SYLLABUS

UNIT-I	**INTRODUCTION TO INTERNET OF THINGS (IoT)**	
Definition and characteristics of IoT, physical design of IoT, logical design of IoT, IoT enabling technologies, IoT levels and deployment, domain specific IoTs.\		
UNIT-II	**IoT AND M2M**	
Introduction, M2M, difference between IoT and M2M, software defined networking (SDN) and network function virtualization (NFV) for IoT, basics of IoT system management with NETCONF-YANG.		
UNIT-III	**IoT PLATFORMS DESIGN METHODOLOGY**	
IoT Architecture: State of the art introduction, state of the art; Architecture reference model: Introduction,reference model and architecture, IoT reference model. Logical design using Python: Installing Python, Python data types and data structures, control flow,functions, modules, packages, file handling.		
UNIT-IV	**IoT PHYSICAL DEVICES AND ENDPOINTS**	
Introduction to Raspberry Pi interfaces (Serial, SPI, I2C), programming Raspberry PI with Python,other IoT devices.		
UNIT-V	**IoT PHYSICAL SERVERS AND CLOUD OFFERINGS**	
Introduction to cloud storage models and communication APIs, WAMP – AutoBahn for IoT, Xively cloud for IoT, case studies illustrating IoT design – home automation, smart cities, smart environment.		
Text Books:		
1. Arshdeep Bahga, Vijay Madisetti, ‐Internet of Things: A Hands-on-Approach‖, VPT, 1st Edition, 2014.		
Reference Books:		
1. Adrian McEwen, Hakim Cassimally, ‐Designing the Internet of Things‖, John Wiley and Sons 2014. 2. Francis daCosta, **“**Rethinking the Internet of Things: A Scalable Approach to Connecting Everything‖,A press Publications, 1st Edition2013.		
Web References:		
1. https://www.upf.edu/pra/en/3376/22580. 2. https://www.coursera.org/learn/iot. 3. https://bcourses.berkeley.edu. 4. www.innovianstechnologies.com.		
E-Text Books:		
1. https://mitpress.mit.edu/books/internet-things 2. http://www.apress.com		

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:

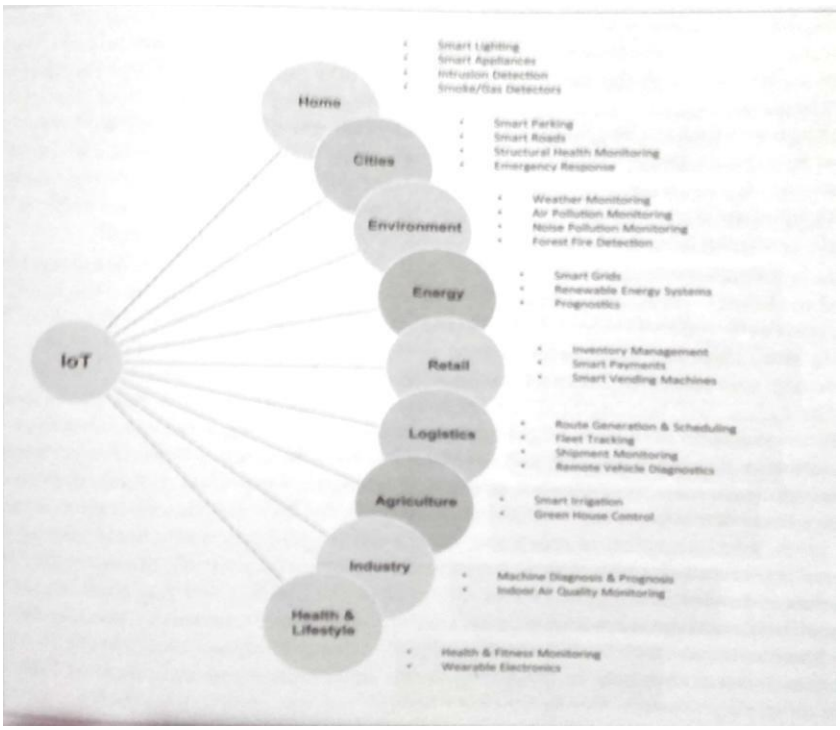
A dynamic global n/w infrastructure with self configuring capabilities based on standard and interoperable communication protocols where physical and virtual –things‖ have identities, physical attributes and virtual personalities and use intelligent interfaces, and are seamlessly integrated into information 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.

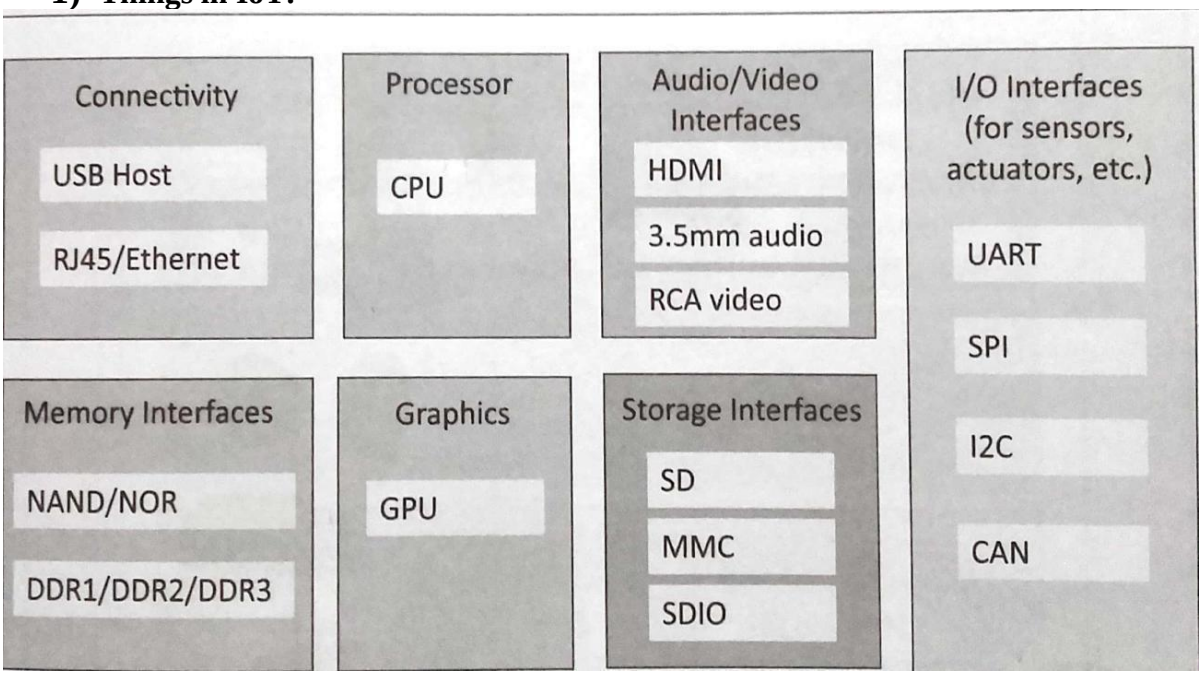
Applications of IoT:

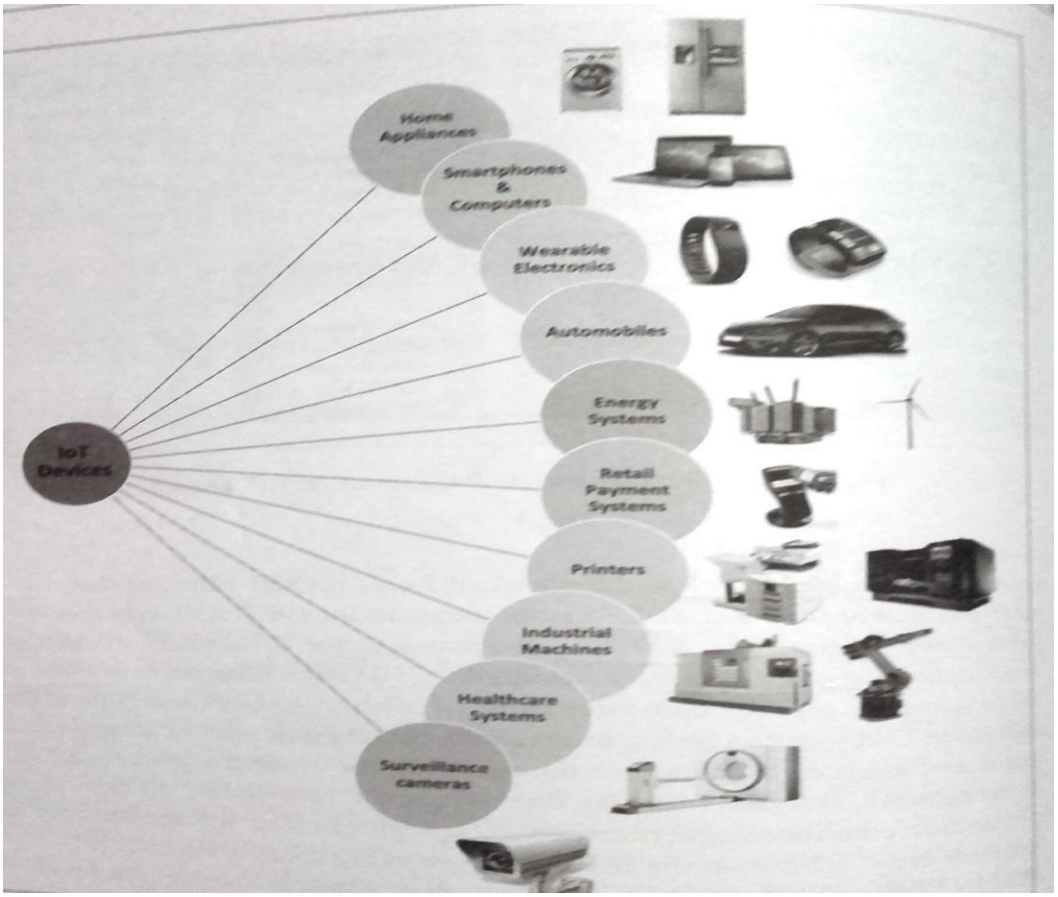
1) Home
2) Cities
3) Environment
4) Energy
5) Retail
6) Logistics
7) Agriculture
8) Industry
9) Health & Life Style



Physical Design of IoT

1) **Things in IoT:**



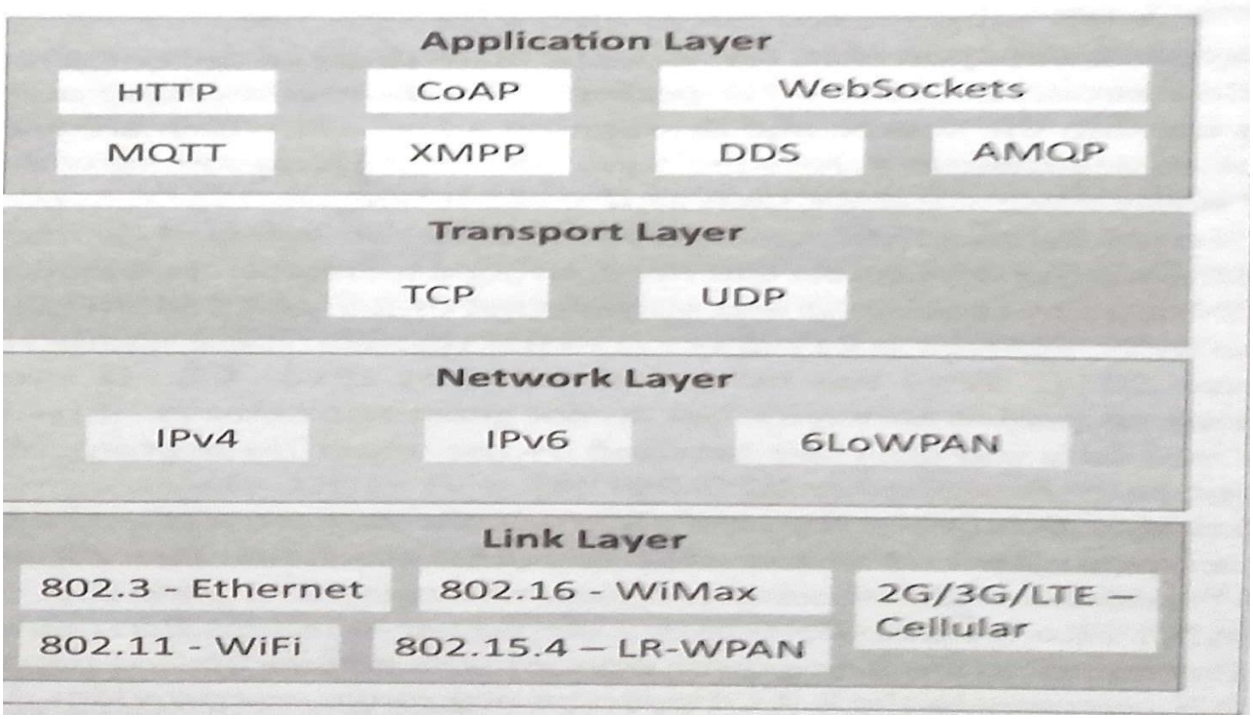


The things in IoT refers to IoT devices which have unique identities and perform remote sensing, actuating and monitoring capabilities. IoT devices can exchange data with other connected devices applications. It collects data from other devices and process data either locally or remotely.

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 and (iv) audio/video interfaces.

2) IoT Protocols:

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



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).

B) **Network/Internet Layer:** Responsible for sending IP datagrams from source n/w to destination n/w. Performs the host addressing and packet routing. Datagrams contains source and destination address.

Protocols:

- **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.
- **IPv6:** Internet Protocol version6 uses 128 bit address scheme and allows 2**128 addresses.

- **6LOWPAN:**(IPv6overLowpowerWirelessPersonalAreaNetwork)operates in 2.4 GHz frequency range and data transfer 250 kb/s.

C) 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.

D) **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:

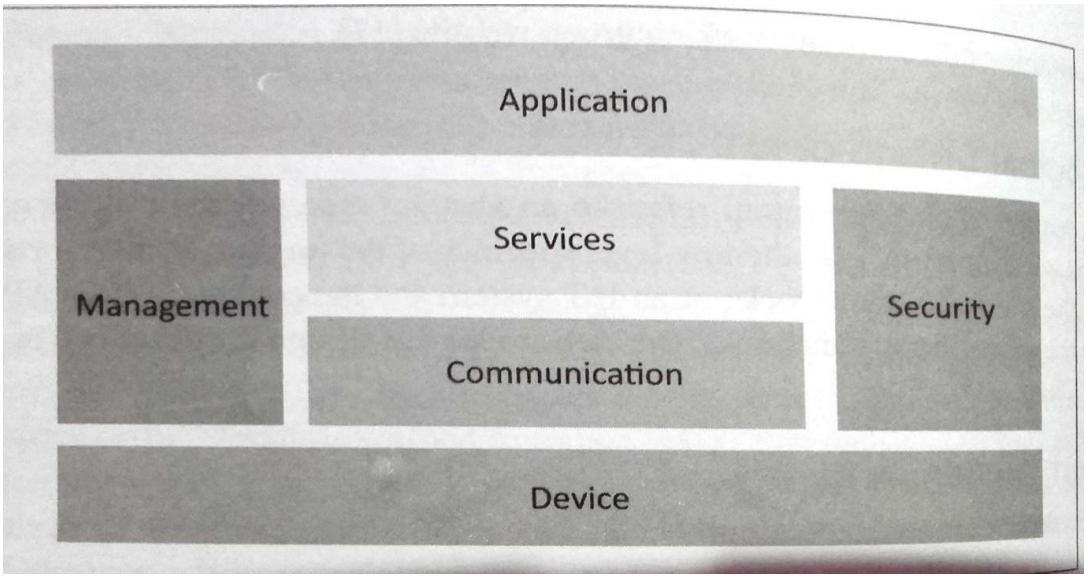
- **HTTP:** Hyper Text Transfer Protocol that forms foundation of WWW. Follow request-response model Stateless protocol.
- **CoAP:** Constrained Application Protocol for machine-to-machine (M2M) applications with constrained devices, constrained environment and constrained n/w. Uses client-server architecture.
- **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 Comm. APIs

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

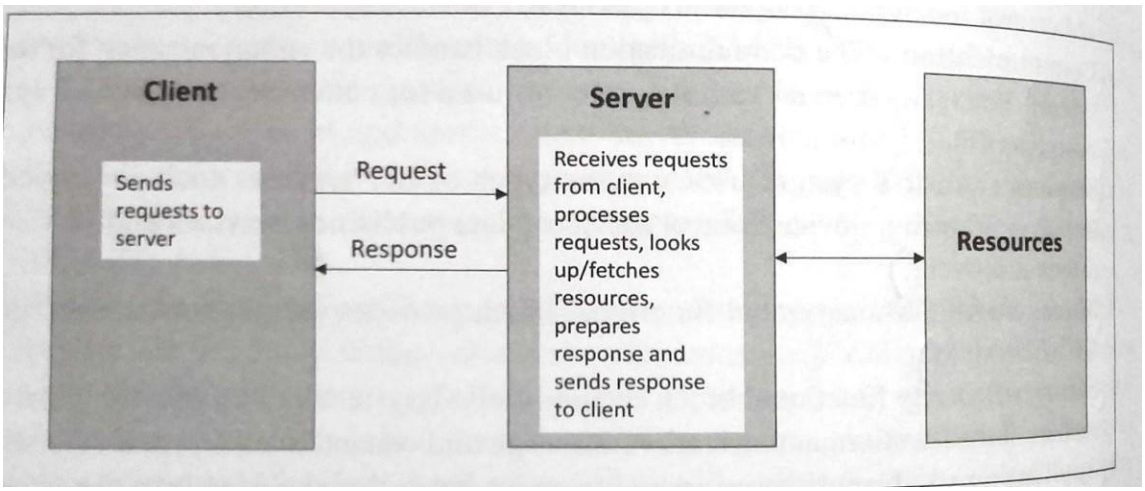


- **Device:** An IoT system comprises of devices that provide sensing, actuation, monitoring and control functions.
- **Communication:** handles the communication for IoTsystem.
- **Services:** for device monitoring, device control services, data publishing services and services for device discovery.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

