**IV Year B.Tech.ECE II - Semester**

**INTERNET OF THINGS**

**Course description and objectives:**

Students will be explored to the interconnection and integration of the physical world and the cyber space. They are also able to design & develop IOT Devices.

**Course Outcomes:**

1. Able to understand the application areas of IOT
2. Able to realize the revolution of Internet in Mobile Devices, Cloud & Sensor Networks
3. Able to understand building blocks of Internet of Things and characteristics.

**Unit I**

Introduction to Internet of Things

Definition : A dynamic global network infrastructure with self-configuring capabilities based on standard and inter-operable communication protocols where physical and virtual "things" have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network, often communicate data associated with users and their environments.

Characteristics of IoT

1. Dynamic and self adapting
2. Self configuring
3. Inter-operable communication protocols
4. Unique identity
5. Integrated into information network

Physical Design of IoT

Things of IoT

A thing, in the context of the Internet of things ([IoT](http://whatis.techtarget.com/definition/Internet-of-Things)), is an entity or physical object that has a [unique identifier](http://whatis.techtarget.com/definition/unique-identifier-UID), an [embedded system](http://searchenterpriselinux.techtarget.com/definition/embedded-system) and the ability to transfer data over a network.

IOT Protocols

1. Link layer

* **802.3 Ethernet**

Ethernet, defined under IEEE 802.3, is one of today's most widely used data communications standards, and it finds its major use in Local Area Network (LAN) applications. With versions including 10Base-T, 100Base-T and now Gigabit Ethernet, it offers a wide variety of choices of speeds and capability. Ethernet is also cheap and easy to install. Additionally Ethernet, IEEE 802.3 offers a considerable degree of flexibility in terms of the network topologies that are allowed. Furthermore as it is in widespread use in LANs, it has been developed into a robust system that meets the needs to wide number of networking requirements.

* **802.11 WiFi**

IEEE 802.11 is a set of [media access control](https://en.wikipedia.org/wiki/Media_access_control) (MAC) and [physical layer](https://en.wikipedia.org/wiki/Physical_layer) (PHY) specifications for implementing [wireless local area network](https://en.wikipedia.org/wiki/Wireless_LAN) (WLAN) computer communication in the 900 MHz and 2.4, [3.6](https://en.wikipedia.org/wiki/IEEE_802.11y-2008), 5, and [60 GHz](https://en.wikipedia.org/wiki/IEEE_802.11ad) frequency bands. They are created and maintained by the [Institute of Electrical and Electronics Engineers](https://en.wikipedia.org/wiki/Institute_of_Electrical_and_Electronics_Engineers) (IEEE) [LAN](https://en.wikipedia.org/wiki/Local_area_network)/[MAN](https://en.wikipedia.org/wiki/Metropolitan_area_network) Standards Committee ([IEEE 802](https://en.wikipedia.org/wiki/IEEE_802)). The base version of the standard was released in 1997, and has had subsequent amendments. The standard and amendments provide the basis for wireless network products using the [Wi-Fi](https://en.wikipedia.org/wiki/Wi-Fi) brand. While each amendment is officially revoked when it is incorporated in the latest version of the standard, the corporate world tends to market to the revisions because they concisely denote capabilities of their products. As a result, in the market place, each revision tends to become its own standard.

* **802.16 WiMax**

IEEE 802.16 is a series of [wireless broadband](https://en.wikipedia.org/wiki/Wireless_broadband) standards written by the [Institute of Electrical and Electronics Engineers](https://en.wikipedia.org/wiki/Institute_of_Electrical_and_Electronics_Engineers) (IEEE). The IEEE Standards Board established a working group in 1999 to develop standards for broadband for [wireless](https://en.wikipedia.org/wiki/Wireless)[metropolitan area networks](https://en.wikipedia.org/wiki/Metropolitan_area_network). The Workgroup is a unit of the [IEEE 802](https://en.wikipedia.org/wiki/IEEE_802)[local area network](https://en.wikipedia.org/wiki/Local_area_network) and [metropolitan area network](https://en.wikipedia.org/wiki/Metropolitan_area_network) standards committee.

Although the 802.16 family of standards is officially called WirelessMAN in IEEE, it has been commercialized under the name "[WiMAX](https://en.wikipedia.org/wiki/WiMAX" \o "WiMAX)" (from "Worldwide Interoperability for Microwave Access") by the WiMAX Forum industry alliance. The Forum promotes and certifies compatibility and interoperability of products based on the IEEE 802.16 standards.

* **802.15.4-LR-WPAN**
* IEEE 802.15.4 is a standard created and maintained by consultants which specifies the [physical layer](https://en.wikipedia.org/wiki/Physical_layer) and [media access control](https://en.wikipedia.org/wiki/Media_Access_Control) for low-rate wireless [personal area networks](https://en.wikipedia.org/wiki/Personal_area_network) (LR-WPANs). It is maintained by the [IEEE 802.15](https://en.wikipedia.org/wiki/IEEE_802.15) working group, which has defined it in 2003. It is the basis for the [ZigBee](https://en.wikipedia.org/wiki/ZigBee), [ISA100.11a](https://en.wikipedia.org/wiki/ISA100.11a), [WirelessHART](https://en.wikipedia.org/wiki/WirelessHART), [MiWi](https://en.wikipedia.org/wiki/MiWi), and [Thread](https://en.wikipedia.org/wiki/Thread_(network_protocol)) specifications, each of which further extends the standard by developing the upper [layers](https://en.wikipedia.org/wiki/Protocol_stack) which are not defined in IEEE 802.15.4. Alternatively, it can be used with [6LoWPAN](https://en.wikipedia.org/wiki/6LoWPAN) as Network Adaptation Layer and standard Internet protocols and/or IETF RFCs defining the upper layers with proper granularity to build a wireless embedded Internet.
* **2G/3G/4G – Mobile Communication**

2G (or 2-G) is short for second-generation [wireless](https://en.wikipedia.org/wiki/Wireless)[telephone](https://en.wikipedia.org/wiki/Telephone)[technology](https://en.wikipedia.org/wiki/Technology). Second generation 2G cellular telecom networks were commercially launched on the [GSM](https://en.wikipedia.org/wiki/GSM) standard in [Finland](https://en.wikipedia.org/wiki/Finland) by [Radiolinja](https://en.wikipedia.org/wiki/Radiolinja) (now part of [Elisa Oyj](https://en.wikipedia.org/wiki/Elisa_Oyj)) in 1991.[[1]](https://en.wikipedia.org/wiki/2G#cite_note-Radiolinja.27s_History-1) Three primary benefits of 2G networks over their predecessors were that phone conversations were digitally encrypted; 2G systems were significantly more efficient on the spectrum allowing for far greater mobile phone penetration levels; and 2G introduced data services for mobile, starting with [SMS](https://en.wikipedia.org/wiki/SMS) text messages. 2G technologies enabled the various mobile phone networks to provide the services such as text messages, picture messages and MMS (multi media messages). All text messages sent over 2G are digitally encrypted, allowing for the transfer of data in such a way that only the intended receiver can receive and read it.

3G, short form of third generation, is the third generation of mobile telecommunications technology. This is based on a set of standards used for mobile devices and mobile telecommunications use services and networks that comply with the International Mobile Telecommunications-2000 (IMT-2000) specifications by the International Telecommunication Union. 3G finds application in wireless voice telephony, [mobile Internet](https://en.wikipedia.org/wiki/Mobile_Internet) access, fixed wireless Internet access, video calls and mobile TV.

3G telecommunication networks support services that provide an information transfer rate of at least 200 [kbit/s](https://en.wikipedia.org/wiki/Kilobit_per_second). Later 3G releases, often denoted [3.5G](https://en.wikipedia.org/wiki/3.5G) and [3.75G](https://en.wikipedia.org/wiki/3.75G), also provide [mobile broadband](https://en.wikipedia.org/wiki/Mobile_broadband) access of several [Mbit/s](https://en.wikipedia.org/wiki/Mbps) to [smartphones](https://en.wikipedia.org/wiki/Smartphone) and [mobile modems](https://en.wikipedia.org/wiki/Mobile_modem) in laptop computers. This ensures it can be applied to wireless voice telephony, mobile Internet access, fixed wireless Internet access, video calls and mobile TV technologies.

4G, short for fourth generation, is the fourth generation of [mobile telecommunications](https://en.wikipedia.org/wiki/Mobile_telephony) technology, succeeding [3G](https://en.wikipedia.org/wiki/3G). A 4G system must provide capabilities defined by [ITU](https://en.wikipedia.org/wiki/ITU) in [IMT Advanced](https://en.wikipedia.org/wiki/IMT_Advanced). Potential and current applications include amended [mobile web](https://en.wikipedia.org/wiki/Mobile_web) access, [IP telephony](https://en.wikipedia.org/wiki/IP_telephony), gaming services, [high-definition](https://en.wikipedia.org/wiki/HDTV)[mobile TV](https://en.wikipedia.org/wiki/Mobile_TV), [video conferencing](https://en.wikipedia.org/wiki/Video_conferencing), [3D television](https://en.wikipedia.org/wiki/3D_television), and [cloud computing](https://en.wikipedia.org/wiki/Cloud_computing).[[citation needed](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]

Two 4G candidate systems are commercially deployed: the [Mobile WiMAX](https://en.wikipedia.org/wiki/Mobile_WiMAX) standard (first used in South Korea in 2007), and the first-release [Long Term Evolution](https://en.wikipedia.org/wiki/Long_Term_Evolution) (LTE) standard (in Oslo, Norway, and Stockholm, Sweden since 2009). It has however been debated if these first-release versions should be considered to be 4G or not, as discussed in the [technical definition](https://en.wikipedia.org/wiki/4G#Technical_understanding) section below.

1. Network/Internet Layer
   * + **IPv4**

Internet Protocol version 4 (IPv4) is the fourth version of the [Internet Protocol](https://en.wikipedia.org/wiki/Internet_Protocol) (IP). It is one of the core protocols of standards-based inter-networking methods in the [Internet](https://en.wikipedia.org/wiki/Internet), and was the first version deployed for production in the [ARPANET](https://en.wikipedia.org/wiki/ARPANET) in 1983. It still routes most Internet traffic today, despite the ongoing deployment of a successor protocol, [IPv6](https://en.wikipedia.org/wiki/IPv6). IPv4 is described in [IETF](https://en.wikipedia.org/wiki/IETF) publication [RFC 791](https://tools.ietf.org/html/rfc791) (September 1981), replacing an earlier definition ([RFC 760](https://tools.ietf.org/html/rfc760), January 1980).

IPv4 is a [connectionless](https://en.wikipedia.org/wiki/Connectionless_communication) protocol for use on [packet-switched](https://en.wikipedia.org/wiki/Packet-switched) networks. It operates on a [best effort delivery](https://en.wikipedia.org/wiki/Best_effort_delivery) model, in that it does not guarantee delivery, nor does it assure proper sequencing or avoidance of duplicate delivery. These aspects, including data integrity, are addressed by an [upper layer](https://en.wikipedia.org/wiki/Upper_layer_protocol) transport protocol, such as the [Transmission Control Protocol](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) (TCP).

* + - **IPv6**

Internet Protocol version 6 (IPv6) is the most recent version of the [Internet Protocol](https://en.wikipedia.org/wiki/Internet_Protocol) (IP), the [communications protocol](https://en.wikipedia.org/wiki/Communications_protocol) that provides an identification and location system for computers on networks and routes traffic across the [Internet](https://en.wikipedia.org/wiki/Internet). IPv6 was developed by the [Internet Engineering Task Force](https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force) (IETF) to deal with the long-anticipated problem of [IPv4 address exhaustion](https://en.wikipedia.org/wiki/IPv4_address_exhaustion). IPv6 is intended to replace [IPv4](https://en.wikipedia.org/wiki/IPv4).

Every device on the Internet is assigned an [IP address](https://en.wikipedia.org/wiki/IP_address) for identification and location definition. With the rapid growth of the Internet after commercialization in the 1990s, it became evident that far more addresses than the IPv4 address space has available were necessary to connect new devices in the future. By 1998, the [Internet Engineering Task Force](https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force) (IETF) had formalized the successor protocol. IPv6 uses a 128-bit address, theoretically allowing 2128, or approximately 3.4×1038 addresses. The actual number is slightly smaller, as multiple ranges are reserved for special use or completely excluded from use. The total number of possible IPv6 address is more than 7.9×1028 times as many as IPv4, which uses 32-bit addresses and provides approximately 4.3 billion addresses. The two protocols are not designed to be [interoperable](https://en.wikipedia.org/wiki/Interoperable), complicating the transition to IPv6. However, several [IPv6 transition mechanisms](https://en.wikipedia.org/wiki/IPv6_transition_mechanisms) have been devised to permit communication between IPv4 and IPv6 hosts.

* + - **6LoWPAN**

6LoWPAN is an [acronym](https://en.wikipedia.org/wiki/Acronym_and_initialism) of [IPv6](https://en.wikipedia.org/wiki/IPv6) over Low power Wireless [Personal Area Networks](https://en.wikipedia.org/wiki/Personal_area_network). 6LoWPAN is the name of a concluded working group in the [Internet](https://en.wikipedia.org/wiki/Internet) area of the [IETF](https://en.wikipedia.org/wiki/IETF).

The 6LoWPAN concept originated from the idea that "the Internet Protocol could and should be applied even to the smallest devices," and that low-power devices with limited processing capabilities should be able to participate in the [Internet of Things](https://en.wikipedia.org/wiki/Internet_of_Things).

The 6LoWPAN group has defined encapsulation and header compression mechanisms that allow IPv6 packets to be sent and received over [IEEE 802.15.4](https://en.wikipedia.org/wiki/IEEE_802.15.4) based networks. [IPv4](https://en.wikipedia.org/wiki/IPv4) and IPv6 are the work horses for data delivery for [local-area networks](https://en.wikipedia.org/wiki/Local-area_networks), [metropolitan area networks](https://en.wikipedia.org/wiki/Metropolitan_area_networks), and [wide-area networks](https://en.wikipedia.org/wiki/Wide-area_networks) such as the Internet. Likewise, IEEE 802.15.4 devices provide sensing communication-ability in the wireless domain. The inherent natures of the two networks though, are different.

3. Transport Layer

* + - **TCP**

The Transmission Control Protocol (TCP) is a core [protocol](https://en.wikipedia.org/wiki/Communications_protocol) of the [Internet protocol suite](https://en.wikipedia.org/wiki/Internet_protocol_suite). It originated in the initial network implementation in which it complemented the [Internet Protocol](https://en.wikipedia.org/wiki/Internet_Protocol) (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides [reliable](https://en.wikipedia.org/wiki/Reliability_(computer_networking)), ordered, and [error-checked](https://en.wikipedia.org/wiki/Error_detection_and_correction) delivery of a stream of [octets](https://en.wikipedia.org/wiki/Octet_(computing)) between applications running on hosts communicating over an IP network. Major Internet applications such as the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web), [email](https://en.wikipedia.org/wiki/Email), [remote administration](https://en.wikipedia.org/wiki/Remote_administration) and [file transfer](https://en.wikipedia.org/wiki/File_transfer) rely on TCP. Applications that do not require reliable data stream service may use the [User Datagram Protocol](https://en.wikipedia.org/wiki/User_Datagram_Protocol) (UDP), which provides a [connectionless](https://en.wikipedia.org/wiki/Connectionless_communication)[datagram](https://en.wikipedia.org/wiki/Datagram) service that emphasizes reduced [latency](https://en.wikipedia.org/wiki/Latency_(engineering)) over reliability.

* + - **UDP**

The User Datagram Protocol (UDP) is one of the core members of the [Internet protocol suite](https://en.wikipedia.org/wiki/Internet_protocol_suite). The protocol was designed by [David P. Reed](https://en.wikipedia.org/wiki/David_P._Reed) in 1980 and formally defined in [RFC 768](https://tools.ietf.org/html/rfc768).

UDP uses a simple [connectionless](https://en.wikipedia.org/wiki/Connectionless) transmission model with a minimum of protocol mechanism. It has no [handshaking](https://en.wikipedia.org/wiki/Handshaking) dialogues, and thus exposes the user's program to any [unreliability](https://en.wikipedia.org/wiki/Reliability_(computer_networking)) of the underlying network protocol. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides [checksums](https://en.wikipedia.org/wiki/Checksums) for data integrity, and [port numbers](https://en.wikipedia.org/wiki/Port_numbers) for addressing different functions at the source and destination of the datagram.

With UDP, computer applications can send messages, in this case referred to as [datagrams](https://en.wikipedia.org/wiki/Datagram), to other hosts on an [Internet Protocol](https://en.wikipedia.org/wiki/Internet_Protocol) (IP) network without prior communications to set up special transmission channels or data paths. UDP is suitable for purposes where error checking and correction is either not necessary or is performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system. If error correction facilities are needed at the network interface level, an application may use the [Transmission Control Protocol](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) (TCP) or [Stream Control Transmission Protocol](https://en.wikipedia.org/wiki/Stream_Control_Transmission_Protocol) (SCTP) which are designed for this purpose.

