

IOT

MODULE I

Introduction to Internet of Things

Introduction-Definition & Characteristics of IoT , **Physical Design of IoT**- Things in IoT , IoT Protocols, **Logical Design of IoT**- IoT Functional Blocks, IoT Communication Models, IoT Communication APIs , **IoT Enabling Technologies**- Wireless Sensor Networks , Cloud Computing, Big Data Analytics , Communication Protocols , Embedded Systems, **IoT Levels & Deployment Templates**.

MODULE II

Domain Specific IoTs

Home Automation: Smart Lighting, Smart Appliances, Intrusion Detection, Smoke/Gas Detectors, **Cities**-Smart Parking, Smart Lighting, Smart Roads, Structural Health Monitoring, Surveillance, Emergency Response, **Environment**-Weather Monitoring, Air Pollution Monitoring, Noise Pollution Monitoring, Forest Fire Detection , River Floods Detection , **Energy**- Smart Grids , Renewable Energy Systems , Prognostics , **Retail**-Inventory Management , Smart Payments , Smart Vending Machines , **Logistics**-Route Generation & Scheduling , Fleet Tracking , Shipment Monitoring , Remote Vehicle Diagnostics, **Agriculture**-Smart Irrigation ,Green House Control ,**Industry** - Machine Diagnosis & Prognosis Indoor Air Quality Monitoring ,**Health & Lifestyle** -Health & Fitness Monitoring, Wearable Electronics **IoT and M2M Introduction**, **M2M-Difference between IoT and M2M**, **SDN and NFV for IoT**- Software Defined Networking , Network Function Virtualization

MODULE III

IoT Platforms Design Methodology

IoT Design Methodology-Purpose & Requirements Specification ,Process Specification, Domain Model Specification, Information Model Specification , Service Specifications , IoT Level Specification, Functional View Specification , Operational View Specification , Device & Component Integration , Application Development, **Case Study on IoT System for Weather Monitoring, Motivation for Using Python**

IoT Physical Devices & Endpoints

What is an IoT Device-Basic building blocks of an IoT Device, **Exemplary Device: Raspberry Pi, About the Board, Linux on Raspberry Pi , Raspberry Pi Interfaces** – Serial, SPI , I2C , **Programming Raspberry Pi with Python**- Controlling LED with Raspberry Pi , Interfacing an LED and Switch with Raspberry Pi ,Interfacing a Light Sensor (LDR) with Raspberry Pi , **Other IoT Devices**- pcDuino, Beagle Bone Black , Cubieboard

MODULE IV

IoT & Beyond : Use of Big Data and Visualization in IoT, Industry 4.0 Concepts. Overview of RFID, Low-power design (Bluetooth Low Energy), range extension techniques (data mining and mesh networking), and data-intensive IoT for continuous recognition applications. Overview of Android / IOS App Development tools & Internet Of Everything

Introduction and concepts

Definition: - The internet of things (IoT) is a computing concept that describes the idea of everyday physical objects being connected to the internet and being able to identify themselves to other devices. It has 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 network and use intelligent interfaces.

- **Dynamic and Self-Adapting:-** IoT devices and system may have the capability to change dynamically depending upon the system and operating conditions or sensed environment.
For example, the surveillance cameras can change their modes based on day or night.
- **Self-configuring:-** IoT devices have self-configuring capability which allows large number of devices to work together to work provide certain functionality they can change their networking and update the software automatically.
- **Interoperable Communication Protocol:** - IoT devices can communicate with number of interoperable (communicate with other devices without special effort) communication protocols.
- Unique ID:** - IoT devices have a unique identity differentiated with unique IP address.
- **Integrated into Information Network:** - IoT devices are integrated into the information network that allows them to communicate and exchange data with other devices and system.



PHYSICAL DESIGN OF IoT

Things in IoT

- The word “Things” refers to IoT devices which have unique identities and can perform remote sensing, actuating and monitoring capabilities. These devices can exchange and communicate with each other.
- The IoT devices consists of several interfaces for connection to other devices both wired and wireless which includes
 1. I/O interfaces for sensors
 2. Interfaces for internet connectivity
 3. Memory and storage
 4. Audio and video interface

IoT Protocols

- **Link Layer**

This protocol determines how the data is physically sent over the network layer (e.g. copper wire, coaxial cable or a radio wave). It determines how the packet are coded and signaled by the hardware device over the medium to which the host is attached. Example:-

1. IEEE 802.3--Ethernet (wired connection)
2. 802.11 –Wi-Fi
3. 802.16—WiMax
4. 2G/3G/4G—Mobile communication

- **Network/Internet Layer**

The network layers are responsible for sending of IP datagram's from the source network to the destination network. It performs host addressing and packet routing. The datagram's consists of source and destination addresses where host identifies using IP schemes as IPV4 and IPV6.

- **IPV4**:- It is used to identify the devices on a network using hierarchical addressing scheme. It uses 32- bit address that allows total 2^{32} or 4 billion devices 128
- **IPV6**:- It is the new version of internet protocol which uses 128-bits address that allows 2^{128} or 3×10^{38} address.

- **Transport Layer**

The transport layer protocols provide end to end message transfer capability independent of the underlying network. The function of the transport layer is to provide functions such as error control, segmentation, flow control and congestion control.

- TCP: - It is most widely used for data transmission in communication network such as internet .it provides process to process communication using port numbers. It uses port number for communication which keeps Track of segments that are received and transmitted.
- UDP: - It is the simplest protocol that involves minimum amount of communication mechanism. It is connectionless, unreliable transport protocol. It does not provide guaranteed delivery of the message.

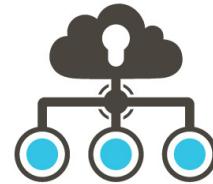
Types of IoT Protocols

IoT protocols and standards can be broadly classified into two separate categories.

Types of IoT Protocols



IoT Network Protocols



IoT Data Protocols

1. IoT Network Protocols

IoT network protocols are used to connect devices over the network. These are the set of communication protocols typically used over the Internet. Using IoT network protocols, end-to-end data communication within the scope of the network is allowed. Following are the various IoT Network protocols:

- **HTTP (HyperText Transfer Protocol)**

HyperText Transfer Protocol is the best example of IoT network protocol. This protocol has formed the foundation of data communication over the web. It is the most common protocol that is used for IoT devices when there is a lot of data to be published. However, the HTTP protocol is not preferred because of its cost, battery-life, energy saving, and more constraints.

Additive manufacturing/3D printing is one of the use cases of the HTTP protocol. It enables computers to connect 3D printers in the network and print three-dimensional objects and pre-determined process prototypes.

- **LoRaWan (Long Range Wide Area Network)**

It is a long-range low power protocol that provides signal detection below the noise level. LoRaWan connects battery operated things wirelessly to the Internet in either private or global networks. This communication protocol is mainly used by smart cities, where there are millions of devices that function with less power and memory.

Smart street lighting is the practical use case of LoRaWan IoT protocol. The street lights can be connected to a LoRa gateway using this protocol. The gateway, in turn, connects to the cloud application that controls the intensity of light bulbs automatically based on the ambient lighting, which helps in reducing the power consumption during day-times.

- **Bluetooth**

Bluetooth is one of the most widely used protocols for short-range communication. It is a standard IoT protocol for wireless data transmission. This communication protocol is secure and perfect for short-range, low-power, low-cost, and wireless transmission between electronic devices. BLE (Bluetooth Low Energy) is a low-energy version of Bluetooth protocol that reduces the power consumption and plays an important role in connecting IoT devices.

Bluetooth protocol is mostly used in smart wearables, smartphones, and other mobile devices, where small fragments of data can be exchanged without high power and memory. Offering ease of usage, Bluetooth tops the list of IoT device connectivity protocols.

- **ZigBee**

ZigBee is an IoT protocol that allows smart objects to work together. It is commonly used in home automation. More famous for industrial settings, ZigBee is used with apps that support low-rate data transfer between short distances. Street lighting and electric meters in urban areas, which provides low power consumption, use the ZigBee communication protocol. It is also used with security systems and in smart homes.

2. IoT Data Protocols

IoT data protocols are used to connect low power IoT devices. These protocols provide point-to-point communication with the hardware at the user side without any Internet connection. Connectivity in IoT data protocols is through a wired or a cellular network. Some of the IoT data protocols are:

- **Message Queue Telemetry Transport (MQTT)**

One of the most preferred protocols for IoT devices, MQTT collects data from various electronic devices and supports remote device monitoring. It is a subscribe/publish protocol that runs over Transmission Control Protocol (TCP), which means it supports event-driven message exchange through wireless networks.

MQTT is mainly used in devices which are economical and requires less power and memory. For instance, fire detectors, car sensors, smart watches, and apps for text-based messaging.

- **Constrained Application Protocol (CoAP)**

CoAP is an internet-utility protocol for restricted gadgets. Using this protocol, the client can send a request to the server and the server can send back the response to the client in HTTP. For light-weight implementation, it makes use of UDP (User Datagram Protocol) and reduces space usage. The protocol uses binary data format EXL (Efficient XML Interchanges).

CoAP protocol is used mainly in automation, mobiles, and microcontrollers. The protocol sends a request to the application endpoints such as appliances at homes and sends back the response of services and resources in the application.

- **Advanced Message Queuing Protocol (AMQP)**

AMQP is a software layer protocol for message-oriented middleware environment that provides routing and queuing. It is used for reliable point-to-point connection and supports the seamless and secure exchange of

data between the connected devices and the cloud. AMQP consists of three separate components namely Exchange, Message Queue, and Binding. All these three components ensure a secure and successful exchange and storage of messages. It also helps in establishing the relationship of one message with the other.

AMQP protocol is mainly used in the banking industry. Whenever a message is sent by a server, the protocol tracks the message until each message is delivered to the intended users/destinations without failure.

- **Machine-to-Machine (M2M) Communication Protocol**

It is an open industry protocol built to provide remote application management of IoT devices. M2M communication protocols are cost-effective and use public networks. It creates an environment where two machines communicate and exchange data. This protocol supports the self-monitoring of machines and allows the systems to adapt according to the changing environment.

M2M communication protocols are used for smart homes, automated vehicle authentication, vending machines, and ATM machines.

- **Extensible Messaging and Presence Protocol (XMPP)**

The XMPP is uniquely designed. It uses a push mechanism to exchange messages in real-time. XMPP is flexible and can integrate with the changes seamlessly. Developed using open XML (Extensible Markup Language), XMPP works as a presence indicator showing the availability status of the servers or devices transmitting or receiving messages.

Other than the instant messaging apps such as Google Talk and WhatsApp, XMPP is also used in online gaming, news websites, and Voice over Internet Protocol (VoIP).

Application Layer

An application layer protocol defines how application processes (clients and servers), running on different end systems, pass messages to each other. In particular, an application layer protocol defines:

- The types of messages, e.g., request messages and response messages.
- The syntax of the various message types, i.e., the fields in the message and how the fields are delineated.
- The meaning of the information that the field is supposed to contain.
- Rules for determining when and how a process sends messages and responds to messages.

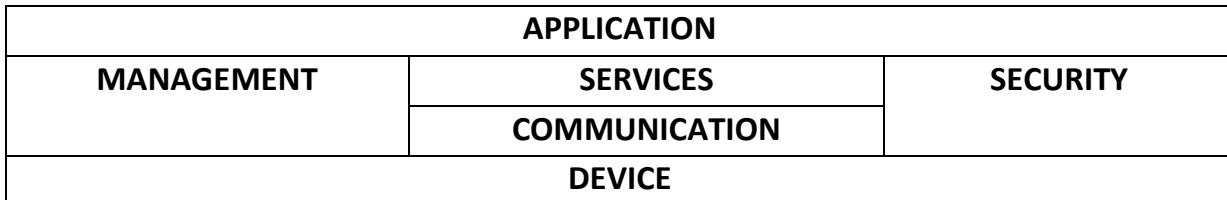
Application layer protocol enables process to process connection using ports.

Application Type	Application-layer protocol
Electronic mail	Send: Simple Mail Transfer Protocol SMTP [RFC 821] Receive: Post Office Protocol v3 POP3 [RFC 1939]
M2M	CoAP
World Wide Web (WWW)	HyperText Transfer Protocol 1.1 HTTP 1.1 [RFC 2068]
File Transfer	File Transfer Protocol FTP [RFC 959] Trivial File Transfer Protocol TFTP [RFC 1350]
Internet telephony	Proprietary (e.g., Vocaltec)

Logical Design of IoT

Logical design of an IoT describes about abstract representation of the entities and process without going to low level specifics of the implementation.

➤ IoT functional Blocks



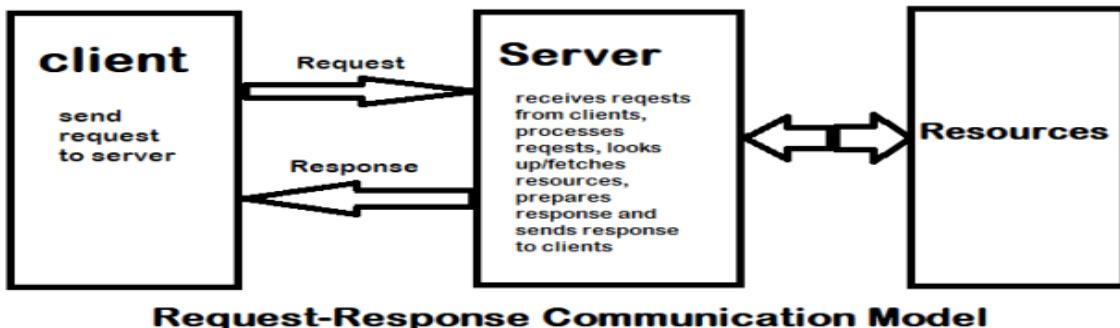
The IoT system consists of different functional blocks which provide the system capabilities.

These blocks includes:

- **Device:** - This block deal with IoT device which provide sensing, monitoring and control functions.
- **Communication:** - This block deals with IoT communication protocol.
- **Services:** - This block deals with various types of IoT services such as device monitoring, device control services and device discovery.
- **Management:** - This block used to monitor the complete IoT system.
- **Security:**- It provides the security by providing the functions such as authentication, authorization and data security.
- **Application:**-IoT provides an user interfaces to monitor various IoT system where user can view and analyze the processing data.

➤ IoT Communication Model

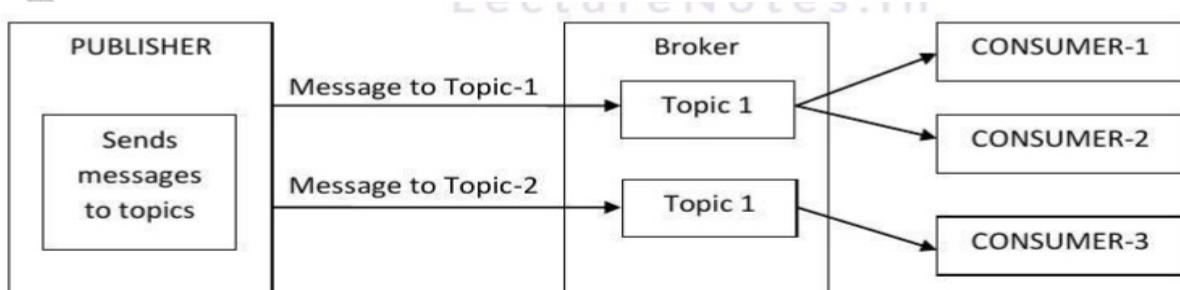
- Request-Response



In Request–Response communication model client sends request to the server and the server responds to the request. When the server receives the request it decides how to respond, fetches the data, retrieves resources, and prepares the response and sends to the client. R-R is a communication model where request and response pair is independent of each others.

- **Publish-Subscribe:**

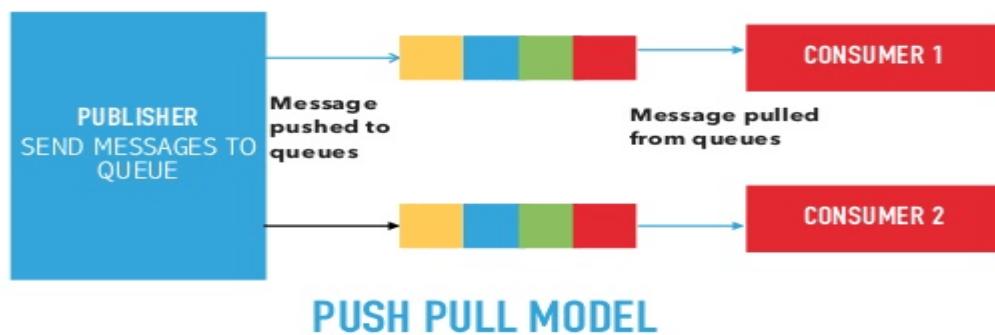
This model involves publishers, brokers and consumers. Publishers are the sources of data. Publishers send the data to the topic which are managed by the broker. Publishers are not aware of the consumers. Consumers subscribe to the topics which are managed by the broker. When broker receives the data from the Publisher, it sends to all the consumers.



- **Push-Pull:**

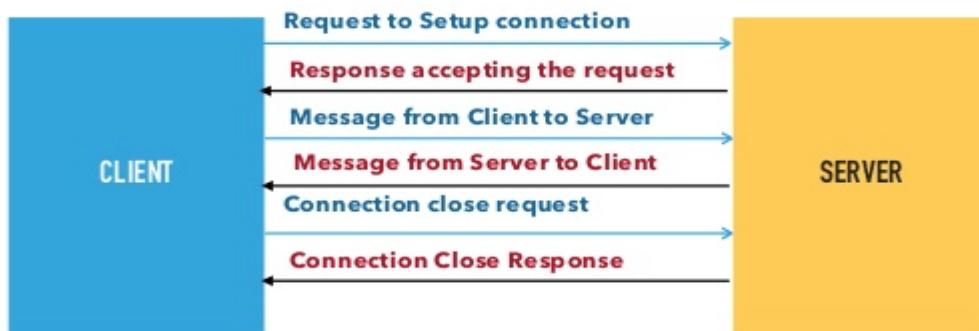
In this model the producers push the data in queues and the consumers pull the data from the queues. Producers do not need to be aware of the consumers. Queues help in decoupling the messaging between

the producer and consumers. Queues also act as buffer which helps in situation when there is mismatch between the rate at which the producers push the data and consumers pull the data.



- **Exclusive Pair:**

Exclusive pair is a bi-directional, fully duplex communication model that uses a persistent connection between the client and server, once the connection is established it remains open until the client sends a request to closer the connection. Client and server can send the message to each other after connection setup. In this model server is aware of all the open connection.



EXCLUSIVE PAIR COMMUNICATION MODEL

IOT Communication API's

The application program (or programming) interface, or API, is arguably what really ties together the connected “things” of the “internet of things.” IoT APIs are the points of interaction between an IoT device and the internet and/or other elements within the network.

As API management company Axway puts it, “APIs are tightly linked with IoT because they allow you to securely expose connected devices to customers, go-to-market channels and other applications in your IT infrastructure.”

Generally we used Two APIs For IoT Communication. These IoT Communication APIs are:

- REST-based Communication APIs
- WebSocket-based Communication APIs

REST-based Communication APIs

Representational state transfer (REST) is a set of architectural principles by which you can design Web services the Web APIs that focus on systems’s resources and how resource states are addressed and transferred. REST APIs that follow the request response communication model, the rest architectural constraint apply to the components, connector and data elements, within a distributed hypermedia system. The rest architectural constraint are as follows:

Client-server – The principle behind the client-server constraint is the separation of concerns. for example clients should not be concerned with the storage of data which is concern of the serve. Similarly the server should not be

concerned about the user interface, which is concern of the client. Separation allows client and server to be independently developed and updated.

Stateless – Each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server. The session state is kept entirely on the client.

Cacheable – Cache constraints requires that the data within a response to a request be implicitly or explicitly leveled as cacheable or non cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests. Caching can partially or completely eliminate some instructions and improve efficiency and scalability.

Layered system – layered system constraints, constrains the behavior of components such that each component cannot see beyond the immediate layer with they are interacting. For example, the client cannot tell whether it is connected directly to the end server or two an intermediary along the way. System scalability can be improved by allowing intermediaries to respond to requests instead of the end server, without the client having to do anything different.

Uniform interface – uniform interface constraints requires that the method of communication between client and server must be uniform. Resources are identified in the requests (by URLs in web based systems) and are themselves separate from the representations of the resources data returned to the client. When a client holds a representation of resources it has all the information required to update or delete the resource you (provided the client has required permissions). Each message includes enough information to describe how to process the message.

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.

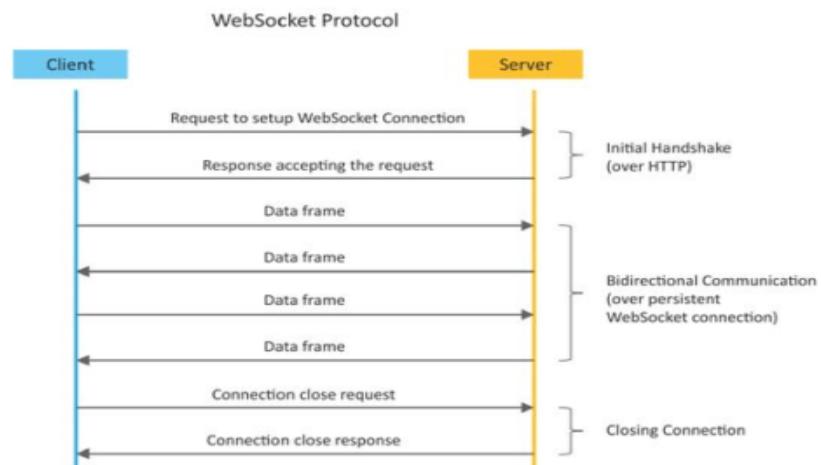
A RESTful web service is a "Web API" implemented using HTTP and REST principles. REST is most popular IoT Communication APIs.

Uniform Resource Identifier (URI)	GET	PUT	PATCH	POST	DELETE
Collection, such as https://api.example.com/resources/	List the URIs and perhaps other details of the collection's members.	Replace the entire collection with another collection.	Not generally used	Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.	Delete the entire collection.
Element, such as https://api.example.com/resources/item5	Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	Replace the addressed member of the collection, or if it does not exist, create it.	Update the addressed member of the collection.	Not generally used. Treat the addressed member as a collection in its own right and create a new entry within it.	Delete the addressed member of the collection.

HTTP methods

WebSocket based communication API

Websocket APIs allow bi-directional, full duplex communication between clients and servers. Websocket APIs follow the exclusive pair communication model. Unlike request-response model such as REST, the WebSocket APIs allow full duplex communication and do not require new connection to be setup for each message to be sent. Websocket communication begins with a connection setup request sent by the client to the server. The request (called websocket handshake) is sent over HTTP and the server interprets it as an upgrade request. If the server supports websocket protocol, the server responds to the websocket handshake response. After the connection setup client and server can send data/messages to each other in full duplex mode. Websocket API reduce the network traffic and latency as there is no overhead for connection setup and termination requests for each message. Websocket is suitable for IoT applications that have low latency or high throughput requirements. So Web socket is most suitable IoT Communication APIs for IoT System.



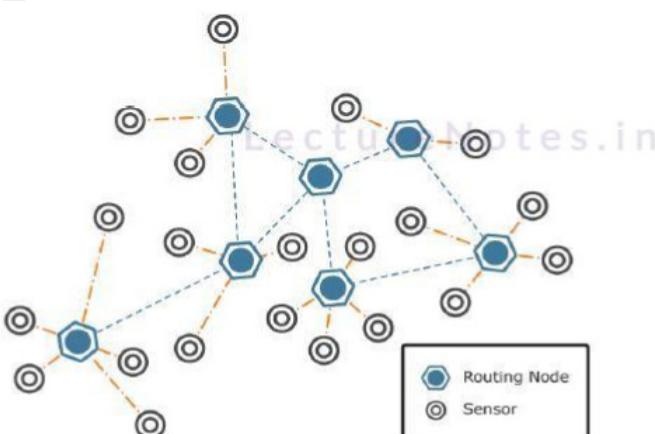
IoT Enabling Technologies:

IoT is enabled by several technologies which includes WSN, cloud computing, big data analysis, embedded systems, web servers, mobile internet etc. Some of the technologies which play the key role in IoT are:

➤ WIRELESS SENSOR NETWORK:

A Wireless Sensor Network (WSN) is a distributed network with large number of sensors which are used to monitor the environmental and physical conditions. A WSN consists of end nodes, routers and coordinator. End nodes have several sensors attached to them where the data is passed to coordinator with the help of routers. The coordinator also acts as gateway that connects WSN to internet. WSN are enabled by wireless communication protocol such as 802.15.4. and ZigBee is one of the most popular wireless technologies. Some of the examples where WSN is used in the IOT are:

1. Weather monitoring system uses WSN which collects data of temperature, humidity which are aggregated and analyzed.
2. Indoor air quality monitoring system uses WSN.
3. Soil Moisture Monitoring system
4. Surveillance systems use WSN for collecting data.
5. Health monitoring system.



