PART I: FOUNDATIONAL ASPECTS

Emergence of IoT

CHAPTER 1

"There are more people in the world who make things than there are people who think of things to make."

- Syd Mead

OBJECTIVES

- introduce the concept of Internet of Things (IoT)
- understand the various definitions of IoT and converge to a working definition
- recap past and current technologies that aided in the development of IoT
- describe the fundamental components of an IoT system
- present various game-changing applications that IoT is driving
- understand how technologies in various disciplines are enabling further evolution of IoT
- explain the issues in the development of disparate IoT systems based on a variety of technologies, and discuss the need for standardization of IoT systems at various levels
- overview of the role of various organizations that are actively involved in the standardization process and their activities

OUTCOMES

- understand the historical context and background of IoT and describe the IoT vision
- describe various definitions of IoT given by different organizations and able to synthesize a definition of your own
- describe key enabling technologies that converged in the development of IoT
- appreciate the interdisciplinary nature of IoT
- understand the need for standardization at various IoT layers
- name various IoT standards and the organizations that are involved in developing them

1.1 BACKGROUND AND VISION

The Internet has dramatically changed the way we conduct our daily life and has become the de facto way of communication. It has spread so deeply in our society that a lot of our routine activities are now driven by technology and are highly dependent on the Internet. By using mobile devices, we are able to conduct both personal and official businesses more efficiently. For example, with just a touch on the mobile screen, a host of services are available, such as ordering food, buying shoes, arranging meetings, keeping in touch with people, and buying tickets. Having Internet connectivity has become a way of life and will continue to percolate in many areas, which are previously unimagined.

It has revolutionized many domains and brought in many new applications to the forefront. The last decade has seen an accelerated synthesis of research in several path-breaking technologies, particularly in the areas of semiconductors, networking, and information processing. In addition, there has been a significant drop in the prices of sensors, transmitters, processors, and computing infrastructure. These have revolutionized both the information and communication technologies (ICT) and non-ICT areas. Consequently, a new paradigm has emerged called the Internet of Things (IoT), which aims to make things (physical objects) smart using sensors/actuators and digitally identifiable over the Internet so that these can be seamlessly accessed and controlled remotely. Further, these things have the ability to react in real time to the events in the environment that they exist, and send information to humans as well as other things autonomously to enable them to contextually respond for decision-making.

Gartner, a leading research and advisory company, predicts that by 2020, there will be 20.8 billion IoT devices connected around the world, overtaking the number of personal computers and smartphones. Another estimate by Cisco projects that nearly half of the devices connected to the Internet by 2023 are IoT devices (see Fig. 1.1).

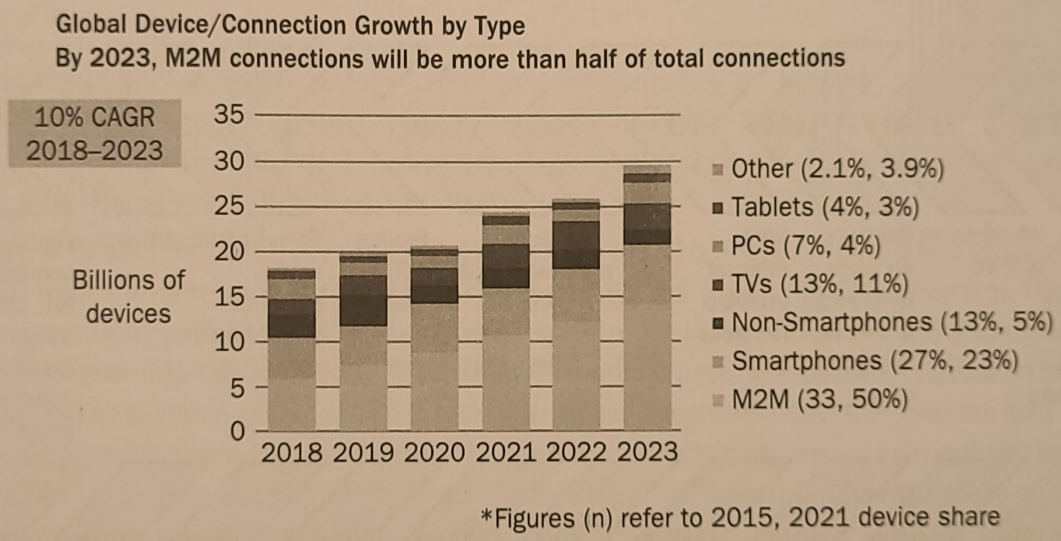


Fig. 1.1 Number of Devices Connected to the Internet by 2023; Half (51%) of Them are IoT Devices

(Source credit: Cisco)

To understand this fast-paced development and adoption of IoT technologies in a better way, it is necessary to travel back in time and look at the key events and technologies that helped shape the current IoT. The historical context of the emergence of IoT is described in the next section.

1.1.1 Background

Although IoT has emerged recently, its roots go back to about two decades. In 1982, a Coke machine at Carnegie Mellon University (CMU), was fitted with micro-switches and was connected to a computer to show how many coke bottles were left in the machine, so that anyone can just ping to it before going

to the vending machine to pick up a bottle. At that time, CMU was part of the ARPANET, universities were also able to find out the number of coke bottles left. Thus, it became one of early examples of an IoT device.

In 1991, ubiquitous computing was proposed by Mark Weiser (Weiser, 1991). In Siemens developed the first Machine-to-Machine (M2M) communication application, used for sales terminals. In the year 1998, the Internet protocol IPv6 was developed and lau by the Internet Engineering Task Force (IETE) to overcome the IP address exhaustion pr of IPv4. IPv4 had only 32-bits, that is, a total of 2^{32}, which are not enough to handle the number of IoT devices currently and for future expansion. IPv6 is 128-bits, so a total of addresses are theoretically possible, which are more than enough to cope with the ever expa IoT devices in the coming years.

In the year 1999, Bill Joy wrote about Device-to-Device communication in his taxonomy of Int (Pontin, 2005). In the same year, the Auto ID centre at MIT was working on the Radio Frequ Identification (RFID) technology for asset tracking and supply chain management, where comp will be able to track their products and reduce operating costs. The team comprising of Kevin Ash Sanjay Sarma and David Brock, developed a way to connect objects to Internet via a RFID tag, ma

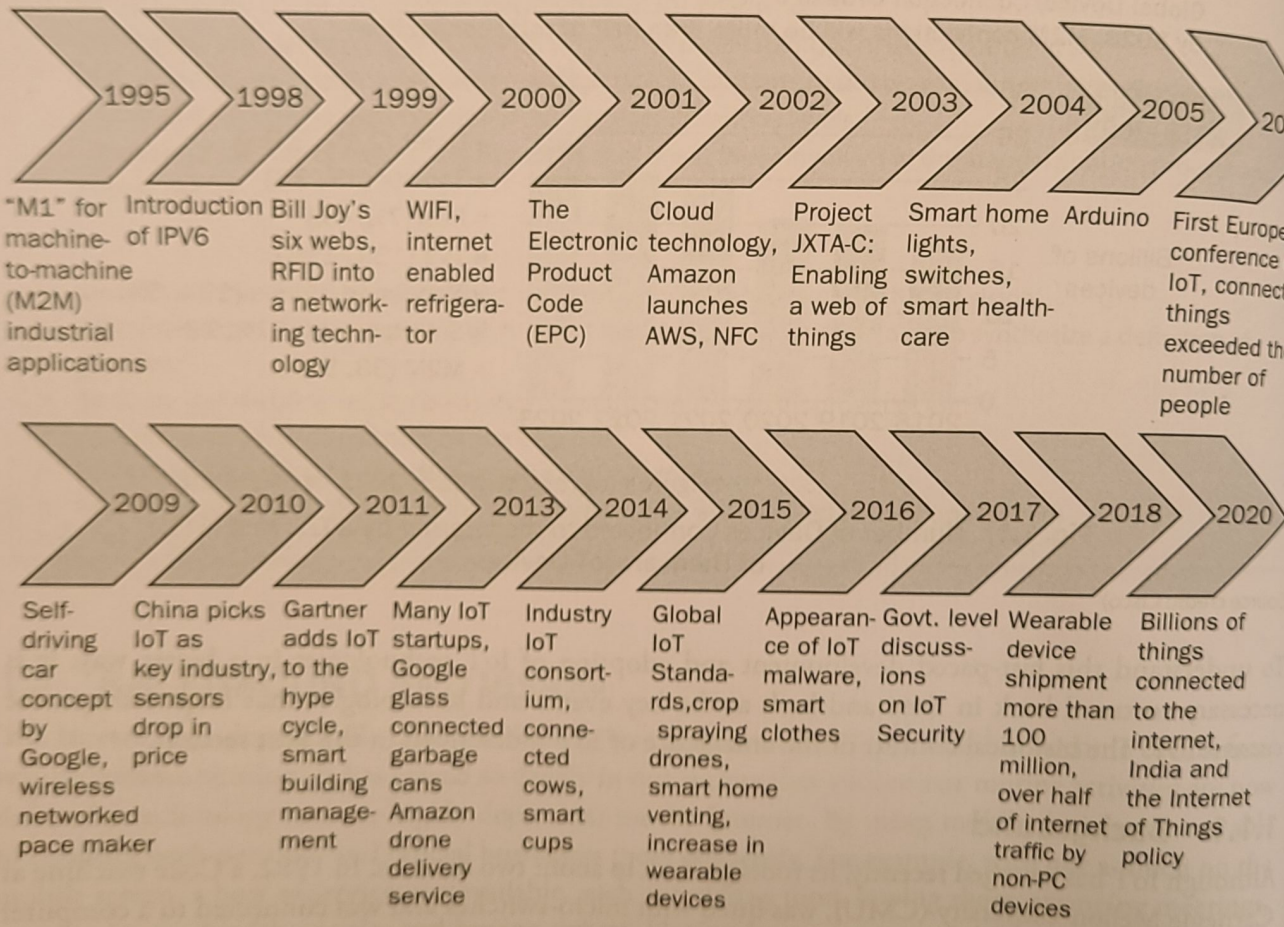


Fig. 1.2 Timeline of IoT Technology Maturity

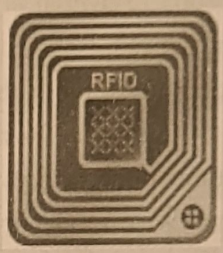
BOX 1.1: VARIOUS AUTO-ID TECHNOLOGIES

Barcodes, RFIDs, QR Codes, and NFC

Barcodes, QR Codes, RFID, and NFC are all systems for conveying large amounts of data in a small format.

Standard UPC-A barcode

Barcodes These are limited to 20 alphanumeric characters and data is stored only in the horizontal direction. UPC consists of 12 numeric digits that are uniquely assigned to each item at the point of sale. The bars represent numbers. These are inexpensive as they can be printed on paper or plastic. The data on the barcodes cannot be rewritten. Barcodes help to identify the type of the item, but not an individual item. The line of sight is important to scan the code properly. Reading items using a barcode could be time consuming as it can only read one item at a time.



Radio Frequency Identification (RFID) It uses electromagnetic fields to read and capture information stored of an object by reading a unique tag attached to it. It can be used for identification, authentication and tracking of items. It can store 1000s of characters.

1. **Passive RFID:** Ultra High Frequency (UHF) Passive RFID Tags operates in 860 ~ 960 MHz and has a read range of about 1 meter. Generation 2 (Gen2) RFID tags can have a read range of up to 12 meters. The new generation of tags, that is, Gen3 and Gen4 are based on new silicone IC and read ranges can reach up to 16 meters.
2. **Active RFID:** It operates in the Ultra high frequency at 433 MHz. These can have a read range of up to 500 meters. Another variant of the active RFID is the 2.45 GHz. Super High Frequency tags. These can have a read range of up to 100 meters. The advantages of the active RFID are: (1) ability to obtain real-time location information, which is useful in any location and tracking applications (2) data on the RFID tags can be rewritten and reused (3) identification of individual item rather than only the item type is possible with it, that is, it gives a count of the number of items that are present in a particular product. An RFID can read potentially 100s or even 1000s of tags simultaneously. Currently, many quick payment systems, anti-car theft devices use 13.56 MHz RFID systems.

Quick Response (QR) codes It is a type of two-dimensional barcode first designed for the automotive industry. The code consists of black modules embedded in a square pattern on white background. It can work both in horizontal and vertical directions and can hold up to 300 times the information that could be encoded in a UPC barcode. QR codes can be quickly read and scanned from any direction.

Near Field Communication (NFC) NFC technology has its roots in RFID. When two electronic devices are brought close (within 4 cm) to each other, a communication is established and data can be exchanged. Normally a 'Tap' is done with the item and the required information is transferred. These are used in many applications such as identity/key cards and credit cards. An NFC-enabled smartphone can be used to make several types of transactions such as payments in a retail store. NFC tags are inexpensive and are becoming quite popular currently.

it a networking technology (Roberti, 2005). A majority of sources attribute Kevin Ashton for coining the term 'Internet of Things' in a presentation he made for Proctor and Gamble (P&G), in which he emphasized the need for a standardized way to make computers understand the real world. The important aspect of this endeavour is connecting the Internet to the physical world (objects connection using RFID) through wireless sensing networks (WSNs). It was a pioneering solution at that time because there was no widespread usage of Internet Protocol (IP) configurations by mobile networks then. See Box 1.1 for more details on various Auto-ID technologies.

In the year 2000 (Fig. 1.2), the Internet-enabled refrigerator was launched by LG. The consumer could get the information about the temperature, freshness of the food, and nutrition information on a LCD screen attached to the refrigerator.

In 2001, David Brook, proposed the Electronic Product code (EPC) to address the issue of tracking a product throughout its lifecycle. He suggested a unified product directory for this purpose. The EPC provides a unique identity to any object. It is designed to be stored on a RFID tag. Bernard Traversat in 2003 (Traversat et al., 2003) published the paper 'project JXTA-C: Enabling the web of things' in which a standard and open source set of protocols for ad hoc, pervasive, peer-to-peer computing were proposed for the Web of Things.

By the year 2004, major transition for smart things has gained and many publications were describing various smart devices in areas such as healthcare, smart bulbs/switches, and transportation. For the first time, Walmart deployed RFIDs on a large scale to keep track of its inventory.

The open hardware-based microcontroller Arduino appeared in 2005. It was developed by faculty members at the Interaction Design Institute (IVREA), Italy. It brought with it a revolutionary change in the embedded computing arena. With its intuitive interface and ease of programming, it enabled non-domain (e.g. non-electronics background) people also to venture into smart devices development. The product ideas were limited only by the resourcefulness and imagination of developer. Several online web portals sprung up offering various sensors, electronic components, and PCB shields that could be easily put together to prototype quickly. Further, several community-based online forums were formed to help developers ideate, implement, and exchange information about innovative IoT devices. In the same year, Ubiquitous Network Societies emerged, almost 15 years after the Ubiquitous term coined by Mark Weiser, who described it as:

(i) A new era in which small non-traditional computing devices will be embedded in everyday objects invisibly at work in the environment around us.
(ii) The applications that are built on the concepts of ubiquitous computing will use these non-traditional computing devices in a large number.
(iii) The computing devices will have sensors to measure various environmental parameters and be able to act independently based on certain predefined conditions.
(iv) The computing devices are mobile and have geographical location and usually the devices in a neighbourhood are often used for a particular task. Hence, communication networks should connect these devices together in an ad hoc and spontaneous manner, and form temporary networks to facilitate anywhere, anytime, always-on communications.

The year 2008 marked the first European conference, which took place in Zurich, Switzerland. During that time, several key industry players such as those from semiconductors, communications, networking, and Internet providers, have made vital contributions to further promote the growth of IoT.

Google launched the ambitious self-driven cars project in 2009, it has heralded a new era in autonomous vehicles area. It is significant from an IoT perspective as it combines the major components of an IoT system, ranging from sensors/actuators that are deployed on various parts of a car (e.g., wheels, bonnet, bumper, and seats) and also those that allow measuring the surroundings around it. These inputs are fed through various models and are continuously learned and used to trigger responses to various events happening inside and outside of the car (e.g., seat belts, temperature, detecting pedestrians, and lanes). Another device that made its entry in the same year was the wireless networked pacemaker that is inserted directly into the heart without surgery and can deliver electric pulses to the heart for its proper functioning.

The fast-paced developments happening in the IoT area prompted China to declare IoT a key industry and included it in its 12th 5-year plan. A national IoT centre was established in Shanghai in the same year. Across the country several initiatives begun on developing a range of standards, industry chains, and applications, which aimed at creating an industry worth more than CNY 500 billion over the coming decade.

Gartner, a leading research and advisory company added the IoT to its hype cycle in 2011. It tracks specific technologies and their advancement through a life cycle from 'technology trigger' to 'plateau of productivity'.