2) IoT Communication Models:

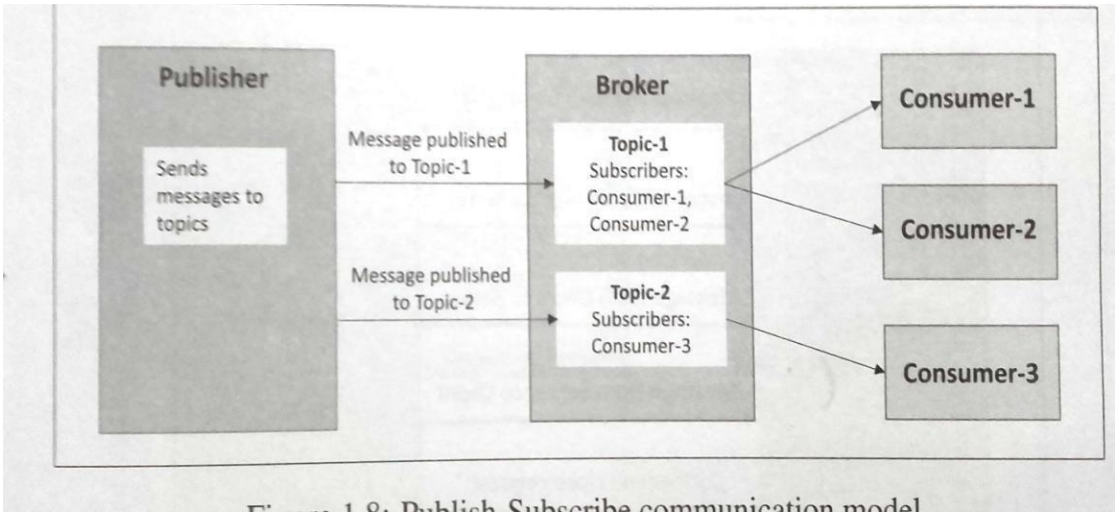
1) Request-Response 2) Publish-Subscibe 3)Push-Pull4) ExclusivePair

1) Request-Response Model:



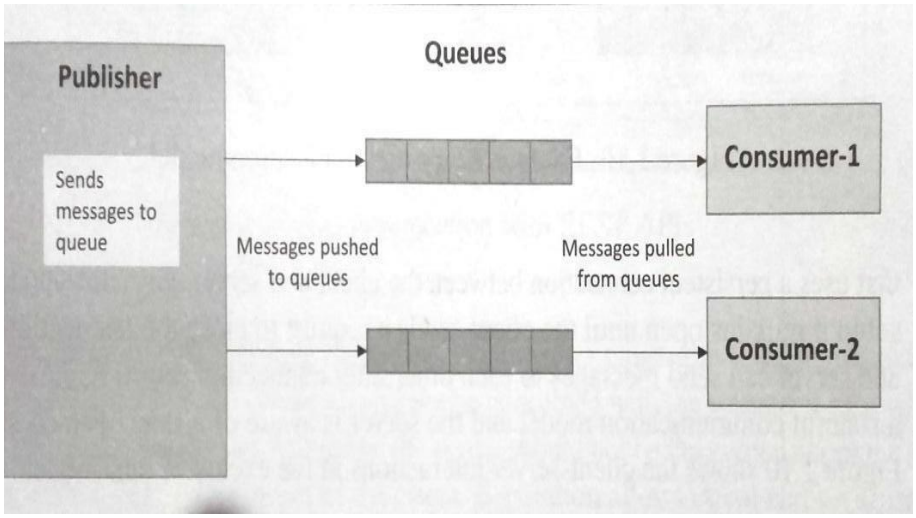
In which the client sends request to the server and the server replies to requests. Is a stateless communication model and each request-response pair is independent of others.

2) **Publish-Subscibe Model:**

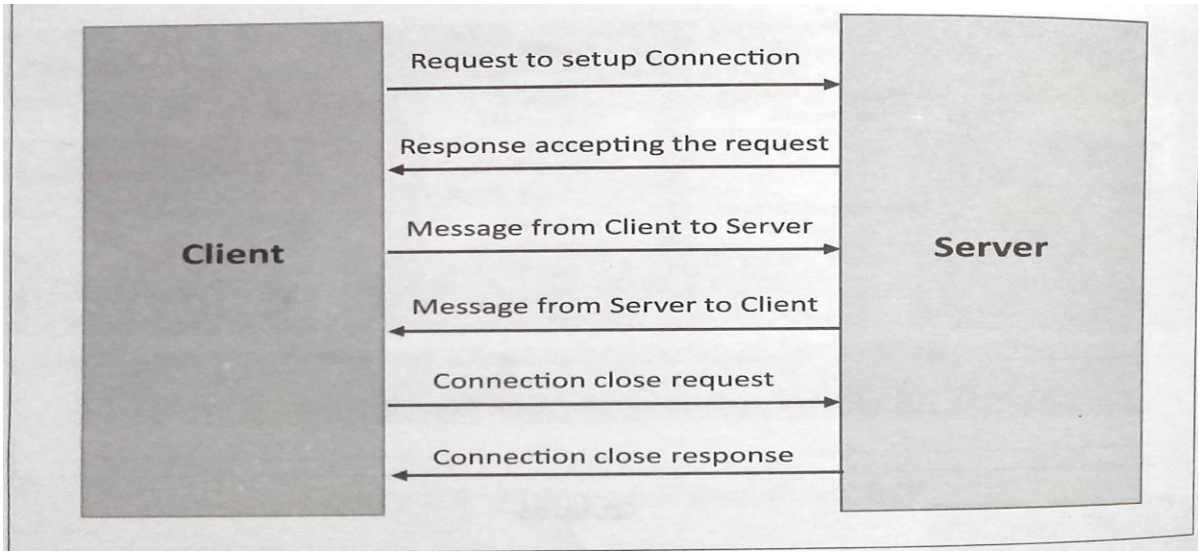


Involves publishers, brokers and consumers. Publishers are source of data. Publishers send data to the topics which are managed by the broker. Publishers are not aware of the consumers. Consumers subscribe to the topics which are managed by the broker. When the broker receives data for a topic from the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



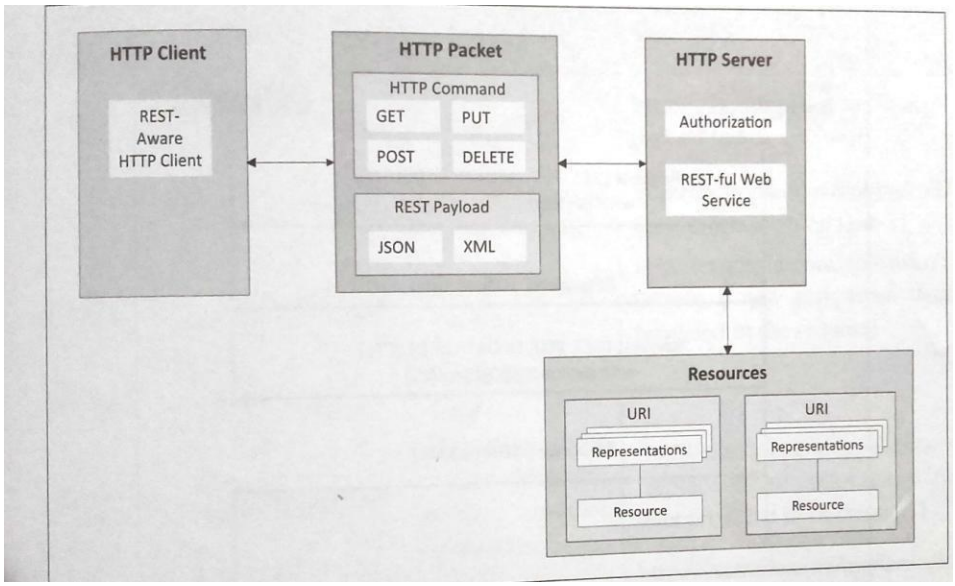
3) IoT Communication APIs:

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

b) WebSocket based Communication APIs(Exclusive PairBased Model)

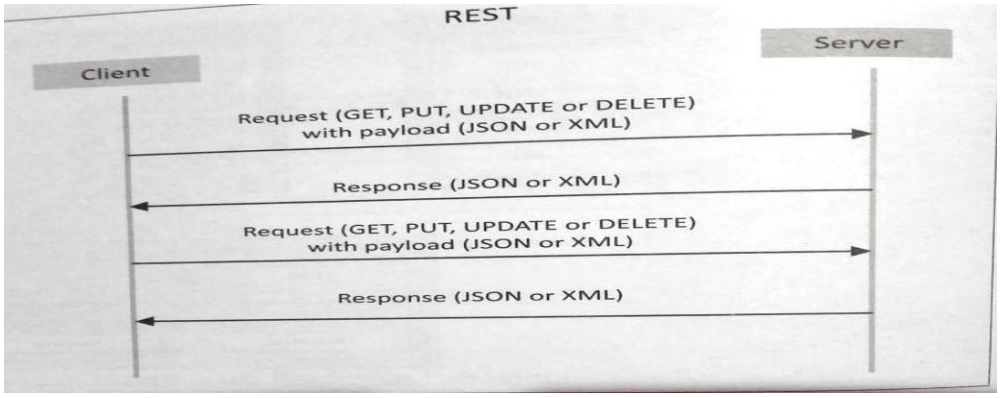
a) REST based communication APIs: Representational State Transfer(REST) is a set of architectural principles by which we can design web services and web APIs that focus on a system_s resources and have resource states are addressed and transferred.

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



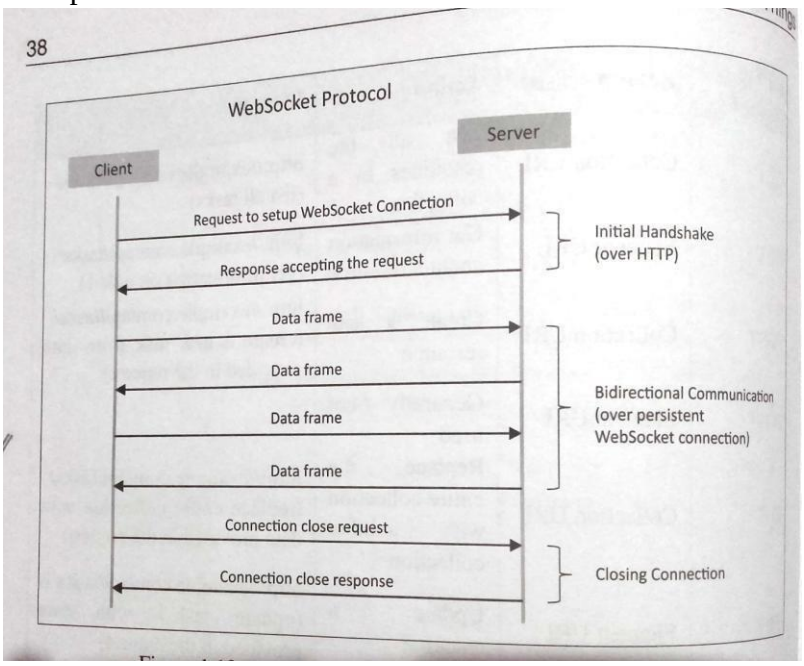
Client-Server: The principle behind client-server constraint is the separation of concerns. Separation allows client and server to be independently developed and updated.
Stateless: Each request from client to server must contain all the info. Necessary to understand the request, and cannot take advantage 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:



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.



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 or the user interface to the application itself.

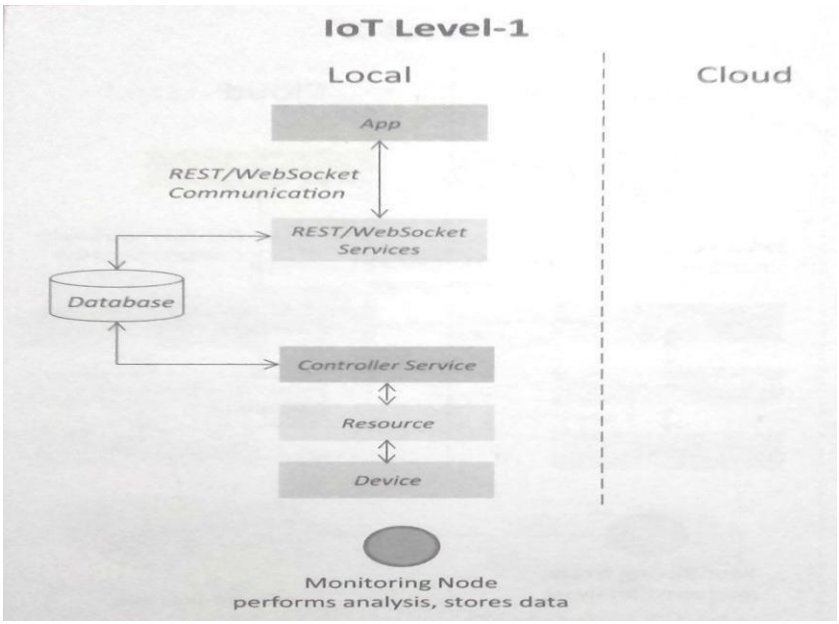
3) **Big Data Analytics:** Some examples of big data generated by IoT are
   - Sensor data generated by IoT systems.
   - Machine sensor data collected from sensors established in industrial and energy systems.
   - Health and fitness data generated IoT devices.
   - Data generated by IoT systems for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

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

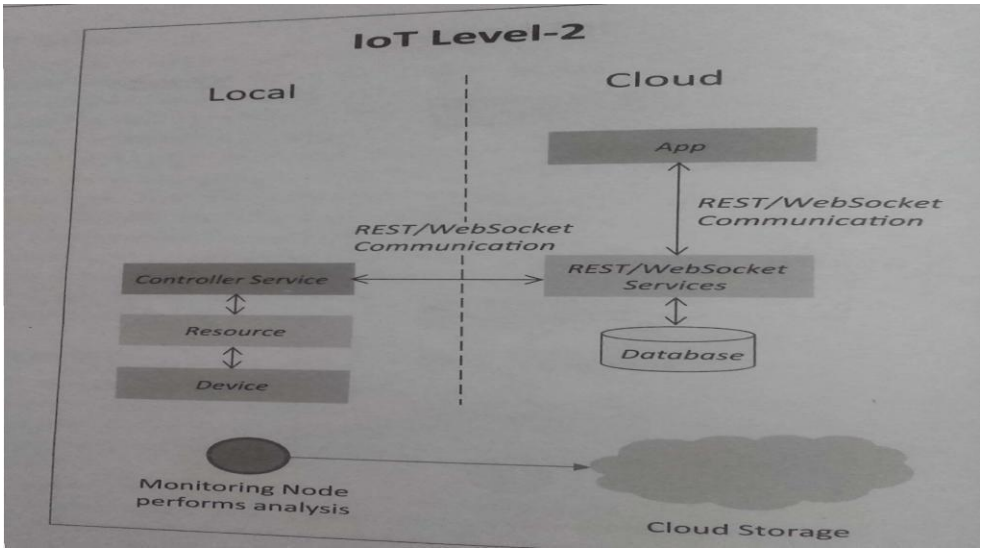
5) **Embedded Systems:** is a computer system that has computer hardware and software embedded to perform specific tasks. Embedded System range from low cost miniaturized devices such as digital watches to devices such as digital cameras, POS terminals, vending machines, appliances etc.,

IoT Levels and Deployment Templates

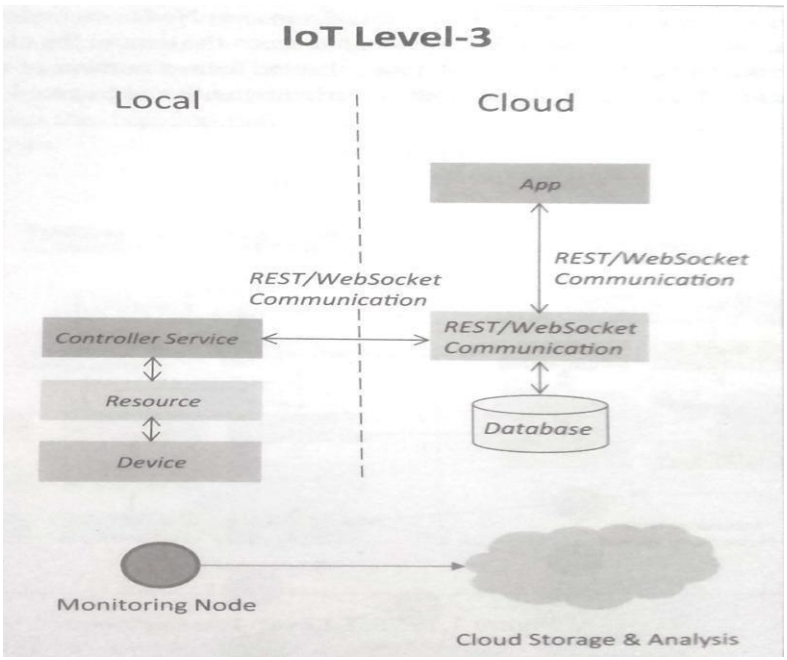
1) **IoT Level1:** System has a single node that performs sensing and/or actuation, stores data, performs analysis and host the application as shown in fig. Suitable for modeling low cost and low complexity solutions where the data involved is not big and analysis requirement are not computationally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



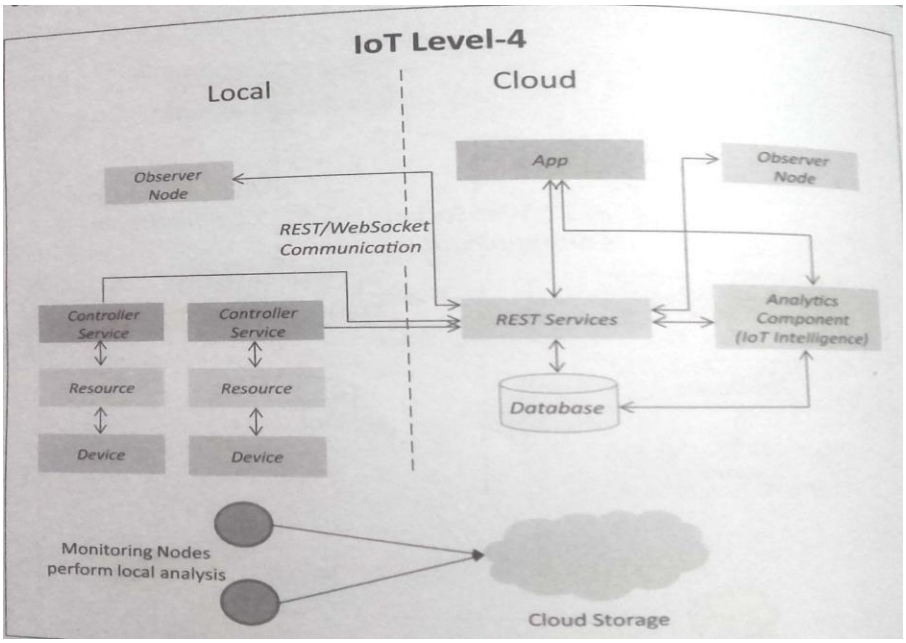
2) **IoT Level2:** has a single node that performs sensing and/or actuating and local analysis as shown in fig. Data is stored in cloud and application is usually cloud based. Level2 IoT systems are suitable for solutions where data are involved is big, however, the primary analysis requirement is not 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.

