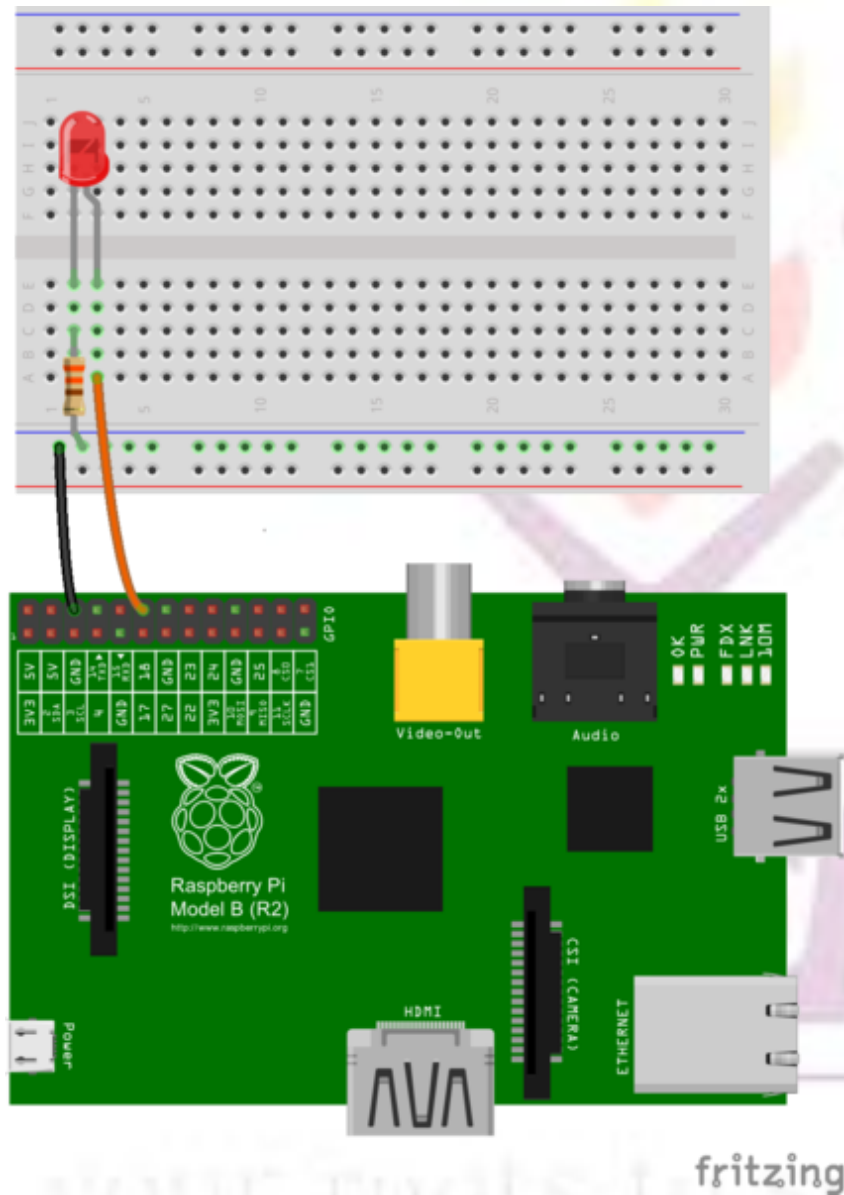
Use one of the jumper wires to connect a ground pin to the rail, marked with blue, on the breadboard. The female end goes on the Raspberry Pi's pin, and the male end goes into a hole on the breadboard. Then connect the resistor from the same row on the breadboard to a column on the breadboard, as shown above.

Next, push the LEDs legs into the breadboard, with the long leg (with the kink) on the right.

