DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

INTERNET OF THINGS (J32ON)

[R20]

B.TECH ECE
(III YEAR – II SEM)
(2022-23)

J. B. INSTITUTE OF ENGINEERING AND TECHNOLOGY
(UGC AUTONOMOUS)
Accredited by NBA & NAAC,
Approved by AICTE & Permanently affiliated to JNTUH
Bhaskar Nagar, Yenkapally(V), Moinabad(M), Ranaga Reddy(D),Hyderabad – 500 075**,**
Telanagana, India.

AY 2020-21 Onwards	J. B. Institute of Engineering and Technology (UGC Autonomous)	B. Tech III Year – II Sem		
Course Code: J32ON	**INTERNET OF THINGS** (Open Elective – II)	**L**	**T**	**P/D**
Credits: 3		3	0	0

Pre-Requisites: Nill

Course Objectives:
Students will learn to

1. Understand the basic building blocks of IoT.
2. Analyze the difference between M2M and IoT along with IoT system Management
3. Extend the knowledge in Logical Design of IoT System using Python.
4. Acquire knowledge about IoT Physical Devices and End points.
5. Identify the IoT Physical Servers and cloud offerings.

Module 1:
Unit 1 Introduction to Internet of Things:
Definition and Characteristics of IoT, Physical Design of IoT –IoT Protocols, IoT communication models, IoT Communication APIs
Unit 2 IoT enabled Technologies:
Wireless Sensor Networks, Cloud Computing, Big data analytics, Communication protocols, Embedded Systems, IoT Levels and Templates, Domain Specific IoTs – Home, City, Environment, Energy, Retail, Logistics, Agriculture, Industry, health and Lifestyle.

Module 2:
Unit 1 IoT and M2M:
Software defined networks, network function virtualization, difference between SDN and NFV for IoT
Unit 2 Basics of IoT System:
Basics of IoT System Management with NETCOZF, YANG- NETCONF, YANG, SNMP NETOPEER.

Module 3:
Unit 1 Introduction to Python:
Language features of Python, Data types, data structures, Control of flow, functions, modules, packaging, file handling, data/time operations, classes, Exception handling.

Unit 2 Python packages:
JSON, XML, HTTPLib, URLLib, SMTPLib.

Module 4:
Unit 1 IoT Physical Devices and Endpoints:
Introduction to Raspberry PI-Interfaces (serial, SPI, I2C) Programming.
Unit 2 Python program with Raspberry PI-1:
Python program with Raspberry PI with focus of interfacing external gadgets, controlling output, reading input from pins.

Module 5:
Unit 1 Python program with Raspberry PI-2:
Python program with Raspberry PI with focus of interfacing external gadgets.
Unit 2: Controllingoutput, reading input from pins.

Text Books:
1. Internet of Things - A Hands-on Approach, ArshdeepBahga and Vijay Madisetti, Universities Press, 2015, ISBN: 9788173719547
2. Getting Started with Raspberry Pi, Matt Richardson & Shawn Wallace, O'Reilly (SPD), 2014, ISBN: 9789350239759

References:
1. Internet of Things by Jeeva Bose 1st edition, Khanna publishing.

Course Outcomes:
Students will be able to
1. **Understand** the basic building blocks of IoT.
2. **Analyze** the difference between M2M and IoT along with IoT system Management
3. **Extend** the knowledge in Logical Design of IoT System using Python.
4. **Acquire** knowledge about IoT Physical Devices and End points.
5. **Identify** the IoT Physical Servers and cloud offerings

UNIT-I
INTRODUCTION OF IOT

IoT comprises things that have unique identities and are connected to internet. By 2020 there will be a total of 50 billion devices /things connected to internet. IoT is not limited to just connecting things to the internet but also allow things to communicate and exchange data.

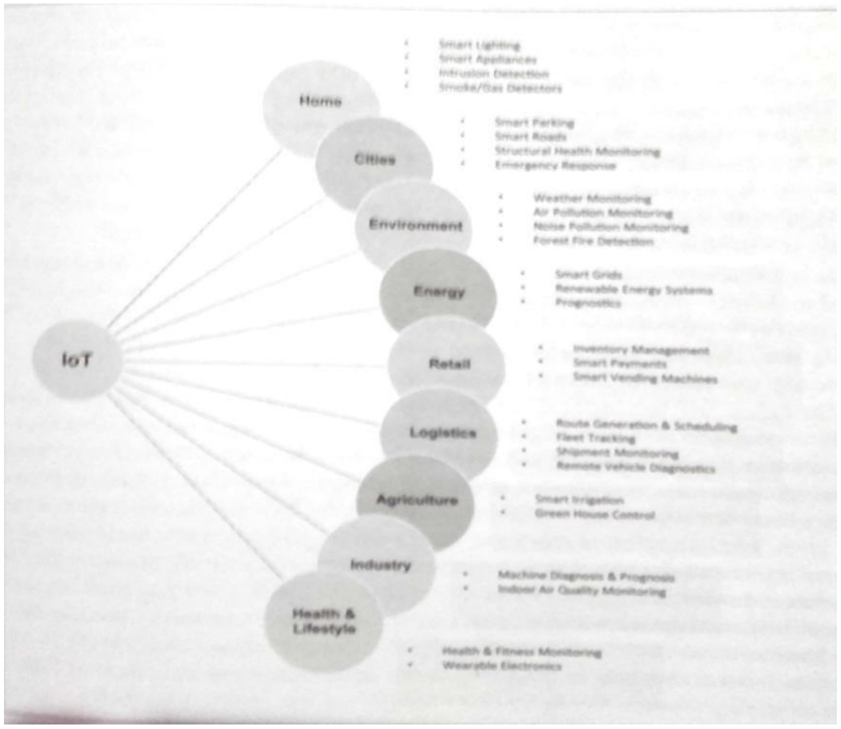
Definition:
A dynamic global n/w infrastructure with self configuring capabilities based on standard and interoperable communication protocols where physical and virtual ―things‖ have identities, physical attributes and virtual personalities and use intelligent interfaces, and are seamlessly integrated into information n/w, often communicate data associated with users and their environments.

Characteristics:

1) **Dynamic & Self Adapting**: IoT devices and systems may have the capability to dynamically adapt with the changing contexts and take actions based on their operating conditions, user‗s context or sensed environment.
   Eg: the surveillance system is adapting itself based on context and changing conditions.
2) **Self Configuring:** allowing a large number of devices to work together to provide certain functionality.
3) **Inter Operable Communication Protocols:** support a number of interoperable communication protocols and can communicate with other devices and also with infrastructure.
4) **Unique Identity:** Each IoT device has a unique identity and a unique identifier (IP address).
5) **Integrated into Information Network:** that allow them to communicate and exchange data with other devices and systems.

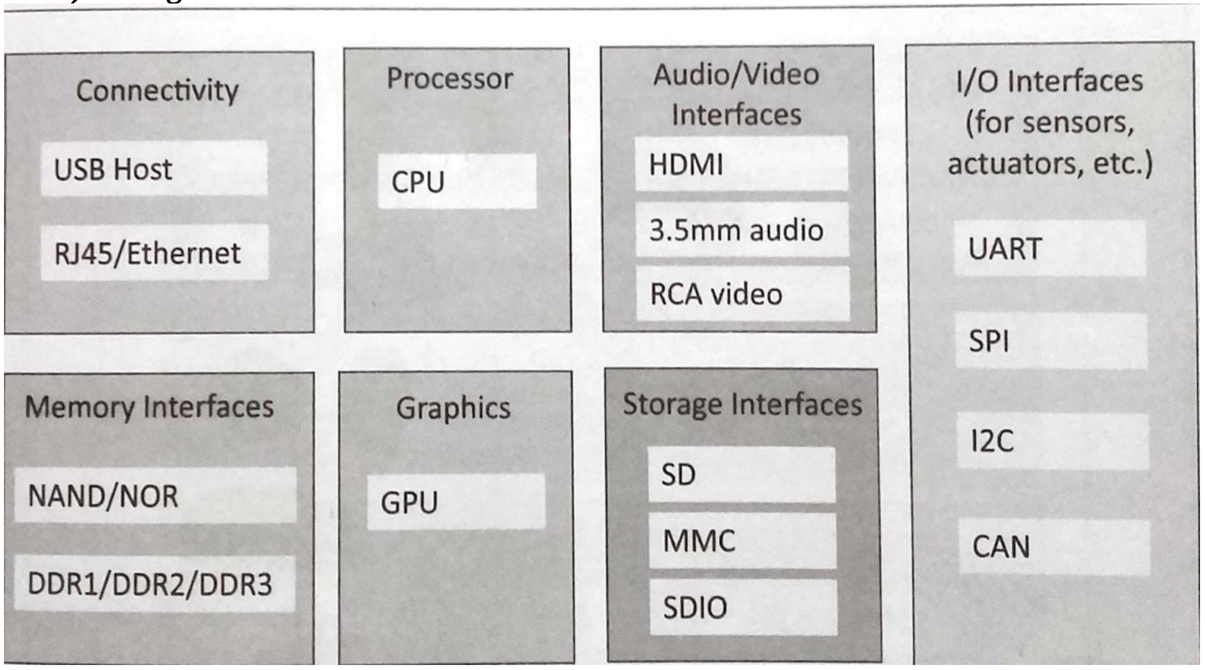
Applications of IoT:

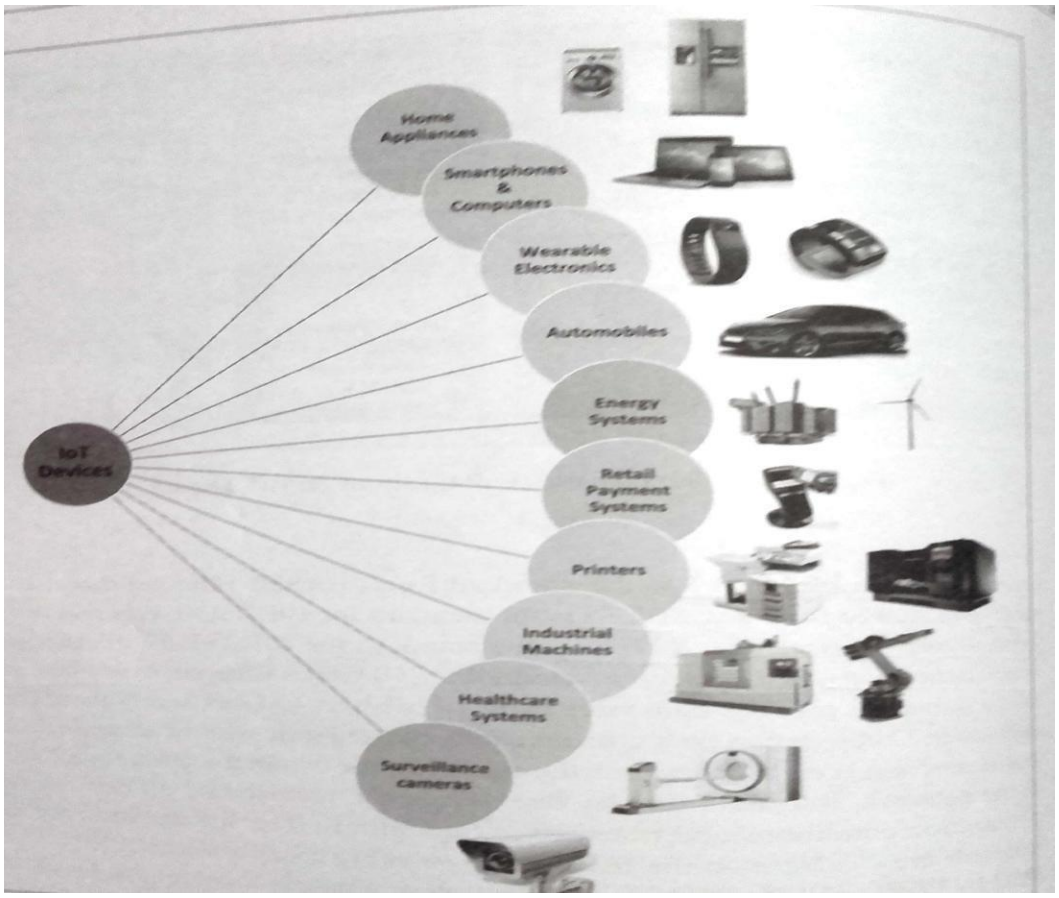
1) Home
2) Cities
3) Environment
4) Energy
5) Retail
6) Logistics
7) Agriculture
8) Industry
9) Health & Life Style



Physical Design of IoT

1) **Things in IoT:**



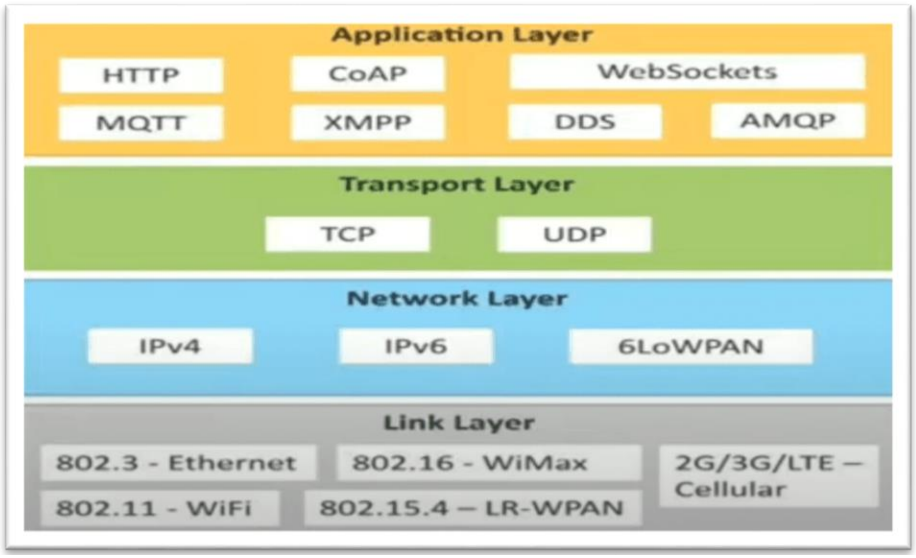


The things in IoT refer to IoT devices which have unique identities and perform remote sensing, actuating and monitoring capabilities. IoT devices can exchange data with other connected devices applications. It collects data from other devices and process data either locally or remotely.