4. Application layer

* **HTTP**

The Hypertext Transfer Protocol (HTTP) is an [application protocol](https://en.wikipedia.org/wiki/Application_protocol) for distributed, collaborative, [hypermedia](https://en.wikipedia.org/wiki/Hypermedia) information systems. HTTP is the foundation of data communication for the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web).

[Hypertext](https://en.wikipedia.org/wiki/Hypertext) is structured text that uses logical links ([hyperlinks](https://en.wikipedia.org/wiki/Hyperlinks)) between [nodes](https://en.wikipedia.org/wiki/Node_(computer_science)) containing text. HTTP is the protocol to exchange or transfer hypertext.

The standards development of HTTP was coordinated by the [Internet Engineering Task Force](https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force) (IETF) and the [World Wide Web Consortium](https://en.wikipedia.org/wiki/World_Wide_Web_Consortium) (W3C), culminating in the publication of a series of [Requests for Comments](https://en.wikipedia.org/wiki/Requests_for_Comments) (RFCs). The first definition of HTTP/1.1, the version of HTTP in common use, occurred in [RFC 2068](https://tools.ietf.org/html/rfc2068) in 1997, although this was obsoleted by [RFC 2616](https://tools.ietf.org/html/rfc2616) in 1999.

A later version, the successor [HTTP/2](https://en.wikipedia.org/wiki/HTTP/2), was standardized in 2015, then supported by major web browsers and already supported by major web servers.

* **CoAP**

Constrained Application Protocol (CoAP) is a software protocol intended to be used in very simple electronics devices that allows them to communicate interactively over the Internet. It is particularly targeted for small low power sensors, switches, valves and similar components that need to be controlled or supervised remotely, through standard Internet networks. CoAP is an [application layer](https://en.wikipedia.org/wiki/Application_layer) protocol that is intended for use in resource-constrained internet devices, such as [WSN](https://en.wikipedia.org/wiki/WSN) nodes. CoAP is designed to easily translate to [HTTP](https://en.wikipedia.org/wiki/HTTP) for simplified integration with the web, while also meeting specialized requirements such as [multicast](https://en.wikipedia.org/wiki/Multicast) support, very low overhead, and simplicity. Multicast, low overhead, and simplicity are extremely important for [Internet of Things](https://en.wikipedia.org/wiki/Internet_of_Things) (IoT) and [Machine-to-Machine](https://en.wikipedia.org/wiki/Machine-to-Machine) (M2M) devices, which tend to be deeply [embedded](https://en.wikipedia.org/wiki/Embedded_system) and have much less memory and power supply than traditional internet devices have. Therefore, efficiency is very important. CoAP can run on most devices that support [UDP](https://en.wikipedia.org/wiki/User_Datagram_Protocol) or a UDP analogue.

The Internet Engineering Task Force ([IETF](https://en.wikipedia.org/wiki/IETF)) Constrained [RESTful](https://en.wikipedia.org/wiki/RESTful) environments ([CoRE](https://datatracker.ietf.org/doc/charter-ietf-core/)) Working Group has done the major standardization work for this protocol. In order to make the protocol suitable to IoT and M2M applications, various new functionalities have been added. The core of the protocol is specified in [RFC 7252](https://tools.ietf.org/html/rfc7252), important extensions are in various stages of the standardization process.

* **WebSocket**

WebSocket is a protocol providing [full-duplex](https://en.wikipedia.org/wiki/Full-duplex) communication channels over a single [TCP](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) connection. The WebSocket protocol was standardized by the [IETF](https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force) as [RFC 6455](https://tools.ietf.org/html/rfc6455) in 2011, and the WebSocket [API](https://en.wikipedia.org/wiki/Application_programming_interface) in [Web IDL](https://en.wikipedia.org/wiki/Web_IDL) is being standardized by the [W3C](https://en.wikipedia.org/wiki/World_Wide_Web_Consortium).

WebSocket is designed to be implemented in [web browsers](https://en.wikipedia.org/wiki/Web_browser) and [web servers](https://en.wikipedia.org/wiki/Web_server), but it can be used by any client or server application. The WebSocket Protocol is an independent TCP-based protocol. Its only relationship to [HTTP](https://en.wikipedia.org/wiki/HTTP) is that its [handshake](https://en.wikipedia.org/wiki/Handshaking) is interpreted by HTTP servers as an [Upgrade request](https://en.wikipedia.org/wiki/HTTP/1.1_Upgrade_header). The WebSocket protocol makes more interaction between a browser and a website possible, facilitating the real-time data transfer from and to the server. This is made possible by providing a standardized way for the server to send content to the browser without being solicited by the client, and allowing for messages to be passed back and forth while keeping the connection open. In this way a two-way (bi-directional) ongoing conversation can take place between a browser and the server. The communications are done over TCP [port](https://en.wikipedia.org/wiki/Port_(computer_networking)) number 80, which is of benefit for those environments which block non-web Internet connections using a [firewall](https://en.wikipedia.org/wiki/Firewall_(computing)). Similar two-way browser-server communications have been achieved in non-standardized ways using stopgap technologies such as [Comet](https://en.wikipedia.org/wiki/Comet_(programming)).

The WebSocket protocol is currently supported in most major browsers including [Google Chrome](https://en.wikipedia.org/wiki/Google_Chrome), [Internet Explorer](https://en.wikipedia.org/wiki/Internet_Explorer), [Firefox](https://en.wikipedia.org/wiki/Firefox), [Safari](https://en.wikipedia.org/wiki/Safari_(web_browser)) and [Opera](https://en.wikipedia.org/wiki/Opera_web_browser). WebSocket also requires web applications on the server to support it.

* **MQTT**

MQTT (formerly Message Queue Telemetry Transport) is an ISO standard (ISO/IEC PRF 20922) [publish-subscribe](https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern) based "light weight" messaging protocol for use on top of the [TCP/IP protocol](https://en.wikipedia.org/wiki/TCP/IP). It is designed for connections with remote locations where a "small code footprint" is required or the network bandwidth is limited. The [publish-subscribe messaging pattern](https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern) requires a [message broker](https://en.wikipedia.org/wiki/Message_broker). The [broker](https://en.wikipedia.org/wiki/Broker_(service-oriented_architecture)) is responsible for distributing [messages](https://en.wikipedia.org/wiki/Messaging_pattern) to interested clients based on the topic of a message. [Andy Stanford-Clark](https://en.wikipedia.org/wiki/Andy_Stanford-Clark) and Arlen Nipper of [Cirrus Link Solutions](http://www.cirrus-link.com) authored the first version of the protocol in 1999.

The specification does not specify the meaning of "small code foot print" or the meaning of "limited network bandwidth". Thus, the protocol's availability for use depends on the context. In 2013, IBM submitted MQTT v3.1 to the [OASIS](https://en.wikipedia.org/wiki/OASIS_(organization)) specification body with a charter that ensured only minor changes to the specification could be accepted. MQTT-SN is a variation of the main protocol aimed at embedded devices on non-TCP/IP networks, such as [ZigBee](https://en.wikipedia.org/wiki/ZigBee).

Historically, the 'MQ' in 'MQTT' came from IBM's MQ [message queuing](https://en.wikipedia.org/wiki/Message_queuing) product line. However, queuing per se is not required to be supported as a standard feature in all situations.

Alternative protocols include the [Advanced Message Queuing Protocol](https://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol), the IETF [Constrained Application Protocol](https://en.wikipedia.org/wiki/Constrained_Application_Protocol) and [XMPP](https://en.wikipedia.org/wiki/XMPP).

* **XMPP**

Extensible Messaging and Presence Protocol (XMPP) is a [communications protocol](https://en.wikipedia.org/wiki/Communications_protocol) for [message-oriented middleware](https://en.wikipedia.org/wiki/Message-oriented_middleware) based on [XML](https://en.wikipedia.org/wiki/XML) (Extensible Markup Language). It enables the near-real-time exchange of structured yet [extensible](https://en.wikipedia.org/wiki/Extensible) data between any two or more network entities. Originally named Jabber, the protocol was developed by the Jabber open-source community in 1999 for [near real-time](https://en.wikipedia.org/wiki/Near_real-time)[instant messaging](https://en.wikipedia.org/wiki/Instant_messaging) (IM), [presence information](https://en.wikipedia.org/wiki/Presence_information), and [contact list](https://en.wikipedia.org/wiki/Contact_list) maintenance. Designed to be [extensible](https://en.wikipedia.org/wiki/Extensibility), the protocol has been used also for [publish-subscribe](https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern) systems, signalling for [VoIP](https://en.wikipedia.org/wiki/Voice_over_IP), video, [file transfer](https://en.wikipedia.org/wiki/File_transfer), [gaming](https://en.wikipedia.org/wiki/Game), [Internet of Things](https://en.wikipedia.org/wiki/Internet_of_Things) (IoT) applications such as the [smart grid](https://en.wikipedia.org/wiki/Smart_grid), and [social networking services](https://en.wikipedia.org/wiki/Social_networking_service).

Unlike most instant messaging protocols, XMPP is defined in an [open standard](https://en.wikipedia.org/wiki/Open_standard) and uses an [open systems](https://en.wikipedia.org/wiki/Open_system_(computing)) approach of development and application, by which anyone may implement an XMPP service and interoperate with other organizations' implementations. Because XMPP is an open protocol, implementations can be developed using any software license; although many server, client, and library implementations are distributed as [free and open-source software](https://en.wikipedia.org/wiki/Free_and_open-source_software), numerous [freeware](https://en.wikipedia.org/wiki/Freeware) and [commercial software](https://en.wikipedia.org/wiki/Commercial_software) implementations also exist.

* **DDS**

The Data Distribution Service for Real-Time Systems (DDS) is an [Object Management Group](https://en.wikipedia.org/wiki/Object_Management_Group) (OMG) [machine-to-machine](https://en.wikipedia.org/wiki/Machine-to-machine)[middleware](https://en.wikipedia.org/wiki/Middleware) "m2m" standard that aims to enable [scalable](https://en.wikipedia.org/wiki/Scalability), [real-time](https://en.wikipedia.org/wiki/Real-time_computing), [dependable](https://en.wikipedia.org/wiki/Safety-critical), [high-performance](https://en.wikipedia.org/wiki/Many-task_computing) and [interoperable](https://en.wikipedia.org/wiki/Interoperable)[data exchanges](https://en.wikipedia.org/wiki/Data_exchange) between [publishers and subscribers](https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern). DDS addresses the needs of applications like [financial trading](https://en.wikipedia.org/wiki/Trader_(finance)), [air-traffic control](https://en.wikipedia.org/wiki/Air-traffic_control), [smart grid](https://en.wikipedia.org/wiki/Smart_grid) management, and other [big data](https://en.wikipedia.org/wiki/Big_data) applications. The standard is used in applications such as smartphone operating systems, transportation systems and vehicles, [software-defined radio](https://en.wikipedia.org/wiki/Software-defined_radio), and by healthcare providers. DDS may also be used in certain implementations of the [Internet of Things](https://en.wikipedia.org/wiki/Internet_of_Things).

* **AMQP**

The Advanced Message Queuing Protocol (AMQP) is an [open standard](https://en.wikipedia.org/wiki/Open_standard)[application layer](https://en.wikipedia.org/wiki/Application_layer) protocol for [message-oriented middleware](https://en.wikipedia.org/wiki/Message-oriented_middleware). The defining features of AMQP are message orientation, queuing, routing (including point-to-point and [publish-and-subscribe](https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe)), reliability and security.

AMQP mandates the behavior of the messaging provider and client to the extent that implementations from different vendors are interoperable, in the same way as SMTP, HTTP, FTP, etc. have created interoperable systems. Previous standardizations of [middleware](https://en.wikipedia.org/wiki/Middleware) have happened at the API level (e.g. [JMS](https://en.wikipedia.org/wiki/Java_Message_Service)) and were focused on standardizing programmer interaction with different middleware implementations, rather than on providing interoperability between multiple implementations. Unlike JMS, which defines an API and a set of behaviors that a messaging implementation must provide, AMQP is a [wire-level protocol](https://en.wikipedia.org/wiki/Wire_protocol). A wire-level protocol is a description of the format of the data that is sent across the network as a stream of [octets](https://en.wikipedia.org/wiki/Octet_(computing)). Consequently, any tool that can create and interpret messages that conform to this data format can interoperate with any other compliant tool irrespective of implementation language.

Logical Design of IoT

IoT Functional Blocks

* **Device**

The things in IoT are nothing but the devices in IoT.

**Services**

An IoT system uses various types of IoT services such as services for device monitoring, device control services, data publishing services etc.

**Communication**

Many communication technologies are well known such as WiFi, Bluetooth, ZigBee and 2G/3G/4G cellular, but there are also several new emerging networking options such as Thread as an alternative for home automation applications, and Whitespace TV technologies being implemented in major cities for wider area IoT-based use cases. Depending on the application, factors such as range, data requirements, security and power demands and battery life will dictate the choice of one or some form of combination of technologies. These are some of the major communication technologies on offer to developers.

**Management**

Management functional block provides various functions to govern the IoT system.

**Security**

Security functional block secures the IoT system and by providing functions such as authentication, authorization, message and content integrity, and data security.

**Application**

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

IoT Communication Model

* **Request-Response**

Request–response, or request–reply, is one of the basic methods computers use to communicate with each other, in which the first computer sends a request for some data and the second computer responds to the request. Usually, there is a series of such interchanges until the complete message is sent; browsing a [web page](https://en.wikipedia.org/wiki/Web_page) is an example of request–response communication. Request–response can be seen as a telephone call, in which someone is called and they answer the call.

Request–response is a [message exchange pattern](https://en.wikipedia.org/wiki/Messaging_pattern) in which a requestor sends a request message to a replier system which receives and processes the request, ultimately returning a message in response. This is a simple, but powerful messaging pattern which allows two applications to have a two-way conversation with one another over a channel. This pattern is especially common in [client–server](https://en.wikipedia.org/wiki/Client%E2%80%93server) architectures.

* **Publish-Subscribe**

In [software architecture](https://en.wikipedia.org/wiki/Software_architecture), publish–subscribe is a [messaging pattern](https://en.wikipedia.org/wiki/Messaging_pattern) where senders of [messages](https://en.wikipedia.org/wiki/Message_passing), called publishers, do not program the messages to be sent directly to specific receivers, called subscribers, but instead characterize published messages into classes without knowledge of which subscribers, if any, there may be. Similarly, subscribers express interest in one or more classes and only receive messages that are of interest, without knowledge of which publishers, if any, there are.

Pub/sub is a sibling of the[message queue](https://en.wikipedia.org/wiki/Message_queue) paradigm, and is typically one part of a larger [message-oriented middleware](https://en.wikipedia.org/wiki/Message-oriented_middleware) system. Most messaging systems support both the pub/sub and message queue models in their [API](https://en.wikipedia.org/wiki/Application_programming_interface), e.g. [Java Message Service](https://en.wikipedia.org/wiki/Java_Message_Service) (JMS).

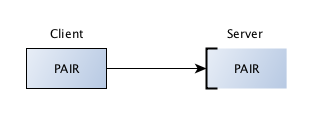
This pattern provides greater network [scalability](https://en.wikipedia.org/wiki/Scalability) and a more dynamic [network topology](https://en.wikipedia.org/wiki/Network_topology), with a resulting decreased flexibility to modify the Publisher and its structure of the data published.

* **Push-Pull**

Most of the business communication tools we use today are “push” tools, where the sender of the message decides who will receive it. Email is the classic example of this; the sender of the message chooses who to put on the To and Cc lines. The recipient gets no choice about whether they receive the message or not, and anyone who is not copied on the message doesn’t even know of its existence. The sender is firmly in control. Instant messaging, SMS and even phone calls are all examples of push.

* **Exclusive Pair**

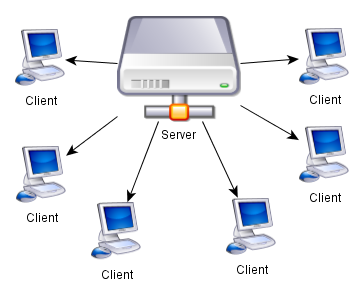
Paired sockets are very similar to regular sockets.

* The communication is bidirectional.
* There is no specific state stored within the socket
* There can only be one connected peer.
* The server listens on a certain port & a client connects to it.
* 

IoT Communication APIs

* **REST-based communication APIs**
* REST (REpresentational State Transfer) is an architectural style, and an approach to communications that is often used in the development of [Web services](http://searchsoa.techtarget.com/definition/Web-Services-Glossary). The use of REST is often preferred over the more heavyweight [SOAP](http://searchsoa.techtarget.com/definition/SOAP) (Simple Object Access Protocol) style because REST does not leverage as much bandwidth, which makes it a better fit for use over the Internet. The SOAP approach requires writing or using a provided server program (to serve data) and a client program (to request data).
* **Client-Server**

The client–server model of computing is a [distributed application](https://en.wikipedia.org/wiki/Distributed_application) structure that partitions tasks or workloads between the providers of a resource or service, called [servers](https://en.wikipedia.org/wiki/Server_(computing)), and service requesters, called [clients](https://en.wikipedia.org/wiki/Client_(computing)). Often clients and servers communicate over a [computer network](https://en.wikipedia.org/wiki/Computer_network) on separate hardware, but both client and server may reside in the same system. A server [host](https://en.wikipedia.org/wiki/Host_(network)) runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests.