The year 2011 also saw the emergence of the smart building concepts. McGlinn et al., 2010, define Smart Buildings as an environment, which is "able to acquire and apply knowledge about the environment and its inhabitants in order to improve their experience in that environment" (Cook and Das, 2007). Smart buildings are characterized by real-time measurement and monitoring of various building management infrastructure [e.g., Heating, Ventilation, and Air Conditioning (HVAC)], external environment such as weather, noise, light, people movement, and energy pricing, and optimize energy consumption and ambience in the building. This can be achieved by IoT devices measuring various parameters in and outside the building. Such real-time data can be used to perform smart analytics (e.g., Edge analytics) and trigger intelligent control mechanisms.

Buoyed by the great potential of IoT and its wide commercial applications, several start-ups began to emerge in the year 2013. Google glass also appeared in the same year. However, it was discontinued in 2015 due to privacy and safety concerns (a new version of Google glass is launched for enterprise applications in 2018). Amazon's Drone-based delivery prototype was announced in 2013 and named it 'Prime Air'. Due to restrictions by Federal Aviation Administration (FAA), the project could not take off immediately. The first delivery happened in UK in 2016 as a proof of concept. IoT application for garbage management was also demonstrated in the same year called the connected cans that can send an alert when the garbage can is full so that the pickup can be planned by the garbage-collecting trucks. The industrial Internet Consortium (IIC) was founded in 2014 to "accelerate the development, adoption and widespread use of interconnected machines and devices and intelligent analytics" (IIC, 2018). Several industries in areas of Energy, Healthcare, Manufacturing, Mining, Retail, Transportation, and Smart Cities are involved in IIC.

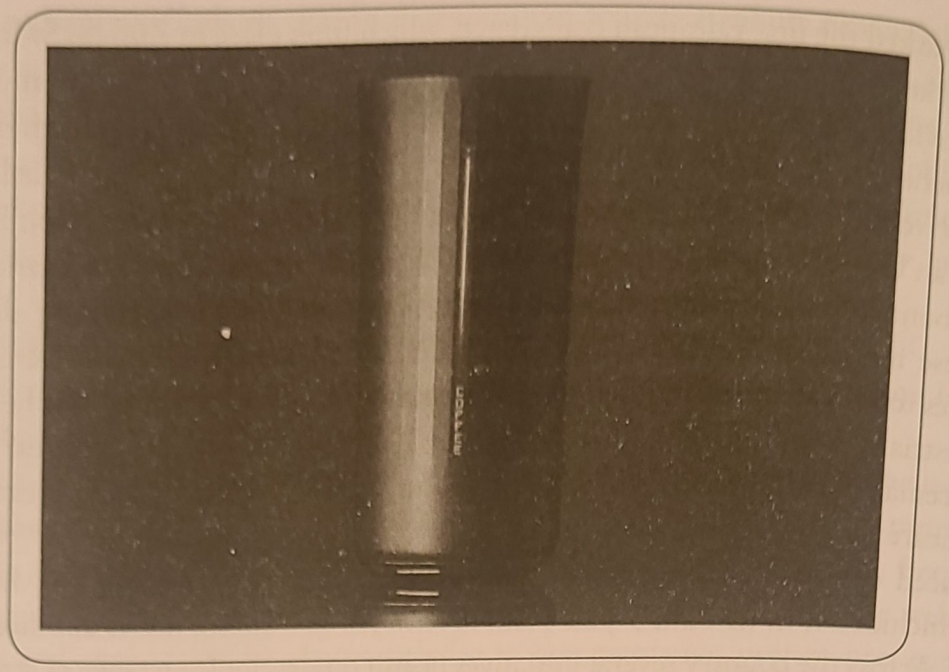


Fig. 1.3 IoT-based Smart Cup Capable of Sending Nutritional Information to a Smart Phone

Connected cow was an interesting application that appeared in the same year. It has transformed t dairy industry by enabling the dairy companies to monitor individual cows by tagging with RFI and other health monitoring sensors and transmit the information in real time to help monitor heal and other location specific attributes. This led to more focussed treatment for diseases (if any), ar also provide optimal nutrition, thus boosting the milk yield. Currently, many companies includir Microsoft, Huawei, and Fujitsu are involved in providing IoT-based solutions for this purpos A 13-ounce smart cup named 'Vessyl' made its first appearance around this time (see Fig. 1.3). An beverage poured into it is recognized and the related nutritional value is displayed on the smartphone Further, it can also keep track of the drinking habits of the user and provide that informatio via a smartphone.

The year 2015 saw the emergence of global IoT standards with a meeting in Geneva, Switzerland A group called the IoT global standards initiative (IoT-GSI), which is part of International Telecommunication Union (ITU) was formed whose aim is to promote a unified approach for developing standards for IoT to enable it to scale at the global level.

The year 2015 also saw the emergence of drones in agriculture. Drone-based sensing was demonstrated in the areas of crop health monitoring, irrigation equipment monitoring, and weed detection. The ability to monitor at high temporal frequency made this technology very attractive to the farmers. Another area that saw the use of IoT is the smart home venting systems that allows adding room-by-room smart vents that can control the central air conditioning installations. This enables to distribute the airflow from rooms that are over conditioned to those rooms that need more air. All these can be achieved by IoT devices and smartphone applications. The wearable devices market also emerged in the year 2015 (see Box 1.2).

BOX 1.2: WEARABLE DEVICES

Wearable technology is related to the development of devices that are worn directly on body or on the user's clothing, which helps to track a user's data related to health and fitness, location, emotions, gestures, and other reflexes of the human body. The wearable technology can help the user to continuously record, monitor, and analyse internal and external stimuli and react based on certain preset controls.

A wearable device can have sensors, computing architecture and display. Depending upon the parameter (e.g., pulse, gesture, etc.) that is being tracked, the wearable may use the computing on its device (very limited) or delegate an external device such as a smart phone to do the computing. Similarly, the display on a wearable depends on the amount of information that needs to be shown. If it is limited, then it is displayed on the wearable itself (e.g., smartwatches); otherwise, it is shown on a smartphone, nearby display, or ear buds (verbal communication).

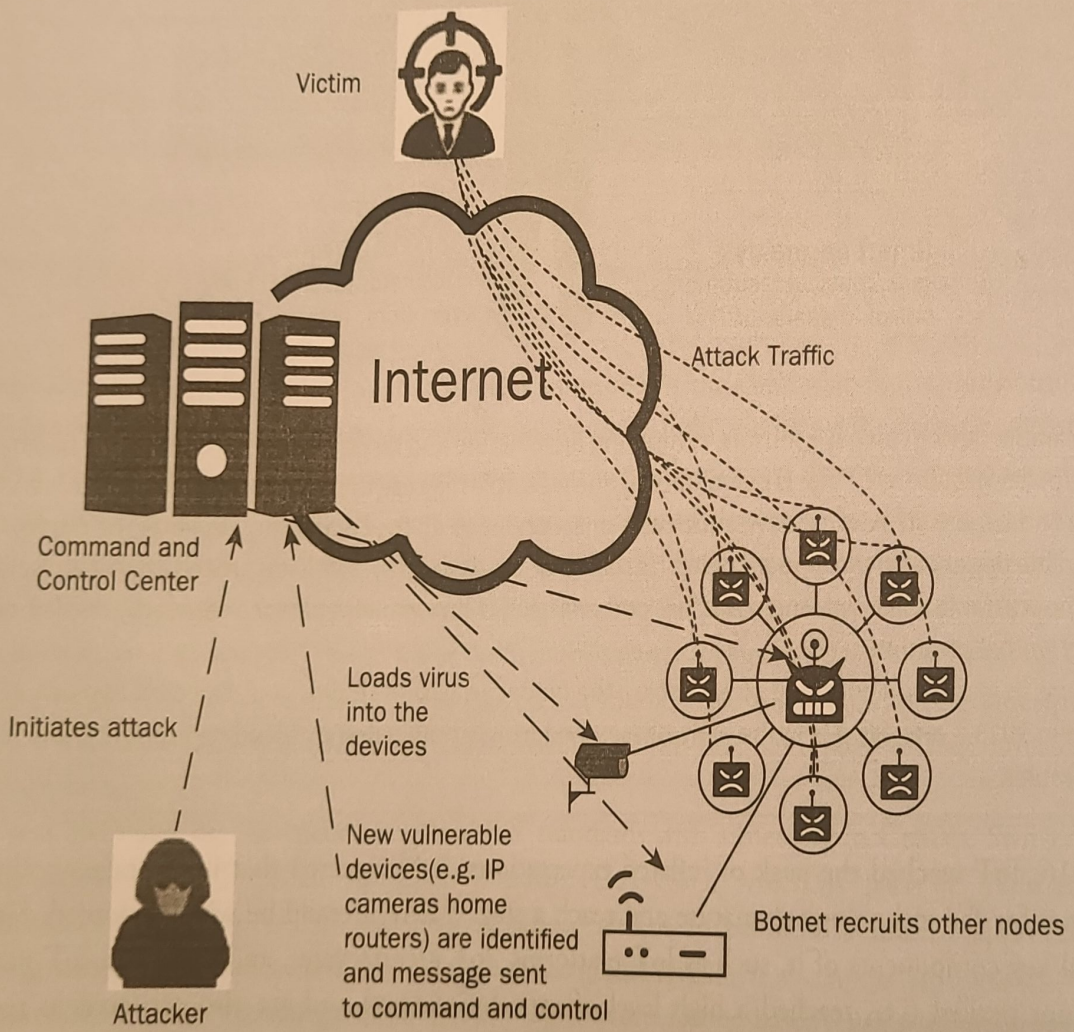
Due to increasing adoption of wearables, the global wearables shipment is expected to reach 2.5 billions in 2023. Currently it is in the early phases, and mainly dominated by healthcare, activity, and fitness wearables.

In 2016, IoT reached the peak of inflated expectations. It is expected that it will take another 5 to 10 years for the technology to mature and reach a stage where it could be widely adopted. Similarly, several key components of it, such as IoT platforms, IoT architectures, and wide-area IoT networks, have not peaked (i.e., reached a high level of expectations yet) and are also expected to reach the plateau of productivity in another 5 to 10 years. Whereas, other components such as IoT services, IoT edge architecture, IoT for customer service, and messaging platforms such as MQTT are expected to quickly reach wide adaptability in a span of another 2 to 5 years. Another key component for real-time decision-making in IoT systems is the event stream processing (although developed independently of IoT) has reached high expectation levels for its application in IoT edge analytics. Many event stream processing engines are emerging, but it is expected to take another 5 to 10 years, for it to be completely adopted and integrated with IoT edge analytics. Another, closely related innovation are the low-cost development boards (e.g., Arduino, Raspberry Pi, etc.), which have played an important role in enabling users from diverse disciplines to work in IoT. These low-cost development boards and related

programming interfaces have reduced the complexity of developing certain class of electronic products, particularly those that are useful for IoT.

In August 2016, the Mirai malware appeared and targeted under-secured networked devices running Linux OS systems into bots and recruited them to be part of a larger botnet network that can trigger large-scale attacks. The devices that were affected by this malware were mostly IP cameras and home routers (see Box 1.3). Smart clothes that monitor the wearer's physical condition began to appear in the wearable segment of IoT in 2016 (see Box 1.2).

BOX 1.3: MIRAI MALWARE CONVERTED INTERNET OF THINGS INTO BOTNET OF THINGS



In October, 2016, Paras Jha, Josiah White, and Dalton Norman used their Mirai botnet, which is a piece of malware that gets installed on under-secured IoT devices such as IP Cameras, home routers. These then become bots and are recruited by other bots and form a network known as a botnet. The attacker then initiates a Distributed Denial of Service (DDoS) attack. In this case, they attacked the domain name server (DNS) Dyn causing the shut down of a number of major websites including Twitter, Reddit, and the New York Times. Subsequently they were indicted by a court in Alaska and have pleaded guilty to charges that carry a sentence of up to 5 years in prison.

(Contd)

Box 1.3 *(Contd)*

The main functionality of the Mirai malware was to search for under-secured IoT devices by scanning wide-ranging IP addresses and finding those devices, which can be easily accessed based on a dictionary of user-name/passwords (usually the default factory set credentials). Subsequently install the virus and make it as a part of a bigger network. Next, based on the instructions received from the command and control centre, use this network of bots to launch DDoS attacks on specified targets.

The security aspects of IoT received wide attention in 2017 owing to the large-scale attacks that happened in the preceding years. The US government enacted a bill called the 'Internet of Things Cybersecurity Improvement Act 2017', which basically mandates all the IoT devices that are sold to the government to have security measures and these include wearables, sensors, IoT tools, etc. Further, compliance in terms of including industry standard protocols, passwords that can be changed (controlled) by the users and do not have any known vulnerabilities (as defined by the National Institute of Standards (NIST)) (NIST, 2018).

In 2018, the number of wearables that were shipped crossed 100 million units. IoT is providing a great market opportunity for companies involved in IoT hardware manufacturing, Internet service providers, and application developers. The global IoT market is expected to have a compound annual growth rate (CAGR) of nearly 27 per cent from 2018 to 2024.

The IoT market is fast gaining ground and many industry leaders such as Google, Amazon, IBM, CISCO, Samsung, Intel, and Apple have announced new products in the IoT landscape. Along with these, a plethora of start-ups are testing ground. The industry is investing mainly in the areas of manufacturing, healthcare, transportation, consumer electronics, smart cars/fleets, home automation, and utilities.

By the year 2020, billions of things are expected to be connected to the Internet.

1.1.2 Vision of IoT

The vision of IoT can be perceived from four major perspectives (Fig. 1.4):

Things perspective The persistent and reliable availability of anything that is of interest to the user, which could be connected in anyway using a variety of technologies. Further, these connected things should be available to be accessed at anytime and from any location. This is similar to the ubiquitous computing concept that was there much earlier before the advent of the IoT.

Standards and semantics perspective Access of the things should be seamless, which is possible only if well-defined standards are adhered to by the providers of these things as they are highly heterogeneous in nature. In addition, the things should be interoperable, that is, the ability to integrate data from heterogeneous IoT devices is a must, otherwise, many IoT applications will be very specific to a particular application domain and cross-domain integration becomes highly challenging. Hence, standards are necessary in every component of the IoT architecture.

Internet perspective The aforementioned two are possible only if the things are able to communicate with people and people with things (people 2 things, i.e., P2T), and further, among things themselves (things 2 things, i.e., T2T), so that certain level of autonomous decision making is achieved. Hence, the web enablement or web addressability is one of the prime goals of IoT.

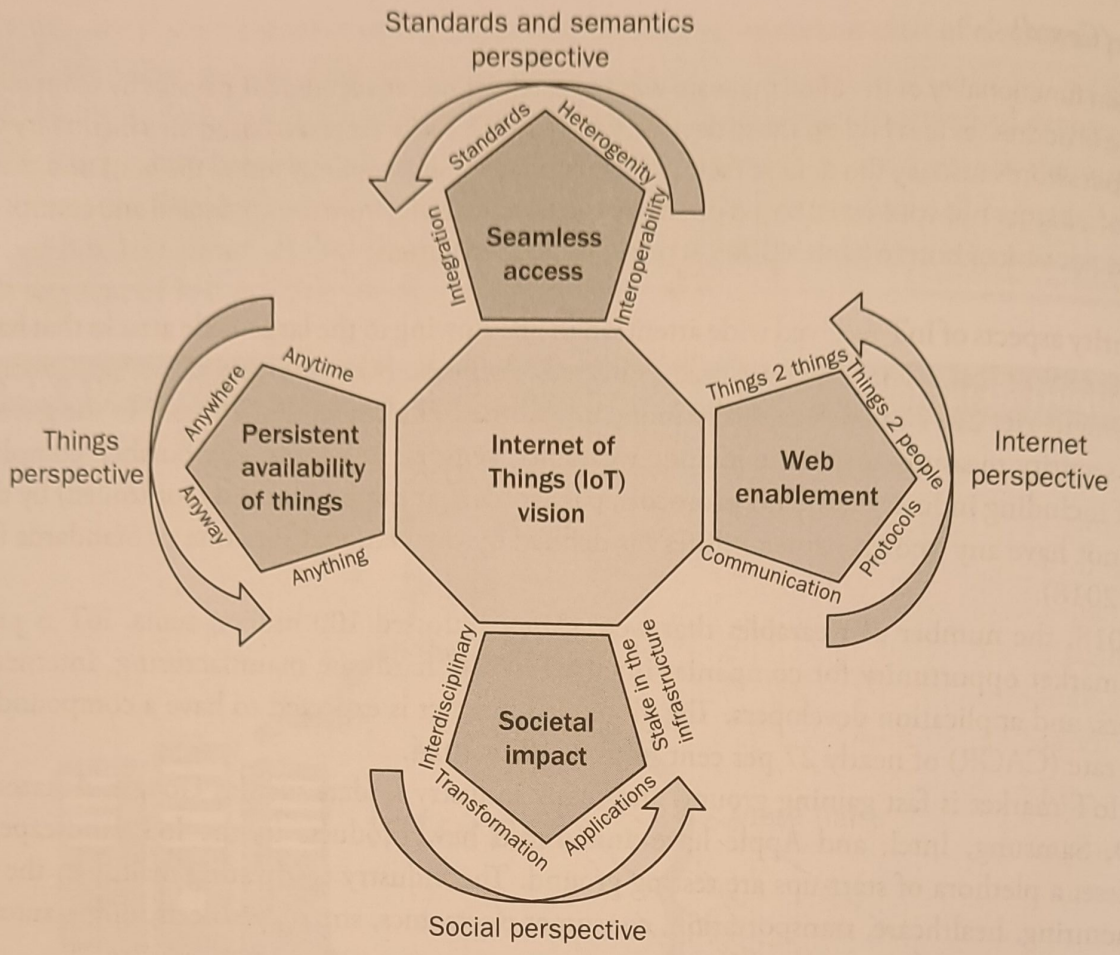


Fig. 1.4 The Vision of IoT from Four Different Perspectives

Social perspective The societal impact of the IoT is primarily driven by the acceptance of the users and their perceived value that it would bring to their lives. It is envisioned that the IoT infrastructure will be owned by the stakeholders in some form of democratic process, who will develop, maintain, and take in directions that are congenial for their growth. The society will be benefited by a spectacular transformation of traditional services into IoT-based smart services that provides applications that are much easier to use and provides far-reaching benefits. These IoT services also operate in an interdisciplinary mode, where they can be easily integrated and a common set of services will come together and cooperate in real time to address a particular problem. Several applications such as monitoring air pollution, improved water conservation, and increased production of food grains.