Lastly, complete the circuit by connecting pin 18 to the right hand leg of the LED. This is shown here with the orange wire.
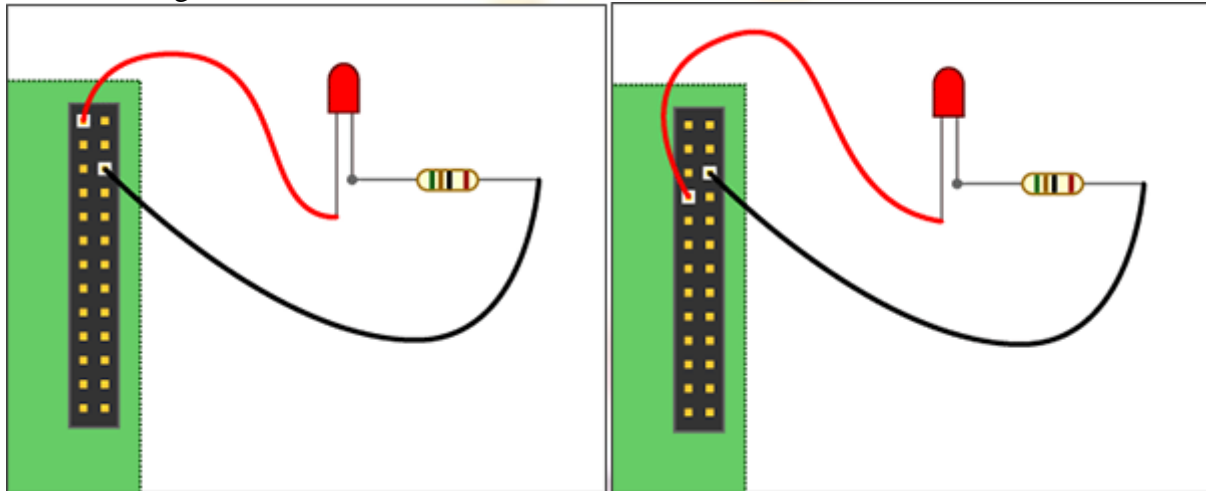
The Code:

You are now ready to write some code to switch the LED on. Turn on your Raspberry Pi and open the terminal window.

Create a new text file "LED.py" by typing the following:

nano LED.py

Type in the following code:

import RPi.GPIO as GPIO import time GPIO.setmode(GPIO.BCM) GPIO.setwarnings(False) GPIO.setup(18,GPIO.OUT)

print "LED on" GPIO.output(18,GPIO.HIGH)
time.sleep(1) print "LED off"
GPIO.output(18,GPIO.LOW)

Once you have typed all the code and checked it, save and exit the text editor with "Ctrl + x" then "y" then "enter".

Running the Code:

To run this code type:

sudo python LED.py

You will see the LED turn on for a second and then turn off.

If your code does not run and an error is reported, edit the code again using nano LED.py.
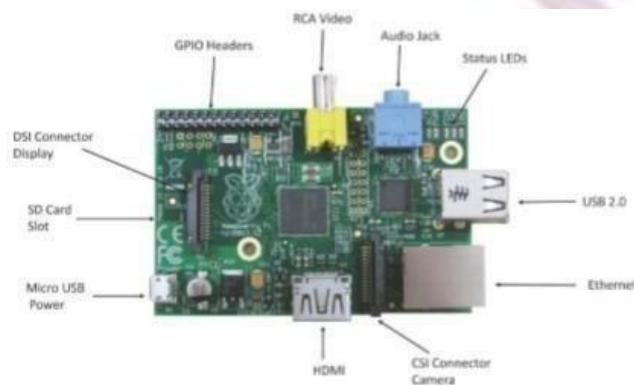

UNIT – 4

Introduction to Raspberry Pi
Raspberry Pi GPIO Pin Description
Basic building blocks of IOT Device
Raspberry Pi interfaces
Other IOT devices

Introduction to Raspberry Pi Raspberry Pi
Raspberry Pi is a low-cost mini-computer with the physical size of a credit card. Raspberry Pi
runs various flavors of Linux and can perform almost all tasks that a normal desktop computer can do. Raspberry Pi also allows interfacing sensors and actuators through the general purpose I/O pins. Since Raspberry Pi runs Linux operating system, it supports Python "out of the box". Raspberry Pi is a low-cost mini-computer with the physical size of a credit card. Raspberry Pi runs various flavors of Linux and can perform almost all tasks that a normal desktop computer can do. Raspberry Pi also allows interfacing sensors and actuators through the general purpose I/O pins. Since Raspberry Pi runs Linux operating system, it supports Python "out of the box".