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

(i) I/O interfaces for sensors,
(ii) Interfaces for internet connectivity
(iii) Memory and storage interfaces
(iv) Audio/video interfaces.

2. IoT Protocols:



i) **Link Layer:** Protocols determine how data is physically sent over the network_s physical layer or medium. Local network connect to which host is attached. Hosts on the same link exchange data packets over the link layer using link layer protocols. Link layer determines how packets are coded and signaled by the h/w device over the medium to which the host is attached.

Protocols:

- 802.3-Ethernet: IEEE802.3 is collection of wired Ethernet standards for the link layer. Eg: 802.3 uses co-axial cable; 802.3i uses copper twisted pair connection; 802.3j uses fiber optic connection; 802.3ae uses Ethernet over fiber.
- 802.11-WiFi: IEEE802.11 is a collection of wireless LAN(WLAN) communication standards including extensive description of link layer. Eg: 802.11a operates in 5GHz band, 802.11b and 802.11g operates in 2.4GHz band, 802.11n operates in 2.4/5GHz band, 802.11ac operates in 5GHz band, 802.11ad operates in 60Ghzband.
- 802.16 - WiMax: IEEE802.16 is a collection of wireless broadband standards including exclusive description of link layer. WiMax provide data rates from 1.5 Mb/s to 1Gb/s.
- 802.15.4-LR-WPAN: IEEE802.15.4 is a collection of standards for low rate wireless personal area network(LR-WPAN). Basis for high level communication protocols such as ZigBee. Provides data rate from 40kb/s to250kb/s.
- 2G/3G/4G-Mobile Communication: Data rates from 9.6kb/s(2G) to up to100Mb/s(4G).

(ii) **Network/Internet Layer:** Responsible for sending IP datagram's from source n/w to destination n/w. performs the host addressing and packet routing. Datagram's contains source and destination address.

Protocols:

a. **IPv4:** Internet Protocol version4 is used to identify the devices on a n/w using a hierarchical addressing scheme. 32 bit address. Allows total of 2**32addresses.
b. **IPv6:** Internet Protocol version6 uses 128 bit address scheme and allows 2**128 addresses.
c. **6LOWPAN:**(IPv6overLowpowerWirelessPersonalAreaNetwork)operates in 2.4 GHz frequency range and data transfer 250 kb/s.

(iii) Transport Layer: Provides end-to-end message transfer capability independent of the underlying n/w. Set up on connection with ACK as in TCP and without ACK as in UDP. Provides functions such as error control, segmentation, flow control and congestion control.
Protocols:

- **TCP:** Transmission Control Protocol used by web browsers(along with HTTP and HTTPS), email(along with SMTP, FTP). Connection oriented and stateless protocol. IP Protocol deals with sending packets, TCP ensures reliable transmission of protocols in order. Avoids n/w congestion and congestion collapse.
- **UDP:** User Datagram Protocol is connectionless protocol. Useful in time sensitive applications, very small data units to exchange. Transaction oriented and stateless protocol. Does not provide guaranteed delivery.

(iv) **Application Layer:** Defines how the applications interface with lower layer protocols to send data over the n/w. Enables process-to-process communication using ports.

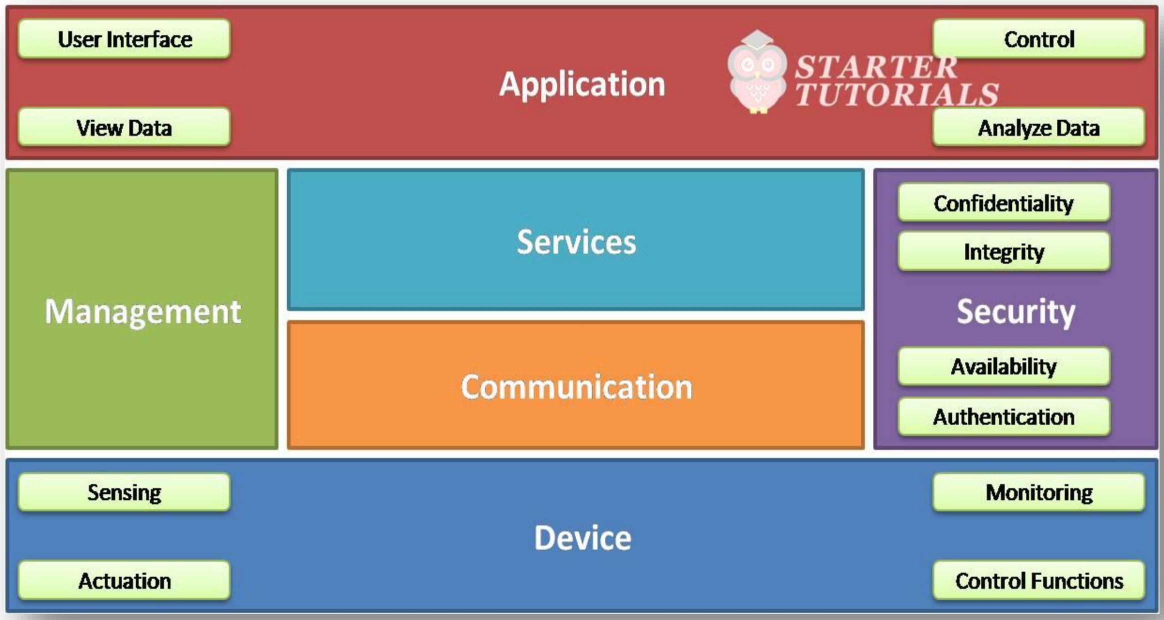
Protocols:

- **HTTP:** Hyper Text Transfer Protocol that forms foundation of WWW. Follow request-response model Stateless protocol.
- **CoAP:** Constrained Application Protocol for machine-to-machine (M2M) applications with constrained devices, constrained environment and constrained n/w. Uses client-server architecture.
- **WebSocket:** allows full duplex communication over a single socket connection.
- **MQTT:** Message Queue Telemetry Transport is light weight messaging protocol based on publish-subscribe model. Uses client server architecture. Well suited for constrained environment.
- **XMPP:** Extensible Message and Presence Protocol for real time communication and streaming XML data between network entities. Support client-server and server-server communication.
- **DDS:** Data Distribution Service is data centric middleware standards for device-to-device or machine-to-machine communication. Uses publish-subscribe model.
- **AMQP:** Advanced Message Queuing Protocol is open application layer protocol for business messaging. Supports both point-to-point and publish-subscribe model.

LOGICAL DESIGN of IoT

Refers to an abstract represent of entities and processes without going into the low level specifies of implementation.
1) IoT Functional Blocks
2) IoT Communication Models 3)
IoT Communication APIs

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

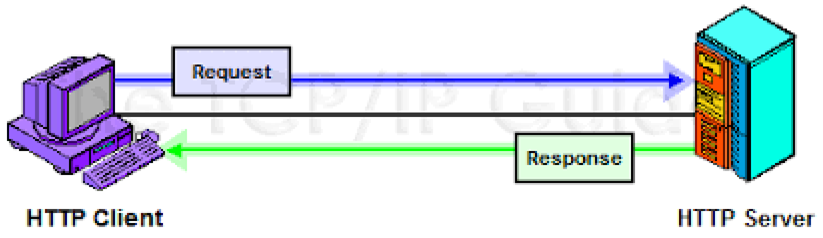


- **Device:** An IoT system comprises of devices that provide sensing, actuation, and monitoringand control functions.
- **Communication:** handles the communication for IoT system.
- **Services:** for device monitoring, device control services, data publishing services and services for device discovery.
- **Management:** Provides various functions to govern the IoT system.
- **Security:** Secures IoT system and priority functions such as authentication ,authorization, message and context integrity and data security.
- **Application:** IoT application provides an interface that the users can use to control andmonitor various aspects of IoT system.

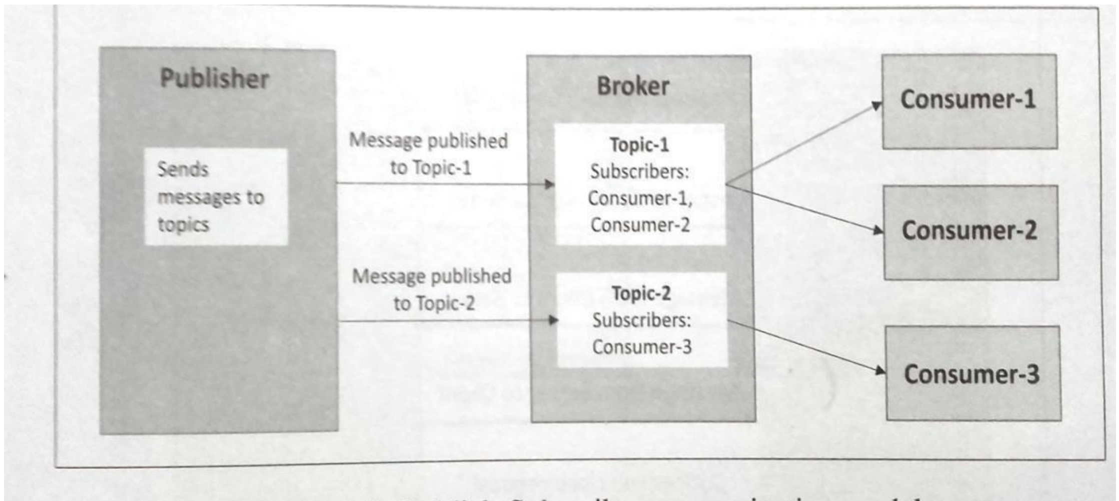
1) IoT Communication Models:

