ATME COLLEGE OF ENGINEERING

13th Kilometer, Bannur Road, Mysuru - 570028

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(ACADEMIC YEAR 2026-27)

ODD SEMESTER

NOTES OF LESSON

SUBJECT: INTERNET OF THINGS

SUB CODE: BCS701

SEMESTER: VII

INSTITUTIONAL VISION AND MISSION

Objectives

- To provide quality education and groom top-notch professionals, entrepreneurs and leaders for different fields of engineering, technology and management.
- To open a Training-R & D-Design-Consultancy cell in each department, gradually introduce doctoral and postdoctoral programs, encourage basic & applied research in areas of social relevance, and develop the institute as a center of excellence.
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels.
- To cultivate strong community relationships and involve the students and the staff in local community service.
- To constantly enhance the value of the educational inputs with the participation of students, faculty, parents and industry.

Vision

- Development of academically excellent, culturally vibrant, socially responsible, and globally competent human resources.

Mission

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow's society.
- To strive to attain ever-higher benchmarks of educational excellence.

Vision of the Department

• To develop highly talented individuals in Computer Science and Engineering to deal with real world challenges in industry, education, research and society.

Mission of the Department

• To inculcate professional behavior, strong ethical values, innovative research capabilities and leadership abilities in the young minds & to provide a teaching environment that emphasizes depth, originality and critical thinking.

• Motivate students to put their thoughts and ideas adoptable by industry or to pursue higher studies leading to research.

Program Educational Objectives (PEO'S)

1. Empower students with a strong basis in the mathematical, scientific and engineering fundamentals to solve computational problems and to prepare them for employment, higher learning and R&D.

2. Gain technical knowledge, skills and awareness of current technologies of computer science engineering and to develop an ability to design and provide novel engineering solutions for software/hardware problems through entrepreneurial skills.

3. Exposure to emerging technologies and work in teams on interdisciplinary projects with effective communication skills and leadership qualities.

4. Ability to function ethically and responsibly in a rapidly changing environment by applying innovative ideas in the latest technology, to become effective professionals in Computer Science to bear a life-long career in related areas.

Program Specific Outcomes (PSOs)

PSO1: Ability to apply skills in the field of algorithms, database design, web design, cloud computing and data analytics.

PSO2: Apply knowledge in the field of computer networks for building network and internet-based applications.

Course Code	Course Title	Core / Elective	Prerequisite	Contact Hours			Total Hrs/ Sessions
				L	T	P	
BCS701	Internet of Things	Core	Networking, Basic programming Knowledge	3	0	2	40 Theory + 8/10 Practical sessions

Course Objectives

1. Understand the fundamentals of Internet of Things and its building blocks along with their characteristics.
2. Understand the recent application domains of IoT in everyday life.
3. Understand the protocols and standards designed for IoT and the current research on it.
4. Understand the other associated technologies like cloud and fog computing in the domain of IoT.
5. Improve their knowledge about the various cutting-edge technologies in the field of IoT and machine learning applications.
6. Gain insights about the current trends of machine learning and AI techniques used in IoT to orient towards the present industrial scenario.

Topics Covered as per Syllabus

Module-1

Introduction to Internet of Things: Introduction, Physical design of IOT, Logical Design of IOT, IOT enabling technologies, IOT Levels & Deployment Templates.

Module-2

IOT and M2M: Introduction: M2M, Difference between IoT and M2M, SDN and NFV for IOT, IOT System Management with NETCONF-YANG, Need for IOT Systems Management, Simple Network Management Protocol (SNMP), Network operator requirements, NETCONF, YANG, IoT Systems Management with NETCONF-YANG.

Module-3

IoT Platforms Design Methodology: Introduction, IoT Design Methodology, Case Study on IoT System for Weather Monitoring, loT Systems - Logical Design using Python: Introduction, Installing Python, Python Data Types and Data structures, Control flow, Functions, Modules, Packages, File Handling, Operations, Classes, Python Packages of Interest for IoT.

Module-4

IoT Physical Devices & End points: What is a loT Device, Raspberry Pi, About the Board, Linux on Raspberry Pi, Raspberry Pi interfaces, Programming Raspberry Pi with Python, Case Studies illustrating IoT design – Home Automation, Cities, Agriculture.

Module-5

Data Analytics for IoT: Introduction, Apache Hadoop, Using Hadoop MapReduce for Batch Data

Analytics, Apache Oozie, Apache Spark, Apache Storm, Using Apache Storm for Real-time Data Analysis. **Laboratory Component:** 1. Develop a program to blink 5 LEDs back and forth. 2. Develop a program to interface a relay with Arduino board. 3. Develop a program to deploy an intrusion detection system using Ultrasonic and sound sensors. 4. Develop a program to control a DC motor with Arduino board. 5. Develop a program to deploy smart street light system using LDR sensor. 6. Develop a program to classify dry and wet waste with the Moisture sensor (DHT22). 7. Develop a program to read the pH value of a various substances like milk, lime and water. 8. Develop a program to detect the gas leakage in the surrounding environment. 9. Develop a program to demonstrate weather station readings using Arduino. 10. Develop a program to setup a UART protocol and pass a string through the protocol. 11. Develop a water level depth detection system using Ultrasonic sensor. 12. Develop a program to simulate interfacing with the keypad module to record the keystrokes.	
List of Textbook	
Arshdeep Bahga, Vijay Madisetti, "Internet of Things- A Hands On Approach", Universities press, 2014.	
Course Outcomes	1. Describe the basics of the Internet of Things, including its design, technologies, and different types of deployments. 2. Explain the concepts of IoT and M2M and describe the use of network management protocols 3. Apply basic IoT design steps and programming to create simple IoT applications 4. Describe the architecture and interfaces of Raspberry Pi and implement Python-based IoT applications for different domains 5. Elaborate the need for Data Analytics in IoT.

Module-1

Introduction to Internet of Things

Introduction

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

- Existing devises, such as networked computers or 4G enabled mobile phones, already have some form of unique identities and are also connected to the internet, the focus on IoT in the configuration, control and networking via the internet of devices or things, that are traditionally not associated with the Internet. These include devices such as thermostats, utility meters, a blue tooth- connected headset, irrigation pumps and sensor or control circuits for an electric car's engine
- The scope of IoT is not limited to just connected things (Devices, appliance, machines) to the Internet.
- Applications on IoT networks extract and create information from lower-level data by filtering, processing, categorizing, condensing and contextualizing the data. The information obtained is then organized and structured to infer knowledge about the system and or its user, its environment and its operations and progress towards its objectives, allowing a smarter performance.

Definition of IoT

A dynamic global network infrastructure with self – configuring capabilities based on standard and interoperable communication protocols where physical and virtual "things" have identified, physical attributes, and virtual personalities and use intelligent interfaces, often communicate data associated with users and their environment.

Characteristics of IoT

- Dynamic and self-Adapting: IoT devices and systems may have the capability to dynamically adapt with the changing contexts and take actions based on their operating condition. Ex: Surveillance cameras can adapt their modes based on whether it is day or night.
- Self – Configuring: IoT devices may have self-Configuring capability allowing many devices to work together to provide certain functionality.
- Interoperable communication protocols: IoT Devices may support a few interoperable communication protocols and can communicate with other devices and with the infrastructure.
- Unique Identity: Each IoT devices has a unique identity and a unique identifier. IP address, URI). IoT systems may have intelligent interfaces which adapt based on the context, allow communication with users and the environment contexts.

- Integrated into information network: IoT devices are usually integrated into the information network that allows them to communicate and exchange data with other devices and systems.

Physical design of IoT

1. Things of IoT

The "Things" in IoT usually refers to IoT devices which have unique identities and can perform remote sensing, Actuating and monitoring capabilities. IoT devices can exchange data with other connected devices and applications (directly or indirectly), or collect data from other devices and process the data locally or send the data to Centralized servers or cloud based applications back ends for processing the data or from some task locally and other task within the IoT infrastructure, based on temporal and space constraints (i.e.: Memory, processing calibrators, communication latencies and speed and deadlines).

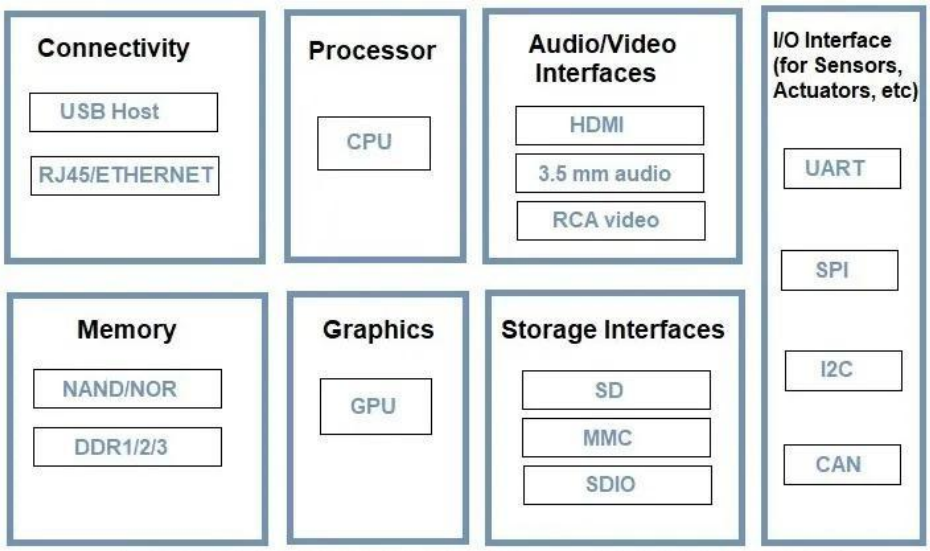

Fig.1.1 Generic block diagram of an IoT device

An IoT device may consist of several interfaces' connections to other devices, both wired and wireless. These include

- IoT interfaces for sensors
- interfaces for internet connectivity
- memory and storage interfaces
- audio video interfaces.

An IoT Device can collect various types of data from the onboard or attached sensors, such as temperature, humidity, light intensity.

IoT devices can also be varied types, for instance, wearable sensors, smart watches, LED light automobiles and industrial machines. Almost all I would advise generate data in Some form or

the other which when processed by Data Analytics systems leads to Useful information to guide further actions locally or remotely.

2. IoT Protocols

Link Layer

Link Layer protocols determine how the data is physically sent over the networks physical layer or medium (example copper wire, electrical cable, or radio wave). The Scope of The Link Layer is the Last Local Network connections to which host is attached. Host on the same link exchange data packets over the link layer using the link layer protocol. Link layer determines how the packets are coded and signalled by the hardware device over the medium to which the host is attached.

- 802.3 Ethernet:

802.3 is a collections of wired Ethernet standards for the link layer. For example, 802.3 10BASE5 Ethernet that uses coaxial cable as a shared medium, 802.3.i is standard for 10 BASET Ethernet over copper twisted pair connection, Standards provide data rates from 10 Mb/s to 40 gigabits per second and the higher. The shared medium in Ethernet can be a coaxial cable, twisted pair wire or and Optical Fiber. Shared medium carries the communication for all the devices on the network.

- 802.1- WI-FI:

IEEE 802.3 is a collections of wireless Local area network (WLAN) communication standards, including extensive descriptions of the link layer. For example, 802.11a operate in the 5 GHz band, 802.11b and 802.11g operate in the 2.4 GHz band. 802.11ac operates in the 5G hertz band.

- 802.16 wiMAX:

IEEE 802.16 is a collection of wireless broadband and Standards, including extensive descriptions for the link layer also called WiMAX. This standard provides a data rates from 1.5Mb/s to 1Gb/s the recent update provides data rates of hundred megabits per second for mobile station.

- 802.15.4 LR-WPAN:

IEEE 802.1 5.4 is a collection of standards for low-rate wireless personal area network (LRWPAN). This standard form the basis of specifications for high level communication Zigbee. LR-WPAN standards provide data rates from 40 k b/ s. These standards provide low cost and low speed Communications for power constrained devices.

- 2G / 3G / 4G mobile communications:

These are the different generations of mobile communication standards including second generation (2G including GSM and CDMA). 3rd Generation (3G including UMTS and CDMA2000) and 4th generation 4G including LTE.

Network / internet layer:

The network layer is responsible for sending of IP datagrams from the source network to the destination network. This layer Performs the host addressing and packet routing. The datagrams contain a source and destination address which are used to route them from the source to the destination across multiple networks. Host Identification is done using the hierarchy IP addressing schemes such as ipv4 or IPv6.

- IPV4: Internet protocol versions for open parents close (IPV4) is there most deployed internet protocol that is used to identify the device is on a network using a hierarchy latest scheme. It uses 32-bit addresses scheme that allows total of 2 32 address. As more and more devices got connected to the internet. The Ipv4 has succeeded by IPv6.
- IPv6: It is the newest versions of internet protocol and successor to IPv4. IPv6 uses 128-bit address schemes that are lost total of 2 128 are 3.4* 10 38 address.
- 6LoWPAN: IPv6 over low power wireless personal area networks brings IP protocol to the low power device which have limited processing capability it operates in the 2.4 GHz frequency range and provide the data transfer rate off to 50 kb/s.

Transport layer:

The Transport layer protocols provide end-to-end message transfer capability independent of the underlying network. The message transfer capability can be set up on connections, either using handshake or without handshake acknowledgements. Provides functions such as error control, segmentation, flow control and congestion control.

- TCP: Transmission control protocol is the most widely used to transport layer protocol that is used by the web browsers along with HTTP, HTTPS application layer protocols email program (SMTP application layer protocol) and file transfer protocol. TCP is a connection Oriented and stateful protocol while IP protocol deals with sending packets, TCP ensures reliable transmissions of packets in order. TCP also provide error deduction capability so that duplicate packets can be discarded and low packets are retransmitted. The flow control capability ensures that the rate at which the sender since the data is now high for the receiver to process.
- UDP: unlike TCP, which requires carrying out an initial setup procedure, UDP is a connection less protocol. UDP is useful for time sensitive application they have very small data units to exchange and do not want the overhead of connection setup. UDP is a transactions oriented and stateless protocol. UDP does not provide guaranteed delivery, ordering of messages and duplicate eliminations.

Application layer:

Application layer protocol defines how the application interfaces with the lower layer protocols to send the data over the network. Data are typically in files, is encoded by the application layer protocol and encapsulated in the transport layer protocol. Application layer protocol enables process-to-process connection using ports.