➤ **CLOUD COMPUTING:**

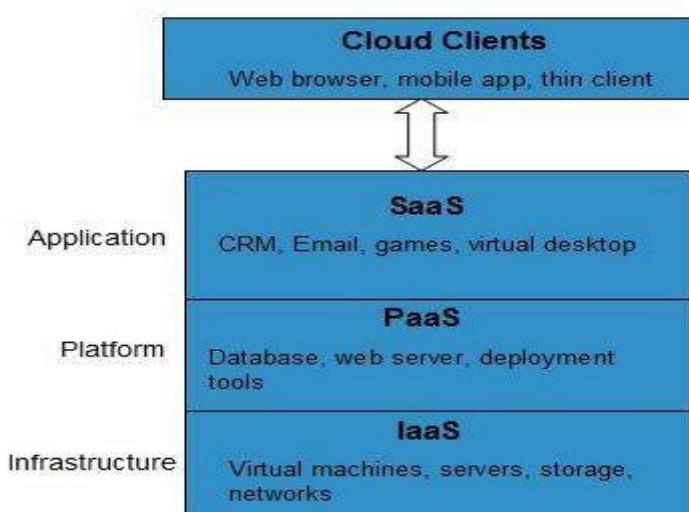
Cloud computing is the use of various services, such as software development platforms, servers, storage and software, over the internet, often referred to as the "cloud." Generally the cloud computing based on three characteristics such as:

1. The back-end of the application (especially hardware) is completely managed by a cloud vendor.
2. A user only pays for services used (memory, processing time and bandwidth, etc.).
3. Services are scalable.

The Cloud computing provides ability to "pay on demand" and scale quickly is largely a result of cloud computing vendors being able to pool resources that may be divided among multiple clients.

The cloud computing provides different services such as:

1. Infrastructure as a Service (IaaS)
2. Platform as a Service (PaaS)
3. Software as a Service (SaaS).



Infrastructure as a Service (IaaS)

Infrastructure as a service provides companies with computing resources including servers, networking, storage, and data center space on a pay-per-use basis.

The benefits of IaaS

- No need to invest in your own hardware
- Infrastructure scales on demand to support dynamic workloads
- Flexible, innovative services available on demand

Platform as a Service (PaaS)

Platform as a service provides a cloud-based environment with everything required to support the complete lifecycle of building and delivering web-based (cloud) applications — without the cost and complexity of buying and managing the underlying hardware, software, provisioning, and hosting.

The benefits of PaaS

- Develop applications and get to market faster
- Deploy new web applications to the cloud in minutes
- Reduce complexity with middleware as a service

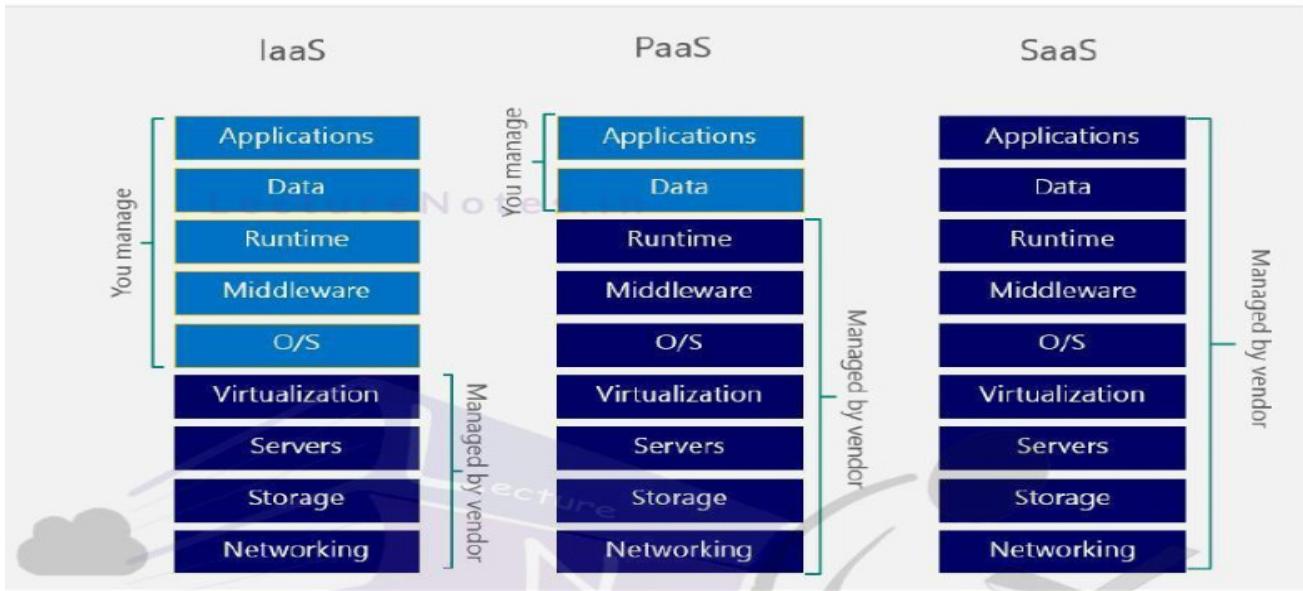
Software as a Service (SaaS)

Software as a service — run on distant computers "in the cloud" that are owned and operated by others and that connect to users' computers via the internet and, usually, a web browser.

The benefits of SaaS

- You can sign up and rapidly start using innovative business apps

- Apps and data are accessible from any connected computer
- No data is lost if your computer breaks, as data is in the cloud
- The service is able to dynamically scale to usage needs



➤ Big Data Analytics:

Big data analytics refers to the strategy of analyzing large volumes of data, or big data. This big data is gathered from a wide variety of sources, including social networks, videos, digital images, sensors, and sales transaction records. The large data is difficult to store, manage, process and analyze the data using traditional databases and data processing tools. There are several steps which involves in analyzing big data are data cleansing, data managing, data processing and visualization. Some examples of big data generated by IoT are:

1. Sensors data generated by weather monitoring stations.
2. Data generated by IoT systems for location and tracking of vehicles
3. Sensors embedded in industry and energy system.
4. Health and fitness data generated by IoT system such as fitness bands.

➤ Embedded systems:

Embedded system is a combination of hardware and software system used to perform special tasks. Embedded system includes microcontroller or microprocessor, memory (RAM, ROM), networking unit, input/output units and storage. It collects the data analyze and sends the data to internet.

IoT Levels and Deployment Templates

The IoT system consists of several systems which includes:

- Database: Database can be either local or in the cloud and stores the data generated by the IoT device.
- Web Service: Web services serve as a link between the IoT device, application, database and analysis components.
- Analysis Component: This is responsible for analyzing the IoT data and generating results in a form that is easy for the user to understand.
- Application: IoT applications provide an interface that the users can use to control and monitor various aspects of the IoT system. Applications also allow users to view the system status and the processed data.

➤ IoT level-1

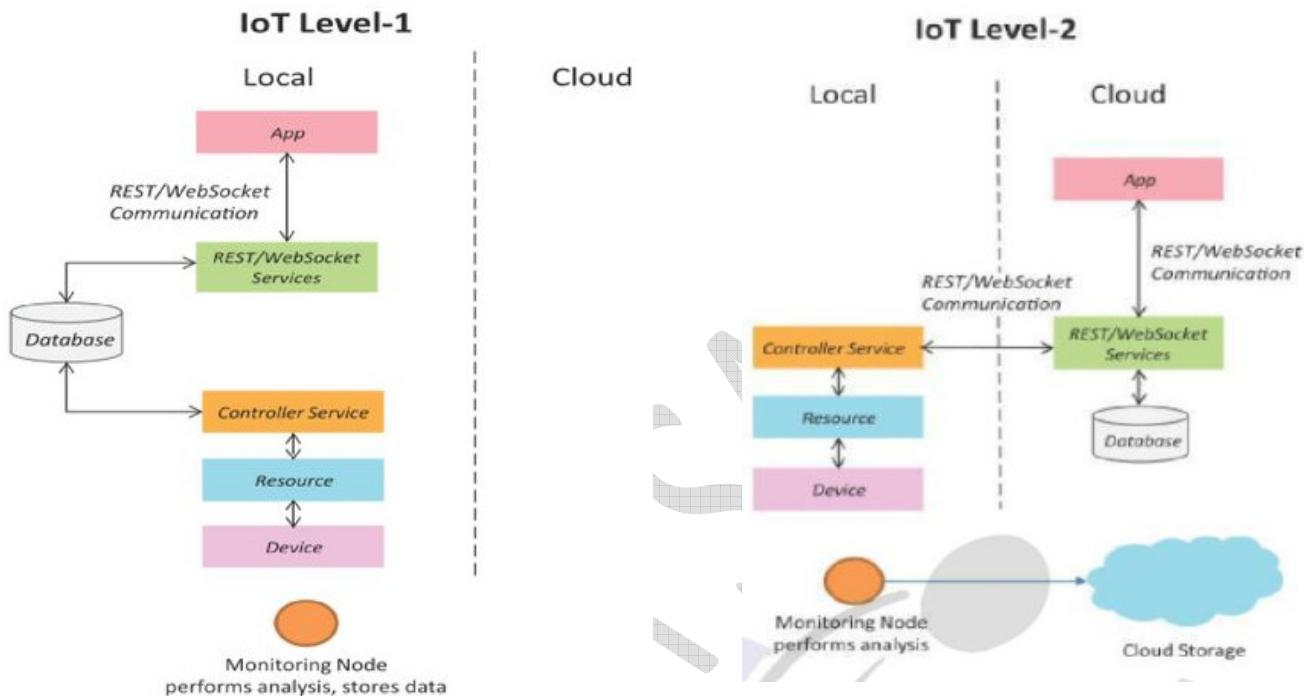
A level - 1 IoT system has a single node/device that performs sensing and/or actuation, stores data, performs analysis and hosts the application. Level - 1 IoT systems are suitable for modelling low cost and low complexity solutions where the data involved is not big and the analysis requirements are not computationally intensive.

➤ IoT level-2:

- A level-2 IoT system has a single node that performs sensing and/or actuation and local analysis. Data is stored in the cloud and the application is usually cloud-based.
- Level-2 IoT systems are suitable for solutions where the data involved is big; however, the primary analysis requirement is not computationally intensive and can be done locally.

➤ IoT level-3:

- A level-3 IoT system has a single node. Data is stored and analyzed in the cloud and the application is cloud-based.
- Level-3 IoT systems are suitable for solutions where the data involved is big and the analysis requirements are computationally intensive.

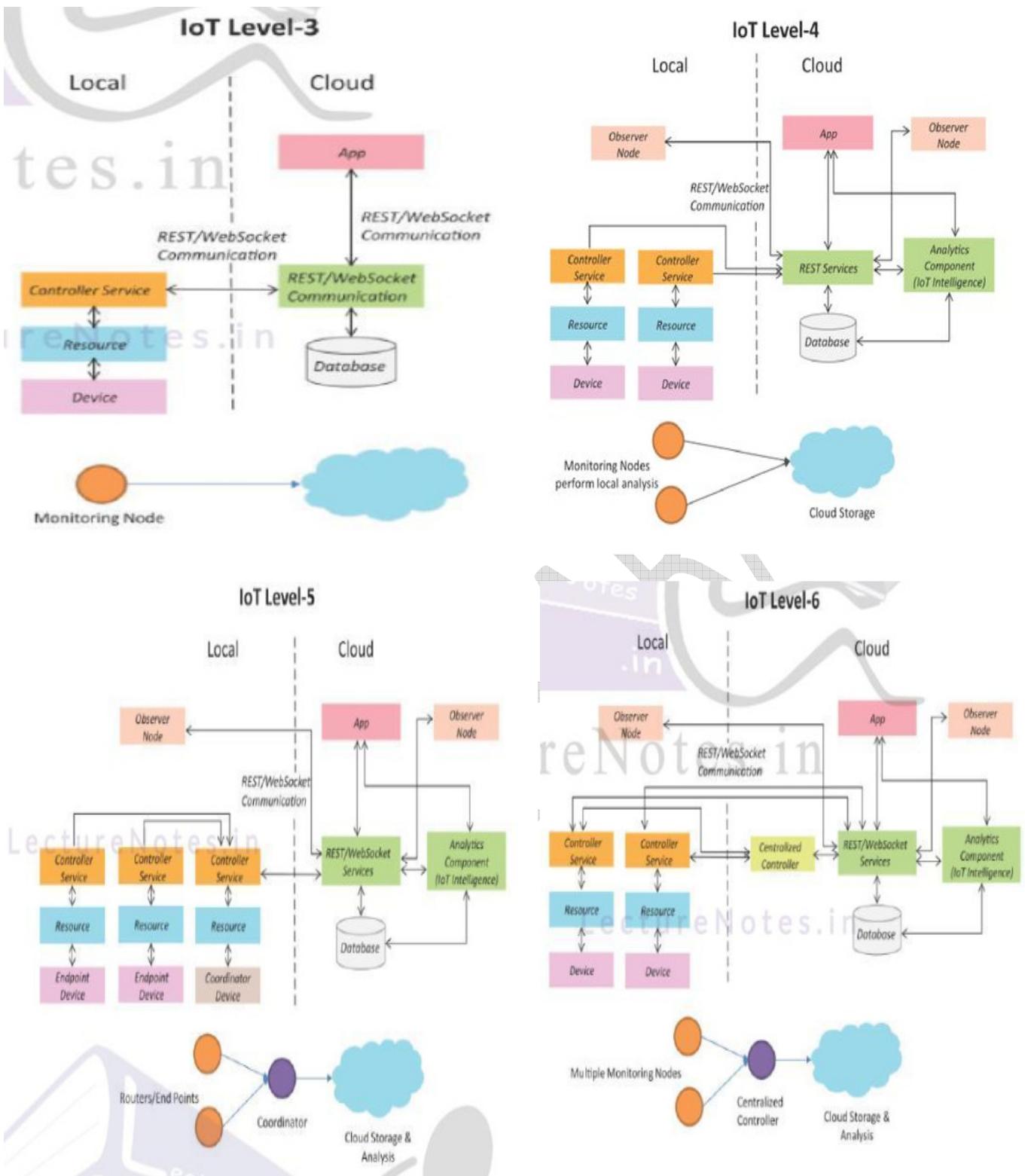


➤ IoT level-4:

- A level-4 IoT system has multiple nodes that perform local analysis. Data is stored in the cloud and the application is cloud-based.
- Level-4 contains local and cloud-based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices.
- Level-4 IoT systems are suitable for solutions where multiple nodes are required, the data involved is big and the analysis requirements are computationally intensive.

➤ IoT level-5:

- A level-5 IoT system has multiple end nodes and one coordinator node. The end nodes perform sensing and/or actuation. The coordinator node collects data from the end nodes and sends it to the cloud. Data is stored and analyzed in the cloud and the application is cloud-based.
- Level-5 IoT systems are suitable for solutions based on wireless sensor networks, in which the data involved is big and the analysis requirements are computationally intensive.



➤ IoT level-6:

- A level-6 IoT system has multiple independent end nodes that perform sensing and/or actuation and send data to the cloud. Data is stored in the cloud and the application is cloud-based. The analytics component analyzes the data and stores the results in the cloud database.
- The results are visualized with the cloud-based application. The centralized controller is aware of the status of all the end nodes and sends control commands to the nodes.

MODULE-2

Domain Specific IoTs

IoT Applications for :

- Home
- Cities
- Environment
- Energy Systems
- Retail
- Logistics
- Industry
- Agriculture
- Health & Lifestyle



IoT applications for smart homes:

- Smart Lighting
- Smart Appliances
- Intrusion Detection
- Smoke / Gas Detectors

Home Automation Smart Lighting

- Smart lighting achieves energy savings by sensing the human movements and their environments and controlling the lights accordingly.
- Key enabling technologies for smart lighting include :
 - Solid state lighting (such as LED lights)
 - IP-enabled lights
- Wireless-enabled and Internet connected lights can be controlled remotely from IoT applications such as a mobile or web application.
- Paper:
 - Energy-aware wireless sensor network with ambient intelligence for smart LED lighting system control [IECON, 2011]-> presented controllable LED lighting system that is embedded with ambient intelligence gathered from a distributed smart WSN to optimize and control the lighting system to be more efficient and user-oriented.

Home Automation Smart Appliances

- Smart appliances make the management easier and provide status information of appliances to the users remotely. E.g: smart washer/dryer that can be controlled remotely and notify when the washing/drying cycle is complete.
- **Open Remote** is an open source automation platform for smart home and building that can control various appliances using mobile and web applications.
- It comprises of three components:
 - a Controller-> manages scheduling and runtime integration between devices.
 - a Designer -> allows to create both configuration for the controller and user interface designs.
 - Control Panel -> allows to interact with devices and control them.
- Paper: - An IoT-based Appliance Control System for Smart Home [ICICIP, 2013] implemented an IoT based appliance control system for smart homes that uses a smart-central controller to set up a wireless sensor and actuator network and control modules for appliances

Home Automation Intrusion Detection

- Home intrusion detection systems use security cameras and sensors to detect intrusions and raise alerts.
- The form of the alerts can be in form: - SMS - Email - Image grab or a short video clip as an email attachment
- Papers :

- Could controlled intrusion detection and burglary prevention stratagems in home automation systems [BCFIC, 2012] ->present a controlled intrusion detection system that uses location-aware services, where the geo-location of each node of a home automation system is independently detected and stored in the cloud

- An Intelligent Intrusion Detection System Based on UPnP Technology for Smart Living [ISDA, 2008] -> implement an intrusion detection system that uses image processing to recognize the intrusion and extract the intrusion subject and generate Universal-Plug-and-Play (UPnP-based) instant messaging for alerts.

Home Automation Smoke / Gas Detectors

- Smoke detectors are installed in homes and buildings to detect smoke that is typically an early sign of fire.
- It uses optical detection, ionization or air sampling techniques to detect smoke
- The form of the alert can be in form :
 - Signals that send to a fire alarm system
- Gas detector can detect the presence of harmful gases such as carbon monoxide (CO), liquid petroleum gas (LPG), etc.
- Paper :
 - Development of Multipurpose Gas Leakage and Fire Detector with Alarm System [TIIEC, 2013]-> designed a system that can detects gas leakage and smoke and gives visual level indication.

Cities IoT applications for smart cities:

1. Smart Parking
2. Smart Lighting for Road
3. Smart Road
4. Structural Health Monitoring
5. Surveillance
6. Emergency Response



Cities Smart Parking

- Finding the parking space in the crowded city can be time consuming and frustrating
- Smart parking makes the search for parking space easier and convenient for driver.
- It can detect the number of empty parking slots and send the information over the Internet to the smart parking applications which can be accessed by the drivers using their smart phones, tablets, and in car navigation systems.
- Sensors are used for each parking slot to detect whether the slot is empty or not, and this information is aggregated by local controller and then sent over the Internet to database.
- Paper :
 - Design and implementation of a prototype Smart Parking (SPARK) system using WSN [International Conference on Advanced Information Networking and Applications Workshop, 2009]-> designed and implemented a prototype smart parking system based on wireless sensor network technology with features like remote parking monitoring, automate guidance, and parking reservation mechanism.

Cities Smart Lighting for Roads

- It can help in saving energy
- Smart lighting for roads allows lighting to be dynamically controlled and also adaptive to ambient conditions.
- Smart light connected to the Internet can be controlled remotely to configure lighting schedules and lighting intensity.
- Custom lighting configurations can be set for different situations such as a foggy day, a festival, etc.
- Paper :
 - Smart Lighting solutions for Smart Cities [International Conference on Advance Information Networking and Applications Workshop, 2013]-> described the need for smart lighting system in smart cities, smart lighting features and how to develop interoperable smart lighting solutions.

Cities Smart Roads

- Smart Roads provides information on driving conditions, travel time estimates and alerts in case of poor driving conditions, traffic congestions and accidents.

- Such information can help in making the roads safer and help in reducing traffic jams
- Information sensed from the roads can be communicated via internet to cloud-based applications and social media and disseminated to the drivers who subscribe to such applications.
- Paper:
 - Sensor networks for smart roads [PerCom Workshop, 2006]-> proposed a distributed and autonomous system of sensor network nodes for improving driving safety on public roads, the system can provide the driver and passengers with a consistent view of the road situation a few hundred metes ahead of them or a few dozen miles away, so that they can react to potential dangers early enough.

Cities Structural Health Monitoring

- It uses a network of sensors to monitor the vibration levels in the structures such as bridges and buildings.
- The data collected from these sensors is analyzed to assess the health of the structures.
- By analyzing the data it is possible to detect cracks and mechanical breakdowns, locate the damages to a structure and also calculate the remaining life of the structure.
- Using such systems, advance warnings can be given in the case of imminent failure of the structure.
- Paper:
 - Environmental Effect Removal Based Structural Health Monitoring in the Internet of Things [International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, 2013]-> proposed an environmental effect removal based structural health monitoring scheme in an IoT environment.
 - Energy harvesting technologies for structural health monitoring applications [IEEE Conference on Technologies for Sustainability, 2013] -> Explored energy harvesting technologies of harvesting ambient energy, such as mechanical vibrations, sunlight, and wind.◊

Cities Surveillance

- Surveillance of infrastructure, public transport and events in cities is required to ensure safety and security.
- City wide surveillance infrastructure comprising of large number of distributed and Internet connected video surveillance cameras can be created.
- The video feeds from surveillance cameras can be aggregated in cloud-based scalable storage solutions.
- Cloud-based video analytics applications can be developed to search for patterns of specific events from the video feeds.

Cities Emergency Response

- IoT systems can be used for monitoring the critical infrastructure cities such as buildings, gas, and water pipelines, public transport and power substations.
- IoT systems for critical infrastructure monitoring enable aggregation and sharing of information collected from larger number of sensors.
- Using cloud-based architectures, multi-modal information such as sensor data, audio, video feeds can be analyzed near real-time to detect adverse events.
- The alert can be in the form :
 - Alerts sent to the public
 - Re-rerouting of traffic
 - Evacuations of the affected areas

Environment IoT applications for smart environments:

1. Weather Monitoring
2. Air Pollution Monitoring
3. Noise Pollution Monitoring
4. Forest Fire Detection
5. River Flood Detection

Environment Weather Monitoring

- It collects data from a number of sensor attached such as temperature, humidity, pressure, etc and send the data to cloud-based applications and store back-ends.
- The data collected in the cloud can then be analyzed and visualized by cloud-based applications.

- Weather alert can be sent to the subscribed users from such applications.
- AirPi is a weather and air quality monitoring kit capable of recording and uploading information about temperature, humidity, air pressure, light levels, UV levels, carbon monoxide, nitrogen dioxide and smoke level to the Internet.
- Paper:
 - PeWeMoS – Pervasive Weather Monitoring System [ICPCA, 2008]-> Presented a pervasive weather monitoring system that is integrated with buses to measure weather variables like humidity, temperature, and air quality during the bus path

Environment Air Pollution Monitoring

- IoT based air pollution monitoring system can monitor emission of harmful gases by factories and automobiles using gaseous and meteorological sensors.
- The collected data can be analyzed to make informed decisions on pollution control approaches.
- Paper: - Wireless sensor network for real-time air pollution monitorings [ICCSWA, 2013]-> Presented a real time air quality monitoring system that comprises of several distributed monitoring stations that communicate via wireless with a back-end server using machine-to machine communication.

Environment Noise Pollution Monitoring

- Noise pollution monitoring can help in generating noise maps for cities.
- It can help the policy maker in making policies to control noise levels near residential areas, school and parks.
- It uses a number of noise monitoring stations that are deployed at different places in a city.
- The data on noise levels from the stations is collected on servers or in the cloud and then the collected data is aggregated to generate noise maps.
- Papers :
 - Noise mapping in urban environments : Applications at Suez city center [ICCIE, 2009]Presented a noise mapping study for a city which revealed that the city suffered from serious noise pollution.
 - SoundOfCity – Continuous noise monitoring for a health city [PerComW,2013]-> Designed a smartphone application that allows the users to continuously measure noise levels and send to a central server here all generated information is aggregated and mapped to a meaningful noise visualization map.

Environment Forest Fire Detection

- IoT based forest fire detection system use a number of monitoring nodes deployed at different location in a forest.
- Each monitoring node collects measurements on ambient condition including temperature, humidity, light levels, etc.
- Early detection of forest fires can help in minimizing the damage.
- Papers:
 - A novel accurate forest fire detection system using wireless sensor networks [International Conference on Mobile Ad-hoc and Sensor Networks, 2011]-> Presented a forest fire detection system based on wireless sensor network. The system uses multi-criteria detection which is implemented by the artificial neural network. The ANN fuses sensing data corresponding to ,multiple attributes of a forest fire such as temperature, humidity, infrared and visible light to detect forest fires.

Environment River Flood Detection

- IoT based river flood monitoring system uses a number of sensor nodes that monitor the water level using ultrasonic sensors and flow rate using velocity sensors.
- Data from these sensors is aggregated in a server or in the cloud, monitoring applications raise alerts when rapid increase in water level and flow rate is detected.
- Papers:

- RFMS : Real time flood monitoring system with wireless sensor networks [MASS, 2008]-> Described a river flood monitoring system that measures river and weather conditions through wireless sensor nodes equipped with different sensors
- Urban Flash Flood Monitoring, Mapping and Forecasting via a Tailored Sensor Network System [ICNSC, 2006] -> Described a motes-based sensor network for river flood monitoring that includes a water level monitoring module, network video recorder module, and data processing module that provides floods information in the form of raw data, predict data, and video feed.