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

1) Request-Response Model

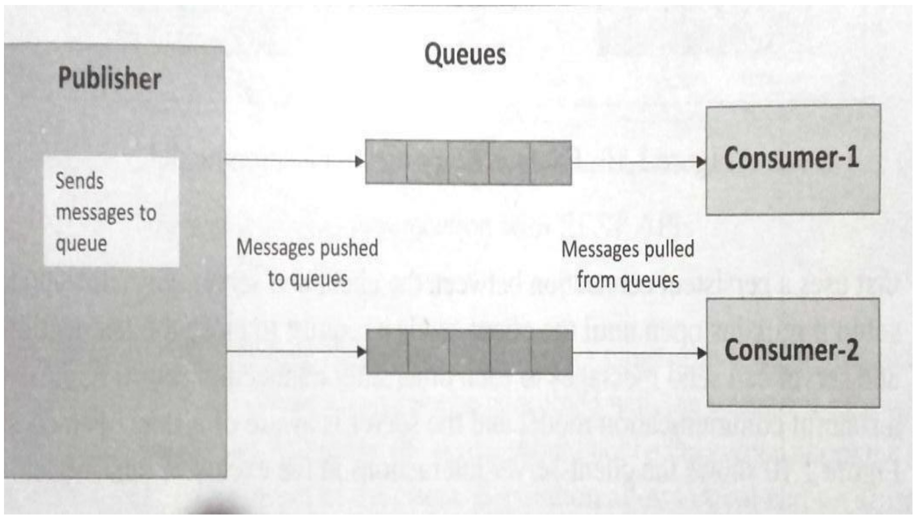


2) **Publish-Subscibe Model:**

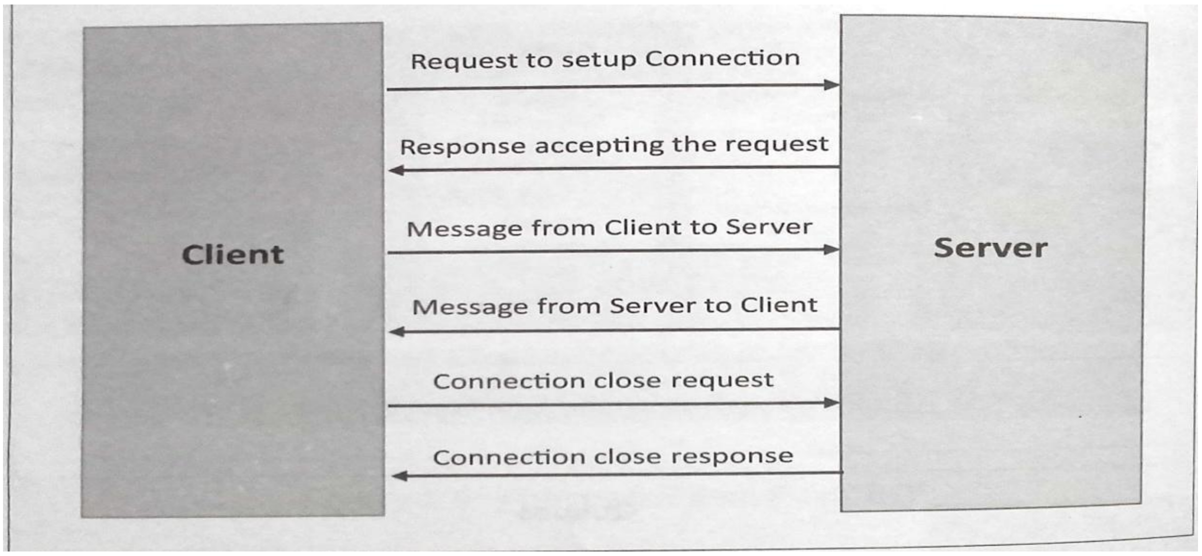


Involves publishers, brokers and consumers. Publishers are source of data. Publishers send data to the topics which are managed by the broker. Publishers are not aware of the consumers. Consumers subscribe to the topics which are managed by the broker. When the broker receives data for a topic from the publisher, it sends the data to all the subscribed consumers.

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



4) **Exclusive Pair:** is bi-directional, fully duplex communication model that uses a persistent connection between the client and server. Once connection is set up it remains open until the client send a request to close the connection. Is a stateful communication model and server is aware of all the open connections.


Client
Server
Request to setup Connection
Response accepting the request
Message from Client to Server
Message from Server to Client
Connection close request
Connection close response

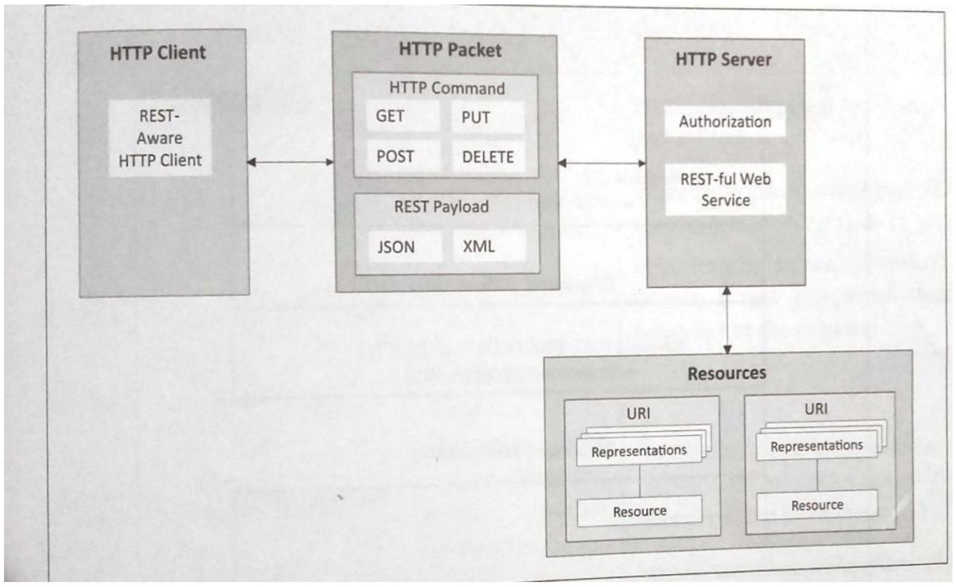
3) **IoT Communication APIs:**

a) REST based communication APIs(Request-Response Based Model)

b) WebSocket based Communication APIs(Exclusive PairBased Model)

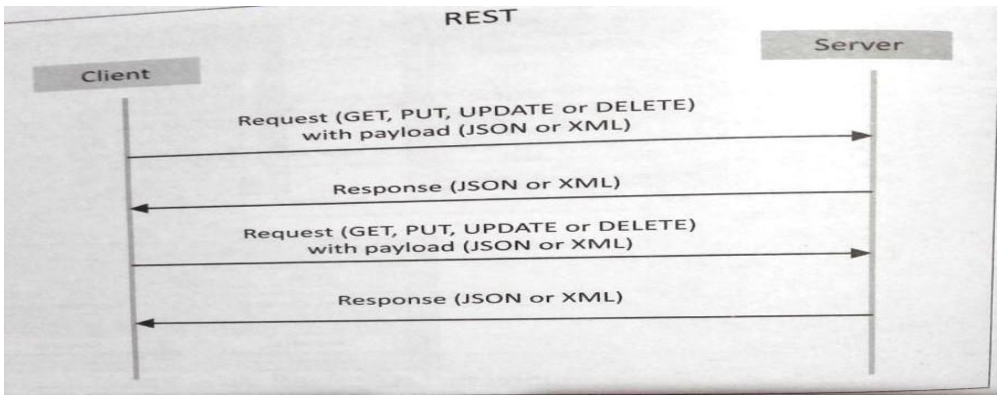
a) REST based communication APIs: Representational State Transfer(REST) is a set of architectural principles by which we can design web services and web APIs that focus on a system_s resources and have resource states are addressed and transferred.

The REST architectural constraints: Fig. shows communication between client server with REST APIs.



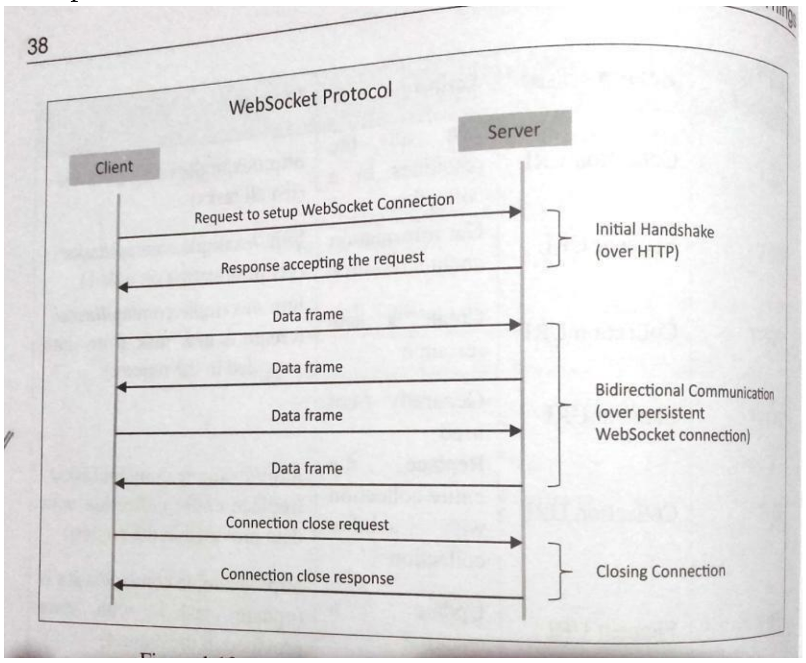
Client-Server: The principle behind client-server constraint is the separation of concerns. Separation allows client and server to be independently developed and updated.
Stateless: Each request from client to server must contain all the info. Necessary to understand the request, and cannot take advantage of any stored context on the server.
Cache-able: Cache constraint requires that the data within a response to a request be implicitly or explicitly labeled as cache-able or non-cacheable. If a response is cache-able, then a client cache is given the right to reuse that response data for later, equivalent requests.
Layered System: constraints the behavior of components such that each component cannot see beyond the immediate layer with which they are interacting.
User Interface: constraint requires that the method of communication between a client and a server must be uniform.
Code on Demand: Servers can provide executable code or scripts for clients to execute in their context. This constraint is the only one that is optional.

Request-Response model used by REST:


REST
Client
Server
Request (GET, PUT, UPDATE or DELETE) with payload (JSON or XML)
Response (JSON or XML)
Request (GET, PUT, UPDATE or DELETE) with payload (JSON or XML)
Response (JSON or XML)

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

b) WebSocket Based Communication APIs: WebSocket APIs allow bi-directional, full duplex communication between clients and servers. WebSocket APIs follow the exclusive pair communication model.


38
WebSocket Protocol
Client
Server
Request to setup WebSocket Connection
Response accepting the request
Initial Handshake (over HTTP)
Data frame
Data frame
Data frame
Data frame
Bidirectional Communication (over persistent WebSocket connection)
Connection close request
Connection close response
Closing Connection

IoT Enabling Technologies

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

1) **Wireless Sensor Network(WSN):** Comprises of distributed devices with sensors which are used to monitor the environmental and physical conditions. Zig Bee is one of the most popular wireless technologies used byWSNs.
WSNs used in IoT systems are described as follows:
   - Weather Monitoring System: in which nodes collect temp, humidity and other data, which is aggregated and analyzed.
   - Indoor air quality monitoring systems: to collect data on the indoor air quality and concentration of various gases.
   - Soil Moisture Monitoring Systems: to monitor soil moisture at variouslocations.
   - Surveillance Systems: use WSNs for collecting surveillance data(motiondata detection).
   - Smart Grids : use WSNs for monitoring grids at variouspoints.

- Structural Health Monitoring Systems: Use WSNs to monitor the health of structures(building, bridges) by collecting vibrations from sensor nodes deployed at various points in the structure.

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

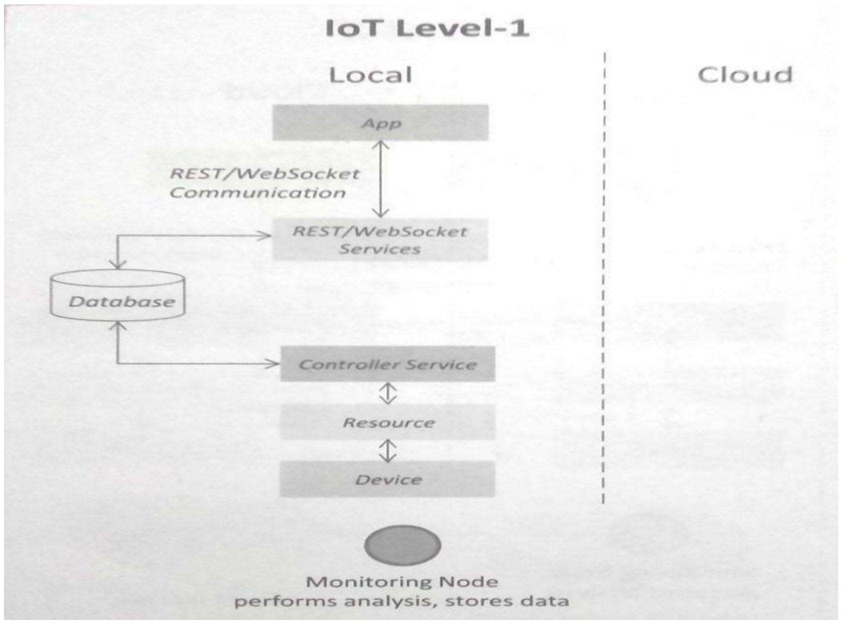
3) **Big Data Analytics:** Some examples of big data generated by IoT are
   - Sensor data generated by IoT systems.
   - Machine sensor data collected from sensors established in industrial and energy systems.
   - Health and fitness data generated IoT devices.
   - Data generated by IoT systems for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