1.1.3 Various Definitions of IoT

The understanding of IoT is being approached from various perspectives and based upon a particular domain and its view of the components involved in the IoT systems. There may be specific emphasis on particular modules that are important in that domain. Hence, there is no global consensus on an overarching IoT definition. Indeed, there have been efforts by organizations such as IEEE to lead an initiative for developing a definition of IoT. Following are some selected definitions of IoT.

1.1.3.1 IEEE Definition

IEEE led an initiative in 2015 on developing a definition for IoT. In the special issue of IEEE on Internet of Things, IoT is defined as:

"A network of items, each embedded with sensors, which are connected to the internet."

1.1.3.2 ITU Definition

The international Telecom Union (ITU) defines IoT as:

"A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies."

1.1.3.3 IETF Definition

The Internet Engineering Task Force (IETF) is playing a crucial role in the development of several standards for IoT. They define IoT as:

"IoT will connect objects around us (electronic, electrical, non-electrical) to provide seamless communication and contextual services provided by them. Development of RFID tags, sensors, actuators, mobile phones make it possible to materialize IoT which interact and cooperate each other to make the service better and accessible anytime, from anywhere."

1.1.3.4 OASIS

Organization for the Advancement of Structured Information Standards (OASIS) is a non-profit consortium that drives the development, convergence, and adoption of open standards for the global information society. It describes IoT as:

"System where the Internet is connected to the physical world via ubiquitous sensors."

In addition to standard bodies such as IEEE, IETF, and ITU, there are several projects that have also given definitions of IoT such as:

1.1.3.5 IoT-A

Internet of Things Architecture (IoT-A) developed an *Architecture Reference Model (ARM),* which addresses the issues of heterogeneity in IoT-related technologies. IoT-A defines it as:

"It can be seen as an umbrella term for interconnected technologies, devices, objects and services."

1.1.3.6 Texas Instruments

Texas instruments in its paper on evolution of Internet of Things defines IoT as:

"The IoT creates an intelligent, invisible network fabric that can be sensed, controlled and programmed. IoT-enabled products employ embedded technology that allows them to communicate, directly or indirectly, with each other or the Internet."

1.1.3.7 Gartner

"The Internet of Things (IoT) is the network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment."

1.1.4 Working Definition

Based on the aforementioned definitions, it is useful to have a working definition whose spirit will be used as the foundation for the rest of the contents of this book.

IoT = Sensing + Communication + Computation + Web Application

"Making things (objects) to sense the environment in which they exist, communicate, access, actuate, and process data autonomously with other things in a network, and also with humans via web applications."

1.1.5 Key Enabling Technologies

The enabling technologies of IoT are distributed in several layers of the IoT systems. The technologies at each of these layers are specialized and serves some key purposes for the functioning of that layer. These can be categorised into four major categories (Fig. 1.5).

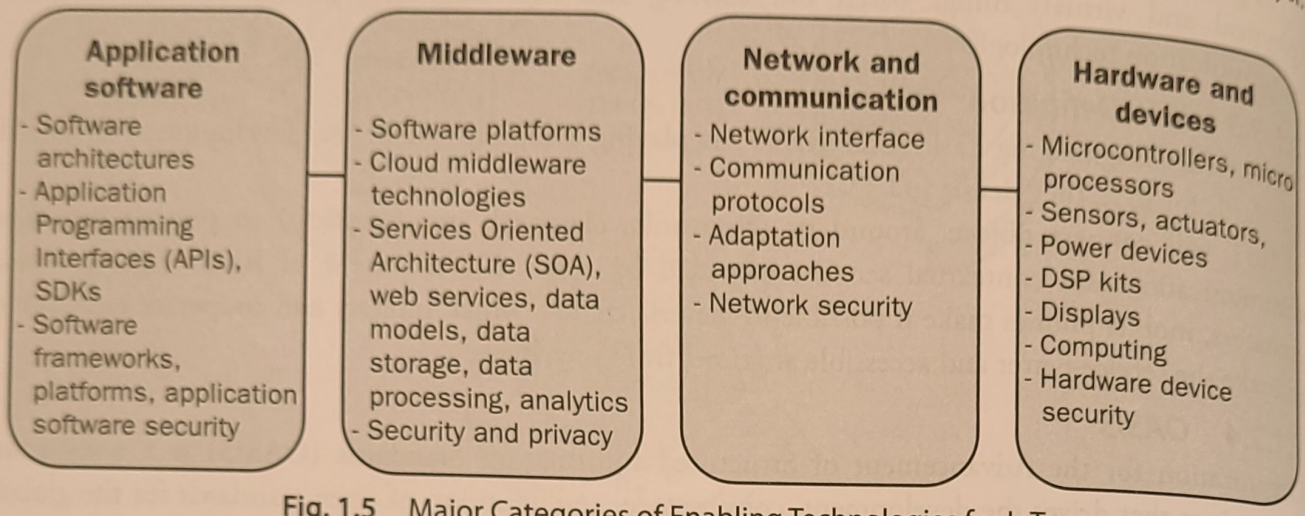


Fig. 1.5 Major Categories of Enabling Technologies for IoT

1.1.5.1 Applications and Software

Currently, the IoT technology is being applied in a variety of fields due to its immense potential to create more smarter ways of performing various tasks in those domains. IoT applications have emerged in many domains such as healthcare, transportation, smart cities, industry, and energy management. Most of the traditional approaches in these domains are getting transformed by using the new IoT system functions to build practical applications. The main enabling technologies for IoT in this layer are:

Application Programming Interfaces (APIs) The application programming interface, or API enables to integrate the 'things' of IoT and act as a glue between the IoT devices and the network (e.g., Internet). It provides an interface for other applications to interact with your application without having direct access. From an IoT perspective, device level APIs can be used to enable applications to communicate with devices. For example, Google IoT product NEST is a smart Thermostat that can control via a mobile phone the temperature in a home/office setting. While the manufacturer of NEST has provided the core applications to interact with the NEST device, they also provided an API. Using the API, developers can connect and use the NEST data in their own applications and sell them. Development of APIs that allow to securely expose connected IoT devices to users are one of the key enabling technologies for IoT. Many IoT-based APIs are available.

Software Development Kits (SDKs) Software development kits have pre-built functionality to make them easier to work than an API thus making them easier to integrate in an application. For example, the amazon IoT SDK allows hardware devices to connect device gateway and device shadow. Similarly, the Azure IoT has three main IoT SDKs including device SDKs, service SDKs, and Gateway SDK.

1.1.5.2 Middleware

The purpose of an IoT middleware is to function as a mediator between the hardware (IoT devices) and the application layers. Its primary tasks include collection, aggregation, filtering, and processing of data from heterogeneous devices over a variety of protocols and network topologies. Some examples of IoT middleware are Google Fit and Apple's HealthKit, which allows users to access and control their fitness data generated by variety of devices (e.g., wearables) and mobile applications. Xively is another IoT-based middleware that allows sensors to connect to its platform easily and store the data persistently, so that it can be seamlessly retrieved anytime. Node-RED is an open-source IoT middleware platform from IBM, which provides a visual tool to compose an IoT application by simply dragging and dropping its various components.

Some of the enabling technologies in these areas are:

IoT platforms An IoT platform is designed to seamlessly manage any kind of connected device, irrespective of the underlying protocols. The IoT platform can:

- Connect various components of the IoT system such as IoT sensors and devices
- Enable communication between different hardware (e.g., edge hardware) and software irrespective of the underlying protocols
- Handle authentication, security, and privacy
- Device management functions such as monitoring, troubleshooting and administration
- Enable aggregation, analysis and visualization of data

These IoT platforms are in between the data collected at the IoT gateways and the end application. It is estimated that the IoT platform market share will cross over $22 billion by 2023.

Data storage Cloud-based data storage is the most common form in IoT rather than storing it on the premise of where the data is collected. Although, it increases the cost, the main advantage is the connectivity it provides to the users. Users can access the data from the device, anytime and from anywhere.

Data processing One of the key functions of an IoT middleware is the ability to manage the huge data that is being generated by the IoT devices. This kind of data can be considered as big data. The attributes of big data such as Velocity, Volume, Variety, Value, and Veracity are present in the data generated by the IoT systems. Various IoT Middleware provides the ability to do analytics on both continuous streaming data as well as data that is stored. A publish/subscribe kind of middleware can provide functionality for continuously processing of the data using various data processing functions such as preprocessing, filtering and data mining approaches.

Edge/Fog computing Various other forms of computing technologies are becoming key enablers in the middleware domain such as Edge computing, which facilitates the computation to be performed on computing infrastructure available close to the devices and sending the result directly to the relevant application. There is a no collection point or cloud in this type of computing. Fog computing is another emerging approach, where the computation is placed at the edge of a network.

Security and privacy Due to the vast amounts of data stored and managed by the IoT middleware, the main concern is of security and privacy. Many cloud middleware systems provide authentication and authorization services. Recently, new security algorithms and tools have been developed to address the requirements of low powered IoT devices.

1.1.5.3 Ubiquitous Connectivity

It is expected that billions of devices are going to be connected to the Internet in the near future. In addition to this connectivity, the devices themselves need to communicate with other devices as well as various other forms of computer systems (e.g., Gateway). As these devices are energy constrained, the optimization of energy during wireless communication is of utmost importance. These communication technologies can also be distinguished in terms of the system performance, frequencies, Medium Access Control (MAC) scheme, and standardization process (i.e., open vs proprietary).

The key enabling technologies in this area can be classified into proximal range, short range, short/medium range, medium range, and long range.

Proximity Each IoT application has its own requirement depending upon the environment in which the IoT system is deployed. The communication protocols such as Radio Frequency Identification (RFID), Near Field Communication (NFC) work in the proximity range, that is, 0 to 10 meters. Several applications particularly in the retail sectors uses NFC for bill payment at checkout by just such as payment at retail checkout by tapping a credit card with an item having an NFC tag.

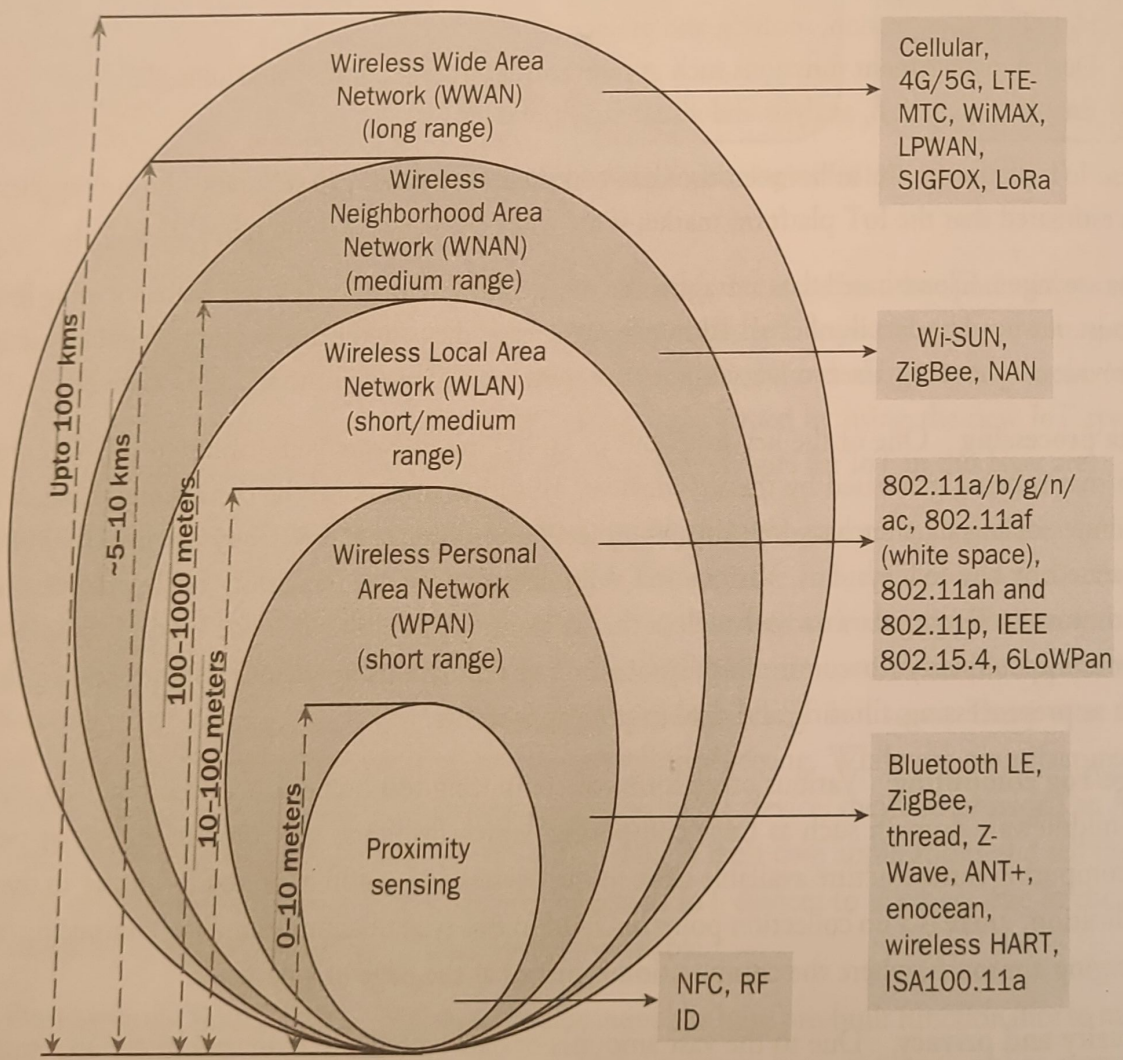


Fig. 1.6 Various Types of Short-, Medium-, and Long-range Connectivity in IoT

Short range In the short-range communication space, applications such as Wearables, Home automation, and Healthcare are popular. Key enabling technologies for this type of communication are Wi-Fi, Z-Wave, ZigBee, IrDA, Bluetooth, Bluetooth Low Energy, and Radio Frequency Identification (RFID) (see Fig. 1.6).

Short-to-medium range In this range of communications, Wi-Fi is a popular technology for many years now. IEEE developed a set of five (designated as a, b, g, n, and ac) media access control (MAC) and physical Layer (PHY) specification for wireless local area network (WLAN) called the IEEE 802.11. These are abbreviated as BGN, ABGN, and A/B/G/N/AC in the specifications for wireless routers, Wi-Fi access points, and Wi-Fi in portable devices. However, since these Wi-Fi protocols have fairly large energy consumption, they are not very useful for IoT devices due to their low-power requirement. To overcome this issue, duty cycling (i.e., keeping the chips in sleep mode for most of the time) and other energy-harvesting methods are being developed. A low power Wi-Fi standard emerged called IEEE 802.11ah. Another enabling technology that is specifically developed by IETF for IoT is the 6LoWPAN, which defines the mechanisms for transmitting IPv6 (128-bit Internet scheme that offers about 3.4×10^{38} unique addresses to accommodate the requirements of the IoT) packets on top of IEEE 802.15.4 networks which defines low-data-rate, low-power, and short-range radio frequency transmissions for wireless personal area networks (WPANs).

Long range There are two main options available for IoT system developers to enable long-range communication in their systems.

Cellular These technologies operate in the licensed spectrum. The key technologies in this space are GSM, WCDMA, 3G/4G/5G, LTE-MTC, and WiMAX providing high-quality voice and data services. The 3rd Generation Partnership Project (3GPP) is a collaboration between groups of telecommunications standards associations that developed Narrowband IoT (NB-IoT), which is a Low Power Wide Area Network (LPWAN) radio technology standard.