- Http: Hypertext transfer protocol is the application layer protocol that forms the foundations of world wide web http includes, commands such as GET, PUT, POST, DELETE, HEAD, TRACE, OPTIONS etc. The protocol follows a request response model where are client sends request to server using the http, commands. Http is a stateless protocol, and each http request is independent father request and http client can be a browser or an application running on the client example and application running on an IoT device, mobile applications or other software.
- CoAP: Constrained application protocol is an application layer protocol for machine-to-machine application M2M meant for constrained environment with constrained devices and constrained networks. Like http CoAP is a web transfer protocol and uses a request- response model, however it runs on the top of the UDP instead of TC CoAP uses a client –server architecture where client communicate with server using connectionless datagrams. It is designed to easily interface with http like http, CoAP supports method such as GET, PUT, DELETE.
- WebSocket: WebSocket protocol allows full duplex communication over a single socket connection for sending message between client and server. WebSocket is based on TCP and Allows streams of messages to be sent back and forth between the client and server while keeping the TCP connection open. The client can be a browser, a mobile application and IoT device
- MQTT: Message Queue Telemetry Transport it is a lightweight message protocol based on public -subscribe model MQTT uses a client server Architecture by the clients such as an IoT device connect to the server also called the MQTT broker and publishers' message to topic on the server. The broker forwards the message to the clients subscribed to topic MQTT is well suited for constrained and environments.
- XMPP: Extensible Messaging and Presence Protocol it is a protocol for real-time communication and streaming XML data between network entities XMPP powers wide range of applications including messaging, presence, data syndication, gaming multiparty chat and voice / voice calls. XMPP Allows sending small chunks of XML data from one network entity to another in real time. XMPP supports both client to server and server –client communication path.
- DDS: Data distribution service is the date centric middleware standard for device to-device machine to machine communication DDS uses a publish subscribe model where publisher example device that generate data create topics to which subscribers per can subscribe publisher is an object responsible for data distributions and the subscriber

responsible for receiving published data. DDS provide quality of service (QoS) control and configurable reliability

- AMQP: Advanced Message Queuing protocols. it is an open application layer protocol for business messaging. AMQP support point to point and publish - subscribe model routing and queuing. AMQP broker receive message from publisher's example devices or applications that generate data and about them over connections to consumers publishers publish the message to exchange which then distribute message copies to queues.

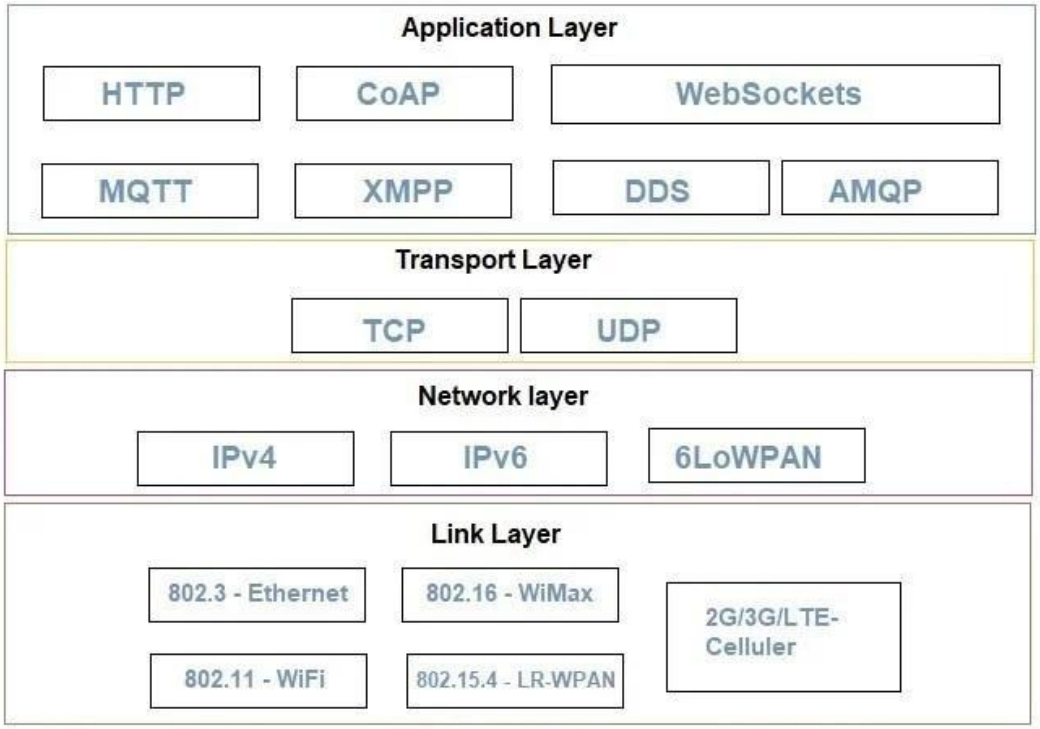

Fig.1.2 IoT protocols

Logical Design of IoT

Logical design of an IoT system refers to an abstract representation of the entities and process without going into low level specification of the implementations.

1. IoT functional block

An IoT system comprises of several functional blocks that provide the system the capabilities for identification, sensing, actuation, communication and Management as shown in fig.1.3. The function blocks are described as follows:

- Devices: An IoT system comprises of the devices that provide sensing, actuation, monitoring and control function
- Communication: communication block handles the communication systems

- Services: An IoT system uses various types of IoT services such as services for device monitoring, device control services, data publishing services and services for device Discovery.
- Management: Functional blocks provide various functions to govern the IoT system
- Security: Security functional block security IoT system and by providing functions such as application authorization message and content integrity and data security.
- Application: IoT application provides and interface that the user can used to control and monitor various aspects of the IoT system. Application also allows users to view the system status and view or analyze the processed to data.

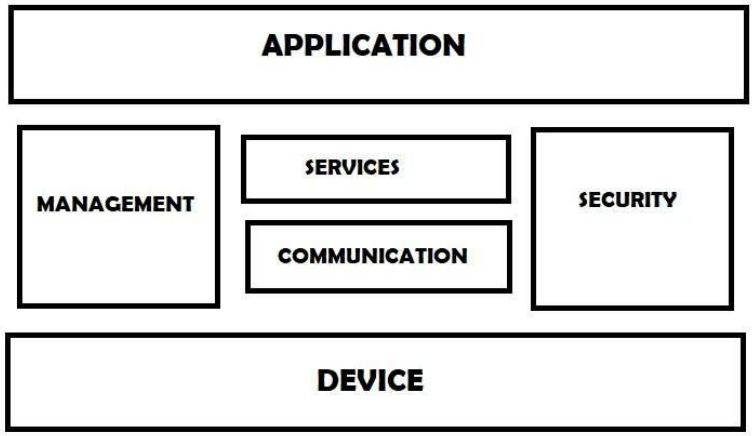

Fig.1.3 Functional blocks of IoT

2. IoT communication model

- Request response: Request-response is a Communications model in which the client sends request to the server and the server responds to the requests. when the server receives a request, it decides how to respond, if it shows the data retrieved resources definitions for the response and then send the response to the client. Access to response model is a stateless communication model and each request response per is independent of others the crime and server interactions in the request response model.
- Publish - Subscribe: Respect is a communication model that involve Publishers brokers and consumers. Publishers are the source of data. Publishers send the data to the topics which is managed by the broker. Publishers are not aware of the consumer. Consumers Subscribe to the topic which are managed by the broker. When the broker receives the data for a topic from the publisher, it sends the data to all the subscribed consumers.
- Push pull: Push pull is communication model in which the data producers push the data to queues and the consumers pull the data from the queues. Producers do not need to be aware of the consumer. Queues help in decoupling the messaging between the Producers and Consumers. It also acts as a buffer which helps in situations when there is a mismatch between the rate at which the produces push data and the rate at which the consumers full the data.

- Exclusive pair: Exclusive pair is a bidirectional, fully duplex communication model that uses a persistent connection between the client and the server. once the condition is setup it remains open until the client sends a request to close the connection. client and server can send messages to each other after connection setup. Exclusive pair is a stateful Communications model and the server is aware of all the open connections.

3. **IoT communication APIs**

<u>REST- based communication API:</u>

Representational state transfer is a set of architectural principles by which you can design web service and Web API that focus on a system resource and how resources states and addressed the transferred. REST API follow the request- response communication model. The REST architectural constraints apply to the components, connectors, and data elements.

- Client server: The principle behind the client-server conference separations of concerns for example client should not be concerned with the storage of data which is their concern of the server. Similarly, the server should not be concerned about the user interface which is a concern of the client. separation allows client and server to be independently deployed 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.
- Catchable: Catch constrain requires that the data within the response to a request be implicitly or explicitly labelled as catchable or non-catchable. Then a client cache is given the right to reuse that response data for later, equivalent requests. completely eliminate some attractions and improve efficiency and scalability.
- Layered system: System constraint come off constraints, constrains the behaviour of components such that each component cannot see beyond the immediate layer with which they are interacting. Example client cannot tell whether it is connected directly to the end server or to an intermediary along the way system scalability can be improved allowing intermediaries to respond to request instead of tender server.
- Uniform interface: Uniform interface constraints requires that the method of communication between client and server must be uniform. Resources are identified in the request and separate from the representation of the resource that are returned to the client. When climbing holds, a representation of your resource it has all the information required to update or delete the resource
- Code on demand: Service can provide executable code script for clients to execute in their context.

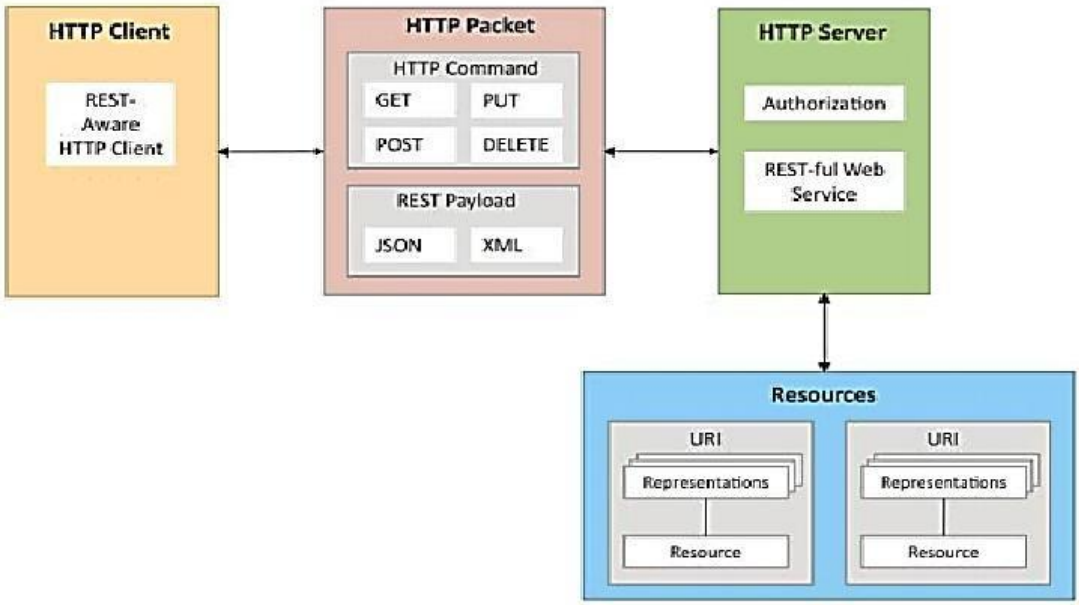

Fig.1.4 Communication with REST APIs

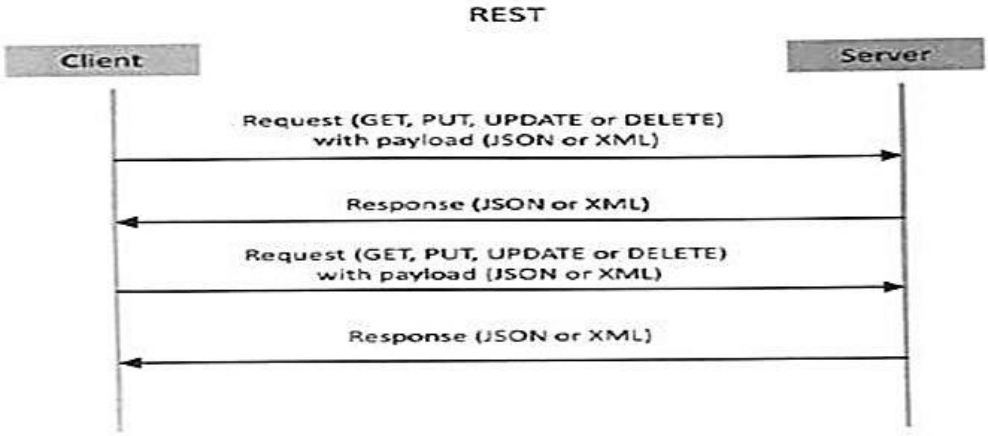

Fig.1.5 Request-response model used by REST

<u>WebSocket based communication API:</u>

WebSocket API allow bidirectional, full duplex communication between client and server. Unlike request-response API allow full duplex communication and do not require new connection to be set up for each message to be sent. WebSocket communication begins with connection setup request send by the client to the server. The request is sent over http, and the server interprets it as an upgrade request. If the server support protocol response to the website handshake response after the connection setup the client and the server can send data or messages to each other in full duplex model. WebSocket API reduce network traffic and latency as there is no overhead for connection setup and determination records to each message.

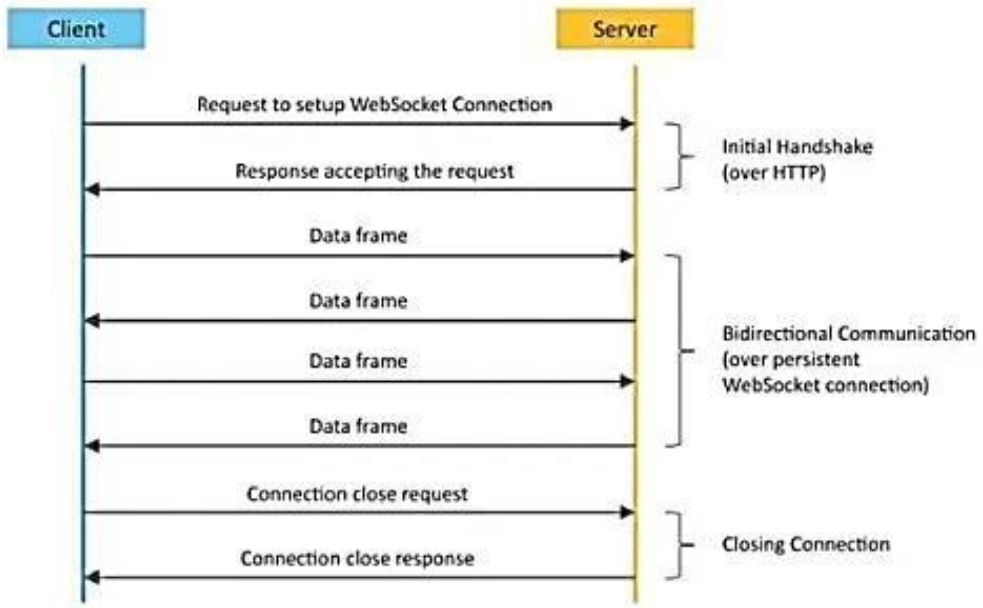

Fig.1.6 Exclusive pair model used by WebSocket APIs

IoT enabling technologies

It is enabled by several Technologies including wireless sensor networks, cloud computing big Data Analytics, embedded system, security protocols and architectures, communication protocols, web service, mobile internet and semantic search engine.