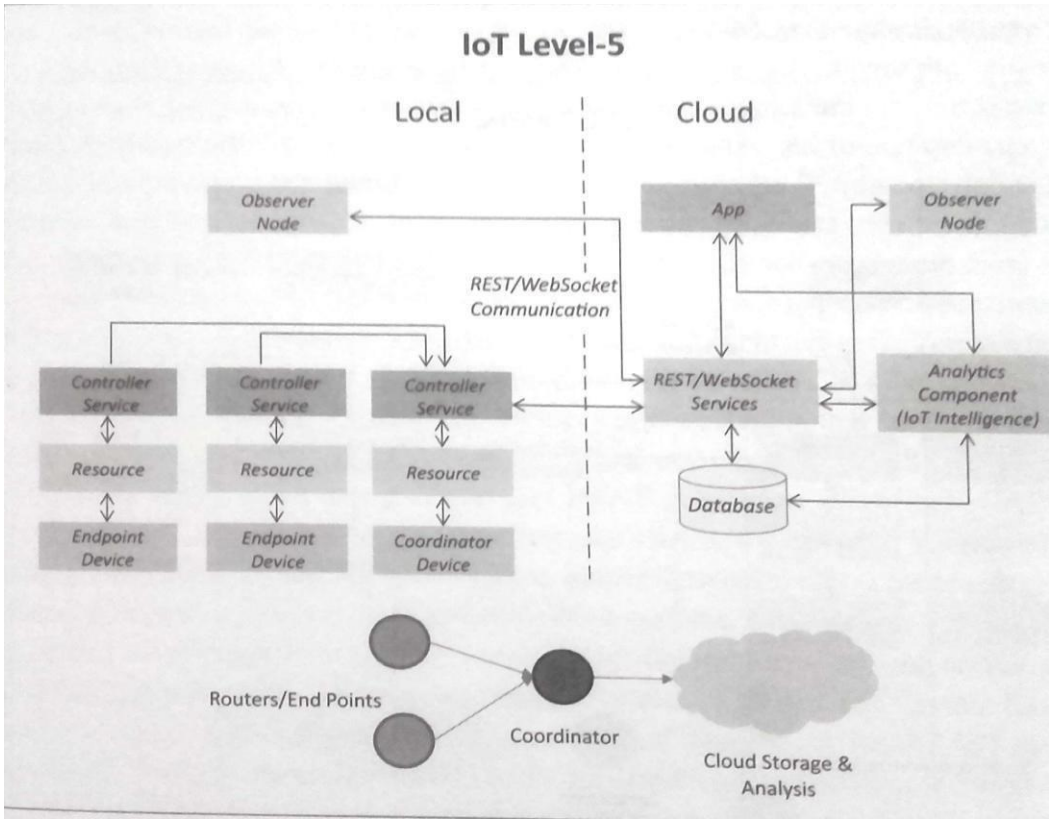


4) **IoT Level4:** System has multiple nodes that perform local analysis. Data is stored in the cloud and application is cloud based as shown in fig. Level4 contains local and cloud based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices. An example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

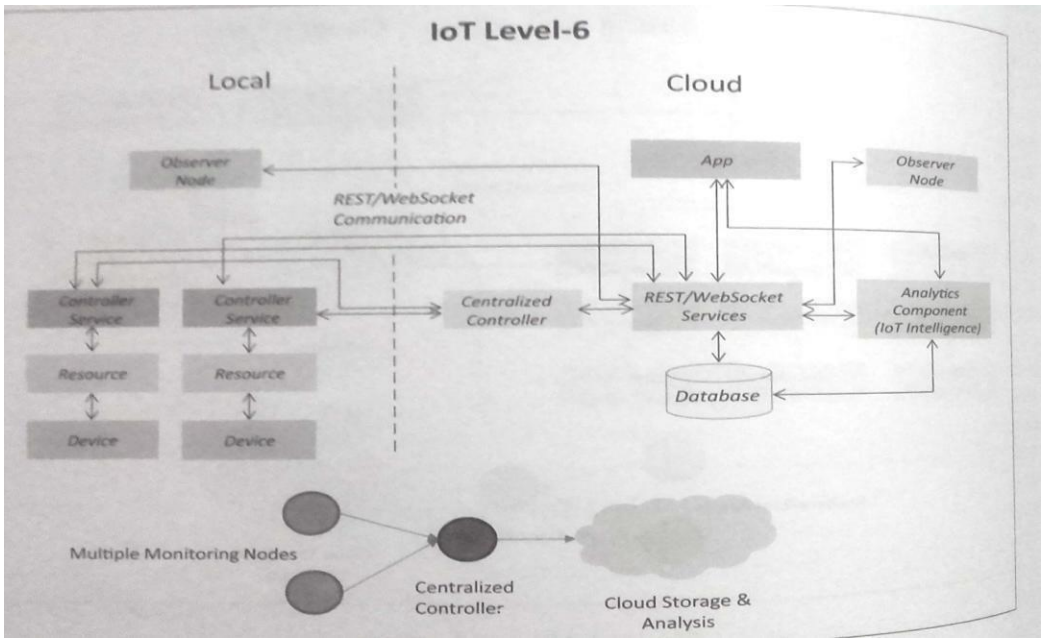


5) **IoT Level5:** System has multiple end nodes and one coordinator node as shown in fig. The end nodes that perform sensing and/or actuation. Coordinator node collects data from theendnodesandsendstothecloud.Dataisstoredandanalyzedinthecloudand

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

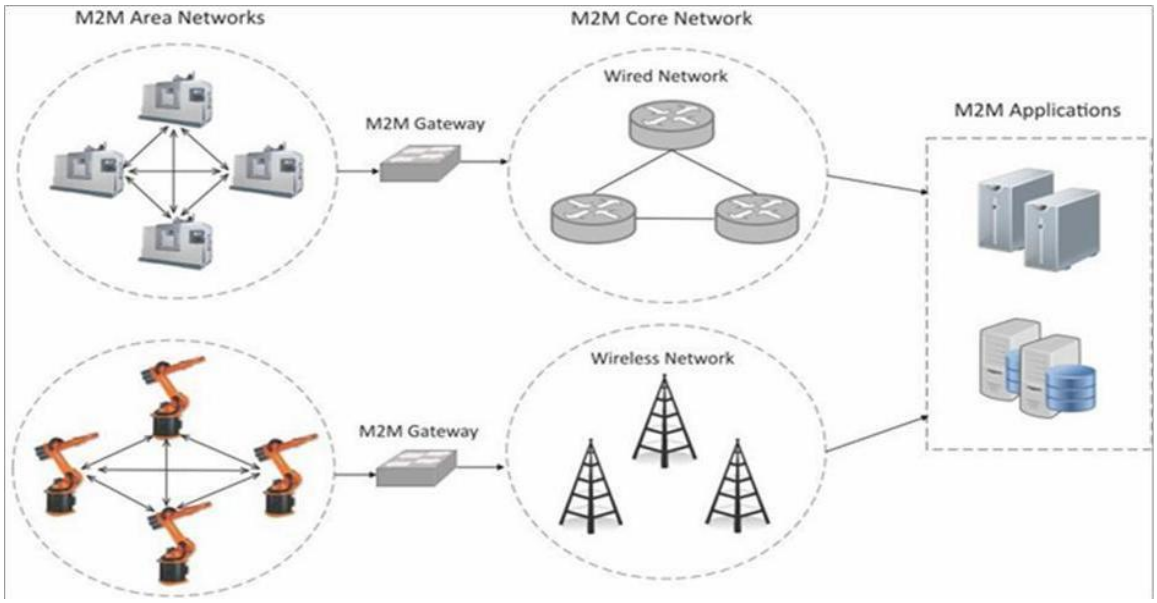
IoT and M2M

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.

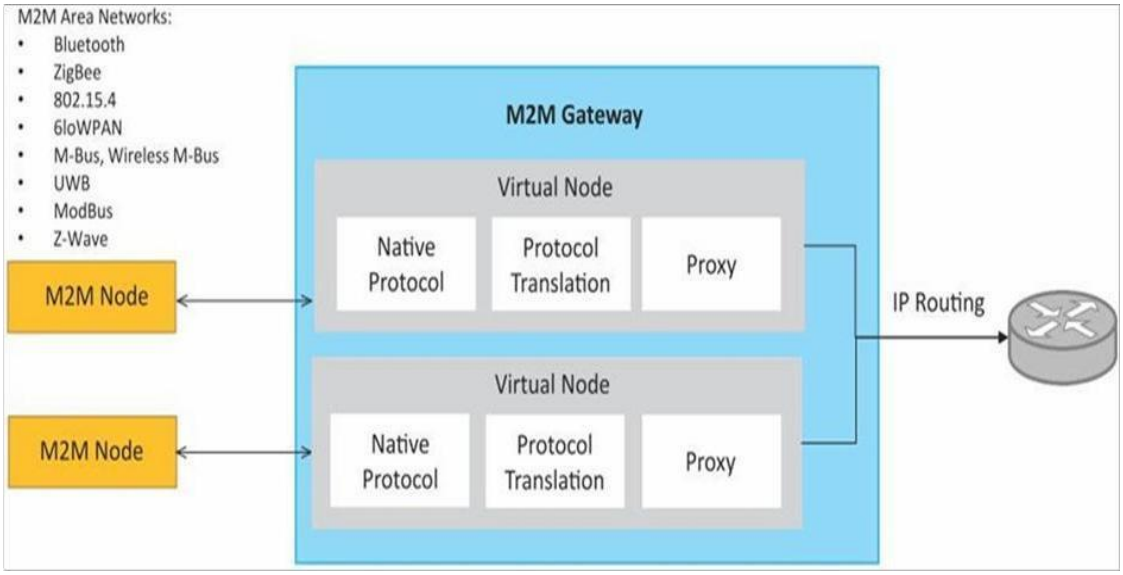


Fig. Shows a block diagram of an M2M gateway. The communication between M2M nodes and the M2M gateway is based on the communication protocols which are naive to the M2M are network. M2M gateway performs protocol translations to enable Ip-connectivity for M2M are networks. M2M gateway acts as a proxy 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

1) **Communication Protocols:**
   - Commonly uses M2M protocols include ZigBee, Bluetooth, ModBus, M-Bus, Wireless M-Bus tec.,
   - In IoT uses HTTP, CoAP, WebSocket , MQTT ,XMPP ,DDS ,AMQP etc.,
2) **Machines in M2M Vs Things in IoT:**
   - Machines in M2M will be homogenous whereas Things in IoT will be heterogeneous.
3) **Hardware Vs Software Emphasis:**
   - the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software.
4) **Data Collection &Analysis**
   - M2M data is collected in point solutions and often in on-premises storage infrastructure.
   - The data in IoT is collected in the cloud (can be public, private orhybrid cloud).

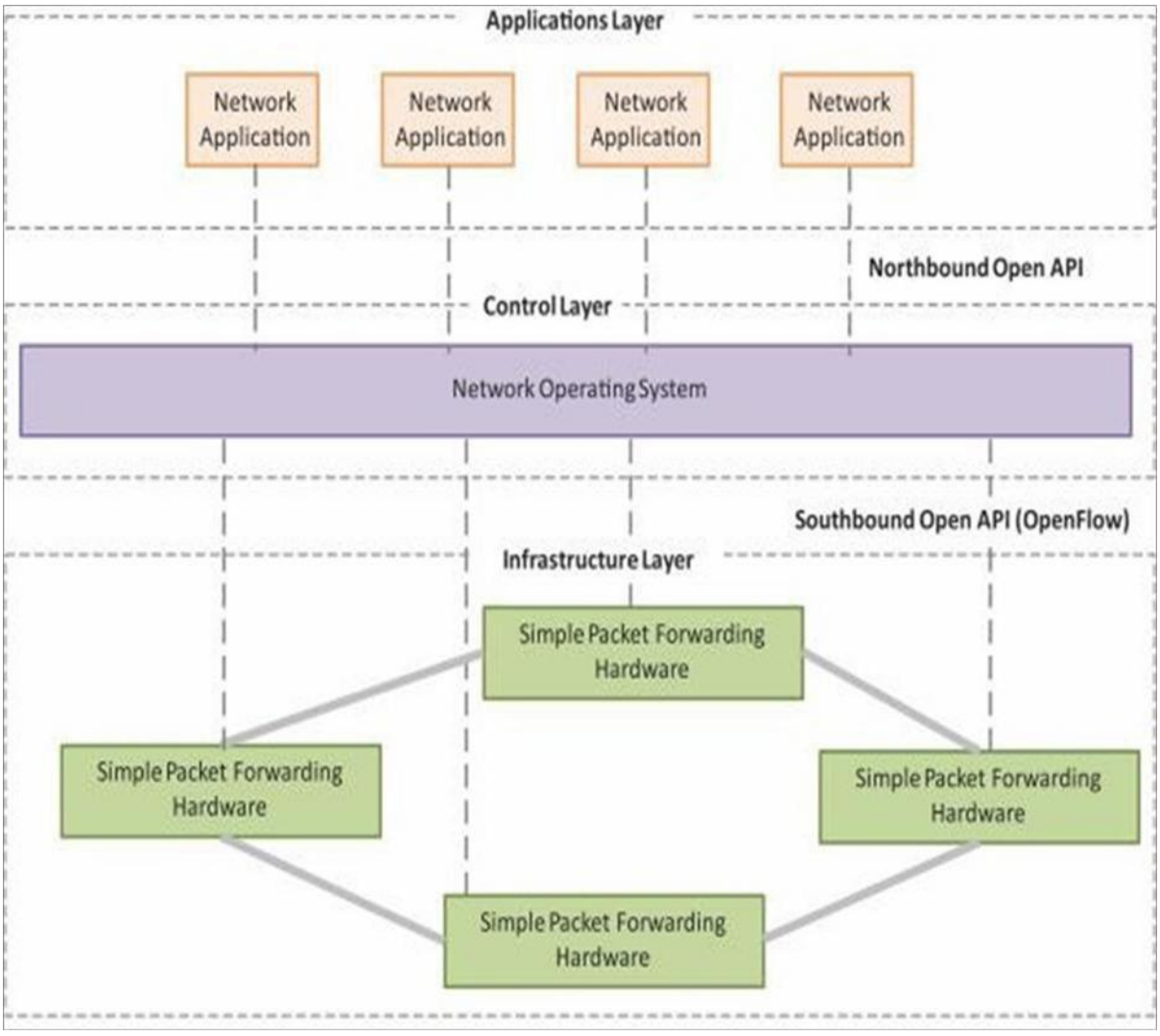
5) **Applications**

- □ M2M data is collected in point solutions and can be accessed by on-premises applications such as diagnosis applications, service management applications, and on- premisis enterprise applications.
- □ IoT data is collected in the cloud and can be accessed by cloud applications such as analytics applications, enterprise applications, remote diagnosis and management applications, etc.