Energy IoT applications for smart energy systems:

1. Smart Grid
2. Renewable Energy Systems
3. Prognostics



Energy Smart Grids

- Smart grid technology provides predictive information and recommendations to utilize their suppliers, and their customers on how best to manage power.
- Smart grid collects the data regarding :
 - Electricity generation
 - Electricity consumption
 - Storage
 - Distribution and equipment health data
- By analyzing the data on power generation, transmission and consumption of smart grids can improve efficiency throughout the electric system.
- Storage collection and analysis of smart grids data in the cloud can help in dynamic optimization of system operations, maintenance, and planning.
- Cloud-based monitoring of smart grids data can improve energy usage levels via energy feedback to users coupled with real-time pricing information.
- Condition monitoring data collected from power generation and transmission systems can help in detecting faults and predicting outages.

Energy Renewable Energy System

- Due to the variability in the output from renewable energy sources (such as solar and wind), integrating them into the grid can cause grid stability and reliability problems.
- IoT based systems integrated with the transformer at the point of interconnection measure the electrical variables and how much power is fed into the grid
- To ensure the grid stability, one solution is to simply cut off the overproductions.
- Paper:
 - Communication systems for grid integration of renewable energy resources [IEEE Network, 2011]-> Provided the closed-loop controls for wind energy system that can be used to regulate the voltage at point of interconnection which coordinate wind turbine outputs and provides reactive power support.

Energy Prognostics

- IoT based prognostic real-time health management systems can predict performance of machines of energy systems by analyzing the extent of deviation of a system from its normal operating profiles.
- In the system such as power grids, real time information is collected using specialized electrical sensors called Phasor Measurement Units (PMU)
- Analyzing massive amounts of maintenance data collected from sensors in energy systems and equipment can provide predictions for impending failures.
- OpenPDC is a set of applications for processing of streaming time-series data collected from Phasor Measurements Units (PMUs) in real-time.

Retail IoT applications in smart retail systems:

1. Inventory Management
2. Smart Payments
3. Smart Vending Machines

Retail Inventory Management

- IoT system using Radio Frequency Identification (RFID) tags can help inventory management and maintaining the right inventory levels.
- RFID tags attached to the products allow them to be tracked in the real-time so that the inventory levels can be determined accurately and products which are low on stock can be replenished.
- Tracking can be done using RFID readers attached to the retail store shelves or in the warehouse.
- Paper:
 - RFID data-based inventory management of time-sensitive materials [IECON, 2005]-> described an RFID data-based inventory management system for time-sensitive materials

Retail Smart Payments

- Smart payments solutions such as contact-less payments powered technologies such as Near field communication (NFC) and Bluetooth.
- NFC is a set of standards for smart-phones and other devices to communicate with each other by bringing them into proximity or by touching them
- Customer can store the credit card information in their NFC-enabled smart-phones and make payments by bringing the smart-phone near the point of sale terminals.
- NFC maybe used in combination with Bluetooth, where NFC initiates initial pairing of devices to establish a Bluetooth connection while the actual data transfer takes place over Bluetooth.

Retail Smart Vending Machines

- Smart vending machines connected to the Internet allow remote monitoring of inventory levels, elastic pricing of products, promotions, and contact-less payments using NFC.
- Smart-phone applications that communicate with smart vending machines allow user preferences to be remembered and learned with time. E.g: when a user moves from one vending machine to the other and pair the smart-phone, the user preference and favourite product will be saved and then that data is used for predictive maintenance.
- Smart vending machines can communicated each others, so if a product out of stock in a machine, the user can be routed to nearest machine
- For perishable items, the smart vending machines can reduce the price as the expiry date nears.

Logistic IoT applications for smart logistic systems:

1. Fleet Tracking
2. Shipment Monitoring
3. Remote Vehicle Diagnostics

Logistics Fleet Tracking

- Vehicle fleet tracking systems use GPS technology to track the locations of the vehicles in the real- time.
- Cloud-based fleet tracking systems can be scaled up on demand to handle large number of vehicles,
- The vehicle locations and routers data can be aggregated and analyzed for detecting bottlenecks in the supply chain such as traffic congestions on routes, assignments and generation of alternative routes, and supply chain optimization
- Paper:
 - A Fleet Monitoring System for Advanced Tracking of commercial Vehicles [IEEE International Conference in Systems, Man and Cybernetics, 2006]-> provided a system that can analyze messages sent from the vehicles to identify unexpected incidents and discrepancies between actual and planned data, so that remedial actions can be taken.

Logistics Shipment Monitoring

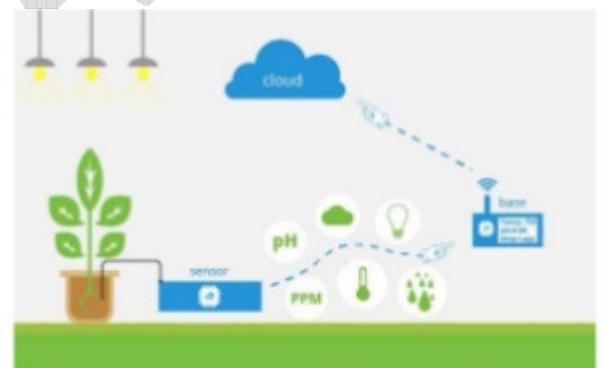
- Shipment monitoring solutions for transportation systems allow monitoring the conditions inside containers.
- E.g : Containers carrying fresh food produce can be monitored to prevent spoilage of food. IoT based shipment monitoring systems use sensors such as temperature, pressure, humidity, for instance, to monitor the conditions inside the containers and send the data to the cloud, where it can be analyzed to detect food spoilage.
- Paper:
 - On a Cloud-Based Information Technology Framework for Data Driven Intelligent Transportation System [Journal of Transportation Technologies, 2013]-> proposed a cloud based framework for real time fresh food supply tracking and monitoring
 - Container Integrity and Condition Monitoring using RF Vibration Sensor Tags [IEEE International Conference on Automation Science and Engineering, 2007] ◊ Proposed a system that can monitor the vibrations patterns of a container and its contents to reveal information related to its operating environment and integrity during transport, handling, and storage.

Logistics Remote Vehicle Diagnostics

- It can detect faults in the vehicles or warn of impending faults.
- These diagnostic systems use on-board IoT devices for collecting data on vehicle operation such as speed, engine RPM, coolant temperature, fault code number and status of various vehicle sub- system.
- Modern commercial vehicles support on-board diagnostic (OBD) standard such as OBD-II
- OBD systems provide real-time data on the status of vehicle sub-systems and diagnostic trouble codes which allow rapidly identifying the faults in the vehicle.
- IoT based vehicle diagnostic systems can send the vehicle data to centralized servers or the cloud where it can be analyzed to generate alerts and suggest remedial actions.

Agriculture IoT applications for smart agriculture:

1. Smart Irrigation
2. Green House Control



Agriculture Smart Irrigation

- Smart irrigation system can improve crop yields while saving water.
- Smart irrigation systems use IoT devices with soil moisture sensors to determined the amount of moisture on the soil and release the flow of the water through the irrigation pipes only when the moisture levels go below a predefined threshold.
- It also collect moisture level measurements on the server on in the cloud where the collected data can be analyzed to plan watering schedules.
- Cultivar's RainCloud is a device for smart irrigation that uses water valves, soil sensors, and a WiFi enabled programmable computer. [<http://ecultivar.com/rain-cloud-product-project/>]

Agriculture Green House Control

- It controls temperature, humidity, soil, moisture, light, and carbon dioxide level that are monitored by sensors and climatological conditions that are controlled automatically using actuation devices.
- IoT systems play an importance role in green house control and help in improving productivity.
- The data collected from various sensors is stored on centralized servers or in the cloud where analysis is performed to optimize the control strategies and also correlate the productivity with different control strategies.
- Paper: - Wireless sensing and control for precision Green house management [ICST, 2012]-> Provided a system that uses wireless sensor network to monitor and control the agricultural parameters like

temperature and humidity in the real time for better management and maintenance of agricultural production.

Industry IoT applications in smart industry:

1. Machine Diagnosis & Prognosis
2. Indoor Air Quality Monitoring

Industry Machine Diagnosis & Prognosis

- Machine prognosis refers to predicting the performance of machine by analyzing the data on the current operating conditions and how much deviations exist from the normal operating condition.
- Machine diagnosis refers to determining the cause of a machine fault.
- Sensors in machine can monitor the operating conditions such as temperature and vibration levels, sensor data measurements are done on timescales of few milliseconds to few seconds which leads to generation of massive amount of data.
- Case-based reasoning (CBR) is a commonly used method that finds solutions to new problems based on past experience.
- CBR is an effective technique for problem solving in the fields in which it is hard to establish a quantitative mathematical model, such as machine diagnosis and prognosis.

Industry Indoor Air Quality Monitoring

- Harmful and toxic gases such as carbon monoxide (CO), nitrogen monoxide (NO), Nitrogen Dioxide, etc can cause serious health problem of the workers.
- IoT based gas monitoring systems can help in monitoring the indoor air quality using various gas sensors. - The indoor air quality can be placed for different locations
- Wireless sensor networks based IoT devices can identify the hazardous zones, so that corrective measures can be taken to ensure proper ventilation.
- Papers:
 - A hybrid sensor system for indoor air quality monitoring [IEEE International Conference on Distributed Computing in Sensor System, 2013]-> presented a hybrid sensor system for indoor air quality monitoring which contains both stationary sensor and mobile sensors.
 - Indoor air quality monitoring using wireless sensor network [International Conference on Sensing Technology, 2012] -> provided a wireless solution for indoor air quality monitoring that measures the environmental parameters like temperature, humidity, gaseous pollutants , aerosol and particulate matter to determine the indoor air quality.◊

Health & Lifestyle IoT applications in smart health & lifestyle:

1. Health & Fitness Monitoring
2. Wearable Electronics

Fitness Monitoring

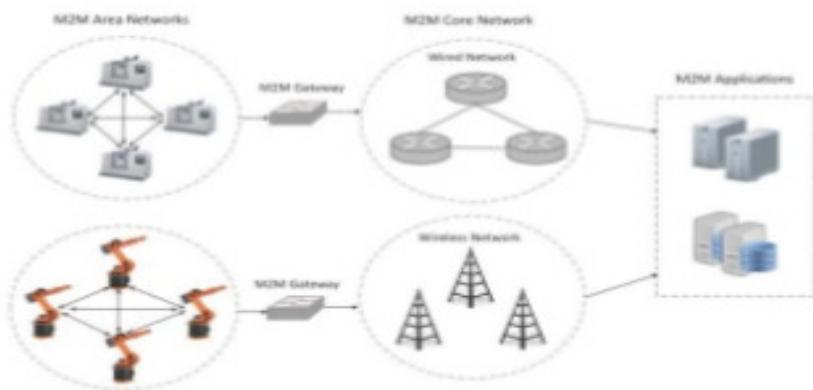
- Wearable IoT devices allow to continuous monitoring of physiological parameters such as blood pressure, heart rate, body temperature, etc than can help in continuous health and fitness monitoring.
- It can analyze the collected health-care data to determine any health conditions or anomalies.
- The wearable devices may can be in various form such as:
 - Belts
 - Wrist-bands
- Papers:
 - Toward ubiquitous mobility solutions for body sensor network health care [IEEE Communications Magazine, 2012]-> Proposed an ubiquitous mobility approach for body sensor network in health-care
 - A wireless sensor network compatible wearable u-healthcare monitoring system using integrated ECG, accelerometer and SpO2 [International Conference of the IEEE Engineering in Medicine and Biology Society, 2008]-> Health & Lifestyle Health & Designed a wearable ubiquitous health-care monitoring system that uses integrated electrocardiogram (ECG), accelerometer and oxygen saturation (SpO2) sensors.◊

Health & Lifestyle Wearable Electronics

- Wearable electronics such as wearable gadgets (smart watch, smart glasses, wristbands, etc) provide various functions and features to assist us in our daily activities and making us lead healthy lifestyles.
- Using the smart watch, the users can search the internet, play audio/video files, make calls, play games, etc.
- Smart glasses allows users to take photos and record videos, get map directions, check flight status or search internet using voice commands
- Smart shoes can monitor the walking or running speeds and jumps with the help of embedded sensors and be paired with smart-phone to visualize the data.
- Smart wristbands can track the daily exercise and calories burnt.

M2M

Machine-to-Machine (M2M) refers to the communication or exchange of data between two or more machines without human interfacing or interaction. The communication in M2M may be wired or wireless systems. The M2M uses a device such as sensor, RFID, meter, etc. to capture an 'event' like temperature, inventory level, etc., which is relayed through a network i.e., wireless, wired or hybrid to an application (software program), that translates the captured event into meaningful information. Unlike SCADA or other remote monitoring tools, M2M systems often use public networks and access methods -- for example, cellular or Ethernet -- to make it more cost-effective.



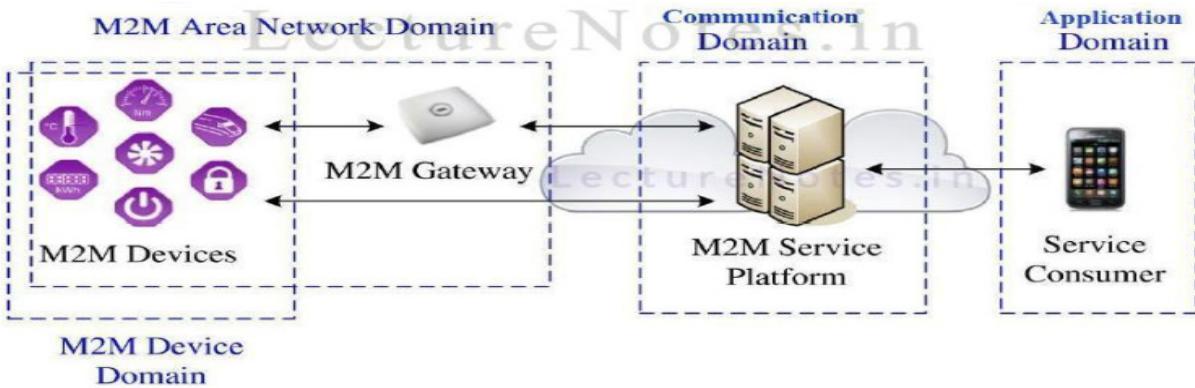
Machine-to-Machine (M2M)

- An M2M area network comprises of machines (or M2M nodes) which have embedded hardware modules for sensing, actuation and communication.
- Various communication protocols can be used for M2M local area networks such as ZigBee, Bluetooth, ModBus, M-Bus, Wireless M-Bus, Power Line Communication (PLC), 6LoWPAN, IEEE 802.15.4, etc.
- The communication network provides connectivity to remote M2M area networks.
- The communication network can use either wired or wireless networks (IP-based).
- While the M2M area networks use either proprietary or non-IP based communication protocols, the communication network uses IP-based networks.

M2M System Architecture

The main components of M2M system are:

1. M2M area networks
2. Communication networks
3. Application domains
4. M2M gateways.

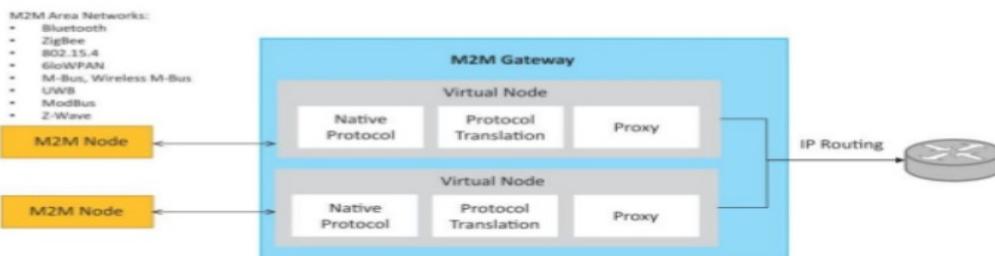


M2M area networks

- M2M network area consists of machines or M2M nodes which communicate with each other. The M2M nodes embedded with hardware modules such as sensors, actuators and communication devices.
- M2M uses communication protocol such as ZigBee, Bluetooth, Power line communication (PLC) etc. These communication protocols provide connectivity between M2M nodes within M2M area network.
- The M2M nodes communicate with in one network it can't communicate with external network node. To enable the communication between remote M2M area networks, M2M gateways are used.

M2M Gateways

- The Gateway module provides control and localization services for data collection. The gateways also double up in concentrating traffic to the operator's core. It supports Bluetooth, ZigBee, GPRS capabilities.
- M2M communication network serves as infrastructure for realizing communication between M2M gateway and M2M end user application or server. For this cellular network (GSM /CDMA), Wire line network and communication satellites may be used.



Communication networks

- The communication network provides the connectivity between M2M nodes and M2M applications.
- It uses wired or wireless network such as LAN, LTE, WiMAX, satellite communication etc.

Application domains

- It contains the middleware layer where data goes through various application services and is used by the specific business-processing engines.
- M2M applications will be based on the infrastructural assets that are provided by the operator. Applications may either target at end users, such as user of a specific M2M solution, or at other application providers to offer more refined building blocks by which they can build more sophisticated M2M solutions and services.

Difference between IoT and M2M

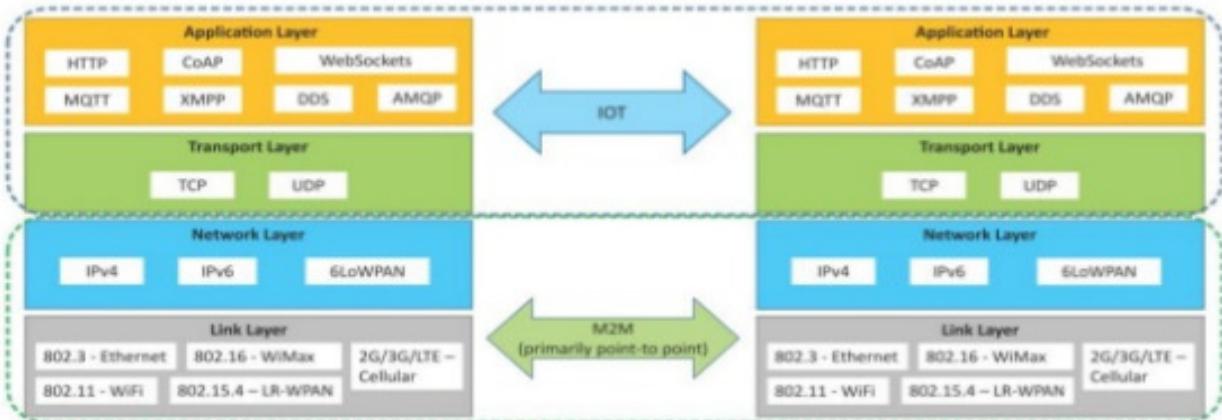
- Communication Protocols
 - M2M and IoT can differ in how the communication between the machines or devices happens.
 - M2M uses either proprietary or non-IP based communication protocols for communication within the M2M area networks.
- Machines in M2M vs Things in IoT

- The "Things" in IoT refers to physical objects that have unique identifiers and can sense and communicate with their external environment (and user applications) or their internal physical states.
- M2M systems, in contrast to IoT, typically have homogeneous machine types within an M2M area network.
- Hardware vs Software Emphasis
 - While the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software.
- Data Collection & Analysis
 - M2M data is collected in point solutions and often in on-premises storage infrastructure.
 - In contrast to M2M, the data in IoT is collected in the cloud (can be public, private or hybrid cloud).
- Applications
 - M2M data is collected in point solutions and can be accessed by on-premises applications such as diagnosis applications, service management applications, and on-premises enterprise applications.
 - IoT data is collected in the cloud and can be accessed by cloud applications such as analytics applications, enterprise applications, remote diagnosis and management applications, etc.

M2M and IoT both are used when electronic devices are connected and share data with each other. There are some differences between IoT and M2M based on technologies, system architectures and types of applications

M2M	IoT
It is Machine to Machine communication and completely hardware based.	It's Machine to Machine, Machine to sensors, or Humans to Machines. And software based.
M2M is a point to point communication and uses non-IP protocols.	Its uses IP networks and protocols as the communication is multipoint.
These devices don't rely on internet.	Devices required internet connections.
Data can be stored locally	Data can be stored locally and also in cloud
Limited integration option devices must have corresponding communication standards	Unlimited integration option, but requires a solution that can manage all the communication

Communication in IoT vs M2M



Software-Defined Networking (SDN)

The Conventional Networking Technologies has lot of Limitations with increases in number of distributed protocols these limitations are as:

- **Complex Network Devices:-**

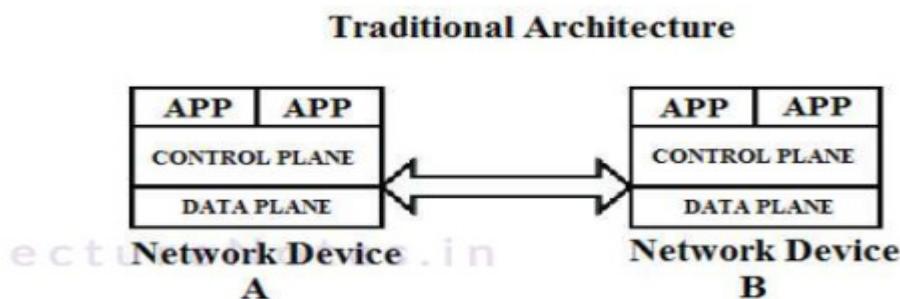
To meet business and technical needs over the last few decades, the industry has evolved networking protocols to deliver higher performance, high speeds and reliability which increases more number of protocols making the device complex. Due to complexity networks are made relatively static to minimize the risk of service disruption.

- **Management Overhead:-**

Network managers find it difficult to manage multiple network devices and interfaces from multiple vendors. Upgradation of network requires configuration changes in multiple devices such as switches, routers, firewalls, etc.

- **Limited Scalability:-**

The virtualization technologies (creation of several virtual machine using software called as hypervisors) used in cloud computing environments has increased the number of virtual host in the cloud. The analytics components of IoT applications exchange huge amount of data in virtual machines which require highly scalable which becomes difficult with conventional networks.



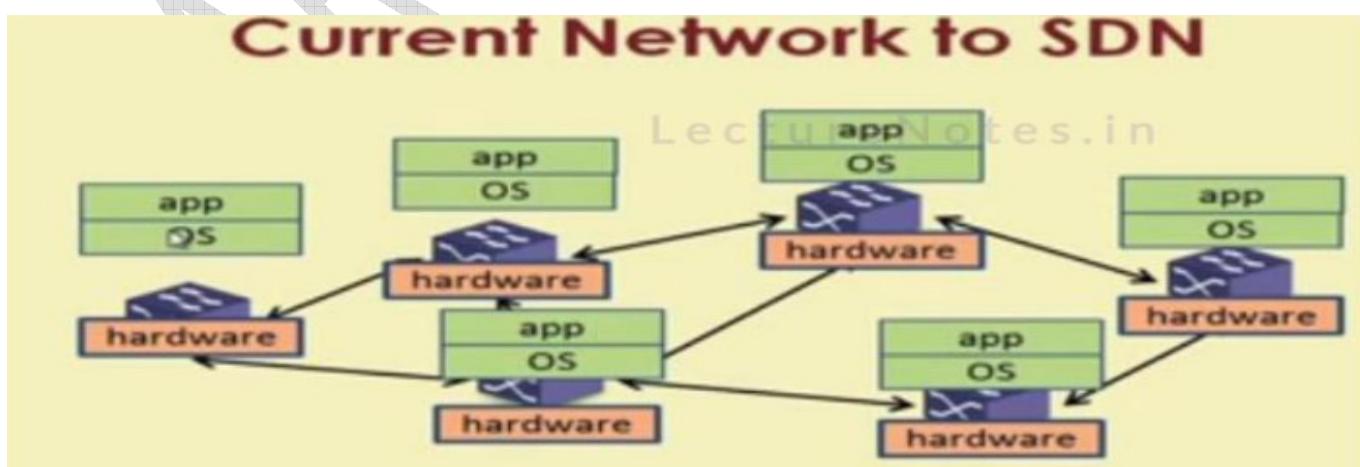
This mismatch between market requirements and network capabilities has brought the industry to a tipping point. In response, the industry has created the Software-Defined Networking (SDN) architecture and is developing associated standards.

Software-Defined Networking (SDN)

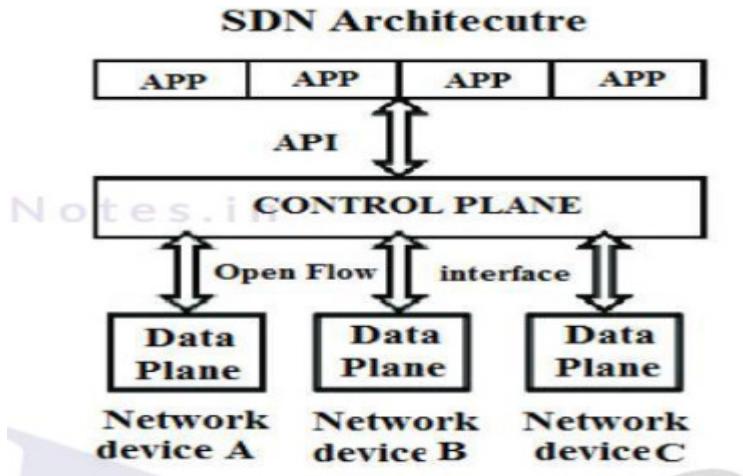
SDN is defined as the physical separation of the networking architecture of control plane from the data plane, and centralizes the network controller.