Unlicensed low power wide area network The technology in this area uses the unlicensed spectrum and mostly proprietary in nature. Technologies such as SIGFOX and LoRA are popular for machine-type communication (MTC) applications addressing the ultra-low-end sensor segment.

1.1.5.4 Hardware and Devices

Miniaturization and composability Novel hardware developments are enabling the development of ultra-compact wireless. Advancements in the miniaturization of the hardware mainly through the use of the microelectromechanical systems (MEMS) technology is enabling the development of a new generation of devices that are ultra-compact and have high computing ability. Further, nano-electromechanical systems (NEMS)-based sensors are miniaturizing the sensors to nanometres size. Wearable medical devices that are almost invisible to other individuals are currently available. The Moore's law states that the density of transistors on silicon chips doubles every 2 years. This is evident from the fact that today's devices (e.g., smartphones) have the computing power of yesteryears high end computing systems and even supercomputers. Further, the increased ability to put together complex systems from simpler components is enabling the development of revolutionary products in many areas.

High durability The IoT sensors are expected to work in harsh situations. In many field-based applications, these sensors are deployed in open environments, exposing them to the elements of weather for years. Some of these sensors are explicitly required to withstand harsh extreme environments such as extreme temperatures, vibration ratings, and dust and liquid resistance. Hence, durability of these sensors is of great concern.

Improvements in System on Chip (SOC) architectures Some key advancements happened in the area of SOC architectures specifically designed for IoT devices such as application processors (high end) (are usually based on technologies adapted from mobile phone/tablet architectures) microcontrollers (low end), and smart analogue.

Lower costs One trend that is driving greater adoption of IoT is the lowering of the cost of the sensors. It is estimated that the average cost of the sensors will drop to $0.38 by 2020, down by $0.92 as compared to 2004. This is helping to sense and acquire more data from a variety of environments and develop more data-driven intelligent applications than before.

1.2 IOT AS A DISRUPTIVE TECHNOLOGY

The term disruptive technology was first coined by Harvard professor Clayton Christensen, he later named it as disruptive innovation. The reason it is called as disruptive is due to its impact on an existing business model and the current system/society.

A disruptive technology (a hardware, software, networking, etc.) has the potential to replace an existing technology or a well working system that is already in place. From a product perspective, it could be something that begins small and steadily moves up the market and eventually becomes a threatening competitor to the existing products, and may eventually replace them.

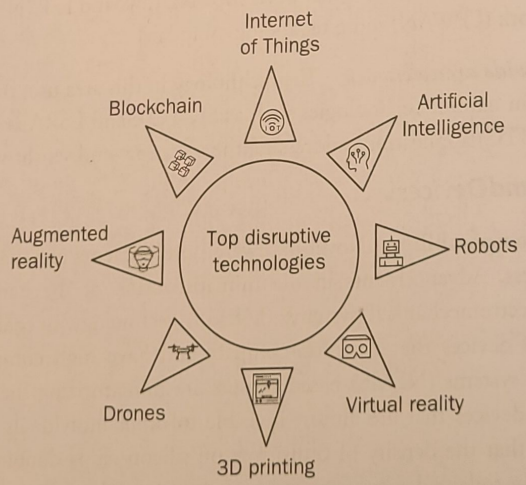


Fig. 1.7 Internet of Things (IoT) Along with Other Disruptive Technologies (Projection for the Year 2020)

Currently, IoT is considered as one of the top disruptive technology along with other technologies such as artificial intelligence, robotics, and virtual reality (see Fig. 1.7). Section 1.2.1 discusses some selected areas that IoT is already playing a crucial role and acting as disruptive technology.

1.2.1 Motivating Scenarios

1.2.1.1 Home and Personal Space

The personal space IoT centres on a person's space. It includes all objects implanted or wearable by the person such as implanted sensors, smartwatches, Google glasses, ECG sensors, and smartphones. It also includes all fixed or mobile objects and devices that come into contact (or reachable) with wearable objects on the person. Devices are reachable when they are within the wireless transmission radius of one another.

Smart buildings IBM is building a solution with the Watson IoT platform and IoT-enabled sensors for tracking every asset in the building using that information in their IoT platform. This will facilitate the owners of the building to understand, monitor, and control of installations such as Heating, Ventilation, Air Conditioning (HVAC) systems, to enable to monitor remotely. The burden of facility management is overtaken by smart sensors that capture every pulse of this infrastructure and to enable remote diagnostics and analysis.

Smart elevators KONE, the elevator company, is trying to understand how people are using the elevators. By using sensors, they are assessing how people move through buildings and estimating how much time can be reduced for the elevator wait. They conclude that even 2 or 3 minutes reduction in the waiting time will make a huge difference in moving people to their relevant floor.

1.2.1.2 Social

Food security In February 2018, a meeting at Rome was organized by the Food and Agriculture Organization (FAO) of the United Nations (UN). During that meeting Vicente Muñoz, Chief Internet of Things Officer, Telefónica said "The future of agriculture hinges on the adoption of technologies such as the Internet of Things (IoT), Big Data and Artificial Intelligence". Further, FAO's director General José Graziano da Silva was of the opinion that the greatest challenges currently faced by humanity is in their fight against hunger, poverty, and effects of climate changes in agriculture, further he said:

"Access to reliable information, including that related to changing weather patterns, is essential to empower farmers, especially those who live in developing countries."

To address the above issues, investment and integration of new technologies such as IoT that will enable farmers to connect with real-time information (e.g., Agro-meteorological) about their farms and facilitate the use of simple and intuitive tools that are data driven to decrease uncertainty and mitigate risk. IoT's contribution is this sector is gaining wider acceptance. Several areas are currently getting transformed by the use of smart sensing systems for monitoring and management of various field tasks and parameters such as soil health, irrigation scheduling, nutrients, and early pest and disease prediction and warning.

Reducing food wastage A Chicago-based tech start-up, Ovie, developed a smart food storage system based on the IoT concept that will eliminate waste and change the way people eat, save, and shop for food. The system tracks the food items in the fridge and sends reminders to eat those before getting spoiled.

Water ATM Piramal Sarvajal, a mission-driven social enterprise, is committed to leveraging technology to bring community-level safe drinking water to the underserved. The organization has developed and implemented innovative market-based drinking water solutions in 16 states in India. Their infrastructure includes remotely monitored water purification units and solar powered, cloud connected water kiosks called water ATMs (see Box 1.4).

BOX 1.4: WATER ATM FOR DISPENSING SAFE DRINKING WATER

Peeth is a village in the heart of the Aravallis in Rajasthan. It is one of the most backward districts of the country and faces a serious water shortage. In Peeth, 84% of the population gets its drinking water from the local well, polluted with dangerously high levels of fluoride and other heavy minerals. Surveys showed that although the locals were aware of hazards of drinking polluted water, but the only alternative was a single private drinking water supplier, who charged ₹2 per litre and only catered to 40 families. Thus, the villagers welcomed the Sarvajal Kendra, as a much-needed solution. Today, the facility serves more than 200 families daily, ensuring they get safe water delivered at their doorstep.

Adapted from www.sarvajal.com; www.downtoearth.com; www.thehindu.com

1.2.1.3 Healthcare

Currently, IoT in Healthcare is being adopted in many areas. It can not only improve the existing healthcare systems, but also enable transformative ways (see Box 1.5) in which the patient gets treatment and care. It is estimated that the global IoT healthcare market to reach over USD 160 billion by 2020. The concept of 'Connected medical devices' is ushering new era in personal fitness and wellness area. Some of the use cases are:

- Remote diagnosis and follow up monitoring
- Memory disabilities monitoring
- Early intervention for detection of critical signs
- Monitoring patient fall
- Timely medicine alerts and enhanced drug management
- Healthcare assets monitoring and tracking

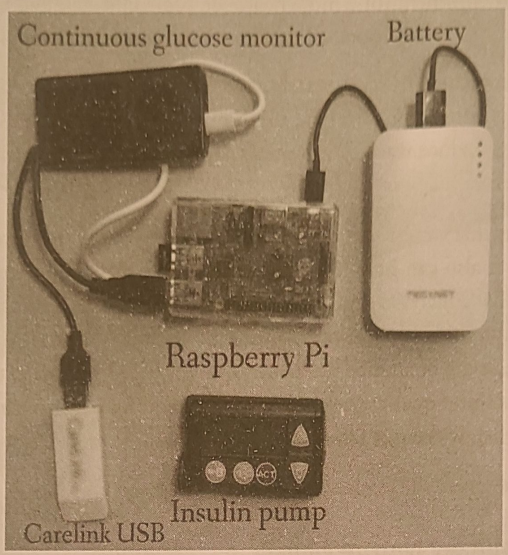
BOX 1.5: OPEN ARTIFICIAL PANCREAS SYSTEM DEVELOPMENT

Dana Lewis and her husband Scott Leibrand have hacked Dana's CGM (continuous glucose monitor) and her insulin pump. Using the data feed from the CGM and a Raspberry Pi computer, their own software completes the loop and continuously alters the amount of insulin Dana's pump delivers. Its success led to the unveiling of the open artificial pancreas system project. "The Open Artificial Pancreas System project (#OpenAPS) is an open and transparent effort to make safe and effective basic Artificial Pancreas System

(Contd

Box 1.5 *(Contd)*

(APS) technology widely available to more quickly improve and save as many lives as possible and reduce the burden of Type 1 diabetes."



Adapted from www.diyps.org; www.healthline.com; www.itas.kit.edu; www.openaps.org

1.2.1.4 Environmental

Saving critical species The International Union for the Conservation of Nature (IUCN) maintains a database of the extinction risk of animal species called the IUCN red list. It reveals that the number of threatened species rose globally from roughly 10,000 in 2000 to over 25,000 in 2018. In the wake of such a revelation, it is necessary to take urgent action to protect these animals from extinction. Many organizations around the world are adopting IoT-based technologies to help identify, track, and protect them. The ruggedness of the terrain, remoteness, and real-time information gathering are some of the aspects of the problem that IoT is able to address. For example, in South Africa, IoT devices enabled collars are sending location and heart rate of Rhinos, which is helping the authorities to monitor them and immediately send rescue teams if they find the animal in distress.

Monitoring and reducing air pollution The effects of air pollution are causing serious problems worldwide. At the individual level, poor air quality is leading to several debilitating conditions in humans such as asthma, attacks, lung cancer, heart diseases, and chronic bronchitis. According to the American Association for the Advancement of Science (AAAS), it is the world's fourth reason for deaths. IoT-based air quality monitoring devices are highly capable of measuring various air quality parameters such as surface ozone, NO_2, SO_2, and particulate matter (PM2.5/PM10), and relay this information in real-time to the authorities to help track the zones of high pollution. Low power wide

area network (LPWAN) technologies such as LoRA are enabling the development of dense air quality monitoring networks due to its ability for long-range connectivity and low-power consumption. An urban sensing project called the 'Array of Things (AoT)' is being carried out in Chicago to collect real-time data on the city's environment for a more healthier, more efficient, and more livable city. The list of sensors includes environmental sensors, air quality sensors, and light and infrared sensors.

Improving water conservation Water conservation and management is essential in many facets of human lives. The various current practices are time consuming and lack real-time tracking and alerts for timely intervention to reduce wastage of water. Out of many applications of IoT in this area, two applications are described as follows:

- Smart water meters that can be used to detect leakage of pipes in the water delivery infrastructure (e.g., water grid). It also can help to precisely understand the water consumption behaviour of the users through data analytics and make and send alerts in real time. Further, by analysing the requirement of water in an area, adjustments to the optimal supply of water can be achieved.
- Water quality aspects can be studied by the deployment of IoT sensors in the water supply network to measure various water quality parameters to quickly react to those conditions when the water quality is deemed below the safe level.

For more compelling use cases, the reader is referred to Chapter 17.

1.2.2 Multidisciplinary Nature of IoT

IoT is driven not only by people who have specific background in Information and communication technologies (ICTs), but also by non-IT people who are showing significant interest in understanding the IoT ecosystem and contributing to its development using their own domain specific expertise. Institutions are increasingly feeling the need to have personnel particularly those who are involved in systems development to understand a host of disciplines to help them design complex IoT systems. Hence, multi-domain research and development teams are currently preferred in the development of IoT system. As shown in Fig. 1.8, there are a multitude of disciplines that are involved in IoT, each of them have their own set of requirements and goals. However, the domain expert in any of these areas, for example, a person with Transportation systems development, has to work with a person who knows about sensors and their deployment, and in turn need to work with an Information Technology (IT) person to help develop the software and integrate with the system. Similarly, doctors and electronics and communication technology engineers have to work hand in hand to develop new IoT devices for healthcare. In addition, social scientists are required to understand about the societal aspects of the developed technology. This line of thought can be extended to many sectors such as defence, environmental monitoring, energy, and industries, who are currently building transformative IoT solutions Therefore, it can be seen that for any of the technology that is being developed in IoT, there are multidisciplinary teams (members with varied but complimentary experience) involved to make the idea take shape and finally come out with tangible product.

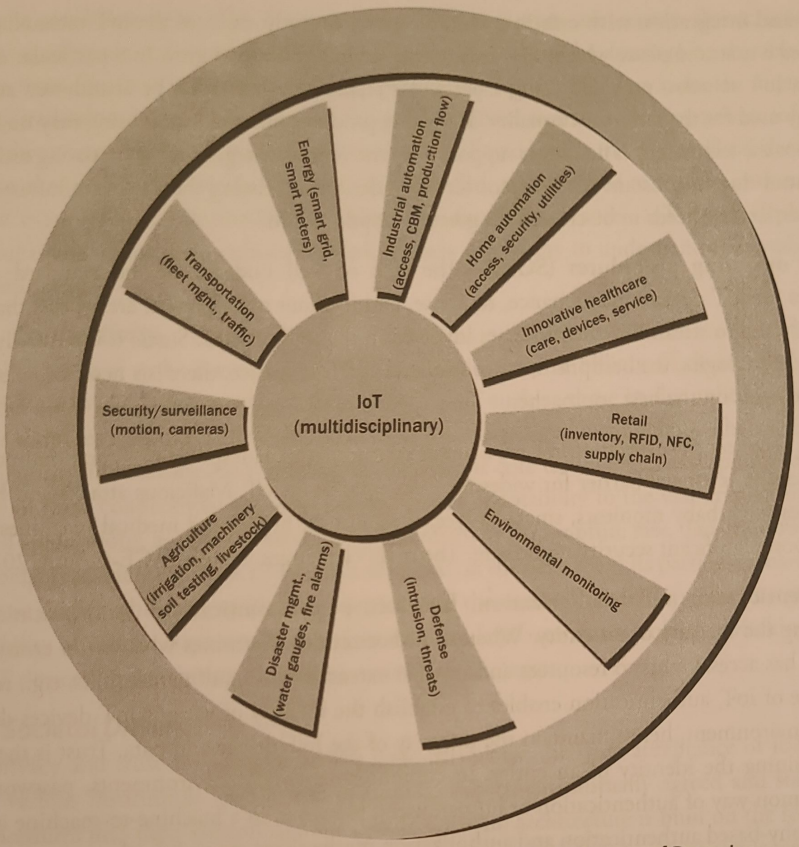


Fig. 1.8 Interdisciplinary Nature of IoT Encompasses a Variety of Domains

1.2.3 Challenges Involved in Its Further Evolution

The main challenges involved in the further evolution of IoT are as follows:

1.2.3.1 Technological Challenges

The technological challenges that could impede the future evolution of IoT include the following:

IoT system design considerations Currently, there are a variety of protocols (network, communication, data protocols, etc.) that are being used by various organizations/entities to design, develop and implement IoT systems in diverse domains. This is leading to non-interoperable systems, whose integration and access is becoming very challenging. Unless, standards are implemented in every step of the IoT systems development process, there will be many systems that could become unusable and obsolete. For example, users will end up with an IoT system (e.g., home automation), which will become highly vendor dependant and is no longer able to integrate with other systems implemented by a different vendor.

Maturity and integration with existing technology Currently, most of the IoT technologies have not reached a matured phase where they are guaranteed to continue to grow in a particular direction. Hence, an IoT solution provided using a particular technology may soon be abandoned and a new technology used for the same functionality. Therefore, products that the user is using may no longer be useful, since the technology is no longer supported by the vendor. Integration with existing technologies and overhead due to constant requirement for upgradation of existing the hardware infrastructure is another barrier that needs to be crossed for greater adoption of IoT.

Standard operating procedures (SOPs) The lack of well understood and standard operating procedures for IoT devices maintenance, response, and incident management are aspects that need to be strengthened to streamline the processes involved in the IoT systems. Since, it is a rapidly evolving cluster of technologies, the best practices are scattered and few. Hence, there is a need to document and guide developers on the best approaches.