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

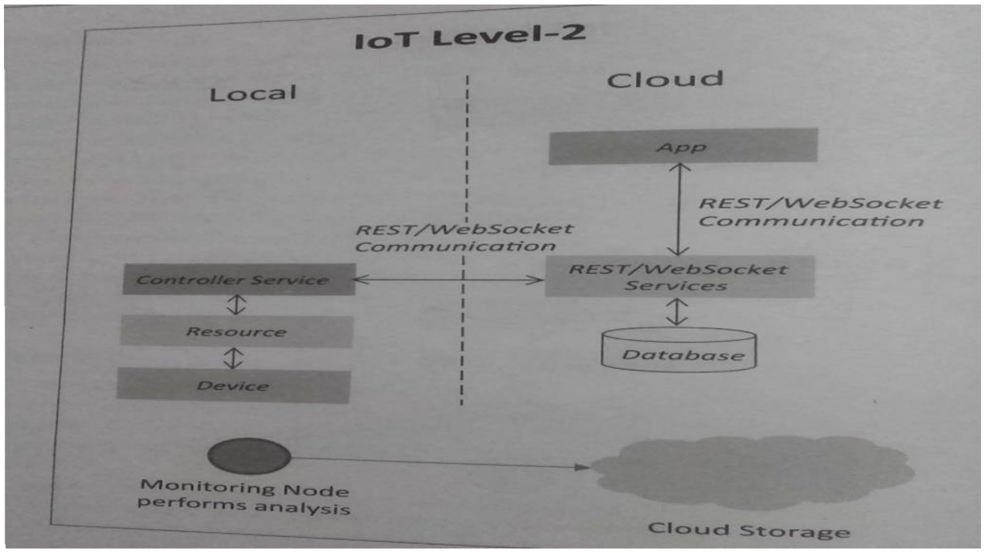
5) **Embedded Systems:** is a computer system that has computer hardware and software embedded to perform specific tasks. Embedded System range from low cost miniaturized devices such as digital watches to devices such as digital cameras, POS terminals, vending machines, appliances etc.,

IoT Levels and Deployment Templates

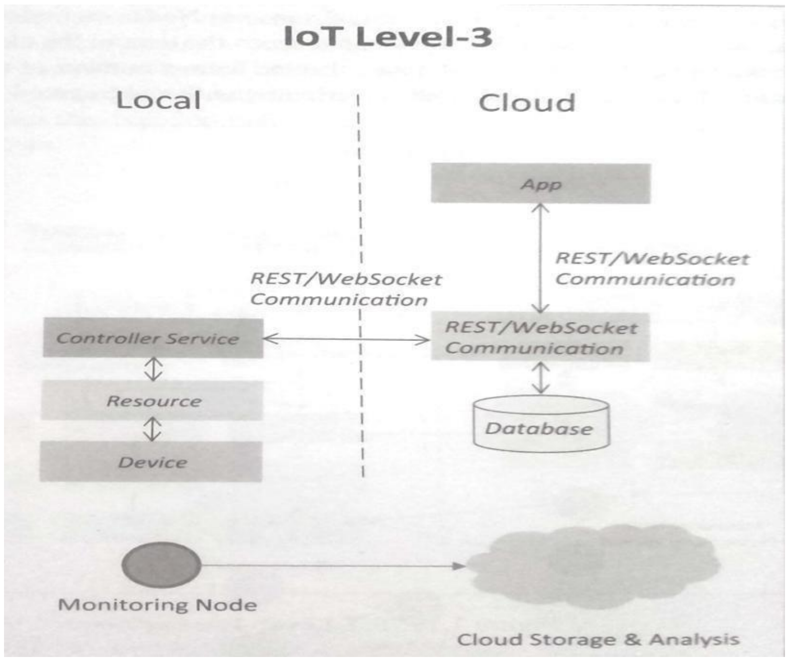
1) **IoT Level1:** System has a single node that performs sensing and/or actuation, stores data, performs analysis and host the application as shown in fig. Suitable for modeling low cost and low complexity solutions where the data involved is not big and analysis requirement are not computationally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



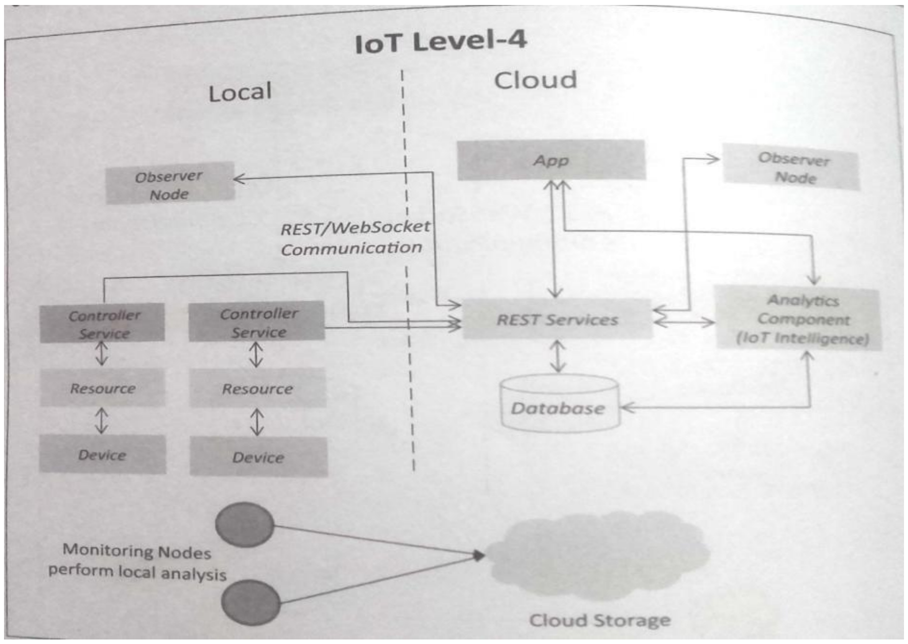
2) **IoT Level2:** has a single node that performs sensing and/or actuating and local analysis as shown in fig. Data is stored in cloud and application is usually cloud based. Level2 IoT systems are suitable for solutions where data are involved is big, however, the primary analysis requirement is not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



3) **IoT Level3:** system has a single node. Data is stored and analyzed in the cloud application is cloud based as shown in fig. Level3 IoT systems are suitable for solutions where the data involved is big and analysis requirements are computationally intensive. An example of IoT level3 system for tracking package handling.

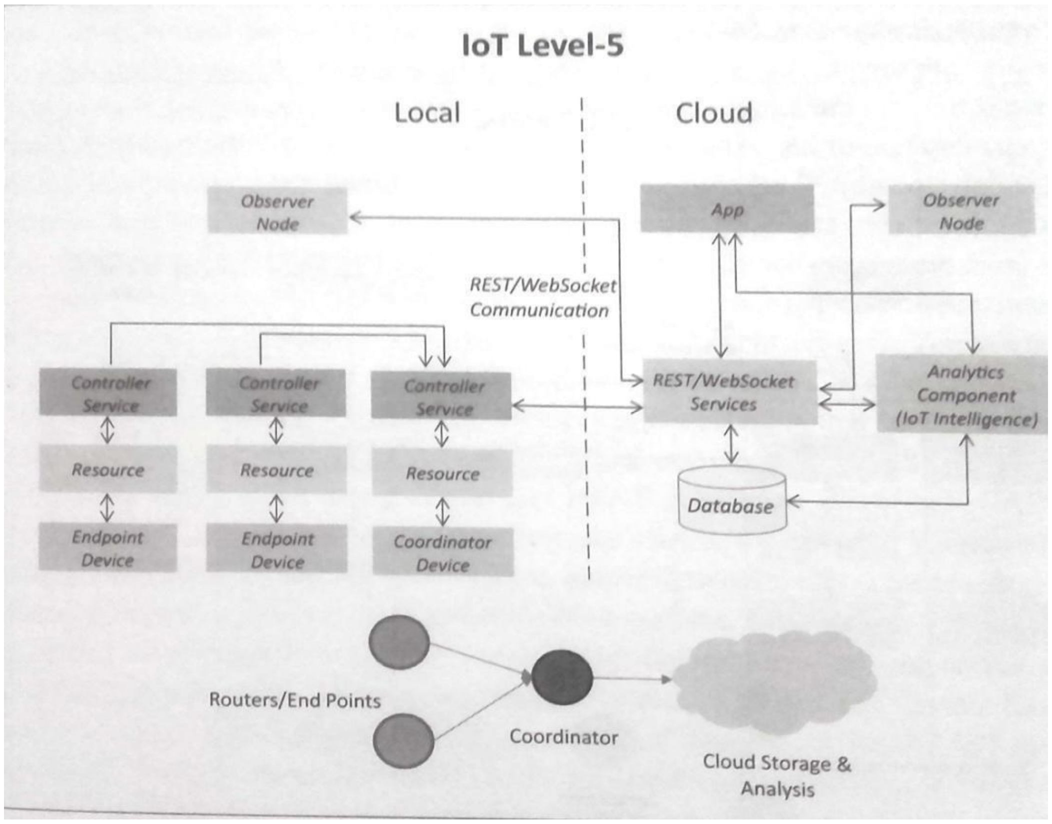


4) **IoT Level4:** System has multiple nodes that perform local analysis. Data is stored in the cloud and application is cloud based as shown in fig. Level4 contains local and cloud based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices. An example of a Level4 IoT system for Noise Monitoring.

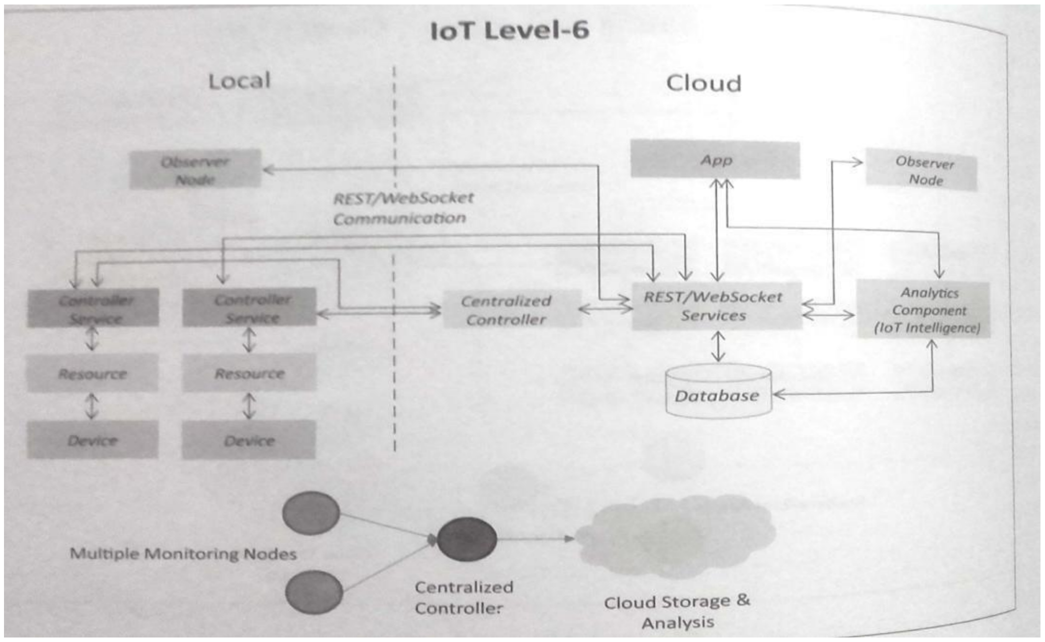


5) **IoT Level5:** System has multiple end nodes and one coordinator node as shown in fig. The end nodes that perform sensing and/or actuation. Coordinator node collects data from theendnodesandsendstothecloud.Dataisstoredandanalyzedinthecloudand

application is cloud based. Level5 IoT systems are suitable for solution based on wireless sensor network, in which data involved is big and analysis requirements are computationally intensive. An example of Level5 system for Forest Fire Detection.



6) **IoT Level6:** System has multiple independent end nodes that perform sensing and/or actuation and sensed data to the cloud. Data is stored in the cloud and application is cloud based as shown in fig. The analytics component analyses the data and stores the result in the cloud data base. The results are visualized with cloud based application. The centralized controller is aware of the status of all the end nodes and sends control commands to nodes. An example of a Level6 IoT system for Weather Monitoring System.