Basic concepts of SDN

- Separate control logic from hardware switches
- Define the control logic in a centralized manner
- Control the entire network including individual switches
- Communication between the application, control, and data planes are done through APIs.



SDN Architecture



Key Components of SDN

- Centralized Network controller
- Programmable open APIs
- Standard communication interface (Open Flow)

Centralized Network controller

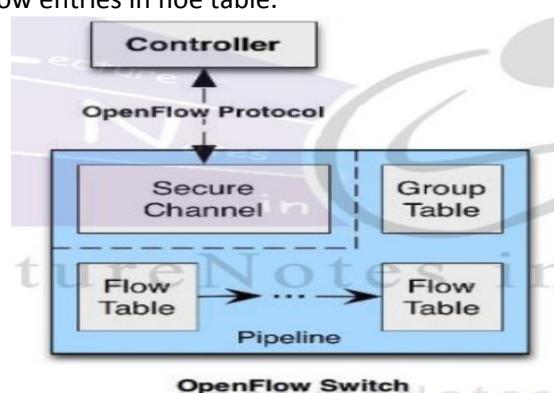
- With separated control plan, data plane and centralized network controller, the network administrator can rapidly configure the network.
- SDN application can be deployed through programmable APIs which speeds up innovation as the network administrator no longer need to wait for the device vendors to embed new features in hardware.

Programmable open APIs

- SDN architecture supports Programmable open APIs for interface between the SDN application and control layers (Northbound interface)
- Northbound APIs:
 - Software-Defined Networking uses northbound APIs to communicate with the applications and business logic “above.” These help network administrators to programmatically shape traffic and deploy services.

Standard communication interface (Open Flow)

- Standard communication interface between control layer and infrastructure layer
- Southbound APIs:
 - Software-defined networking uses southbound APIs to relay information to the switches and routers “below”.
 - Open Flow, considered the first standard in SDN, was the original southbound API and remains as one of the most common protocols
 - Open flow protocol the network device can be directly accessed and manipulated. It uses the concepts of flows to identify network traffic based on pre-defined matched rules.
 - The controller manages the switch via open flow switch protocol where controller can add, update and delete flow entries in floe table.



Network Function Virtualization (NFV)

- Network functions virtualization (NFV) is the concept of replacing dedicated network appliances such as routers and firewalls with software running on general-purpose CPUs or virtual machines, operating on standard servers.
- NFV provides the infrastructure on which SDN can run. NFV and SDN are mutually beneficial to each other but not dependent.
- NFV replaces proprietary hardware network elements (NEs) with software running on standard servers, SDN deals with the replacement of standardized networking protocols with centralized control
- The key elements of NFV architecture are:

1. NFV infrastructure (NFVI):

NFV infrastructure layer consists of different layers such as hardware resources like computing resources (RAM, servers), storage resources (hard-disc), and network resources (routers, switch, and firewalls). Second layer is Virtualization layer which separates hardware and replaces it with software and third layer is virtualized resources such as virtual compute, network and storage

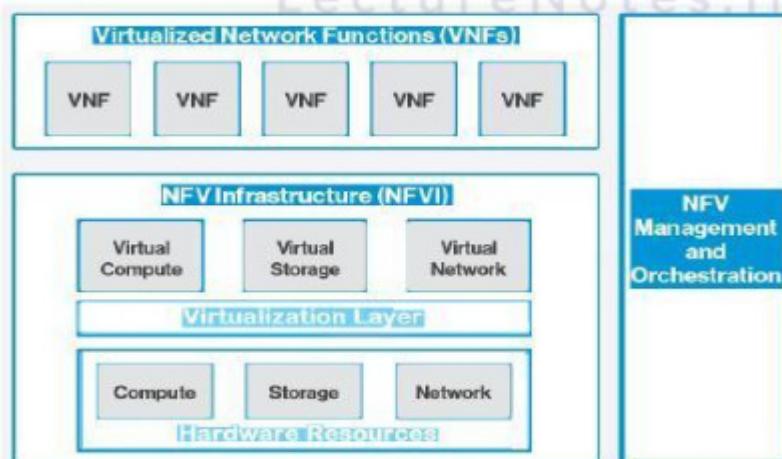
2. Virtualized network function (VNF):

VNF is a software implementation of a network function which is capable of running over the NFV infrastructure (NFVI). Example: - vFirewall, vRouters

3. NFV management and orchestration:

It has three parts

- a. Virtualized infrastructure manager: - it controls and manages network functions with NFVI resources and monitors virtualization layer.
- b. VNF manager: - it manages the life cycle of VNF such as initialize, update, query, scale, terminate etc.
- c. Orchestrator: - it manages the life cycle of network services which includes policy management, performance measurement and monitoring.



Need For IoT Systems Management

- Automating Configuration
- Monitoring Operational & Statistical Data
- Improved Reliability
- System Wide Configurations
- Multiple System Configurations
- Retrieving & Reusing

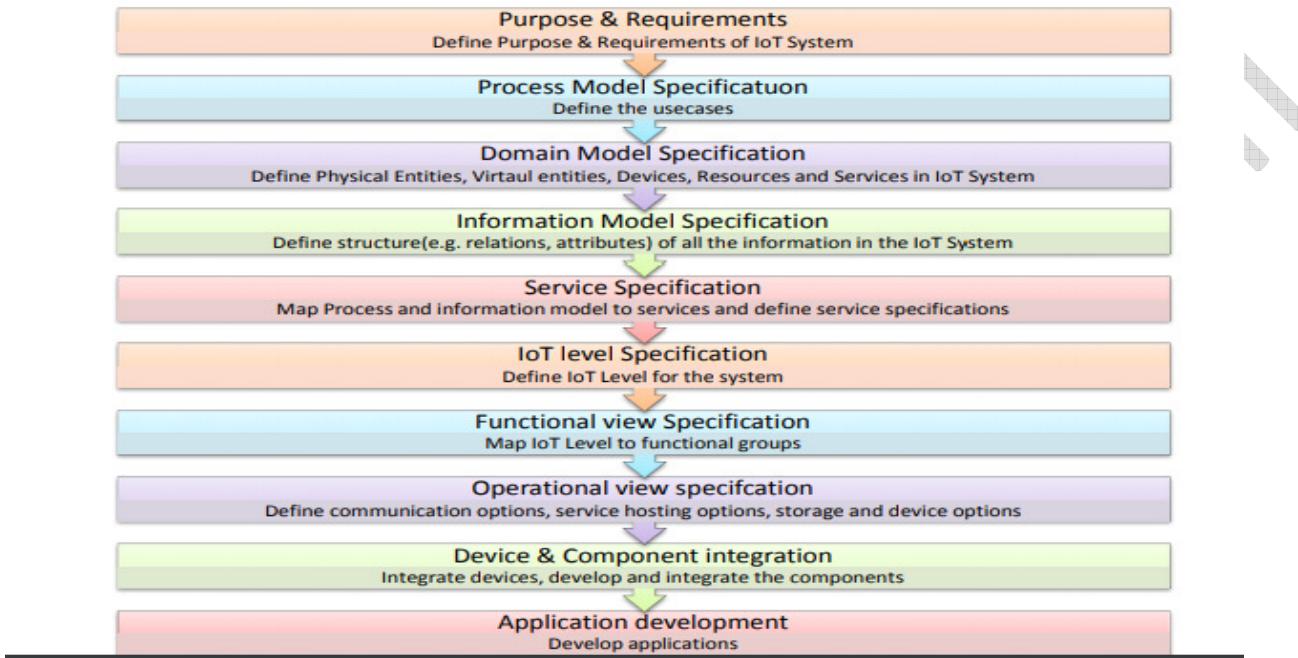
MODULE-3

IoT Design Methodology that includes:

- Purpose & Requirements Specification
- Process Specification
- Domain Model Specification
- Information Model Specification

- Service Specifications
- IoT Level Specification
- Functional View Specification
- Operational View Specification
- Device & Component Integration
- Application Development

Steps involved in IoT System Design Methodology



Advantages of Using Design methodology

- Reducing the design, testing and maintenance time
- Provide better interoperability
- Reduce the complexity

Step 1 : Purpose and Requirement Specification

Defines

- System purpose
- behavior and
- Requirements (such as data collection requirements, data analysis requirement, system management requirements, data privacy and security requirements, User interfaces requirements)
- **Purpose** : An automated irrigation mechanism which turns the pumping motor ON and OFF on detecting the moisture content of the earth without the intervention of human
- **Behavior** : System should monitor the amount of soil moisture content in soil. In case the soil moisture of the soil deviates from the specified range, the watering system is turned ON/OFF. In case of dry soil, it will activate the irrigation system, pumping water for watering the plants.
- **System Management Requirements** : system should remotely provide monitoring and control functions
- **Data Analysis Requirements** : system should perform local analysis of data
- **Application Deployment Requirement** : Deployed locally on device, but acts remotely without manual intervention.
- **Security** : Authentication to Use the system must be available

Step 1: Purpose & Requirements Specification

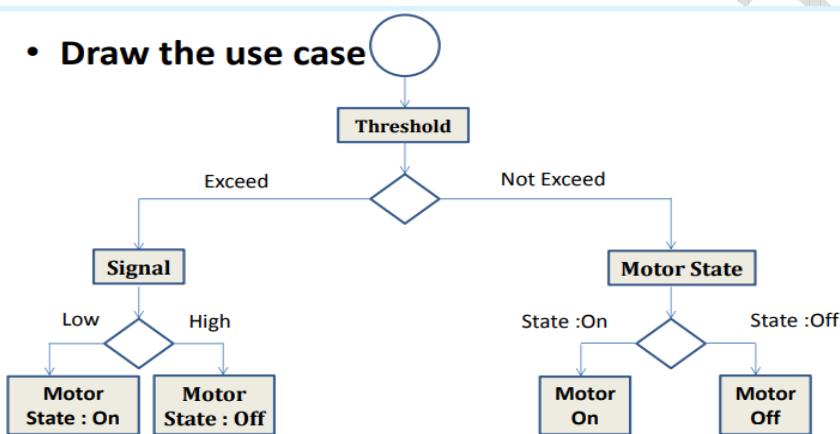
- The first step in IoT system design methodology is to define the purpose and requirements of the system. In this step, the system purpose, behavior and requirements (such as data collection requirements, data analysis requirements, system management requirements, data privacy and security requirements, user interface requirements, ...) are captured

Step 2 : Process Specification

- Define the process with the help of use cases
- The use cases are formally described based on Purpose & requirement specification In this
- use case : – Circle denotes a state or an attribute

Step 2: Process Specification

- The second step in the IoT design methodology is to define the process specification. In this step, the use cases of the IoT system are formally described based on and derived from the purpose and requirement specifications.



Step 3 : Domain Model Specification

- Describes the main concepts, entities and objects in the domain of IoT system to be designed
- It defines the attributes of the objects and relationships between them
- Entities , Objects and Concepts include the following : Physical entity, Virtual entity , Device, Resource, Service
- Physical Entity:**
 - Discreet identifiable entity in physical environment
 - For eg. Pump, motor, LCD
 - The IoT System provides the information about the physical entity (using sensors) or performs actuation upon the Physical entity(like switching a motor on etc.)
 - In smart irrigation example, there are three Physical entities involved :
 - Soil (whose moisture content is to be monitored)
 - Motor (to be controlled)
 - Pump (To be controlled)
- Virtual Entity:**
 - Representation of physical entity in digital world
 - For each physical entity there is a virtual entity
- Device:**
 - Medium for interactions between Physical and Virtual Entities.
 - Devices (Sensors) are used to gather information from the physical entities
 - Devices are used to identify Physical entities (Using Tags)
 - In Smart Irrigation System, device is soil moisture sensor and buzzer as well as the actuator (relay switch) attached to it.

- In smart irrigation system there are three services :
 - A service that sets the signal to low/ high depending upon the threshold value
 - A service that sets the motor state on/off
 - A controller service that runs and monitors the threshold value of the moisture and switches the state of motor on/off depending upon it. When threshold value is not crossed the controller retrieves the motor status from database and switches the motor on/off.

Step 3: Domain Model Specification

- The third step in the IoT design methodology is to define the Domain Model. The domain model describes the main concepts, entities and objects in the domain of IoT system to be designed. Domain model defines the attributes of the objects and relationships between objects. Domain model provides an abstract representation of the concepts, objects and entities in the IoT domain, independent of any specific technology or platform. With the domain model, the IoT system designers can get an understanding of the IoT domain for which the system is to be designed.

Step 4 : Information Model Specification

- Defines the structure of all the information in the IoT system (such as attributes, relations etc.)
- It does not describe the specifics of how the information is represented or stored.
- This adds more information to the Virtual entities by defining their attributes and relations
- I: e, Draw Class diagram

Step 4: Information Model Specification

- The fourth step in the IoT design methodology is to define the Information Model. Information Model defines the structure of all the information in the IoT system, for example, attributes of Virtual Entities, relations, etc. Information model does not describe the specifics of how the information is represented or stored. To define the information model, we first list the Virtual Entities defined in the Domain Model. Information model adds more details to the Virtual Entities by defining their attributes and relations.

Step 5 : Service Specification

- Define the services in IoT System, service types, service inputs/outputs, service endpoints, service schedules, service preconditions and service effects
- Services can be controller service, Threshold service, state service for smart irrigation system
- These services either change the state/attribute values or retrieve the current values.
- For eg.
 - Threshold service sets signal to high or low depending upon the soil moisture value.
 - State service sets the motor state : on or off
 - Controller service monitors the threshold value as well as the motor state and switches the motor on/off and updates the status in the database

Step 5: Service Specifications

- The fifth step in the IoT design methodology is to define the service specifications. Service specifications define the services in the IoT system, service types, service inputs/output, service endpoints, service schedules, service preconditions and service effects.

Step 6 : IoT Level Specification

- Decide the deployment level of IoT System. Here I am using Deployment Level 1.

Step 6: IoT Level Specification

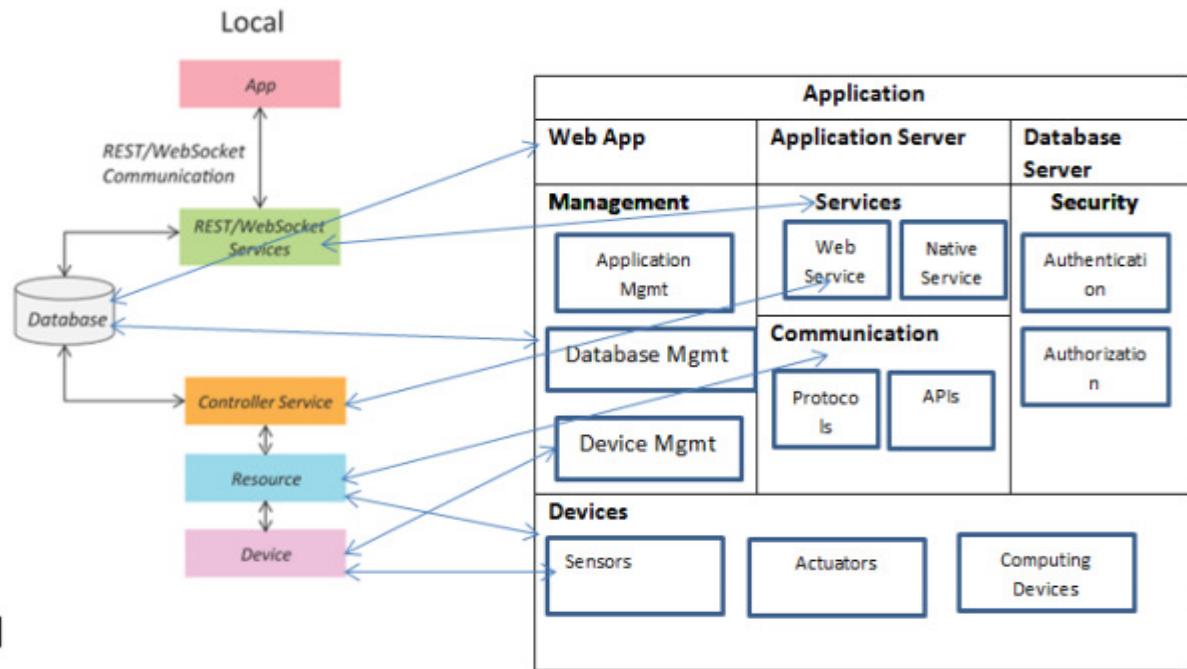
- The sixth step in the IoT design methodology is to define the IoT level for the system. In Chapter-1, we defined five IoT deployment levels.

Step 7 : Functional View Specification

- Define the functions of IoT System grouped into various functional groups.
- These functional groups provide functionalities for interacting with the concepts defined in Domain model specification.

Step 7: Functional View Specification

- The seventh step in the IoT design methodology is to define the Functional View. The Functional View (FV) defines the functions of the IoT systems grouped into various Functional Groups (FGs). Each Functional Group either provides functionalities for interacting with instances of concepts defined in the Domain Model or provides information related to these concepts.



Step 8 : Operational View Specification

- Define the Operations/options related to IoT System development
- Such as Device options, Storage options, Application hosting option

Step 8: Operational View Specification

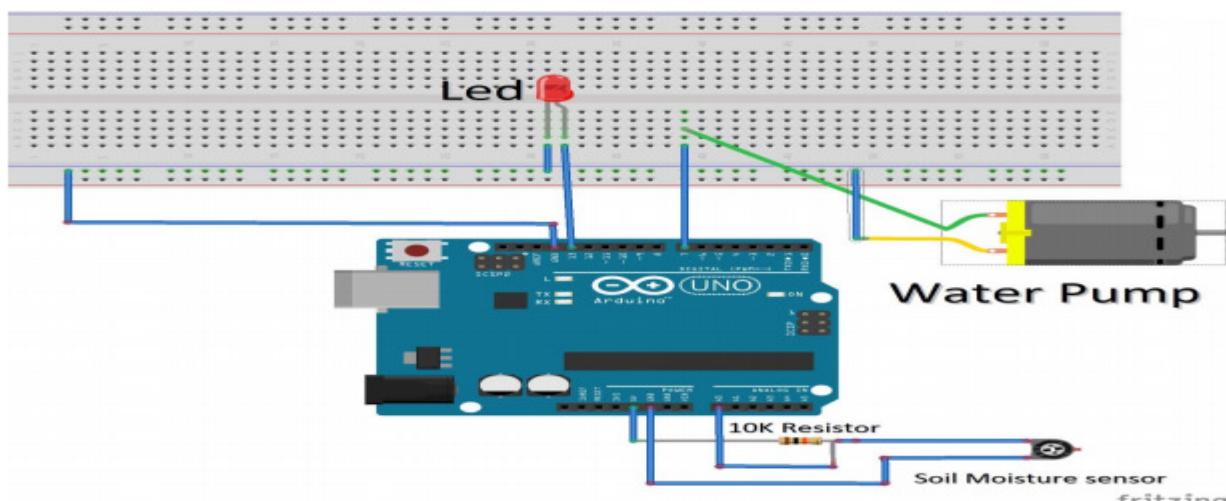
- The eighth step in the IoT design methodology is to define the Operational View Specifications. In this step, various options pertaining to the IoT system deployment and operation are defined, such as, service hosting options, storage options, device options, application hosting options, etc

Step 9 : Device and Component Integration

- Integrates the devices and components and draw a schematic diagram showing the same

Step 9: Device & Component Integration

- The ninth step in the IoT design methodology is the integration of the devices and components.



Step 10 : Application development

- GUI / Screenshot of IoT Application

Step 10: Application Development

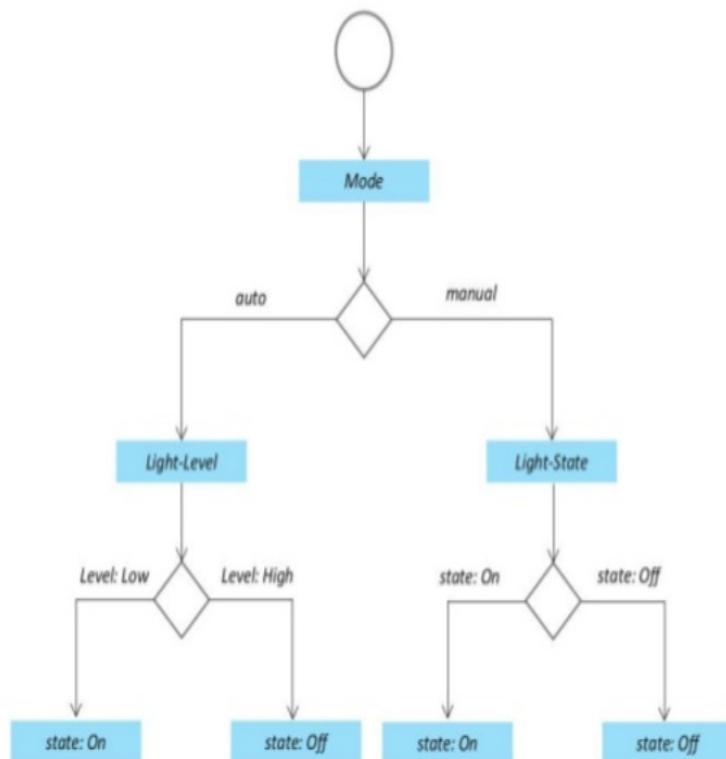
- The final step in the IoT design methodology is to develop the IoT application.

Home Automation Case Study

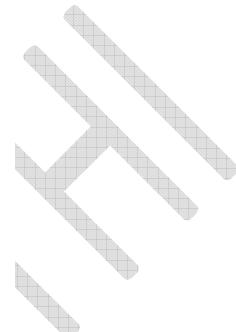
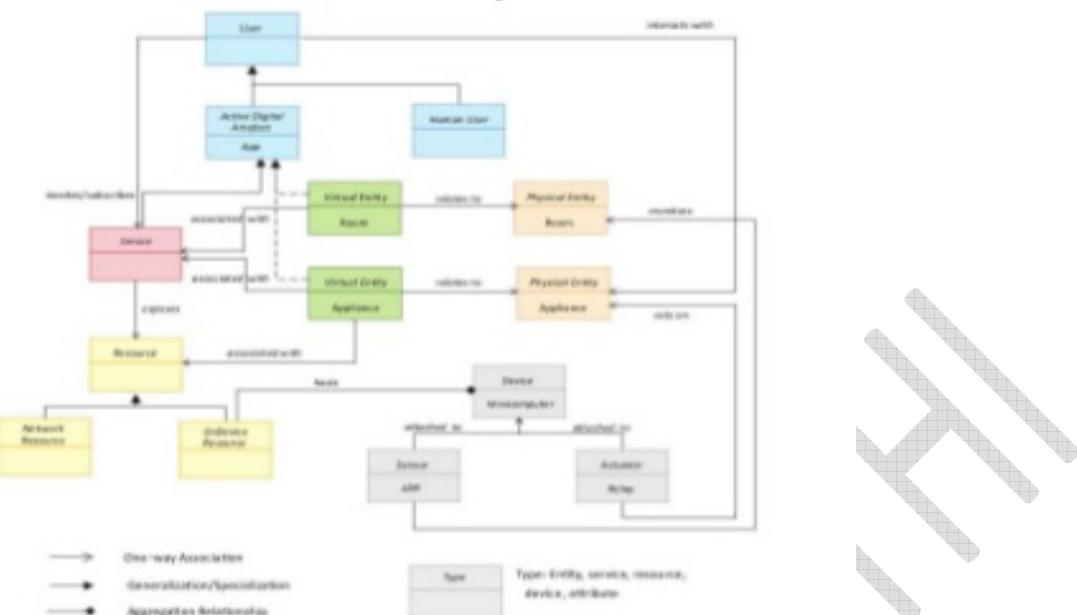
Step:1 - Purpose & Requirements

- Applying this to our example of a smart home automation system, the purpose and requirements for the system may be described as follows:
 - Purpose : A home automation system that allows controlling of the lights in a home remotely using a web application.
 - Behaviour : The home automation system should have auto and manual modes. In auto mode, the system measures the light level in the room and switches on the light when it gets dark. In manual mode, the system provides the option of manually and remotely switching on/off the light.
 - System Management Requirement : The system should provide remote monitoring and control functions.
 - Data Analysis Requirement : The system should perform local analysis of the data.
 - Application Deployment Requirement : The application should be deployed locally on the device, but should be accessible remotely.
 - Security Requirement : The system should have basic user authentication capability.