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**

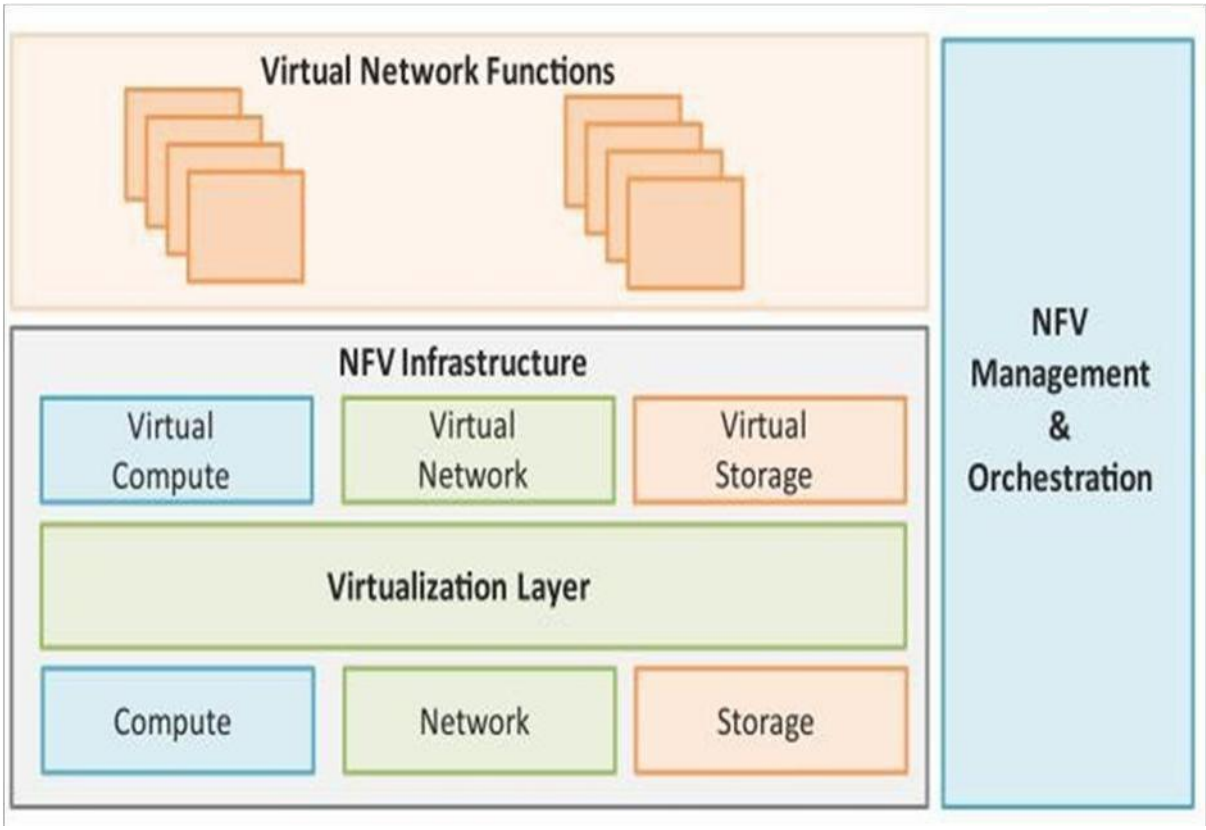
   SDN architecture supports programmable open APIs for interface between the SDN application and control layers (Northbound interface).

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.

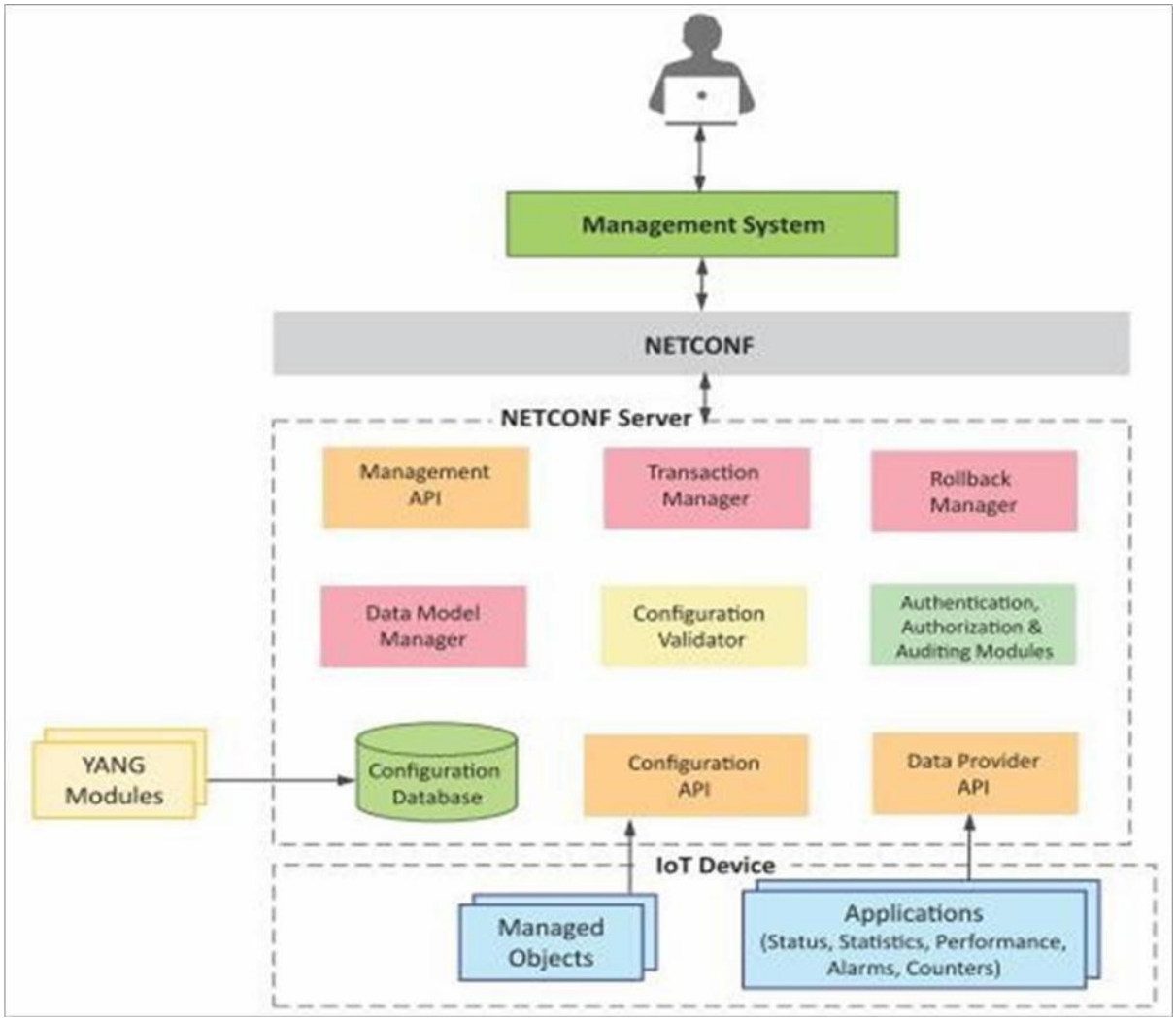
1) **Automating Configuration :** IoT system management capabilities can helpin automating the system configuration.
2) **Monitoring Operational & Statistical Data :** Management systems can help in monitoring opeartional and statistical data of a system. This data can be used for fault diagnosis or 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 capability of 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.

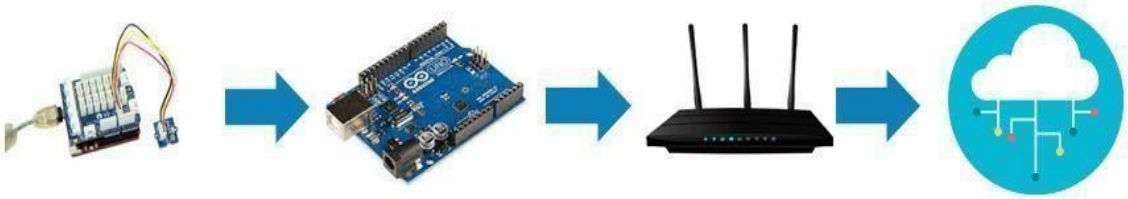
UNIT-III

IOT ARCHITECTURE AND PYTHON

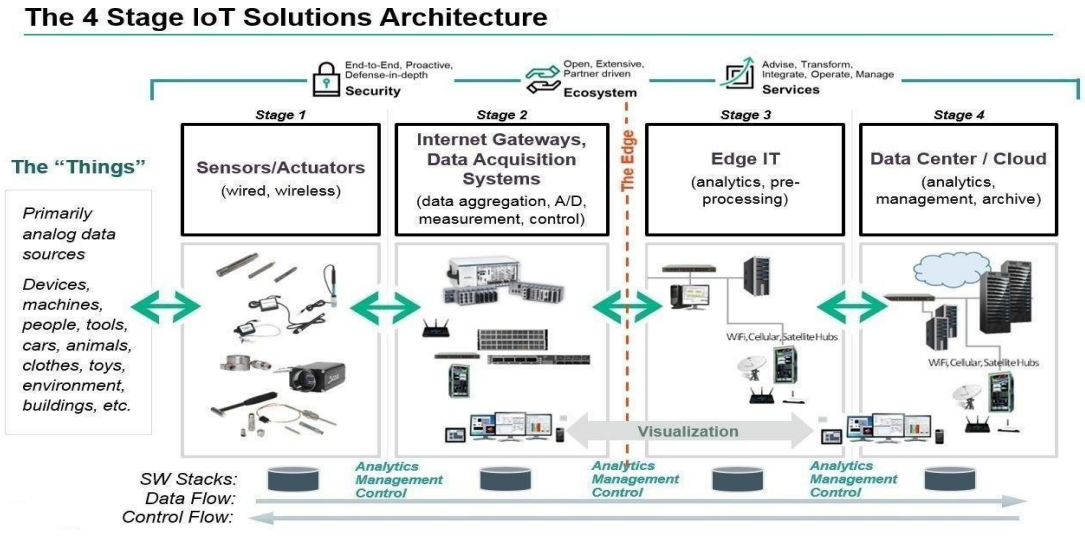
State of the art

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

1) Sensors
2) Devices
3) Gateway
4) Cloud



Stages of IoT Architecture



Stage 1:-

Sensors/actuators

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

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

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

Stage 2:-

The Internet gateway

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

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

Stage 3:-

Edge IT

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

aggregate and convert the data, analyze it, and send only projections as to when each device will fail or need service.

Stage 4:-

The data center and cloud

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

REFERENCE MODEL AND ARCHITECTURE

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

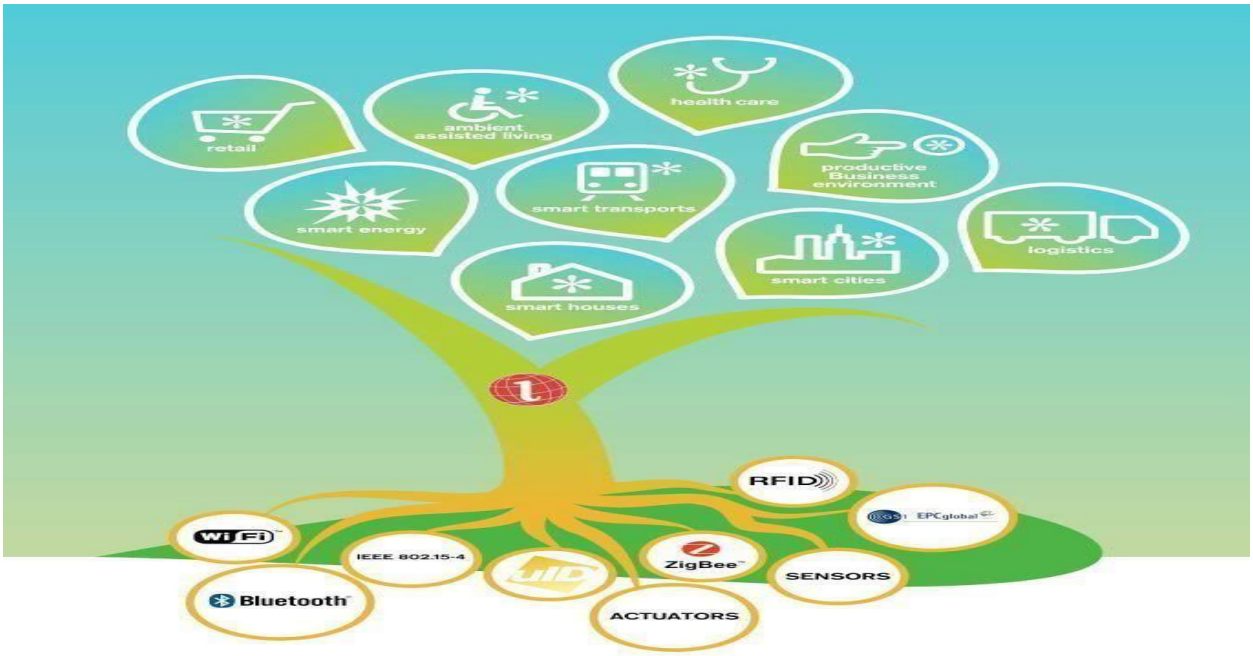
The central choice of the IoT-A project was to base its work on the current state of the art, rather than using a clean-slate approach. Due to this choice, common traits are derived to form the base line of the Architectural Reference Model (ARM). This has the major advantage of ensuring backward compatibility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

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

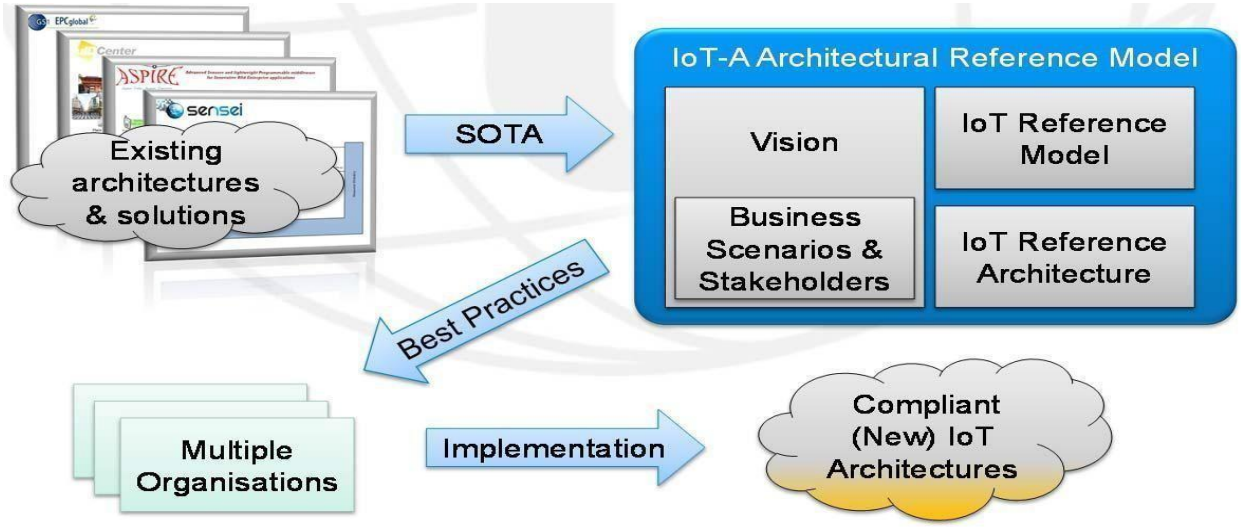
As any metaphoric representation, this tree does not claim to be fully consistent in its depiction; it should therefore not be interpreted too strictly. On the one hand, the roots of this tree are spanning across a selected set of communication protocols (6LoWPAN, Zigbee, IPv6,…) and device technologies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

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

The IOT-A Tree



IoT-A architectural reference model building blocks.



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

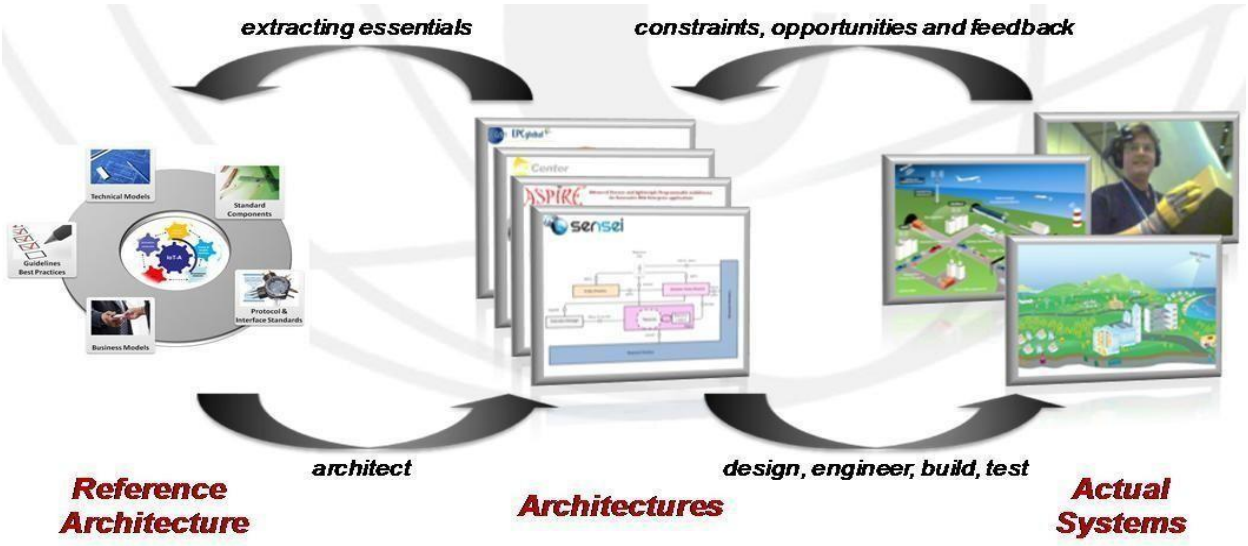
The vision summarizes the rationale for providing an architectural reference model for the IoT. At the same time it discusses underlying assumptions, such as motivations. It also

discusses how the architectural reference model can be used, the methodology applied to the architecture modeling, and the business scenarios and stakeholders addressed.