DOMAIN SPECIFIC IoTs

1) **Home Automation:**
   a) **Smart Lighting:** helps in saving energy by adapting the lighting to the ambient conditions and switching on/off or diming the light when needed.
   b) **Smart Appliances:** make the management easier and also provide status information to the users remotely.
   c) **Intrusion Detection:** use security cameras and sensors(PIR sensors and door sensors) to detect intrusion and raise alerts. Alerts can be in the form of SMS or email sent to the user.
   d) **Smoke/Gas Detectors:** Smoke detectors are installed in homes and buildings to detect smoke that is typically an early sign of fire. Alerts raised by smoke detectors can be in the form of signals to a fire alarm system. Gas detectors can detect the presence of harmful gases such as CO, LPGetc.,

2) **Cities:**
   a) **Smart Parking:** make the search for parking space easier and convenient for drivers. Smart parking are powered by IoT systems that detect the no. of empty parking slots and send information over internet to smart application backends.
   b) **Smart Lighting:** for roads, parks and buildings can help in saving energy.
   c) **Smart Roads:** Equipped with sensors can provide information on driving condition, travel time estimating and alert in case of poor driving conditions, traffic condition and accidents.
   d) **Structural Health Monitoring:** uses a network of sensors to monitor the vibration levels in the structures such as bridges and buildings.
   e) **Surveillance:** The video feeds from surveillance cameras can be aggregated in cloud based scalable storage solution.

f) **Emergency Response:** IoT systems for fire detection, gas and water leakage detection can help in generating alerts and minimizing their effects on the critical infrastructures.

3) Environment:

a) **Weather Monitoring:** Systems collect data from a no. of sensors attached and send the data to cloud based applications and storage back ends. The data collected in cloud can then be analyzed and visualized by cloud based applications.

b) **Air Pollution Monitoring:** System can monitor emission of harmful gases(CO2, CO, NO, NO2 etc.,) by factories and automobiles using gaseous and meteorological sensors. The collected data can be analyzed to make informed decisions on pollutions control approaches.

c) **Noise Pollution Monitoring:** Due to growing urban development, noise levels in cities have increased and even become alarmingly high in some cities. IoT based noise pollution monitoring systems use a no. of noise monitoring systems that are deployed at different places in a city. The data on noise levels from the station is collected on servers or in the cloud. The collected data is then aggregated to generate noise maps.

d) **Forest Fire Detection:** Forest fire can cause damage to natural resources, property and human life. Early detection of forest fire can help in minimizing damage.

e) **River Flood Detection:** River floods can cause damage to natural and human resources and human life. Early warnings of floods can be given by monitoring the water level and flow rate. IoT based river flood monitoring system uses a no. of sensor nodes that monitor the water level and flow rate sensors.

4) Energy:

a) **Smart Grids:** is a data communication network integrated with the electrical grids that collects and analyze data captured in near-real-time about power transmission, distribution and consumption. Smart grid technology provides predictive information and recommendations to utilities, their suppliers, and their customers on how best to manage power. By using IoT based sensing and measurement technologies, the health of equipment and integrity of the grid can be evaluated.

b) **Renewable Energy Systems:** IoT based systems integrated with the transformers at the point of interconnection measure the electrical variables and how much power is fed into the grid. For wind energy systems, closed-loop controls can be used to regulate the voltage at point of interconnection which coordinate wind turbine outputs and provides power support.

c) **Prognostics:** In systems such as power grids, real-time information is collected using specialized electrical sensors called Phasor Measurment Units(PMUs) at the substations. The information received from PMUs must be monitored in real-time for estimating the state of the system and for predicting failures.

5) Retail:

a) **Inventory Management:** IoT systems enable remote monitoring of inventory using data collected by RFIDreaders.

b) **Smart Payments:** Solutions such as contact-less payments powered by technologies such as Near Field Communication (NFC) and Bluetooth.
c) **Smart Vending Machines:** Sensors in a smart vending machines monitors its operations and send the data to cloud which can be used for predictive maintenance.

6) **Logistics:**
   a) **Route generation & scheduling:** IoT based system backed by cloud can provide first response to the route generation queries and can be scaled upto serve a large transportation network.
   b) **Fleet Tracking:** Use GPS to track locations of vehicles inreal-time.
   c) **Shipment Monitoring:** IoT based shipment monitoring systems use sensors such as temp, humidity, to monitor the conditions and send data to cloud, where it can be analyzed to detect foods poilage.
   d) **Remote Vehicle Diagnostics:** Systems use on-board IoT devices for collecting data on Vehicle operations(speed, RPMetc.,) and status of various vehicle subsystems.

7) **Agriculture:**
   a) **Smart Irrigation:** to determine moisture amount in soil.
   b) **Green House Control:** to improve productivity.

8) **Industry:**
   a) Machine diagnosis and prognosis
   b) Indoor Air Quality Monitoring

9) **Health and LifeStyle:**
   a) Health & Fitness Monitoring
   b) Wearable Electronics

UNIT-II

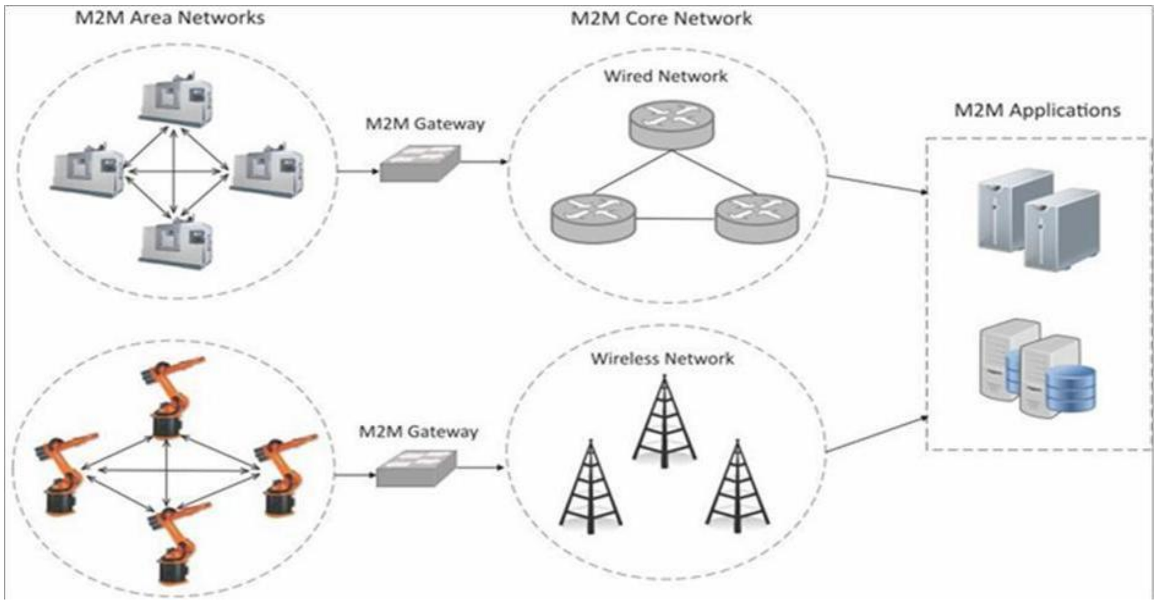
IoT and M2M

M2M:

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

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

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



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

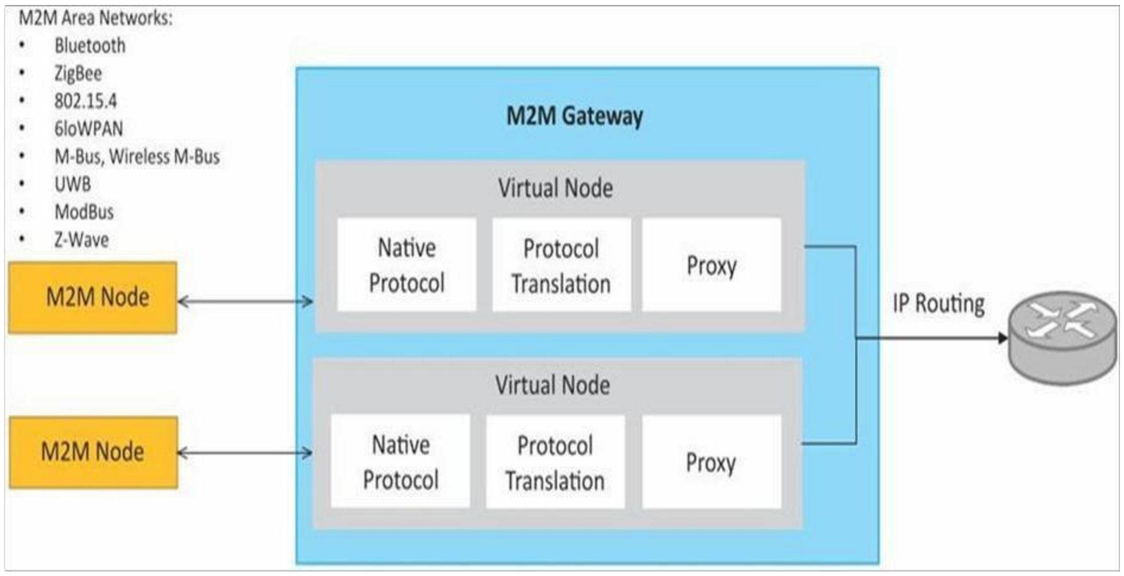
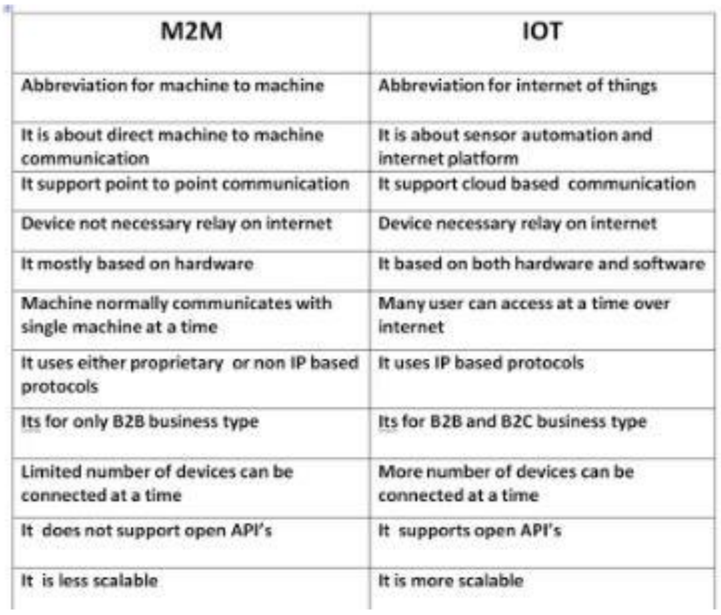


Fig. Shows a block diagram of an M2M gateway. The communication between M2M nodes and the M2M gateway is based on the communication protocols which are naive to the M2M are network. M2M gateway performs protocol translations to enable Ip-connectivity for M2M are networks. M2M gateway acts as a proxy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

Differences between IoT and M2M



M2M	IOT
Abbreviation for machine to machine	Abbreviation for internet of things
It is about direct machine to machine communication	It is about sensor automation and internet platform
It support point to point communication	It support cloud based communication
Device not necessary relay on internet	Device necessary relay on internet
It mostly based on hardware	It based on both hardware and software
Machine normally communicates with single machine at a time	Many user can access at a time over internet
It uses either proprietary or non IP based protocols	It uses IP based protocols
Its for only B2B business type	Its for B2B and B2C business type
Limited number of devices can be connected at a time	More number of devices can be connected at a time
It does not support open API's	It supports open API's
It is less scalable	It is more scalable

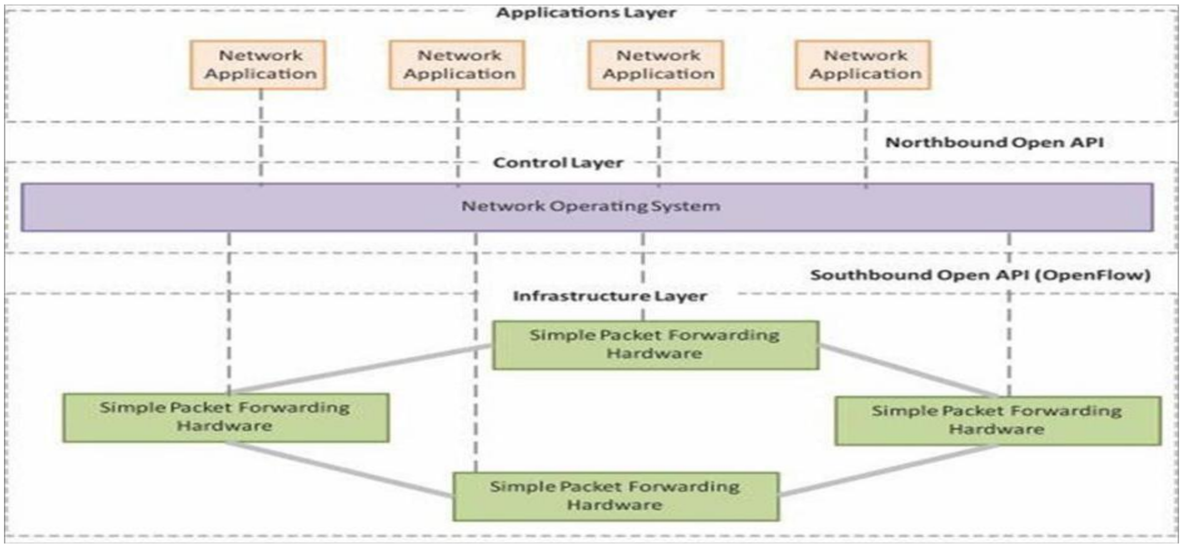
.