Step:2 - Process Specification



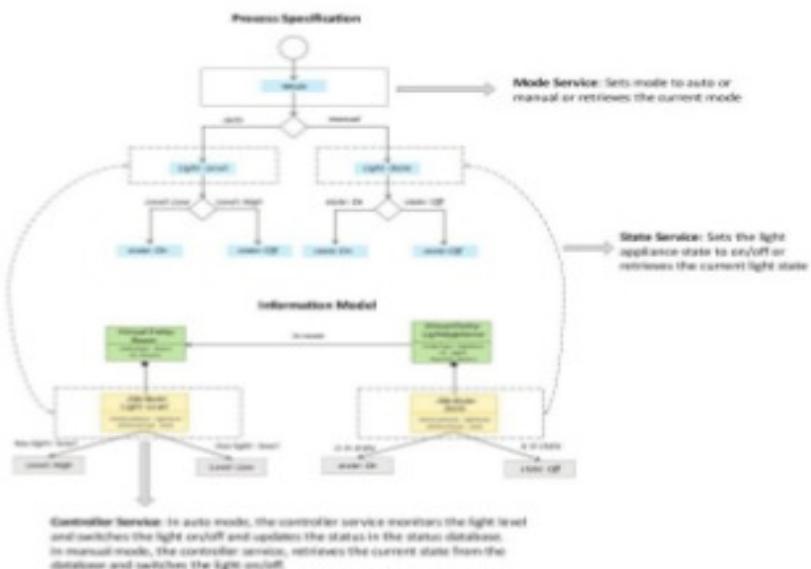
Step 3: Domain Model Specification



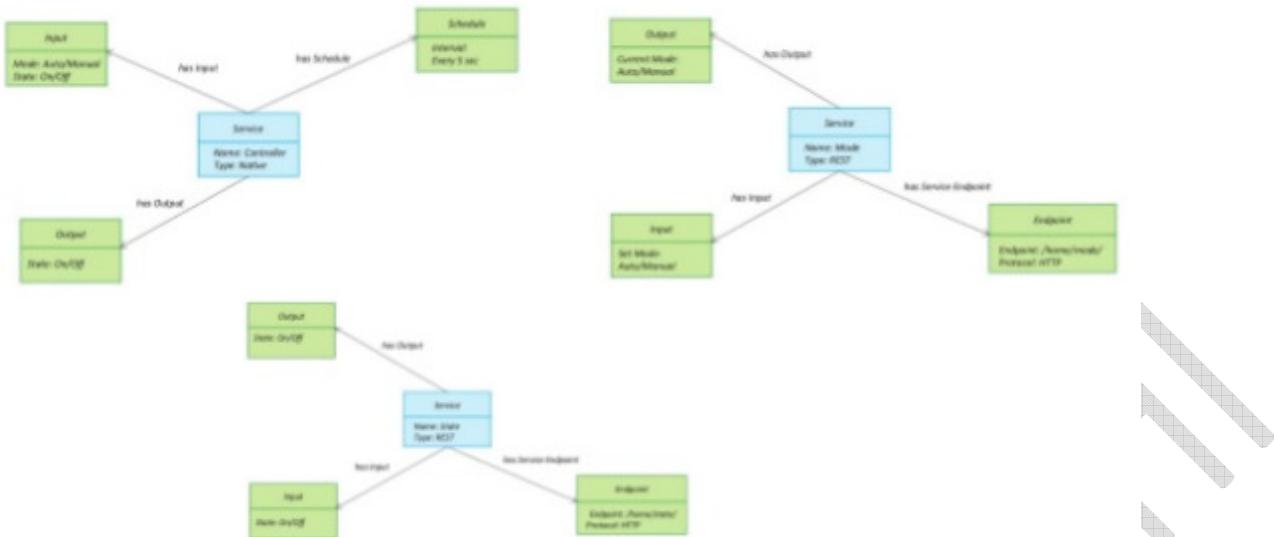
Step 4: Information Model Specification



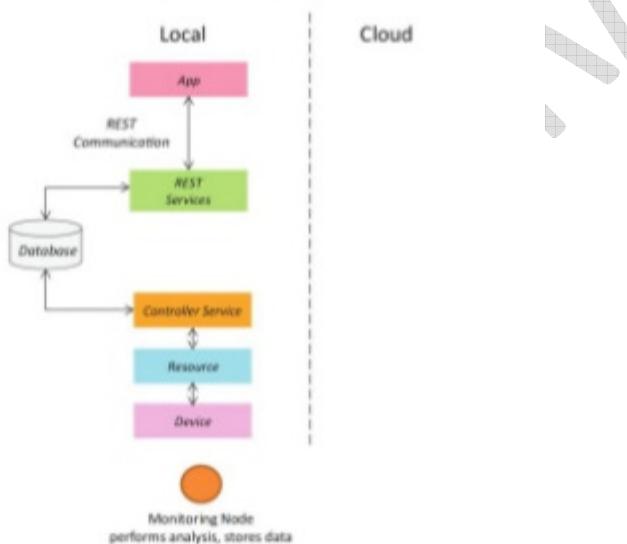
Step 5: Service Specifications



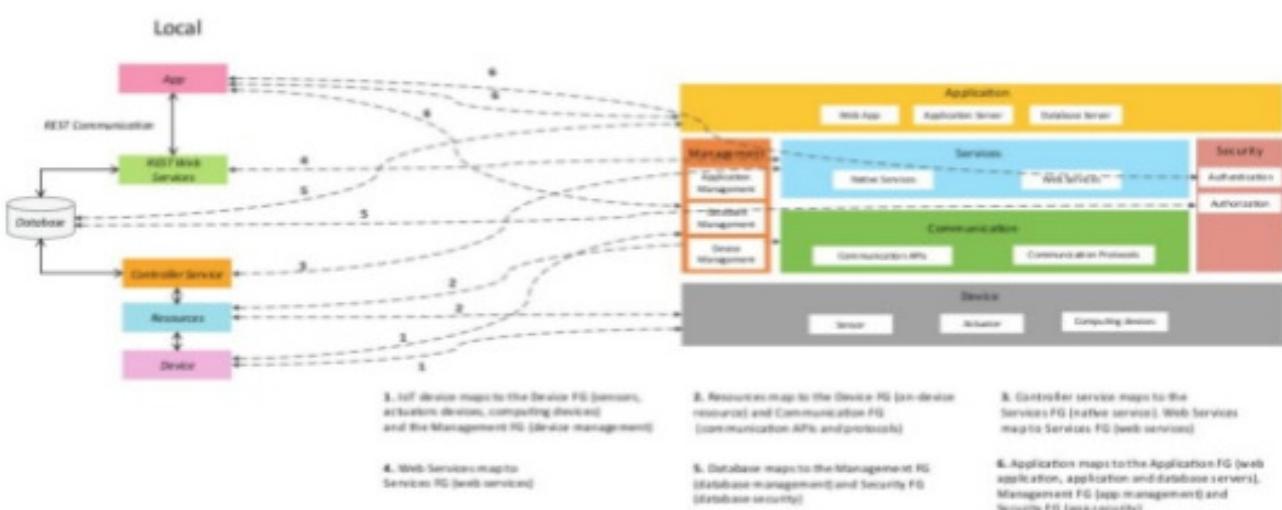
Step 5: Service Specifications



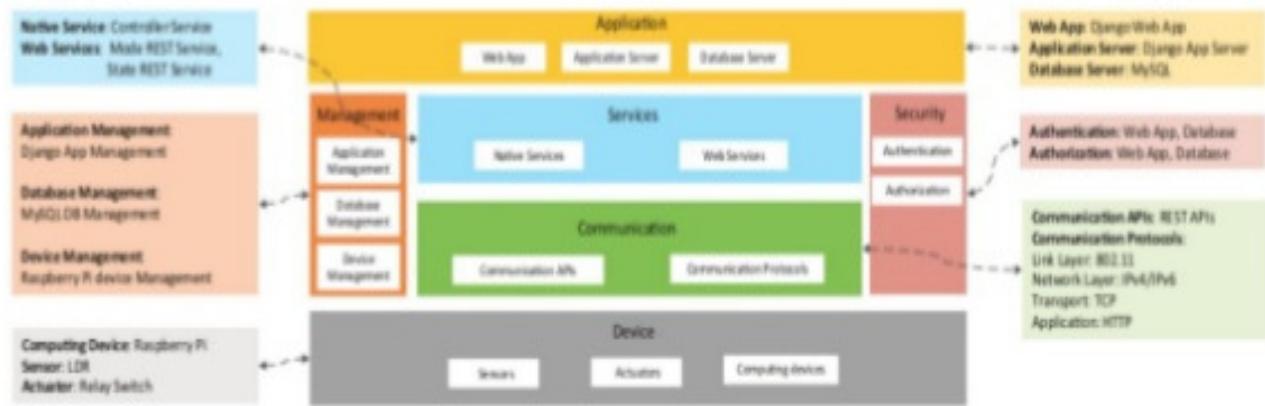
Step 6: IoT Level Specification



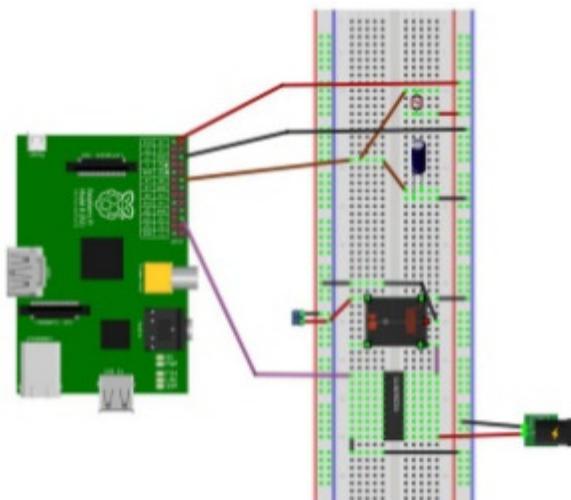
Step 7: Functional View Specification



Step 8: Operational View Specification



Step 9: Device & Component Integration



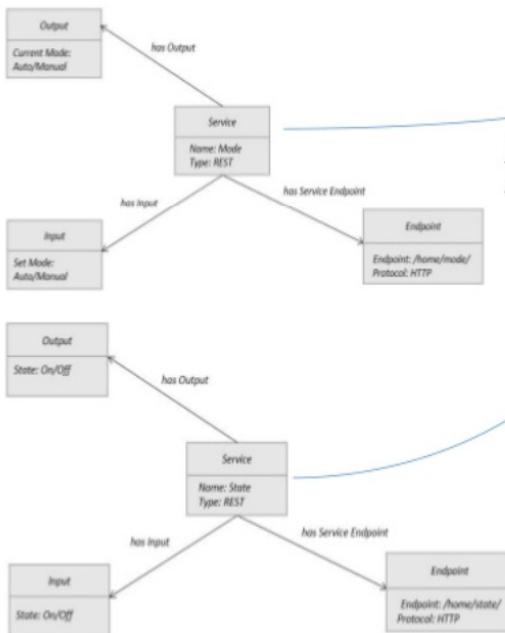
Step 10: Application Development

- Auto
 - Controls the light appliance automatically based on the lighting conditions in the room
- Light
 - When Auto mode is off, it is used for manually controlling the light appliance.
 - When Auto mode is on, it reflects the current state of the light appliance.



Implementation: RESTful Web Services

REST services implemented with Django REST Framework



1. Map services to models. Model fields store the states (on/off, auto/manual)

```
# Models - models.py
from django.db import models

class Mode(models.Model):
    name = models.CharField(max_length=50)

class State(models.Model):
    name = models.CharField(max_length=50)
```

2. Write Model serializers. Serializers allow complex data (such as model instances) to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types.

```
# Serializers - serializers.py
from myapp.models import Mode, State
from rest_framework import serializers

class ModeSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Mode
        fields = ('url', 'name')

class StateSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = State
        fields = ('url', 'name')
```

Implementation: RESTful Web Services

```
# Models - models.py
from django.db import models

class Mode(models.Model):
    name = models.CharField(max_length=50)

class State(models.Model):
    name = models.CharField(max_length=50)
```

3. Write ViewSets for the Models which combine the logic for a set of related views in a single class.

```
# Views - views.py
from myapp.models import Mode, State
from rest_framework import viewsets
from myapp.serializers import ModeSerializer, StateSerializer

class ModeViewSet(viewsets.ModelViewSet):
    queryset = Mode.objects.all()
    serializer_class = ModeSerializer

class StateViewSet(viewsets.ModelViewSet):
    queryset = State.objects.all()
    serializer_class = StateSerializer
```

```
# URL Patterns - urls.py
from django.conf.urls import patterns, include, url
from django.contrib import admin
from rest_framework import routers
from myapp import views
admin.autodiscover()
router = routers.DefaultRouter()
router.register(r'mode', views.ModeViewSet)
router.register(r'state', views.StateViewSet)
urlpatterns = patterns('',
    url(r'^', include(router.urls)),
    url(r'^api-auth/', include('rest_framework.urls', namespace='rest_framework')),
    url(r'^admin/', include(admin.site.urls)),
    url(r'^home/$', 'myapp.views.home'),
)
```

4. Write URL patterns for the services. Since ViewSets are used instead of views, we can automatically generate the URL conf by simply registering the viewsets with a router class. Routers automatically determine how the URLs for an application should be mapped to the logic that deals with handling incoming requests.

Implementation: RESTful Web Services

Screenshot of browsable State REST API

```

    Api Root - State List
    State List
    GET /state/
    HTTP 200 OK
    Vary: Accept
    Content-Type: text/html; charset=utf-8
    Allow: GET, POST, HEAD, OPTIONS

    {
        "count": 1,
        "next": null,
        "previous": null,
        "results": [
            {
                "url": "http://localhost:8000/state/1",
                "name": "light"
            }
        ]
    }
  
```

Screenshot of browsable Mode REST API

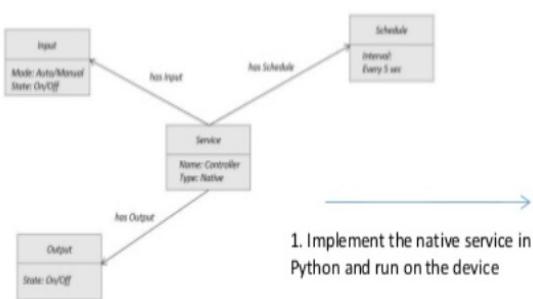
```

    Api Root - Mode List - Mode Instance
    Mode Instance
    GET /mode/1/
    HTTP 200 OK
    Vary: Accept
    Content-Type: text/html; charset=utf-8
    Allow: GET, PUT, DELETE, HEAD, OPTIONS, PATCH

    {
        "url": "http://localhost:8000/mode/1",
        "name": "manual"
    }
  
```

Implementation: Controller Native Service

Native service deployed locally



#Controller service

```

import RPi.GPIO as GPIO
import time
import sqlite3 as lite
import sys

con = lite.connect('database.sqlite')
cur = con.cursor()

GPIO.setmode(GPIO.BCM)
threshold = 1000
LDR_PIN = 18
LIGHT_PIN = 25

def readldr(PIN):
    reading=0
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, GPIO.LOW)
    time.sleep(0.1)
    GPIO.setup(PIN, GPIO.IN)
    while (GPIO.input(PIN)==GPIO.LOW):
        reading=reading+1
    return reading

def switchOnLight(PIN):
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, GPIO.HIGH)

def switchOffLight(PIN):
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, GPIO.LOW)
  
```

```

def runAutoMode():
    ldr_reading = readldr(LDR_PIN)
    if ldr_reading < threshold:
        switchOnLight(LIGHT_PIN)
        setCurrentState('on')
    else:
        switchOffLight(LIGHT_PIN)
        setCurrentState('off')

def runManualMode():
    state = getCurrentState()
    if state=="on":
        switchOnLight(LIGHT_PIN)
        setCurrentState('on')
    elif state=="off":
        switchOffLight(LIGHT_PIN)
        setCurrentState('off')

def getCurrentMode():
    cur.execute("SELECT * FROM myapp_mode")
    data = cur.fetchone()      #[1, 'auto']
    return data[1]

def getCurrentState():
    cur.execute("SELECT * FROM myapp_state")
    data = cur.fetchone()      #[1, 'on']
    return data[1]

def setCurrentState(val):
    query="UPDATE myapp_state set name='"+val+"'"
    cur.execute(query)

while True:
    currentMode=getCurrentMode()
    if currentMode=="auto":
        runAutoMode()
    elif currentMode=="manual":
        runManualMode()
    time.sleep(5)
  
```

Implementation: Application

1. Implement Django Application View

```
# Views - views.py
def home(request):
    out=""
    if 'on' in request.POST:
        values = {"name": "on"}
        r=requests.put("http://127.0.0.1:8000/state/1/", data=values, auth=('username', 'password'))
        result=r.text
        output = json.loads(result)
        out=output['name']
    if 'off' in request.POST:
        values = {"name": "off"}
        r=requests.put("http://127.0.0.1:8000/state/1/", data=values, auth=('username', 'password'))
        result=r.text
        output = json.loads(result)
        out=output['name']
    if 'auto' in request.POST:
        values = {"name": "auto"}
        r=requests.put("http://127.0.0.1:8000/mode/1/", data=values, auth=('username', 'password'))
        result=r.text
        output = json.loads(result)
        out=output['name']
    if 'manual' in request.POST:
        values = {"name": "manual"}
        r=requests.put("http://127.0.0.1:8000/mode/1/", data=values, auth=('username', 'password'))
        result=r.text
        output = json.loads(result)
        out=output['name']

    r=requests.get("http://127.0.0.1:8000/mode/1/", auth=('username', 'password'))
    result=r.text
    output = json.loads(result)
    currentmode=output['name']
    r=requests.get("http://127.0.0.1:8000/state/1/", auth=('username', 'password'))
    result=r.text
    output = json.loads(result)
    currentstate=output['name']
    return render_to_response('lights.html',{'out':out,'currentmode':currentmode,'currentstate':currentstate},
    context_instance=RequestContext(request))
```

Implementation: Application

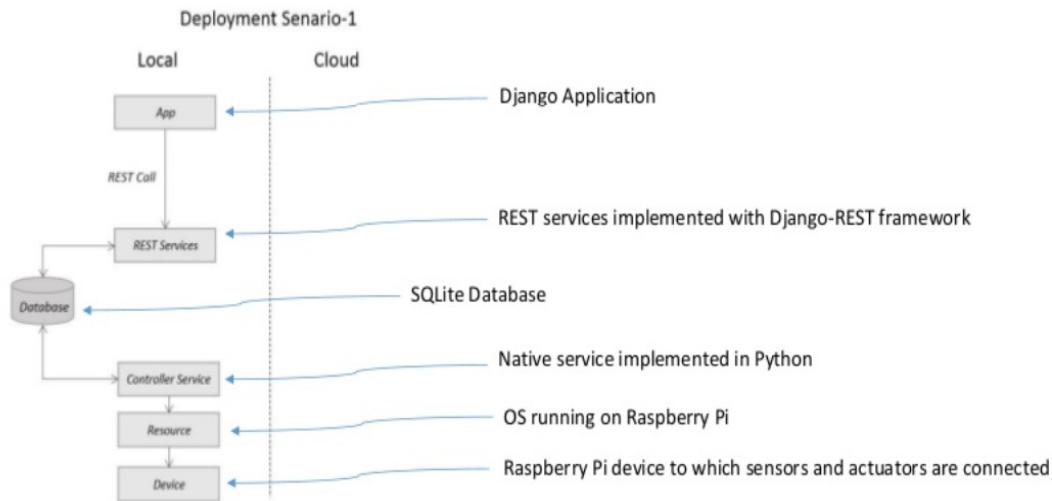
2. Implement Django Application Template

```
<div class="app-content-inner">
<fieldset>
<div class="field draft">
<label class="input-label icon-lamp" for="lamp-state">Auto</label>
<input id="lamp-state" class="input js-lamp-state hidden" type="checkbox">
{if currentmode == 'auto'}
<div class="js-lamp-state-toggle ui-toggle" data-toggle="js-lamp-state">
{else}
<div class="js-lamp-state-toggle ui-toggle js-toggle-off" data-toggle="js-lamp-state">
{endif}
<span class="ui-toggle-slide draft">
<form id="my_form1" action="" method="post">{{ csrf_token }}</form>
<input name="auto" value="auto" type="hidden" />
<a href="#" onclick="$(<this>.closest('form')).submit()"><strong class="ui-toggle-off">Off</strong></a>
</form>
<strong class="ui-toggle-handle brushed-metal"></strong>
<form id="my_form3" action="" method="post">{{ csrf_token }}</form>
<input name="manual" value="manual" type="hidden" />
<a href="#" onclick="$(<this>.closest('form')).submit()"><strong class="ui-toggle-on">On</strong></a>
</form>
</span>
</div>
<div class="field draft">
<label class="input-label icon-lamp" for="tv-state">Light</label>
<input id="tv-state" class="input js-tv-state hidden" type="checkbox">
{if currentmode == 'on'}
<div class="js-tv-state-toggle ui-toggle" data-toggle="js-tv-state">
{endif}
<div class="js-tv-state-toggle ui-toggle js-toggle-off" data-toggle="js-tv-state">
{endif}
{if currentmode == 'manual'}
<span class="ui-toggle-slide draft">
<form id="my_form2" action="" method="post">{{ csrf_token }}</form>
<input name="on" value="on" type="hidden" />
<a href="#" onclick="$(<this>.closest('form')).submit()"><strong class="ui-toggle-off">Off</strong></a>
</form>
<strong class="ui-toggle-handle brushed-metal"></strong>
<form id="my_form3" action="" method="post">{{ csrf_token }}</form>
<input name="off" value="off" type="hidden" />
<a href="#" onclick="$(<this>.closest('form')).submit()"><strong class="ui-toggle-on">On</strong></a>
</form>
</span>
{endif}
{if currentmode == 'auto'}
{if currentstate == 'on'}
<strong class="ui-toggle-on">&nbsp;&nbsp;&nbsp;ON</strong>
{else}
<strong class="ui-toggle-on">&nbsp;&nbsp;&nbsp;Off</strong>
{endif}
{endif}
</div>
</div>
</fieldset></div></div></div>
```



Finally - Integrate the System

- Setup the device
- Deploy and run the REST and Native services
- Deploy and run the Application
- Setup the database



Outline

- Introduction to Python
- Installing Python
- Python Data Types & Data Structures
- Control Flow
- Functions
- Modules
- Packages
- File Input/Output
- Date/Time Operations
- Classes

Python

- Python is a general-purpose high level programming language and suitable for providing a solid foundation to the reader in the area of cloud computing.
- The main characteristics of Python are:
 - Multi-paradigm programming language
 - Python supports more than one programming paradigms including object-oriented programming and structured programming
 - Interpreted Language
 - Python is an interpreted language and does not require an explicit compilation step. The Python interpreter executes the program source code directly, statement by statement, as a processor or scripting engine does.
 - Interactive Language
 - Python provides an interactive mode in which the user can submit commands at the Python prompt and interact with the interpreter directly

Python - Benefits

- Easy-to-learn, read and maintain

- Python is a minimalistic language with relatively few keywords, uses English keywords and has fewer syntactical constructions as compared to other languages. Reading Python programs feels like English with pseudo-code like constructs. Python is easy to learn yet an extremely powerful language for a wide range of applications.
- Object and Procedure Oriented
 - Python supports both procedure-oriented programming and object-oriented programming. Procedure oriented paradigm allows programs to be written around procedures or functions that allow reuse of code. Procedure oriented paradigm allows programs to be written around objects that include both data and functionality.
- Extendable
 - Python is an extendable language and allows integration of low-level modules written in languages such as C/C++. This is useful when you want to speed up a critical portion of a program.
- Scalable
 - Due to the minimalistic nature of Python, it provides a manageable structure for large programs.
- Portable
 - Since Python is an interpreted language, programmers do not have to worry about compilation, linking and loading of programs. Python programs can be directly executed from source
- Broad Library Support
 - Python has a broad library support and works on various platforms such as Windows, Linux, Mac, etc.

Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them. For example

```
var1 = 1
var2 = 10
```

Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows either pair of single or double quotes. Subsets of strings can be taken using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 to the end.