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

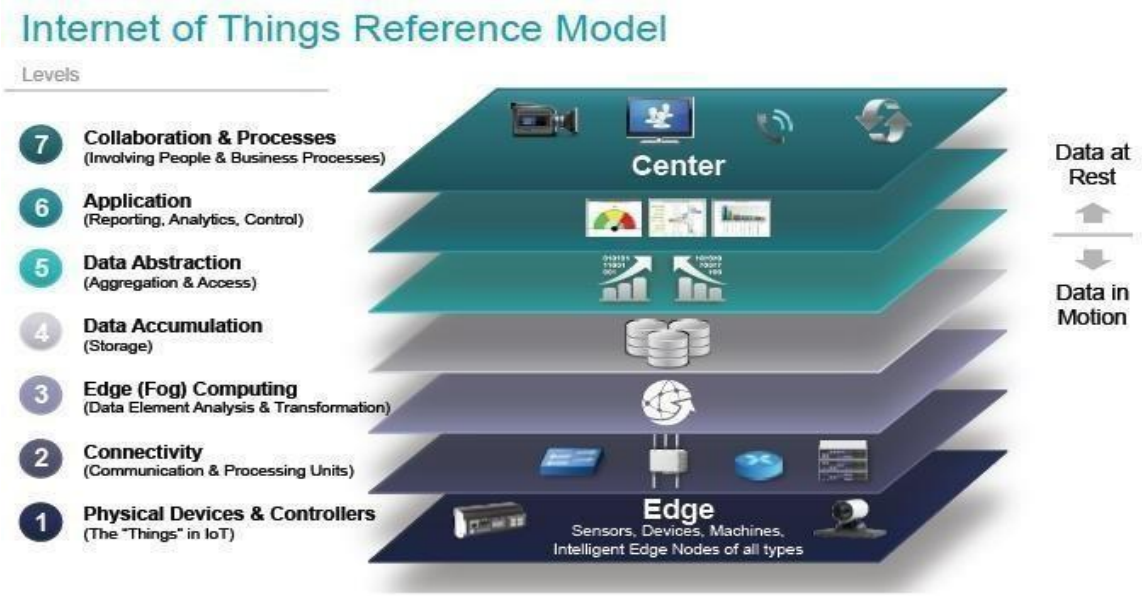
The IoT Reference Model provides the highest abstraction level for the definition of the IoT-A Architectural Reference Model. It promotes a common understanding of the IoT domain. The description of the IoT Reference Model includes a general discourse on the IoT domain, an IoT Domain Model as a top-level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

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



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

- ✓ **Simplifies**: It helps break down complex systems so that each part is more understandable. **Clarifies**: It provides additional information to precisely identify levels of the IoT and to establish common terminology.
- ✓ **Identifies**: It identifies where specific types of processing is optimized across different parts of the system.
- ✓ **Standardizes**: It provides a first step in enabling vendors to create IoT products that work with each other.
- ✓ **Organizes**: It makes the IoT real and approachable, instead of simply conceptual.



Level 1: Physical Devices and Controllers

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

Level 2: Connectivity

Communications and connectivity are concentrated in one level—Level 2. The most important function of Level 2 is reliable, timely information transmission. This includes transmissions:

- Between devices (Level 1) and the network
- Across networks(east-west)
- Between the network (Level 2) and low-level information processing occurring at Level 3

Traditionaldatacommunicationnetworkshavemultiplefunctions,asevidencedbythe International Organization for Standardization (ISO) 7-layer reference model. However, a complete IoT system contains many levels in addition to the communications network. One objective of the IoT Reference Model is for communications and processing to be executed by existing networks. The IoT Reference Model does not require or indicate creation of a different network—it relies on existing networks. As Level 1 devices proliferate, the ways in which they interact with Level 2 connectivity equipment may change. Regardless of the details, Level 1 devices communicate through the IoT system by interacting with Level 2 connectivity equipment.

Connectivity
(Communication & Processing Units)

Level 2 functionality focuses on East-West communications

Connectivity includes:

Communicating with Bnd between the Level I devices

- Reliable delivery Bcross the network(s)
- Implementation of various protocols
- Switching and routing

Translation between protocols Security at the network level

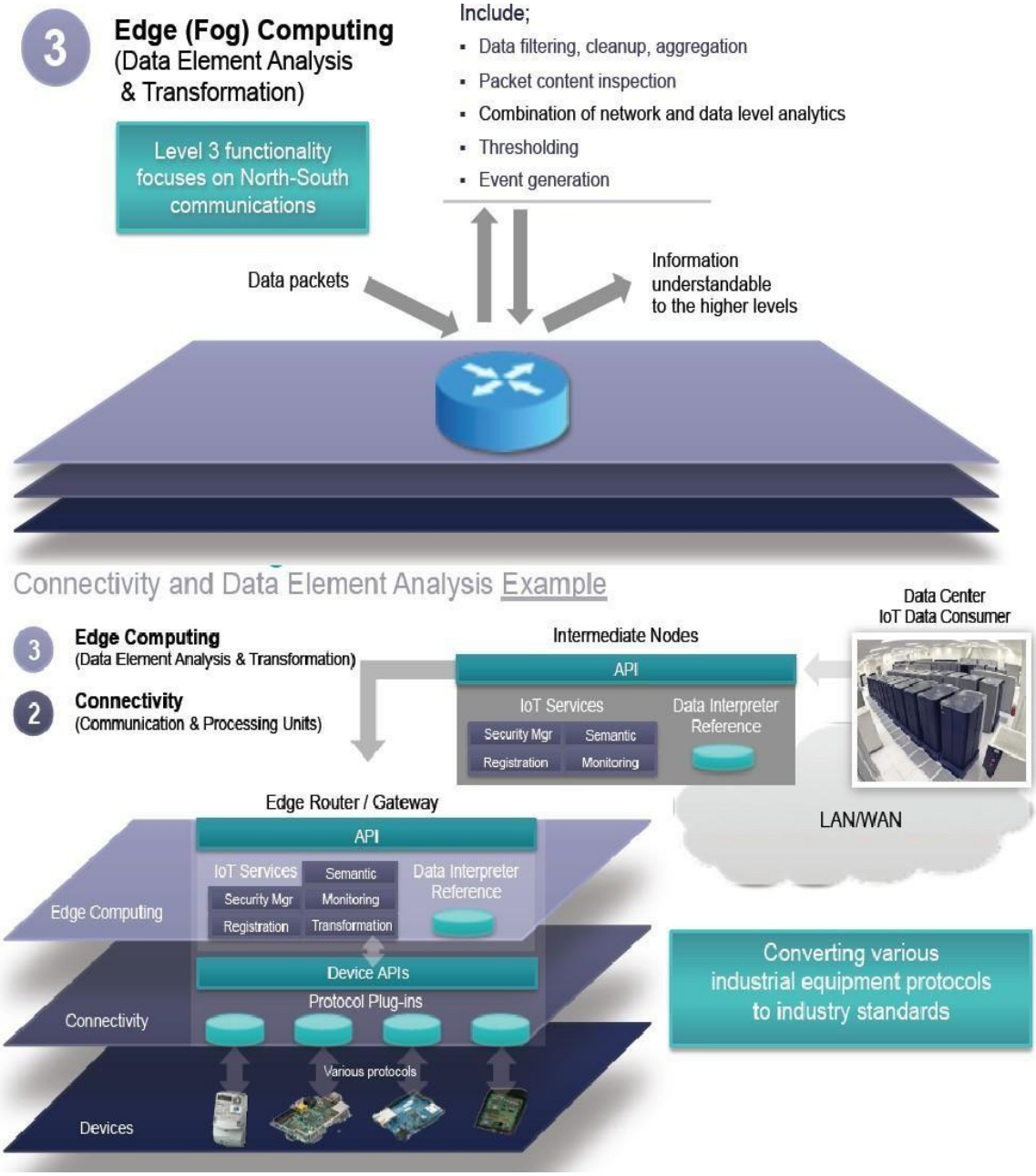
(Self Learning) Networking Anal cs

Level 3: Edge (Fog) Computing

The functions of Level 3 are driven by the need to convert network data flows into information that is suitable for storage and higher level processing at Level 4 (data accumulation). This means that Level 3 activities focus on high-volume data analysis and transformation. For example, a Level 1 sensor device might generate data samples multiple times per second, 24 hours a day, 365 days a year. A basic tenet of the IoT Reference Model is that the most intelligent system initiates information processing as early and as close to the edge of the network as possible. This is sometimes referred to as fog computing. Level 3 is where this occurs.

Given that data is usually submitted to the connectivity level (Level 2) networking equipment by devices in small units, Level 3 processing is performed on a packet-by-packet basis. This processing is limited, because there is only awareness of data uni nol "sessions‖ or ‒transactions." Level 3 processing can encompass many examples, such as:

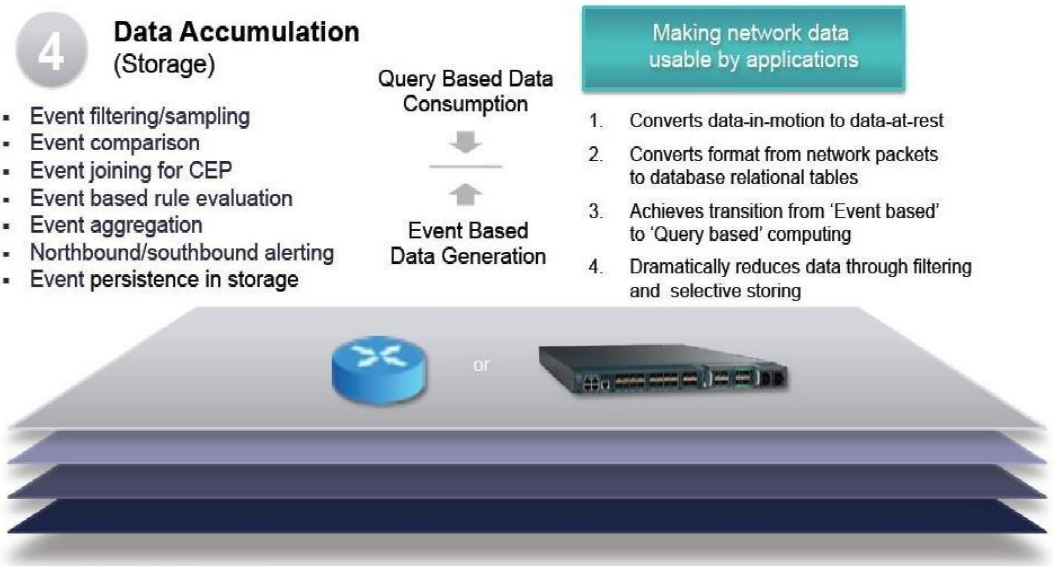
- Evaluation: Evaluating data for criteria as to whether it should be processed at a higher level
- Formatting: Reformatting data for consistent higher-level processing
- Expanding/decoding. Handling cryptic data with additional context (such as the origin)
- Distillation/reduction: Reducing and/or summarizing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


3
Edge (Fog) Computing
(Data Element Analysis
& Transformation)
Include;
Data filtering, cleanup, aggregation
Packet content inspection
Combination of network and data level analytics
Thresholding
Event generation
Level 3 functionality focuses on North-South communications
Data packets
Information understandable to the higher levels
Connectivity and Data Element Analysis Example
3
Edge Computing
(Data Element Analysis & Transformation)
2
Connectivity
(Communication & Processing Units)
Intermediate Nodes
API
IoT Services
Security Mgr
Semantic
Registration
Monitoring
Data Interpreter Reference
Data Center
IoT Data Consumer
LAN/WAN
Edge Router / Gateway
API
IoT Services
Semantic
Security Mgr
Monitoring
Registration
Transformation
Data Interpreter Reference
Edge Computing
Device APIs
Protocol Plug-ins
Connectivity
Various protocols
Devices
Converting various industrial equipment protocols to industry standards

Level 4: Data Accumulation

Networking systems are built to reliably move data. The data is "in motion." Prior to Level 4, data is moving through the network at the rate and organization determined by the devices generating the data. The model is event driven. As defined earlier, Level 1 devices do not include computing capabilities themselves. However, some computational activities could occur at Level 2, such as protocol translation or application of network security policy. Additional compute tasks can be performed at Level 3, such as packet inspection. Driving computational tasks as close to the edge of the IoT as possible, with heterogeneous systems distributed across multiple management domains represents an example of fog computing. Fog computing and fog services will be a distinguishing characteristic of the IoT.

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



Level 5: Data Abstraction

IoT systems will need to scale to a corporate—or even global—level and will require multiple storage systems to accommodate IoT device data and data from traditional enterprise ERP, HRMS, CRM, and other systems. The data abstraction functions of Level 5 are focused on rendering data and its storage in ways that enable developing simpler, performance-enhanced applications.

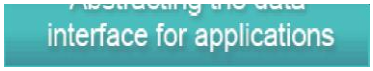
With multiple devices generating data, there are many reasons why this data may not land in the same data storage:

- There might be too much data to put in one place.
- Moving data into a database might consume too much processing power, so that retrieving it must be separated from the data generation process. This is done today with online transaction processing (OLTP) databases and data warehouses.
- Devices might be geographically separated, and processing is optimized locally.
- Levels 3 and 4 might separate "continuous streams of raw data" from "data that represents an event." Data storage for streaming data may be a big data system, such as Hadoop. Storage for event data may be a relational database management system (RDBMS) with faster query times.
- Different kinds of data processing might be required. For example, in-store processing will focus on different things than across-all-stores summary processing.

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

- Reconciling multiple data formats from different sources
- Assuring consistent semantics of data across sources
- Confirming that data is complete to the higher-level application

Data Abstraction
(Aggregation & Access)



Information Integration

1. GreBtes schemas and views of dBta in the manner that applications want
2. Combines data from multiple sources, simplifying the application

3 Filtering, selecting, projecting, and reformatting the data to serve the client applications

4. Reconciles differences in data shape, format, semantics, access protocol, and securir/

Level 6: ApDlication

Level 6 is the application level, where information **interpretation occurs.** Software at th is level interacts with Level 5 and data at rest, so it does not have to operate at network **speeds.**

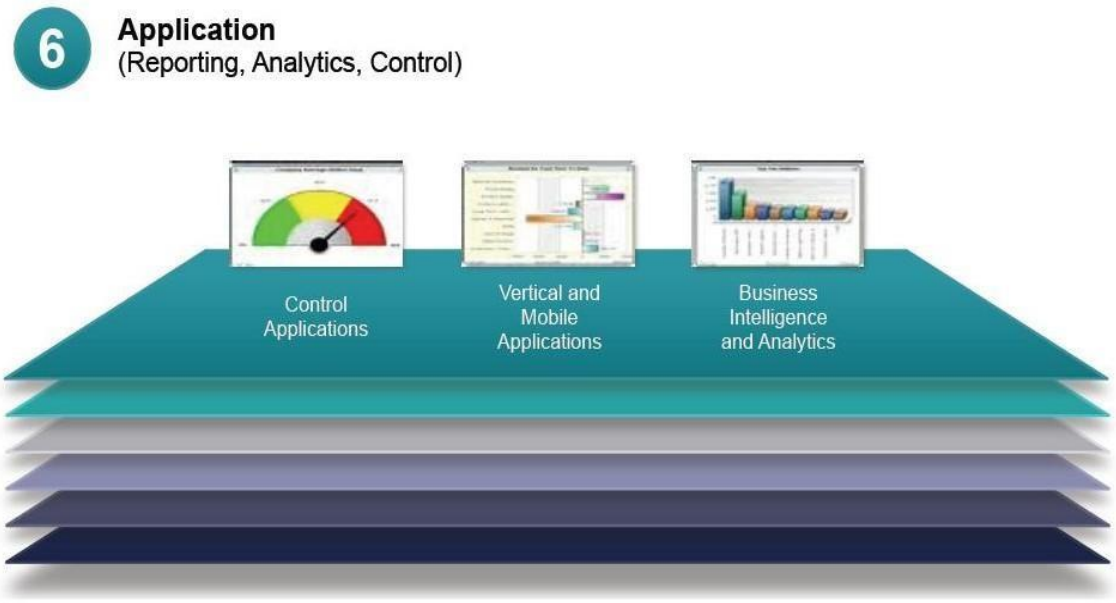
The IoT Reference Model does not strictly define an application. Applications vary based on vertical markets, the nature of device data, and **business** needs. For example, some applications will focus an monitoring device data. Some will focus on controlling devices. Some will combine device and non-device data. Monitoring and control applications represent many different application models, programming patterns, ancJ software stacks, leading to discussions of operating systems, mobility, application servers, hypervisors, multi-threading, multi-tenancy, etc. These topics are beyond the scope of the IoT Reference Model discussion. Suffice it to say that application complexity will vary widely.