1. **Wireless Sensor Network**

Wireless sensor network comprises of distributed devices with the sensor which are used to monitor the environmental and physical conditions. A WSN consists of a number of end nodes and routers and a coordinator. End nodes have several sensors attached to them. End node can also act as a router. Routers are responsible for routing the data packet from end nodes to the coordinator. The coordinator node collects the data from all the notes coordinators also act as a Gateway that connects the WSN to the internet. IoT systems are described as follows

- Weather monitoring system using WSN in which the nodes collect temperature, humidity and other data which is aggregated and analysed.
- Indoor air quality monitoring system using WSN to collect data on the indoor air quality and connections of various gases.
- Soil moisture monitoring system using WSN to monitor soil moisture at various location.
- Surveillance systems use WSN for collecting surveillance data (motion detection data)
- Smart grids use wireless sensor network for monitoring the grid at various point.
- Structural health monitoring systems use WSN to monitor the health of structure by writing vibration data from sensor nodes deployed at various points in the structure.

2. Cloud computing

Cloud Computing is a transformative computing paradigm that involves delivering applications and services over the internet. Cloud Computing involves provisioning of computing networking and storage resources on demand and providing these resources as metered services to the users, in a "pay as you go" model. Cloud Computing resources can be provisioned on demand by the user without requiring interactions with the Cloud Service Provider. The process of provisioning resources used automatic Cloud Computing resources can be accessed then it worked using standard access mechanism that provide platform-independent access using heterogeneous client platforms such as workstations laptops tablets and Smartphones the computing and storage resources provided by Cloud Service Provider our food to serve multiple users using multi-Tenancy. Multi-tenant aspects on the multiple users to be served by the same physical hardware.

Cloud Computing services are offered to user in different forms

- Infrastructure as a service (IAAS): IaaS provides the user the ability provision computing and storage resources. These resources are provided to the users as virtual machine instances and virtual storage. Users can start, stop configure and manage the virtual machines instance on the virtual storage using can deploy operating systems and applications on their choice on the actual resources provisions in the cloud. Cloud Service Provider manages the underlying infrastructure.
- Platform as a service (PaaS): Platform as a service provides the user the ability to develop and deploy application in the cloud using the deployment tool application programming interfaces API, software libraries and services provided by the Cloud Service Provider. The Cloud Service Provider manages the underlying cloud infrastructure including servers, network, operating systems and storage.
- Software as a service (SaaS): Provide the user a complete software application of the user interface to the application itself. The Cloud Service Provider manage the underlying cloud infrastructure including server, network storage and application software, and the user is unaware of the underlying architecture of the cloud. Applications are provided to the user through a thin client interface example Browser application. SaaS applications are accessed from various client smartphones running different operating system.

3. Big Data Analytics

Big data is defined as collections of data set whose volume, velocity in terms of its temporal variations) or variety, is so large that it is difficult to store, manage, process and analyse the data using traditional database and data processing tools. Big Data Analytics involving several steps starting from Data cleaning data munging data processing and visualization.

Some examples of big data generated by IoT systems are described as follows:
1. Sensor data generated by IoT system such as weather monitoring stations

2. Machine sensor data collected from sensor embedded in Industrial and energy system for monitoring their files and protecting failure
3. Health and fitness data generated by IoT devices such as wearable fitness band.
4. Data generated by IoT system for Location tracking of vehicle.
5. Data generated by retail inventory monitoring system.

Characteristics of data include:

- Volume: Through there is no fixed threshold for volume of data to be considered as big data, however the term big data is used for massive scale data that is difficult to store, manage and process using traditional data bases and data processing architecture. The volume of data generated by modern IT, industrial and Healthcare systems for example is a growing exponentially driven by the lowering cost of data storage and processing architectures and the need to extract valuable insights from the data to improve business processes, efficiency and services to consumer.
- Velocity: Velocity is another important characteristic of big data and the primary reasons for exponential growth of data velocity of the data of a store how fast the data is generated and how frequently it varies. Modern IT Industrial and other systems are generating data at increasing the highest speeds.
- Variety: Variety refers to the forms of the data. Big data comes in for different forms such as structured or unstructured data including text data, audio, video and sensor data.

4. Communications protocol

Communications protocols form the backbone of IoT system and enable network connectivity and coupling to applications. Communications protocols allow device to exchange data over the network. These protocols define the data exchange formats and data encoding schemes for devices and routing of packets from source to destination. Other function of the protocol includes sequence control flow control and transmissions of Lost packet.

5. Embedded systems

An Embedded system is computer system that has computer hardware and software embedded perform specific task. In contrast to general purpose computers or personal computers which can perform various types of tasks, embedded systems are designed to perform a specific set of tasks. Embedded system includes Microprocessor and Microcontroller memory Ram ROM cache networking units (Ethernet WI-FI adaptor) input/output unit display keyboard, display and storage such as Flash Memory some embedded system have specialist processes such as digital signal processor DSP graphic processor and application.

IoT levels and Deployment Templates

This section defines various levels of IoT systems with increasing completely. IoT system comprises of the following components:

1. Device: An IoT device allow identification, remote sensing, actuating and remote monitoring capabilities.
2. Resources: Resources are software components on the device for accessing and storing information for controlling actuator connected to the device also include software components that enable network access for the device.
3. controller service: Controller Service is a native service that runs on the device and interact with the web services. Controller service sends data from the device to the web service receive command from the application from controlling the device.
4. Database: Database can be either local or in the cloud and stores the data generated by the IoT device.
5. Web service: Serve as a link between the device, application database and analysis components. Web Services can be implemented using HTTP and REST principles or using website protocol.

A comparison of restaurant website is provided below:

- Stateless/stateful: Rest services stateless in nature. Each request contains all the information needed to process it. Requests are independent of each other. Website on the other hand is stateful in nature where the server maintains the state and is aware of all the open connections.
- Directional / Bi-directional: REST service operates over http and unidirectional. Request is always sent by a client and the server response to the request. And other hand website is a bidirectional product server to send message to each other
- Request response / full duplex: REST service follower request response Communications model where the client sends request and the server response to the request. Website and the other hand Allow full-duplex Communications between the client and server, it means both client and server can send messages to can independently.
- TCP connections: For REST Service each http request involves setting up in a new TCP connections WebSocket on the other hand involves a single TCP connection over which the client and server communicate in a full duplex mode.
- Headache Overhead: REST service operates over http, and each request is independent of others. Thus, each request carries http header which is an overhead. Due to the overhead of http headers, REST is not suitable for real time applications left hand does not involve overhead of headers. After the initial handshake the client and server exchange messages with minimal frame information.
- Scalability: Scalability is easier in this case of the REST services of request are independent and no state information needs to be maintained by the server. Thus, both horizontal out and vertical scaling solutions are possible for REST services. For WebSocket's horizontal scaling can be cumbersome due to stateful nature of the communication. Since the server maintains the state of our connection, vertical scaling is easier for WebSocket than horizontal scaling.

- Analysis component: The analysis component is responsible for analysing the IoT data and generate results in the form which are easy for the user to understand. Analysis of IoT data can be performed either locally or in the cloud. Analysed results are stored in the local or cloud database.
- Application: IoT applications provide an interface that the user can use to control and monitor various aspects of the IoT system. Applications also allow user to view the system status and view the processed data.

IoT level 1

Level One IoT system has a single node / device that performs sensing and/or actuation, stores data, reforms analysis and the host to the application. Level 1 IoT systems are suitable for modelling low cost and low complexity solutions where the data involving is not big, and the analysis requirements are not computationally intensive.

Consider an example of Level 1 IoT system for home automation. This system consists of the single node that allows controlling the lights and appliances in your home remotely. The device used in this system interface with their lights and appliances using electronic relay switches.

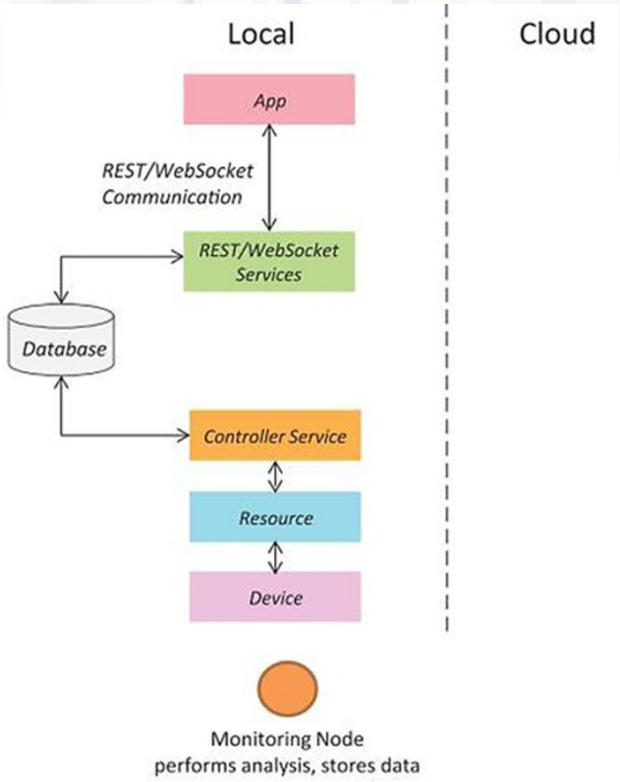

Fig.1.7 IoT level 1

The status information of each light or appliance is maintained in a local database. REST service deployed locally Allow retrieving and updating the state of each light or appliances in the status database.

The controller service continuously monitors the state of each light or appliance and triggers the relay switches accordingly. The applications which are deployed locally has a user interface for controlling the lights or appliances. since the device is connected to the internet, the application can be accessed remotely as well.

IoT level 2

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

Construct an example of Level 2 IoT system for smart irrigation.

The system consists of the single node that monitor the soil moisture level and control segregation system. The device used in this system collect soil moisture data from sensor the controller service continuously monitors the moisture level. If the monster level drops below a threshold t, the irrigation system is turned on. For controlling the irrigation system actuators such as solenoid valve can be used. Rest Web Services is used for storing and retrieving data which is stored in the cloud database. A cloud-based application is used for visualizing the moisture level over a period, which can help in making decisions about irrigation schedules.

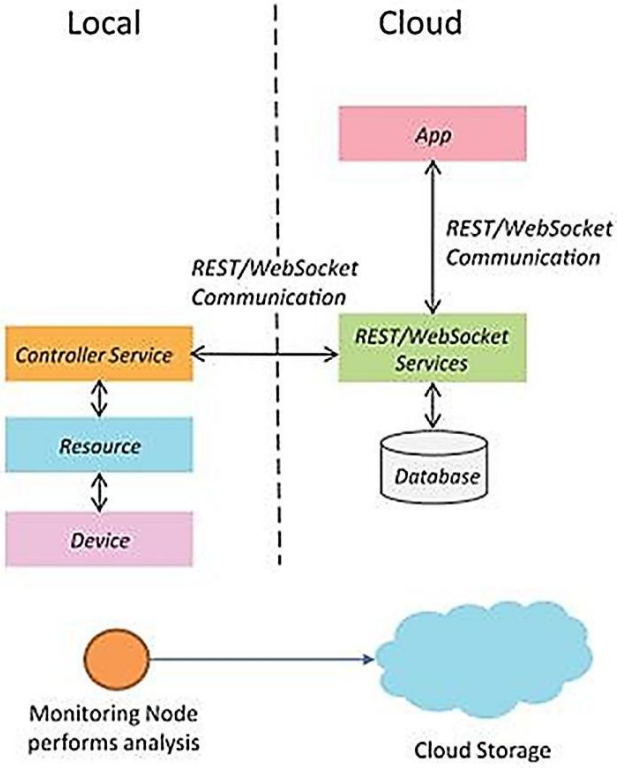

Fig.1.8 IoT level 2

IoT Level 3

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

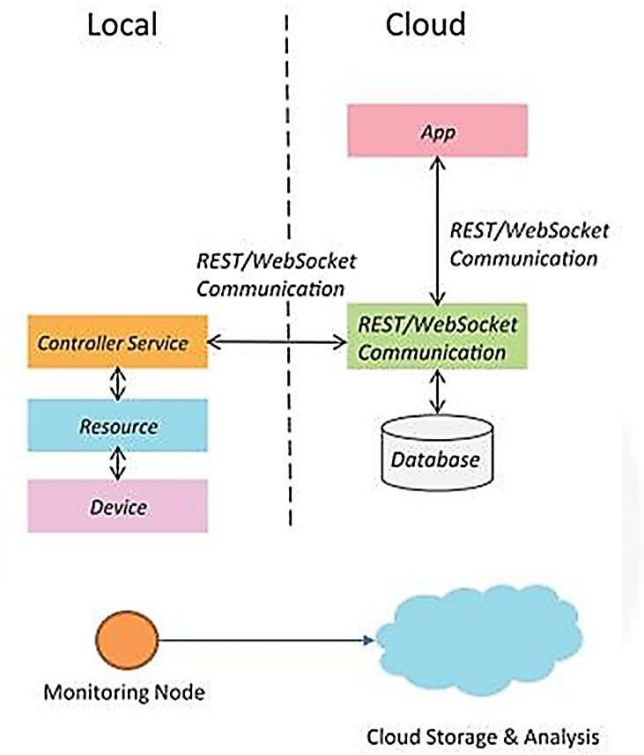

Fig.1.9 IoT level 3

Consider example of Level 3 IoT system tracking package handling. The system consists of a single node that monitors the vibration level for package being shipped. The device in the system uses accelerometer and gyroscope sensor for monitoring vibration levels. The controller service sends sensor data to the cloud in real time using a website service. The data is stored in the cloud and visualized using a cloud-based application.

The analysis component in the cloud can Trigger alert the vibration level becomes greater than threshold. The benefit of using WebSocket service instead of the REST service this example the sensor data can be sent in real-time to the cloud. Cloud based application can subscribe to the sensor data feeds for you in the real-time data.

IoT level 4

A level 4 IoT system has multiple notes that perform local analysis. Data is stored in the cloud and application is cloud based, level 4 contains local and cloud-based observer notes which can subscribe to and receive information collected in the cloud from IoT devices.

Observer node can process information and use it for various applications; however, observer notes do not perform any control function. 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.