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator. For example –

```
str = 'Hello World!'

print (str)          # Prints complete string
print (str[0])       # Prints first character of the string
print (str[2:5])     # Prints characters starting from 3rd to 5th
print (str[2:])       # Prints string starting from 3rd character
print (str * 2)       # Prints string two times
print (str + "TEST")  # Prints concatenated string
```

Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One of the differences between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator. For example –

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print (list)          # Prints complete list
print (list[0])       # Prints first element of the list
print (list[1:3])     # Prints elements starting from 2nd till 3rd
```

```

print (list[2:])      # Prints elements starting from 3rd element
print (tinylist * 2)  # Prints list two times
print (list + tinylist) # Prints concatenated lists

```

Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parenthesis.

The main difference between lists and tuples are – Lists are enclosed in brackets ([]) and their elements and size can be changed, while tuples are enclosed in parentheses (()) and cannot be updated. Tuples can be thought of as **read-only** lists. For example –

```

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')

print (tuple)          # Prints complete tuple
print (tuple[0])       # Prints first element of the tuple
print (tuple[1:3])     # Prints elements starting from 2nd till 3rd
print (tuple[2:])       # Prints elements starting from 3rd element
print (tinytuple * 2)   # Prints tuple two times
print (tuple + tinytuple) # Prints concatenated tuple

```

Python Dictionary

Python's dictionaries are kind of hash-table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]). For example –

```

dict = {}
dict['one'] = "This is one"
dict[2]      = "This is two"

tinydict = {'name': 'john', 'code':6734, 'dept': 'sales'}

print (dict['one'])        # Prints value for 'one' key
print (dict[2])           # Prints value for 2 key
print (tinydict)          # Prints complete dictionary
print (tinydict.keys())    # Prints all the keys
print (tinydict.values())  # Prints all the values

```

Data Type Conversion

Sometimes, may need to perform conversions between the built-in types. To convert between types, you simply use the type-names as a function.

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

Sr.No.	Function & Description
1	int(x [,base]) Converts x to an integer. The base specifies the base if x is a string.
2	float(x) Converts x to a floating-point number.

3	complex(real [,imag]) Creates a complex number.
4	str(x) Converts object x to a string representation.
5	repr(x) Converts object x to an expression string.
6	eval(str) Evaluates a string and returns an object.
7	tuple(s) Converts s to a tuple.
8	list(s) Converts s to a list.
9	set(s) Converts s to a set.
10	dict(d) Creates a dictionary. d must be a sequence of (key,value) tuples.
11	frozenset(s) Converts s to a frozen set.
12	chr(x) Converts an integer to a character.
13	unichr(x) Converts an integer to a Unicode character.
14	ord(x) Converts a single character to its integer value.
15	hex(x) Converts an integer to a hexadecimal string.
16	oct(x) Converts an integer to an octal string.

Control Flow – if statement

Decision-making is the anticipation of conditions occurring during the execution of a program and specified actions taken according to the conditions.

Decision structures evaluate multiple expressions, which produce TRUE or FALSE as the outcome. You need to determine which action to take and which statements to execute if the outcome is TRUE or FALSE otherwise.

Following is the general form of a typical decision making structure found in most of the programming languages –

The IF statement is similar to that of other languages. The **if** statement contains a logical expression using which the data is compared and a decision is made based on the result of the comparison.

Syntax

if expression:
 statement(s)

```
var1 = 100
if var1:
    print ("1 - Got a true expression value")
    print (var1)
```

An **else** statement can be combined with an **if** statement. An **else** statement contains a block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

The **else** statement is an optional statement and there could be at the most only one **else** statement following **if**.
Syntax

The syntax of the **if...else** statement is –

```
if expression:
    statement(s)
```

```
else:
    statement(s)
```

```
amount = int(input("Enter amount: "))

if amount<1000:
    discount = amount*0.05
    print ("Discount",discount)
else:
    discount = amount*0.10
    print ("Discount",discount)

print ("Net payable:",amount-discount)
```

In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.

Syntax

The syntax of the nested **if...elif...else** construct may be –

```
if expression1:
    statement(s)
    if expression2:
        statement(s)
    elif expression3:
        statement(s)
    else:
        statement(s)
elif expression4:
    statement(s)
else:
    statement(s)
```

```
num = int(input("enter number"))
if num%2 == 0:
    if num%3 == 0:
        print ("Divisible by 3 and 2")
    else:
        print ("divisible by 2 not divisible by 3")
else:
    if num%3 == 0:
        print ("divisible by 3 not divisible by 2")
    else:
```

```
print ("not Divisible by 2 not divisible by 3")
```

The for statement in Python has the ability to iterate over the items of any sequence, such as a list or a string.

Syntax

```
for iterating_var in sequence:  
    statements(s)
```

The range() function

The built-in function range() is the right function to iterate over a sequence of numbers. It generates an iterator of arithmetic progressions.

Example

```
>>> range(5)  
range(0, 5)  
>>> list(range(5))  
[0, 1, 2, 3, 4]
```

Example

range() generates an iterator to progress integers starting with 0 upto n-1. To obtain a list object of the sequence, it is typecasted to list(). Now this list can be iterated using the for statement.

```
>>> for var in list(range(5)):  
    print (var)
```

A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

Syntax

The syntax of a **while** loop in Python programming language is –

```
while expression:  
    statement(s)
```

```
count = 0  
while (count < 9):  
    print ('The count is:', count)  
    count = count + 1  
  
print ("Good bye!")
```

Python programming language allows the usage of one loop inside another loop. The following section shows a few examples to illustrate the concept.

Syntax

```
for iterating_var in sequence:  
    for iterating_var in sequence:  
        statements(s)  
    statements(s)
```

The syntax for a nested while loop statement in Python programming language is as follows –

while expression:

```
    while expression:  
        statement(s)  
    statement(s)
```

```
import sys  
for i in range(1,11):  
    for j in range(1,11):
```

```

k = i*j
print (k, end=' ')
print()

```

Loop Control Statements

The Loop control statements change the execution from its normal sequence. When the execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Python supports the following control statements.

Sr.No.	Control Statement & Description
1	break statement Terminates the loop statement and transfers execution to the statement immediately following the loop.
2	continue statement Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
3	pass statement The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

Function

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called *user-defined functions*.

Defining a Function

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword **def** followed by the function name and parentheses (()).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Syntax

```

def functionname( parameters ):
    "function_docstring"
    function_suite
    return [expression]

```

Example

The following function takes a string as input parameter and prints it on standard screen.

```

def printme( str ):
    "This prints a passed string into this function"
    print (str)
    return

```

Calling a Function

Defining a function gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.

Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is an example to call the `printme()` function –

```
# Function definition is here
def printme( str ):
    "This prints a passed string into this function"
    print (str)
    return

# Now you can call printme function
printme("This is first call to the user defined function!")
printme("Again second call to the same function")
```

Pass by Reference vs Value

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example –

```
# Function definition is here
def changeme( mylist ):
    "This changes a passed list into this function"
    print ("Values inside the function before change: ", mylist)

    mylist[2]=50
    print ("Values inside the function after change: ", mylist)
    return

# Now you can call changeme function
mylist = [10,20,30]
changeme( mylist )
print ("Values outside the function: ", mylist)
```

Function Arguments

You can call a function by using the following types of formal arguments –

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

Required Arguments

Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition.

To call the function `printme()`, you definitely need to pass one argument, otherwise it gives a syntax error as follows –

```
# Function definition is here
def printme( str ):
    "This prints a passed string into this function"
    print (str)
    return

# Now you can call printme function
printme()
```

Keyword Arguments

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters. You can also make keyword calls to the **printme()** function in the following ways –

```
# Function definition is here
def printme( str ):
    "This prints a passed string into this function"
    print (str)
    return

# Now you can call printme function
printme( str = "My string")
```

Default Arguments

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument. The following example gives an idea on default arguments, it prints default age if it is not passed –

```
# Function definition is here
def printinfo( name, age = 35 ):
    "This prints a passed info into this function"
    print ("Name: ", name)
    print ("Age ", age)
    return

# Now you can call printinfo function
printinfo( age = 50, name = "miki" )
printinfo( name = "miki" )
```

Variable-length Arguments

You may need to process a function for more arguments than you specified while defining the function. These arguments are called *variable-length* arguments and are not named in the function definition, unlike required and default arguments.

Syntax for a function with non-keyword variable arguments is given below –

```
def functionname([formal_args,] *var_args_tuple ):
    "function_docstring"
    function_suite
    return [expression]
```

An asterisk (*) is placed before the variable name that holds the values of all nonkeyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call. Following is a simple example –

```
# Function definition is here
def printinfo( arg1, *vartuple ):
    "This prints a variable passed arguments"
    print ("Output is: ")
    print (arg1)

    for var in vartuple:
        print (var)
    return

# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

Module

- Python allows organizing the program code into different modules which improves the code readability and management.
- A module is a Python file that defines some functionality in the form of functions or classes.
- Modules can be imported using the import keyword.
- Modules to be imported must be present in the search path

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.

Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

Example

The Python code for a module named aname normally resides in a file named aname.py. Here is an example of a simple module, support.py –

```
def print_func( par ):  
    print "Hello : ", par  
    return
```

The import Statement

You can use any Python source file as a module by executing an import statement in some other Python source file.

The **import** has the following syntax –

```
import module1[, module2[... moduleN]]
```

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module. For example, to import the module hello.py, you need to put the following command at the top of the script –

```
# Import module support  
import support  
  
# Now you can call defined function that module as follows  
support.print_func("Zara")
```

The from...import Statement

Python's **from** statement lets you import specific attributes from a module into the current namespace.

The **from...import** has the following syntax –

```
from modname import name1[, name2[, ... nameN]]
```

For example, to import the function fibonacci from the module fib, use the following statement –

```
def fib(n): # return Fibonacci series up to n  
    result = []  
    a, b = 0, 1  
    while b < n:  
        result.append(b)  
        a, b = b, a + b  
    return result  
>>> from fib import fib  
>>> fib(100)  
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

The from...import * Statement

It is also possible to import all the names from a module into the current namespace by using the following import statement –

```
from modname import *
```

This provides an easy way to import all the items from a module into the current namespace; however, this statement should be used sparingly.

Packages

- Python package is hierarchical file structure that consists of modules and subpackages.
- Packages allow better organization of modules related to a single application environment.

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and subpackages and sub-subpackages, and so on.

Consider a file *Pots.py* available in *Phone* directory. This file has the following line of source code –

```
def Pots():
    print ("I'm Pots Phone")
```

Similar, we have other two files having different functions with the same name as above. They are –

- *Phone/Isdn.py* file having function *Isdn()*
- *Phone/G3.py* file having function *G3()*

Now, create one more file *__init__.py* in the *Phone* directory –

- *Phone/__init__.py*

To make all of your functions available when you have imported *Phone*, you need to put explicit import statements in *__init__.py* as follows –

```
from Pots import Pots
from Isdn import Isdn
from G3 import G3
```

After you add these lines to *__init__.py*, you have all of these classes available when you import the *Phone* package.

```
# Now import your Phone Package.
import Phone

Phone.Pots()
Phone.Isdn()
Phone.G3()
```

File Handling

- Python allows reading and writing to files using the *file* object.
- The *open(filename, mode)* function is used to get a file object.
- The mode can be read (*r*), write (*w*), append (*a*), read and write (*r+* or *w+*), read-binary (*rb*), write-binary (*wb*), etc.
- After the file contents have been read the *close* function is called which closes the file object.

Example of reading an entire file

```
>>>fp = open('file.txt','r')
>>>content = fp.read()
>>>print content
This is a test file.
>>>fp.close()
```

Example of reading line by line

```
>>>fp = open('file1.txt','r')
>>>print "Line-1: " + fp.readline()
Line-1: Python supports more than one programming paradigms.
>>>print "Line-2: " + fp.readline()
Line-2: Python is an interpreted language.
>>>fp.close()
```

Example of reading lines in a loop

```
>>>fp = open('file1.txt','r')
>>>lines = fp.readlines()
>>>for line in lines:
    print line
```

Python supports more than one programming paradigms.

Python is an interpreted language.

Example of reading a certain number of bytes

```
>>>fp = open('file.txt','r')
>>>fp.read(10)
'Python sup'
>>>fp.close()
```

Example of seeking to a certain position

```
>>>fp = open('file.txt','r')
>>>fp.seek(10,0)
>>>content = fp.read(10)
>>>print content
ports more
>>>fp.close()
```

Example of getting the current position of read

```
>>>fp = open('file.txt','r')
>>>fp.read(10)
'Python sup'
>>>currentpos = fp.tell
>>>print currentpos
<built-in method tell of file object at 0x000000002391390>
>>>fp.close()
```

Example of writing to a file

```
>>>fo = open('file1.txt','w')
>>>content='This is an example of writing to a file in
Python.'
>>>fo.write(content)
>>>fo.close()
```

Date/Time Operations

- Python provides several functions for date and time access and conversions.
- The datetime module allows manipulating date and time in several ways.
- The time module in Python provides various time-related functions.

Examples of manipulating with date

```
>>>from datetime import date
>>>now = date.today()
>>>print "Date: " + now.strftime("%m-%d-%Y")
Date: 07-24-13

>>>print "Day of Week: " + now.strftime("%A")
Day of Week: Wednesday

>>>print "Month: " + now.strftime("%B")
Month: July

>>>then = date(2013, 6, 7)
>>>timediff = now - then
>>>timediff.days
```

Examples of manipulating with time

```
>>>import time
>>>nowtime = time.time()
>>>time.localtime(nowtime)
time.struct_time(tm_year=2013, tm_mon=7, tm_mday=24, tm_hour=16, tm_min=14, tm_sec=51, tm_wday=2, tm_yday=205,
tm_isdst=0)

>>>time.asctime(time.localtime(nowtime))
'Wed Jul 24 16:14:51 2013'

>>>time.strftime("The date is %d-%m-%y. Today is a %A. It is %H hours, %M minutes and %S seconds now.")
'The date is 24-07-13. Today is a Wednesday. It is 16 hours, 15 minutes and 14 seconds now.'
```

Python has been an object-oriented language since the time it existed. Due to this, creating and using classes and objects are downright easy. This chapter helps you become an expert in using Python's object-oriented programming support.

If you do not have any previous experience with object-oriented (OO) programming, you may want to consult an introductory course on it or at least a tutorial of some sort so that you have a grasp of the basic concepts.

However, here is a small introduction of Object-Oriented Programming (OOP) to help you –

Overview of OOP Terminology

- **Class** – A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- **Class variable** – A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- **Data member** – A class variable or instance variable that holds data associated with a class and its objects.
- **Function overloading** – The assignment of more than one behavior to a particular function. The operation performed varies by the types of objects or arguments involved.
- **Instance variable** – A variable that is defined inside a method and belongs only to the current instance of a class.
- **Inheritance** – The transfer of the characteristics of a class to other classes that are derived from it.
- **Instance** – An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.
- **Instantiation** – The creation of an instance of a class.
- **Method** – A special kind of function that is defined in a class definition.
- **Object** – A unique instance of a data structure that is defined by its class. An object comprises both data members (class variables and instance variables) and methods.
- **Operator overloading** – The assignment of more than one function to a particular operator.

Creating Classes

The `class` statement creates a new class definition. The name of the class immediately follows the keyword `class` followed by a colon as follows –

```
class ClassName:  
    'Optional class documentation string'  
    class_suite
```

```
class Employee:  
    'Common base class for all employees'  
    empCount = 0  
  
    def __init__(self, name, salary):  
        self.name = name  
        self.salary = salary  
        Employee.empCount += 1  
  
    def displayCount(self):  
        print ("Total Employee %d" % Employee.empCount)  
  
    def displayEmployee(self):  
        print ("Name : ", self.name, " , Salary: ", self.salary)  
  
  
#This would create first object of Employee class"  
emp1 = Employee("Zara", 2000)  
#This would create second object of Employee class"  
emp2 = Employee("Manni", 5000)  
emp1.displayEmployee()  
emp2.displayEmployee()  
print ("Total Employee %d" % Employee.empCount)
```

Class Inheritance

Instead of starting from a scratch, you can create a class by deriving it from a pre-existing class by listing the parent class in parentheses after the new class name.

The child class inherits the attributes of its parent class, and you can use those attributes as if they were defined in the child class. A child class can also override data members and methods from the parent.

Syntax

Derived classes are declared much like their parent class; however, a list of base classes to inherit from is given after the class name –

```
class SubClassName (ParentClass1[, ParentClass2, ...]):  
    'Optional class documentation string'  
    class_suite
```

```
class Parent:          # define parent class  
    parentAttr = 100  
    def __init__(self):  
        print ("Calling parent constructor")  
  
    def parentMethod(self):  
        print ('Calling parent method')  
  
    def setAttr(self, attr):  
        Parent.parentAttr = attr  
  
    def getAttr(self):  
        print ("Parent attribute :", Parent.parentAttr)  
  
class Child(Parent): # define child class  
    def __init__(self):  
        print ("Calling child constructor")  
  
    def childMethod(self):  
        print ('Calling child method')  
  
c = Child()           # instance of child  
c.childMethod()       # child calls its method  
c.parentMethod()     # calls parent's method  
c.setAttr(200)        # again call parent's method  
c.getAttr()          # again call parent's method
```

Overriding Methods

You can always override your parent class methods. One reason for overriding parent's methods is that you may want special or different functionality in your subclass.

Example

```
class Parent:          # define parent class  
    def myMethod(self):  
        print ('Calling parent method')  
  
class Child(Parent): # define child class  
    def myMethod(self):  
        print ('Calling child method')  
  
c = Child()           # instance of child  
c.myMethod()          # child calls overridden method
```

Overloading Operators

Suppose you have created a Vector class to represent two-dimensional vectors. What happens when you use the plus operator to add them? Most likely Python will yell at you.

You could, however, define the `__add__` method in your class to perform vector addition and then the plus operator would behave as per expectation –

Example

```
class Vector:  
    def __init__(self, a, b):  
        self.a = a  
        self.b = b  
  
    def __str__(self):  
        return 'Vector (%d, %d)' % (self.a, self.b)  
  
    def __add__(self, other):  
        return Vector(self.a + other.a, self.b + other.b)  
  