Security The security aspects of IoT are a major concern to its widespread proliferation in the future. It is considered as crucial barrier for widespread acceptance of IoT. Malicious attacks and hacking of IoT devices (e.g., baby monitors, smart fridges, thermostats, health and medical machines, cameras, etc.) can pose significant security nightmare. The main challenges include (ACE-OAuth, 2018):

(i) Authentication and authorization for secure communication Authentication allows establishing the identity of an entity. Whereas authorization determines whether an entity (a device or a user) has access rights to resources and to what extent (i.e., level of permissions, e.g., read/write). In the case of IoT, authentication enables to establish the identity of various IoT devices deployed in a shared environment, hence maintain the integrity of the IoT device and data. Trust is the backbone for ascertaining the identity of an entity. In computer-networked environments, passwords are the most common way of authentication of human users. However, in a machine-to-machine interaction, cryptography-based authentication and authorization mechanisms are useful, where cryptographic keys are commonly used. A digital certificate issued and digitally signed by a certificate authority (CA) contains a public key and identity of the owner. The IoT devices can use these digital certificates for authentication and create the required Trust for all parties in the network. However, such CAs are non-existent at present for the IoT domains. Hence, efficient ways to deploy the keys and manage them is an ongoing effort. Further, current security protocols and cryptography approaches require good amount of memory space, which is difficult to implement on low-powered IoT devices. Therefore, the challenge is to develop approaches for deploying and managing the keys that can adapt without causing additional overheard on the IoT node (Yang, et al., 2016). In addition, the authorization and access control should be customized based on the type of the IoT Node (Li, et al., 2017). Recently, IETF proposed the authentication and authorization in constrained environments (ACE-OAuth) framework for IoT devices. It is based on OAuth 2.0 (OAuth is a widely used open standard for delegation) and Constrained Open access Protocol (CoAP).

(ii) Privacy IoT devices that collect sensitive user's data are posing an immense threat to an individual's privacy. For example, data gathered by sensors related to health, home appliances usage, tracking devices for phones, work habits, etc. could be transmitted to a cloud service or a third party

without the user being aware of it. Existing privacy approaches are user-centric, that is, it is based on individual's preference to the content and quantum of data that he/she deems to be sharable and what remains private. The data-collection entities are obligated to inform the users about the intended usage of the acquired information. However, in the case of IoT nodes that are collecting private information, there is a threat to privacy at (i) endpoints where each IoT node emits the data, and (ii) data obtained from networked IoT nodes, which are collected, combined, and analysed to reveal a pattern and thus giving out more sensitive information. In addition to data privacy issues, IoT device deployment at sensitive areas and across socio-cultural borders requires a new way to understand the implications of privacy invasion.

1.2.3.2 Connectivity

Huge challenges need to be overcome for ensuring connectivity of billions of devices in IoT. Many technologies are available for enabling connectivity in IoT such as those in the unlicensed spectrum, low-power wide area networks (LPWAN), cellular, and satellite-based technologies. These are at various stages of maturity and continuously evolving in terms of the core technologies and applications to IoT. The future of these connectivity solutions depend on the adaptability to the vast array of diverse IoT devices and applications, as the requirements vary in terms of data capture rate, data transmission rate, latency, storage, etc. Hence, connectivity needs are based on the device and its particular characteristics. Therefore, no single technology will be able to cater to all the needs of the IoT systems. Integration and interoperability of various connectivity solutions will be a key factor for seamlessly switching between various IoT devices implementing varied connectivity technologies.

1.2.3.3 Societal Drivers

Issues of privacy and trust are going to be the main drivers for the wide acceptance of IoT. There is a need for various institutions (public and private) to implement mutually agreed and standardized privacy procedures into their systems and the core IoT system architecture is built on the foundations of privacy, trust, and data protection. For example, an IoT device could capture the location of the user to give location-based services, but does not use that data for any other purpose, such as tracking the individual. Similarly, the data captured by the device may send only the summary and does not transmit the original raw data, or the data lives only for a short period or anonymised so that it cannot be identified with any particular user. Such kind of approaches will make the user to feel protected when using privacy friendly IoT applications. Currently, such things are not fully implemented in the IoT ecosystem. The current focus is more on giving better functionality and user experience rather than focusing on implementing the principle of *Privacy by Design*. According to it, starting from the fundamental building blocks in the IoT system development, privacy is given utmost importance and embedded in each and every process of the system development. Such systems are robust and have better chances of gaining the trust of the consumer.

1.2.3.4 Uncertain Returns on Investment

The investment in IoT according to the market reports is very high (see Box 1.3). However, businesses are struggling to understand and make an estimate of the return on investment (ROI) due to the emerging nature of the IoT technology and lack of historical data or business cases (World Economic Forum report, 2015) that could be used as a benchmark or point of reference to estimate the ROI.

1.3 STANDARDIZATION

The massive scale of IoT comprises billions of devices and subsystems. To be able to interconnect them and derive meaningful information requires that these heterogeneous systems need to be interoperable, that is, the ability to work with any underlying protocols, data models, and content types. Since, there are many public and private stakeholders involved in IoT, there will be many proprietary and open solutions in various domains with each solution providing its own IoT infrastructure, devices, APIs, and data formats due to which it becomes challenging to integrate them. To achieve interoperability and enable scaling up of IoT, standards are required.

1.3.1 Need for Standardization at Various Layers of IoT

As IoT is a multi-layered system comprising of various layers related to sensing, networking, communication, session, etc., standardization is required at each of these layers (the architecture reference model of IoT is explained in Chapter 14). Considering this need, several standards have been proposed with a focus on specific requirement in the development of an IoT system. For example, the data link layer connects two things or thing and gateway device that connects a group of things to the Internet. Various short-, medium-, and long-range protocols have been developed for connectivity (see Figs 1.6 and 1.9). Specialized routing protocols were developed to enable several IoT devices to communicate and aggregate information and then send it on to the Internet. Further, in the network layer, among other protocols such as UDP, 6TiSCH, and THREAD, the 6LoWPAN is developed for power constrained devices, standardizes a way to carry packet data in the form of IPv6 over IEEE 802.15.4. The messaging among various components of the communication subsystem is addressed in the session layer. Among them, the Constrained open Access protocol (CoAP) was developed by IETF Constrained RESTful Environments Working Group (CoRE). It works on UDP or UDP analogue. The Message Queuing Telemetry Transport (MQTT) originally developed by IBM works on the TCP/IP. More details on the IoT standards and protocols are discussed in Chapter 4.

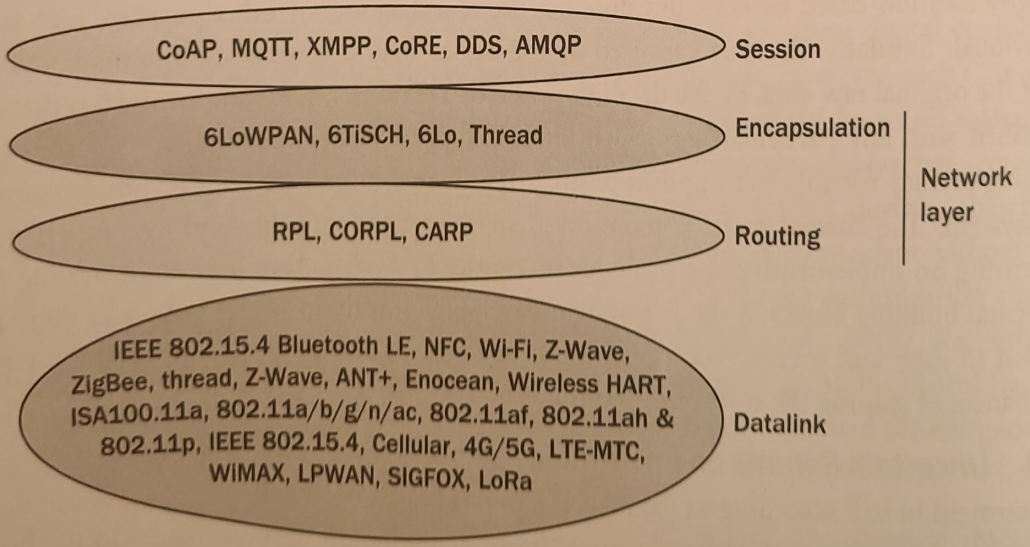


Fig. 1.9 Variety of Standard Protocols Currently Available for IoT

1.3.2 Organizations and Their Efforts for Standardization

Various organizations are involved in the IoT standardization efforts. Following are some key institutions and their contributions:

Institute of Electrical and Electronics Engineers (IEEE) IEEE is a global, professional engineering organization whose mission is to foster technological innovation and excellence for the benefit of humanity. The standards body of IEEE called the IEEE standards association is involved in developing standards, specifications, and best practices for IoT. The IEEE P2413 defines an architectural framework for the IoT, including descriptions of various IoT domains, definitions of IoT domain abstractions, and identification of commonalities between different IoT domains.

ETSI (European Telecommunications Standards Institute) ETSI develops standards for information and communications technologies (ICT), including fixed, mobile, radio, converged, broadcast, and Internet technologies. In the IoT space, it works with ONE M2M (another standards body) to provide standardized M2M interfaces. The goal is to enable IoT devices to connect seamlessly and to ensure that they are network agnostic. From an M2M standpoint, various M2M technologies are supported by ETSI such as smart appliances, smart metering, smart cities, smart grids, e-health, and intelligent transportation systems. Further, specific to IoT, ETSI is working in the areas of: security for the IoT, low power supplies in the IoT, radio spectrum requirements, and embedded communications modules.

OneM2M It is a Global standards initiative for M2M and IoT. It has over 200 member organizations. It works in a partnership mode with various standards organizations and provides a detailed standard for M2M/IoT in the areas of architecture, interfaces, security, communication protocols, etc. The oneM2M-layered model consists of network layer, common services layer, and the applications layer. The goal is to have applications to share common infrastructure, environments, and network elements to enable interoperability. The common services layer can be readily embedded within various hardware and software and provides functions that M2M applications across different domains commonly need (e.g., data transport, management, discovery and policy enforcement, security/encryption, etc.). The Representational State Transfer (REST)-based architecture of oneM2M allows the use of multiple protocols such as HTTP, CoAP, MQTT, or WebSocket to work with oneM2M application servers worldwide.

Internet Engineering Task Force (IETF) It is an open, international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet. Its mission is to "make the Internet work better by producing high quality, relevant technical documents that influence the way people design, use, and manage the Internet." Within IETF, specific IoT-based focus is towards:

6LoWPAN 6LoWPAN is IPv6 adaptation layer and header compression mechanism that is suitable for low power constrained IoT devices and networks.

Routing Over Low power and Lossy networks (ROLL) ROLL establishes routing protocols for constrained-node networks.

Constrained RESTful Environments (CoRE) CORE extends the Web architecture to most constrained networks and embedded devices.

International Telecommunications Union (ITU) It released the ITU report on The Internet of Things in 2005, which describes the various visions of IoT, and terms IoT as an 'ubiquitous network'. It published recommendations in the areas of tag-based identification services, ubiquitous sensor networks (USN), and ubiquitous applications in next-generation networks. Specifically for IoT, this organization contributed towards IoT terminology, common requirements and capabilities, APIs and protocols for IoT, IoT testing, network, security, and privacy protection aspects of IoT. Further, various focus groups were formed for several application areas such as e-health, smart grids, home networks, smart sustainable cities, and smart water management.

Object Management Group (OMG) It developed the Data Distribution Service (DDS) for IoT. It is a middleware protocol and API standard for data-centric connectivity. It is pub/sub standard, which enables scalable, real-time, reliable, high performance, and interoperable data exchanges between publishers and subscribers. In a data-centric system, the focus is on user-defined data (the data model). The middleware understands the context of the data and ensures that all interested subscribers have a correct and consistent view of the data.

OASIS It is a standards body that is responsible for providing a lightweight publish/subscribe reliable messaging transport protocol suitable for communication in M2M/IoT contexts where a small code footprint is required and/or network bandwidth is at a premium.

1.3.3 Factors for Widespread Adoption of IoT

IoT-based technologies are driving a wide range of application areas that have stagnated and lacking much innovation in the preceding years of IoT. It has already permeated in various walks of life. The various factors that led to the widespread adoption are mainly due to the technological advancements and economic viability in the areas of the following.

1.3.3.1 Technology Viewpoint

Communication and network Connectivity is one of the key pillars of IoT. There are a host of technologies that are being developed (some of them have already reached mature levels) and in multiple ranges (short, medium range) which are enabling the IoT revolution gain widespread acceptance. Customized versions of Internet protocols for low-powered devices such as the 6LoWPAN, which is based on the IPV6 Competing wireless technologies such Cellular (4G/5G) and LPWAN (e.g., LoRA) technologies are giving the IoT system developers options to develop IoT networks that can be deployed in remote challenging areas. More details are presented in Chapter 4.

Computing A computing revolution has happened in the last decade, with new paradigms of computing emerging such as the cloud computing. It has completely changed the way in which data is stored, processed, analysed, and disseminated. The low-cost nature and always available kind of computing resources is an attractive alternative to the user as compared to in-house computing infrastructure.

The IoT has considerably gained by cloud-based infrastructure, since there is a need to process data from a very high number of sensors, which requires considerable computing power. Organizations and individuals working in deploying IoT-based solutions can now subscribe to a cloud-provider for a considerably low rate and quickly deploy the IoT solutions. Various cloud providers (e.g., Amazon Web Services (AWS) IoT, IBM Watson IoT, etc.) are providing off the shelf IoT platforms to enable end-to-end management of the IoT resources. Further, the emergence of 'Data Science' area has ushered in new ways for analytics, both at the edge of the IoT network through streaming data processing analytics and also batch-processing/offline approaches that includes machine learning-based techniques.

Low-cost devices Sensors and their integration with IoT is becoming affordable. Due to it, an ever-increasing number of domains are able to use IoT sensors to drive their processes; these solutions are contributing towards developing automation in several sectors.

1.3.3.2 Consumer Viewpoint

From a consumer perspective, usefulness, price, connectivity, security, and privacy are the main enablers for high adoption of IoT products. The IoT products such as wearables in the fitness and wellness area have already showed high promise due to the practical use and affordable pricing. As the consumers get more educated in the value that IoT products bring to their daily lives, greater will be the adoption. Companies have already embarked on targeted campaigns to create consumer awareness and change their perception of this new technology.

REVIEW QUESTIONS

1. Describe the historical context that led to the emergence of IoT.
2. Describe various Auto-ID technologies.
3. What is the vision of IoT? Explain the different perspectives from which IoT's vision can be understood.
4. Various organizations have defined IoT. Describe in what aspects these are similar (if any). What components or parts of these definitions are similar?
5. Give your own definition of IoT. How is it different from other definitions?
6. What is an enabling technology? Describe the key enabling technologies of IoT.
7. Explain the evolution of a disruptive technology.
8. Why is IoT considered as a disruptive technology?
9. Give some motivating scenarios where IoT has been a disruptive technology.
10. What is the need for IoT standardization?
11. Give an overview of IoT Standards and mention the organizations that are involved in IoT Standardization and their specific roles.
12. What is leading to the widespread adoption of IoT?

REFERENCES

1. ACE-OAUTH [ONLINE]. Available at: https://tools.ietf.org/pdf/draft-ietf-ace-oauth-authz-13.pdf [Accessed 2/10/2018]
2. Array of Things [ONLINE]. Available at: https://arrayofthings.github.io/ [Accessed 2 Oct 2018]

Concept of Smart Things/Objects

CHAPTER 2

"There's no such thing as simple. Simple is hard."

-Martin Scorsese

OBJECTIVES

- introduce the fundamental concept of a Thing
- explore the nature of a Thing
- understand the needs of a Thing
- identify various domains where commonly used Things are becoming smart
- understand what is making Things smart
- introduce Machine-to-Machine (M2M) technology
- key differences between M2M and IoT

OUTCOMES

- describe a Thing from an IoT standpoint
- think about a Thing from its needs perspective
- gain ability to visualize how commonly used things turn into Internet of Things
- explain the core components of an M2M architecture
- tell how IoT is quickly gaining ground and set to influence the M2M technology

REVISION

Chapter 1 introduced the emergence of IoT. Several definitions of IoT were given and converged on a working definition. The basic building blocks of IoT are explained. Further, various IoT domains and motivating examples where IoT is making an impact are described in that chapter. This will form the necessary background for this chapter, which is focused on understanding the concept of a Thing in much more depth.

2.1 THING IN THE CONTEXT OF IOT

A Thing in the context of IoT is an entity or physical object that is a unique identifier, an embedded system, having the ability to transfer data over a network. It has some distinct characteristics as shown in Fig. 2.1.