Consider an example of level four IoT system for noise monitoring. The system consists of multiple notes placed in different locations for monitoring noise level in an area. In this example with sound sensor. Nodes are independent of each other each node runs in one controller service that sends the data to the cloud. The data is stored in a cloud database the analysis of the data collected from several notes is done in the cloud

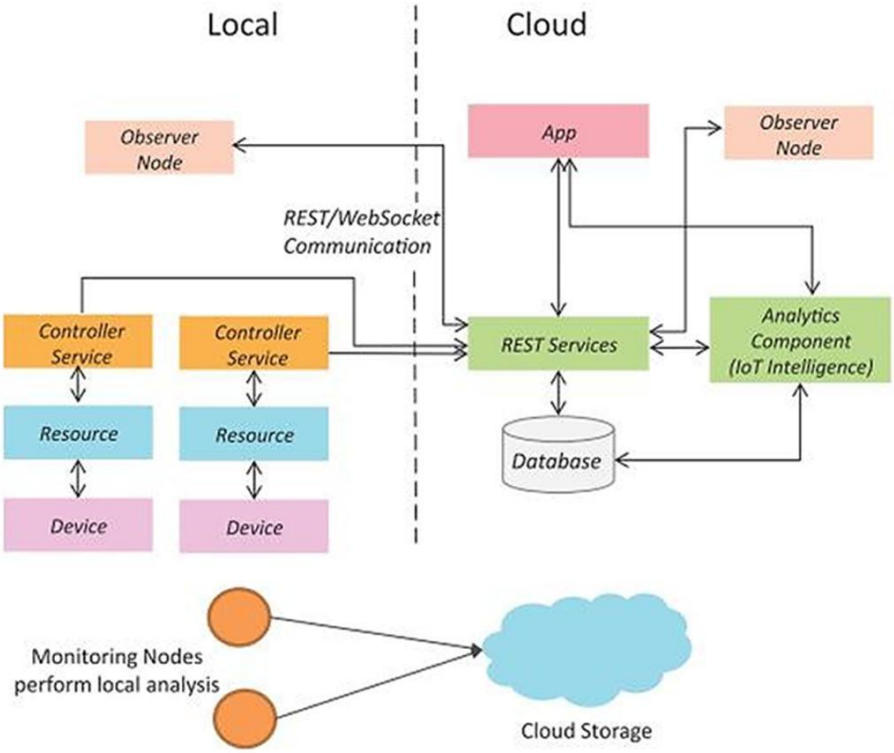

Fig.1.10 IoT level 4

IoT Level 5

IoT system has multiple end nodes and one coordinator notes and notes that perform sensing and / or actuation. Coordinator node collects data from the entry and send to the cloud. Data is stored and analysed in the cloud and applications is cloud based. Level 5 IoT system are suitable for forest fire detection. The system consists of multiple nodes placed in different locations for monitoring temperature, humidity and carbon dioxide levels in a forest.

The endnotes in this example are equipped with various sensors such as temperature humidity and to CO2. The coordinator node collects the data from the end nodes and act as a Gateway that provides internet connectivity to the IoT system. The controller service on the coordinator

device sends the collected data to the cloud. The data is stored in the cloud database. The analysis of the data is done in the computing cloud to aggregate the data and make prediction.

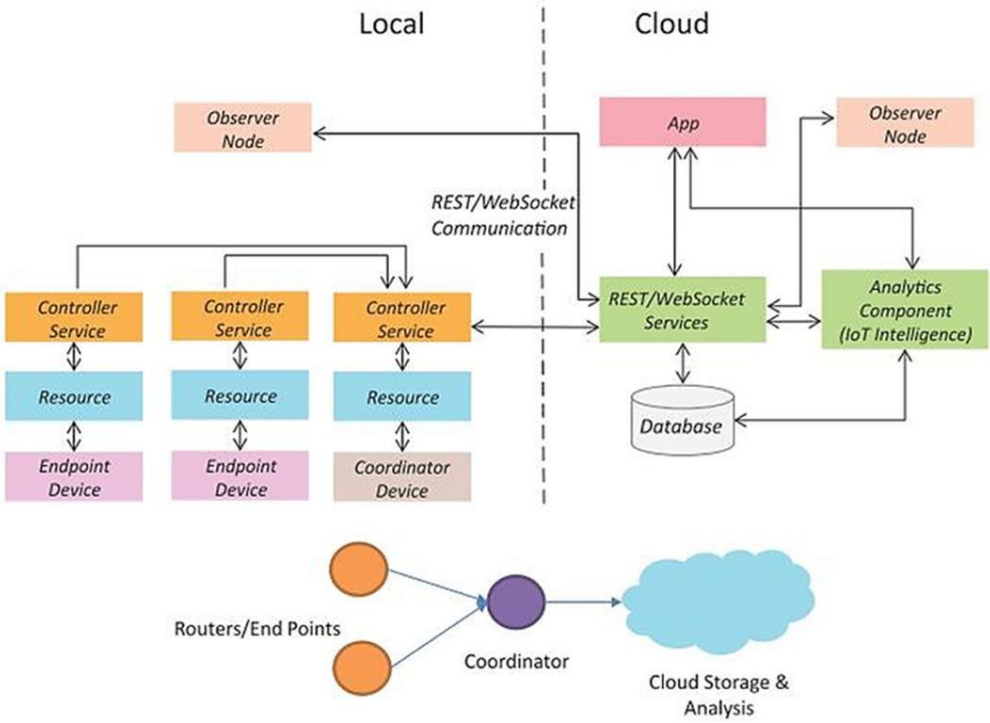

Fig.1.11 IoT level 5

IoT Level 6

IoT Level 6 system has multiple Independent and nodes that perform sensing and / or actuations and send data to the cloud. Data is stored in the cloud and applications is cloud based.

The analytics component analyses 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 notes and send control commands to the notes.

Consider an example of the level 6 IoT system for weather monitoring. The system consists of multiple nodes placed in different location for monitoring temperature, humidity and pressure in an area. The end nodes are equipped with various sensors such as temperature, pressure and humidity. The end nodes send the data to the cloud in real time using a WebSocket service. The data is stored in a cloud database. The analysis of the data is done in the cloud to aggregate the data and make predictions. A cloud-based applications is used for visualizing the data.

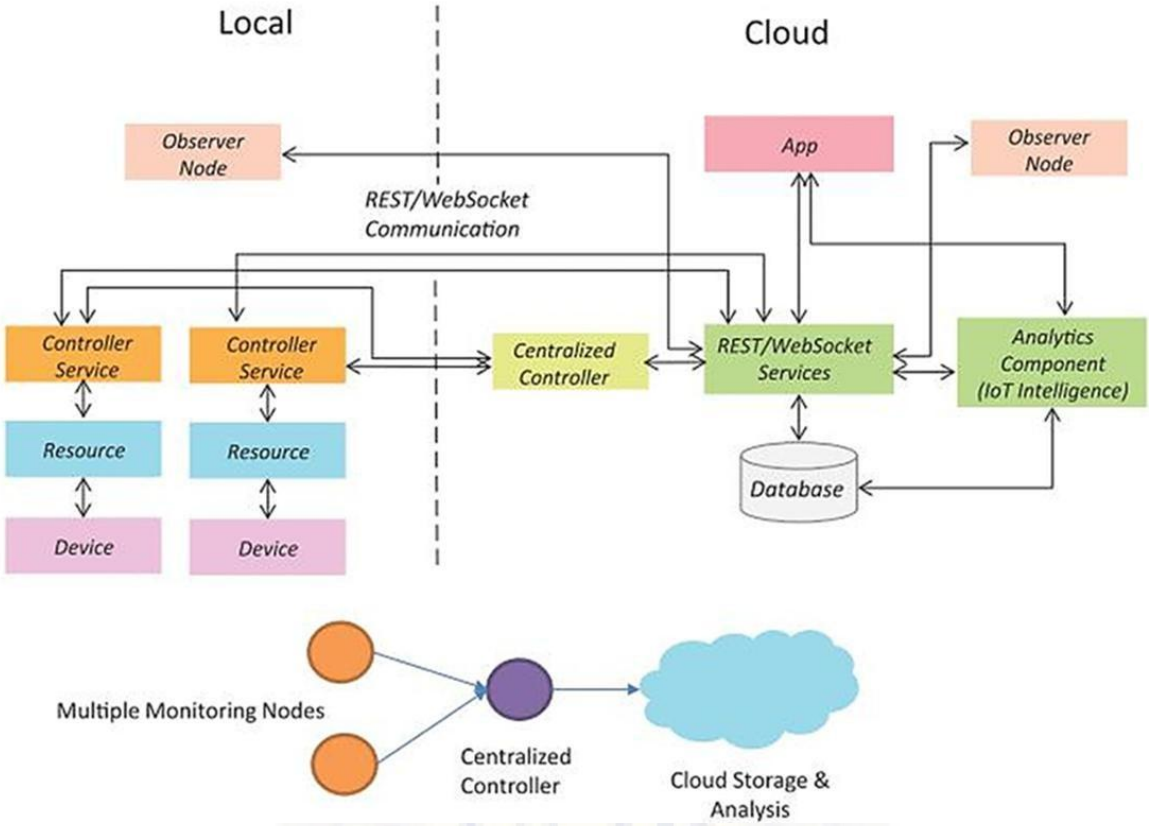

Fig.1.12 IoT level 6

Module-2

Chapter 1: IoT and M2M

Introduction to M2M

Machine to machine (M2M) refers to networking of Machines for the purpose of remote monitoring and control and data exchange. The end-to-end architecture for M2M systems comprising of M2M area networks, Communications Network and application domain.

An M2M area network comprises of machines which have embedded hardware module for sensing actuation and communication. Various Communication protocols can be used for M2M local area network such as Zigbee, Bluetooth, Modbus M–bus, wireless, power LINE Communication ,6LoWPAN.

These Communications protocols provide connectivity between M2M nodes within and M2M area network. The Communications Network provides connectivity to remote M2M area network. communication network can use wired or wireless network. The M2M area network use either proprietary or non-IP based protocol.

The communication between the M2M nodes and the M2M Gateway is based on the communication protocol. M2M Gateway protocol translation to enable IP connectivity for M2M. M2M Gateway act as a proxy performing translation from / to native protocol to M2M area network.

M2M data is gathered into point solution such as enterprise applications, service management application for remote monitoring applications. M2M has various application domain such as smart metering, Home Automation, industrial Automation, smart grid.

Difference between IoT and M2M

The difference between m2m and IoT are described as follows:

- Communication protocols:

M2m and IoT can differ in how the communication between the machines are device happens. M2M uses other proprietary or not IP based communication protocol for communication with in the M2M area networks. Commonly uses M2m protocol include zigbee, Bluetooth, ModBus, wireless M-Bus, power line communication. The focus of communication in M2M is usually on the protocols below the network layer. Focus of communication in IoT is usually a protocol in network layer such as http web sockets, MQTT, XMPP, DDS, AMQP.

- Machines in M2M vs Things in IoT:

The " things " IoT refers to Physical objects that have unique identifier and can sense and communicate with the external environment or their internal physical status. The unique identifiers the things in IoT are the IP addresses. Things have software component for accessing

processing and storing sensor information on controlling actuator connector. IoT system can include IoT devices of various types such as fire alarms, door alarms, lighting control devices.

- Hardware versus software emphasis:

while the emphasis of M2M is more on hardware with embedded modules, the emphasis modules, the emphasis of IoT is more on software. IoT devices run specialist software sensor Data Collection, data analysis and interfacing with cloud through IP based communication.

- Data collection and analysis:

M2M data is collected in point solutions and often in on premises storage infrastructure. In contrast to M2M, the data in IoT is connected in the cloud. The analytical component analysis the data and stores the result in the cloud database. Data and analysis results are visualized with the cloud-based applications. The centralized controller is aware of the status of all the nodes and send Control Commands to the nodes.

- Applications:

M2m data is collected in point solutions and can be accessed by on premises application diagnosis applications, service management applications, and on-premises enterprise application.

SDN and NFV for IOT

Software defined networking (SDN) and the network function virtualization (NFV) and their applications for IoT.

Software Defined Networking

Software defined networking (SDN) is the networking architecture that separates the control plan from the data plan and centralizes race the network controller. Conventional network architecture builds with specialized hardware (switches, router etc).

Network device in conventional architectures is getting exceedingly Complex with the increasing number of distributed products has been implemented and the use of proprietary hardware and interfaces. Control plan is the part of the network that carries the signal and routing message traffic while the data plan is a part of network that carries the payload data traffic.

The limitations of the conventional network architecture as follows:

- Complex network devices: Interoperability is limited due to the lack of standard and open interfaces. Network devices use proprietary hardware and software and have slow product lifecycle limiting innovations. The convention networks were well suited for static traffic pattern and had many products was decided for specific applications which are applied in cloud computing environment traffic patterns are more dynamic. Due to complexity of conventional network devices making changes in the networks to meet the dynamic traffic pattern has become increasingly difficult.

- Management overhead: conventional networks involve significant manager overhead. Network managers find it increasingly difficult to manage multiple network devices and interfaces from multiple vendors. Up gradation of network configuration changes in multiple devices.
- Limited scalability: The Virtualization technologies used in cloud computing environment has increased the number of its host repairing network access IoT applications hosted in the cloud are distributed across multiple virtual machines that require exchange of traffic.

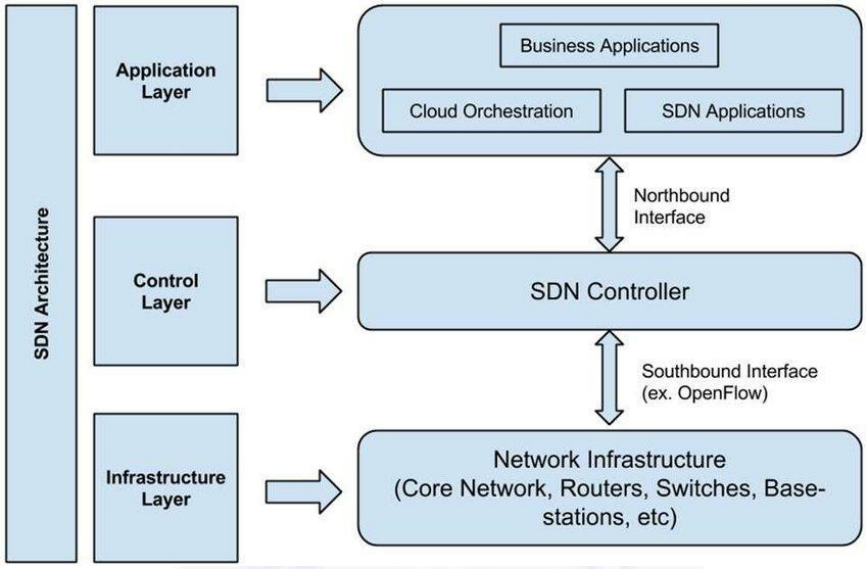

Fig.2.1 SDN layers

Components of IoT applications run distributed algorithms on many virtual machines that require huge amounts of data exchange between virtual machines. Such computing environment requires highly scalable and easy to manage network architectures with minimal manual configuration which is becoming raising a difficult with a conventional network. Key Elements of SDN and follow:

- Centralized network controller: with Decoupled control and the data plan and centralized network controller, the network administrator can rapidly configure the network. SDN applications can be deployed Programmable open API. This speed has innovation as the network status no longer need to wait for other device vendors to embed features in their proprietary hardware.
- Programmable open APIs: SDN architecture propose Programmable open API for interface between the SDN application and control layers with these open API is various network services can be implemented such as routing quality of services access.
- Standard communication interface (Open Flow): SDN architecture uses a standard communication interface between the control and infrastructure layers. OpenFlow, which is Defined by the open networking Foundation is the broadly accepted SDN

protocol for the southbound interface. Open flow, the forwarding plan of the network devices can be directly accessed and manipulated.