* **Stateless**

In computing, a stateless protocol is a [communications protocol](https://en.wikipedia.org/wiki/Communications_protocol) that treats each request as an independent transaction that is unrelated to any previous request so that the communication consists of independent pairs of [request and response](https://en.wikipedia.org/wiki/Request%E2%80%93response). A stateless protocol does not require the [server](https://en.wikipedia.org/wiki/Server_(computing)) to retain [session](https://en.wikipedia.org/wiki/Session_(computer_science)) information or status about each communications partner for the duration of multiple requests. In contrast, a protocol which requires keeping of the internal state on the [server](https://en.wikipedia.org/wiki/Server_(computing)) is known as a [stateful](https://en.wikipedia.org/wiki/Stateful) protocol.

Examples of stateless protocols include the [Internet Protocol](https://en.wikipedia.org/wiki/Internet_Protocol) (IP) which is the foundation for the [Internet](https://en.wikipedia.org/wiki/Internet), and the [Hypertext Transfer Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) (HTTP) which is the foundation of data communication for the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web).

* **Cache-able**

A cacheable communications protocol includes accommodating the ability for client-side caching and a set of specifications for when a response to a query can be cached. The HTTP 1.1 protocol is such a protocol and includes an entire section on the rules of the road for caching

* **Layered system**

Communication programs are often layered. The reference model for communication programs, Open System Interconnection ( [OSI](http://searchnetworking.techtarget.com/definition/OSI) ) is a layered set of protocols in which two multilayered programs, one at either end of a communications exchange, use an identical set of layers. In the OSI model, each multilayer program contains seven layers, each reflecting a different function that has to be performed in order for program-to-program communication to take place between computers.

[TCP/IP](http://searchnetworking.techtarget.com/definition/TCP-IP) is an example of a two-layer ( [TCP](http://searchnetworking.techtarget.com/definition/TCP) and [IP](http://searchunifiedcommunications.techtarget.com/definition/Internet-Protocol) ) set of programs that provide transport and network address functions for Internet communication. A set of TCP/IP and other layered programs is sometimes referred to as a [protocol](http://searchnetworking.techtarget.com/definition/protocol)[stack](http://searchcio-midmarket.techtarget.com/definition/stack) .

Web-Socket based communication APIs

WebSocket is a protocol providing [full-duplex](https://en.wikipedia.org/wiki/Full-duplex) communication channels over a single [TCP](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) connection. The WebSocket protocol was standardized by the [IETF](https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force) as [RFC 6455](https://tools.ietf.org/html/rfc6455) in 2011, and the WebSocket [API](https://en.wikipedia.org/wiki/Application_programming_interface) in [Web IDL](https://en.wikipedia.org/wiki/Web_IDL) is being standardized by the [W3C](https://en.wikipedia.org/wiki/World_Wide_Web_Consortium).

WebSocket is designed to be implemented in [web browsers](https://en.wikipedia.org/wiki/Web_browser) and [web servers](https://en.wikipedia.org/wiki/Web_server), but it can be used by any client or server application. The WebSocket Protocolis an independent TCP-based protocol. Its only relationship to [HTTP](https://en.wikipedia.org/wiki/HTTP) is that its [handshake](https://en.wikipedia.org/wiki/Handshaking) is interpreted by HTTP servers as an [Upgrade request](https://en.wikipedia.org/wiki/HTTP/1.1_Upgrade_header). The WebSocket protocol makes more interaction between a browser and a website possible, facilitating the real-time data transfer from and to the server. This is made possible by providing a standardized way for the server to send content to the browser without being solicited by the client, and allowing for messages to be passed back and forth while keeping the connection open. In this way a two-way (bi-directional) ongoing conversation can take place between a browser and the server. The communications are done over TCP [port](https://en.wikipedia.org/wiki/Port_(computer_networking)) number 80, which is of benefit for those environments which block non-web Internet connections using a [firewall](https://en.wikipedia.org/wiki/Firewall_(computing)). Similar two-way browser-server communications have been achieved in non-standardized ways using stopgap technologies such as [Comet](https://en.wikipedia.org/wiki/Comet_(programming)).

IoT Enabling Technologies

Wireless-Sensor networks

Wireless sensor networks (WSN), sometimes called wireless sensor and actuator networks (WSAN), are spatially distributed [autonomous](https://en.wikipedia.org/wiki/Autonomous)[sensors](https://en.wikipedia.org/wiki/Sensor) to monitor physical or environmental conditions, such as [temperature](https://en.wikipedia.org/wiki/Temperature), [sound](https://en.wikipedia.org/wiki/Sound), [pressure](https://en.wikipedia.org/wiki/Pressure), etc. and to cooperatively pass their data through the network to a main location. The more modern networks are bi-directional, also enabling control of sensor activity. The development of wireless sensor networks was motivated by military applications such as battlefield surveillance; today such networks are used in many industrial and consumer applications, such as industrial process monitoring and control, machine health monitoring, and so on.

The WSN is built of "nodes" – from a few to several hundreds or even thousands, where each node is connected to one (or sometimes several) sensors. Each such sensor network node has typically several parts: a [radio](https://en.wikipedia.org/wiki/Radio)[transceiver](https://en.wikipedia.org/wiki/Transceiver) with an internal [antenna](https://en.wikipedia.org/wiki/Antenna_(radio)) or connection to an external antenna, a [microcontroller](https://en.wikipedia.org/wiki/Microcontroller), an electronic circuit for interfacing with the sensors and an energy source, usually a [battery](https://en.wikipedia.org/wiki/Battery_(electricity)) or an [embedded](https://en.wikipedia.org/wiki/Embedded_system) form of [energy harvesting](https://en.wikipedia.org/wiki/Energy_harvesting). A [sensor node](https://en.wikipedia.org/wiki/Sensor_node) might vary in size from that of a shoebox down to the size of a grain of dust, although functioning "motes" of genuine microscopic dimensions have yet to be created. The cost of sensor nodes is similarly variable, ranging from a few to hundreds of dollars, depending on the complexity of the individual sensor nodes. Size and cost constraints on sensor nodes result in corresponding constraints on resources such as energy, memory, computational speed and communications bandwidth. The topology of the WSNs can vary from a simple [star network](https://en.wikipedia.org/wiki/Star_network) to an advanced [multi-hop](https://en.wikipedia.org/wiki/Mesh_networking)[wireless mesh network](https://en.wikipedia.org/wiki/Wireless_mesh_network). The propagation technique between the hops of the network can be [routing](https://en.wikipedia.org/wiki/Routing) or [flooding](https://en.wikipedia.org/wiki/Flooding_algorithm).

Cloud computing

Cloud computing, also on-demand computing, is a kind of Internet-based computing that provides shared processing resources and data to computers and other devices on demand. It is a model for enabling ubiquitous, on-demand access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services), which can be rapidly provisioned and released with minimal management effort. Cloud computing and storage solutions provide users and enterprises with various capabilities to store and process their data in third-party [data centers](https://en.wikipedia.org/wiki/Data_center).[[3]](https://en.wikipedia.org/wiki/Cloud_computing#cite_note-cloudid-3) It relies on sharing of resources to achieve coherence and [economy of scale](https://en.wikipedia.org/wiki/Economies_of_scale), similar to a utility (like the [electricity grid](https://en.wikipedia.org/wiki/Electrical_grid)) over a network.

**Infrastructure-as-a-Service (IaaS)**

In the most basic cloud-service model—and according to the IETF (Internet Engineering Task Force)—providers of IaaS offer computers—physical or (more often) virtual machines—and other resources. IaaS refers to online services that abstract the user from the details of infrastructure like physical computing resources, location, data partitioning, scaling, security, backup etc. A [hypervisor](https://en.wikipedia.org/wiki/Hypervisor), such as [Xen](https://en.wikipedia.org/wiki/Xen), [Oracle VirtualBox](https://en.wikipedia.org/wiki/VirtualBox), [Oracle VM](https://en.wikipedia.org/wiki/Oracle_VM_Server_for_x86),[KVM](https://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine), [VMware ESX/ESXi](https://en.wikipedia.org/wiki/VMware_ESX), or [Hyper-V](https://en.wikipedia.org/wiki/Hyper-V) runs the virtual machines as guests. Pools of hypervisors within the cloud operational system can support large numbers of virtual machines and the ability to scale services up and down according to customers' varying requirements. IaaS clouds often offer additional resources such as a virtual-machine [disk-image](https://en.wikipedia.org/wiki/Disk_image) library, raw [block storage](https://en.wikipedia.org/wiki/Block_storage), file or [object storage](https://en.wikipedia.org/wiki/Object_storage), firewalls, load balancers, IP addresses, [virtual local area networks](https://en.wikipedia.org/wiki/VLAN) (VLANs), and software bundles.[[63]](https://en.wikipedia.org/wiki/Cloud_computing#cite_note-DHAC-63) IaaS-cloud providers supply these resources on-demand from their large pools of equipment installed in [data centers](https://en.wikipedia.org/wiki/Data_centers). For [wide-area](https://en.wikipedia.org/wiki/Wide_area_network) connectivity, customers can use either the Internet or [carrier clouds](https://en.wikipedia.org/wiki/Carrier_cloud) (dedicated [virtual private networks](https://en.wikipedia.org/wiki/Virtual_private_network)).

**Platform-as-a-Service (PaaS)**

PaaS vendors offer a development environment to application developers. The provider typically develops toolkit and standards for development and channels for distribution and payment. In the PaaS models, cloud providers deliver a [computing platform](https://en.wikipedia.org/wiki/Computing_platform), typically including operating system, programming-language execution environment, database, and web server. Application developers can develop and run their software solutions on a cloud platform without the cost and complexity of buying and managing the underlying hardware and software layers. With some PaaS offers like [Microsoft Azure](https://en.wikipedia.org/wiki/Microsoft_Azure) and [Google App Engine](https://en.wikipedia.org/wiki/Google_App_Engine), the underlying computer and storage resources scale automatically to match application demand so that the cloud user does not have to allocate resources manually. The latter has also been proposed by an architecture aiming to facilitate real-time in cloud environments. Even more specific application types can be provided via PaaS, such as media encoding as provided by services like bitcodin.com or media.io.

**Software-as-a-Service (SaaS)**

In the software as a service (SaaS) model, users gain access to application software and databases. Cloud providers manage the infrastructure and platforms that run the applications. SaaS is sometimes referred to as "on-demand software" and is usually priced on a pay-per-use basis or using a subscription fee.]

In the SaaS model, cloud providers install and operate application software in the cloud and cloud users access the software from cloud clients. Cloud users do not manage the cloud infrastructure and platform where the application runs. This eliminates the need to install and run the application on the cloud user's own computers, which simplifies maintenance and support. Cloud applications differ from other applications in their scalability—which can be achieved by cloning tasks onto multiple [virtual machines](https://en.wikipedia.org/wiki/Virtual_machines) at run-time to meet changing work demand. [Load balancers](https://en.wikipedia.org/wiki/Load_balancer) distribute the work over the set of virtual machines. This process is transparent to the cloud user, who sees only a single access-point. To accommodate a large number of cloud users, cloud applications can be [multitenant](https://en.wikipedia.org/wiki/Multitenant), meaning that any machine may serve more than one cloud-user organization.

Big data Analytics

*Characteristics of big data : volume, velocity, variety*

Big data analytics is the process of examining large data sets containing a variety of data types -- i.e., [big data](http://searchcloudcomputing.techtarget.com/definition/big-data-Big-Data) -- to uncover hidden patterns, unknown correlations, market trends, customer preferences and other useful business information. The analytical findings can lead to more effective marketing, new revenue opportunities, better customer service, improved operational efficiency, competitive advantages over rival organizations and other business benefits.

Communication Protocols

In [telecommunications](https://en.wikipedia.org/wiki/Telecommunications), a communications protocol is a system of rules that allow two or more entities of a [communications system](https://en.wikipedia.org/wiki/Communications_system) to transmit [information](https://en.wikipedia.org/wiki/Information) via any kind of variation of a [physical quantity](https://en.wikipedia.org/wiki/Physical_quantity). These are the rules or standard that defines the [syntax](https://en.wikipedia.org/wiki/Syntax), [semantics](https://en.wikipedia.org/wiki/Semantic) and [synchronization](https://en.wikipedia.org/wiki/Synchronization) of [communication](https://en.wikipedia.org/wiki/Communication) and possible [error recovery methods](https://en.wikipedia.org/wiki/Error_detection_and_correction). Protocols may be implemented by [hardware](https://en.wikipedia.org/wiki/Computer_hardware), [software](https://en.wikipedia.org/wiki/Software), or a combination of both.

Communicating systems use well-defined formats ([protocol](https://en.wiktionary.org/wiki/protocol)) for exchanging messages. Each message has an exact meaning intended to elicit a response from a range of possible responses pre-determined for that particular situation. The specified behavior is typically independent of how it is to be [implemented](https://en.wikipedia.org/wiki/Implementation). Communications protocols have to be agreed upon by the parties involved. To reach agreement, a protocol may be developed into a [technical standard](https://en.wikipedia.org/wiki/Technical_standard). A [programming language](https://en.wikipedia.org/wiki/Programming_language) describes the same for computations, so there is a close analogy between protocols and programming languages: protocols are to communications as programming languages are to computations.

Embedded System

An embedded system is a [computer](https://en.wikipedia.org/wiki/Computer)[system](https://en.wikipedia.org/wiki/System) with a dedicated function within a larger mechanical or electrical system, often with [real-time computing](https://en.wikipedia.org/wiki/Real-time_computing) constraints. It is embedded as part of a complete device often including hardware and mechanical parts. Embedded systems control many devices in common use today. 98 percent of all [microprocessors](https://en.wikipedia.org/wiki/Microprocessors) are manufactured as components of embedded systems.

Examples of properties of typically embedded computers when compared with general-purpose counterparts are low power consumption, small size, rugged operating ranges, and low per-unit cost. This comes at the price of limited processing resources, which make them significantly more difficult to program and to interact with. However, by building intelligence mechanisms on top of the hardware, taking advantage of possible existing sensors and the existence of a network of embedded units, one can both optimally manage available resources at the unit and network levels as well as provide augmented functions, well beyond those available. For example, intelligent techniques can be designed to manage power consumption of embedded systems.

IoT Levels

IoT System comprises of the following components:

* Device
* Resource
* Controller Service
* Database
* Web Service
  + Stateless/Stateful
  + Unidirectional/Bi-directional
  + Request-Response/Full Duplex
  + TCP connection
  + Header overhead
  + Scalability
* Analysis component
* Application

IoT Level-1

Location technologies can be used to provide a context to an application. The application can use the location to present the user with appropriate data and actions that are relevant to them in that location. There are three key pieces of the puzzle here. Firstly, who is the user? Is it an engineer, a loyal customer or another stakeholder? Secondly, what is the context? Is it a specific location in a building, or a specific piece of equipment or thing? Where is the data coming from, is it open data feeds or data feeds that the company has control or access to? Thirdly, what information is important at that time? What does the user need and how are you going to get that data and present it to them, to help them or help the business?

At this level, the most basic level, things are given context and the ability to make their awareness known, connecting digital and physical things together.

IoT Level-2

At the second level, environmental awareness, driven by various different sensors is added. Our phones already have this capability, to an extent, but at this level, the environment is also talking to the phone and telling it about its status. This may also be taken to another level, where the user doesn’t need to be physically there, but the space can still send information back to a backend system. For example, a company is able to monitor their entire infrastructure remotely, without employees being there. This is one of the most common forms of Internet of Things, especially within large manufacturing, engineering or utility firms.

This is already largely used with weather systems, to track changes in the weather and is largely used throughout the agriculture industry.

IoT Level-3

The third level of IoT is linked with remote control of environment and things, whether someone is in a physical space or sitting in an office 5,000 miles away. This is being seen increasingly in the automotive and smart home industry, where people can monitor the status of their cars or homes, turn on heating, or control the lighting and other elements.

This is also seen in large scale modern factories and even offices that have smart heating systems, smart printers and security systems. The third level is also about a level of intelligence within remote systems and things; where they can tell a user or a backend system when they are developing an issue and when they need help. At this level, it goes beyond awareness, from communication to insight, mixed with control.

To get started with IoT consider the levels. What level are you trying to achieve and is it logical to work through level one to three in a linear fashion? Or is it about experimenting and learning as you go? There is an argument for jumping straight in at level two and three, but this will depend on the current infrastructure and capability.

What is interesting to note is that to get started is relatively straightforward. There are ways and means to prove concepts, technologies and user experiences without spending a fortune. Experimenting and aiming to prove or disprove a case is great way to start exploring the possible.