SDN and NVF for IoT

Software Defined Networking(SDN):

- Software-Defined Networking (SDN) is a networking architecture that separates the control plane from the data plane and centralizes the network controller.
- Software-based SDN controllers maintain a united view of the network
- The underlying infrastructure in SDN uses simple packet forwarding hardware as opposed to specialized hardware in conventional networks.

SDN Architecture



Key elements of SDN:

1) **Centralized Network Controller**

With decoupled control and data planes and centralized network controller, the network administrators can rapidly configure the network.

2) **Programmable Open APIs**

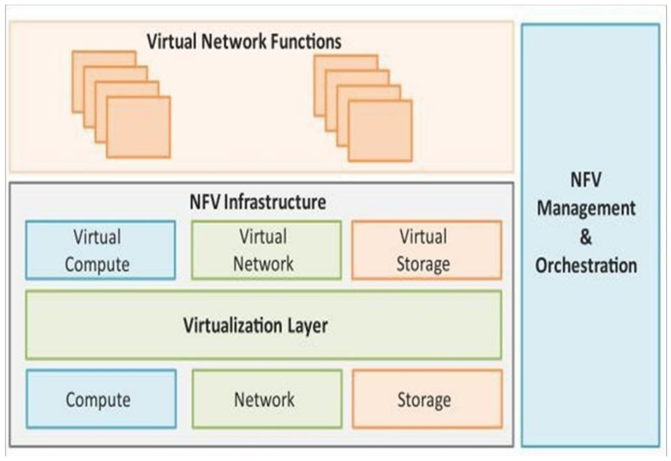
SDN architecture supports programmable open APIs for interface between the SDN application and control layers (Northbound interface).

3) **Standard Communication Interface(OpenFlow)**

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

Network Function Virtualization(NFV)

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



NFV Architecture

Key elements of NFV:

1) **Virtualized Network Function(VNF):**

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

2) **NFV Infrastructure(NFVI):**

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

3) **NFV Management and Orchestration:**

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

Need for IoT Systems Management

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

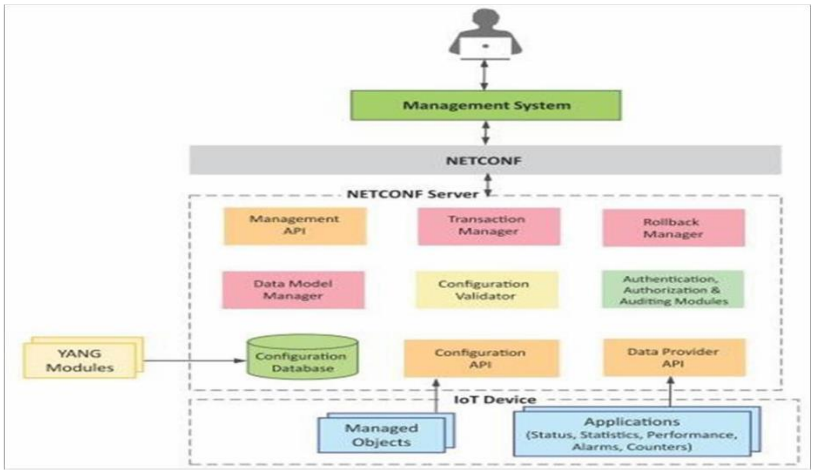
1) **Automating Configuration :** IoT system management capabilities can helpin automating the system configuration.
2) **Monitoring Operational & Statistical Data :** Management systems can help in monitoring opeartional and statistical data of a system. This data can be used for fault diagnosis or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

IoT Systems Management with NETCONF-YANG

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

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

1) Management System
2) Management API
3) Transaction Manager
4) Rollback Manager
5) Data Model Manager
6) Configuration Validator
7) Configuration Database
8) Configuration API
9) Data Provider API

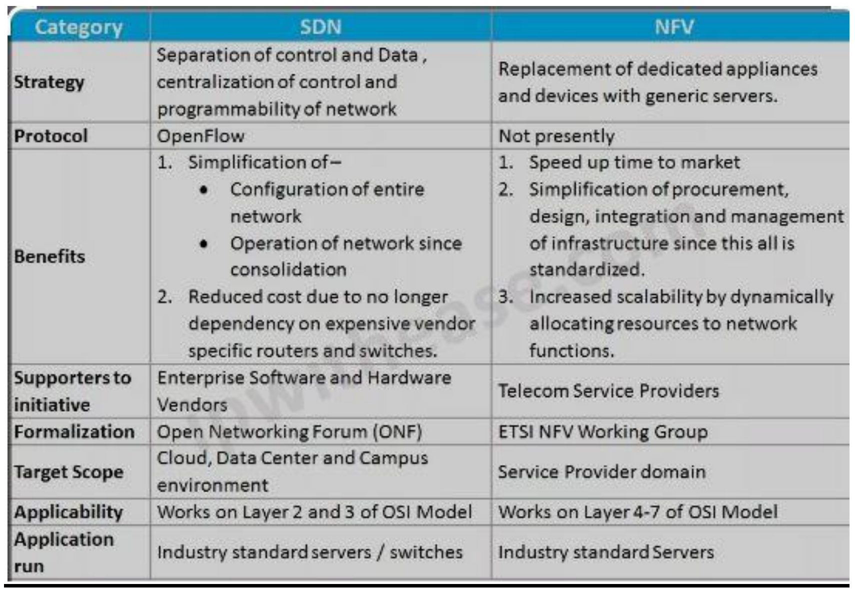


1) **Management System :** The operator uses a management system to send NETCONF messages to configure the IoT device and receives state information and notifications from the device as NETCONF messages.
2) **Management API :** allows management application to start NETCONF sessions.
3) **Transaction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

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 withLibnetconf.
3) Fill in the IoT device management code in the TransAPImodule.
4) Build the callbacks C file to generate the libraryfile.
5) Load the YANG module and the TransAPImodule into the Netopeer server using Netopeer manager tool.
6) The operator can now connect from the management system to the Netopeer server using the NetopeerCLI.
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 IoTdevice.

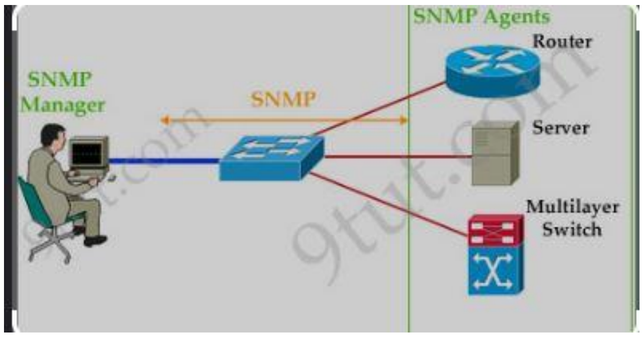
SDN vs NFV



Category	SDN	NFV
Strategy	Separation of control and Data , centralization of control and programmability of network	Replacement of dedicated appliances and devices with generic servers.
Protocol	OpenFlow	Not presently
Benefits	1. Simplification of– • Configuration of entire network • Operation of network since consolidation 2. Reduced cost due to no longer dependency on expensive vendor specific routers and switches.	1. Speed up time to market 2. Simplification of procurement, design, integration and management of infrastructure since this all is standardized. 3. Increased scalability by dynamically allocating resources to network functions.
Supporters to initiative	Enterprise Software and Hardware Vendors	Telecom Service Providers
Formalization	Open Networking Forum (ONF)	ETSI NFV Working Group
Target Scope	Cloud, Data Center and Campus environment	Service Provider domain
Applicability	Works on Layer 2 and 3 of OSI Model	Works on Layer 4-7 of OSI Model
Application run	Industry standard servers / switches	Industry standard Servers

SNMP PROTOCOL

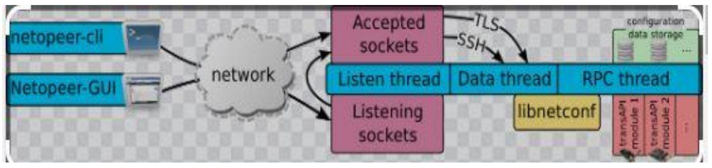
SNMP provides a common mechanism for network devices to relay management information within single and multi-vendor LAN or WAN environments. It is an application layer protocol in the OSI model framework. Typically, the SNMP protocol is implemented using the User Datagram Protocol (UDP).



NETOPEER

Netopeer GUI

The Apache module with a web-based GUI allowing user to connect to a NETCONF-enabled device and to obtain and manipulate its configuration data from a graphical interface.



UNIT-III

PYTHON

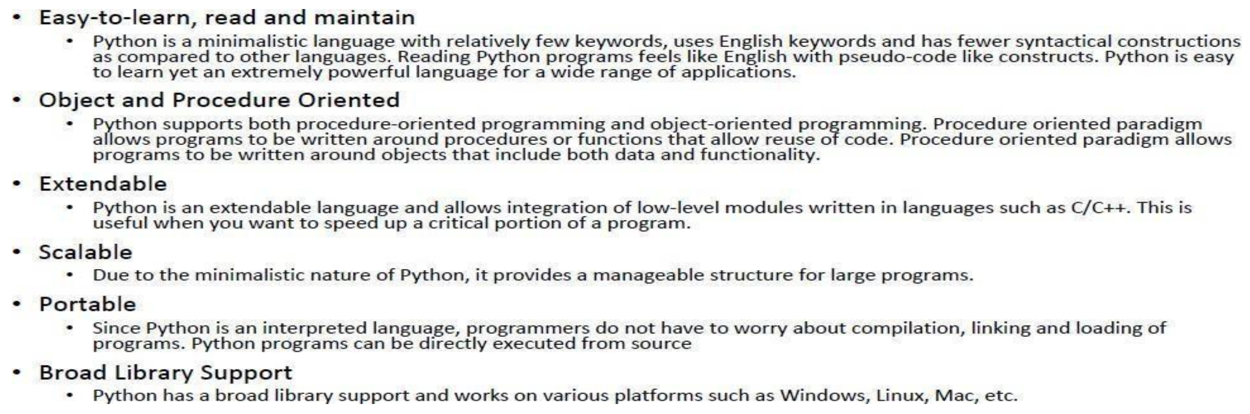
Python

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

The main characteristics of Python are:

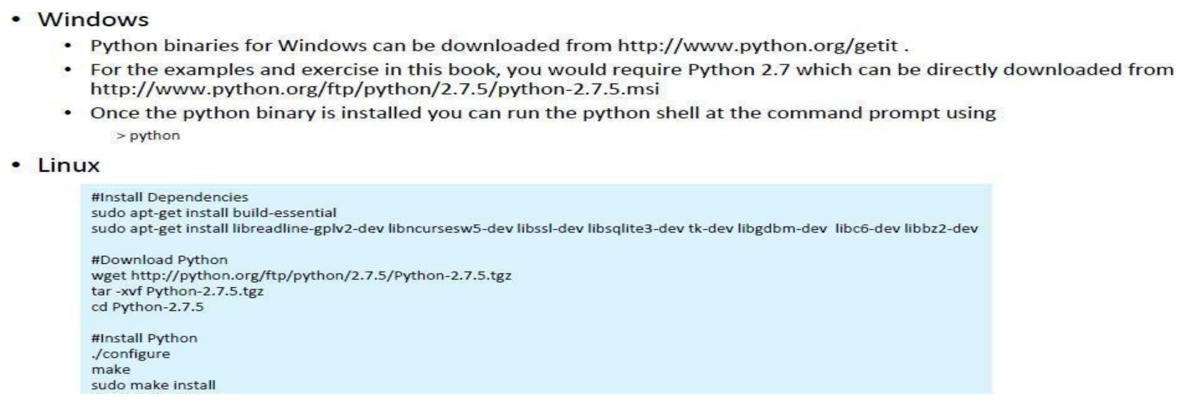
1) Multi-paradigm programminglanguage.
2) Python supports more than one programming paradigms including object- oriented programming and structured programming.
3) InterpretedLanguage.
4) Python is an interpreted language and does not require an explicit compilationstep.
5) The Python interpreter executes 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 interpreterdirectly.