Open floor uses the concept of close try different network traffic based on three different rules. Floor can be programmed statically and dynamically by the SDN control software. Components of and OpenFlow switch comprising one or more tables and their group table which perform packet lookup and forwarding, and open flow channels to an external Controller System OpenFlow protocol is implemented at both sides of the interface between the controller and the network devices.

Network function virtualization

Network function virtualization is a technology that leverages virtualization to consolidate the heterogeneous network devices on to industry standard high-volume service switches and storage. NFV is complementary to SDN as NFV can provide at the infrastructure on which SDN can run. NFV and SDN mutually beneficial to each other but not dependent. Network functions can be virtualized without SDN, similarly SDN can run without NFV.

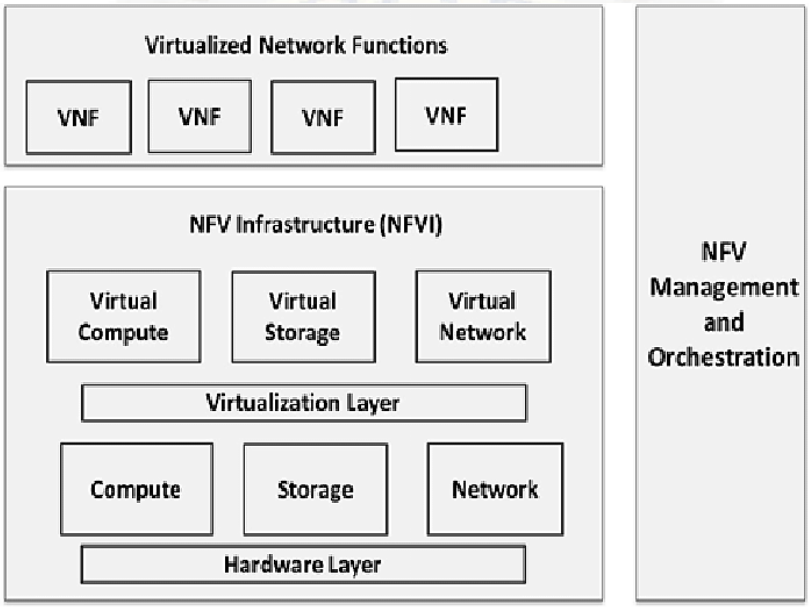

Fig.2.2 NFV architecture

Elements of architecture as follows:

- Virtualize to network function VNF: VNF is a software implementation of a network function which can run over the enough Infrastructures
- NVF Infrastructure NFVI: It into Computer Network and storage resources that are visualized.
- NFV management and orchestrations: Orchestrations focuses on all visualization specific management task and the covers the orchestrations and lifecycle management

of physical and / or software resources that support the infrastructure utilization and the lifecycle management of VNF.

NFV comprises of network functions implemented in software that run on Virtualized resources in the cloud. NFC enabled separations of network function which are implemented in software from the underlying hardware.

Chapter 2: IoT System Management with NETCONF-YANG

Need for IoT Systems Management

Managing multiple devices within a single system requires advanced management capabilities. The need for managing IoT systems is described as follows:

- Automating Configuration: IoT system management capabilities can help in automating the system configuration.
- Monitoring Operational & Statistical Data: Management systems can help in monitoring operational and statistical data of a system. This data can be used for fault diagnosis or prognosis.
- Improved Reliability: A management system that allows validating the system Configurations before they are put into effect can help in improving the system reliability.
- System Wide Configurations: For IoT systems that consists of multiple devices or nodes, ensuring system wide configuration can be critical for the correct functioning of the system.
- Multiple System Configurations: For some systems it may be desirable to have multiple valid configurations which are applied at different times or in certain conditions.
- Retrieving & Reusing Configurations: Management systems which have the capability of retrieving configurations from devices can help in reusing the configurations for other devices of the same type.

Simple Network Management Protocol (SNMP)

SNMP allows monitoring and configuring network devices such as routers, switches, servers, printers etc. Entities involved in managing a device with SNMP are Network Management Station (NMS), Managed devices, Management information base (MIB) and the SNMP Agent. SNMP is an application layer protocol that uses UDP as the transport protocol.

Limitations of SNMP

- SNMP was designed to provide a simple management interface between the management applications and the managed devices. SNMP is stateless in nature and each SNMP request contains all the information to process the request. The application needs to be intelligent to manage the device.

- SNMP is a connectionless protocol which uses UDP as the transport protocol, making it unreliable as there was no support for acknowledgement of requests.
- It is difficult to differentiate between configuration and state data in MIBs.
- Retrieving the current configuration from a device can be difficult with SNMP.
- Earlier versions of SNMP did not have strong security features making the management information vulnerable to network intruders.

Network operator requirements

- Ease of use: From the operator's point of view, ease of use is the key requirement for any network management technology.
- Distinction between configuration and state data: Configuration data is the set of writable data that is required to transform the system from its initial state to its current state. State data includes operational data which is collected by the system at runtime and statistical data which describes the system performance. It is important to make a clear distinction between configuration and state data.
- Fetch configuration and state data separately
- Configuration of the network as a whole
- Configuration transactions across devices
- Configuration deltas
- Dump and restore configurations
- Configuration validation
- Configuration database schemas
- Comparing configuration
- Role based access control
- Consistency of access control lists
- Multiple configuration sets
- Support for both data-oriented and task-oriented access control

NETCONF

Network Configuration Protocol is a session-based network management protocol. It allows retrieving state or configuration data and manipulating configuration data on network devices.

The configuration data resides within a NETCONF datastore on the server. The NETCONF server resides on the network device. The management application plays the role of a NETCONF client. For managing a network device, the client establishes a NETCONF session with the server. When a session is established the client and server exchange 'hello' messages which contain information on their capabilities.

Client can then send multiple requests to the server for retrieving or editing the configuration data. NETCONF allows the management client to discover the capabilities of the server. NETCONF gives access to the native capabilities of the device.

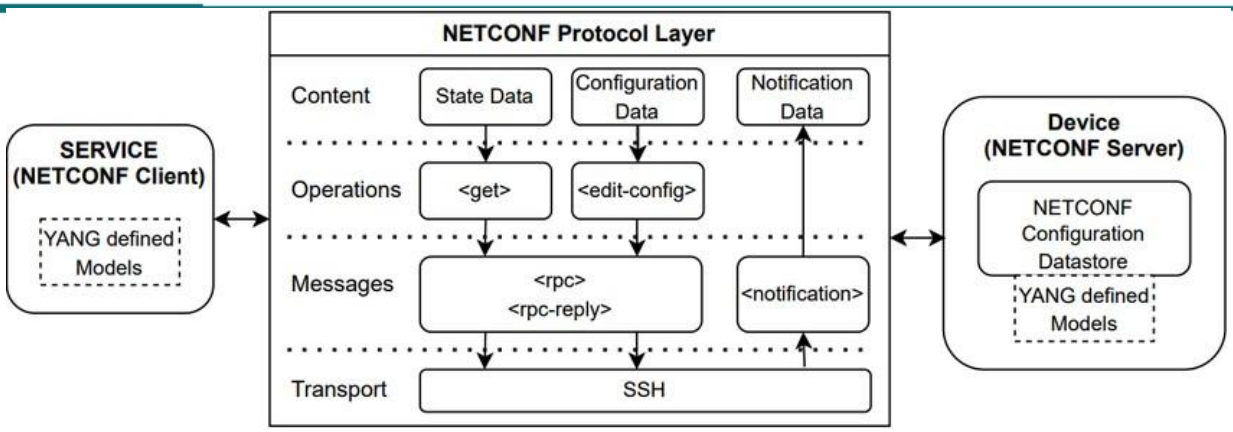

Fig.2.3 NETCONF protocol layers

YANG

YANG is a data modelling language used to model configuration and state data manipulated by the NETCONF protocol. YANG modules contain the definitions of the configuration data, state data, RPC calls that can be issued and the format of the notifications.

YANG modules define the data exchanged between the NETCONF client and server. A module comprises of several 'leaf' nodes which are organized into a hierarchical tree structure. The 'leaf' nodes are specified using the 'leaf' or 'leaf-list' constructs. Leaf nodes are organized using 'container' or 'list' constructs. YANG can model both configuration data and state data using 'config' statement. YANG defines 4 types of nodes for data modelling.

Table: YANG Node Types

Node Type	Description
Leaf Nodes	Contains simple data structures such as an integer or a string. Leaf has exactly one value of a particular type and no child nodes.
Leaf-list Nodes	Is a sequence of leaf nodes with exactly one value of a particular type per leaf.
Container Nodes	Used to group related nodes in a subtree. A container has only child nodes and no value. A container may contain any number of child nodes of any type (including leafs, lists, containers and leaf-lists).
List Nodes	Defines a sequence of list entries. Each entry is like a structure or a record instance and is uniquely identified by the value of its key leafs. A list can define multiple key leafs and may contain any number of child nodes of any type.

IoT Systems Management with NETCONF-YANG

Figure 2.4 shows the generic approach of IoT device management with NETCONF-YANG.

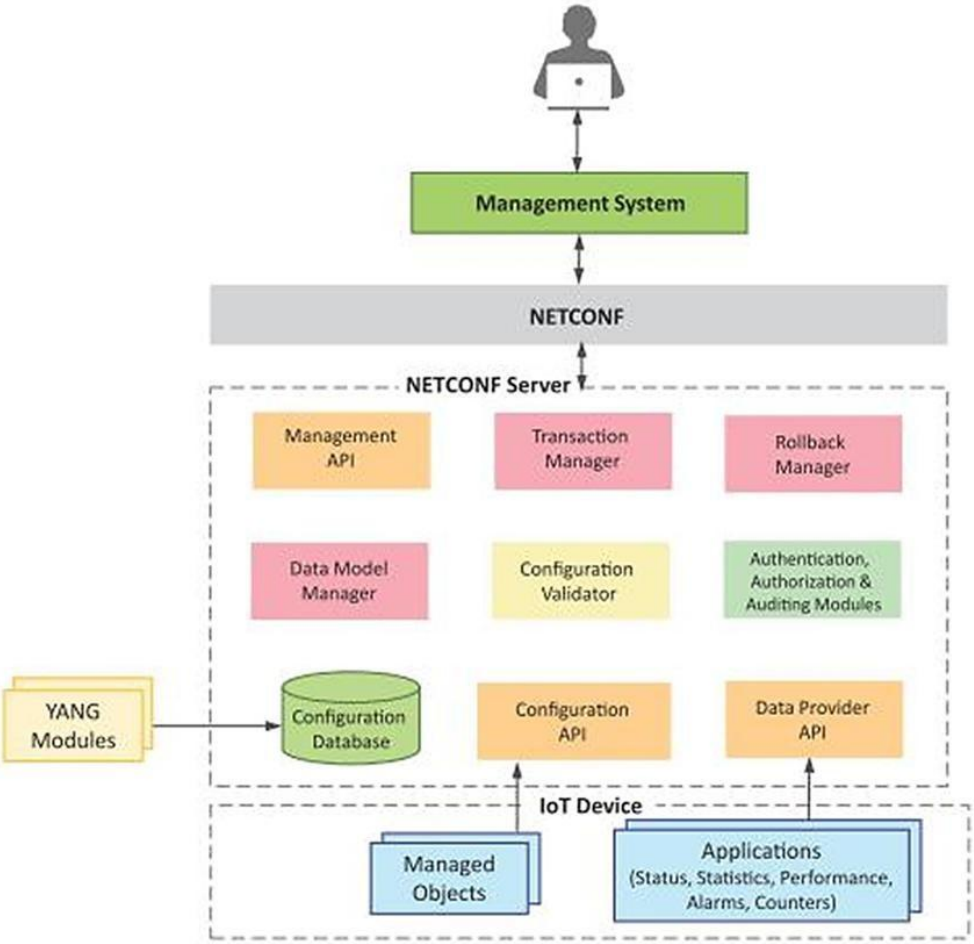

Fig.2.4 IoT device management with NETCONF-YANG – a generic approach

Roles of various components are:

- Management System: The operator uses a management system to send NETCONF messages to configure the IoT device and receives state information and notifications from the device as NETCONF messages.
- Management API: allows management application to start NETCONF sessions.
- Transaction Manager: executes all the NETCONF transactions and ensures that ACID properties hold true for the transactions.
- Rollback Manager: is responsible for generating all the transactions necessary to rollback a current configuration to its original state.
- Data Model Manager: Keeps track of all the YANG data models and the corresponding managed objects. Also keeps track of the applications which provide data for each part of a data model.

- Configuration Validator: checks if the resulting configuration after applying a transaction would be a valid configuration.
- Configuration Database: contains both configuration and operational data.
- Configuration API: Using the configuration API the application on the IoT device can be read configuration data from the configuration datastore and write operational data to the operational datastore.
- Data Provider API: Applications on the IoT device can register for callbacks for various events using the Data Provider API. Through the Data Provider API, the applications can report statistics and operational data.

NETOPEER

It is set of open source NETCONF tools built on the Libnetconf library. The Netopeer tools include:

- Netopeer-server: It is a NETCONF protocol server that runs on the managed device. Netopeer-server provides an environment for configuring the device using NETCONF RPC operations and also retrieving the state data from the device.
- Netopeer-agent: It is the NETCONF protocol agent running as a SSH/TLS subsystem. It accepts incoming NETCONF connection and passes the NETCONF RPC operations received from the NETCONF client to the Netopeer server.
- Netopeer-cli: It is a NETCONF client that provides a command line interface for interacting with the Netopeer-server. The operator can use the Netopeer-cli from the management system to send NETCONF RPC operations for configuring the device and retrieving the state information.
- Netopeer-manager: Netopeer-manager allows managing the YANG and Libnetconf Transaction API (TransAPI) modules on the Netopeer-server. With Netopeer-manager modules can be loaded or removed from the server.
- Netopeer-configurator: It is a tool that can be used to configure the Netopeer-server.

Steps for IoT device Management with NETCONF-YANG

1) Create a YANG model of the system that defines the configuration and state data of the system.
2) Complete the YANG model with the 'Inctool' which comes with Libnetconf.
3) Fill in the IoT device management code in the Trans API module.
4) Build the callbacks C file to generate the library file.
5) Load the YANG module and the TransAPI module into the Netopeer server using Netopeer manager tool.
6) The operator can now connect from the management system to the Netopeer server using the Netopeer CLI.
7) Operator can issue NETCONF commands from the Netopeer CLI. Command can be issued to change the configuration data, get operational data or execute an RPC on the IoT device.