IoT Level-4

A level-4 IoT system has multiple nodes that perform local analysis. Level-4 contains local and cloud-based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices.

IoT Level-5

A level-5 IoT system has multiple end nodes and one coordinator node. The end node perform sensing and/or actuation. Coordinator node collects data from the end nodes and sends to the cloud.

IoT Level-6

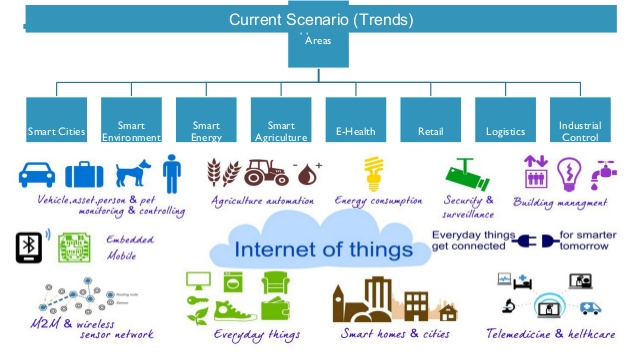
A level-6 IoT system has multiple independent end nodes that perform sensing and/or actuation and send data to the cloud.

Questions

1. Describe an example of an IoT system in which information and knowledge are inferred from data.
2. Why do IoT systems have to be self-adapting and self-configuring?
3. What is the role of things and Internet in IoT?
4. What is the functioon of communication functional block in an IoT system?
5. Describe an example of IoT service that uses publish-subscribe communication model.
6. Describe an example of IoT service that uses WebSocket-based communication.
7. What are the architectural constraints of REST?
8. What is the role of a coordinator in wireless sensor network?
9. What is the role of controller service in an IoT system

**Unit II**

**Domain Specific IOTs:**

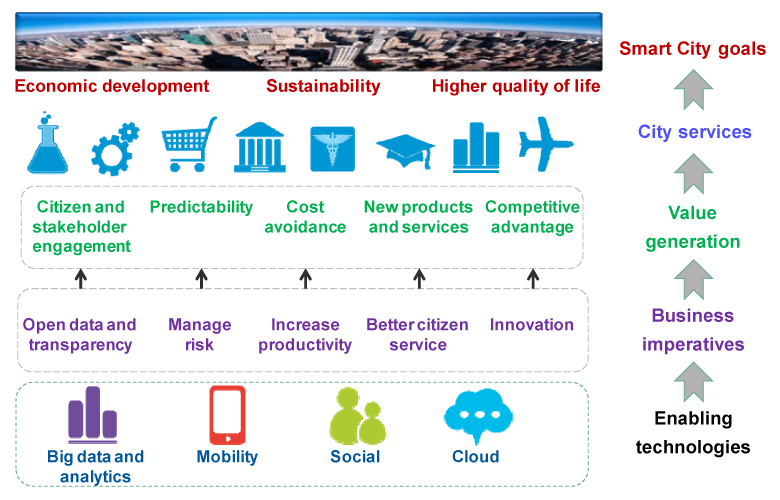


Home Automation

Home automation is the use and control of [home appliances](https://en.wikipedia.org/wiki/Home_appliance) remotely or automatically. Early home automation began with labour-saving machines like [washing machines](https://en.wikipedia.org/wiki/Washing_machine). Some home automation appliances are stand alone and do not communicate, such as a programmable light switch, while others are part of the [internet of things](https://en.wikipedia.org/wiki/Internet_of_things) and are networked for remote control and data transfer. Hardware devices can include sensors (like cameras and thermometers), controllers, actuators (to do things), and communication systems. Remote control can range from a simple [remote control](https://en.wikipedia.org/wiki/Remote_control) to a smartphone with [Bluetooth](https://en.wikipedia.org/wiki/Bluetooth), to a computer on the other side of the world connected by internet. Home automation systems are available which consist of a suite of products designed to work together. These typically connected through [Wi-Fi](https://en.wikipedia.org/wiki/Wi-Fi) or [power line communication](https://en.wikipedia.org/wiki/Power_line_communication) to a hub which is then accessed with a software application. Popular applications include thermostats, security systems, blinds, lighting, smoke/CO detectors, and door locks.[[1]](https://en.wikipedia.org/wiki/Home_automation#cite_note-1) Popular suites of products include [X10](https://en.wikipedia.org/wiki/X10_(industry_standard)), [Z-Wave](https://en.wikipedia.org/wiki/Z-Wave), and [Zigbee](https://en.wikipedia.org/wiki/Zigbee) all of which are incompatible with each other. Home automation is the domestic application of [building automation](https://en.wikipedia.org/wiki/Building_automation)

* Smart Lighting
* Smart Appliances
* Intrusion detection
* Smoke/Gas detectors

Cities



A smart city is an [urban development](https://en.wikipedia.org/wiki/Urban_development) vision to integrate multiple [information and communication technology](https://en.wikipedia.org/wiki/Information_and_communication_technology) (ICT) solutions in a [secure](https://en.wikipedia.org/wiki/Information_security) fashion to manage a city’s assets – the city’s assets include, but not limited to, local departments information systems, [schools](https://en.wikipedia.org/wiki/School), [libraries](https://en.wikipedia.org/wiki/Libraries), [transportation systems](https://en.wikipedia.org/wiki/Transportation_system), [hospitals](https://en.wikipedia.org/wiki/Hospital), [power plants](https://en.wikipedia.org/wiki/Power_plant), [water supply networks](https://en.wikipedia.org/wiki/Water_supply_network), [waste management](https://en.wikipedia.org/wiki/Waste_management), [law enforcement](https://en.wikipedia.org/wiki/Law_enforcement), and other [community services](https://en.wikipedia.org/wiki/Community_service). The goal of building a smart city is to improve [quality of life](https://en.wikipedia.org/wiki/Quality_of_life) by using technology to improve the [efficiency](https://en.wikipedia.org/wiki/Efficiency) of services and meet residents’ needs. ICT allows city officials to interact directly with the community and the city [infrastructure](https://en.wikipedia.org/wiki/Infrastructure) and to monitor what is happening in the city, how the city is evolving, and how to enable a better quality of life. Through the use of sensors integrated with real-time monitoring systems, [data are collected](https://en.wikipedia.org/wiki/Data_collection) from citizens and devices - then processed and analyzed. The information and knowledge gathered are keys to tackling inefficiency.

ICT is used to enhance quality, performance and [interactivity](https://en.wikipedia.org/wiki/Interactivity) of [urban services](https://en.wikipedia.org/w/index.php?title=Urban_service&action=edit&redlink=1), to [reduce costs](https://en.wikipedia.org/wiki/Cost_reduction) and [resource consumption](https://en.wikipedia.org/wiki/Resource_consumption) and to improve contact between citizens and government. Smart city applications are developed with the goal of improving the management of urban flows and allowing for real time responses to challenges. A smart city may therefore be more prepared to respond to challenges than one with a simple 'transactional' relationship with its citizens. Yet, the term itself remains unclear to its specifics and therefore, open to many interpretations and subject

* Smart parking
* Smart Lighting
* Smart roads
* Structural health monitoring
* Surveillance
* Emergency Response

Environment

The concept of smart environments evolves from the definition of [ubiquitous computing](https://en.wikipedia.org/wiki/Ubiquitous_computing) that, according to [Mark Weiser](https://en.wikipedia.org/wiki/Mark_Weiser), promotes the ideas of "a physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network."[[1]](https://en.wikipedia.org/wiki/Smart_environment#cite_note-Weiser-99-1)

Smart environments are envisioned as the byproduct of [pervasive computing](https://en.wikipedia.org/wiki/Pervasive_computing) and the availability of cheap computing power, making human interaction with the system a pleasant experience.

Smart environments are broadly classified to have the following features

1. *Remote control of devices*, like power line communication systems to control devices.
2. *Device Communication*, using middleware, and Wireless communication to form a picture of connected environments.
3. *Information Acquisition/Dissemination from sensor networks*
4. *Enhanced Services by Intelligent Devices*
5. *Predictive and Decision-Making capabilities*

* Weather Monitoring
* Air pollution Monitoring
* Noise pollution Monitoring
* Forest Fire detection
* River Floods Detection

Energy

**Smart Grids**

A smart grid is an [electrical grid](https://en.wikipedia.org/wiki/Electrical_grid) which includes a variety of operational and energy measures including smart meters, smart appliances, renewable energy resources, and energy efficiency resources. Electronic power conditioning and control of the production and distribution of electricity are important aspects of the smart grid.

Smart grid policy is organized in Europe as Smart Grid European Technology Platform.

Roll-out of smart grid technology also implies a fundamental re-engineering of the electricity services industry, although typical usage of the term is focused on the technical infrastructure.

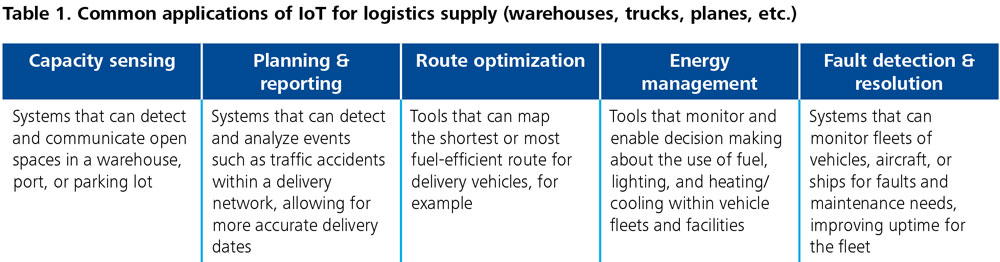
* Renewable Energy Systems
* Prognostics

Retail

IoT has caught the imagination of technology enthusiasts and there are many predictions of how it might revolutionize industries and practices as we know them today. The retail industry has already undergone a wave of disruption with the onset of e-commerce and online retail. There is every chance that IoT heralds the next wave of disruption along the following areas in retail:  
– Supply Chain Management  
– Inventory & Warehouse Management  
– Marketing  
– In-Store Experience

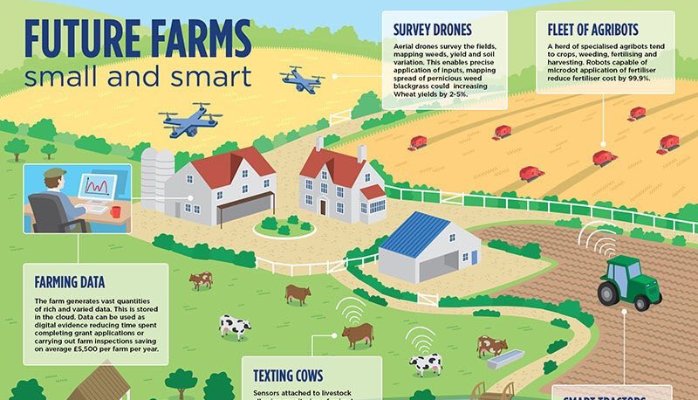
* Inventory management
* Smart payments
* Smart vending machines

Logistics



* Route Generation & Scheduling
* Fleet tracking
* Shipment Monitoring
* Remote Vehicle Diagnostics

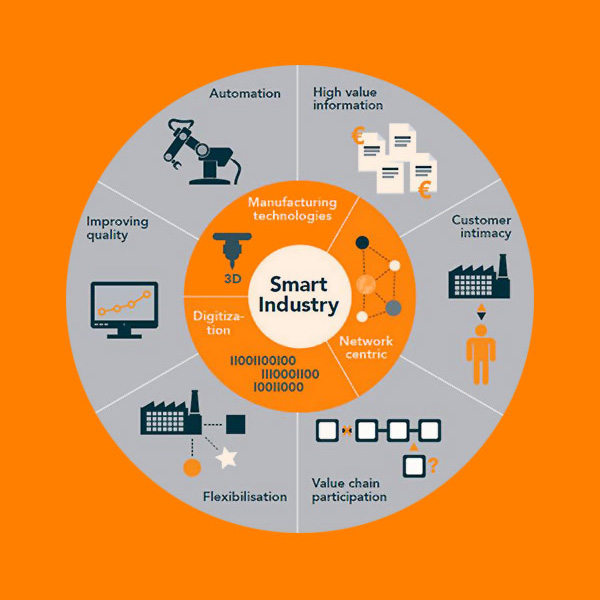
Agriculture



Agriculture has been evolving with new technology such as the Internet of Things (IoT). For example, greenhouses are connected to each other, and their environments are controlled automatically and optimized for the best quality of agricultural products. In addition, the advanced cattle sheds are built based on the IoT technologies. These efforts enhance the quality and safety of agricultural products and mitigate information asymmetry between producers and consumers.

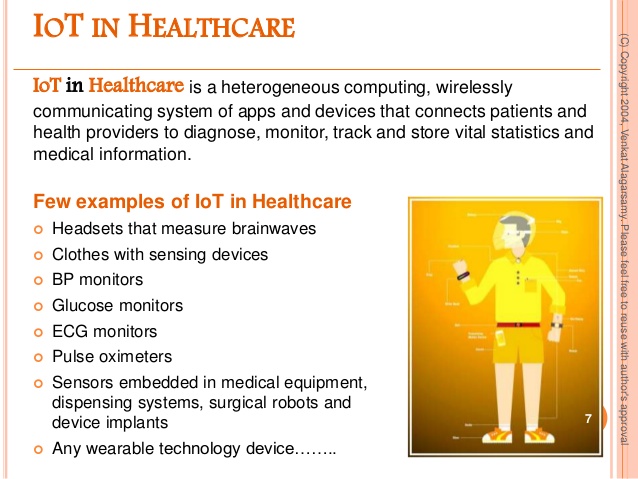
* Smart Irrigation
* Green House control

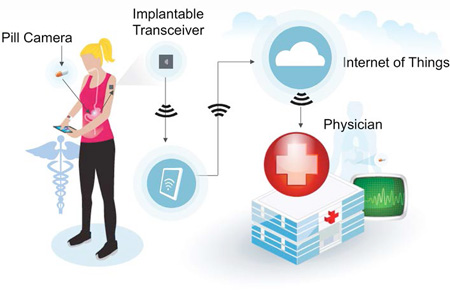
Industry



* Machine Diagnosis and prognosis
* Indoor air quality monitoring

Health & Life Style





* Health & Fittness monitoring
* Wearable electronics

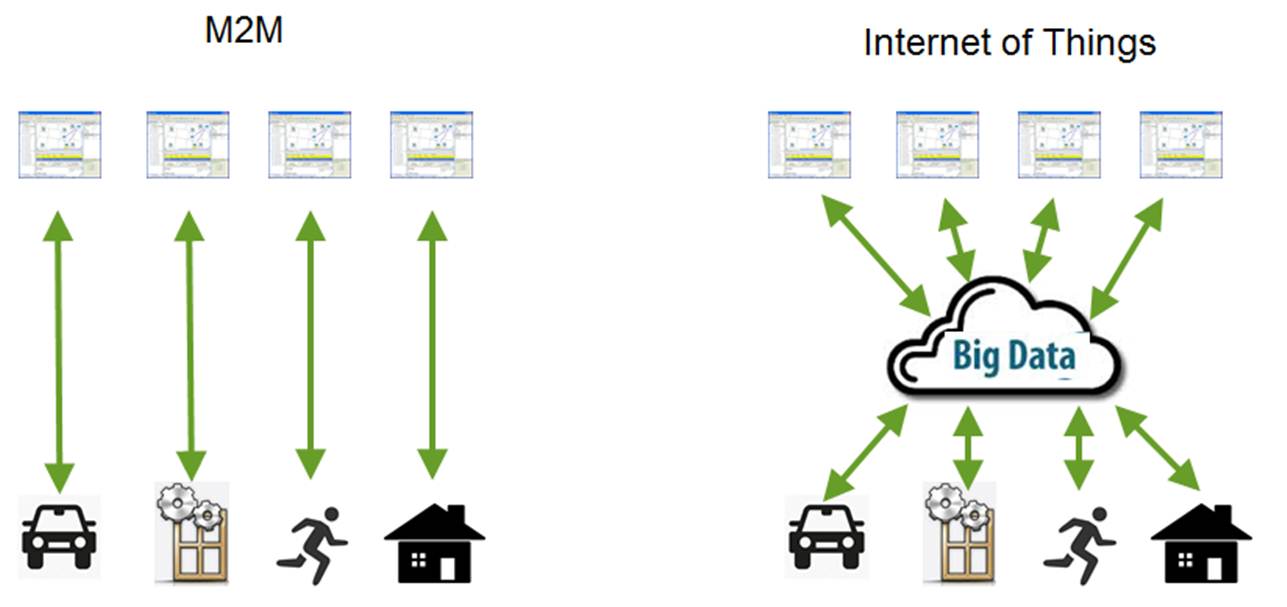
Questions

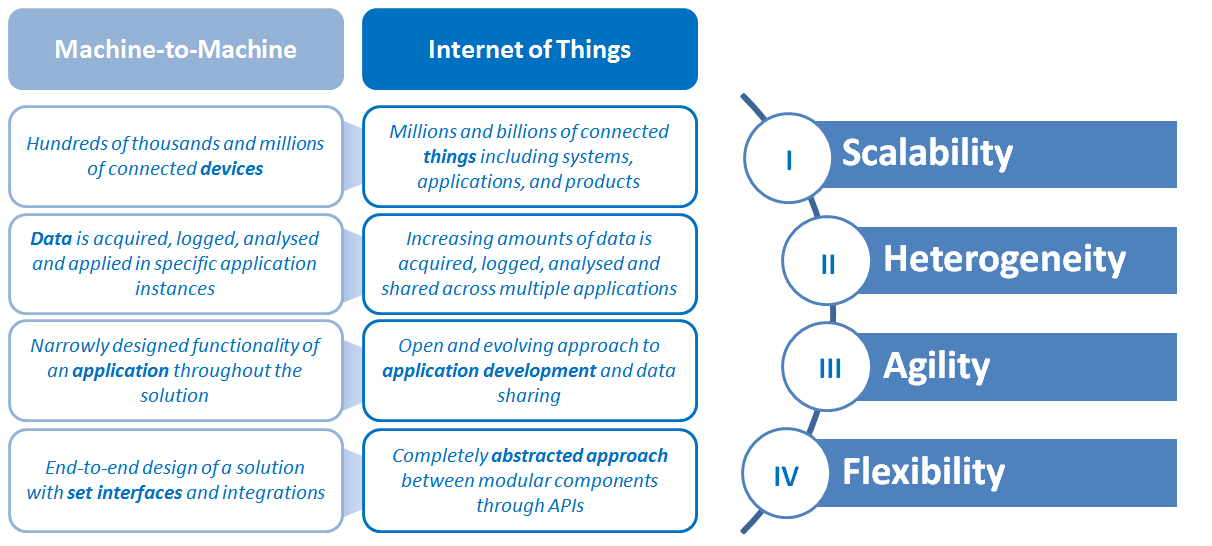
1. Determine the IoT-levels for designing home automation IoT systems including smart leghting and intrusion detection.
2. Determine the IoT-levels for designing structural health monitoring system.
3. Determine various communication models that can be used for weather monitoring system. What is more appropriate model for this system. Describe the pros and cons.
4. Determine the type of data generated by a forest fire detection system? Describe alternative approaches for storing the data. What type of analysis is required for forest fire detection from the data collected.