Examples include:

Mission-critical business applications, such as generalized ERP or specialized industry solutions Mobile applications that handle simple interactions

- Business intelligence reports, where the application is the BI server Analytic applications that interpret data for businessdecisions

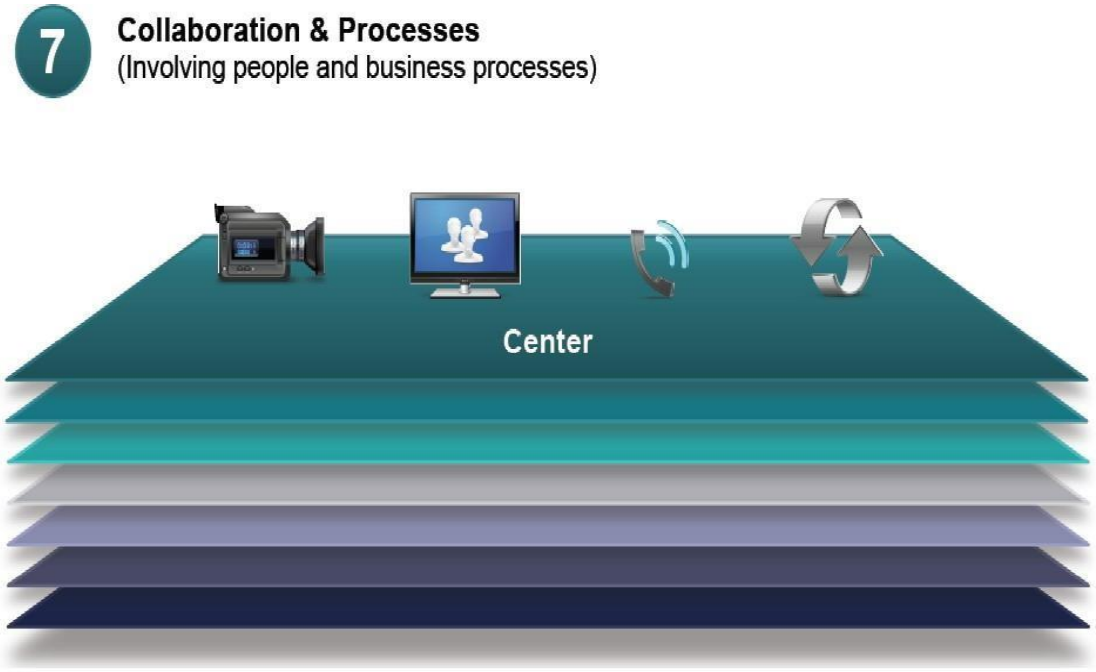
- System management/con1rol center applications that control the IoT system ibelf and don't act on the data produced by it



Level 7: Collaboration and Processes

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

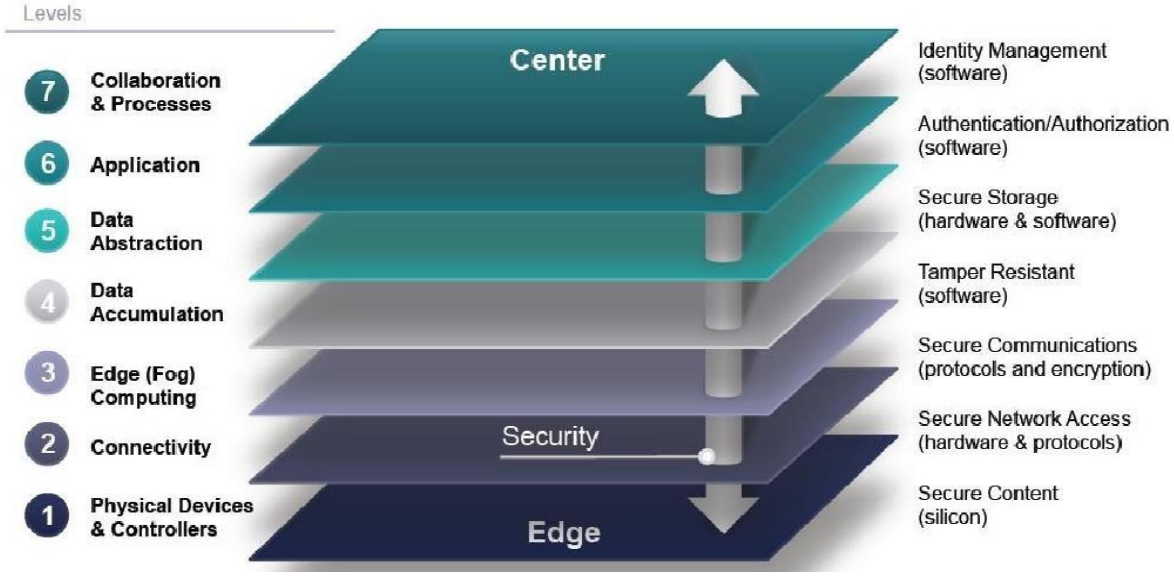
But frequently, the action needed requires more than one person. People must be able to communicate and collaborate, sometimes using the traditional Internet, to make the IoT useful. Communication and collaboration often requires multiple steps. And it usually transcends multiple applications. This is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



Security in the IoT

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

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



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

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

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

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

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

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

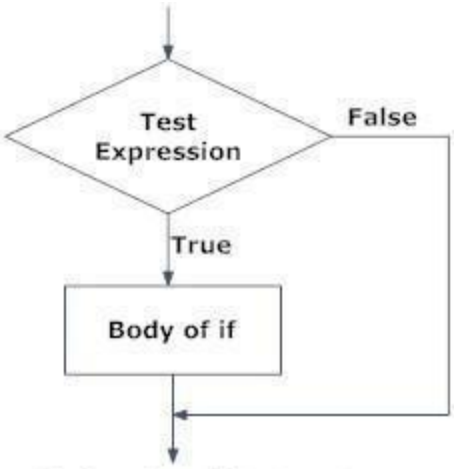


Fig: Operation of if statement

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.

Python if..else Flowchart

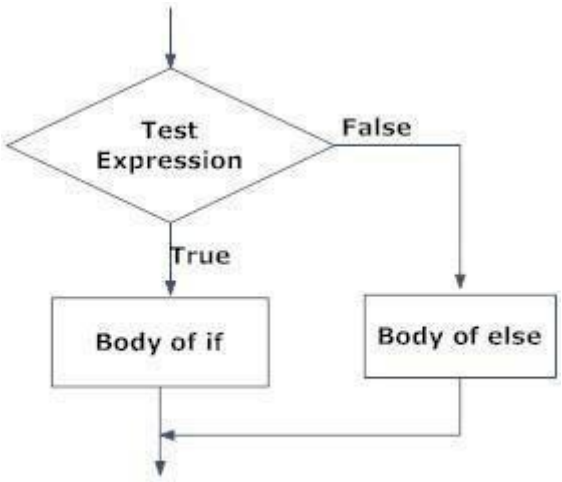


Fig: Operation of if...else statement

Example of if...else

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

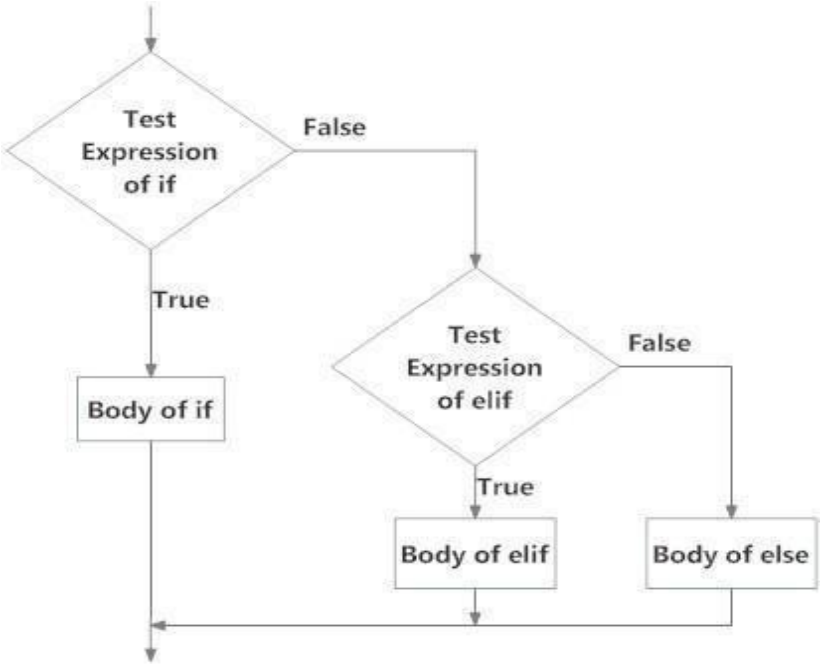


Fig: Operation of if...elif...else statement

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

The range() function

We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers). We can also define the start, stop and step size as range(start,stop,step size). step size defaults to 1 if not provided. This function does not store all the values in emory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on thego.

To force this function to output all the items, we can use the function list().

The following example will clarify this.

```
# Output: range(0, 10)
print(range(10))
# Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(list(range(10)))
```

```
# Output: [2, 3, 4, 5, 6, 7]
print(list(range(2, 8)))
# Output: [2, 5, 8, 11, 14, 17]
print(list(range(2, 20, 3)))
```

We can use the range() function in for loops to iterate through a sequence of numbers. It can be combined with the len() function to iterate though a sequence using indexing. Here is an example.

```
# Program to iterate through a list using indexing
genre = ['pop', 'rock', 'jazz']
# iterate over the list using index
for i in range(len(genre)):
        print("I like", genre[i])
```

When you run the program, the output will be:

```
I  likepop
I likerock
I likejazz
```

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 asFalse.

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"}
```

Example

Import only the person1 dictionary from the module:

```
from mymodule import person1
print (person1["age"])
```

Note: When importing using the from keyword, do not use the module name when referring to elements in the module. Example: person1["age"], not mymodule.person1["age"].

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 conceptuallyclear.

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 followingorder.

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 currentdirectory
>>> f = open("C:/Python33/README.txt") # specifying full path
```

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

Python File Modes

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

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

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

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

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 doneinternally.

```
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 readline() method to read individual lines of a file. This method reads a file till the newline, including the newlinecharacter.

```
>>> 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=SEEK_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.

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 returnit.
fileno()Return an integer number (file descriptor) of thefile.
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 readfrom.
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 ifspecified.
seek(offset,from=SEEK_SET) Change the file position to offset bytes, in reference to from (start, current,end).
seekable() Returns True if the file stream supports randomaccess.
tell() Returns the current filelocation.
truncate(size=None) Resize the file stream to size bytes. If size is not specified, resize to currentlocation.
writable() Returns True if the file stream can be writtento.
write(s) Write string s to the file and return the number of characterswritten.
writelines(lines) Write a list of lines to thefile.

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.

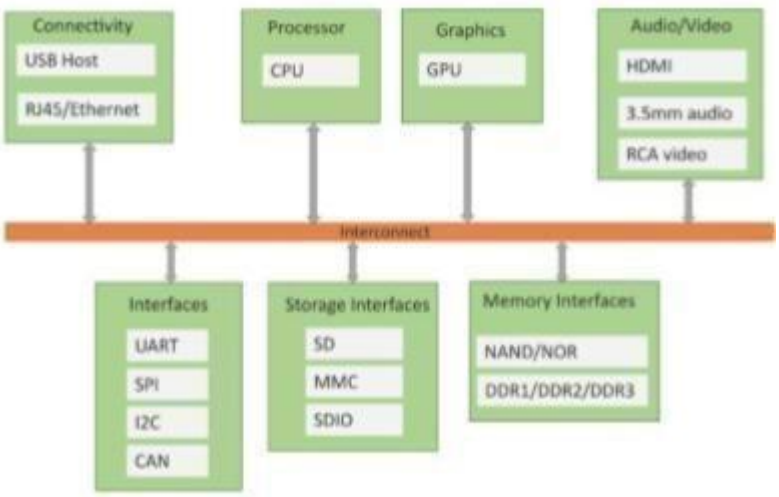
IoT Device Examples

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

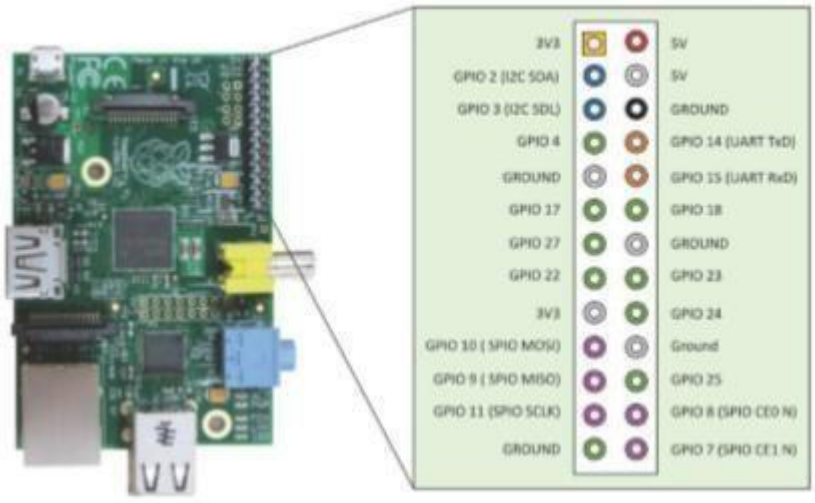
I/O pins. Since Raspberry Pi runs Linux operating system, it supports Python "out of the box". Raspberry Pi is a low-cost mini-computer with the physical size of a credit card. Raspberry Pi runs various flavors of Linux and can perform almost all tasks that a normal desktop computer can do. Raspberry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