v1 = Vector(2,10)  
v2 = Vector(5,-2)  
print (v1 + v2)
```

IoT Physical Devices & Endpoints

Outline

- Basic building blocks of an IoT Device
- Exemplary Device: Raspberry Pi
- Raspberry Pi interfaces
- Programming Raspberry Pi with Python
- Other IoT devices

What is an IoT Device

- A "Thing" in Internet of Things (IoT) can be any object that has a unique identifier and which can send/receive data (including user data) over a network (e.g., smart phone, smart TV, computer, refrigerator, car, etc.).
- IoT devices are connected to the Internet and send information about themselves or about their surroundings (e.g. information sensed by the connected sensors) over a network (to other devices or servers/storage) or allow actuation upon the physical entities/environment around them remotely.

IoT Device Examples

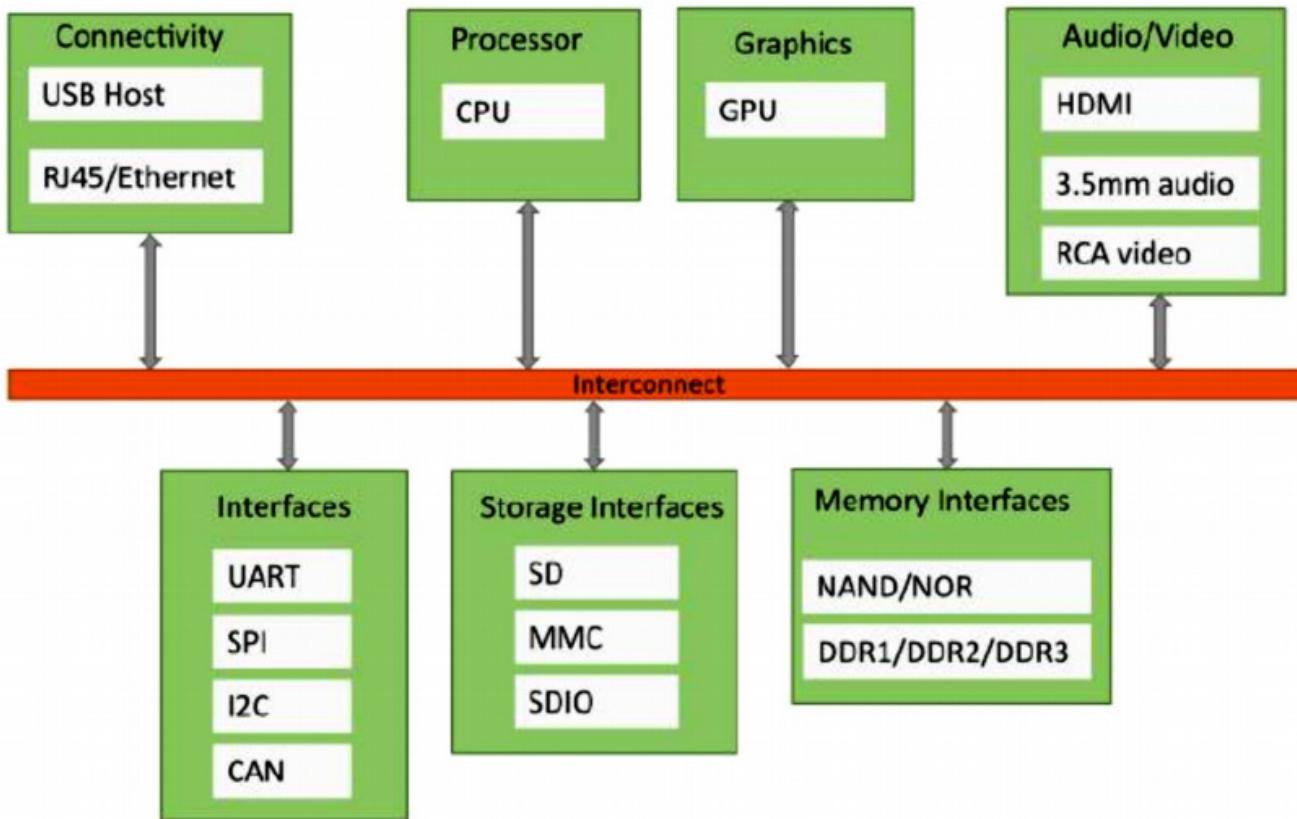
- A home automation device that allows remotely monitoring the status of appliances and controlling the appliances.
- An industrial machine which sends information about its operation and health monitoring data to a server.
- A car which sends information about its location to a cloud-based service.
- A wireless-enabled wearable device that measures data about a person such as the number of steps walked and sends the data to a cloud-based service.

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 the device.
- 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



Stages of IoT Solutions Architecture

There are several layers of IoT built upon the capability and performance of IoT elements that provides the optimal solution to the business enterprises and end-users. The IoT architecture is a fundamental way to design the various elements of IoT, so that it can deliver services over the networks and serve the needs for the future.

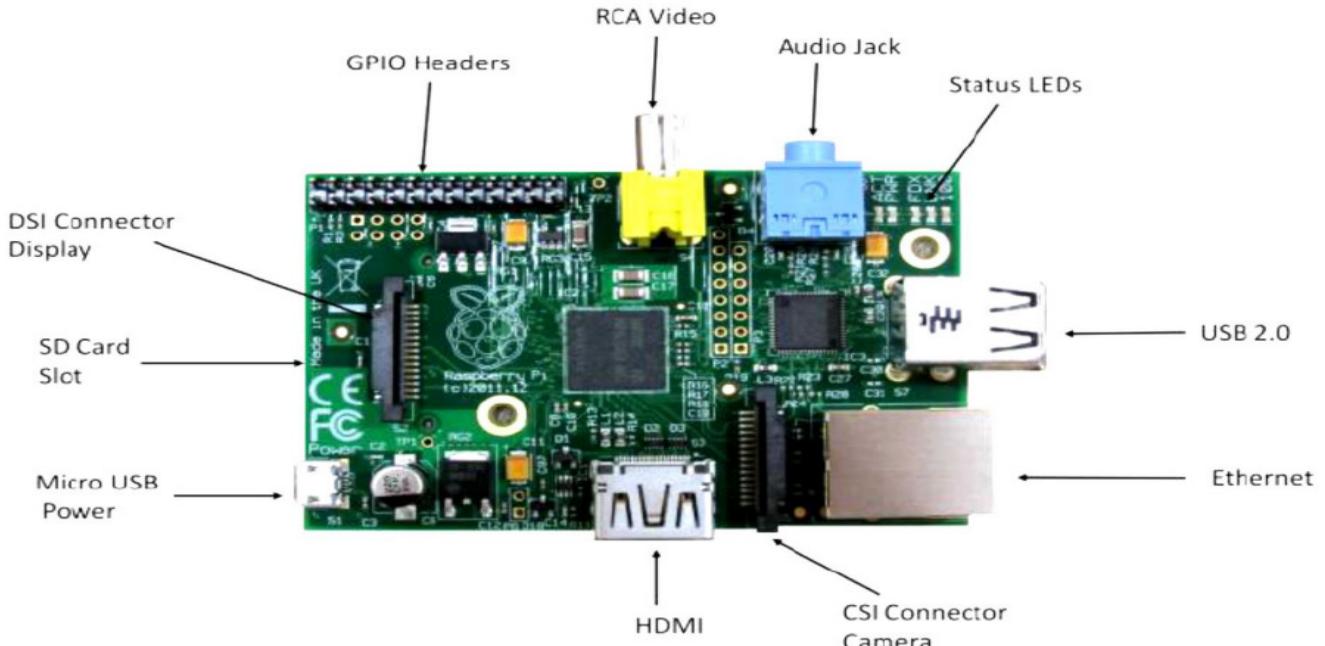
Following are the primary stages (layers) of IoT that provides the solution for IoT architecture.

1. **Sensors/Actuators:** Sensors or Actuators are the devices that are able to emit, accept and process data over the network. These sensors or actuators may be connected either through wired or wireless. This contains GPS, Electrochemical, Gyroscope, RFID, etc. Most of the sensors need connectivity through sensors gateways. The connection of sensors or actuators can be through a Local Area Network (LAN) or Personal Area Network.
2. **Gateways and Data Acquisition:** As the large numbers of data are produced by this sensors and actuators need the high-speed Gateways and Networks to transfer the data. This network can be of type Local Area Network (LAN such as WiFi, Ethernet, etc.), Wide Area Network (WAN such as GSM, 5G, etc.).
3. **Edge IT:** Edge in the IoT Architecture is the hardware and software gateways that analyze and pre-process the data before transferring it to the cloud. If the data read from the sensors and gateways are not changed from its previous reading value then it does not transfer over the cloud, this saves the data used.
4. **Data center/ Cloud:** The Data Center or Cloud comes under the Management Services which process the information through analytics, management of device and security controls. Beside this security controls and device management the cloud transfer the data to the end users application such as Retail, Healthcare, Emergency, Environment, and Energy, etc.

Exemplary Device: 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



Raspberry Pi

The history of the Raspberry Pi was basically introduced in 2006. Its main concept is based on Atmel ATmega644 which is particularly designed for educational use and intended for Python. A Raspberry Pi is of small size i.e., of credit-card-sized single-board computer, which is developed in the United Kingdom(U.K) by a foundation called Raspberry Pi. The main motto of this foundation is to promote the teaching of basic computer science in the education institutes and also in developing countries. The first generation of Raspberry (Pi 1) was released in the year 2012, which has two types of models namely model A and model B.

In the subsequent year, A+ and B+ models were released. Again in 2015, Raspberry Pi2 model B was released and an immediate year Raspberry Pi3 model B was released in the market.

Raspberry Pi can be plugged into a TV, computer monitor, and it uses a standard keyboard and mouse. It is user-friendly as it can be handled by all the age groups. It does everything you would expect a desktop computer to do like word-processing, browsing the internet spreadsheets, playing games to playing high definition videos. It is used in many applications like in a wide array of digital maker projects, music machines, parent detectors to the weather station and tweeting birdhouses with infrared cameras.

All models feature on a Broadcom system on a chip (SOC), which includes chip graphics processing unit GPU(a Video Core IV), an ARM-compatible and CPU. The CPU speed ranges from 700 MHz to 1.2 GHz for the Pi 3 and onboard memory range from 256 MB to 1 GB RAM. An operating system is stored in the secured digital SD cards and program memory in either the MicroSDHC or SDHC sizes. Most boards have one to four USB slots, composite video output, HDMI and a 3.5 mm phone jack for audio. Some models have WiFi and Bluetooth.

The Raspberry Pi Foundation provides Arch Linux ARM and Debian distributions for download, and promotes Python as the main programming language, with support for BBC BASIC, Java, C, Perl, Ruby, PHP, Squeak Smalltalk, C++, etc. The following are essential to get started

- Video cable to suit the TV or monitor used
- SD card containing Linux Operating system
- Power supply (see Section 1.6 below)
- USB keyboard
- TV or monitor (with DVI, HDMI, Composite or SCART input)

Internet Gateway Device

Internet Gateway Device has the ability to route data approaching from the WSN network to the internet and Send data coming from the internet to the WSN network. It is like a Wi-Fi router for the Internet of Things. In the internet gateway device, we use raspberry pi model B, it features a quad-core ARM Cortex- A7 CPU is running at 900MHz (for

a 6x presentation improve on the first generation Raspberry Pi Model B+) and 1GB of LPDDR2 SDRAM (for a 2x memory increase). And yes, there is total compatibility with Raspberry Pi1 we are secured. Broadcom's new SoC, the BCM2836, is the key factor.

Five steps we are using Internet Gateway Device

- Port Linux operating system on Raspberry Pi
- Modify Linux to work with Our Prototype
- Developed Python Library for Communication of RPI with Xbee ZB
- Wrote Program from sensors and Device controlling
- Create WI-FI functionality on RPI for Internet Connection

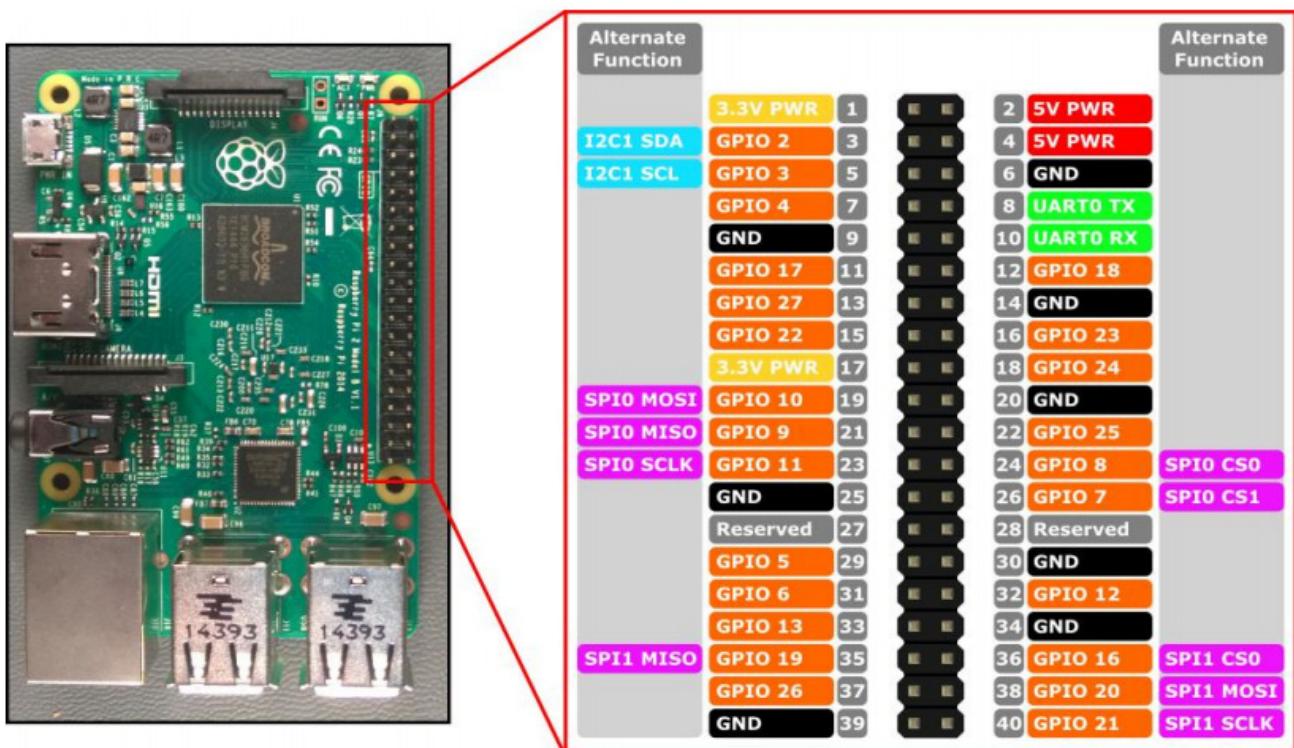
WSN Nodes

A wireless sensor network (WSN) consists of three main components: nodes, gateways, and software. The spatially dispersed measurement nodes interface with the sensors to monitor assets or their surroundings. The acquired information is wirelessly transmitted to the gateway, which provides a connection to the wired globe where you can collect, procedure, analyze, and present your measurement information using the software. Routers are an individual type of dimension node that you can use to expand the distance and dependability in a WSN. Sensors can be dispersed on the roads, vehicles, hospitals, buildings, people and allow dissimilar applications such as medical services, battlefield operations, disaster response, disaster relief, and environmental monitoring.

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



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).

Raspberry Pi Example:

Interfacing LED and switch with Raspberry Pi

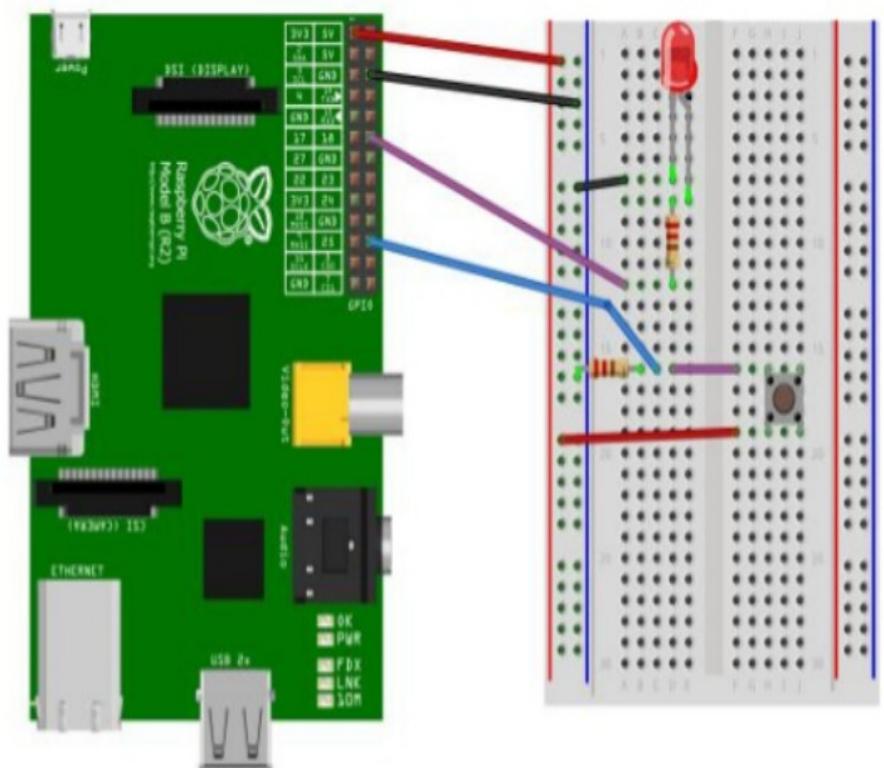
```
from time import sleep
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
```

```
#Switch Pin
GPIO.setup(25, GPIO.IN)
```

```
#LED Pin
GPIO.setup(18, GPIO.OUT)
state=False
```

```
def toggleLED(pin):
    state = not state
    GPIO.output(pin, state)
```

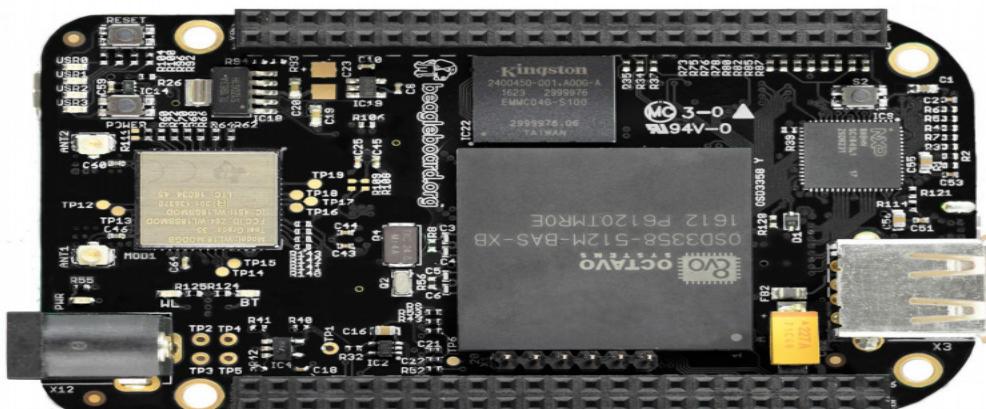
```
while True:
    try:
        if (GPIO.input(25) == True):
            toggleLED(18)
            sleep(.01)
    except KeyboardInterrupt:
        exit()
```



Other Devices

- pcDuino
- BeagleBone Black
- Cubieboard

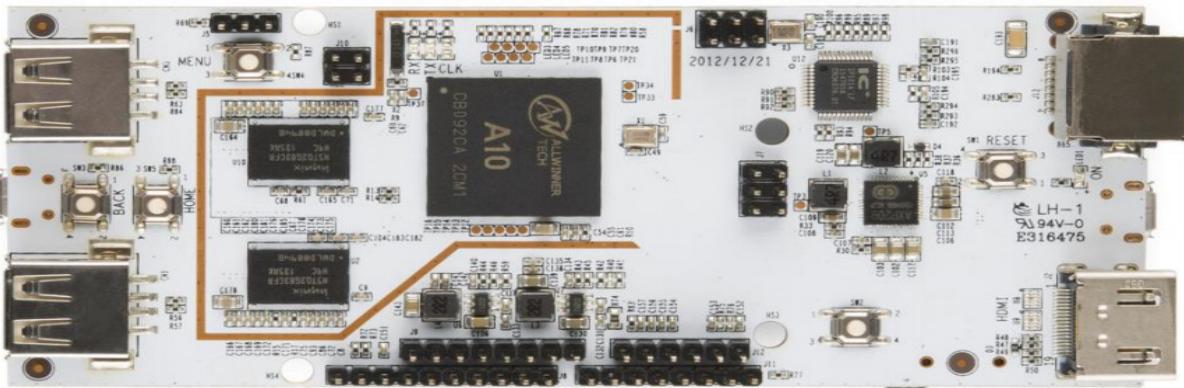
BeagleBone Black



It's similar to a Raspberry Pi but It's more powerful, Based on the TI Sitara AM335x, an application processor SoC containing an ARM Cortex-A8 core. You have more pins to control. They recently won "2013 Top Embedded Innovator award".

Chip	TI AM3359
CPU	1 GHz ARM Cortex-A8
GPU	PowerVR SGX530
Memory	512 MB DDR3
Pins	2x 46 pin headers
USB 2.0	USB 2.0 type A host port. Dedicated single mini-USB 2.0 client port (no additional 2-port hub)
Video Output	microHDMI
Audio Output	microHDMI
Onboard Storage	2 GB 8-bit embedded MMC on-board flash version microSD card 3.3 V Supported (No Card Supplied)
Operating System	Linux, Android, Cloud9 IDE on Node.js w/ BoneScript library, plus more
Dimensions	86.40 × 53.3 mm (3.402 × 2.10 in)
Price:	\$45

pcDuino



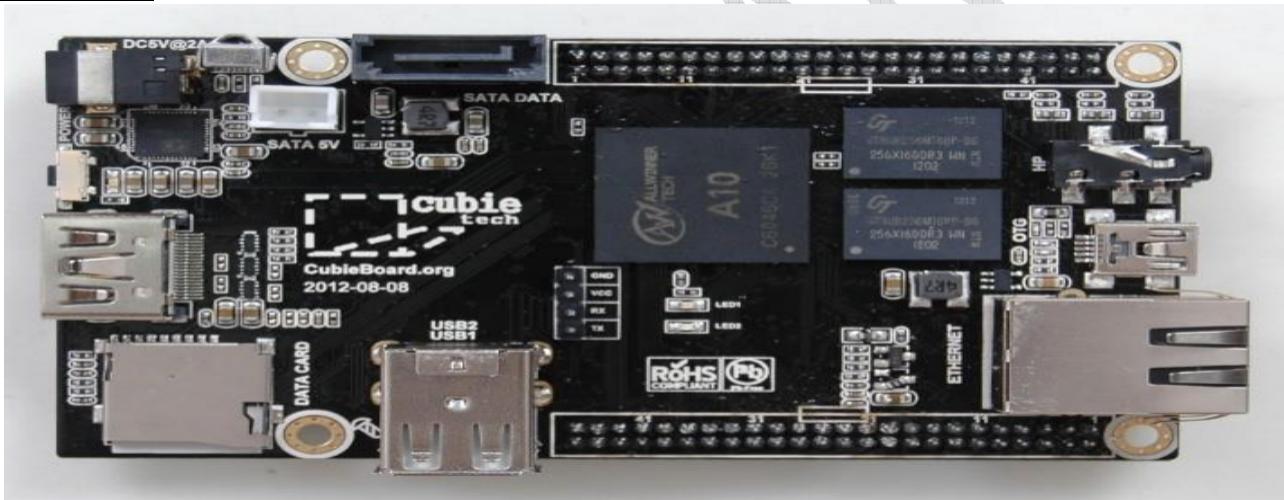
PCDuino = "PC + Arduino"

This board is similar to the Raspberry Pi, it's becoming like the new choice of developers. It has all the features of the Raspberry Pi but it's more powerful. Which ultimately makes it more expensive. It already comes with 2GB of flash memory. And you can add an SD card up to 32GB. Which is a really advantage over the Rpi.

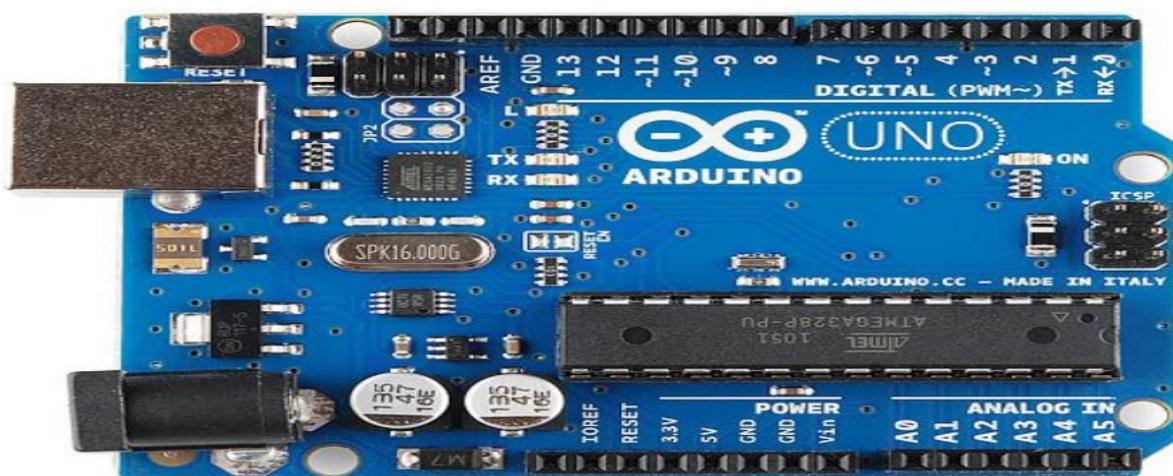
Specifications :

CPU:	1GHz ARM Cortex A8
GPU:	OpenGL ES2.0, OpenVG 1.1 Mali 400 core
RAM:	1GB
Onboard Storage:	2GB Flash, SD card slot for up to 32GB
Video Output:	HDMI
OS:	Linux + Android
Extension Interface:	2.54 mm Headers compatible with Arduino
Network interface:	RJ45 and USB WiFi Dongle
Price:	59 \$

Cubieboard



Arduinno



Arduino

Arduino devices are the microcontrollers and microcontroller kit for building digital devices that can be sense and control objects in the physical and digital world. Arduino boards are furnished with a set of digital and analog input/output pins that may be interfaced to various other circuits. Some Arduino boards include USB (Universal Serial Bus) used for loading programs from the personal computer.

The Arduino is simply perfect for electronics projects and prototyping. You can easily connect some LED's, sensors, motors into the board directly. With their user friendly board it's easy to do that. To program the Arduino you need their software (that can be downloaded for free here). Basically with that software you can upload your source code directly into your Arduino through USB.

After you upload the Arduino code you can unplug the USB cable and attach a battery to your Arduino Board and it will run your program forever. (Read 5 ways to Power Up your Arduino).

The heart of the Arduino is ATMega328P microcontroller.

The most common board is the Arduino UNO. But you can choose a wide range of options. See [here](#) more options.

Specifications:

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz
Price:	\$20

Raspberry Pi

On the other hand The Raspberry Pi is a complete mini computer. It needs an operating system to work. All the storage is provided from a SD card. You can connect this to your network with an Ethernet Cable.



Below I'll show the specifications for the Model B. The brain of the Pi is a ARM1176JZF-S 700 MHz. It has graphics it has a HDMI output. You can plug in a keyboard and monitor, load up Linux, and the less technically savvy might have no clue how tiny the machine driving everything really is. The Pi is an incredibly powerful platform in a very small package it's credit card sized and perfect for embedded systems, or projects requiring more interactivity and processing power.

Specifications:

Chip	Broadcom BCM2835 SoC full HD multimedia applications processor
CPU	700 MHz Low Power ARM1176JZ-F Applications Processor
GPU	Dual Core VideoCore IV® Multimedia Co-Processor
Memory	512MB SDRAM
Ethernet	onboard 10/100 Ethernet RJ45 jack
USB 2.0	Dual USB Connector
Video Output	HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC)
Audio Output	3.5mm jack, HDMI
Onboard Storage	SD, MMC, SDIO card slot
Operating System	Linux
Dimensions	8.6cm x 5.4cm x 1.7cm
Price:	\$35

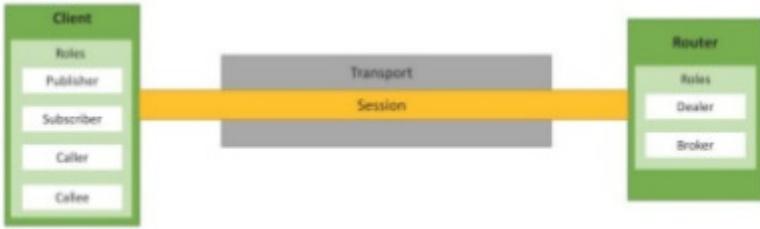
IoT Physical Servers & Cloud Offerings

Outline

- WAMP - AutoBahn for IoT
- Python for Amazon Web Services
- Python for MapReduce
- Python Packages of Interest
- Python Web Application Framework - Django
- Development with Django

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 - Concepts

- 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 the payload.

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 the topic.

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).

Amazon EC2 – Python Example

- Boto is a Python package that provides interfaces to Amazon Web Services (AWS)
- In this example, a connection to EC2 service is first established by calling `boto.ec2.connect_to_region`.
- The EC2 region, AWS access key and AWS secret key are passed to this function. After connecting to EC2 , a new instance is launched using the `conn.run_instances` function.
- The AMI-ID, instance type, EC2 key handle and security group are passed to this function

```
#Python program for launching an EC2 instance
import boto.ec2
from time import sleep
ACCESS_KEY=<enter access key>
SECRET_KEY=<enter secret key>

REGION="us-east-1"
AMI_ID = "ami-d0f89fb9"
EC2_KEY_HANDLE = "<enter key handle>"
INSTANCE_TYPE="t1.micro"
SECURITY_GROUP_HANDLE="default"

conn = boto.ec2.connect_to_region(REGION, aws_access_key_id=ACCESS_KEY,
                                 aws_secret_access_key=SECRET_KEY)

reservation = conn.run_instances(image_id=AMI_ID, key_name=EC2_KEY_HANDLE,
                                  instance_type=INSTANCE_TYPE,
                                  security_groups = [ SECURITY_GROUP_HANDLE, ] )
```

Amazon AutoScaling – Python Example

- **AutoScaling Service**
 - A connection to AutoScaling service is first established by calling `boto.ec2.autoscale.connect_to_region` function.
- **Launch Configuration**
 - After connecting to AutoScaling service, a new launch configuration is created by calling `conn.create_launch_configuration`. Launch configuration contains instructions on how to launch new instances including the AMI-ID, instance type, security groups, etc.
- **AutoScaling Group**
 - After creating a launch configuration, it is then associated with a new AutoScaling group. AutoScaling group is created by calling `conn.create_auto_scaling_group`. The settings for AutoScaling group such as the maximum and minimum number of instances in the group, the launch configuration, availability zones, optional load balancer to use with the group, etc.

```
#Python program for creating an AutoScaling group (code excerpt)
import boto.ec2.autoscale
:
print "Connecting to Autoscaling Service"
conn = boto.ec2.autoscale.connect_to_region(REGION,
aws_access_key_id=ACCESS_KEY,
aws_secret_access_key=SECRET_KEY)

print "Creating launch configuration"

lc = LaunchConfiguration(name='My-Launch-Config-2',
                        image_id=AMI_ID,
                        key_name=EC2_KEY_HANDLE,
                        instance_type=INSTANCE_TYPE,
                        security_groups = [ SECGROUP_HANDLE, ])
conn.create_launch_configuration(lc)

print "Creating auto-scaling group"

ag = AutoScalingGroup(group_name='My-Group',
                      availability_zones=['us-east-1b'],
                      launch_config=lc, min_size=1, max_size=2,
                      connection=conn)
conn.create_auto_scaling_group(ag)
```

Amazon S3 – Python Example

- In this example, a connection to S3 service is first established by calling `boto.connect_s3` function.
- The `upload_to_s3_bucket_path` function uploads the file to the S3 bucket specified at the specified path.

```
# Python program for uploading a file to an S3 bucket
import boto.s3

conn=boto.connect_s3(aws_access_key_id='<enter>',
aws_secret_access_key='<enter>')

def percent_cb(complete, total):
    print ('.')

def upload_to_s3_bucket_path(bucketname, path, filename):
    mybucket = conn.get_bucket(bucketname)
    fullkeyname=os.path.join(path,filename)
    key = mybucket.new_key(fullkeyname)
    key.set_contents_from_filename(filename, cb=percent_cb, num_cb=10)
```

Amazon RDS – Python Example

- In this example, a connection to RDS service is first established by calling boto.rds.connect_to_region function.
- The RDS region, AWS access key and AWS secret key are passed to this function.
- After connecting to RDS service, the conn.create_dbinstance function is called to launch a new RDS instance.
- The input parameters to this function include the instance ID, database size, instance type, database username, database password, database port, database engine (e.g. MySQL5.1), database name, security groups, etc.