**Unit III**

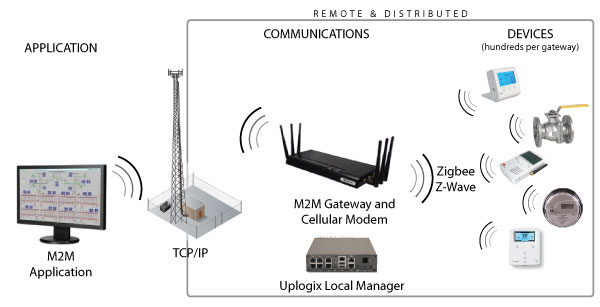
**M2M & System Management with NETCONF-YANG:**

M2M

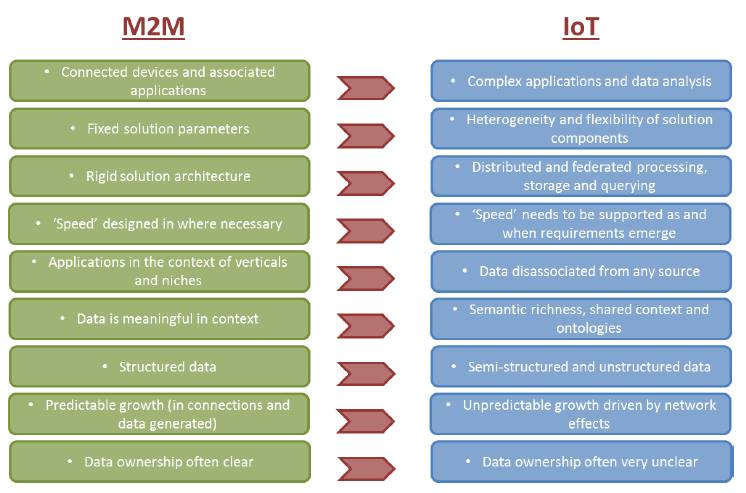


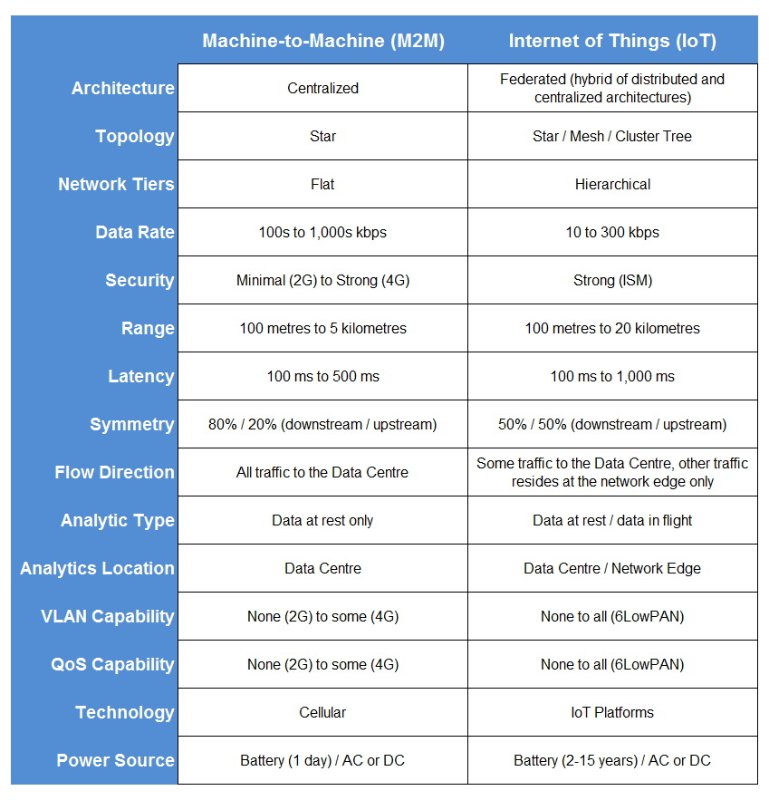


The aptly named IoT subset M2M initially represented closed, point-to-point communication between physical-first objects. The explosion of mobile devices and IP-based connectivity mechanisms has enabled data transmission across a system of networks. M2M is more recently referred to technologies that enable communication between machines without human intervention. Examples include telemetry, traffic control, robotics, and other applications involving device-to-device communications.



Difference between IOT and M2M

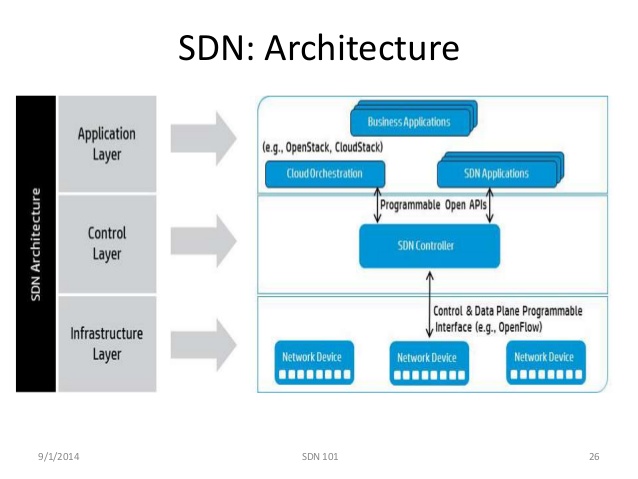


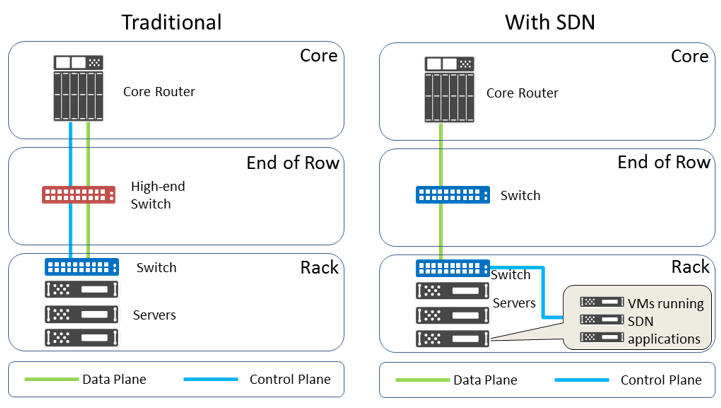


1. Communication protocols
2. Machines in M2M vs Things in IoT
3. Hardware vs Software Emphasis
4. Data Collection & Analysis
5. Applications

SDN and NFV for IOT

Software defined Networking (SDN)





Software-Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The OpenFlow® protocol is a foundational element for building SDN solutions. The SDN architecture is:

* **Directly programmable**: Network control is directly programmable because it is decoupled from forwarding functions.
* **Agile**: Abstracting control from forwarding lets administrators dynamically adjust network-wide traffic flow to meet changing needs.
* **Centrally managed**: Network intelligence is (logically) centralized in software-based SDN controllers that maintain a global view of the network, which appears to applications and policy engines as a single, logical switch.
* **Programmatically configured**: SDN lets network managers configure, manage, secure, and optimize network resources very quickly via dynamic, automated SDN programs, which they can write themselves because the programs do not depend on proprietary software.
* **Open standards-based and vendor-neutral**: When implemented through open standards, SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple, vendor-specific devices and protocols.

Limitations of the conventional network architectures:

* Complex Network Devices
* management Overhead
* Limited Scalability

Key elements of SDN

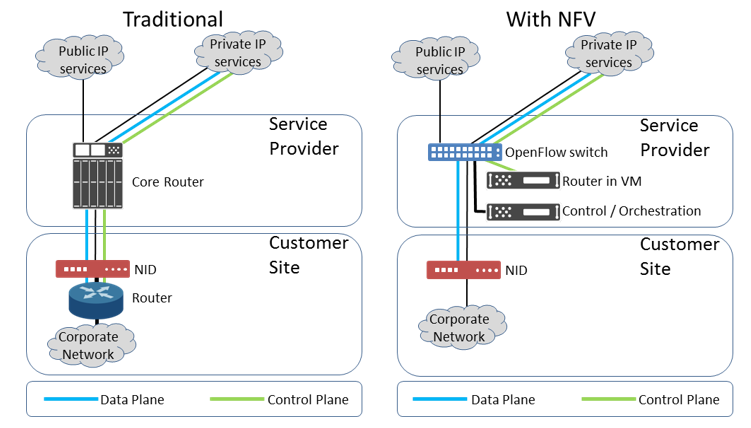
* Centralized network Controller
* Programmable Open APIs
* Standard Communication Interface (OpenFlow)

Network Function Virtualization (NFV)

Network function virtualization (NFV) is a [network architecture](https://en.wikipedia.org/wiki/Network_architecture) concept that uses the technologies of IT [virtualization](https://en.wikipedia.org/wiki/Virtualization) to virtualize entire classes of [network node](https://en.wikipedia.org/wiki/Network_node) functions into building blocks that may connect, or chain together, to create communication services.

NFV relies upon, but differs from, traditional server-[virtualization](https://en.wikipedia.org/wiki/Virtualization) techniques, such as those used in enterprise IT. A virtualized network function, or VNF, may consist of one or more [virtual machines](https://en.wikipedia.org/wiki/Virtual_machines) running different software and processes, on top of standard high-volume servers, switches and storage, or even [cloud computing](https://en.wikipedia.org/wiki/Cloud_computing) infrastructure, instead of having custom hardware appliances for each network function.

For example, a virtual [session border controller](https://en.wikipedia.org/wiki/Session_border_controller) could be deployed to protect a network without the typical cost and complexity of obtaining and installing physical units. Other examples of NFV include virtualized [load balancers](https://en.wikipedia.org/wiki/Load_balancing_(computing)), [firewalls](https://en.wikipedia.org/wiki/Firewall_(computing)), [intrusion detection devices](https://en.wikipedia.org/wiki/Intrusion_detection_system) and [WAN accelerators](https://en.wikipedia.org/wiki/WAN_optimization).



Key elements of NFV architecture

* Virtualized Network Function (VNF)
* NFV Infrastructure (NFVI)
* NFV Management and Orchestration

Need for IOT Systems Management

* Automating Configuration
* Monitoring Operational & statistical data
* Improved Reliablity
* System Wide Configuration
* Multiple System Configuration
* Retrieving and reusing configuration

Simple Network Management Protocol (SNMP)

Simple Network Management Protocol (SNMP) is an [Internet-standard protocol](https://en.wikipedia.org/wiki/Internet_protocol_suite) for collecting and organizing information about managed devices on [IP](https://en.wikipedia.org/wiki/Internet_Protocol) networks and for modifying that information to change device behavior. Devices that typically support SNMP include routers, switches, servers, workstations, printers, modem racks and more.

SNMP is widely used in [network management systems](https://en.wikipedia.org/wiki/Network_management_systems) to [monitor](https://en.wikipedia.org/wiki/Network_monitoring) network-attached devices for conditions that warrant administrative attention. SNMP exposes management data in the form of variables on the managed systems, which describe the system configuration. These variables can then be queried (and sometimes set) by managing applications.

Limitations of SNMP

* SNMP is stateless
* SNMP is connectionless, so unreliable
* SNMP can be used only for device monitoring and status polling.
* It is difficult to differentiate between configuration and stste data in MIBs.
* SNMP does not support easy retrieval and playback of configurations.
* SNMP latest version have security addede at the cost of increased complexity.

Network Operator Requirements

* Ease of use
* Distinction between configuration and state data.
* Fetch and state data seperately
* configuration of the network as a whole
* configuration transactions across devices
* configuration deltas
* Dump and restore configurations
* configuration validation
* configuration database schemas
* Comparing configurations
* Role-based access control lists
* Multiple configuration sets
* Support for both data-oriented and task-oriented access control

NETCONF

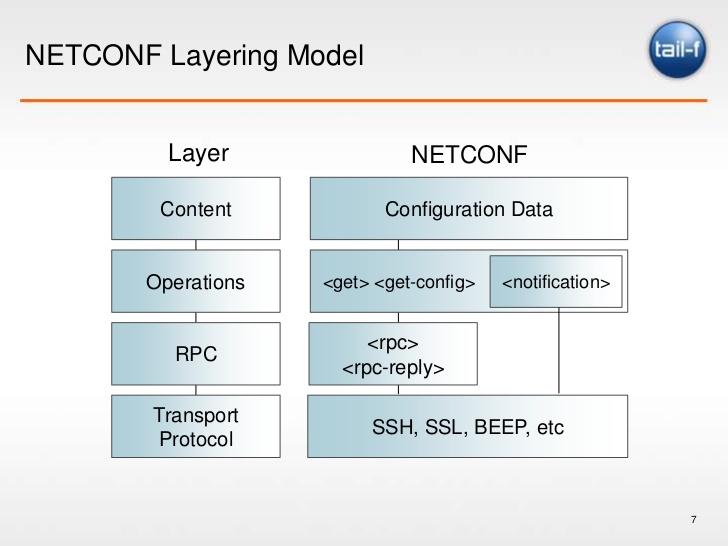
The Network Configuration Protocol (NETCONF) is a [network management](https://en.wikipedia.org/wiki/Network_management) protocol developed and standardized by the [IETF](https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force). It was developed in the NETCONF working group and published in December 2006 as [RFC 4741](https://tools.ietf.org/html/rfc4741) and later revised in June 2011 and published as [RFC 6241](https://tools.ietf.org/html/rfc6241). The NETCONF protocol specification is an Internet Standards Track document.

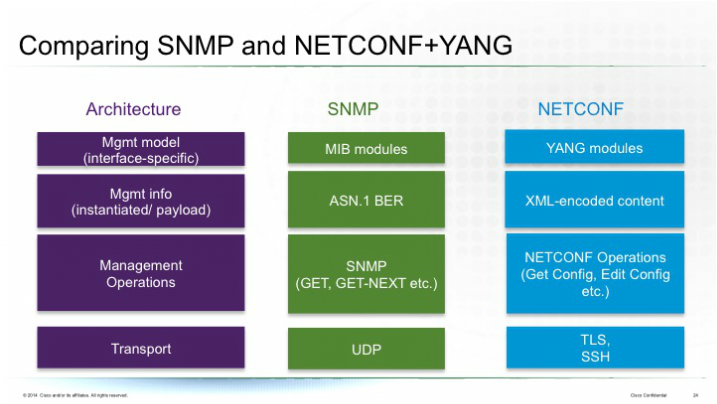
NETCONF provides mechanisms to install, manipulate, and delete the configuration of network devices. Its operations are realized on top of a simple [remote procedure call](https://en.wikipedia.org/wiki/Remote_procedure_call) (RPC) layer. The NETCONF protocol uses an [Extensible Markup Language](https://en.wikipedia.org/wiki/Extensible_Markup_Language) (XML) based data encoding for the configuration data as well as the protocol messages. The protocol messages are exchanged on top of a secure transport protocol.