Module-3

Chapter 1: IoT Platforms Design Methodology

Introduction

Designing IoT systems can be a complex and challenging task as these systems involve interactions between various components such as IoT devices and network resources, web services, analytics components, applications and database servers. IoT system designers often tend to design IoT systems keeping specific products/services in mind. Updating the system design to add new features or replacing a particular product becomes very complex, and in many cases may require complete re-design of the system.

IoT Design Methodology

Figure.3.1 shows the steps involved in the IoT system design methodology.

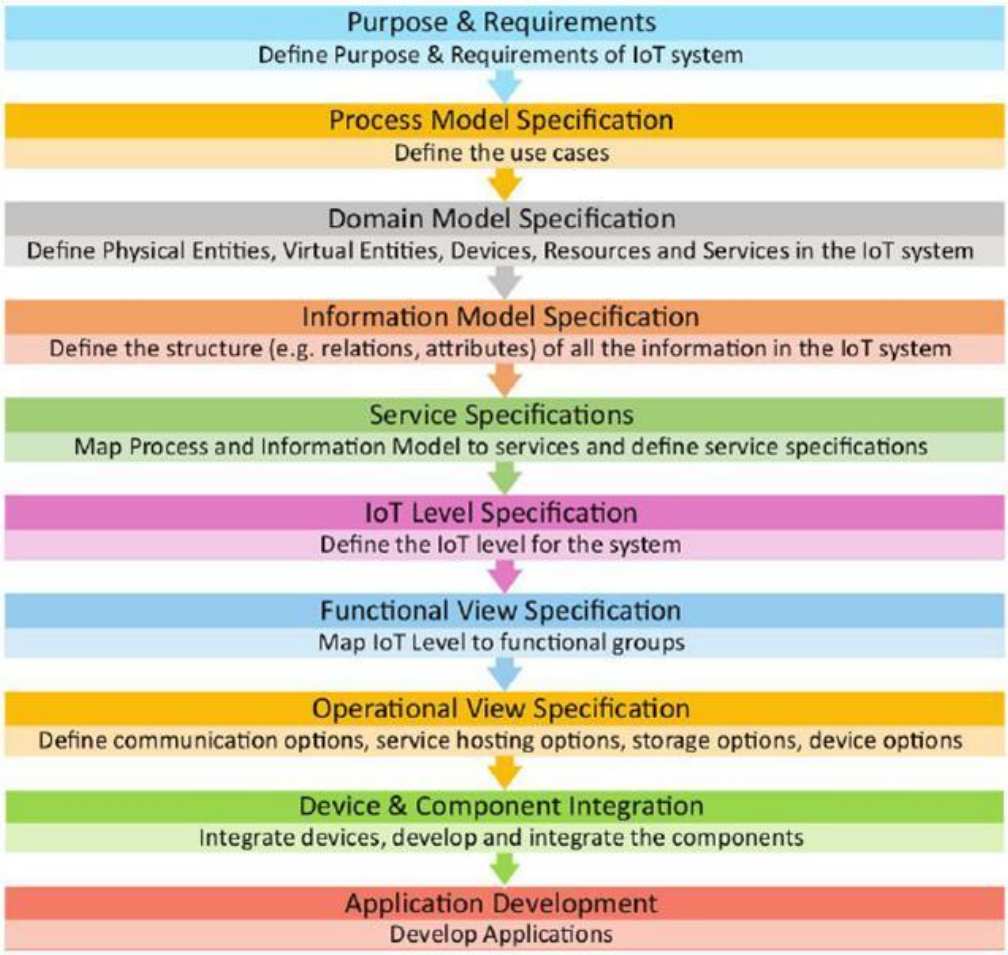

Fig.3.1 Steps involved in IoT system design methodology

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, behaviour 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:

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:

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:

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 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:

The sixth step in the IoT design methodology is to define the IoT level for the system.

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:

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 & Component Integration:
The ninth step in the IoT design methodology is the integration of the devices and components. Figure 3.2 shows a schematic diagram of the home automation IoT system.

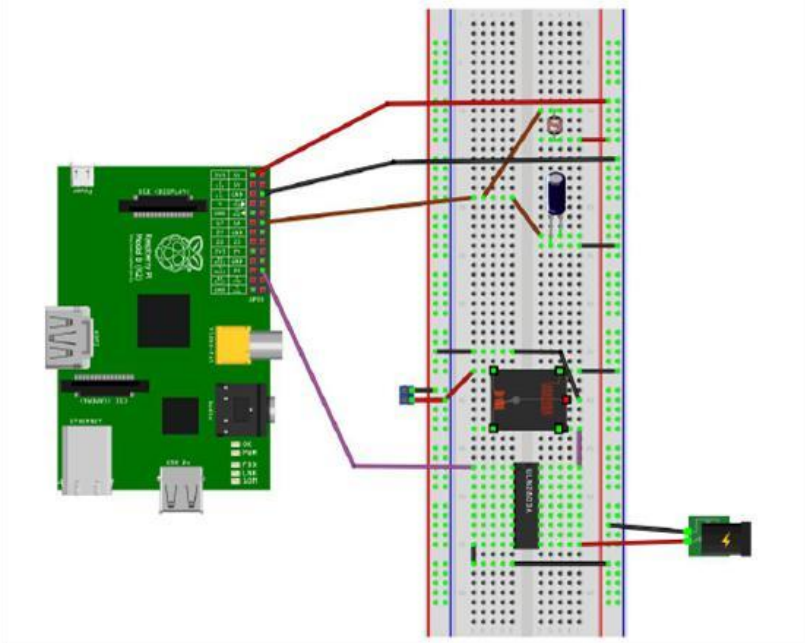

Fig.3.2 Schematic diagram of the home automation IoT system showing the device, sensor and actuator integrated.

Step 10: Application Development:

The final step in the IoT design methodology is to develop the IoT application. Figure 3.3 shows a screenshot of the home automation web application.

Fig. 3.3 Home automation web application screenshot

Case Study on IoT System for Weather Monitoring

The purpose of the weather monitoring system is to collect data on environmental conditions such as temperature, pressure, humidity and light in an area using multiple end nodes. The end nodes send the data to the cloud where the data is aggregated and analysed.

Figure 3.4 shows the process specification for the weather monitoring system. The process specification shows that the sensors are read after fixed intervals, and the sensor measurements are stored.

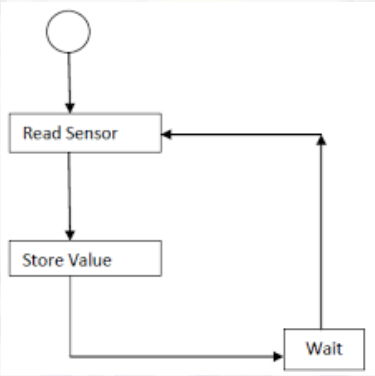

Fig.3.4 Process specification for weather monitoring IoT system

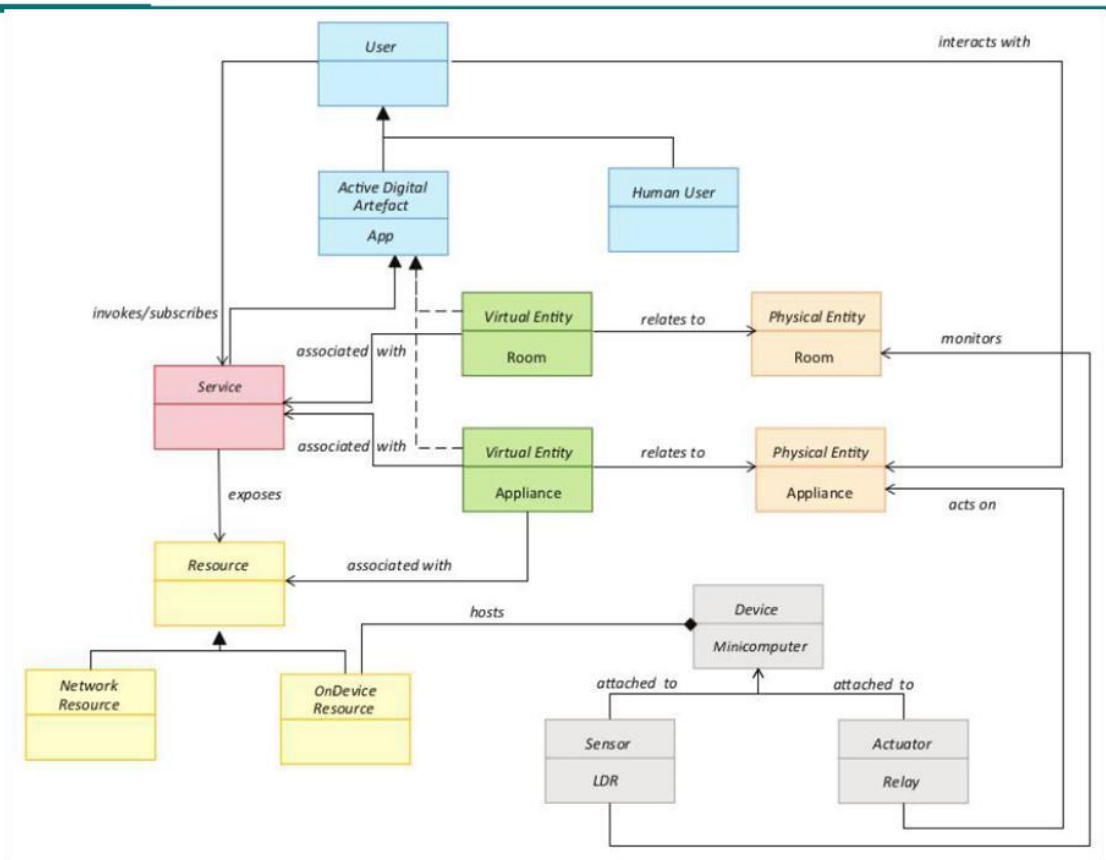

Fig.3.5 Domain model for weather monitoring IoT system

Figure 3.5 shows the domain model for the weather monitoring system. The physical entity is the environment which is being monitored. There is a virtual entity for the environment. Resources are software components which can be either on-device or network-resources. Services include the controller service that monitors the temperature, pressure, humidity and light and sends reading to the cloud.

Chapter 2: IoT Systems - Logical Design using Python: Introduction

Python is a general-purpose high level programming language and suitable for providing a solid foundation to the reader around cloud computing.

The main characteristics of Python are:

1. Multi-paradigm programming language.
2. Python supports more than one programming paradigms including object- oriented programming and structured programming.
3. Interpreted Language.
4. Python is an interpreted language and does not require an explicit compilation step.
5. The Python interpreter executes the program source code directly, statement by statement, as a processor or scripting engine does.

6. Interactive Language
7. Python provides an interactive mode in which the user can submit commands at the Python prompt and interact with the interpreter directly.

Installing Python

Python is a highly portable language that works on various platforms such as Windows, Linux, Mac etc.

Windows:

Download the Installer:
Visit the official Python downloads page (python.org/downloads) and download the latest stable Windows installer (e.g., "Windows installer (64-bit)").

Run the Installer:
Execute the downloaded .exe file.

Add to PATH:
Crucially, on the first screen of the installer, check the box that says "Add Python X.Y to PATH" (where X.Y is the version number). This allows you to run Python commands from any command prompt.

Install:
Click "Install Now" (or customize features if desired) and follow the prompts to complete the installation.

Verify:
Open a Command Prompt or PowerShell and type python --version to confirm the installation and version.

Linux:

Many Linux distributions come with Python pre-installed. However, it might be an older version or Python 2. To ensure you have Python 3 and its necessary components: Update Package Lists.

Code
```
sudo apt update # For Debian/Ubuntu-based systems
sudo yum update # For Red Hat/CentOS-based systems
```

Install Python 3 and pip.
Code
```
sudo apt install python3 python3-pip # For Debian/Ubuntu-based systems
sudo yum install python3 python3-pip # For Red Hat/CentOS-based systems
```

Verify: In the terminal, run python3 --version and pip3 --version.

Python Data Types and Data structures

Numbers: Number data types are used to store numeric values. Numbers are immutable data types, therefore changing the values of a number data type results in a newly allocated object.

```
#Integer
>>>a=5 >>>type(a)
<type 'int'>

#Floating point
>>>b=2.5
>>>type(b)
<type 'float'>

#Long
>>>x=243577889L
>>>type(x)
<type 'long'>

#Complex
>>>Y=2+5j
>>>y
(2+5j)
>>>type(y)
<type 'complex'>
>>>y.real
2
>>>y.imag
5

#Addition
>>>c=a+b
>>>c
7.5
>>>type(c)
<type 'float'>

#Subtraction
>>>d=a-b
>>>d
2.5
>>>type(d)
<type 'float'>

#Multiplication
```

```
>>>e=a*b
>>>e
12.5
>>>type(e)
<type 'float'>

#division
>>>f=b/a
>>>f
0.5
>>>type(f)
<type 'float'>

#Power
>>>g=a**2
>>>g
25
```

Strings

A string is simply a list of characters in order. There are no limits to the number of characters you can have in a string. A string which has zero characters is called an empty string.

Working with Strings in Python:

```
#Create String
>>>s= "hello World!"
>>>type(S)
<type 'str'>

#String Concatenation
>>>t= "This is sample program".
>>>r=s+t
>>>r
'Hello World! This is sample program'.

#Get length of string
>>>len(s)
12

#Convert String to integers
>>>X= "100"
>>>type(s)
<type 'str'>
>>>y=int(x)
>>>y
```

100

#Print String
>>>print S
Hello World!

#Formatting Output
>>>print "The string(%s) has %d characters "%(S, len(S)))
The string (Hello World!) has 12 characters

#Convert to upper/lower case
>>>S.upper()
'HELLO WORLD'
>>>S.lower() 'hello world!'

#Accessing Sub-String
>>>S[0]
'H'
>>>S[6:]
'world!'
>>>S[6:-1]
'world'

#Strip: Return a copy of the string with
#the leading and trailing characters removed.
>>>S.strip("!")
'Hello World'

Lists

List is a compound data type used to group together other values. List items need not all have the same type. A list contains items separated by commas and enclosed within square brackets.