Raspberry Pi

Linux on Raspberry Pi

Raspbian: Raspbian Linux is a Debian Wheezy port optimized for Raspberry Pi.
Arch: Arch is an Arch Linux port for AMD devices.
Pidora: Pidora Linux is a Fedora Linux optimized for Raspberry Pi.
RaspBMC: RaspBMC is an XBMC media-center distribution for Raspberry Pi.
OpenELEC: OpenELEC is a fast and user-friendly XBMC media-center distribution.
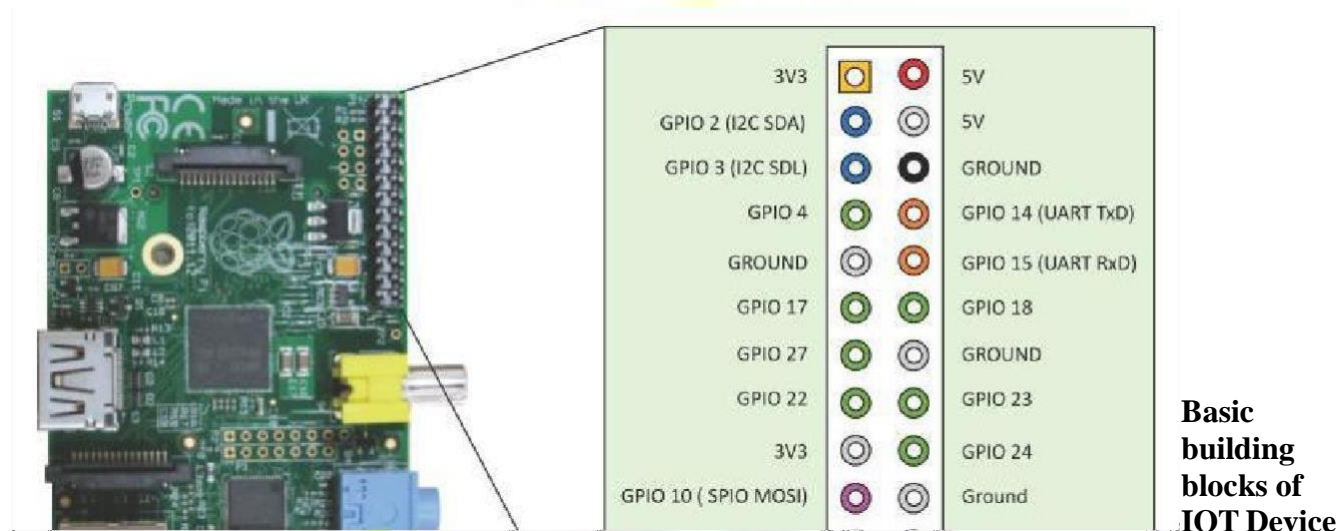RISC OS: RISC OS is a very fast and compact operating system.

Raspberry Pi GPIO Pin Description Raspberry Pi GPIO

Basic building blocks of IOT Device

Basic building blocks of an IoT Device

Sensing: Sensors can be either on-board the IoT device or attached to the device.
Actuation: IoT devices can have various types of actuators attached that allow taking actions upon the physical entities in the vicinity of thedevice.
Communication: Communication modules are responsible for sending collected data to other devices or cloud-based servers/storage and receiving data from other devices and commands from remote applications.

Analysis & Processing: Analysis and processing modules are responsible for making sense of the collected data.
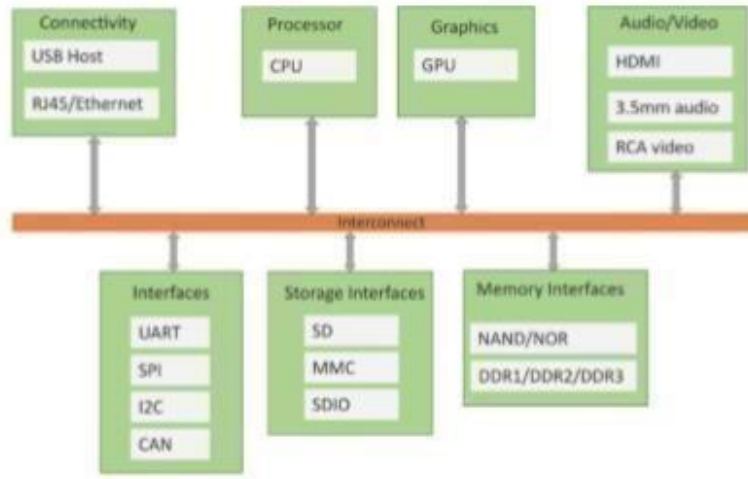
Block diagram of an IoT Device

Raspberry Pi interfaces

Serial: The serial interface on Raspberry Pi has receive (Rx) and transmit (Tx) pins for communication with serial peripherals.
SPI: Serial Peripheral Interface (SPI) is a synchronous serial data protocol used for communicating with one or more peripheral devices.
I2C: The I2C interface pins on Raspberry Pi allow you to connect hardware modules. I2C interface allows synchronous data transfer with just two pins - SDA (data line) and SCL (clock line)