```
#Python program for launching an RDS instance (excerpt)
import boto.rds

ACCESS_KEY=<enter>
SECRET_KEY=<enter>
REGION="us-east-1"
INSTANCE_TYPE="db.t1.micro"
ID = "MySQL-db-instance-3"
USERNAME = 'root'
PASSWORD = 'password'
DB_PORT = 3306
DB_SIZE = 5
DB_ENGINE = 'MySQL5.1'
DB_NAME = 'mytestdb'
SECGROUP_HANDLE="default"

#Connecting to RDS
conn = boto.rds.connect_to_region(REGION,
 aws_access_key_id=ACCESS_KEY,
 aws_secret_access_key=SECRET_KEY)

#Creating an RDS instance
db = conn.create_dbinstance(ID, DB_SIZE, INSTANCE_TYPE,
 USERNAME, PASSWORD, port=DB_PORT, engine=DB_ENGINE,
 db_name=DB_NAME, security_groups = [ SECGROUP_HANDLE, ] )
```

Amazon DynamoDB – Python Example

- In this example, a connection to DynamoDB service is first established by calling boto.dynamodb.connect_to_region.
- After connecting to DynamoDB service, a schema for the new table is created by calling conn.create_schema.
- The schema includes the hash key and range key names and types.
- A DynamoDB table is then created by calling conn.create_table function with the table schema, read units and write units as input parameters.

```
# Python program for creating a DynamoDB table (excerpt)
import boto.dynamodb

ACCESS_KEY=<enter>
SECRET_KEY=<enter>
REGION="us-east-1"

#Connecting to DynamoDB
conn = boto.dynamodb.connect_to_region(REGION,
 aws_access_key_id=ACCESS_KEY,
 aws_secret_access_key=SECRET_KEY)

table_schema = conn.create_schema(
    hash_key_name='msgid',
    hash_key_proto_value=str,
    range_key_name='date',
    range_key_proto_value=str
)

#Creating table with schema
table = conn.create_table(
    name='my-test-table',
    schema=table_schema,
    read_units=1,
    write_units=1
)
```

Python for MapReduce

- The example shows inverted index mapper program.
- The map function reads the data from the standard input (stdin) and splits the tab-delimited data into document-ID and contents of the document.
- The map function emits key-value pairs where key is each word in the document and value is the document-ID.

#Inverted Index Mapper in Python

```
#!/usr/bin/env python
import sys
for line in sys.stdin:
    doc_id, content = line.split("\t")
    words = content.split()
    for word in words:
        print '%s%s' % (word, doc_id)
```

- The example shows inverted index reducer program.
- The key-value pairs emitted by the map phase are shuffled to the reducers and grouped by the key.
- The reducer reads the key-value pairs grouped by the same key from the standard input (stdin) and creates a list of document-IDs in which the word occurs.
- The output of reducer contains key value pairs where key is a unique word and value is the list of document-IDs in which the word occurs.

#Inverted Index Reducer in Python

```
#!/usr/bin/env python
import sys
current_word = None
current_docids = []
word = None

for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # parse the input we got from mapper.py
    word, doc_id = line.split("\t")
    if current_word == word:
        current_docids.append(doc_id)
    else:
        if current_word:
            print '%s%s' % (current_word, current_docids)
        current_docids = []
        current_docids.append(doc_id)
        current_word = word
```

Python Web Application Framework - Django

- 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.
- Django is based on the Model-Template-View architecture and provides a separation of the 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 Architecture

- Django is 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.

Visualization in IOT.

Internet of things (IoT) is about capturing and making sense of data from the physical world. Soon, every physical device will be a source of data and this data can potentially become a source of business value. We have all heard the predictions about the explosion of IoT data and the IoT data deluge. All of this IoT data is useless unless it is collected, understood and used to make smart decisions.

Real-time visualization helps improve response and event management by aggregating numerous IoT data streams from various systems, sensors, vehicles and video, providing an integrated operational view across large physical environments – like a sports stadium. Live Earth is just such a platform.

For IoT data streams to be useful and productive, visualization systems must execute the following:

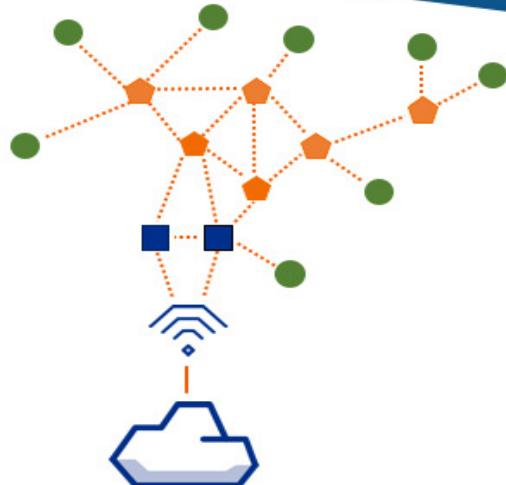
- Develop situational awareness by presenting data streams in context within a physical environment making data actionable
- Create location-based intelligence by presenting multiple concurrent data streams
- Support real-time decision making by combining multiple data sources into a single multi-layered visual display
- Enable multiple data correlations to be viewed and analyzed in real time
- Combine new IoT data with existing data to discover new business patterns and trends
- Monitor IoT devices and infrastructure to ensure stable and proper operation, ensuring IoT data flows continuously without interruption

What is a wireless mesh network?

A wireless mesh network is an infrastructure of nodes (a mesh topology) that are wirelessly connected to each other. These nodes piggyback off each other to extend a radio signal (like a Wi-Fi or cellular connection) to route, relay, and proxy traffic to/from clients. Each node spreads the radio signal a little further than the last, minimizing the possibility of dead zones.

IoT devices play different roles within a mesh network

- Endpoints**
Mesh-only devices that do not route messages for other devices in the mesh network
- Repeaters**
Devices that forward messages between endpoints in a mesh network
- Gateways**
Devices that have additional connectives beyond Mesh to pass messages



What are the components of a mesh network?

- Gateway** — Border routers are the devices that have additional connectivities beyond mesh that allow them to pass messages between networks. You can think of these devices as providing a “backhaul” to the internet for the local mesh network.
- Repeater** — Routers are devices that forward messages between end devices (endpoints) in a mesh network. They are not typically designed to sleep because they are a part of the mesh networks’ infrastructure.
- Endpoint** — End devices are mesh-only devices that do not route messages for other devices in the mesh network. Because they have no networking responsibilities, they can enter sleep mode and are good candidates for battery-powered nodes and sensors.

How does mesh networking and IoT work together?

It should be noted that each mesh networking solution works differently. So for this article, will focus on how Particle Mesh technology works. Particle Mesh is a wireless mesh network designed to connect the spaces in between existing Wi-Fi and cellular deployments with local networks that are low-cost, secure, and reliable.



Particle Mesh Hardware — Argon, Boron, and Xenon

Traditional IoT devices that use Wi-Fi and cellular connectivity depend on the cloud to relay messages between devices. This works great when you're making a standalone product — but sometimes you need more than that. Particle Mesh development kits aren't just connected to the Internet, they're gateways to the Internet and create a local wireless mesh that other devices can join. These devices work together to ensure that messages get where they're going, and power products that aren't possible or economically feasible with Wi-Fi and cellular connectivity. Particle Mesh gives every IoT device a local network to understand and connect with the world around it, ensuring products have the information they need.

Why use mesh networking for IoT?

While wireless mesh networking technologies has been around for some time, only recently has the power of mesh reached a point of maturity alongside high availability from chip and silicon vendors. With newer approachable costs, wireless mesh networking has become ideal for IoT builders. And with the rise of connected homes and industry support on open source resources like Thread, Mesh is now truly accessible while being low-cost enough to scale for production. As such, wireless mesh networking is becoming a much more viable, real choice for industrial and commercial IoT applications. It can provide additional services in a system where extending a connection between two nodes is limited:

1. **Smart Cities** — Wireless mesh networking is great for extending radio signals through parking garages, campus grounds, business parks, and other outdoor facilities. Parking garages that utilize space availability checkers benefit greatly from mesh networks because they can extend the signal throughout the whole space, and be able to communicate when a spot has been taken by other clients.
2. **Healthcare Equipment** — Wireless mesh networks can help monitor and locate medical devices quickly. They can also act as a backup for medical equipment that always needs to remain online. If one node loses connectivity, another node can step in to keep the connection alive.
3. **Smart Home** — Wireless mesh networks can help you track and manage temperatures across your house. Setup one powered gateway and use temperature sensors and Mesh-enabled nodes in each room to capture live data and adjust settings automatically.
4. **Farming** — Wireless mesh networking is also great for tracking sun exposure and water levels across your crops. You can scale at a low cost with Mesh-enabled nodes across a whole acreage to create a cellular-connected IoT farm.
5. **Industrial Internet** — Wireless mesh networking is also great for tracking pallets and monitoring large physical objects with a highly reliable wireless connectivity network. With wireless mesh networks, you can easily track key data across your factory floor, and across multiple locations to identify issues before they happen.

Is wireless mesh networking right for you?

When using wireless mesh networks for your IoT project, it is important that you consider these three core variables: **installation**, **device management**, and **support**.

1. **Installation** — This aspect entirely depends upon your intended application. You need to ask yourself if you actually needed a distributed set of mesh nodes for your use case. If you intend to implement mesh for commercial or industrial applications, you should setup a small-scale, mesh network to determine the efficiency of the system before deploying a mesh networking system at large.
2. **Device Management** — When comparing mesh solutions, it's important to find one that allows you to manage fleets of devices, monitor event logs, perform diagnostics and send updates wirelessly.
3. **Support** — When selecting a mesh-solution, it's also important to consider the community surrounding it. Mesh networking solutions with limited adoption will have fewer resources available to aid you in development.

Looking beyond mesh networking

If you're looking to implement wireless mesh-networking into your IoT infrastructure, you must examine the whole IoT system and not just a singular component. To build any IoT product or infrastructure you need hardware, software, and connectivity. To integrate these three components, you must research IoT platforms that can provide these components to you and consult IoT domain experts to help you scope these the three complexities.

Bluetooth Low Energy

Bluetooth Low Energy is the intelligent, power-friendly version of Bluetooth wireless technology. It is already playing a significant role in transforming smart gadgets to smarter gadgets by making them compact, affordable, and less complex.

Bluetooth Low Energy, also marketed as Bluetooth Smart, started as part of the Bluetooth 4.0 Core Specification. Initially designed by Nokia as Wibree before being adopted by the Bluetooth Special Interest Group (SIG), its initial focus was to provide a radio standard with the lowest possible power consumption, specifically optimized for low cost, low bandwidth, low power, and low complexity.

These design goals are evident through the core specification, which attempts to make BLE a genuine low-power standard, designed to actually be implemented by semiconductor manufacturers and used in real-world applications tight on energy with minimal budget. It is already a widely adopted technology that can realistically stake claim to run for an extended period of time off a single coin cell.

While BLE is a superior technology on its own merit, what has driven its phenomenal adoption rate is that it is the right technology, with the right compromises, at the right time. For a relatively young standard, the number of product designs that already include BLE puts it well ahead of other wireless technologies at the same point in their release cycles.

The challenges classic Bluetooth faced were fast battery draining and frequent loss of connection, requiring frequent pairing and re-pairing. Being able to successfully address these is one of the reasons for BLE's rapid growth. Further driving adoption is the phenomenal growth in smartphones, tablets, and mobile computing. Early and active adoption of BLE by mobile industry heavyweights broke open the doors for wider implementation of BLE. This in turn has pushed semiconductor manufacturers to commit their limited resources to the technology they felt was the most likely to flourish in the long run.

While the mobile and tablet markets have become increasingly mature, the need for connectivity between the outside world and these devices has a huge growth potential. It offers peripheral vendors a unique opportunity to develop innovative devices that solve problems consumers might not even realize that they have today. So many benefits have converged around BLE, creating an opportunity for small and nimble product designers to gain access to a potentially massive market with task-specific, creative, and innovative products on a relatively modest design budget. BLE also allows these developers to design viable products today that can talk to any modern mobile platform using chips, tools, and standards that are easy to access.

Physical Layer

Bluetooth Low Energy uses 80 MHz of spectrum in the 2.4 GHz ISM band to provide low-power, low-rate wireless communications over a range of 10 to 1,000 metres (depending on environment/configuration). Gaussian frequency-shift keying (GFSK) modulation is used with a maximum transmit power of 20 dBm, and together the link and physical layers implement a frequency hopping scheme that provides spread spectrum resilience to narrow-band interferers by working over 40 channels that are each 2 MHz wide.

The Bluetooth Low Energy physical layer provides four transmission modes with bit rates ranging between 125 Kbps and 2 Mbps, although packet length limitations and protocol overheads mean that maximum application throughput will be lower than PHY bit rate by 20-60% depending on protocol features available/enabled.

Communication Topologies

Bluetooth Low Energy has network topologies that are geared toward ad hoc communication scenarios involving relatively-infrequent transfer of small quantities of data. The technology can also support connection-oriented use cases such as those involving virtual serial ports or human interface devices.

The Bluetooth specification defines four roles that a Bluetooth Low Energy device may take: *broadcaster*, *observer*, *peripheral* and *central*.

- Broadcaster devices are transmit-only, and periodically broadcast advertising packets that may be detected by devices acting in the observer role.
- Observer devices are the counterpart of broadcasters - they are receive-only and listen for advertisements from broadcaster devices.
- Peripheral devices initially act like broadcasters, but transmit *connectable* advertising packets and accept connections from central devices.
- Central devices initiate connections to peripherals by listening for connectable advertising packets and then exchanging packets with the peripheral device.

Bluetooth Low Energy peripheral devices may only be connected to a single central device at a time. Central devices, on the other hand, can have multiple connected peripherals if required. Many common Bluetooth Low Energy usage scenarios see a smartphone acting in the role of central, with sensors, wearables, and other "things" connected as peripherals.

Taken individually the four roles directly cater for the majority of Bluetooth Low Energy application scenarios, but many transceivers available provide further flexibility by implementing time-slicing mechanisms to allow simultaneous operation in more than one role.

Upper Layers

Bluetooth Low Energy introduces a protocol and a profile that work hand-in-hand to provide the basis for all Bluetooth Low Energy applications. These are the *Attribute Protocol* and the *Generic Attribute Profile*, respectively, and together they allow applications to expose state variables to a peer, and query or manipulate state variables of a peer.

The Attribute Protocol (ATT) defines server and client roles, and the basic operations that can be performed on state in the attribute database of the server. For example:

- A client may discover the attributes available on the server.
- A client may read the value of an attribute from the server.
- A client may write a value to an attribute on the server (with variants requiring and not requiring ATT-layer acknowledgement from the server).
- A server may send the value of an attribute to the client (with variants requiring and not requiring ATT-layer acknowledgement from the client).

Building on the facilities provided by ATT, the Generic Attribute Profile (GATT) defines the structure of the attribute database. At the GATT layer (and thus from the point of view of the application) the database on the attribute server is made up of *characteristics* which are grouped to form *services*. These characteristics and services are each identified by universally unique identifiers (UUIDs), which are themselves stored as attributes in the database. This enables client applications to recognise and access services that they know how to make use of.

Above the GATT layer lie profiles that may either be vendor-specific or defined by the Bluetooth SIG. These profiles define the roles, services and characteristics that must be implemented by GATT and ATT layers to provide functionality for a particular application.

A number of standard profiles are available for a diverse range of applications including blood pressure monitoring, location and navigation, proximity detection, and time synchronisation. There is ongoing effort within the Bluetooth SIG to specify further GATT profiles so that a wider range of applications can enjoy the interoperability benefits that arise from a qualified device implementing a standard profile.

Security

Bluetooth Low Energy has comprehensive security features that work across layers to address the privacy, authenticity, and integrity of user data.

At the link layer, Bluetooth Low Energy makes use of the AES-128 block cipher to encrypt data transmitted over the air. While near-universally supported by implementations, use of this encryption is at the discretion of the application. In cases where privacy is not essential, authenticity and integrity of data sent over unencrypted connections can still be protected through use of a cipher-based message authentication code (CMAC; again, based on AES-128).

With wearables and personal devices being a key application for Bluetooth Low Energy, it is important to limit the ability of unauthorised third parties to track individuals or assets. To address this aspect of privacy Bluetooth Low Energy includes a feature that allows devices to change their Bluetooth device address frequently. To a third-party device this "private" address is essentially random, but it can still be resolved by trusted peer devices that might wish to initiate or accept connections.

Features

1. The lowest power consumption

Everything from physical design to use models is designed to keep power consumption at a minimum. To reduce power consumption, a BLE device is kept in sleep mode most of the time. When an event occurs, the device wakes and a short message is transferred to a gateway, PC, or smartphone. Maximum/peak power consumption is less than 15 mA and the average power consumption is about 1 μ A. The active power consumption is reduced to a tenth of the energy consumption of classic Bluetooth. In low duty cycle applications, a button cell battery could provide 5-10 years of reliable operation.

2. Cost efficient and compatible

To offer compatibility with classic Bluetooth technology and cost efficiency for small battery-operated devices, there are two chipset types:

- Dual-mode technology with both BLE and classic Bluetooth functionality
- Stand-alone BLE technology optimized for small battery-operated devices with low cost and low power consumption as their focus

3. Robustness, security, and reliability

BLE technology uses the same adaptive frequency hopping (AFH) technology as classic Bluetooth technology. This enables BLE to achieve robust transmission in the 'noisy' RF environments found in the home, industrial, and medical applications. To minimize the cost and energy consumption of using AFH, BLE technology has reduced the number of channels to 40 2-MHz wide channels instead of the 79 1-MHz wide channels used with classic Bluetooth technology.

4. Wireless co-existence

Bluetooth technology, Wireless LAN, IEEE 802.15.4/ZigBee, and several proprietary radios use the license-free 2.4GHz Industrial Scientific Medical (ISM) band. With so many technologies sharing the same radio space, interference can decrease wireless performance (i.e., increasing latency and decreasing throughput) due to the need for error correction and retransmission. In demanding applications, interference can be reduced through frequency planning and special antenna design. As both classic Bluetooth technology and BLE technology utilize AFH, which minimizes interference with other radio technologies, Bluetooth transmission is robust and reliable.

5. Connection range

BLE technology has a slightly different modulation than classic Bluetooth technology. This modulation differentiation offers a range of up to 300 meters with a 10 dBm radio chipset (BLE maximum).

6. Ease of use and integration

A BLE piconet is typically based on a master connected to a number of slaves. A device is either a master or a slave, but never both. The master controls how often the slaves are allowed to communicate, and the slave only communicates by request from the master. A new feature BLE adds compared to classic Bluetooth technology is "advertising" functionality. With this feature, a device acting as a slave can announce that it has something to transmit to the master. An advertisement message can also include an event or a measurement value.

Data-intensive

Data-intensive computing is a class of parallel computing applications which use a data parallel approach to process large volumes of data typically terabytes or petabytes in size and typically referred to as big data. Computing applications which devote most of their execution time to computational requirements are deemed compute-intensive, whereas computing applications which require large volumes of data and devote most of their processing time to I/O and manipulation of data are deemed data-intensive.

Data-intensive is used to describe applications that are I/O bound or with a need to process large volumes of data. Such applications devote most of their processing time to I/O and movement and manipulation of data. Parallel processing of data-intensive applications typically involves partitioning or subdividing the data into multiple segments which can be processed independently using the same executable application program in parallel on an appropriate computing platform, then reassembling the results to produce the completed output data. The greater the aggregate distribution of the data, the more benefit there is in parallel processing of the data. Data-intensive processing requirements normally scale linearly according to the size of the data and are very amenable to straightforward parallelization. The fundamental challenges for data-intensive computing are managing and processing exponentially growing data volumes, significantly reducing associated data analysis cycles to support practical, timely applications, and developing new algorithms which can scale to search and process massive amounts of data. Researchers coined the term BORPS for "billions of records per second" to measure record processing speed in a way analogous to how the term MIPS applies to describe computers' processing speed

Data-parallelism

Computer system architectures which can support data parallel applications were promoted in the early 2000s for large-scale data processing requirements of data-intensive computing.^[12] Data-parallelism applied computation independently to each data item of a set of data, which allows the degree of parallelism to be scaled with the volume of data. The most important reason for developing data-parallel applications is the potential for scalable performance, and may result in several orders of magnitude performance improvement. The key issues with developing applications using data-parallelism are the choice of the algorithm, the strategy for data decomposition, load balancing on processing nodes, message passing communications between nodes, and the overall accuracy of the results

Characteristics

Several common characteristics of data-intensive computing systems distinguish them from other forms of computing:

1. The principle of collection of the data and programs or algorithms is used to perform the computation. To achieve high performance in data-intensive computing, it is important to minimize the movement of data.^[19] This characteristic allows processing algorithms to execute on the nodes where the data resides reducing system overhead and increasing performance.^[20] Newer technologies such as InfiniBand allow data to be stored in a separate repository and provide performance comparable to collocated data.

2. The programming model utilized. Data-intensive computing systems utilize a machine-independent approach in which applications are expressed in terms of high-level operations on data, and the runtime system transparently controls the scheduling, execution, load balancing, communications, and movement of programs and data across the distributed computing cluster.[21] The programming abstraction and language tools allow the processing to be expressed in terms of data flows and transformations incorporating new dataflow programming languages and shared libraries of common data manipulation algorithms such as sorting.
3. A focus on reliability and availability. Large-scale systems with hundreds or thousands of processing nodes are inherently more susceptible to hardware failures, communications errors, and software bugs. Data-intensive computing systems are designed to be fault resilient. This typically includes redundant copies of all data files on disk, storage of intermediate processing results on disk, automatic detection of node or processing failures, and selective re-computation of results.
4. The inherent scalability of the underlying hardware and software architecture. Data-intensive computing systems can typically be scaled in a linear fashion to accommodate virtually any amount of data, or to meet time-critical performance requirements by simply adding additional processing nodes. The number of nodes and processing tasks assigned for a specific application can be variable or fixed depending on the hardware, software, communications, and distributed file system architecture.

System architectures

A variety of system architectures have been implemented for data-intensive computing and large-scale data analysis applications including parallel and distributed relational database management systems which have been available to run on shared nothing clusters of processing nodes for more than two decades. However most data growth is with data in unstructured form and new processing paradigms with more flexible data models were needed. Several solutions have emerged including the Map Reduce architecture pioneered by Google and now available in an open-source implementation called Hadoop used by Yahoo, Facebook, and others.

MapReduce

The MapReduce architecture allows programmers to use a functional programming style to create a map function that processes a key-value pair associated with the input data to generate a set of intermediate key-value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Since the system automatically takes care of details like partitioning the input data, scheduling and executing tasks across a processing cluster, and managing the communications between nodes, programmers with no experience in parallel programming can easily use a large distributed processing environment.

The programming model for MapReduce architecture is a simple abstraction where the computation takes a set of input key-value pairs associated with the input data and produces a set of output key-value pairs. In the Map phase, the input data is partitioned into input splits and assigned to Map tasks associated with processing nodes in the cluster. The Map task typically executes on the same node containing its assigned partition of data in the cluster. These Map tasks perform user-specified computations on each input key-value pair from the partition of input data assigned to the task, and generates a set of intermediate results for each key. The shuffle and sort phase then takes the intermediate data generated by each Map task, sorts this data with intermediate data from other nodes, divides this data into regions to be processed by the reduce tasks, and distributes this data as needed to nodes where the Reduce tasks will execute. The Reduce tasks perform additional user-specified operations on the intermediate data possibly merging values associated with a key to a smaller set of values to produce the output data. For more complex data processing procedures, multiple MapReduce calls may be linked together in sequence

Hadoop

Apache Hadoop is an open source software project sponsored by The Apache Software Foundation which implements the MapReduce architecture. Hadoop now encompasses multiple subprojects in addition to the base core, MapReduce, and HDFS distributed filesystem. These additional subprojects provide enhanced application

processing capabilities to the base Hadoop implementation and currently include Avro, Pig, HBase, ZooKeeper, Hive, and Chukwa. The Hadoop MapReduce architecture is functionally similar to the Google implementation except that the base programming language for Hadoop is Java instead of C++. The implementation is intended to execute on clusters of commodity processors.

Hadoop implements a distributed data processing scheduling and execution environment and framework for MapReduce jobs. Hadoop includes a distributed file system called HDFS which is analogous to GFS in the Google MapReduce implementation. The Hadoop execution environment supports additional distributed data processing capabilities which are designed to run using the Hadoop MapReduce architecture. These include HBase, a distributed column-oriented database which provides random access read/write capabilities; Hive which is a data warehouse system built on top of Hadoop that provides SQL-like query capabilities for data summarization, ad hoc queries, and analysis of large datasets; and Pig – a high-level data-flow programming language and execution framework for data-intensive computing.

HPCC

HPCC (High-Performance Computing Cluster) was developed and implemented by LexisNexis Risk Solutions. The development of this computing platform began in 1999 and applications were in production by late 2000. The HPCC approach also utilizes commodity clusters of hardware running the Linux operating system. Custom system software and middleware components were developed and layered on the base Linux operating system to provide the execution environment and distributed filesystem support required for data-intensive computing. LexisNexis also implemented a new high-level language for data-intensive computing.