Working with lists in Python

>>>fruits=['apple', 'orange', 'banana', 'mango']
>>>type(fruits)
<type 'list'>
>>>len(fruits)
4

>>>fruits[1]
'orange'
>>>fruits[1:3]
['orange' , 'banana']
>>>fruits[1:]

```
['orange','banana','mango']

#Appending an item to a list
>>>fruits.append('pear')
>>>fruits
['apple', 'orange', 'banana', 'mango', 'pear']

#Removing an item from a list
>>>fruits.remove('mango')
>>>fruits ['apple', 'orange', 'banana', 'pear']

#Inserting an item to a list
>>>fruits.insert(1, 'mango')
>>>fruits ['apple', 'mango', 'orange', 'banana', 'pear']

#Combining lists
>>>vegetables=['potato', 'carrot', 'onion', 'beans', 'radish']
>>>vegetables
['potato', 'carrot', 'onion', 'beans', 'radish']
>>>eatable=fruits+vegetables
>>>eatable
['apple', 'mango', 'orange', 'banana', 'pear', 'potato', 'carrot', 'onion' ,'beans', 'radish']

#Mixed data types in a list
>>>mixed=['data', 5, 100.1, 8287398L]
>>>type(mixed)
<type 'list'>
>>>type(mixed[0])
<type 'str'>
>>>type(mixed[1])
<type 'int'>
>>>type(mixed[2])
<type 'float'>
>>>type(mixed[3])
<type 'long'>

#It is possible to change individual elements of a list
>>>mixed[0]=mixed[0]+ "items"
>>>mixed[1]=mixed[1]+1
>>>mixed[2]=mixed[2]+ 0.05
>>>mixed
['data items', 6, 100.14999999999999, 8287398L]

#Lists can be nested
>>>nested=[fruits, vegetables]
```

```
>>>nested
[[‘apple’, ‘mango’, ‘orange’, ‘banana’, ‘pear’],[‘potato’, ‘carrot’, ‘onion’, ‘beans’, ‘radish’]]
```

Tuples

A Tuple is a sequence data type that is similar to the list. A Tuple consists of values separated by commas and enclosed within parentheses. Unlike lists, the elements of tuple cannot be changed, so tuple can be thought of as read-only lists.

Working with Tuples in Python:

```
>>>fruits=("apple", "mango", "banana", "pineapple")
>>>fruits
(‘apple’, ‘mango’, ‘banana’, ‘pineapple’)
>>>type(fruits)
<type ‘tuple’>

#Get length of tuple
>>>len(fruits)
4

#Get an element from a Tuple
>>>fruits[0]
‘apple’
>>>fruits[:2]
(‘apple’, ‘mango’)

#Combining tuples

>>>vegetable= (‘potato’, ‘carrot’, ‘onion’, ‘radish’)
>>>eatebles= fruits+vegetable
>>>eatables
(‘apple’, ‘mango’, ‘banana’, ‘pineapple’, ‘potato’, ‘carrot’, ‘onion’, ‘radish’)
```

Dictionaries

Dictionary is a mapping data type or a kind of hash table that maps key to values. Keys in a dictionary can be of any data type, through numbers and strings are commonly used for keys. Values in a dictionary can be any data type or object.

Working with Dictionaries with Python:

```
>>>students={‘name’: ‘Mary’, ‘id’: ‘8776’, ‘major’: ‘cs’}
>>>students
{‘major’: ‘cs’, ‘name’: ‘Mary’, ‘id’: ‘8776’}
>>>type(student) <type ‘dict’>

#Get length of a Dictionary
```

```
>>>len(student)
3

#Get the value of a key in dictionary
>>>student['name']
'Mary'

#Get all items in a dictionary
>>>student.tems()
[('major','CS'), ('name', 'Mary'), ('id', '8776')]

#Get all keys in a dictionary
>>>student.keys()
['major', 'name', 'id']

#Get all values in a dictionary
>>>student.values()
[ 'cs', 'Mary', '8776']

>>>students
{'major': 'cs', 'name': 'Mary', 'id': '8776'}

#A value in a dictionary can be another dictionary
>>>student1= {'major': 'ece', 'name': 'David', 'id': '9876'}
>>>students={'1':student, '2':student1}
{'1':{'major':'cs', 'name': 'Mary', 'id': '8776'},
'2': { 'major': 'ece', 'name': 'David', 'id': '9876'}}

#Check if dictionary has a key
>>>student.has_key('name')
True
>>>student.has_key('grade')
False
```

Type Conversions

Type Conversion examples

```
#Convert to String
>>>a=10000
>>>str(a)
'10000'

#Convert to int
>>>b= "2013"
>>>int(b)
2013
```

```
#Convert to float
>>>float(b)
2013.0

#Convert to long
>>>long(b)
2013L

#Convert to list
>>>s= "aeiou"
>>>list(s)
['a', 'e', 'i', 'o', 'u']

#Convert to Set
>>>x=['mango', 'apple', 'banana', 'mango', 'banana']
>>>set(x)
set(['mango', 'apple', 'banana'])
```

Control Flow

Let us look at the control flow statements in Python.

if

The if statement in Python is similar to the if statement in other languages.

examples:

```
>>>a=25**5
>>>if a>10000:
      print "More"
else:
      print "Less"
```

More

```
>>>if a>10000:
      If a<100000:
          print "Between 10k and 100k"
      else:
          print "More than 100k"
elif a==10000:
     print "Equal to 10k"
else:
     print "Less than 10k"
```

More than 100k

```
>>>S= "Hello World"
>>>if "world" in S:
    S=S+ "!"
    Print S
```

Hello World!

```
>>>student ={‘name’: ‘mary’, ‘id’: ‘8776’}
>>>if not student.has_key(„major“):
Student[‘major’]= ‘cs’
>>>student
{‘major’: ‘cs’, ‘name’: ‘Mary’, ‘id’: ‘8776’}
```

for

The for statement in python iterates over items of any sequence (list, String, etc.) in the order in which they appear in the sequence. This behaviour is different from the for statement in other languages such as C in which an initialization, incrementing and stopping criteria are provided.
Example:

```
helloString= "Hello World"
fruits=[‘apple’, ‘orange’, ‘banana’, ‘mango’]
student = ‘name’ : ‘Mary’, ‘id’ : ‘8776’ , ‘major’: ‘CS’

#looping over characters in a string
for c in helloString:
        print c

#looping over items in a list
i=0
for item in fruits:
        print “Fruit-%d: %s” %(i, item)
        i=i+1

#looping over keys in a dictionary
for key in student:
        print”%s: %s” % (key, student[key])
```

while

The while statement in Python executes the statements within the while loop as long as the while condition is true.

Example:

```
#prints even numbers upto 100
>>>i=0
>>>while i<<=100:
```

```
if i%2 == 0:
print i
i=i+1
```

range

the range statement n Python executes the statements within the while loop as long as the while condition is true.

Example:

```
#Generate a list of numbers from 0-9
>>>range (10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

#Generate a list of numbers from 10-100 with increments of 10
>>>range(10,110,10)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

break/continue

The break and continue statements in python are similar to the statement in C. The break statement breaks out of the for/while loop where as the continue statement continues with the next iteration.

Example:

```
#Break Statement example
>>>y=1
>>>for x in range(4, 256, 4):
        y=y*x
        if y>512:
            break
        print y

4
32
384

#continue Statement example
>>>fruit=['apple', 'orange' , 'banana' , 'mango']
>>> for item in fruits :
        if item == "banana" :
            continue
        else:
            print item
```

```
apple
orange
mango
```

pass

The pass statement in python is a null operation. The pass statement is used when a statement is required syntactically but you do not want any command or code to execute.

Example:

```
fruits = [ ‘apple’, ‘orange’, ‘banana’, ‘mango’]
for item in fruits:
  if item == "banana":
        pass
  else: print item
```

```
apple
orange
mango
```

Functions

A function is a block of code that takes information in (in the form of parameters), does some computation, and returns a new piece of information based on the parameter information. A function in python is a block of code that begins with the keyword def followed by the function name and parenthesis. The function parameters are enclosed within the parenthesis. The code block within a function begins after a colon that comes after the parenthesis. The code block within a function begins after a colon that comes after the parenthesis enclosing the parameters. The first statement of the function body can optionally be a documentation string or docstring.

Example:

```
students= {‘1’: {‘name’: ‘Bob’, ‘grade’: 2.5},
‘2’: {‘name’: ‘Mary’, ‘grade’ :3.5},
‘3’: {‘name’: ‘David’, ‘grade’ :4.2},
‘4’: {‘name’: ‘John’, ‘grade’ :4.1},
‘5’: {‘name’: ‘Alex’, ‘grade’ :3.8}}

def averageGrade (students):
        "This function computes the average grade"
        sum = 0.0
        for key in student:
                sum=sum +student[key][‘garde’]
                average= sum/len(students)
                return average
```

```
avg= averageGrade(students)
print " The average grade is: %0.2f" %(avg)
```

Function can have default values of the parameters. If a function with default values is called with fewer parameters or without any parameter, the default values of the parameter are used as shown in the example.

Example of function with default arguments

```
>>>def displayFruits(fruits = ['apple', 'orange']):
print "There are %d fruits in the list "% (len(fruits))
for item in fruits:
    print item

#Using default arguments
>>>displayFruits()
Apple
Orange

>>>fruits=['banana' , 'pear', 'mango']
>>>displayFruits(fruits)
Banana
Pear
Mango
```

All parameter in the python functions are passed by reference. Therefore, if a parameter is changed within a function the change also reflected block in the calling function.

Example of passing by reference

```
>>>def displayFruits(fruits):
print "There are %d fruits is the list" % (len(fruits))
for item in fruits:
print item
print "Adding one more fruit"
fruits.append('mango')

>>>fruits=['banana', 'pear', 'apple']
>>>displayFruits(fruits)
There are 3 fruits in the list
Banana
Pear
Apple
Adding one more fruits

>>>print "There are %d fruits in the list"%(len(fruits))
```

There are 4 fruits in the list

Function can also be called using keyword arguments that identify the arguments by the parameter name when the function is called.

examples of keyword arguments

```
>>>def printStudentRecords(name, age=20, major= 'CS'):
        print "Name:" +name
        print "Age:" +str(age)
        print "Major:" + major

#This will give error as name is required argument
>>>printStudentRecords()
Traceback (most recent call last):
File "<stdin>", line1, in <module>
TypeError: printStudentRecords() takes at least 1 argument ( 0 given)
>>>printStudentRecords(name= 'Alex')
Name: Alex
Age:20
Major:CS

>>>printStudentRecords(name= 'Bob', age=22, major= 'ECE')
Name: Bob
Age:22
Major:ECE

>>> printStudentRecords(name= 'Alan', major= 'ECE')
Name: Alan
Age:20
Major: ECE

#name is a format argument.
#**kwargs is a keyword argument that receives all
#arguments except the formal argument as a dictionary
>>>def student(name, **kwargs):
        print "Student Name:" +name
        for key in kwargs:
                print key + ':' + kwargs[key]

>>>Student(name= 'Bob', age= '20', major= 'CS')
Student Name: Bob
Age :20
Major: CS
```

Python functions can have variable length arguments. These variable length arguments are passed as a tuple to the function with an argument prefixed with asterix(*)

Example of variable length arguments

```
def student(name, *varargs):
    print "Student Name:" +name
    for item in varargs :
      print item
```

```
>>>Student ( 'Nav')
Student Name : Nav
```

```
>>>Student ('Amy' , 'Age: 24')
Student Name: Amy
Age:24
```

```
>>>Student('Bob' , 'Age:20', 'Major: CS')
Student Name: Bob
Age:20
Major: CS
```

Modules

Python allows organizing of the program code into different 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.

Module Student

```
def averageGrade(student):
sum=0.0
for key in students:
sum=sum+students[key]['grade']
average =sum/len(students)
return average
def printRecords(students):
print "There are %d students" %(len(students))
i=1
for key in students:
print "Student -%d :" %(i)
print "Name:" +Students[key]['name']
print "Grade:" +str(students[key]['grade'])
i=i+1
```

Using module Student

```
>>>import student

>>>student= '1': 'name': 'Bob', 'grade':2.5,
'2': 'name': 'mary', 'grade':3.5,
'3': 'name': 'David', 'grade':4.2,
'4': 'name': 'John', 'grade':4.1,
'5': 'name': 'Alex', 'grade':3.8

>>>Student.printRecords(Students)
There are 5 students
Student - 1:
Name: Bob
Grade:2.5

Student - 2:
Name: David
Grade:4.2

Student - 3:
Name: Mary
Grade:3.5

Student - 4:
Name: Alex
Grade: 3.8

Student - 5:
Name: John
Grade: 4.1

>>>Avg= Student averageGrade (students)
>>>print "The average grade is: % 0.2f" %(avg)
3.62
```

The import keyword followed by module name imports all the functions in the module. If you want to use only a specific function it is recommended to import only that function using the keyword from as shown in the example. Importing a specific function from the module

```
>>>from student import averageGrade

>>> Students = „1" : 'Name' : 'Bob', 'Grade' : 2.5,
'2' : 'Name' : 'Mary' , 'Grade' : '3.5',
'3':'Name':  'David' , 'grade': '4.2,
'4':'Name': 'John', 'grade': '4.1',
'5':'Name': 'Alex', 'grade': 3.8
```

```
>>>avg =averageGrade(students)
>>>print "The average grade is :%0.2f" %(avg)
3.62
```

Python comes with a number of standard,modules such as system related modules(sys), OS related module(OS), mathematical modules(math, fractions). Standard modules are available in the python documentation.

Listing all names defined in a module

```
>>>import email
>>>dir(email)
['charset' , 'encoder', 'errors', 'feedparser', 'Generator', 'Header', 'Iterators', 'LazyImporter', 'MIMEaudio', 'MIMEImage', 'MIMEMessage', 'MIMEMultipart', 'MIMENonMultipart', 'MIMEText', 'Message', 'Parser', 'utils', '_LOWERNAMES', '_MINENAME', '_all_', '_builtins_', '_doc_', '_file_', '_name_', '_package_', '_path_', '_version_', '_name', 'base64MIME', 'email', 'importer', 'message_from_file', 'message_from_string', 'mime', 'quoprimeMINE', 'Sys']
```

Packages

Python package is hierarchical file structure that consists of modules and subpackages. Packages allow better organization of modules related to a single applications environment. For example, below show the listing of the skimage package that provides image processing algorithms. The package is organized into a root directory (skimage) with sub-directories (color, draw,etc) which are sub_packages within the skimage package. Each directory contains a special file named_init_. Py which tells python to treat directories as packages. This file can either be an empty file or contain some initialization code for the package.

Skimage package listing

```
Skimage/            Top level package
   __init__.py      Treat directionary as a package
   Color/           color subpackage
            __init__.py
          Colorconv.py
          Colorlabel.py
          rgb_colors.py
 Draw/              draw subpackage
          __init__.py
          draw.py
          setup.py
exposure/           exposure subpackage
          __init__.py
          __adapthist.py
```

```
        Exposure.py
Feature/                feature subpackage
        __init__.py
        __brief.py
        __daisy.py
```

File Handling

Python allows reading and writing to files using the files 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.

Shows an example of reading an entire file with read function. 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
Python supports more than one programming paradigms
including object-oriented programming and structured programming.
Python is an interpreted language and does not require an explicit compilation step.
>>>fp.close()

Shows an example of reading line by line from a file using the readline function. Example of reading line by line

>>>fp.close()
>>>fp=open('file.txt', 'r')
>>>print "Line-1: "+fp.readline()
Line-1:python supports more than one programming paradigms including object-oriented programming and structure programming.

>>>print "Line-2:" +fp.readline()
Line-2: Python is an interpreted language and does not require an explicit compilation step.

>>>fp.close

Shows an example of reading lines of a file in a loop using the read lines function. Example of reading line in a loop

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

Python supports more than one programming paradigms including object- Oriented programming and structured programming.

Python is an interpreted language and does not require an explicit compilation step.

Shows an example of reading a certain number of bytes from a file using the read(size) function Example of reading a certain number of byte

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

Shows an example of getting the current position of read using the tell function. 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 of file object at 0x0000000000002391390>
>>>fp.close
```

Shows an example of seeking to a certain position in a file using the seek function. Example of seeking a certain position

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

Shows an example of writing a file using the write function. Example of writing to a file

```
>>>fp=open ('file1.txt', 'w')
>>>content = 'This is an example of writing to a file in pyhton'.
>>>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.

Example of manipulating with date

```
>>> from datetime import date
>>>now =date.today()
>>>print "Date:" + now.strftime("%m-%d-%y")
Date:07-24-13
>>>print "Date 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
47
```

The time module in python provides various time-related functions. Example 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_ec=51, tm_wday=2, tm_yday=205, tm_isdst=0)
>>>time.asct(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.'
```

Classes

Python is an Object-Oriented Programming (OOP) language. Python provides all the standard features of object-oriented programming such as classes, class variables, class methods, inheritance, function overloading and operator overloading.