Other IOT devices
pcDuino
BeagleBone Black
Cubieboard

<div align="center">

UNIT – 5

</div>

INTRODUCTION TO CLOUD COMPUTING
CLOUD STORAGE API'S
WAMP FOR IOT
PYTHON PACKAGES
PYTHON WEB APPLICATION FRAMEWORK – DIJANGO
CASE STUDY IN IOT- SMART CITIES

INTRODUCTION TO CLOUD COMPUTING

The Internet of Things (IoT) involves the internet-connected devices we use to perform the processes and services that support our way of life. Another component set to help IoT succeed is cloud computing, which acts as a sort of front end. Cloud computing is an increasingly popular service that offers several advantages to IOT, and is based on the concept of allowing users to perform normal computing tasks using services delivered entirely over the internet. A worker may need to finish a major project that must be submitted to a manager, but perhaps they encounter problems with memory or space constraints on their computing device. Memory and space constraints can be minimized if an application is instead hosted on the internet. The worker can use a cloud computing service to finish their work because the data is managed remotely by a server. Another example: you have a problem with your mobile device and you need to reformat it or reinstall the operating system. You can use Google Photos to upload your photos to internet-based storage. After the reformat or reinstall, you can then either move the photos back to you device or you can view the photos on your device from the

internet when you want.
Concept

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.
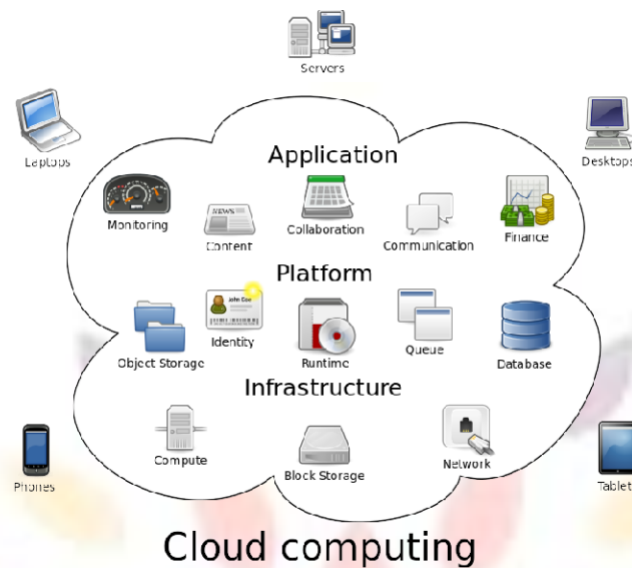
Cloud computing

Service models

Service delivery in cloud computing comprises three different service models: software as a service (SaaS), platform as a service (PaaS), and 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 possible disadvantages.

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.

CLOUD STORAGE API'S

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 ccess 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.

Cloud Models are relied on Communication API
Communication API facilitate data transfer, control information   transfer from application to cloud, one service to another
It also exist in the form of Communication Protocols
It supports RPC, PUBSUB and WAMP
Eg. Popular API is RESTful API (communication in cloud model)
Django web framework is used to implement Communication API

WAMP FOR IOT

Web Application Messaging Protocol (WAMP) is a sub-protocol of Websocket which provides publish-subscribe and remote procedure call (RPC) messaging patterns.

WAMP

Transport: Transport is channel that connects two peers.
Session: Session is a conversation between two peers that runs over a transport.

Client: Clients are peers that can have one or more roles. In publish-subscribe model client can have following roles:

Publisher: Publisher publishes events (including payload) to the topic maintained by the broker.

Subscriber: Subscriber subscribes to the topics and receives the events including thepayload. In RPC model client can have following roles: –

Caller: Caller issues calls to the remote procedures along with call arguments. – Callee: Callee executes the procedures to which the calls are issued by the caller and returns the results back to the caller. • Router: Routers are peers that perform generic call and event routing. In publish-subscribe model Router has the role of a Broker: – Broker: Broker acts as a router and

routes messages published to a topic to all subscribers subscribed to thetopic. In RPC model Router has the role of a Broker: –

Dealer: Dealer acts a router and routes RPC calls from the Caller to the Callee and routes results from Callee to Caller.

Application Code: Application code runs on the Clients (Publisher, Subscriber, Callee or Caller).

PYTHON PACKAGES JSON

JavaScript Object Notation (JSON) is an easy to read and write data-interchange format. JSON is used as an alternative to XML and is easy for machines to parse and generate.