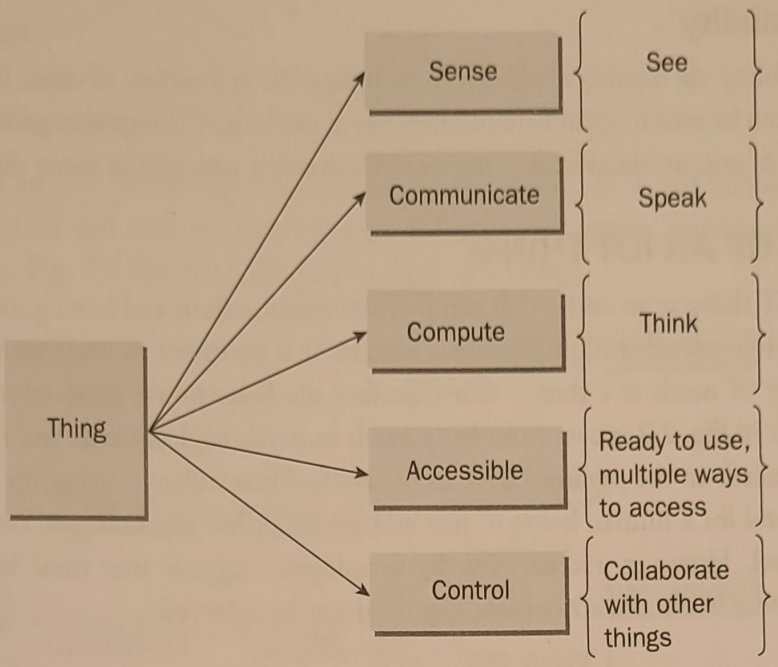


Fig. 2.1 Attributes of a Thing

2.1.1 Capability to Sense the Environment

Things can see, hear, and feel the environment around it through the embedding of sensors, which can make a dumb thing smart.

2.1.2 Ability to Communicate

Things can communicate with humans as well as other Things and collaborate with them so as to solve a bigger problem. It can be a main actor or just one node out of many that will come together dynamically to accomplish a task.

2.1.3 Computation Capabilities

A Thing is capable of thinking and analysing what it sensed; although the extent of this capability depends on the characteristics of the computing hardware that is available on the Thing. Normally, the computations at the Thing level are not too intensive or complex; it basically captures the change of a particular variable and uses in-built and pre-configured processes to output the observation value.

2.1.4 Control Other Things

An IoT Thing can remotely access other Thing(s) to affect a change in its state. It can trigger certain functionality in a remote Thing by commanding it to make some data acquisition by changing the state of an actuator to begin the process.

2.1.5 Accessibility

The Thing's availability via internet enables it to be integrated in a variety of ways. It can have a uniqu address/code that can be used to identify it uniquely out of millions of Things that are available via Interne This characteristic of unique identification makes it discoverable and used in many different ways.

2.2 NEEDS OF AN IOT THING

Considering an IoT thing as an entity that can perform many actions and having some ability to think analyse, and react to external stimuli just like a human, it is pertinent to understand its needs as well Maslow's hierarchy of needs is a theory that describes the behavioural motivation of a human. The hierarchy as shown in Fig. 2.2 moves from basic needs to psychological needs and then self-fulfilment The bottom level needs in the pyramid have to be satisfied first before moving up to the next level of needs. This is natural for a human being to first address his or her psychological needs and then move on to the next level. However, recent work by sociologists suggests that these levels are somewhat overlapping and lower levels and strict hierarchy need not be enforced.

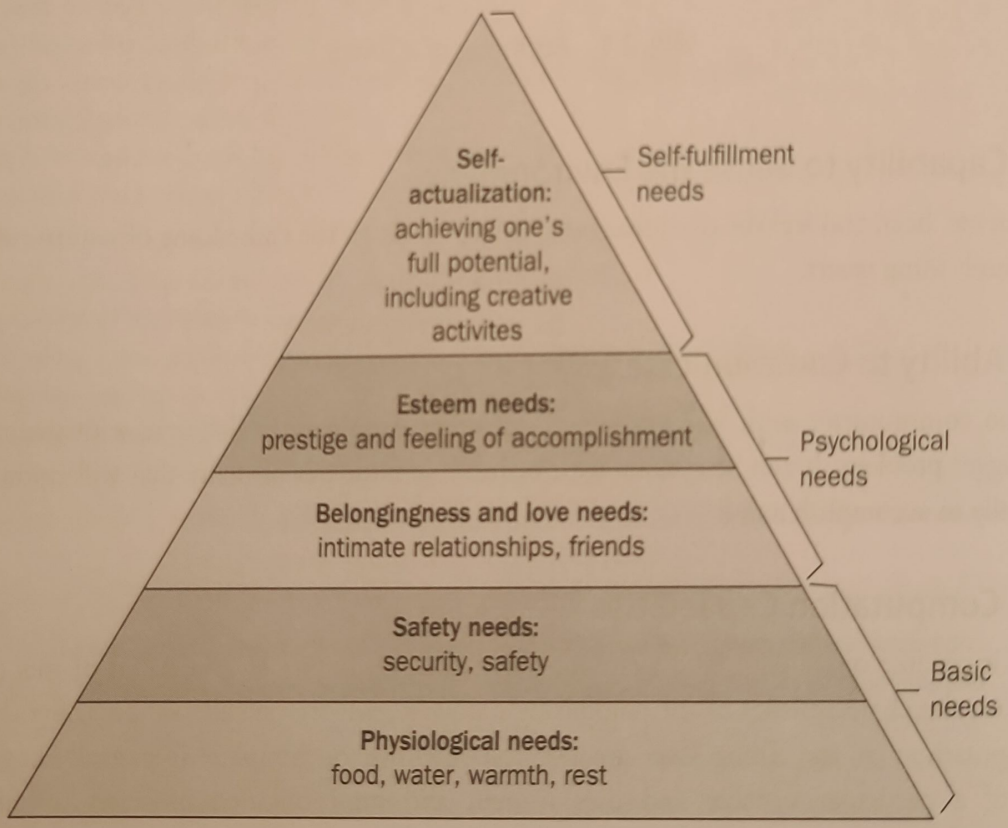


Fig. 2.2 Maslow's 'Theory of Human Motivation' Describes a Hierarchy of Needs

The Maslow's theory has been found useful to be applied in many domains in addition to IoT. A few are listed below:

- Data science
- Choosing a right data center
- Employee needs

- Market analysis
- Retail
- Technology integration

This perspective of an IoT Thing helps to have a more in-depth look at the various ways in which a Thing can express itself and how the technology is aiding in this aspect. Treating a Thing as a living being and a human, Fig. 2.3 shows a hierarchy of needs of a Thing.

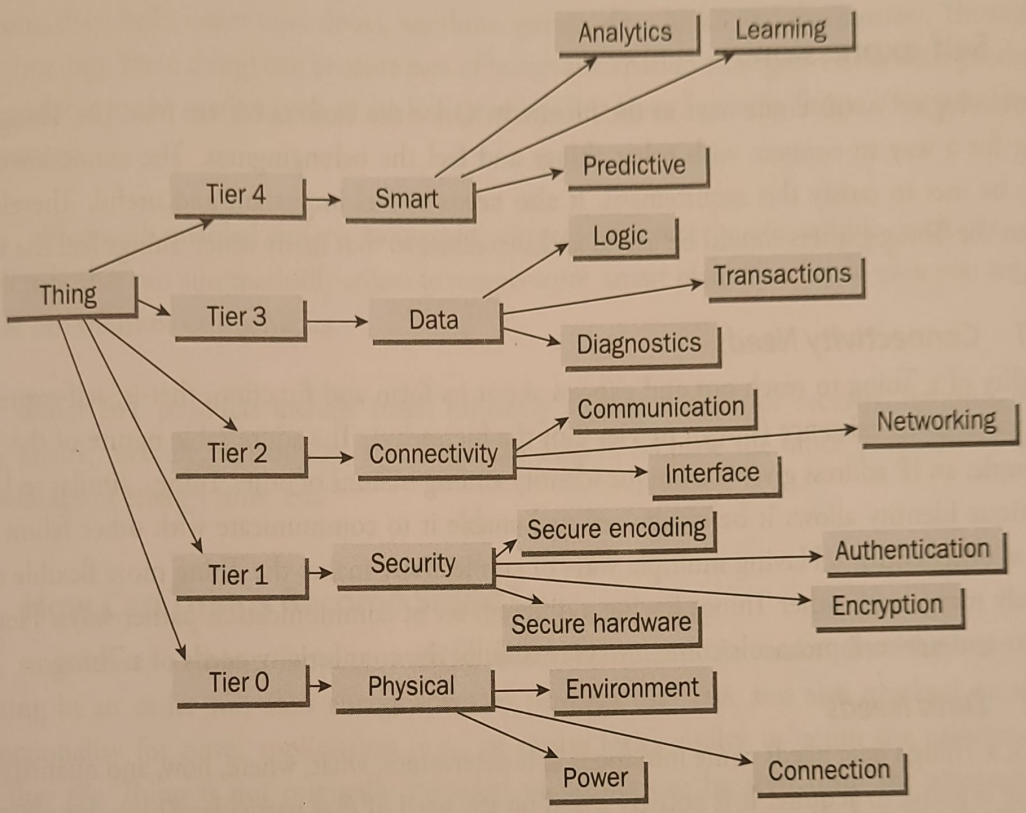


Fig. 2.3 Hierarchy of Needs of a Thing in Various Tiers Based on Maslow's Hierarchy of Needs Theory in Psychology (Needs lower down in the hierarchy must be satisfied before Things can fulfil needs higher up).

2.2.1 Self-existence

These needs are very basic in the sense that they are more physiological in nature and determine the very existence or survival of the Thing. Hence, these are at the bottom, that is, Tier 0 of the hierarchy.

2.2.1.1 Physical Needs

The physical needs are its core of existence, which points to the basic necessity of having power to be "on" and having the ability to sense the environment around it. A connection so that the values of the sensed environment variable are broadcasted/transmitted, so some radio equipment is needed. Furthermore, the Thing needs physical protection from damage/accident for being in the open external environment. Hence, various Things need different kinds of physical enclosures to protect against the elements of both natural and human interaction. This is the Tier 0 at the bottom of the hierarchy. According to Maslow, if the needs at this level are not fulfilled, the Thing will not feel the need for higher level tiers to exist.

2.2.1.2 Security Needs

The Thing needs to feel secure. In addition to having a protective casing or shield (at Tier 1 level), it also needs to be able to distinguish between a genuine user asking for its attributes or sensed readings, and a rogue (hacker or a bot) trying to extract information from it through fraudulent means. Hence, the Thing needs some form of security built into it so that it can allow or disallow access to it. Therefore, protection in the form of encryption, authentication, and authorization are necessary to enable the Thing to feel protected and exist without succumbing to threats.

2.2.2 Self-expression

The psychological needs come next in the hierarchy. Once the basic needs are met, the Thing is now looking for a way to connect with other things and feel the belongingness. The connectivity needs have to be met to satisfy this requirement. It also needs to feel important and useful. Therefore, the data that the Thing gathers should be useful and shareable, so that many other Things feel the need for its data.

2.2.2.1 Connectivity Needs

The ability of a Thing to reach out and express about its form and function, that is, self-expression is fundamental to its existence (shown in Tier 2 of the hierarchy). The addressable nature of the 'Thing', for example, an IP address gives it a unique identity among billions of other Things, similar to humans. This unique identity allows it be recognized and enable it to communicate with other fellow Things as well as other entities. Having multiple ways of connectivity makes the Thing more flexible and can seamlessly merge with other Things having a different set of communication pather ways. Hence, the transport and network protocols come into play to fulfil the connectivity needs of a Thing.

2.2.2.2 Data Needs

In Tier 3, a Thing's data needs come into focus as it determines, what, where, how, and quantity of data the Thing is going to acquire. It is entirely based on the need of that particular Thing, for which it has come into existence. Therefore, the kind of environment that it will collect the data, at what locations this data would be acquired, the modalities of acquisitions, and the frequency of the data collection are all programmed and placed into the brain of the Thing. Further, the Thing needs a way to pre-process and store the data in a way to make it available for higher-level processing.

2.2.3 Self-actualization

This level reflects the ability of the Thing to reach its full potential in all the dimensions for which it was born to accomplish. It is able to display intelligent and learned behaviour.

2.2.3.1 Smart Needs

This forms the top of the Tier, that is, Tier 4, where the Thing finally is able to realize itself, and get the feeling of self-fulfilment, that is, it is able to perform the function(s) it is made for. It is also now a part of a bigger goal or a system to provide a collective solution. The Thing can have several attributes at this stage such as predictivity, self-configuration, and learned behaviour, and has reached its full potential.

2.3 COMMONLY USED SMART THINGS

There are a variety of applications that benefit from IoT including smart cities, smart metering, smart retails, smart logistics, smart industrial control, smart environment, etc. However, in this section, commonly used Things are described to give a flavour of how ordinary things are made smart by IoT technology:

Home Many things around the home can become smart such as power outlets, lighting and switches, door knobs, door bells, water taps, doors, windows, garage, door locks, and thermostats. Through IoT-based technology, these things can become part of home-automation strategies. Many such products are already available in the market such as the IoT products line from Samsung (https://www.smartthings.com/products).

Kitchen Bluetooth-enabled cutlery, connected refrigerator, remotely controllable crock-pot, cooker, and cook tops that can automatically adjust to temperature, smart plates that know what you are eating, forks that can monitor eating habits.

Office Office IoT products include smart products such as badges for security, lighting, window shading, chairs, meeting, scheduling, etc. These can help to increase the office productivity and also reduce wastage of energy, time, etc.

2.3.1 How Can Things Become Smart?

Adding smartness to things implies making them react and do tasks that they are already performing in an easier way (less energy spent to perform that task and also quicker) or adding a new functionality for novel applications (e.g., an eating fork's ability to count the calories of food intake) that the Thing is not originally intended to do. As can be seen from the aforementioned examples (Section 2.3), there are several things that are performing multiple tasks and allowing the user to gain more insights and feedback in his or her daily routine. To enable this smartness, certain technological integration is necessary. Following are some major approaches to make things smart.

2.3.1.1 Sensors and Actuators

Sensors enable things to sense their surroundings and obtain certain information, which can be converted to perform tasks. On the other hand, an actuator is not only a sensor, but is also able to perform a certain action based on the change of state of a certain parameter. In IoT, a Thing can command another Thing to perform a certain action; an actuator comes into picture at this juncture and plays the role of an intermediary to make the Thing perform that action. An actuator is able to move and control a mechanism (e.g., open a valve) after it receives a signal either from the same Thing or an external Thing of a Thing such as a tap. Chapter 5 on sensors and actuators gives more details on these concepts.

2.3.1.2 Communication over Network Interfaces

The ability of a Thing to interconnect with other Things is a central part of IoT and requires communication protocols that are specifically tailored for such type of interaction. Some of the earlier technologies such as HTTP, TCP, and IP stack are being transformed to specifically suit (e.g., IPV6 is preferred over IPV4 for IoT devices) IoT devices communication. Because the IoT devices are deployed in diverse ways (e.g. wearables, connected homes, factories, transportation, etc.), there are several communication protocols such as Wi-Fi and low-power technologies such as LPWAN and IEEE 802.15.4 that are chosen based on specific requirements, for example, if the communication is required only between Things that are in local network or there is a need to communicate remotely. More details on the IoT protocols are described in Chapter 4 on IoT standards and protocols.

2.3.1.3 Connecting to the Internet

Things usually connect to other things via a local network in situations where only local communication is necessary; for example, connected home. However, the main feature of IoT is the ability to remotely control Things and transmit data measured by Things over the Internet, which is possible by connecting to an IP network. One of the recommended ways is to use 6LoWPAN (IPv6 over Low-power Wireless Personal Area Networks), thus enabling global communication. Another, indirect way to make Things visible on the Internet is to use a gateway, which acts as an intermediary between the Things on a local network and the Internet. Multiple Things can connect to a gateway, which is further connected to the Internet.

2.4 MACHINE-TO-MACHINE TECHNOLOGY

Machine-to-Machine (M2M) technology mainly resides in the non-consumer world. It is mostly a business solution for automation and instrumentation, where machines of same type or function are usually connected either by wired and wireless means. It helps in the remote monitoring and management of equipment. However, the remote access is achieved by point-to-point communication through hardware integrated with the machine. These are commonly proprietary solutions tied to a particular solution provider's cellular or wired networks to access and exchange data. Some examples where M2M technology is prevalent are: supply chain, traffic control systems, logistic services, telemedicine, etc.

2.4.1 European Telecommunication Standards Institute (ETSI) – M2M

European Telecommunications Standards Institute (ETSI) developed a set of M2M specifications based on Representational State Transfer (RESTful) (a distributed Service Oriented Architecture [SOA] that uses web protocols) architecture to (Swetina et al., 2014) the way heterogeneous devices can offer services and access to them seamlessly. The core components of an M2M are shown in Fig. 2.4. These components form the major building block of any M2M system. Following is a brief description of each of these components.