The NETCONF protocol can be conceptually partitioned into four layers:

1. The Content layer consists of configuration data and notification data.
2. The Operations layer defines a set of base protocol operations to retrieve and edit the configuration data.
3. The Messages layer provides a mechanism for encoding remote procedure calls (RPCs) and notifications.
4. The Secure Transport layer provides a secure and reliable transport of messages between a client and a server.

The NETCONF protocol has been implemented in network devices such as routers and switches by some major equipment vendors. One particular strength of NETCONF is its support for robust configuration change transactions involving a number of devices.





YANG

YANG is a data modeling language for the [NETCONF](https://en.wikipedia.org/wiki/NETCONF) network configuration protocol. The name is an acronym for "Yet Another Next Generation". The YANG data modeling language was developed by the NETMOD working group in the [Internet Engineering Task Force](https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force) (IETF) and was published as [RFC 6020](https://tools.ietf.org/html/rfc6020) in October 2010. The data modeling language can be used to model both configuration data as well as state data of network elements. Furthermore, YANG can be used to define the format of event notifications emitted by network elements and it allows data modelers to define the signature of remote procedure calls that can be invoked on network elements via the NETCONF protocol.

YANG is a modular language representing data structures in an [XML](https://en.wikipedia.org/wiki/XML) tree format. The data modeling language comes with a number of builtin data types. Additional application specific data types can be derived from the builtin data types. More complex reusable data structures can be represented as groupings. YANG data models can use [XPATH](https://en.wikipedia.org/wiki/XPATH) expressions to define constraints on the elements of a YANG data model.

IoT Systems management with NETCONF-YANG.

* Management System
* Management API
* Transaction Manager
* Rollback Manager
* Data Model Manager
* Configuration Validator
* Configuration Database
* Configuration API
* Data Provider API

NETOPEER

Netopeer server is the leading open source NETCONF reference implementation. It has many features but sometimes it may prove to be a challenging task to gather all pieces and get it installed successfully on your Linux box.

The Netofeer tools include

* Netopeer-server
* Netopeer-agent
* Netopeer-cli
* Netopeer-manager
* Netopeer-configurator

Questions:

1. Which communication protocols are used for M2M local area networks?
2. What are the differences between Machines in M2M and Things in IoT?
3. How do data collection and analysis approaches differ in M2M and IoT?
4. What are the differences between SDN and NFV?
5. Describe how SDN can be used for various levels of IoT?
6. What is the function of a centralized network controller in SDN?
7. Describe how NFV can be used for virtualizing IoT devices?
8. Why is network wide configuration important for IoT systems with multiple nodes?
9. Which limitation make SNMP unsuitable for IoT systems?
10. What is the difference between configuration and state data?
11. What is the role of a NETCONF server?
12. What is the function of a data model manager?
13. Describe the roles of YANG and TransAPI modules in device management.

**Unit IV**

**Developing Internet of Things & Logical Design using Python:**

IOT Design Methodology

* Step 1: Purpose & Requirements Specification
* Step 2: Process specification
* Step 3: Domain Model Specification
* Step 4: Information Model Specification
* Step 5: Service Specifications
* Step 6: IoT Level Specification
* Step 7: Functional View Specification
* Step 8: Operational View Specification
* Step 9: Device and Component Integration
* Step 10: Application Development

## Features of Python

**Simple**

Python is a simple and minimalistic language. Reading a good Python program feels almost like reading English, although very strict English! This pseudo-code nature of Python is one of its greatest strengths. It allows you to concentrate on the solution to the problem rather than the language itself.

**Easy to Learn**

As you will see, Python is extremely easy to get started with. Python has an extraordinarily simple syntax, as already mentioned.

**Free and Open Source**

Python is an example of a FLOSS (Free/LibrÃ© and Open Source Software). In simple terms, you can freely distribute copies of this software, read it's source code, make changes to it, use pieces of it in new free programs, and that you know you can do these things. FLOSS is based on the concept of a community which shares knowledge. This is one of the reasons why Python is so good - it has been created and is constantly improved by a community who just want to see a better Python.

**High-level Language**

When you write programs in Python, you never need to bother about the low-level details such as managing the memory used by your program, etc.

**Portable**

Due to its open-source nature, Python has been ported (i.e. changed to make it work on) to many platforms. All your Python programs can work on any of these platforms without requiring any changes at all if you are careful enough to avoid any system-dependent features.

You can use Python on Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE and even PocketPC !

**Interpreted**

This requires a bit of explanation.

A program written in a compiled language like C or C++ is converted from the source language i.e. C or C++ into a language that is spoken by your computer (binary code i.e. 0s and 1s) using a compiler with various flags and options. When you run the program, the linker/loader software copies the program from hard disk to memory and starts running it.

Python, on the other hand, does not need compilation to binary. You just run the program directly from the source code. Internally, Python converts the source code into an intermediate form called bytecodes and then translates this into the native language of your computer and then runs it. All this, actually, makes using Python much easier since you don't have to worry about compiling the program, making sure that the proper libraries are linked and loaded, etc, etc. This also makes your Python programs much more portable, since you can just copy your Python program onto another computer and it just works!

**Object Oriented**

Python supports procedure-oriented programming as well as object-oriented programming. In procedure-oriented languages, the program is built around procedures or functions which are nothing but reusable pieces of programs. In object-oriented languages, the program is built around objects which combine data and functionality. Python has a very powerful but simplistic way of doing OOP, especially when compared to big languages like C++ or Java.

**Extensible**

If you need a critical piece of code to run very fast or want to have some piece of algorithm not to be open, you can code that part of your program in C or C++ and then use them from your Python program.

**Embeddable**

You can embed Python within your C/C++ programs to give 'scripting' capabilities for your program's users.

**Extensive Libraries**

The Python Standard Library is huge indeed. It can help you do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, ftp, email, XML, XML-RPC, HTML, WAV files, cryptography, GUI (graphical user interfaces), Tk, and other system-dependent stuff. Remember, all this is always available wherever Python is installed. This is called the 'Batteries Included' philosophy of Python.

Besides, the standard library, there are various other high-quality libraries such as [wxPython](http://www.wxpython.org), [Twisted](http://www.twistedmatrix.com/products/twisted), [Python Imaging Library](http://www.pythonware.com/products/pil/index.htm) and many more.

Python Data Types & Data Structures

**Numbers**

Number data types store numeric values. They are immutable data types, means that changing the value of a number data type results in a newly allocated object.

Python supports different numerical types −

int (signed integers): They are often called just integers or ints, are positive or negative whole numbers with no decimal point. Integers in Python 3 are of unlimited size. Python 2 has two integer types - int and long. There is no 'long integer' in Python 3 any more.

float (floating point real values) : Also called floats, they represent real numbers and are written with a decimal point dividing the integer and fractional parts. Floats may also be in scientific notation, with E or e indicating the power of 10 (2.5e2 = 2.5 x 102 = 250).

complex (complex numbers) : are of the form a + bJ, where a and b are floats and J (or j) represents the square root of -1 (which is an imaginary number). The real part of the number is a, and the imaginary part is b. Complex numbers are not used much in Python programming.

* **Strings**

Strings are amongst the most popular types in Python. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes. Creating strings is as simple as assigning a value to a variable.

#!/usr/bin/python3

var1 = 'Hello World!'

var2 = "Python Programming"

print ("var1[0]: ", var1[0])print ("var2[1:5]: ", var2[1:5])

* **Lists**

The list is a most versatile data-type available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example −

list1 = ['physics', 'chemistry', 1997, 2000];

list2 = [1, 2, 3, 4, 5 ];

list3 = ["a", "b", "c", "d"]

* **Tuples**

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example −

tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5 );

tup3 = "a", "b", "c", "d";

The empty tuple is written as two parentheses containing nothing −

tup1 = ();

To write a tuple containing a single value you have to include a comma, even though there is only one value −

tup1 = (50,);

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

* **Dictionaries**

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example−

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

print "dict['Name']: ", dict['Name']print "dict['Age']: ", dict['Age']

When the above code is executed, it produces the following result −

dict['Name']: Zara

dict['Age']: 7

* **Type conversion**

Below is a table of the conversion functions in Python and their examples.

|  |  |  |
| --- | --- | --- |
| **Function** | **Converting what to what** | **Example** |
| int() | string, floating point → integer | >>> int('2014') 2014 >>> int(3.141592)3 |
| float() | string, integer → floating point number | >>> float('1.99') 1.99 >>> float(5)5.0 |
| str() | integer, float, list, tuple, dictionary → string | >>> str(3.141592) '3.141592' >>> str([1,2,3,4])'[1, 2, 3, 4]' |
| list() | string, tuple, dictionary → list | >>> list('Mary') # list of characters in 'Mary' ['M', 'a', 'r', 'y'] >>> list((1,2,3,4)) # (1,2,3,4) is a tuple[1, 2, 3, 4] |
| tuple() | string, list → tuple | >>> tuple('Mary')('M', 'a', 'r', 'y') >>> tuple([1,2,3,4]) # [ ] for list, ( ) for tuple(1, 2, 3, 4) |

Control Flow

* **if statement**

Python provide various tools for flow control. Some of them are if , if .. elif .. else, if..else,while ,for , switch, pass, range, break, else, continue, function etc. In this article I’ll be covering only if-else and loops.

**if – if** this is the case **then** do this

This control statement indicate that if something happens then do this. It’s a good way of handling some short conditions. An if block can be followed by zero or any number of else block.

if (condition):

statements…

Note: Use of colon ( ":" ) in python is same as we use brackets in java or C++. Python uses colon and indentation for defining the scope or code block. So if you are getting an error like the following picture then correct your code indentation.

## if … else

It’s like **if** have money **then** spend **else** wait for salary. I hope it’s clear from the previous line what this statement means. It’s like if the conditions of if block is evaluated to true then execute that block else execute the else block. The else block is optional and one if can have any number of else blocks.

Syntax:

if (condition):

statements…

else:

default statements…

## if … elif … else

It’s like checking multiple conditions. Let’s Take an example if pocketMoney greater then 90T then Okay else if pocket money is equal to 50T and Less then 90T then its average else it’s not enough. Basically that statement can replace switch statement. You can put any number of elif blocks after if and else block is optional.

Syntax:

if (option1 condition):

option1 statements…

elif(option2 condition):

option2 statements…

elif(option3 condition):

option3 statements…

else:

default option statements…

* **for**

It is used for looping over a sequence. Python doesn’t supports old for loop or c-style for loop. In traditional style for loop we have one variable which iterates over a sequence and we can change the value of sequence and variable as well but in modern for loop we have iteration variable that iterates over a fixed sequence. We can not change the sequence as well as iteration variable during iteration.

Syntax:

for iterator\_name in iterating\_sequence:

…statements…

* **while**

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

### Syntax

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

while expression:

statement(s)

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

* **range**

Sometimes it is required that we just want to iterate over number sequence like 1,2,3,4,… To solve this purpose python provides range function which generate the arithmetic progression with number of terms equal to the parameter passed in it. We have 3 variations of range() function. One take only

Syntax:

1. for iterator\_name in range(10):

…statements…

1. for iterator\_name in range(start,end):

…statements…

1. for iterator\_name in range(start,stop,increment):

…statements…

* **break/continue**

Break is used for terminating the loop abnormally. i.e that even the sequence is not completed but loop is exited.

Continue is used for continuing to next iteration of loop without doing anything inside the loop.

Else is introduced in python and it is placed in loop without if. It will execute only if the loop is terminated without break.

Note: there are two more else statement, one is for **if** that I already explained and other is with try. The difference between try else block and loop else is try else block executes when no code is executed and loop else executes when no break is executed.

* **pass**

Pass statement is used when you don’t want to do anything but it is required for the sake of syntactical correctness. Pass has two common uses.

1. It is used for creating minimal classes.
2. It is used as place holder. For example consider the following snippet

Functions

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

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

## Defining a Function

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

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

## Syntax

def functionname( parameters ):

"function\_docstring"

function\_suite

return [expression]

By default, parameters have a positional behavior and you need to inform them in the same order that they were defined.

Modules

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

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

## Example

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

def print\_func( par ):

print "Hello : ", par

return

## The *import* Statement

You can use any Python source file as a module by executing an import statement in some other Python source file. The *import* has the following syntax:

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

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module.

Packages

Packages are namespaces which contain multiple packages and modules themselves. They are simply directories, but with a twist.

Each package in Python is a directory which **MUST** contain a special file called \_\_init\_\_.py. This file can be empty, and it indicates that the directory it contains is a Python package, so it can be imported the same way a module can be imported.

If we create a directory called foo, which marks the package name, we can then create a module inside that package called bar. We also must not forget to add the \_\_init\_\_.py file inside the foo directory.

To use the module bar, we can import it in two ways:

import foo.bar

File Handling

## Opening and Closing Files

Until now, you have been reading and writing to the standard input and output. Now, we will see how to use actual data files.

Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a **file** object.

## The open Function

Before you can read or write a file, you have to open it using Python's built-in *open()* function. This function creates a **file** object, which would be utilized to call other support methods associated with it.

## Syntax

file object = open(file\_name [, access\_mode][, buffering])

Here are parameter details:

**file\_name:** The file\_name argument is a string value that contains the name of the file that you want to access.

**access\_mode:** The access\_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).

**buffering:** If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

Here is a list of the different modes of opening a file −

|  |  |
| --- | --- |
| **Modes** | **Description** |
| r | Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode. |
| rb | Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode. |
| r+ | Opens a file for both reading and writing. The file pointer placed at the beginning of the file. |
| rb+ | Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file. |
| w | Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing. |
| wb | Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing. |
| w+ | Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing. |
| wb+ | Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing. |
| a | Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing. |
| ab | Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing. |
| a+ | Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing. |
| ab+ | Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing. |

Date/ Time Operations

Time intervals are floating-point numbers in units of seconds. Particular instants in time are expressed in seconds since 12:00am, January 1, 1970(epoch).

There is a popular **time** module available in Python which provides functions for working with times, and for converting between representations. The function *time.time()* returns the current system time in ticks since 12:00am, January 1, 1970(epoch).

## Example

#!/usr/bin/pythonimport time; # This is required to include time module.

ticks = time.time()

print "Number of ticks since 12:00am, January 1, 1970:", ticks

Classes

Class :A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.

Instance/Object : An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.

Inheritance :The transfer of the characteristics of a class to other classes that are derived from it.

Function Overloading :The assignment of more than one behavior to a particular function. The operation performed varies by the types of objects or arguments involved.

Operator Overloading :The assignment of more than one function to a particular operator.

Python Packages of Interest for IoT

* **JSON (JavaScript Object Notation)**

JSON or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange. The JSON format was originally specified by Douglas Crockford, and is described in RFC 4627. The official Internet media type for JSON is application/json. The JSON filename extension is .json. This tutorial will help you understand JSON and its use within various programming languages such as PHP, PERL, Python, Ruby, Java, etc.

Before you start with encoding and decoding JSON using Python, you need to install any of the JSON modules available. For this tutorial we have downloaded and installed [Demjson](http://deron.meranda.us/python/demjson/) as follows −

$tar xvfz demjson-1.6.tar.gz

$cd demjson-1.6

$python setup.py install

## JSON Functions

|  |  |
| --- | --- |
| **Function** | **Libraries** |
| Encode | Encodes the Python object into a JSON string representation. |
| Decode | Decodes a JSON-endoded string into a Python object. |

## Encoding JSON in Python (encode)

Python encode() function encodes the Python object into a JSON string representation.

### Syntax

demjson.encode(self, obj, nest\_level=0)

### Example

The following example shows arrays under JSON with Python.

#!/usr/bin/pythonimport demjson

data = [ { 'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4, 'e' : 5 } ]

json = demjson.encode(data)

print json

While executing, this will produce the following result −

[{"a":1,"b":2,"c":3,"d":4,"e":5}]

## Decoding JSON in Python (decode)

Python can use demjson.decode() function for decoding JSON. This function returns the value decoded from json to an appropriate Python type.

### Syntax

demjson.decode(self, txt)

### Example

The following example shows how Python can be used to decode JSON objects.

#!/usr/bin/pythonimport demjson

json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';

text = demjson.decode(json)

print text

On executing, it will produce the following result −

{u'a': 1, u'c': 3, u'b': 2, u'e': 5, u'd': 4}

* **XML**

XML is a portable, open source language that allows programmers to develop applications that can be read by other applications, regardless of operating system and/or developmental language.

## What is XML?

The Extensible Markup Language (XML) is a markup language much like HTML or SGML. This is recommended by the World Wide Web Consortium and available as an open standard.

XML is extremely useful for keeping track of small to medium amounts of data without requiring a SQL-based backbone.

## XML Parser Architectures and APIs

The Python standard library provides a minimal but useful set of interfaces to work with XML.

The two most basic and broadly used APIs to XML data are the SAX and DOM interfaces.