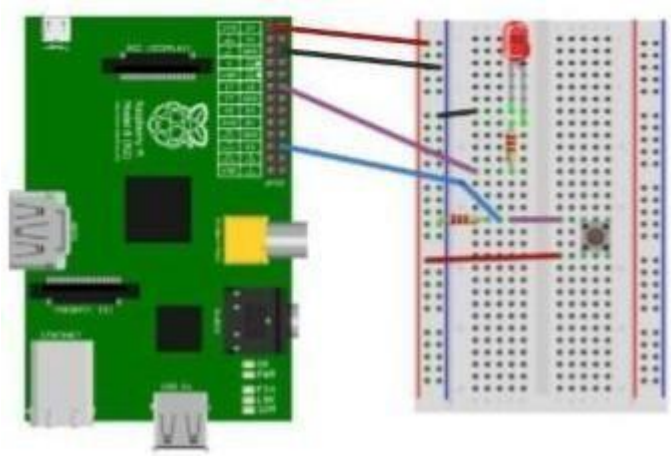
Raspberry Pi GPIO



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()
```

Other Devices

1. pcDuino
2. BeagleBoneBlack
3. Cubieboard

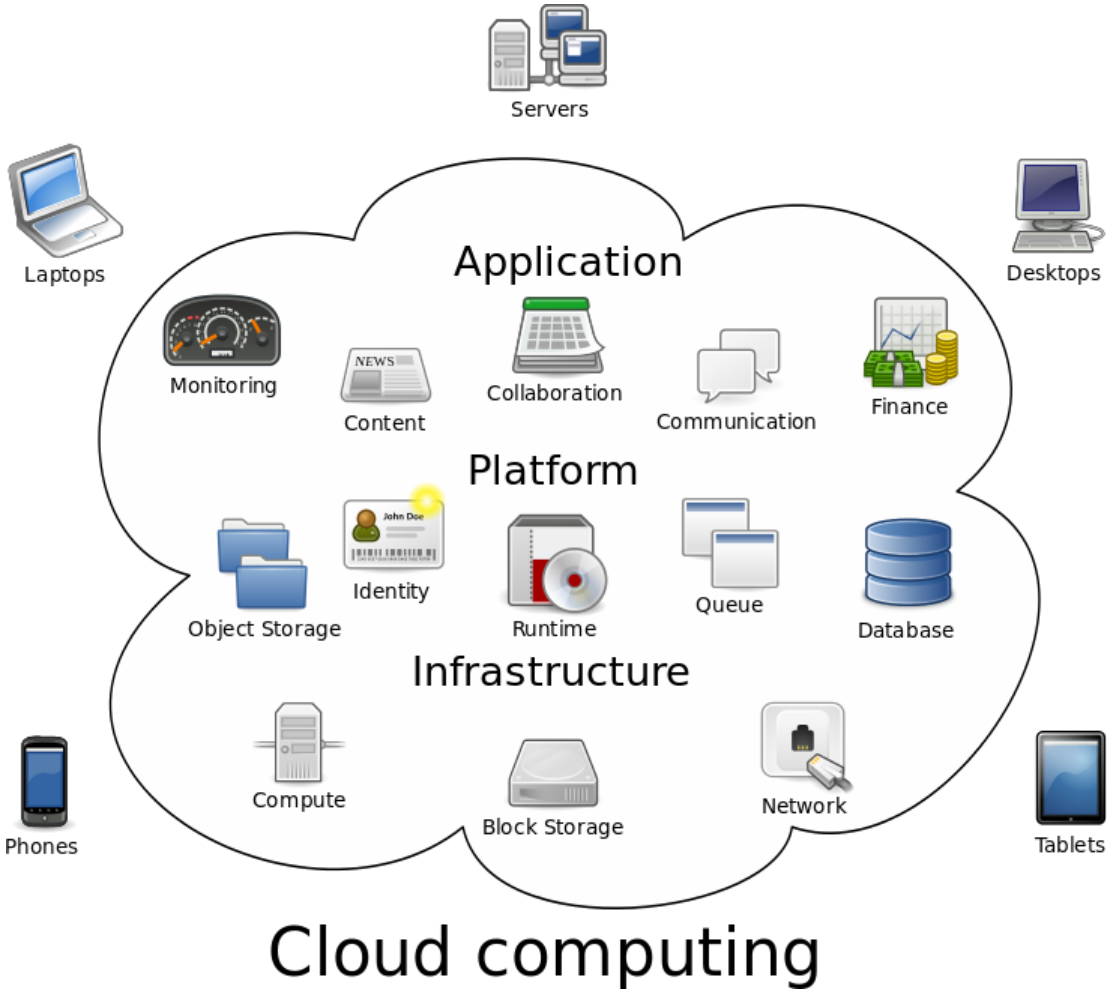
UNIT V
IoT PHYSICAL SERVERS AND CLOUD OFFERINGS

Introduction to Cloud Computing

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

Concept

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



Characteristics

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

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

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

share the "address" of the cloud-based protected and stored information with others and pool resources.

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

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

Service and Deployment

Service models

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

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

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

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

Deployment models
Deployment in cloud computing comprises four deployment models: private cloud, public cloud, community cloud and hybrid cloud.

A private cloud has infrastructure that_s provisioned for exclusive use by a single organization comprising multiple consumers such as business units. It may be owned, managed and operated by the organization, a third party or some combination of them, and it may exist on or off premises.

A public cloud is created for open use by the general public. Public cloud sells services to anyone on the internet. (Amazon Web Services is an example of a large public cloud provider.) This model is suitable for business requirements that require management of load spikes and the applications used by the business, activities that would otherwise require greater investment in infrastructure for the business. As such, public cloud also helps reduce capital expenditure and bring down operational IT costs.

A community cloud is managed and used by a particular group or organizations that have shared interests, such as specific security requirements or a common mission.
Finally, a hybrid cloud combines two or more distinct private, community or public cloud infrastructures such that they remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability. Normally, information that_s not critical is outsourced to the public cloud, while business-critical services and data are kept within the control of the organization.

CLOUD STORAGE API

A cloud storage API is an application program interface that connects a locally-based application to a cloud-based storage system, so that a user can send data to it and access and work with data stored in it. To the application, the cloud storage system is just another target device, like tape or disk-based storage. An application program interface (API) is code that allows two software programs to communicate with each other. The API defines the correct way for a developer to write a program that requests services from an operating system (OS) or other application. APIs are implemented by function calls composed of verbs and nouns. The required syntax is described in the documentation of the application being called.

How APIs work
APIs are made up of two related elements. The first is a specification that describes how information is exchanged between programs, done in the form of a request for processing and a return of the necessary data. The second is a software interface written to that specification and published in some way for use. The software that wants to access the features and capabilities of the API is said to call it, and the software that creates the API is said to publish it.

Why APIs are important for business
The web, software designed exchange information via the internet and cloud computing have all combined to increase the interest in APIs in general and services in particular. Software that was

once custom-developed for a specific purpose is now often written referencing APIs that provide broadly useful features, reducing development time and cost and mitigating the risk of errors.APIs have steadily improved software quality over the last decade, and the growing number of web services exposed through APIs by cloud providers is also encouraging the creation of cloud-specific applications, internet of things (IoT) efforts and apps to support mobile devices and users.

Three basic types of APIs
APIs take three basic forms: local, web-like and program-like.

1. **Local APIs** are the original form, from which the name came. They offer OS or middleware services to application programs. Microsoft's .NET APIs, the TAPI (Telephony API) for voice applications, and database access APIs are examples of the local APIform.

2. **Web APIs** are designed to represent widely used resources like HTML pages and are accessed using a simple HTTP protocol. Any web URL activates a web API. Web APIs are often called REST (representational state transfer) or RESTful because the publisher of REST interfaces doesn't save any data internally between requests. As such, requests from many users can be intermingled as they would be on the internet.

3. **Program APIs** are based on remote procedure call (RPC) technology that makes a remote program component appear to be local to the rest of the software. Service oriented architecture (SOA) APIs, such as Microsoft's WS-series of APIs, are program APIs.

IoT / Cloud Convergence
Internet-of-Things can benefit from the scalability, performance and pay-as-you-go nature of cloud computing infrastructures. Indeed, as IoT applications produce large volumes of data and comprise multiple computational components (e.g., data processing and analytics algorithms), their integration with cloud computing infrastructures could provide them with opportunities for cost-effective on-demand scaling. As prominent examples consider the following settings:

A Small Medium Enterprise (SME) developing an energy management IoT product, targeting smart homes and smart buildings. By streaming the data of the product (e.g., sensors and WSN data) into the cloud it can accommodate its growth needs in a scalable and cost effective fashion. As the SMEs acquires more customers and performs more deployments of its product, it is able tocollectandmanagegrowingvolumesofdatainascalableway,thustakingadvantageofa –pay-as-you-grow|model. Moreover, cloud integration allows the SME to store and process massive datasets collected from multiple (rather than a single)deployments.

A smart city can benefit from the cloud-based deployment of its IoT systems and applications. A city is likely to deploy many IoT applications, such as applications for smart energy management, smart water management, smart transport management, urban mobility of the citizensandmore.Theseapplicationscomprisemultiplesensorsanddevices,alongwith

computational components. Furthermore, they are likely to produce very large data volumes. Cloud integration enables the city to host these data and applications in a cost-effective way. Furthermore, the elasticity of the cloud can directly support expansions to these applications, but also the rapid deployment of new ones without major concerns about the provisioning of the required cloud computing resources.

A cloud computing provider offering pubic cloud services can extend them to the IoT area, through enabling third-parties to access its infrastructure in order to integrate IoT data and/or computational components operating over IoT devices. The provider can offer IoT data access and services in a pay-as-you-fashion, through enabling third-parties to access resources of its infrastructure and accordingly to charge them in a utility-based fashion.

These motivating examples illustrate the merit and need for converging IoT and cloud computing infrastructure. Despite these merits, this convergence has always been challenging mainly due to the conflicting properties of IoT and cloud infrastructures, in particular, IoT devices tend to be location specific, resource constrained, expensive (in terms of development/ deployment cost) and generally inflexible (in terms of resource access and availability). On the other hand, cloud computing resources are typically location independent and inexpensive, while at the same time providing rapid and flexibly elasticity. In order to alleviate these incompatibilities, sensors and devices are virtualized prior to integrating their data and services in the cloud, in order to enable their distribution across any cloud resources. Furthermore, service and sensor discovery functionalities are implementing on the cloud in order to enable the discovery of services and sensors that reside in different locations.

Based on these principles the IoT/cloud convergence efforts have started since over a decade i.e. since they very early days of IoT and cloud computing. Early efforts in the research community (i.e. during 2005-2009) have focused on streaming sensor and WSN data in a cloud infrastructure. Since 2007 we have also witnessed the emergence of public IoT clouds, including commercial efforts. One of the earliest efforts has been the famous Pachube.com infrastructure (used extensively for radiation detection and production of radiation maps during earthquakes in Japan). Pachube.com has evolved (following several evolutions and acquisitions of this infrastructure) to Xively.com, which is nowadays one of the most prominent public IoT clouds. Nevertheless, there are tens of other public IoT clouds as well, such as ThingsWorx, ThingsSpeak,Sensor-Cloud, Realtime.io and more. The list is certainly non-exhaustive. These public IoT clouds offer commercial pay-as-you-go access to end-users wishing to deploying IoT applications on the cloud. Most of them come with developer friendly tools, which enable the development of cloud applications, thus acting like a PaaS for IoT in the cloud. Similarly to cloud computing infrastructures, IoT/cloud infrastructures and related services can be classified to the following models:

1. **Infrastructure-as-a-Service (IaaS) IoT/Clouds**: These services provide the means for accessing sensors and actuator in the cloud. The associated business model involves the IoT/Cloud provide to act either as data or sensor provider. IaaS services for IoT provide access control to resources as a prerequisite for the offering of related pay-as-you-go services.

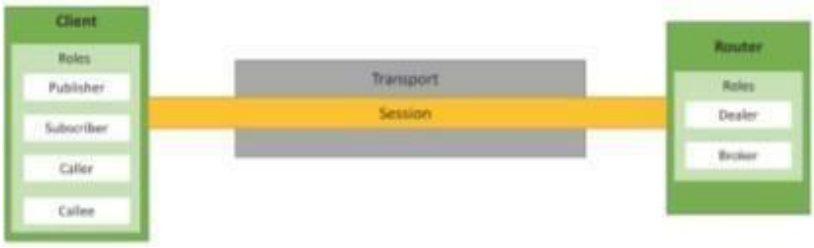
2. **Platform-as-a-Service (PaaS) IoT/Clouds**: This is the most widespread model for IoT/cloud services, given that it is the model provided by all public IoT/cloud infrastructures outlined above. As already illustrate most public IoT clouds come with a range of tools and related environments for applications development and deployment in a cloud environment. A main characteristic of PaaS IoT services is that they provide access to data, not to hardware. This is a clear differentiator comparing to IaaS.

3. **Software-as-a-Service (SaaS) IoT/Clouds**: SaaS IoT services are the ones enabling their uses to access complete IoT-based software applications through the cloud, on-demand and in a pay-as-you-go fashion. As soon as sensors and IoT devices are not visible, SaaS IoT applications resemble very much conventional cloud-based SaaS applications. There are however cases where the IoT dimension is strong and evident, such as applications involving selection of sensors and combination of data from the selected sensors in an integrated applications. Several of these applications are commonly called Sensing-as-a-Service, given that they provide on-demand access to the services of multiple sensors. Note that SaaS IoT applications are typically built over a PaaS infrastructure and enable utility based business models involving IoT software and services.

These definitions and examples provide an overview of IoT and cloud convergence and why it is important and useful. More and more IoT applications are nowadays integrated with the cloud in order to benefit from its performance, business agility and pay-as-you-go characteristics. In following chapters of the tutorial, we will present how to maximize the benefits of the cloud for IoT, through ensuring semantic interoperability of IoT data and services in the cloud, thus enabling advanced data analytics applications, but also integration of a wide range of vertical (silo) IoT applications that are nowadays available in areas such as smart energy, smart transport and smart cities. We will also illustrate the benefits of IoT/cloud integration for specific areas and segments of IoT, such as IoT-based wearable computing.

WAMP for IoT

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



WAMP

1. **Transport:** Transport is channel that connects two peers.
2. **Session:** Session is a conversation between two peers that runs over a transport.
3. **Client:** Clients are peers that can have one or more roles. In publish-subscribe model client can have following roles:
    a) Publisher: Publisher publishes events (including payload) to the topic maintained by the broker.
    b) Subscriber: Subscriber subscribes to the topics and receives the events including the payload.

In RPC model client can have following roles: –

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

In RPC model Router has the role of a Broker: –

1. Dealer: Dealer acts a router and routes RPC calls from the Caller to the Callee and routes results from Callee toCaller.
2. Application Code: Application code runs on the Clients (Publisher, Subscriber, Callee or Caller).

Amazon EC2 – Python Example

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

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

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

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

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

Amazon AutoScaling – Python Example

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

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

print "Creating launch configuration"

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

print "Creating auto-scaling group"

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

Amazon AutoScaling – Python Example

```
#Creating auto-scaling policies

scale_up_policy = ScalingPolicy(name='scale_up',

        adjustment_type='ChangeInCapacity',

        as_name='My-Group',

        scaling_adjustment=1,

        cooldown=180)

scale_down_policy =ScalingPolicy(name='scale_down',

        adjustment_type='ChangeInCapacity',

        as_name='My-Group', scaling_adjustment=-1,

        cooldown=180)

conn.create_scaling_policy(scale_up_policy)

conn.create_scaling_policy(scale_down_policy)
```

AutoScaling Policies:

1. After creating an AutoScaling group, the policies for scaling up and scaling down are defined.
2. In this example, a scale up policy with adjustment type ChangeInCapacity and scaling_ad justment = 1 isdefined.
3. Similarly a scale down policy with adjustment type ChangeInCapacity and scaling_ad justment = -1 isdefined.

CloudWatch Alarms