2.4.1.1 M2M Device

The M2M devices consist of sensors and communication equipment that are at the lowest end of the M2M system; they are usually connected to an operator's own network or can connect based on existing standards such as Zigbee, bluetooth, Device Language Message Specification (DLMS),

European Committee for Standardization (CEN), Meter-Bus (M-Bus), etc. In this case, the gateways play a major role in ensuring that the data is further moved for processing. For example, sensors placed at selected locations around a water body that can detect contaminants for water-quality monitoring, the data can be sent to a gateway for real-time analysis, aggregation, and further processing. In this case, the gateways are responsible for the proper addressing and routing of the devices. It is outside the scope of the network operator for management purposes. On the other hand, the devices that are connected to a fixed network (embedded SIM, radio stack or fixed line access, etc.), form the end points of the network and hence are managed by the network operator itself.

2.4.1.2 M2M Area Network Layer

This is the connection between the M2M device and the gateways, which is possible via various area networks, such as WBAN and WPAN using technologies such as IEEE 802.15.6 Zigbee/IEEE 802.15.4, Bluetooth, Bluetooth Low Energy (BLE), wireless USB, and proprietary solutions such as ANT, Sensium, Z-wave, and Zarlin.

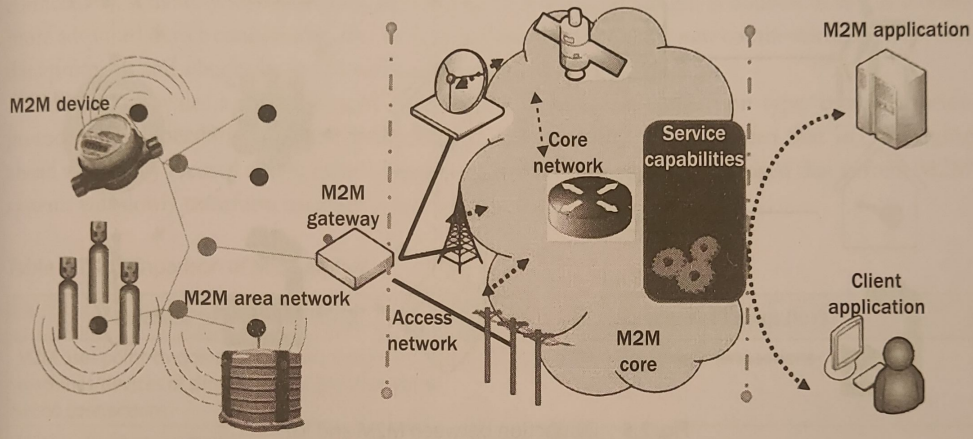


Fig. 2.4 Core Components of ETSI M2M Architecture

2.4.1.3 M2M Gateway

Gateways along with routers are equipment that ensure the connection of the M2M devices to communicate with M2M applications via a communication network. As shown in Fig. 2.5, the M2M gateway acts as a conduit between the communication and data transmission functions of an M2M system. If the M2M devices are not connecting directly to the network, then the gateways act as end point to the operators network. This means that the accessibility of an M2M device from an M2M application or an operator-specific interface is mainly dependent on the gateway, which needs to ensure the proper access with suitable authentication and other security features.

2.4.1.4 M2M Communication Network

During the formative years of M2M technology, the communication aspects of the M2M applications (e.g., monitoring and control of machines, alarms, etc.) relied on specialized communication networks.

However, this has changed over a period of time with M2M technology being widely applied to a wide variety of communication networks. It is mainly the communications between the M2M gateway and M2M application(s). Technologies such as GSM, UMTS, xDSL, LTE, LTE-A, WiMAX, WLAN, satellite, etc. are being used for this purpose.

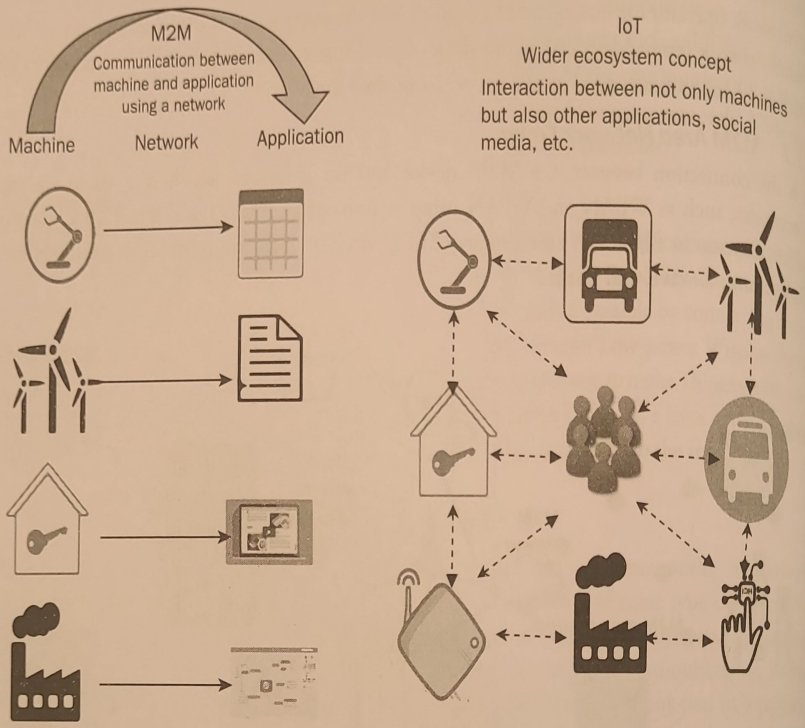


Fig. 2.5 Distinction between M2M and IoT

2.4.2 M2M Service Layer

This layer is one of the core M2M layers which provides M2M standardized device and data related functionalities such as data transport and device management. Irrespective of the communication technologies used, the M2M service layer enables to seamlessly connect various M2M devices and applications across various application domains, thus ensuring interoperability. oneM2M is a global initiative to standardize a common M2M service layer platform.

2.4.3 M2M Applications

The application layer is at the top of the M2M system. Usually, a middleware layer sits between the communication layer and the applications. It is a software that enables making the connection between diverse components and applications, which otherwise are unable to talk with each other by providing

services to ensure effective communication. More details on IoT middleware is provided in Chapter 7: IoT middleware. Some typical M2M applications are as follows:

- Health monitoring
- Remote access, control, and monitoring of equipment such as pumps and remote diagnostics of vehicles
- Smart grid applications, smart meters, and industrial meters
- Tracking and monitoring of assets
- Monitoring production chain in a manufacturing unit

2.4.4 Key Differences Between M2M and IoT

M2M and IoT are technologies that offer similar functionality in terms of communication, monitoring, data acquisition, and exchange, to enable automation. Hence, it is quite possible to assume that they are synonymous with each other. However, there are some key differences between them in terms of connectivity, scalability, and data usage and sharing. M2M is now thought as a subset of IoT as it offers more advanced device connectivity, and services and incorporates many new communication, network, data protocols and standards, and in various application domains (Fig. 2.5).

Table 2.1 provides a comparison of M2M and IoT, based on connection type, communication protocols, interoperability, targeted business type, and scalability. It can be seen that IoT is forging ahead with many focused developments happening in many areas and it is envisaged that current M2M systems will slowly transform into IoT systems in the near future.

Table 2.1 Comparison of M2M and IoT

Machine-to-Machine (M2M)	Internet of Things (IoT)
M2M uses point-to-point connections between machines/devices of similar types using wireless or wired connections	IoT is based on IP network connections
M2M is all about machines	IoT is about Things
Older protocols and communication techniques	Varying new protocols specifically developed for IoT protocols such as 6LoWPAN for lower-powered IoT devices
Mainly hardware-based, i.e., the M2M technology is integrated into the hardware	IoT is both hardware and software based. Emphasis is on integration of a variety of Things
M2M has less built-in intelligence	More intelligent form of machine communications, cloud, context, collaboration, and insights
Ability for Integration and interoperability are limited due to lack of well-recognized standards and more proprietary systems are in vogue. Data is usually not shared	High levels of integration is possible. Plethora of standards has emerged for low-powered IoT devices. Data sharing from multiple assets is highly recommended for developing cross-domain applications
Business-to-Business (B2B)	Business-to-Business (B2B), Business-to-Consumer (B2C)
In majority of M2M systems, Internet connection is not required	Internet connection is required for majority of IoT systems
Limited scalability	Potential for high scalability due to Internet enablement and cloud services

Wireless Sensor Networks in IoT

CHAPTER 3

"When wireless is perfectly applied, the whole earth will be converted into a huge brain, which in fact it is, where all things being particles of a real and rhythmic whole."

— Nikola Tesla

OBJECTIVES

- learn about wireless sensor network (WSN)
- understand challenges of WSN technology
- list various characteristics of WSN
- gain knowledge of different types of WSN and their architecture
- Study WSN communication patterns and data integration approaches
- gain insight into various applications of WSN and best practices

OUTCOMES

- understand the challenges while designing WSN
- describe the desired characteristics of WSN
- gain the ability to select a particular type of WSN for an application
- select communication protocol based on domain requirements.
- comprehensive understanding of WSN application areas

REVISION

In Chapters 1 and 2, an introduction and an overview and comprehension of important technologies, required for Internet of Things are covered.

Wireless sensor network as one of the most important technologies in IoT and a core base of IoT is explored in this chapter.

3.1 INTRODUCTION

Recent advances in efficient and affordable integrated electronic devices have a great impact on the state of Wireless Sensor Networks (WSN). WSNs are enabling applications in a variety of domains and paving the way to new market opportunities. However, there are significant challenges in their design, deployment, and maintenance, which cannot be ignored. The development of WSNs requires knowledge of software, hardware, as well as embedded engineering.

3.1.1 Wireless Sensor Network (WSN)

A WSN consists of a group of sensors that are deployed in an area with the purpose of collecting information about the surrounding environment. The WSN can be a small network consisting of a few nodes or hundreds of nodes forming a large scale network that can monitor vast areas (and in terrains

SUMMARY

The IoT revolution has ushered in new ways of interacting with everyday things by adding smartness using various sensors/actuators, communication, network, and cloud technologies. The life of a Thing is now analogous to a human and has its needs to be fulfilled to realize its full potential. This perspective of Things is put in terms of Maslow's theory on human motivation to understand the various needs such as psychological, safety, belongingness, esteem, and s actualization. Further, M2M is introduced in term its core components and areas in which M2M syste are prevalent. The key differences between IoT a M2M are discussed to show the various strengths each of these systems. It can be concluded that IoT potential for developing systems that have far-reac ing impact and applicable in a variety of domains.

REVIEW QUESTIONS

1. Explain the distinct characteristics of a Thing.
2. Describe Maslow's theory of human motivation.
3. Describe the connection between Maslow's theory and a Thing.
4. What additional needs do you think a Thing requires over and above those that were described using Maslow's Theory?
5. It now believed that there is a somewhat overlap of layers in the Maslow's Pyramid of human motivation. Where you think these overlap co happen from a Thing perspective?
6. What makes commonly used things to becom smart? Explain with real-world examples.
7. What is M2M?
8. Explain the core components of M2M.
9. How is M2M different from IoT?

REFERENCES

1. Maslow, A. H. (1943), A theory of human motivation. *Psychological Review* 50(4): 370–96.
2. Needs of a Thing. Available at: https://techcrunch.com/2015/09/05/the-hierarchy-of-iot-thing-needs/[Accessed: 10/22/2019]
3. Swetina, J., Lu G., Jacobs P., Ennesser F. and Song J., (2014, June), Toward a standardized common M2M service layer platform: Introduction to oneM2M, *IEEE Wireless Communications*, 21(3):20-26, DOI: 10.1109/MWC.2014.6845045.

that are hostile in nature) without much human intervention. A node in a WSN is a self contain unit consisting of sensor(s), trans-receiver, power source, and microcontroller. All these are encased a compact form, which can be easily placed in the field for acquiring the data.

In a WSN, the data obtained by the node is transmitted to a base station by routing it through other connected sensors in a multi-hop fashion. It also depends on the topology of the WSN, wh is elaborated in section 3.4. A bidirectional WSN is useful to remotely interact and control the nod and is particularly helpful for custom data acquisition (e.g., increase the frequency of data collecti from 1 hour to 15 minutes).

The cost and size of a sensor node depends on the application for which it is designed, and also facto such as on-board memory requirement to store the data, computing efficiency, power requirem based on the frequency of data collection and transmission, etc.

Various sensors (e.g., seismic, magnetic, thermal, infrared, acoustic, visual, and radar) used in WSN can have resource constraints such as computation capabilities, energy consumption, memo communication speed, etc. The major WSN functionalities include sensing, computing and data transf

The ability to self-organise is an important characteristic of WSN, which enables the node to configure the self's based on the requirements of a particular application such as surveillanc disaster monitoring, health monitoring, etc., where human intervention may not be possib either due to accessibility or safety reasons. Figure 3.1 shows the important modules of WS namely data acquisition, data processing, and data distribution.

The wireless nature of WSN results in some unique challenges such as power limitation communication network constraints, loss of connectivity of nodes, limited memory, no GUI/display, et

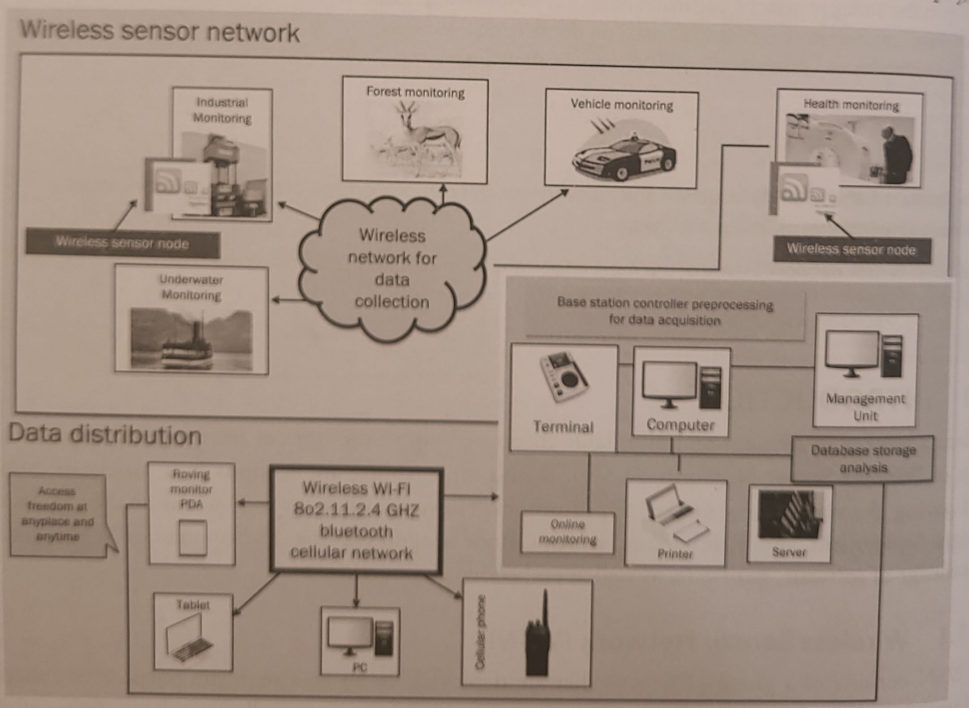


Fig. 3.1 Wireless Sensor Network

3.1.1.1 Categories of WSN Services

Wireless sensor networks are categorized into four major categories based on the services offered:

Monitoring WSNs can be deployed to acquire information about surrounding environment for monitoring purposes. For example, a WSN deployed for precision agriculture can monitor the meteorological conditions such as temperature, humidity, wind speed, etc. and send this information on an hourly basis. Similarly, a WSN for water quality information on a water body can make measurements of various parameters such as turbidity, dissolved oxygen, etc., and send this information twice a day. Such monitoring examples are ample in various domains and WSNs have proven to provide accurate, consistent and timely information which can be used for decision making purposes.

Alerting A WSN can be deployed in remote or in accessible terrains or disaster prone areas. The WSN nodes can be subscribed to send alerts when the measured parameter crosses certain threshold, for example water level above 10 meters for water level sensor nodes in flood a monitoring WSN. Similarly, alerts can be issued from a Tsunami monitoring WSN consisting of sensor nodes on Buoys deployed on the oceans. A WSN deployed for monitoring landslides can send advance alerts to avoid accidents and loss of life and property.

Information on Demand The network can be queried about the actual values of certain features of interest and it responds by providing information accordingly. For example, soil moisture values can be queried at a particular location, or soil moisture values in the last two days. Further, subsetting of WSN archived data based on current time instant, arithmetic, boolean, logical operators can also be performed.

Actuating Based on the context of the environment in which the WSN is monitoring, the sensor nodes can send a signal and change the behaviour of an external system. Some examples of actuations are (i) sensing weather conditions by a WSN node may trigger a sensor node to capture GPS location of the mobile entity on which it is deployed, (ii) activating a sensor such as a camera to acquire data for specific monitoring purposes.

3.2 CHARACTERISTICS OF WIRELESS SENSOR NETWORK

WSN is deployed to gather information from unattended physical environment. Hence, for efficient design and deployment of WSN, it is necessary to take into consideration some unique characteristics of WSN.