**Simple API for XML (SAX) :** Here, you register callbacks for events of interest and then let the parser proceed through the document. This is useful when your documents are large or you have memory limitations, it parses the file as it reads it from disk and the entire file is never stored in memory.

**Document Object Model (DOM) API :** This is a World Wide Web Consortium recommendation wherein the entire file is read into memory and stored in a hierarchical (tree-based) form to represent all the features of an XML document.

SAX obviously cannot process information as fast as DOM can when working with large files. On the other hand, using DOM exclusively can really kill your resources, especially if used on a lot of small files.

SAX is read-only, while DOM allows changes to the XML file. Since these two different APIs literally complement each other, there is no reason why you cannot use them both for large projects.

* **HTTPLib & URLLib**

The [httplib](https://docs.python.org/2/library/httplib.html#module-httplib) module has been renamed to http.client in Python 3. The [2to3](https://docs.python.org/2/glossary.html#term-2to3) tool will automatically adapt imports when converting your sources to Python 3.

This module defines classes which implement the client side of the HTTP and HTTPS protocols. It is normally not used directly — the module [urllib](https://docs.python.org/2/library/urllib.html#module-urllib) uses it to handle URLs that use HTTP and HTTPS.

Here is an example session that uses the "GET" method:

>>> import httplib

>>> conn = httplib.HTTPConnection("www.python.org")

>>> conn.request("GET", "/index.html")

>>> r1 = conn.getresponse()

>>> print r1.status, r1.reason

200 OK

>>> data1 = r1.read()

>>> conn.request("GET", "/parrot.spam")

>>> r2 = conn.getresponse()

>>> print r2.status, r2.reason

404 Not Found

>>> data2 = r2.read()

>>> conn.close()

Here is an example session that shows how to "POST" requests:

>>> import httplib, urllib

>>> params = urllib.urlencode({'spam': 1, 'eggs': 2, 'bacon': 0})

>>> headers = {"Content-type": "application/x-www-form-urlencoded",

... "Accept": "text/plain"}

>>> conn = httplib.HTTPConnection("musi-cal.mojam.com:80")

>>> conn.request("POST", "/cgi-bin/query", params, headers)

>>> response = conn.getresponse()

>>> print response.status, response.reason

200 OK

>>> data = response.read()

>>> conn.close()

The urllib module in Python 3 allows you access websites via your program. This opens up as many doors for your programs as the internet opens up for you. urllib in Python 3 is slightly different than urllib2 in Python 2, but they are mostly the same. Through urllib, you can access websites, download data, parse data, modify your headers, and do any GET and POST requests you might need to do.

Some websites do not appreciate programs accessing their data and placing weight on their servers. When they find out that a program is visiting them, they may sometimes choose to block you out, or serve you different data that a regular user might see. This can be annoying at first, but can be overcome with some simple code. To do this, you just need to modify the user-agent, which is a variable within your header that you send in. Headers are bits of data that you share with servers to let them know a bit about you. This is where Python, by default, tells the website that you are visiting with Python's urllib and your Python version. We can, however, modify this, and act as if we are a lowly Internet Explorer user, a Chrome user, or anything else really!

I would not recommend just blindly doing this, however, if a website is blocking you out. Websites will also employ other tactics as well, but usually they are doing it because they also offer an API that is specifically made more programs to access. Programs are usually just interested in the data, and do not need to be served fancy HTML or CSS data, nor data for advertisements, etc.

* **SMTPLib**

Simple Mail Transfer Protocol (SMTP) is a protocol, which handles sending e-mail and routing e-mail between mail servers.

Python provides **smtplib** module, which defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.

Here is a simple syntax to create one SMTP object, which can later be used to send an e-mail −

import smtplib

smtpObj = smtplib.SMTP( [host [, port [, local\_hostname]]] )

Here is the detail of the parameters:

**host:** This is the host running your SMTP server. You can specifiy IP address of the host or a domain name like tutorialspoint.com. This is optional argument.

**port:** If you are providing *host* argument, then you need to specify a port, where SMTP server is listening. Usually this port would be 25.

**local\_hostname**: If your SMTP server is running on your local machine, then you can specify just *localhost* as of this option.

An SMTP object has an instance method called **sendmail**, which is typically used to do the work of mailing a message. It takes three parameters −

The *sender* - A string with the address of the sender.

The *receivers* - A list of strings, one for each recipient.

The *message* - A message as a string formatted as specified in the various RFCs.

Questions:

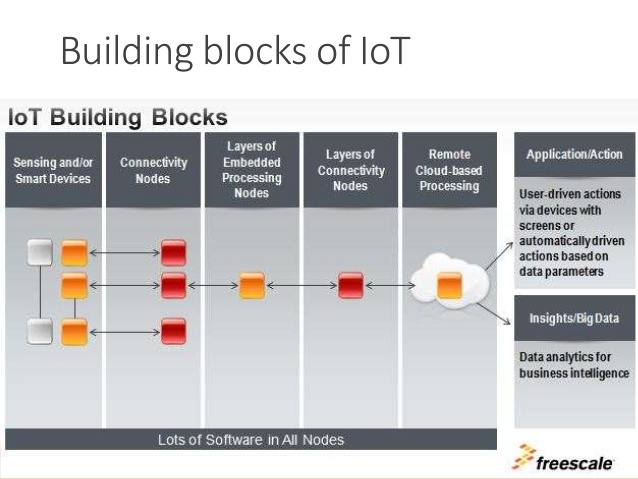
1. What is the difference between a physical and virtual entity?
2. What is an IoT device?
3. What is the purpose of information model?
4. What are the various service types?
5. What is the need for a controller service?
6. What is the difference between procedure-oriented programming and object-oriented programming?
7. What is an interpreted language?
8. Describe a use case of Python dictionary.
9. What is the keyword argument in Python?
10. What are variable length arguments?
11. What is the difference between a Python module and a package?
12. How is function overriding implemented in Python?

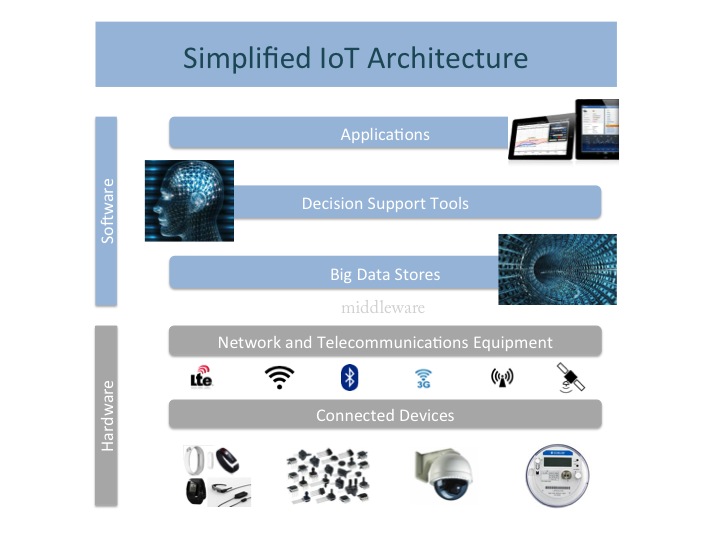
**Unit V**

**IoT Physical Devices & Endpoints:**

What is an IoT Device

The **Internet of Things** (**IoT**) is the network of physical objects—**devices**, vehicles, buildings and other items—embedded with electronics, software, sensors, and network connectivity that enables these objects to collect and exchange data.





* **Sensing**

IoT devices and systems include sensors that track and measure activity in the world. One example is Smartthings' open-and-close sensors that detect whether or not a drawer, window, or door in your home is open or closed.

* **Actuation**

Actuation is nothing but responding back to the environment based on the processing of collected data at a sensor device

* **Communication**

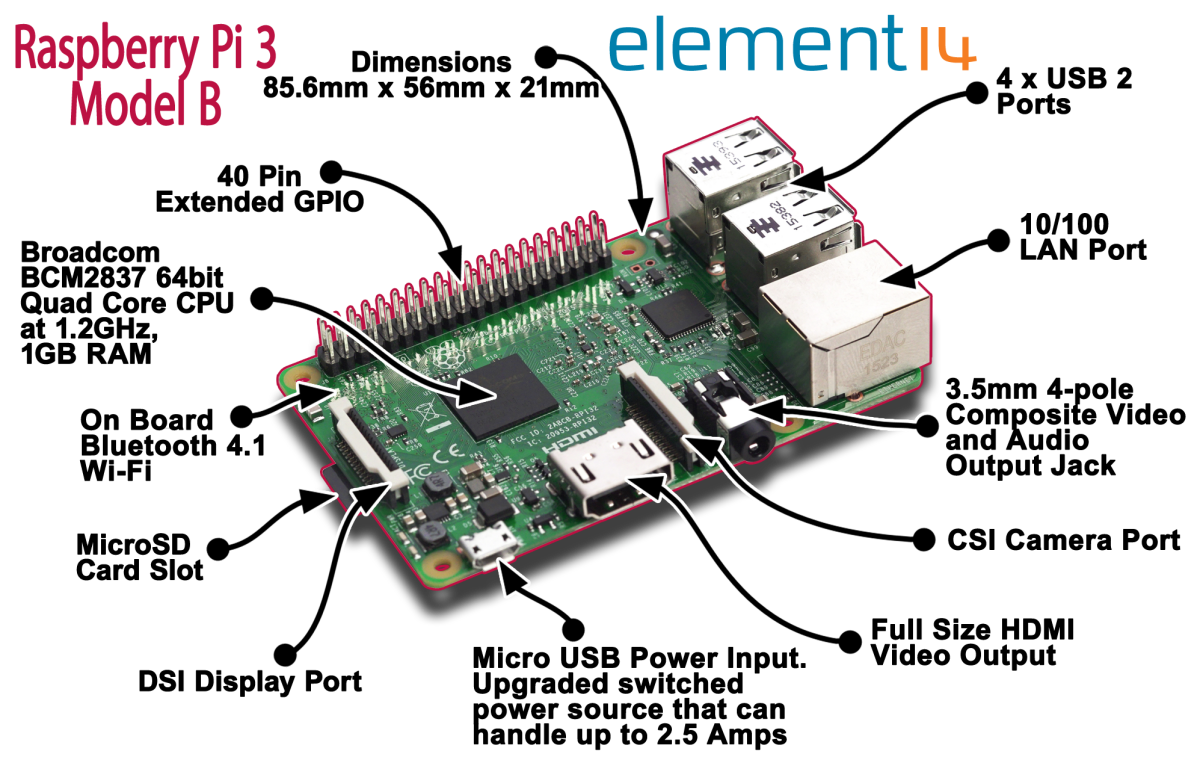
Once the embedded sensors have gathered data, they are tasked with transmitting this data to an identified destination. This transference can utilize different connectivity methodologies, depending on the requirements of the corresponding device, but will most often use wired/wireless or PAN/BAN/LAN communication links. Regardless of the method used, the links will generally only need to transmit small kilobytes of data, unless higher bandwidths are required.

* **Analysis & Processing**

The reliance on communication to create cohesion between the physical and the technological realms places importance on the microprocessors that enable this connection to occur. Whether these microprocessors allow objects to sense their surroundings, exchange data with other components, or interact with the cloud, their incorporation into the overall schema of the IoT is integral to the engagement of the varied systems that must cooperate with one another. Given the changing nature of the landscape, microprocessors that are low power, cost-effective and leave a smaller imprint will be those that are favored within the IoT.

Exemplary Device: Raspberry Pi

The Raspberry Pi is a low cost, **credit-card sized computer** that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It’s capable of doing everything you’d expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.



About the Board

**Processor and RAM**

The [system on a chip](https://en.wikipedia.org/wiki/System_on_a_chip) (SoC) used in the first generation Raspberry Pi is somewhat equivalent to the chip used in older [smartphones](https://en.wikipedia.org/wiki/Smartphone) (such as [iPhone](https://en.wikipedia.org/wiki/IPhone), [3G](https://en.wikipedia.org/wiki/IPhone_3G), [3GS](https://en.wikipedia.org/wiki/IPhone_3GS)). The Raspberry Pi is based on the [Broadcom](https://en.wikipedia.org/wiki/Broadcom) BCM2835 SoC, which includes an 700 [MHz](https://en.wikipedia.org/wiki/Hertz)[ARM11](https://en.wikipedia.org/wiki/ARM11)76JZF-S processor, [VideoCore](https://en.wikipedia.org/wiki/VideoCore) IV [graphics processing unit](https://en.wikipedia.org/wiki/Graphics_processing_unit) (GPU), and RAM. It has a Level 1 [cache](https://en.wikipedia.org/wiki/CPU_cache) of 16 [KB](https://en.wikipedia.org/wiki/Kibibyte) and a Level 2 cache of 128 KB. The Level 2 cache is used primarily by the GPU. The SoC is [stacked](https://en.wikipedia.org/wiki/Package_on_package) underneath the RAM chip, so only its edge is visible.

On the older beta model B boards, 128 MB was allocated by default to the GPU, leaving 128 MB for the CPU.[[25]](https://en.wikipedia.org/wiki/Raspberry_Pi#cite_note-25) On the first 256 MB release model B (and model A), three different splits were possible. The default split was 192 MB (RAM for CPU), which should be sufficient for standalone 1080p video decoding, or for simple 3D, but probably not for both together. 224 MB was for Linux only, with only a 1080p [framebuffer](https://en.wikipedia.org/wiki/Framebuffer), and was likely to fail for any video or 3D. 128 MB was for heavy 3D, possibly also with video decoding (e.g. XBMC).

**USB Ports**

The Raspberry Pi 3 shares the same SMSC LAN9514 chip as its predecessor, the Raspberry Pi 2, adding 10/100 Ethernet connectivity and four USB channels to the board.

As before, the SMSC chip connects to the SoC via a single USB channel, acting as a USB-to-Ethernet adaptor and USB hub.

Though the model A and A+ and Zero do not have an [8P8C](https://en.wikipedia.org/wiki/8P8C) ("RJ45") Ethernet port, they can be connected to a network using an external user-supplied USB Ethernet or [Wi-Fi](https://en.wikipedia.org/wiki/Wi-Fi) adapter.

On the model B and B+ the Ethernet port is provided by a built-in USB Ethernet adapter.

The Raspberry Pi 3 is equipped with 2.4 GHz WiFi 802.11n and Bluetooth 4.1 in addition to the 10/100 Ethernet port.

**Ethernet Ports**

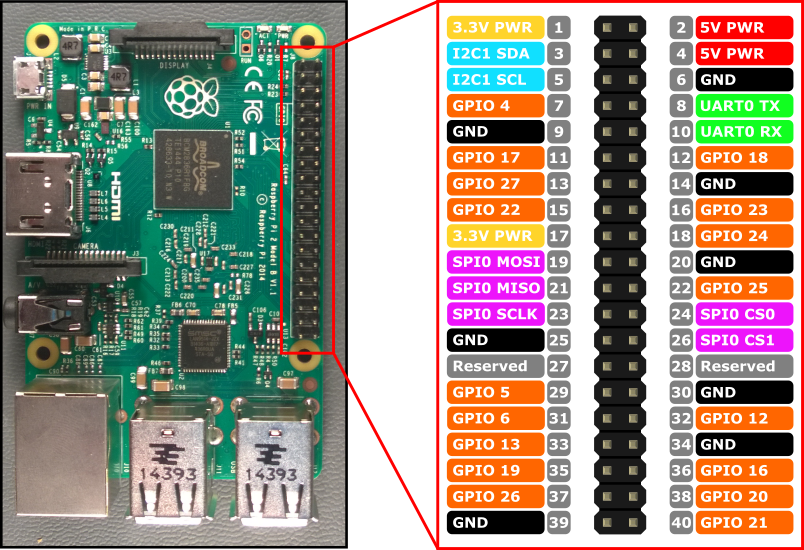
* Though the model A and A+ and Zero do not have an [8P8C](https://en.wikipedia.org/wiki/8P8C) ("RJ45") Ethernet port, they can be connected to a network using an external user-supplied USB Ethernet or [Wi-Fi](https://en.wikipedia.org/wiki/Wi-Fi) adapter.
* On the model B and B+ the Ethernet port is provided by a built-in USB Ethernet adapter.
* The Raspberry Pi 3 is equipped with 2.4 GHz WiFi 802.11n and Bluetooth 4.1 in addition to the 10/100 Ethernet port.