Class

A class is simply a representation of a type of object and user-defined prototype for an object that is composed of three things: a name, attribute and operations/methods.

Instance/object

Object is an instance of the data structure defined by a class.

Inheritance

Inheritance is the process of forming a new class from an existing class or base class.

Function Overloading

Function overloading is a form of polymorphism that allows a function to have different meanings, depending on its context.

Operator overloading

Operator overloading is a form of polymorphism that allows assignment of more than one function to a particular operator.

Function overriding

Function overriding allows a child class to provide a specific implementation of a function that is already provided by the base class. Child class implemented of the overridden function has the same name, parameters and return type as the function in the base class.

Shows an example of a class. The variable student Count is a class variable that is shared by all instances of the class student and is accessed by student and is accessed by student. Student Count. The variables name, id and grades are instance variables which are specific to each instance of the class. There is a special method by the name_init_() which is the constructor.

The class constructor initializes new instances when it is created. The function _del_() is the class destructor.

Example of a class

```
>>>class student:
        StudentCount=0
        Def_init_(self,name,id):
                Print "Constructor called"
                Self.name=name
                Self.id=id
                Student.studentCount = Student.studentCount+1
                Self.grades=

        def_del_(self):
                print "Destruction called"

        def getStudentCount(self):
                return student.studentCount

        def addGrade (self,key,value):
                self.grades(key)=value

        def getGrade(self,key):
                return self.grades(key)

        def printGrades(self):
```

```
            for key in self.grades:
                print key + ":"+Self.grades(key)
```

```
>>> S = Student ('Steve', '98928')
Constructor called
>>> S.addGrade('Math', '90')
>>>S.addGrade('physics', '85')
>>>S.printGrade()
Physics: 85
Math: 90
>>>mathgrade=S.getGrade('Math')
>>>print mathgrade
90
>>>count =S.getStudentCount()
>>>print count
1
>>>del s
Destructor called
```

Shows an example **of class inheritance**. In this example shape is the base class and circle is the derived class. The class circle inherits the attributes of the shape class. The child class Circle overrides the methods and attributes of the base class (eg. draw() function defined in the base class shape is overridden in child class Circle). It is possible to hide some class attributes by naming them with a double underscore prefix. For example, _label attribute is hidden and cannot be directly accessed using the object (cir._label gives an error). To hide the attributes with double underscore prefix, python changes their names internally and prefixes the class name(e.g._label is changed to _Circle_label).

Examples of class inheritance:

```
>>>class Shape:
    def _init_(self):
        print "Base class constructor"
        self. color = 'Green'
        self. lineWeight = 10.0
    def draw(self):
        print "Draw -to be implemented"
    def SetColor(self,c):
        self.color=c
    def getColor(self):
        return self.color
    def setLineWeight(self,lwt):
        self.LineWeight=lwt
```

```
        def setLineWeight(self):
                return self.LineWeight

>>>Class Circle (Shape):
        def_init_(self,c,r):
                print "Circle class constructor"
                self.center=c
                self.radius=r
                self.color= 'Green'
                self.lineWeight=10.0
                self._label = 'Hidden circle label'
        def setCenter (self,c):
                self.centre=c
        def getCenter(self):
                return self.center
        def setRadius(self,r):
                self.radius=r
        def getRadius(self):
                return self.radius
        def draw(self)
                print "Draw circle (overridden function)"

>>>class point:
        def_init_(self,x,y):
                self.x Cooradinate=x
                self.y Cooradinate=y

        def setx Coordinate(self,X):
                self.X Cooradinate=X
        def getXCoordinate (self):
                return self.xcoordinate
        def setyCoordinate(self,y):
                self.yCoordinate =y
        def getycoordinate(self):
                return self.ycoordinate

>>>p=point(2,4)
>>>circ =circle(p,7)
Child class constructor
>>>circ.getColor()
'Green'
>>>circ.setColor('Red')
>>>circgetColor()
'Red'
```

```
>>>circ.getLineWeight()
10.0
>>>circ.getCenter().getXCoordinate()
2
>>>circ.getCenter().getYCoordinate()
4

>>>circ.draw()
Draw Circle (overridden function)
>>>circle.radius
7
>>>circ._label Traceback (most recent call last):
File "<stdin>", line1 , in<module>
AttributeError: Circle instance has no attribute '_Label'
>>>circ._Circle_label
'Hidden circle label'
```

Python Packages of Interest for IoT

1. **JSON**

Javascript object Notation (JSON) is an easy to read and write data-interchange format. JSON is used as an alternative to XML and is wasy for machines to parse and generate. JSON is built on two structure- a collection of name-value pairs (e.g. a python dictionary) and ordered lists of values (e.g. a python list).

JSON format is often used for serializing and transmitting structure data over a network connection, for example, transmitting data between a server and web application.

JSON Example -A Twitter tweet object

```
{
"created_at":
"sat Jun 01 11:39:43+000 2013",
"id" : 340794787059875841,
"text": "What a bright and sunny day today!",
"truncated" : false,
"in_reply_to_status_id": null,
"user": {
        "id":383825039,
```

"name": "Harry",
"followers_count":316,
"friends_count": 298,
"listed_count": 0,
"create_at": "sun oct 02 15:15:61 +0000 2011",
"favourites_count":251,
"statuses_count":1707,
:
"notifications": null
},
"geo":{
"type": "point",
"coordinate":[26.92782727,75.78908449]
},
"coordinates":{
"type":"point",
"coordinates":[75.78908449, 26.92782727]
},
"place":null,
"contributors": null,
"retweet_count":0,
"favourite_count":0,
"entities": {
"hastags":[ ],
"symbol": [ ],
"urls": [ ],
"user_mentions": [ ]
},
"favourite": false,
"retweeted":false,
"filter_level": "medium",
"lang": "nl"
}

Exchange of information encoded as JSON involves encoding and decoding steps. The python JSON package[109] provides functions for encoding and decoding JSON.

Encoding and Decoding JSON in python:

```
>>>import json

>>>meassage={
"Created": "wed Jun 31 2013",
"id": "001",
```

"text": "This is a test message.", }

```
>>>json.dumps(message)
 '{"text": "This is a test message." "id" : "001", "created": "Wed Jun 31 2013"}'

>>>decodeMsg = json.loads ('{"text": " This a test message", "id": "001", "created": "Wed Jun 31 2013"}'}

>>>decode Msg ['created']
u 'Wed Jun 31 2013'
>>>decodedMsg ['text']
U 'This is a test message'.
```

2. XML

XML (Extensible Markup Language) is a data format for structured document interchange. Shows an example of an XML file. In this section you will learn how to parse, read and write XML with python. The python minidom library provides a minimal implementation of the document object Model interface and has an API similar to that in other language. Shows a python program for parsing an XML file.

Shows a python program for creating an XML file. XML example

```
<?xml version = "1.0"?>
<catalog>
<plant id= '1'>
<common> Bloodroot</common>
<botanical>Sanguinaria Canadensis </botanical>
<Zone> 4</Zone>
<light> Mostly Shady </light>
<price> 2.44 </price>
<availablility> 031599</availability>
</plant>
<plant id= '2'>
<common> Columbine </common>
<botanical> Aquilegia Canadensis </botanical>
<zone> 3</zone>
<light> Mostly Shady </light>
<price> 9.37 </price>
<availability> 030699 </availability>
</plant>
<plant id= '3'>
<common> Marsh Marigold </common>
<botanical> Caltha palustris </botanical>
 <zone>4</zone>
```

```
<light>Mostly Sunny</light>
<price> 6.81 </price>
<availability> 051799 </availability>
</plant>
</catalog>
```

Parsing an XML file in python

```
From xml.dom.minidom import parse
dom= parse ("test.xml")
For node in dom.getElementByTagName('plant'):
        Id= node.gteAttribute('id')
        print "plantID:", id
        Common=node.getElementsByTagName('common')[0].childNodes[0].nodeValue
        print "Common:", common
        botanical =node.GetElementsByTagName('botanical')[0].childNodes[0].nodeValue
        print "Botanical:", botanical zone= node.getElementsByTagName('zone')[0]
        childNodes[0].nodeValue
        print "Zone:",Zone
```

Creating an XML file with python

```
#python example to create the following XML:
#'<?xml version = "1.0"> <class><student>
#<Name>Alex</Name> <Major> ECE </Major> </student></class>

From xml.dom.minidom import Document
doc= Document()

#Create base element
Base= doc.createElement('class')
doc.appendChild(base)

#create an entry element
Entry=doc.createElement('Student')
Base.appendChild(entry)

#create an element and append to entry element
name= doc.createElement('Name')
nameContent = doc.CreateTextNode('Alex')
name.appendChild(nameContent)
entry.appendChild(name)

#create an element and append to entry element
major=doc.createElement('Major')
majorContent=doc.createTextNode('ECE')
major.appendChild(majorContent)
```

```
entry.appendChild(major)

fp=open('foo.xml', 'w')
doc.writexml()
fp.close()
```

3. HTTPLib & URLLib

HTTPLib2 and URLLib2 are python libraries used in network/internet programming. HTTPLib2 is an HTTP client library and URRLib2 is a library for fetching URLS.

Shows an example of an HTTP GET request using the HTTPLib. The variable resp contains the response headers and content contains the content retrieved from the URL.

HTTP GET request example using HTTPLib

```
>>> import httplib2
>>> h=httplib2.HTTP()
>>> resp,content =h.request("http://example.com", "GET")

>>>resp { 'status' : '200', 'content-length': '1270', 'content-location': 'http://example.com',
'x-cache': 'HIT', 'acccept_range': 'bytes', 'server' : 'ECS (cmp/F858', 'last_modified':
'Thu,25 APR 2013 16:13:23 GMT', 'etag': ' "780602-4f6-4db3lb2978 eco"', 'date': 'wed, 31
Jul 2013 12:36:05 GMT', 'content-type': 'text/html ; charset= UTF-8'}

>>>content
'<!doctype html> \n <html>\n <head> \n
<title> example Domain </title>\n\n
<meta charset = "utf-8"/>\n:
```

Show an HTTP Request example using URLLib2. A request object is created by calling urllib2. Request with the URL to fetch as input parameter. Then urllib2.urlopen is called with the request object which returns the response object for the requested URL. The response object is read by calling read function.

HTTP request example using URLLib2

```
>>>import urllib2
>>>
>>>req=urllib2.Request('http://example.com')
>>>response=urllib2.urlopen(req)
>>>response_page=response.read()
>>.response_page
'<!doctype html>\n <html>\n <head>\n
<title> Example Domain </title>\n\n
<meta charset = "utf-8"/>\n
```

Shows an example of an HTTP POST request. The data in the POST body is encoded using the urlencode function from urllib.

HTTP POST example using HTTPLib2

```
>>>import httplib2
>>>import urllib
>>>h = httplib2.Http()
>>>data = {'title': 'Cloud computing'}
>>>resp, content =h.request("http://www.htmlcodetutorial.com/cgi-bin/mycgi.p1", "POST", urllib.urlencode(data))
>>>resp
{'Status': '200', 'transfer-encoding': 'chunked', 'server': 'Apache/2.0.64(unix) mod\-ssd/ 2.0.64 openSSL/0.9.70 mod_auth_passthrought/2.1 mod_bwlimited/1.4 frontpage/5.0.2.2635 PHP/5.3.10', 'connection': 'close', 'date': 'Wed, 31 Jul 2013 12:41:20 GMT', 'content-type': 'text/html; charset=ISO -8859-1'}

>>>content
'<HTML> \n <HEAD> \n <TITLE> Idocs Guide to HTML: My CGI </TITLE>\n <\HEAD>:
```

Shows an example of sending data to a URL using URL using URLLib2 (e.g an HTML from submission). This example is similar to the HTTP POST example and uses URLLib2 request object instead of HTTPLib2.

Example of sending data to a URL

```
>>>import urllib
>>>import urllib2
>>>
>>> url = 'http://www.htmlcodetutorial.com/cgi-bin/mycgi.p1'
>>>values ={'title' : 'cloud computing', ... 'language' : 'python'}
>>>
>>>data =urllib.urlencode(values)
>>>req=urllib2.Request(url,data)
>>>response =urllib2.urlopen(req)
>>>the_page =response.read()
>>>the_page '<HTML>/n <HEAD>\n <TITLE>Idocs Guide to HTML: My CGI </TITLE>\n
</HEAD>
```

4. SMTPLib

Simple Mail Transfer Protocol (SMTP) is a protocol which handle sending email and routing e-mail between mail servers. The python smtplib module provides an STMP client session object that can be used to send email.

Shows a python example of sending email from a Gmail account. The string message contains the email message to be sent. To send email from a Gmail account the Gmail STMP server is specified in the server string.

To send an email, first a connection is established with the STMP server by calling smtplib.SMTP with SMTP server name and port. The username and password provided are then used to login into the server. The email is then sent by calling server.sendmail function with the fom address, to address list and message as input parameters.

Python example of sending email

```
Import smtplib

from_email= '<enter-gmail-address>'
recipients_list =['<enter_sender_email>']
cc_list=[]
subject= 'Hello'
message = 'This is a test message'
username= '<enter -gmail_username>'
password= '<enter-gmail_password>'
server = 'smtp.gmail.com:587'

def sendemail (from_addr, to_addr_list, cc_addr_list, subject, message, login, password, smtpserver):

header= 'from: %s\n' %from_addr
header+ = 'To: %s\n' % ','.join(to-addr_list)
header+= 'Cc: %s\n' % ','.join(cc_addr_list)
header+= 'subject : %s\n\n' %subject
message =header+message

server =smtplib.SMTP(smtpserver)
server.starttls()
server.login (login, password)
problems = server.sendmial (from_addr, to_addr_list, message)
server.quit()

#send email
Sendmail(from_email, recipients_list, cc_list, subject, message, username, password, server)
```