```
#Connecting to CloudWatch

cloudwatch = boto.ec2.cloudwatch.connect_to_region(REGION,

                    aws_access_key_id=ACCESS_KEY,
```

```
                                aws_secret_access_key=SECRET_KEY)
alarm_dimensions = {"AutoScalingGroupName": 'My-Group'}
#Creating scale-up alarm
scale_up_alarm = MetricAlarm(
        name='scale_up_on_cpu', namespace='AWS/EC2',
        metric='CPUUtilization', statistic='Average',
        comparison='>', threshold='70',
        period='60', evaluation_periods=2,
alarm_actions=[scale_up_policy.policy_arn],
dimensions=alarm_dimensions)
cloudwatch.create_alarm(scale_up_alarm)
#Creating scale-down alarm
scale_down_alarm =MetricAlarm(
    name='scale_down_on_cpu',namespace='AWS/EC2',
    metric='CPUUtilization', statistic='Average',
    comparison='<',threshold='40',
    period='60', evaluation_periods=2,
    alarm_actions=[scale_down_policy.policy_arn],
    dimensions=alarm_dimensions) cloudwatch.create_alarm(scale_down_alarm)
```

1. With the scaling policies defined, the next step is to create Amazon CloudWatch alarms that trigger thesepolicies.
2. The scale up alarm is defined using the CPUUtilization metric with the Average statistic and threshold greater 70% for a period of 60 sec. The scale up policy created previously is associated with this alarm. This alarm is triggered when the average CPU utilization of the instances in the group becomes greater than 70% for more than 60seconds.
3. The scale down alarm is defined in a similar manner with a threshold less than50%.

Python for MapReduce

#Inverted Index Mapper in Python

#!/usr/bin/env python import sys for line in sys.stdin: doc_id, content = line.split(__) words = content.split() for word in words: print _%s%s_ % (word, doc_id)

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

Python for MapReduce

#Inverted Index Reducer in Python

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

for line in sys.stdin: # remove leading and trailing whitespace line = line.strip() # parse the input we got from mapper.py word, doc_id = line.split(__) if current_word == word: current_docids.append(doc_id) else: if current_word: print _%s%s_ % (current_word, current_docids) current_docids = [] current_docids.append(doc_id) current_word = word

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

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.

Python Web Application Framework - Django

Django is an open source web application framework for developing web applications in Python. A web application framework in general is a collection of solutions, packages and best practices that allows development of web applications and dynamic websites. Django is based on the Model-Template-View architecture and provides a separation of the data model from the business rules and the user interface. Django provides a unified API to a database backend. Thus web applications built with Django can work with different databases without requiring any code changes. With this flexibility in web application design combined with the powerful capabilities of the Python language and the Python ecosystem,

Django is best suited for cloud applications. Django consists of an object-relational mapper, a web templating system and a regular-expression based URL dispatcher.

Django Architecture

Django is Model-Template-View (MTV) framework.

1. Model: The model acts as a definition of some stored data and handles the interactions with the database. In a web application, the data can be stored in a relational database, non-relational database, an XML file, etc. A Django model is a Python class that outlines the variables and methods for a particular type ofdata.
2. Template: In a typical Django web application, the template is simply an HTML page with a few extra placeholders. Django_s template language can be used to create various forms of text files (XML, email, CSS, Javascript, CSV,etc.)
3. View :The view ties the model to the template. The view is where you write the code that actually generates the web pages. View determines what data is to be displayed, retrieves the data from the database and passes the data to the template.

Case studies illustrating IoT design

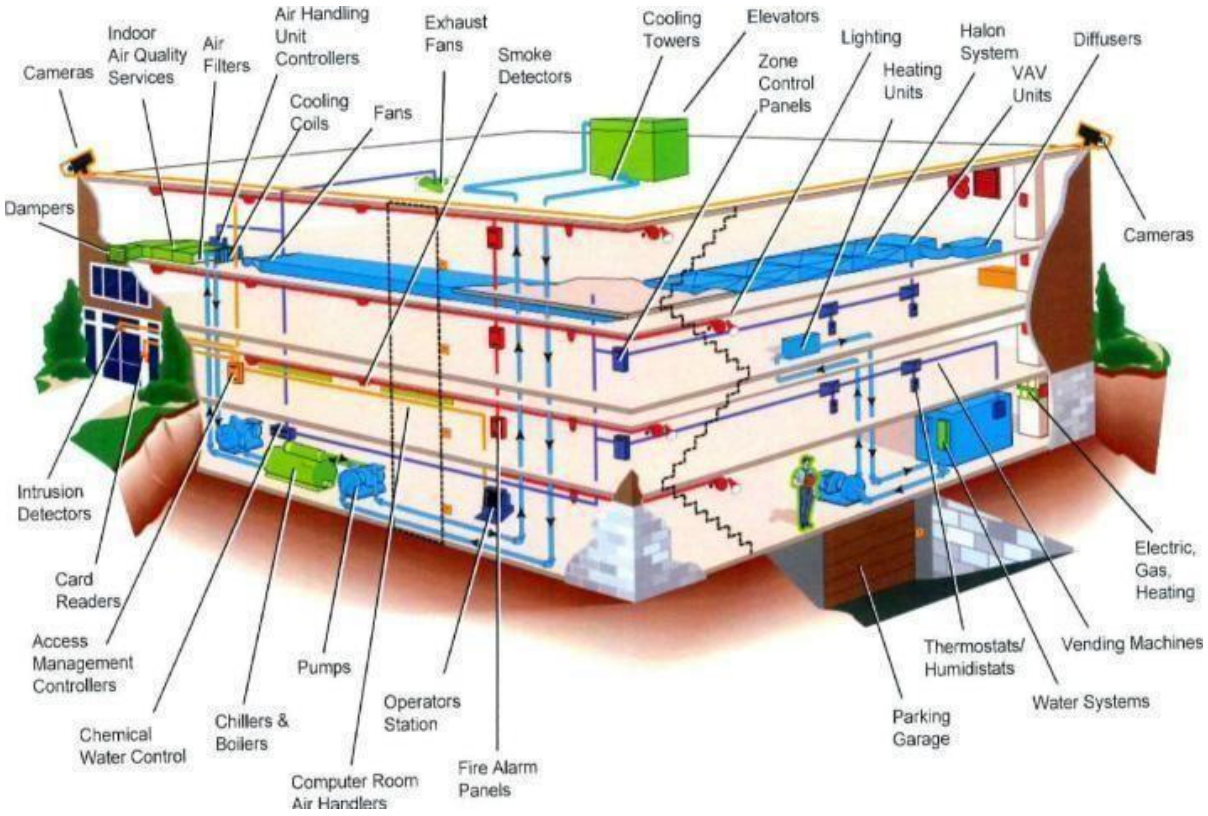
Case Study in IoT: Home Automation

An IoT software-based approach on the field of Home Automation. Common use-cases include measuring home conditions, controlling home appliances and controlling home access through RFID cards as an example and windows through servo locks. However, the main focus of this paper is to maximize the security of homes through IoT. More specifically, monitoring and controlling servo door locks, door sensors, surveillance cameras, surveillance car and smoke detectors, which help ensuring and maximizing safety and security of homes.

A user has the following features through a mobile application in which he/she:

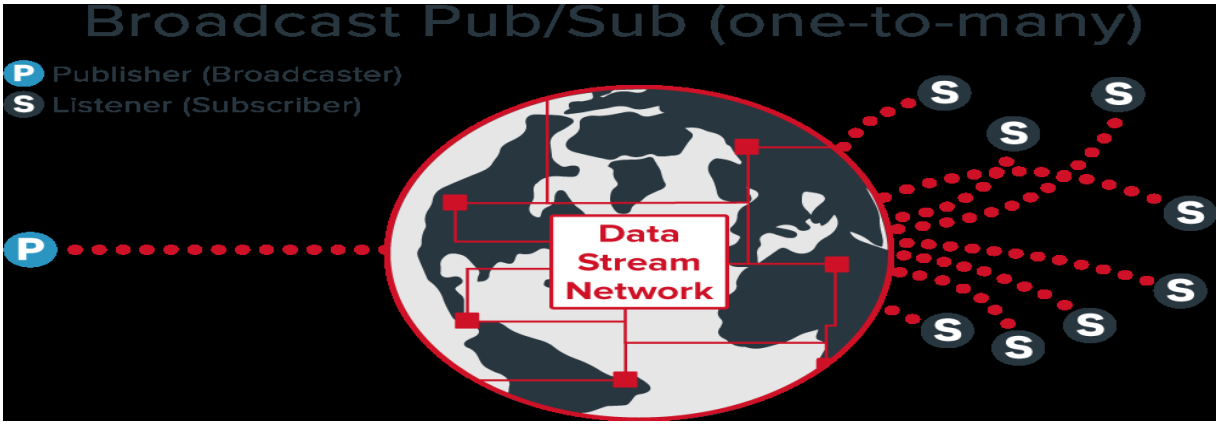
1. can turn on or o_ LED lights and monitor the state of the LED.
2. can lock and unlock doors through servo motors and monitor if the doors are locked or unlocked.
3. can monitor if the doors are closed or opened through IR sensors.
4. is notified through email if the door is left open for too long.
5. is notified of who entered through the door as the camera captures the face image and send it to him/her via email.
6. is notified through email if the _re detector detect ssmoke.
7. is able to control the surveillance car from anywhere to monitor his/herhome.

As the field of Home Automation through IoT is a wide application in a very wide and challenging field due to the reasons mentioned in the previous paragraphs, I chose to work on that field as part of this thesis, specifically in maintaining and ensuring security and safety inside home.

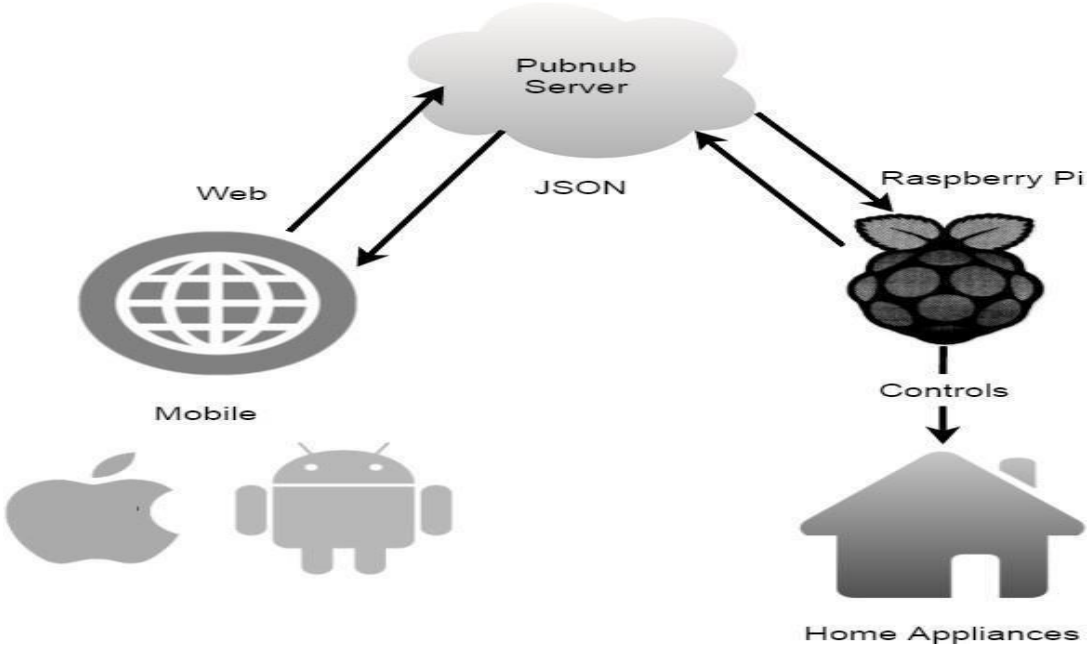


IoT aims in creating a network between objects embedded with sensors, that can store, analyze, communicate and exchange data together over the internet. This leads to efficient industry, manufacturing, efficient energy management, resource management, accurate health care, smarter business decisions based on analyzed data, safer driving through smart cars that are able to communicate together, smart home automation and countless more applications.

The system designed for the home automation project presented in this paper needs a control unit, a computer, to be able to control the different electrical devices connected to it. Raspberry Pi, is a credit-card tiny computer, that can be plugged to a monitor, uses standard keyboard and mouse, that enables people of different ages learn how to program.



Illustrates the publish/subscribe model provided by PubNub



Illustrates the system architecture used in this home automation project.

To simplify the publish/subscribe model along with the system architecture used in this Home Automation project, here is the explanation of the steps of constructing it: Different sensors,

cameras and servo motors were connected to the Raspberry Pi. It was programmed to collect and publish the data, in the form of JSON string, acquired from these devices to PubNub. Data is published from the Raspberry Pi by providing it with the "publish key" and the "channel name". The data is sent to the channel provided by PubNub servers, and forwarded by PubNub to the subscribers of this channel.

The subscriber in this scenario, of a user acquiring data and readings by the sensors and monitoring devices, is the web/mobile application. The "subscription key" and "channel name" is embedded in the web/mobile application's code. Allowing it to receive messages forwarded by PubNub. On the other hand, in a scenario where the user wants to send a command to home appliances, controlling the LED lights for example, the web/mobile application is the publisher provided by the "publish key" and the "channel name".The commend is sent in the form of JSON string to PubNub servers, while the "subscription key" and "channel name" is embedded in the Raspberry Pi code. This allows the Raspberry Pi to receive any published strings on the channel it is subscribed to. Upon receiving the JSON string, the Raspberry Pi take the action specified by that string. This allows full control and monitoring of all devices connected to the Raspberry Pi by theuser.

Case Study in IoT: Smart Cities

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

An illustration based smart city

IoT based smart city taxonomy

Communication Protocols	Service providers	N'e.twork.: :types	A!c'tivities of Stando'r.d bodies	Offered services	Requi!rements
ZigBee/ Bluetooth	Telefoni ca	WLANs	IETF	transportation	Lowrost
Wi-Fi	; SX telecom :	BANs	3GPP	e-healthcare	Low power consumption
6LoWPAN,I LoRaWAN	Nlokia, Ericsson	WPANS	ETSI	Safety and security	. Tratfic mod eiing
IEEE 802.11p	- Vodafone	WANs	1 fiEE	Smart parking	Interoperability /connectivity
SigFox, WiMAX	NTT Docomo	MANs	OMA	tVa5te ment	Security and privacy
GSM/GPRS, UMTS	Orange, Cisco	Mobile networks	oneM2M, FIWARE	Smart energy metering	
LTE/LTE-A	Telenor, AT &T				
Upcoming 5G					

A representation of IoT-based smart city taxonomy

IOT BASED SMART CITY TAXONOMY

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

Communication Protocols

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

Service Providers

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

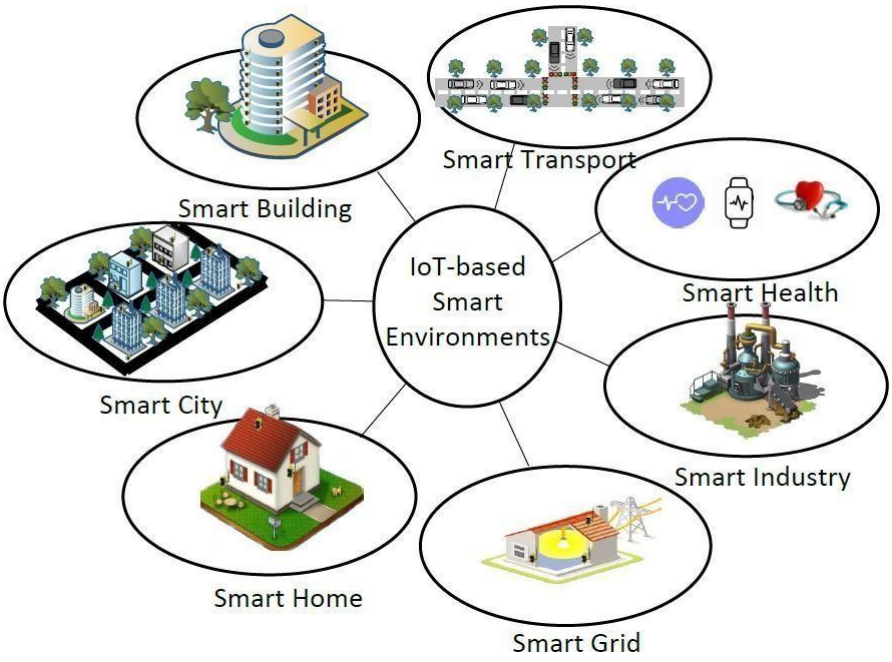
Network Types

IoT based smart city applications rely on numerous network topologies to accomplish a fully autonomous environment. The capillary IoT networks offer services over a short range. Examples include wireless local area networks (WLANs), BANs and wireless personal area networks (WPANs). The application areas include indoor e-healthcare services, home automation, street lighting. On the other hand, applications such as ITS, mobile e-healthcare and waste management use wide area networks (WANs), metropolitan area networks (MANs),and

mobile communication networks. The above networks pose distinct features in terms of data, size, coverage, latency requirements, and capacity.

Case Study in IoT: Smart Environment

The rapid advancements in communication technologies and the explosive growth of Internet of Things (IoT) have enabled the physical world to invisibly interweave with actuators, sensors, and other computational elements while maintaining continuous network connectivity. The continuously connected physical world with computational elements forms a smart environment. A smart environment aims to support and enhance the abilities of its dwellers in executing their tasks, such as navigating through unfamiliar space and moving heavy objects for the elderly, to name a few. Researchers have conducted a number of efforts to use IoT to facilitate our lives and to investigate the effect of IoTbased smart environments on human life. This paper surveys the state-of-the-art research efforts to enable the IoT-based smart environments. We categorize and classify the literature by devising a taxonomy based on communication enablers, network types, technologies, local area wireless standards, objectives, and characteristics. Moreover, the paper highlights the unprecedented opportunities brought about by IoT-based smart environments and their effect on human life. Some reported case studies from different enterprises are also presented. Finally, we discuss open research challenges for enabling IoT-based smart environments.

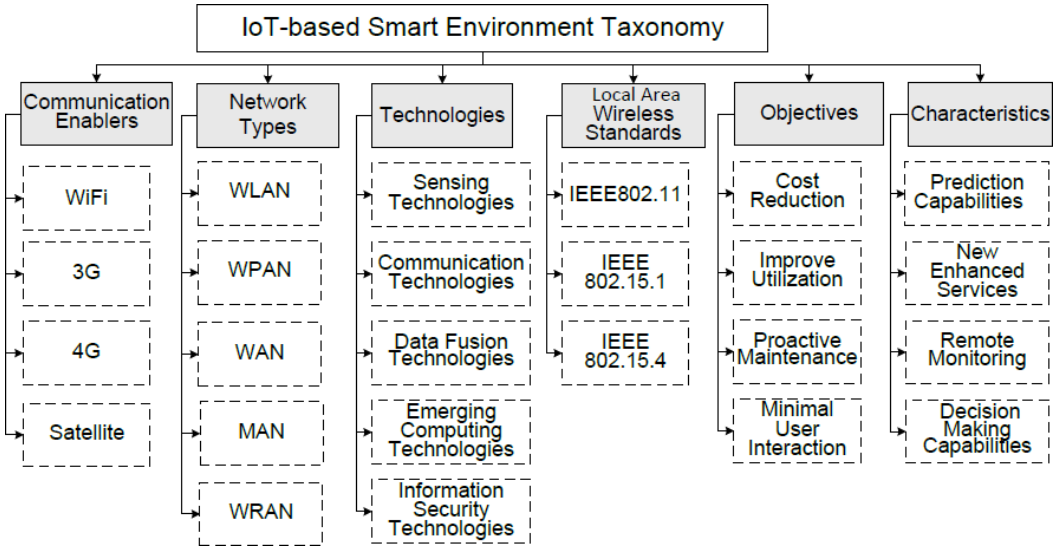


IoT-based Smart Environments

Immense developments and increasing miniaturization of computer technology have enabled tiny sensors and processors to be integrated into everyday objects. This advancement is further supported by tremendous developments in areas such as portable appliances and devices, pervasive computing, wireless sensor networking, wireless mobile communications, machine learning-based decision making, IPv6 support, human computer interfaces, and agent technologies to make the dream of smart environment a reality. A smart environment is a connected small world where sensor-enabled connected devices work collaboratively to make the lives of dwellers comfortable. The term smart refers to the ability to autonomouslyobtain and applies knowledge; and the term environment refers to the surroundings. Therefore, a smart environment is one that is capable of obtaining knowledge and applying it to adapt according to its inhabitants_ needs to ameliorate their experience of thatenvironment.

The functional capabilities of smart objects are further enhanced by interconnecting them with other objects using different wireless technologies. In this context, IPv6 plays a vital role because of several features, including better security mechanisms, scalability in case of billion of connected devices, and the elimination of NAT barriers1. This concept of connecting smart objects with the Internet was first coined by Kevin Ashton as –Internet of Things (IoT).

Nowadays, IoT is receiving attention in a number of fields such as healthcare, transport, and industry, among others. Several research efforts have been conducted to integrate IoT with smart environments. The integration of IoT with a smart environment extends the capabilities of smart objects by enabling the user to monitor the environment from remote sites. IoT can be integrated with different smart environments based on the application requirements. The work on IoT-based smart environments can generally be classified into the following areas: a) smart cities, b) smart homes, c) smart grid, d) smart buildings, e) smart transportation, f) smart health, and g) smart industry.illustrates the IoT-based smartenvironments.



The taxonomy of the IoT based smart environment. The devised taxonomy is based on the following parameters: communication enablers, network types, technologies, wireless standards, objectives, and characteristics

Communication Enablers

Communication enablers refer to wireless technologies used to communicate across the Internet. The key wireless Internet technologies are WiFi, 3G, 4G, and satellite. WiFi is mainly used in smart homes, smart cities, smart transportation, smart industries, and smart building environments; whereas, 3G and 4G are mainly used in smart cities and smart grid environments. Satellites are used in smart transportation, smart cities, and smart grid environments. Table presents the comparative summary of the communication technologies used in IoT based smart environments.

Network Types

IoT-based smart environments rely on different types of networks to perform the collaborative tasks for making the lives of inhabitants more comfortable. The main networks are wireless local area networks (WLANs), wireless personal area networks (WPANs), wide area networks (WANs), metropolitan area networks (MANs), and wireless regional area networks (WRANs). These networks have different characteristics in terms of size, data transfer, and supported reach ability.

Technologies

IoT-based smart environments leverage various technologies to form a comfortable and suitable ecosystem. These technologies are include sensing, communication, data fusion, emerging computing, and information security. Sensing technologies are commonly used to acquire data from various locations and transmit it using communication technologies to a central location. The emerging computing technologies, such as cloud computing and fog computing, deployed in the central location, leverage the data fusion technologies for integrating the data coming from heterogeneous resources. In addition, smart environments also use information security technologies to ensure data integrity and user privacy.

Local Area Wireless Standards

The commonly used local area wireless standards in IoT-based smart environments are IEEE 802.11, IEEE 802.15.1, and IEEE 802.15.4. These standard technologies are used inside the smart environment to transfer the collected data among different devices. IEEE 802.11 is used in smart homes, smart buildings, and smart cities. IEEE 802.15.1 and IEEE 802.15.4 have relatively shorter coverage than IEEE 802.11 and are used mainly in sensors and other objects deployed in the smartenvironments.