**HMDI Output and Composite Video Output**

* The video controller is capable of standard modern TV resolutions, such as HD and Full HD, and higher or lower monitor resolutions and older standard CRT TV resolutions.
* As shipped (i.e. without custom overclocking) it is capable of the following: 640×350 [EGA](https://en.wikipedia.org/wiki/Enhanced_Graphics_Adapter); 640×480 [VGA](https://en.wikipedia.org/wiki/Video_Graphics_Array); 800×600 [SVGA](https://en.wikipedia.org/wiki/Super_video_graphics_array); 1024×768 [XGA](https://en.wikipedia.org/wiki/XGA); 1280×720 [720p](https://en.wikipedia.org/wiki/720p)[HDTV](https://en.wikipedia.org/wiki/High-definition_television#High-definition_display_resolutions); 1280×768 [WXGA](https://en.wikipedia.org/wiki/Graphic_display_resolutions#WXGA) variant; 1280×800 [WXGA](https://en.wikipedia.org/wiki/Graphic_display_resolutions#WXGA) variant; 1280×1024 [SXGA](https://en.wikipedia.org/wiki/SXGA); 1366×768 [WXGA](https://en.wikipedia.org/wiki/Graphic_display_resolutions#WXGA) variant; 1400×1050 [SXGA+](https://en.wikipedia.org/wiki/SXGA+); 1600×1200 [UXGA](https://en.wikipedia.org/wiki/UXGA); 1680×1050 [WXGA+](https://en.wikipedia.org/wiki/WXGA+); 1920×1080 [1080p](https://en.wikipedia.org/wiki/1080p)[HDTV](https://en.wikipedia.org/wiki/High-definition_television#High-definition_display_resolutions); 1920×1200 [WUXGA](https://en.wikipedia.org/wiki/WUXGA).

**GPIO Pins**

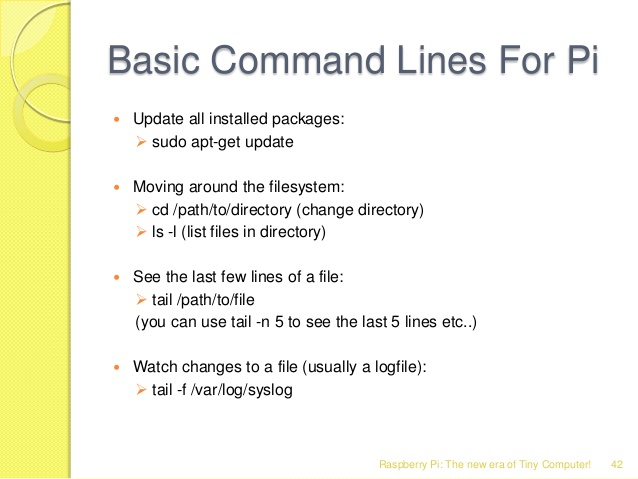
* The Raspberry Pi 3 features the same 40-pin general-purpose input-output (GPIO) header as all the Pis going back to the Model B+ and Model A+.
* Any existing GPIO hardware will work without modification; the only change is a switch to which UART is exposed on the GPIO’s pins, but that’s handled internally by the operating system.



Linux on Raspberry Pi

Raspberry Pi supports various flavors of Linux including

* Raspbian
* Arch
* Pidora
* RaspBMC
* OpenELEC
* RISC OS



Raspberry Pi Interfaces

* Serial
* SPI : There are 5 pins on Raspberry Pi for SPI interface
  + MISO (Master In Slave Out)
  + MOSI (Master Out Slave In)
  + SCK (Serial Clock)
  + CE0 (Chip Enable 0)
  + CE0 (Chip Enable 1)
* I2C

Programming Raspberry Pi with Python

* Controlling LED with Raspberry Pi
* Interfacing an LED and switch with Raspberry Pi
* Interfacing an Light Sensor (LDR) with Raspberry Pi

Other IOT Devices

**pcDuino**

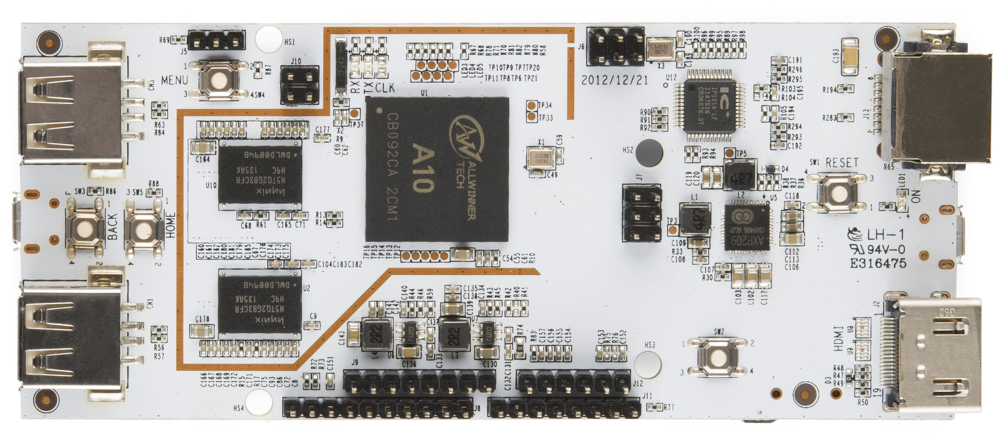
The beauty of the pcDuino lies in its extraordinarily well exposed hardware peripherals. However, using these peripherals is more complex than using them on, say, an Arduino-compatible board.

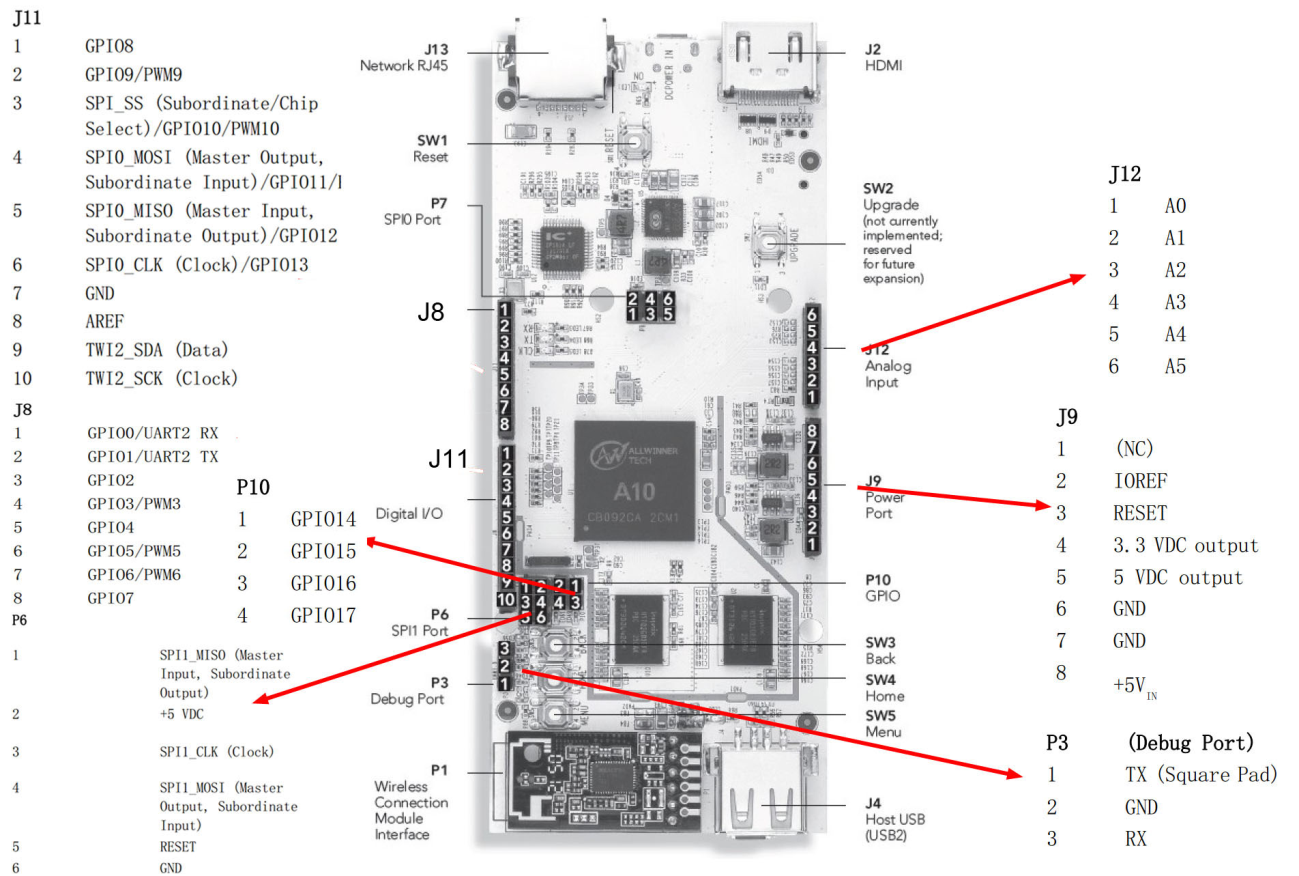
This tutorial will help you sort out the various peripherals, what they can do, and how to use them.

Before we get started, there are a few things you should be certain you’re familiar with, to get the most out of this tutorial:

* [pcDuino](https://www.sparkfun.com/products/11712) - some familiarity with the basics of the pcDuino is needed before you jump into this. Please review our [Getting Started with pcDuino](https://learn.sparkfun.com/tutorials/pcduino-hookup-guide) tutorial before going any further.
* Linux - the biggest thing you should be familiar with it the Linux OS. Remember, pcDuino is not an Arduino–it is a modern microcomputer running a fully-functional, if compact, operating system.
* [SPI](https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi) - a synchronous (clocked) serial peripheral interface used for communications between chips at a board level. Requires a minimum of four wires (clock, master-out-slave-in data, master-in-slave-out data, and slave chip select), and each additional chip added to the bus requires one extra chip select line.
* [I2C](https://learn.sparkfun.com/tutorials/i2c) - also known as IIC (inter-integrated circuit), SMBus, or TWI (two-wire interface), I2C uses only two wires (bidirectional data and clock lines) to communicate with multiple devices.
* [Serial Communication](https://learn.sparkfun.com/tutorials/serial-communication) - an asynchronous (no transmitted clock) data interface with at least two wires (data transmit and data receive; sometimes, additional signals are added to indicate when a device is ready to send or receive).
* [Pulse Width Modulation](https://learn.sparkfun.com/tutorials/pulse-width-modulation) - a digital-to-analog conversion technique using a fixed frequency square wave of varying duty cycle, which can be easily converted to an analog signal between 0V and the full amplitude of the digital IC driving the signal.
* [Analog-to-Digital Conversion](https://learn.sparkfun.com/tutorials/analog-to-digital-conversion) - measurement of an analog voltage and conversion of that voltage into a digital value.

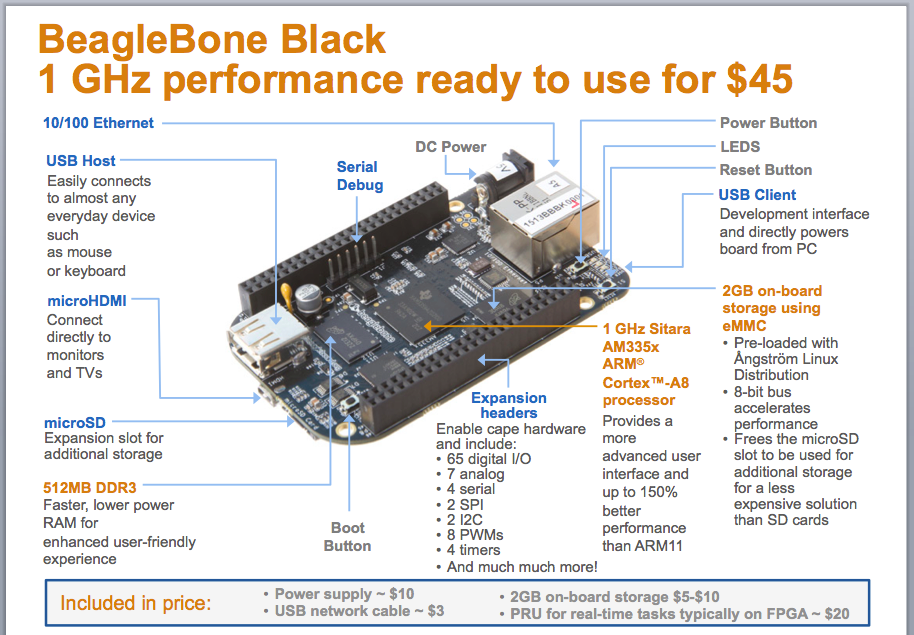
All of the code in this tutorial can be found online, in our [pcDuino Github repository](https://github.com/sparkfun/pcDuino). It’s not a bad idea to check there for any updates to the code since this tutorial was written.





**BeagleBone Black**

The BeagleBoard is a low-power [open-source hardware](https://en.wikipedia.org/wiki/Open-source_hardware)[single-board computer](https://en.wikipedia.org/wiki/Single-board_computer) produced by [Texas Instruments](https://en.wikipedia.org/wiki/Texas_Instruments) in association with [Digi-Key](https://en.wikipedia.org/wiki/Digi-Key) and [Newark element14](https://en.wikipedia.org/wiki/Newark_element14). The BeagleBoard was also designed with [open source software](https://en.wikipedia.org/wiki/Open_source_software) development in mind, and as a way of demonstrating the Texas Instrument's [OMAP3530](https://en.wikipedia.org/wiki/Texas_Instruments_OMAP)[system-on-a-chip](https://en.wikipedia.org/wiki/System-on-a-chip).[[8]](https://en.wikipedia.org/wiki/BeagleBoard#cite_note-edn_take_advantage-8) The board was developed by a small team of engineers as an educational board that could be used in colleges around the world to teach open source hardware and software capabilities. It is also sold to the public under the [Creative Commons](https://en.wikipedia.org/wiki/Creative_Commons)[share-alike](https://en.wikipedia.org/wiki/Share-alike) license. The board was designed using [Cadence](https://en.wikipedia.org/wiki/Cadence_Design_Systems)[OrCAD](https://en.wikipedia.org/wiki/OrCAD) for schematics and Cadence Allegro for PCB manufacturing; no simulation software was used.

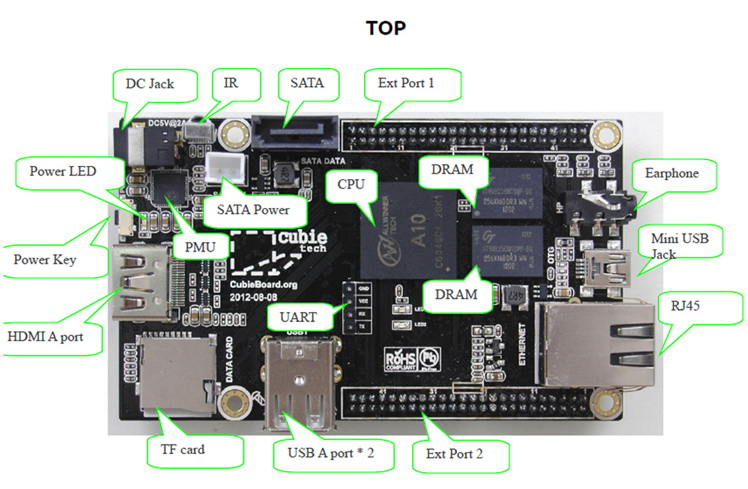


**Cubieboard**

Cubieboard is a [single-board computer](https://en.wikipedia.org/wiki/Single-board_computer), made in Zhuhai, Guangdong, China. The first short run of prototype boards were sold internationally in September 2012, and the production version started to be sold in October 2012. It can run [Android 4 ICS](https://en.wikipedia.org/wiki/Android_Ice_Cream_Sandwich), [Ubuntu 12.04](https://en.wikipedia.org/wiki/List_of_Ubuntu_releases#Ubuntu_12.04_LTS_.28Precise_Pangolin.29) desktop, [Fedora](https://en.wikipedia.org/wiki/Fedora_(operating_system)) 19 ARM Remix desktop, [Arch Linux ARM](https://en.wikipedia.org/wiki/Arch_Linux_ARM), a Debian-based Cubian distribution, or [OpenBSD](https://en.wikipedia.org/wiki/OpenBSD).

It uses the [AllWinner A10](https://en.wikipedia.org/wiki/AllWinner_A1X)[SoC](https://en.wikipedia.org/wiki/System_on_a_chip), popular on cheap tablets, phones and media PCs. This SoC is used by developers of the [lima](https://en.wikipedia.org/wiki/Free_and_open-source_graphics_device_driver#ARM) driver, an open-source driver for the ARM Mali GPU. It was able, at the 2013 [FOSDEM](https://en.wikipedia.org/wiki/FOSDEM) demo, to run [ioquake 3](https://en.wikipedia.org/wiki/Id_Tech_3#ioquake3) at 47 fps in 1024×600.

The Cubieboard team managed to run an [Apache Hadoop](https://en.wikipedia.org/wiki/Apache_Hadoop)[computer cluster](https://en.wikipedia.org/wiki/Computer_cluster) using the [Lubuntu](https://en.wikipedia.org/wiki/Lubuntu) Linux distribution.



Questions:

1. How is Raspberry Pi different from a desktop computer?
2. What is the use of GPIO pins?
3. What is the use of SPI and I2C interfaces on Raspberry Pi?

**Text Book :** Vijay Madisetti, Arshdeep Bahga, “Internet of Things A Hands-On- Approach”, 2014, ISBN : 978 0996025515