JSON is built on two structures: a collection of name–value pairs (e.g., a Python dictionary) and ordered lists of values (e.g., a Python list).

XML

XML (Extensible Markup Language) is a data format for structured document interchange. The Python minidom library provides a minimal implementation of the Document Object Model interface and has an API similar to that in other languages.

HTTPLib & URLLib

HTTPLib2 and URLLib2 are Python libraries used in network/internet programming.

SMTPLib

Simple Mail Transfer Protocol (SMTP) is a protocol which handles sending email and routing email between mail servers. The Python SMTPLib module provides an SMTP client session object that can be used to send email.

NumPy

NumPy is a package for scientific computing in Python. NumPy provides support for large multi-dimensional arrays and matrices.

Scikit-learn

Scikit-learn is an open source machine learning library for Python that provides implementations of various machine learning algorithms for classification, clustering, regression and dimension reduction problems.

PYTHON WEB APPLICATION FRAMEWORK – DIJANGO

Django is an open source web application framework for developing web applications in Python.
A web application framework in general is a collection of solutions, packages and best practices that allows development of web applications and dynamic websites.

sed on the Model–Template–View architecture and provides data model from the business rules and the user interface.

Django provides a unified API to a database backend.

Thus, web applications built with Django can work with different databases without requiring any code changes.

With this flexibility in web application design combined with the powerful capabilities of the Python language and the Python ecosystem, Django is best suited for cloud applications.

Django consists of an object-relational mapper, a web templating system and a regular- expression-based URL dispatcher.

Django uses a Model–Template–View (MTV) framework.

Model

The model acts as a definition of some stored data and handles the interactions with the database. In a web application, the data can be stored in a relational database, non-relational database, an XML file, etc. A Django model is a Python class that outlines the variables and methods for a particular type of data.

Template

In a typical Django web application, the template is simply an HTML page with a few extra placeholders. Django's template language can be used to create various forms of text files (XML, email, CSS, Javascript, CSV, etc.).

View

The view ties the model to the template. The view is where you write the code that actually generates the web pages. View determines what data is to be displayed, retrieves the data from the database and passes the data to the template.

Case Study in IoT: Smart Cities

The Internet-of-Things (IoT) is the novel cutting-edge technology which proffers to connect plethora of digital devices endowed with several sensing, actuation and computing capabilities with the Internet, thus offers manifold new services in the context of a smart city. The appealing IoT services and big data analytics are enabling smart city initiatives all over the world. These services are transforming cities by improving infrastructure, transportation systems, reduced traffic congestion, waste management and the quality of human life. In this paper, we devise a taxonomy to best bring forth a generic overview of IoT paradigm for smart cities, integrated information and communication technologies (ICT), network types, possible opportunities and major requirements. Moreover, an overview of the up-to-date efforts from standard bodies is presented. Later, we give an overview of existing open source IoT platforms for realizing smart city applications followed by several exemplary case studies. In addition, we summarize the latest synergies and initiatives worldwide taken to promote IoT in the context of smart cities. Finally, we highlight several challenges in order to give future research directions.
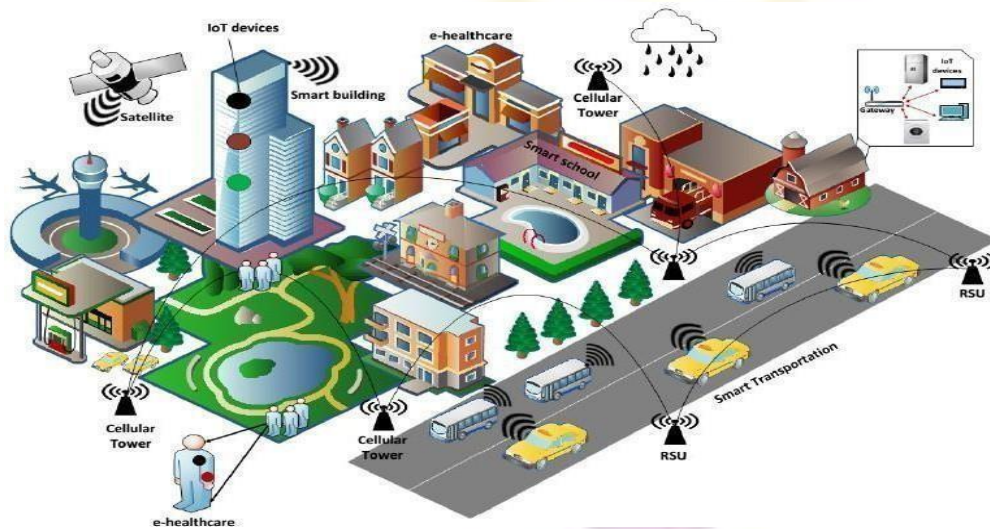
An illustration of IoT based smart city
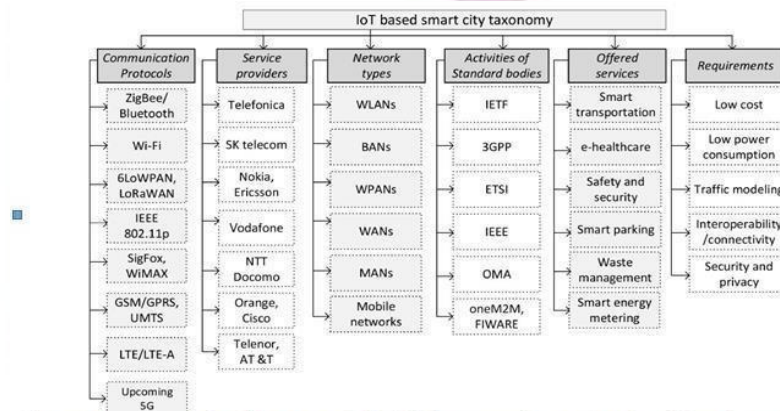
A representation of IoT-based smart city taxonomy