Python Benefits



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

Python - Setup



- Windows
  - Python binaries for Windows can be downloaded from http://www.python.org/getit .
  - For the examples and exercise in this book, you would require Python 2.7 which can be directly downloaded from http://www.python.org/ftp/python/2.7.5/python-2.7.5.msi
  - Once the python binary is installed you can run the python shell at the command prompt using

```
> python
```

- Linux

```
#Install Dependencies
sudo apt-get install build-essential
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev  libc6-dev libbz2-dev

#Download Python
wget http://python.org/ftp/python/2.7.5/Python-2.7.5.tgz
tar -xvf Python-2.7.5.tgz
cd Python-2.7.5

#Install Python
./configure
make
sudo make install
```

Datatypes

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

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

Python Numbers

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

Script.py

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

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

```
>>> a = 1234567890123456789

>>> a

1234567890123456789

>>> b = 0.1234567890123456789

>>> b

0.12345678901234568
```

```
>>> c = 1+2j
>>> c
(1+2j)
```

Python List

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

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

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

Script.py

```
a = [5,10,15,20,25,30,35,40]
# a[2] = 15
print("a[2] = ", a[2])
# a[0:3] = [5, 10, 15]
print("a[0:3] = ", a[0:3])
# a[5:] = [30, 35, 40]
print("a[5:] = ", a[5:])
```

Lists are mutable, meaning; value of elements of a list can be altered.

```
>>> a = [1,2,3]
>>> a[2]=4
>>> a
[1, 2, 4]
```

Python Tuple

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

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

Script.py

```
t = (5,'program', 1+3j)
# t[1] = 'program'
print("t[1] = ", t[1])
# t[0:3] = (5, 'program', (1+3j))
print("t[0:3] = ", t[0:3])
# Generates error
# Tuples are immutable
t[0] = 10
```

Python Strings

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

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

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

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

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

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

>>> type(d)

<class 'dict'>
```

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

Script.py

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

Python if...else Statement

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

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

Python if Statement

Syntax

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

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

Python if Statement Flowchart

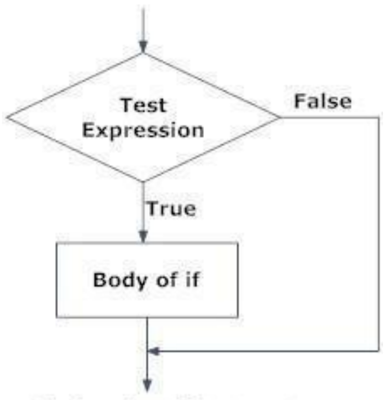


Fig: Operation of if statement

Example: Python if Statement

```
# If the number is positive, we print an appropriate message

num = 3
if num > 0:
    print(num, "is a positive number.")
print("This is always printed.")
num = -1
if num >0:
    print(num, "is a positive number.")
print("This is also always printed.")
```

When you run the program, the output willbe:

```
3 is a positivenumber
This is alwaysprinted
This is also always printed.
```

In the above example, num > 0 is the test expression. The body of if is executed only if this evaluates to True.

When variable num is equal to 3, test expression is true and body inside body of if is executed. If variable num is equal to -1, test expression is false and body inside body of if is skipped. The print() statement falls outside of the if block (unindented). Hence, it is executed regardless of the testexpression.

Python if...else Statement

Syntax

```
if test expression:
    Body of if
else:
    Body of else
```

The if..else statement evaluates test expression and will execute body of if only when test condition is True.

If the condition is False, body of else is executed. Indentation is used to separate the blocks

Example of if...else

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

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

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

If num is equal to 0, the test expression is true and body of if is executed and body of else is skipped.

Python if...elif...else Statement

Syntax

```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

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

Flowchart of if...elif...else

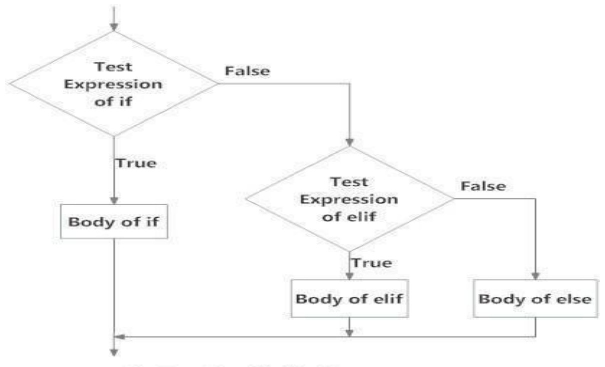


Fig: Operation of if...elif...else statement

Example of if...elif...else

```
# In this program,
# we check if the number is positive or
# negative or zero and
# display an appropriate message
num = 3.4
# Try these two variations as well:
# num = 0
# num = -4.5
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

When variable num is positive, Positive number is printed.

If num is equal to 0, Zero is printed.
If num is negative, Negative number is printed

Python Nested if statements

We can have a if...elif...else statement inside another if...elif...else statement. This is called nesting in computer programming. Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. This can get confusing, so must be avoided if we can.

Python Nested if Example

```
# In this program, we input a number
# check if the number is positive or
# negative or zero anddisplay
# an appropriate message
# This time we use nested if

num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

Output 1

```
Enter a number: 5
Positive number
Output 2
Enter a number: -1
Negative number
Output 3
Enter a number: 0
Zero
```

Python for Loop

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

Syntax of for Loop

```
for val in sequence:
        Body of for
```

Here, val is the variable that takes the value of the item inside the sequence on each iteration. Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

Flowchart of for Loop

Syntax

```
# Program to find the sum of all numbers stored in a list
# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
# variable to store the sum
sum = 0

# iterate over the list
for val in numbers:
        sum =    sum+val
# Output: The sum is 48
print("The sum is", sum)
```

when you run the program, the output will be:

```
The sum is 48
```

What is while loop in Python?

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true. We generally use this loop when we don't know beforehand, the number of times to iterate.

Syntax of while Loop in Python

```
while
   test_expression:
   Body of while
```

In while loop, test expression is checked first. The body of the loop is entered only if the test_expression evaluates to True. After one iteration, the test expression is checked again. This process continues until the test_expression evaluates to False. In Python, the body of the while loop is determined through indentation. Body starts with indentation and the first unindented line marks the end. Python interprets any non-zero value as True. None and 0 are interpreted as False.

Flowchart of while Loop

```
# Program to add
natural # numbers
upto
# sum = 1+2+3+...+n
# To take input from the
user, # n = int(input("Enter
n: "))
n = 10
# initialize sum and
counter sum = 0
i = 1
while i <= n:
   sum = sum +
   i
   i=i+1     #
updatecounter # print
thesum
print("The sum is", sum)
```

When you run the program, the output will be:

```
Enter n: 10
The sum is
55
```

In the above program, the test expression will be True as long as our counter variable i is less than or equal to n (10 in ourprogram).
We need to increase the value of counter variable in the body of the loop. This is very important (and mostly forgotten). Failing to do so will result in an infinite loop (never ending loop).
Finally the result is displayed.

Python Modules

A file containing a set of functions you want to include in the application is called Module.

Create a Module

To create a module just save the code you want in a file with the file extension .py:

Example

Save this code in a file named mymodule.py

```
def greeting(name):
  print("Hello, " + name)
```

Use a Module

Now we can use the module we just created, by using the import statement:

Example

Import the module named mymodule, and call the greeting function:

```
import mymodule
mymodule.greeting("Jonathan")
```

Note: When using a function from a module, use the syntax: module_name.function_name.

Variables in Module

The module can contain functions, as already described, but also variables of all types(arrays, dictionaries, objects etc):

Example

Save this code in the file mymodule.py

```
person1 = {"name": "John","age": 36,"country": "Norway"}
```

Example

Import the module named mymodule, and access the person1 dictionary:

```
import mymodule
a = mymodule.person1["age"]
print(a)
```

Naming a Module

You can name the module file whatever you like, but it must have the file extension .py

Re-naming a Module

You can create an alias when you import a module, by using the as keyword:

Example

Create an alias for mymodule called mx:

```
import mymodule as mx
a = mx.person1["age"]
print(a)
```

Built-in Modules

There are several built-in modules in Python, which you can import whenever you like.

Example

Import and use the platform module:

```
import platform
x = platform.system()
print(x)
```

Using the dir() Function

There is a built-in function to list all the function names (or variable names) in a module. The dir() function:

Example

List all the defined names belonging to the platform module:

```
import platform
x = dir(platform)
 print(x)
```

Note: The dir() function can be used on all modules, also the ones you create yourself.

Import from Module

You can choose to import only parts from a module, by using the from keyword.

Example

The module named mymodule has one function and one dictionary:

```
def greeting(name):
print("Hello, " + name)
person1 = {"name": "John", "age": 36, "country": "Norway"}
```

Packages

We don't usually store all of our files in our computer in the same location. We use a well-organized hierarchy of directories for easier access. Similar files are kept in the same directory, for example, we may keep all the songs in the "music" directory. Analogous to this, Python has packages for directories and modules for files. As our application program grows larger in size with a lot of modules, we place similar modules in one package and different modules in different packages. This makes a project (program) easy to manage and conceptually clear.

Similar, as a directory can contain sub-directories and files, a Python package can have sub-packages and modules. A directory must contain a file namedinit.py in order for Python to consider it as a package. This file can be left empty but we generally place the initialization code for that package in this file. Here is an example. Suppose we are developing a game, one possible organization of packages and modules could be as shown in the figure below.

Package Module Structure in Python Programming
Importing module from a package

We can import modules from packages using the dot (.) operator. For example, if want to import the start module in the above example, it is done as follows.

```
import Game.Level.start
```

Now if this module contains a function named select_difficulty(), we must use the full name to reference it.

```
Game.Level.start.select_difficulty(2)
```

If this construct seems lengthy, we can import the module without the package prefix as follows.
```
from Game.Level import start
```

We can now call the function simply as follows.

```
start.select_difficulty(2)
```

Yet another way of importing just the required function (or class or variable) form a module within a package would be as follows.

```
from Game.Level.start import select_difficulty
```

Now we can directly call this function.

```
select_difficulty(2)
```

Although easier, this method is not recommended. Using the full namespace avoids confusion and prevents two same identifier names from colliding. While importing packages, Python looks in the list of directories defined in sys.path, similar as for module search path.

Files

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

1. Open afile
2. Read or write (perform operation)
3. Close thefile

How to open a file?

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

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

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

Python File Modes

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

How to close a file Using Python?

When we are done with operations to the file, we need to properly close the file. Closing a file will free up the resources that were tied with the file and is done using Python close() method. Python has a garbage collector to clean up unreferenced objects but, we must not rely on it to close the file.

```
f = open("test.txt",encoding = 'utf-8')
# perform file operations
f.close()
```

This method is not entirely safe. If an exception occurs when we are performing some operation with the file, the code exits without closing the file.

A safer way is to use a try...finally block.

```
try:
   f = open("test.txt",encoding = 'utf-8')
   # perform file operations
finally:
   f.close()
```

This way, we are guaranteed that the file is properly closed even if an exception is raised, causing program flow to stop. The best way to do this is using the with statement. This ensures that the file is closed when the block inside with is exited. We don't need to explicitly call the close() method. It is done internally.

```
with open("test.txt",encoding = 'utf-8') as f:
   # perform file operations
```

How to write to File Using Python?

In order to write into a file in Python, we need to open it in write 'w', append 'a' or exclusive creation 'x' mode. We need to be careful with the 'w' mode as it will overwrite into the file if it already exists. All previous data are erased. Writing a string or sequence of bytes (for binary files) is done using write() method. This method returns the number of characters written to the file.

```
with open("test.txt",'w',encoding = 'utf-8') as f:
   f.write("my first file\n")
   f.write("This file\n\n")
   f.write("contains three lines\n")
```

This program will create a new file named 'test.txt' if it does not exist. If it does exist, it is overwritten. We must include the newline characters ourselves to distinguish different lines.

How to read files in Python?