This section presents a taxonomy of IoT based smart cities which categorizes the literature on the basis of existing communication protocols, major service providers, network types, standardization efforts, offered services, and crucial requirements.

Communication Protocols

IoT based smart city realization significantly relies on numerous short and wide range communication protocols to transport data between devices and backend servers. Most prominent short range wireless technologies include Zig-Bee, Bluetooth, Wi-Fi, Wireless Metropolitan Area Network (WiMAX) and IEEE 802.11p which are primarily used in smart metering, e-healthcare and vehicular communication. Wide range technologies such as Global System for Mobile communication (GSM) and GPRS, Long-Term Evolution (LTE), LTE- Advanced are commonly utilized in ITS such as vehicle-to infrastructure (V2I), mobile e-healthcare, smart grid and infotainment services. Additionally, LTE-M is considered as an evolution for cellular IoT (C-IoT). In Release 13, 3GPP plans to further improve coverage, battery lifetime as well as device complexity [7]. Besides well-known existing protocols, LoRa alliance standardizes the LoRaWAN protocol to support smart city applications to primarily ensure interoperability between several operators. Moreover, SIGFOX is an ultra narrowband radio technology with full star-based infrastructure offers a high scalable global network for realizing smart city applications with extremely low power consumption. A comparative summary2 of the major communication protocols.

Service Providers
Pike Research on smart cities estimated this market will grow to hundreds of billion dollars by 2020, with an annual growth of nearly 16 billion. IoT is recognized as a potential source to increase revenue of service providers. Thus, well-known worldwide service providers have already started exploring this novel cutting edge communication paradigm. Major service providers include Telefonica, SK telecom, Nokia, Ericsson, Vodafone, NTT Docomo, Orange, Telenor group and AT&T which offer variety of services and platforms for smart city applications such as ITS and logistics, smart metering, home automation and e-healthcare.

Network Types
IoT based smart city applications rely on numerous network topologies to accomplish a fully autonomous environment. The capillary IoT networks offer services over a short range. Examples include wireless local area networks (WLANs), BANs and wireless personal area networks (WPANs). The application areas include indoor e-healthcare services, home automation, street lighting. On the other hand, applications such as ITS, mobile e-healthcare and waste management use wide area networks (WANs), metropolitan area networks (MANs), and mobile communication networks. The above networks pose distinct features in terms of data, size, coverage, latency requirements, and capacity.

14.Tutorial topics and Questions

UNIT WISE 2 MARKS
UNIT-I

Define Internet of Things?