To read a file in Python, we must open the file in reading mode. There are various methods available for this purpose. We can use the read(size) method to read in size number of data. If size parameter is not specified, it reads and returns up to the end of the file.

```
>>> f = open("test.txt",'r',encoding = 'utf-8')
>>> f.read(4) # read the first 4 data
'This'

>>>f.read(4)     # read the next 4 data
' is'

>>>f.read()      # read in the rest till end of file
'my first file\nThis file\ncontains threelines\n'

>>> f.read() # further reading returns empty sting
''
```

We can see that, the read() method returns newline as '\n'. Once the end of file is reached, we get empty string on further reading. We can change our current file cursor (position) using the seek() method. Similarly, the tell() method returns our current position (in number of bytes).

```
>>>f.tell()     # get the current file position
56

>>> f.seek(0) # bring file cursor to initial position
0

>>> print(f.read()) # read the entire file
This is my first file
This file
contains three lines
```

We can read a file line-by-line using a for loop. This is both efficient and fast.

```
>>> for line in f:
... print(line, end = '')
...
This is my first file
This file
contains three lines
```

The lines in file itself has a newline character '\n'.

Moreover, the print() end parameter to avoid two newlines when printing. Alternately, we can use read line() method to read individual lines of a file. This method reads a file till the newline, including the new line character.

```
>>> f.readline()
'This is my first file\n'

>>> f.readline()
'This file\n'>>>
f.readline()
'contains three
lines\n'

>>> f.readline()
''
```

Lastly, the readlines() method returns a list of remaining lines of the entire file. All these reading method return empty values when end of file (EOF) is reached.

```
>>> f.readlines()

['This is my first file\n', 'This file\n', 'contains three lines\n']
```

Python File Methods

There are various methods available with the file object. Some of them have been used in above examples. Here is the complete list of methods in text mode with a brief description.

Python File Methods

Method	**Description**
close()	Close an open file. It has no effect if the file is already closed.
detach()	Separate the underlying binary buffer from the TextIOBase and return it.
fileno()	Return an integer number (file descriptor) of the file.
flush()	Flush the write buffer of the file stream.
isatty()	Return True if the file stream is interactive.
read(n)	Read at most n characters form the file. Reads till end of file if it is negative or None.
readable()	Returns True if the file stream can be read from.
readline(n=-1)	Read and return one line from the file. Reads in at most n bytes if specified.
readlines(n=-1)	Read and return a list of lines from the file. Reads in at most n bytes/characters if specified.
seek(offset,from=SE EK_SET)	Change the file position to offset bytes, in reference to from (start, current, end).
seekable()	Returns True if the file stream supports random access.
tell()	Returns the current file location.
truncate(size=None)	Resize the file stream to size bytes. If size is not specified, resize to current location.
writable()	Returns True if the file stream can be written to.
write(s)	Write string s to the file and return the number of characters written.
writelines(lines)	Write a list of lines to the file.

Python Packages of Interest

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 is easy for machines to parse and generate. JSON is built on two structures - a collection of name-value pairs (e.g. a Python dictionary) and ordered lists of values (e.g.. a Python list).
2. **XML:** XML (Extensible Markup Language) is a data format for structured document interchange. The Python minion library provides a minimal implementation of the Document Object Model interface and has an API similar to that in other languages.
3. **HTTPLib & URLLib:** HTTPLib2 and URLLib2 are Python libraries used in network/internet programming
4. **SMTPLib:** Simple Mail Transfer Protocol (SMTP) is a protocol which handles sending email and routing e-mail between mail servers. The Python smtplib module provides an SMTP client session object that can be used to send email.
5. **NumPy:**NumPy is a package for scientific computing in Python. NumPy provides support for large multi-dimensional arrays andmatrices
6. **Scikit-learn:** Scikit-learn is an open source machine learning library for Python that provides implementations of various machine learning algorithms for classification, clustering, regression and dimension reduction problems.

JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on two structures:

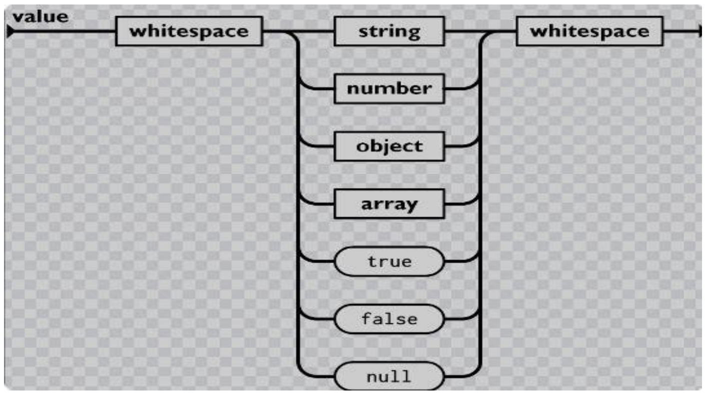
- A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.

These are universal data structures.It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

JSON Syntax :

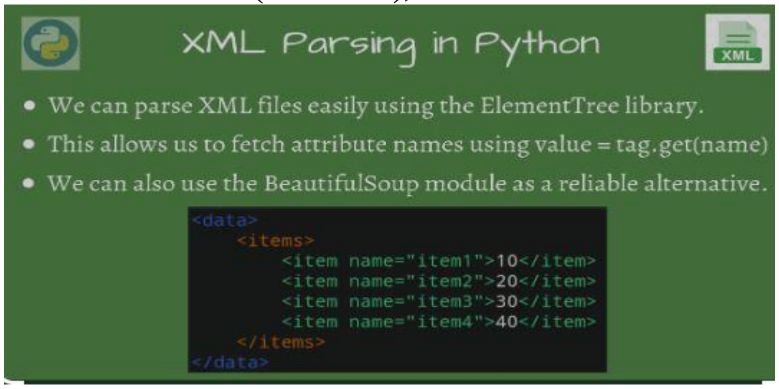
Convert from JSON to Python:

```
import json
# some JSON:
x =  '{ "name":"John", "age":30, "city":"New York"}'
# parse x:
y = json.loads(x)
# the result is a Python dictionary:
print(y["age"])
```



XML

The Extensible Markup Language (XML) is a simple text-based format for representing structured information: documents, data, configuration, books, transactions, invoices, and much more. It was derived from an older standard format called SGML (ISO 8879), in order to be more suitable for Web use.



XML Parsing in Python

- We can parse XML files easily using the ElementTree library.
- This allows us to fetch attribute names using value = tag.get(name)
- We can also use the BeautifulSoup module as a reliable alternative.

```
<data>
    <items>
        <item name="item1">10</item>
        <item name="item2">20</item>
        <item name="item3">30</item>
        <item name="item4">40</item>
    </items>
</data>
```

HTTP Lib.

This module defines classes which implement the client side of the HTTP and HTTPS protocols. It is normally not used directly — the module urllib uses it to handle URLs that use HTTP and HTTPS. Note. HTTPS support is only available if the socket module was compiled with SSL support.

SMTP Lib

The **smtplib** module defines an SMTP client session object that can be used to send mail to any internet machine with an SMTP or ESMTP listener daemon. For details of SMTP and ESMTP operation, consult RFC 821 (Simple Mail Transfer Protocol) and RFC 1869 (SMTP Service Extensions**).**

```
from smtplib import SMTP
>>> with SMTP("domain.org") as smtp:
...     smtp.noop()
...
(250, b'Ok')
>>>
```

UNIT IV

IoT PHYSICAL DEVICES AND ENDPOINTS

IoT Device

A "Thing" in Internet of Things (IoT) can be any object that has a unique identifier and which can send/receive data (including user data) over a network (e.g., smart phone, smartTV, computer, refrigerator, car, etc.).

• IoT devices are connected to the Internet and send information about themselves or about their surroundings (e.g. information sensed by the connected sensors) over a network (to other devices or servers/storage) or allow actuation upon the physical entities/environment around them remotely.

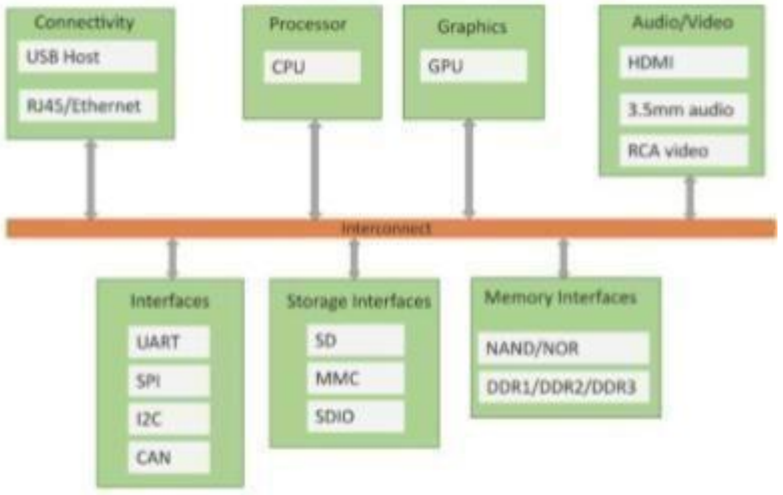
IoT Device Examples

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

Basic building blocks of an IoT Device

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

Block diagram of an IoT Device

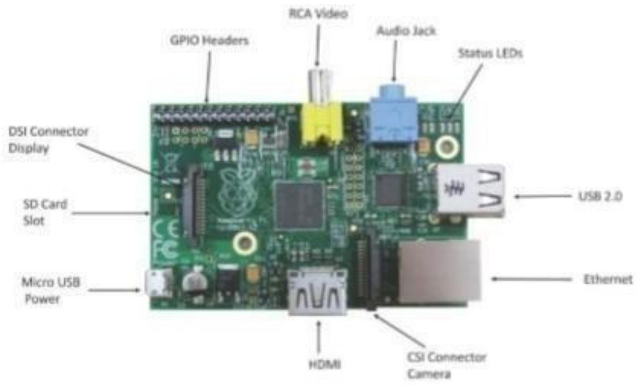


Exemplary Device: Raspberry Pi

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

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

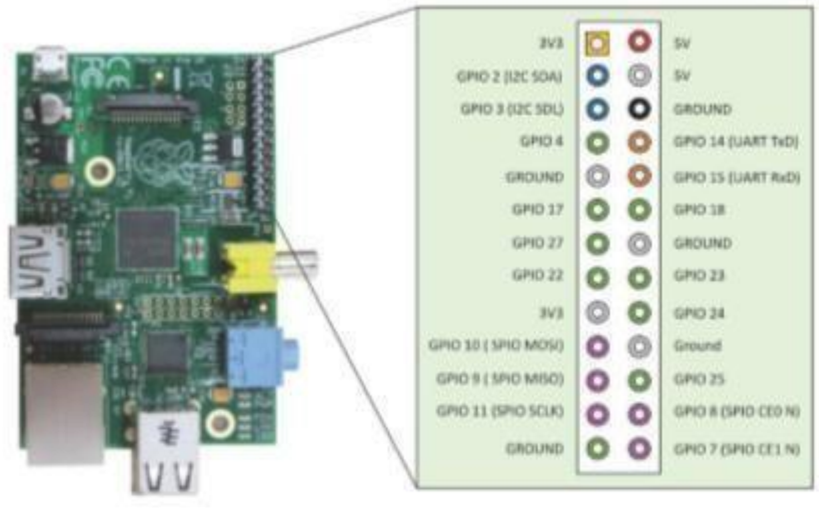
Raspberry Pi



Linux on Raspberry Pi

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

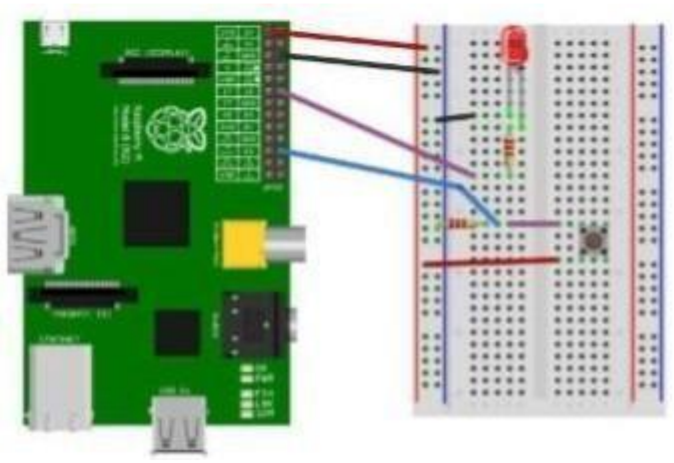
Raspberry Pi GPIO



Raspberry Pi Interfaces

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

Raspberry Pi Example: Interfacing LED and switch with Raspberry Pi



```
from time import sleeP

import RPi.GPIO asGPIO

GPIO.setmode(GPIO.BCM)

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

 #LEDPin

GPIO.setup(18,GPIO.OUT)

state=false


deftoggleLED(pin):

        state = not state

        GPIO.output(pin,state)

        whileTrue:try:

if (GPIO.input(25) ==True):

                        exceptKeyboardInterrupt:

                                exit()
```