A. Internet of Things (loT) comprises things that have unique identities and are connected to the Internet.

What are the characteristics of IoT?
The characteristics of IoT
Self-Configuring
Dynamic and Self-Adapting
Interoperable Communication Protocols
Unique Identify
Integrated into information Network

List of IoT protocols used in Link Layer?
A. IoT protocols used in Link Layer are
1. 802.3-Ethernet 2. 802.11-WiFi
3. 802.16-WiMax
4. 802.15.4-LR_WPAN 2G/3G/4G

What are IoT protocols used in Transport layer?
There are only two protocols used in Transport layer are TCP and UDP

5. List out the various applications of IoT?
A. The various applications of
1.Cities
2.Environment
3.Energy
4.Retail
5.Logistics
6.Agriculture
7. Industry Health & Lifestyle
8. IoTHome

6. Define Knowledge?
A. Knowledge is inferred from information by organizing and structuring, information and is put into action to achieve specific objectives. The various protocols used in Network Layer are
1. IPv4
2. IPv6 6LoWPAN

7. What are the communications models are used in IoT?

   A. IoT communication models are

Request-Response

Publish-Subscribe

Push-Pull

Exclusive Pair.

8. Define REST-based communication Protocol?

A. Representational State Transfer (REST) is a style of architecture based on a set of principles that describe how networked resources are defined and addressed.

3 MARKS

List of IoT protocols used in Application Layer?

The protocols used in application layer are

HHTP, CoAP, WebSocket, MQTT, XMPP, DDS, AMQP

What is the advantage of using WebSocket-based Communication API's?

WebSocket API's allows bi-directional, full duplex communication between clients and servers.

Define Stateless?

Each request from client to server must contain all the information needed for understating the request and does not take any advantage of stored data.

What is the action taken by the 'PUT' HHTP Method?

PUT HTTP command updates a resource present in the database.

What communication model is used in WebSocket-based Communication API?

Exclusive Pair Communication model is used are establish communication between client and   server.

What are the main components used for making an IoT System?

A. The main components used for making an IoT System are

   Device

   Communication
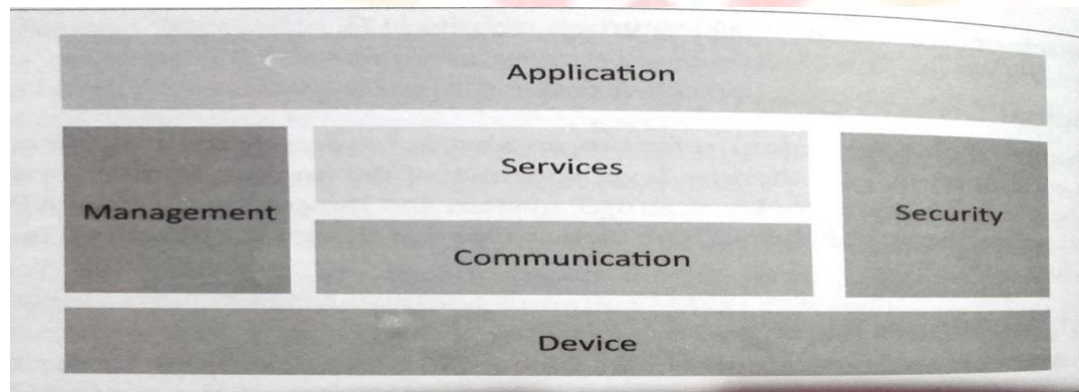
   Services

   Management

   Security

5 Marks

1. Explain Logical Devices of IOT?

LOGICAL DESIGN of IoT

Refers to an abstract represent of entities and processes without going into the low level specifies of implementation.

1) IoT Functional Blocks
2) IoT Communication Models
3) IoT Comm. APIs

IoT Functional Blocks: Provide the system the capabilities for identification, sensing, actuation
Communication and management.

Device: An IoT system comprises of devices that provide sensing, actuation, monitoring and control functions.

Communication: handles the communication for IoT system.

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

**Management:** Provides various functions to govern the IoT system.

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

Application: IoT application provide an interface that the users can use to control and monitor various aspects of IoT system.

1)   IoT Communication Models:

1) Request-Response
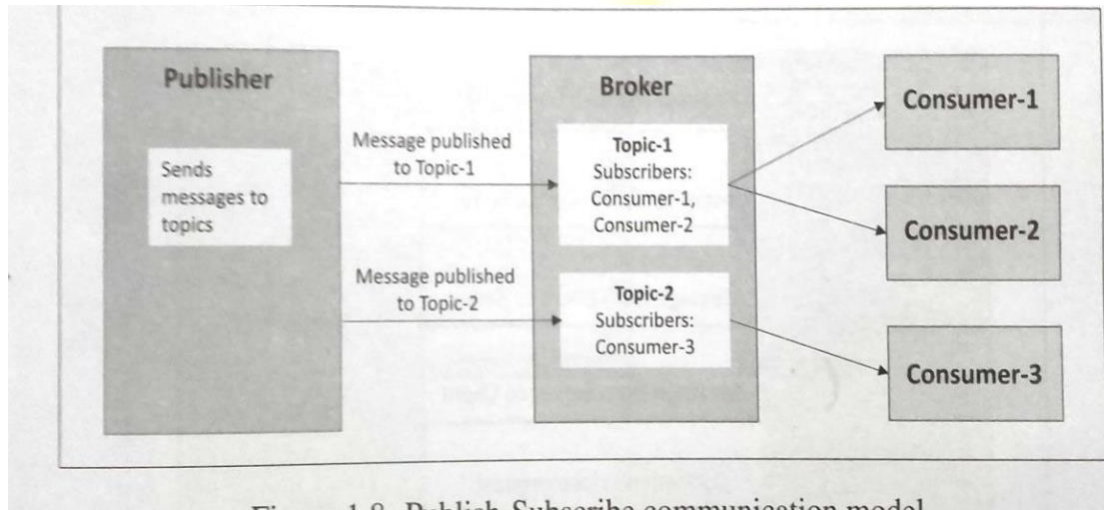 2) Publish-Subscibe
3) Push-Pull
4) Exclusive Pair

1)   Request-Response Model:
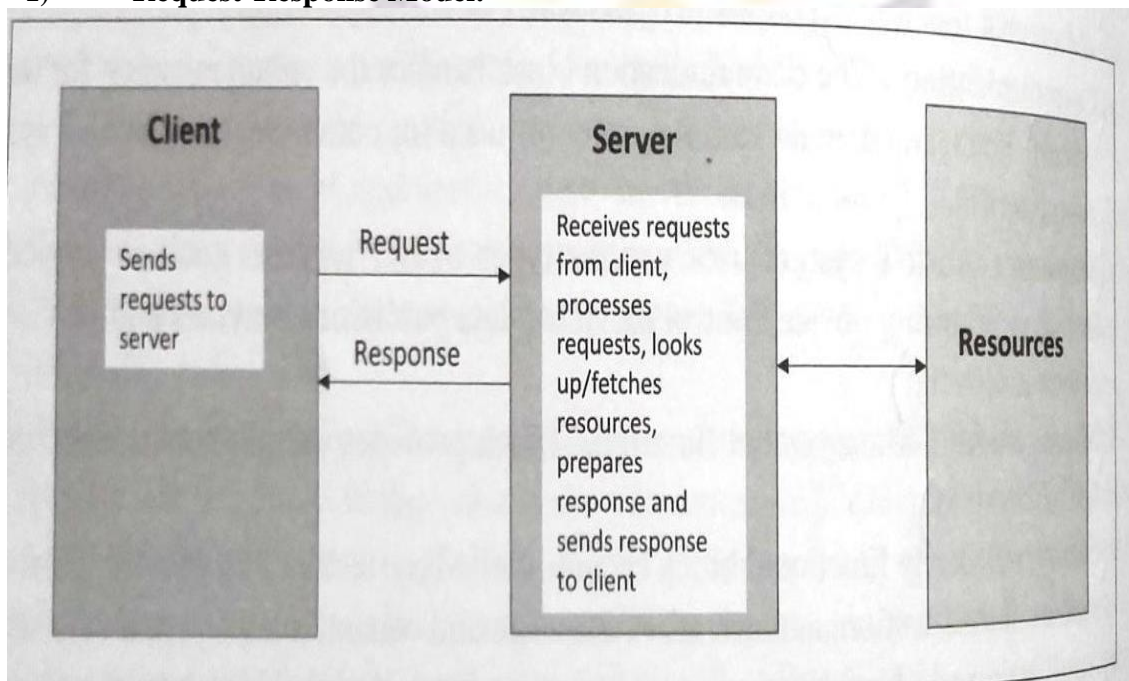
In which the client sends request to the server and the server replies to requests. Is a stateless
communication model and each request-response pair is independent of others.

2) **Publish-Subscibe Model:**

Figure 1.8: Publish Subscribe communication model

Involves publishers, brokers and 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 publisher, it sends the data to all the subscribed consumers.

3) **Push-Pull Model:** in which data producers push data to queues and consumers pull data from the queues. Producers do not need to aware of the consumers. Queues help in decoupling the message between the producers and consumers.

4) Exclusive Pair: is bi-directional, fully duplex communication model that uses a persistent connection between the client and server. Once connection is set up 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 connections.