3.2.1 Significant Characteristics of WSN

Energy efficient In WSN, energy is used for various tasks such as sensing, communication, computation, and storage. If the sensor nodes run out of power they cannot be recharged at a remote location, and consequently, can fail to sense the surrounding physical environment. Hence, to reduce the power consumption of a sensor node, efficient protocols and algorithms are used while designing a WSN.

While designing energy efficient WSN, the following should be kept in mind:

- All remotely deployed sensors are generally battery operated.
- Lifetime of the sensor node depends on the battery lifetime. Sometimes a nearby power source may be available for charging the battery.

- Lifetime of the sensor is an important factor while designing a WSN for its optimal use and high efficiency.
- The power is utilized mainly for sensing, computation, and communication.

Low cost Generally, for measuring the physical environment, a large number of sensor nodes are employed in a WSN. It is desirable to have the cost of the individual sensor nodes as low as possible and consequently reduce the overall cost of the network. Further, deployment cost also needs to be taken into account while using a large WSN.

Computational power Typically, the sensor node has limited computational capabilities and is usually decided by cost, size, and energy of the node.

Communication capabilities Generally, radio waves are used for communication in WSN. The communication channel can be unidirectional or bidirectional. Sometimes, WSN is used in a remote and inaccessible terrain environment; hence, it must be robust and resilient enough to quickly recover in case of failure of some of the nodes of WSN.

Cross-layer design This type of design is recent emerging in wireless communication as it improves WSN performance in terms of energy efficiency, datarate, Quality of Service (QoS), etc. The traditional layered approach faces problems such as:

- In traditional layered approach, network optimality cannot be ensured as there is no data communication to share information among different layers leading to incomplete information at each layer.
- Dynamically changing environment cannot be adapted by traditional layered approach of WSN.
- Other limitations include interference between different users, access conflict, fading, etc.

Distributed sensing and processing To enable robustness and resilience, the sensor nodes in a WSN are distributed randomly and uniformly. A sensor node sends the data to a sink node which is capable of collecting, processing, sorting, aggregating, and sending the data.

Security and privacy Sufficient security must be provided to each sensor node to protect from unauthorized access, unintentional damage, and malicious attacks to damage the information inside the node.

Dynamic network topology A sensor node can fail due to battery exhaustion or physical or sensor data tampering. To overcome such issues, WSN nodes must have the capabilities to dynamically reconfigure and adjust itself under various conditions such as addition of new nodes or replacement of existing ones.

Multi-hop communication In case a node requires to communicate with another node or a base station beyond its radio frequency range, then to reach to the sink node, a multi-hop route is needed.

Self-organization Sensor nodes should have the capability to self-organize themselves in a collaborative fashion to fit in a distributed topology to form a self-adjusted network for optimal performance. For example, if some sensor nodes fail, replacement with new nodes becomes necessary and has to be carried out autonomously. These new sensor nodes need to have self-configuring capability to adjust with the current network topology of other sensor nodes, and able to support the communication regardless of energy constraints.

Robust operation WSN nodes are left unattended most of the time and are prone to physical damage, battery drainage, communication failures, and impact of harsh environment. Therefore, it is necessary to have sensor nodes with capabilities of fault and error which is a measure of its reliability. The sensor nodes must have the capability of self-testing, self-configuring, and self-repairing for robust use.

Small physical size A wireless sensor node has four main units: sensing, processing, communication, and energy/power. All these factors have to be considered while designing a sensor mode and it in turn determines the size of the node. Future sensor nodes are envisaged to be as small as the grain of sand.

Application oriented WSN is highly dependent on the specific application (e.g., military, health, environmental, etc.) for which it is designed.

3.3 TYPES OF WSN AND THEIR ARCHITECTURE

Depending on the application environment, the type of WSNs are chosen. Broadly, the various types of WSNs are:

- Multimedia WSNs
- Mobile WSNs
- Terrestrial WSNs
- Underwater WSNs
- Underground WSNs

3.3.1 Multimedia Wireless Sensor Networks

Multimedia WSN is used in situations where tracking and monitoring of the events is performed using multimedia (images, audio, and video). These WSNs consist of low cost sensor nodes with devices such as camera and microphone. The sensor nodes are connected to form a network using wireless connections to collaborate with each other for performing data pre-processing operations such as data retrieval, correction, and compression. However, these are challenging to perform as they require high bandwidth and energy (see Fig. 3.2).

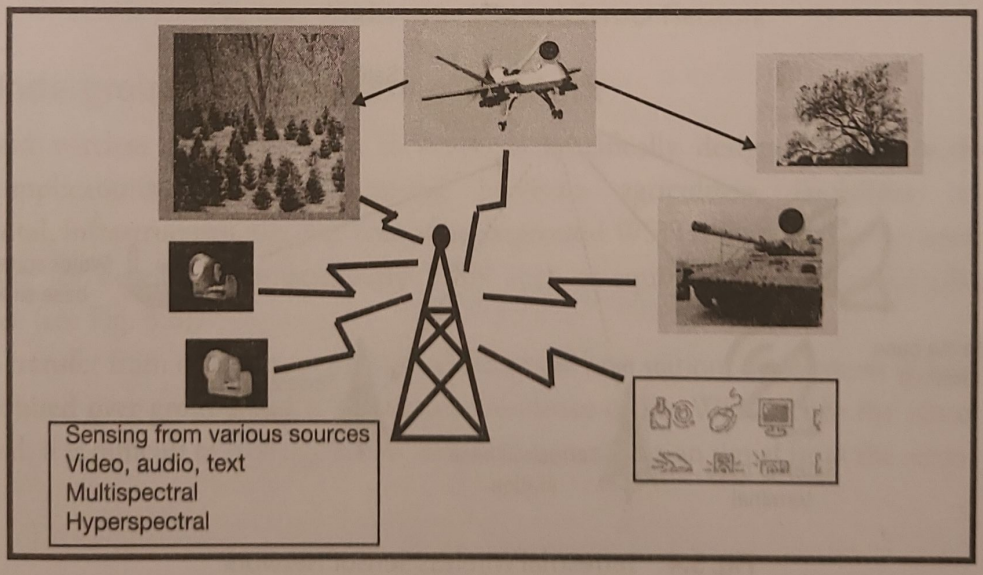


Fig. 3.2 Multimedia Wireless Sensor Network

3.3.2 Mobile Wireless Sensor Networks

A collection of sensor nodes that can move on their own and interact with the surrounding physical environment forms a mobile WSN. The mobile sensor nodes have computing and communication capabilities (see Fig. 3.3).

As the sensor nodes can move, mobile WSNs have wide versatile applications than static sensor networks as these WSNs have improved coverage of the surrounding environment, better energy efficiency, and channel capacity.

3.3.3 Terrestrial Wireless Sensor Networks

Terrestrial WSN is capable of communicating efficiently with the base station. It can have a very large number of sensor nodes, which are randomly distributed for covering the required terrain for effective mapping. These nodes are deployed in a structured or unstructured way in a target area to optimally acquire data for a specific application. The deployment of a structured WSN requires considering various optional models such as 2D, 3D, or grid placement to gain maximum efficiency (see Fig. 3.4).

Further, in this type of WSN, the sensor nodes have limited power and solar panels are usually used as additional sources of power. The energy is managed by approaches such as optimal routing, minimizing delays, low duty cycle operations, repeaters etc.

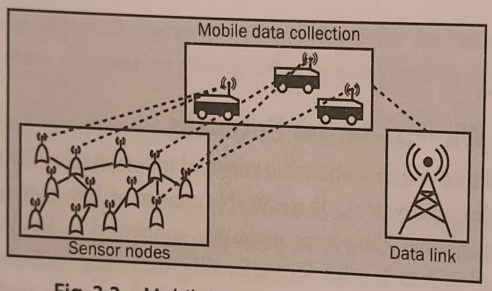


Fig. 3.3 Mobile Wireless Sensor Network

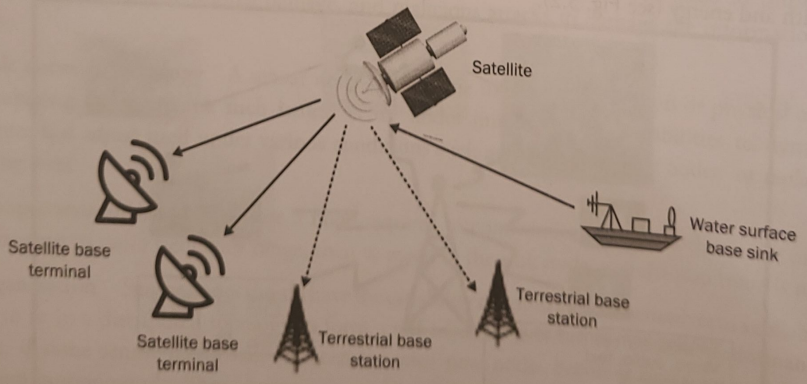


Fig. 3.4 Terrestrial Wireless Sensor Network

3.3.4 Underwater Wireless Sensor Network

Although more than 70% of the earth is covered by water, its monitoring is not as advanced as that of the terrestrial land mass. This is due to the huge challenges involved in deploying sensors underwater. A WSN where the sensor nodes are deployed under water is called an underwater WSN (see Fig. 3.5). Specific challenges of underwater WSN are long propagation delay, bandwidth limitations, and sensor node failure. For designing and developing underwater WSN, the major issue is energy conservation. The data is gathered from these sensor nodes by using autonomous underwater vehicles. For example, in the ocean observing systems, both drifting as well as moored buoys are used to measure various marine, biological, and water parameters.

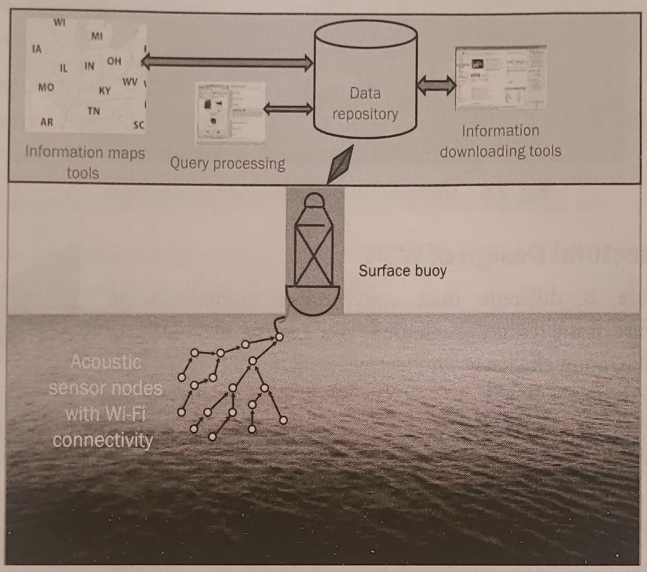


Fig. 3.5 Underwater Wireless Sensor Network

3.3.5 Underground Wireless Sensor Network

Underground wireless sensor network (UWSN) is specifically designed for subsurface region. Potential applications include monitoring precision agriculture, landslides, earthquakes, environmental, infrastructure, etc. The cost of underground WSN is high due to the need for special infrastructure for developing underground WSN such as equipment cost, careful planning, and maintenance (see Fig. 3.6).

For data transfer from underground sensor nodes to the base station, deployment of additional sink nodes is required over ground and it also secures resilience of the WSN. Since the sensor nodes are underground, it is difficult to recharge them, also the communication signal from the sensor nodes gets attenuated.

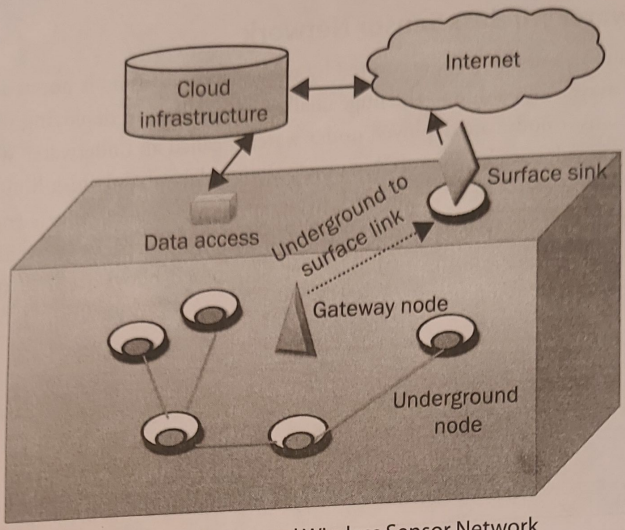


Fig. 3.6 Underground Wireless Sensor Network

3.3.6 Architectural Design of WSN

WSN architecture is different than conventional communication architecture because of the resource limitations that are imposed on it, yet it has to deliver maximum efficiency with less overhead. Hence, WSN does not strictly follow the layered architecture of Open Systems Interconnection model. However, for categorizing protocols and adding security features for reliable communication, the layered model is very useful. Figure 3.7 below shows the communication protocol model for WSN.

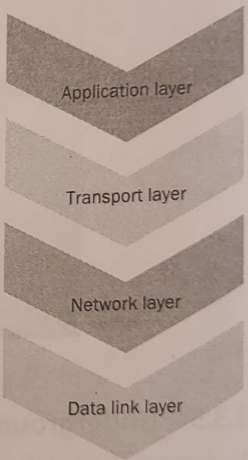


Fig. 3.7 Communication Protocol Stack for WSN

The structure of a sensor node has a sensing unit (with/without analogue-to-digital converter), processing unit (processor and storage), communication unit, and a power supply unit. The attributes such as low cost, fault tolerance and low energy consumption should be considered while designing a sensor node. The block diagram for sensor node structure is shown in Fig. 3.8.

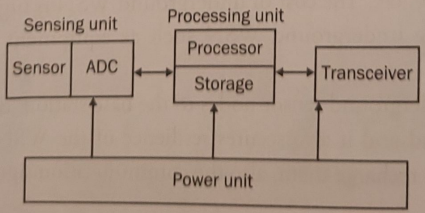


Fig. 3.8 Structure of a Sensor Node

3.3.6.1 Sensor Device Unit

The sensor unit consists of an analogue or/and a digital sensor and optionally an analogue-to-digital converter unit (ADC). A suitable sensor is selected as per the need of the application by thoroughly studying the constraints of the working environment.

3.3.6.2 Communication Unit

A transceiver unit is made up of a transmitter and a receiver. Communication of information is performed using network protocols. A suitable communication method is selected (such as radio, infrared, optical, etc.) based on the requirements of the application.

3.3.6.3 Processing Unit (A Microcontroller Unit and Storage)

It also has an operating system and a timer. The microcontroller unit collects data from many sources and then processes and stores the data.

3.3.6.4 Power Unit

The power unit supplies energy to the sensor node for monitoring the physical environment at a low cost. The sensor nodes life is dependent on the life of the battery.

Some important goals to be considered while designing WSN architecture are:

- requirements analysis of the specific application/domain for which the WSN is to be designed
- quantitative analysis of the needs of the application
- understanding and selecting latest and relevant technologies that can suit the intended application, and achieving optimal configuration by selection of hardware and software recommended best practices.
- developing a best fit solution that takes into consideration various parameters and alternatives (design costs and other WSN constraints) in a complex and heterogeneous WSN deployment scenario.
- employing power optimisation strategies that are tailored to the needs of a particular application

3.4 NETWORK TOPOLOGIES IN WIRELESS SENSOR NETWORK

Wireless sensor networks are classified into four basic network topologies categories based on communication pattern: one-way, bi-directional, star, and mesh.

3.4.1 Different Types of Topologies in WSN

Traditional topologies in network have been adapted with modification wherever required for development and deployment of WSNs. Different topologies in WSN are: Bus, Tree, Ring, Star, Mesh, Circular, and Grid.

3.4.1.1 Bus Topology

In bus topology, a sensor node sends a broadcast message to the nodes on the network and only the intended recipient accepts the message. Bus topology is easy to design and install but not good to take care of traffic and congestion. When limited nodes are used in the WSN, bus topology works best (see Fig. 3.9).

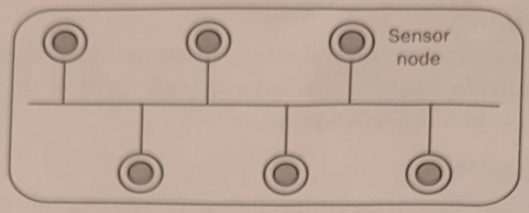


Fig. 3.9 Bus Topology

3.4.1.2 Tree Topology

In this network topology, there is a central hub as a root node and it acts as the main router. The tree network is a combination of star and peer-to-peer network topologies. In distributed implementation, tree topology is more suitable. However, load-balancing problem is faced in fat trees while communicating between sensor nodes (see Fig. 3.10).

The intermediate nodes need to work more (depending on the number of sub nodes) to forward the information and may gradually loose power and have to be replaced. The leaf nodes do not participate in routing. The root node acts as a sink/ base station.

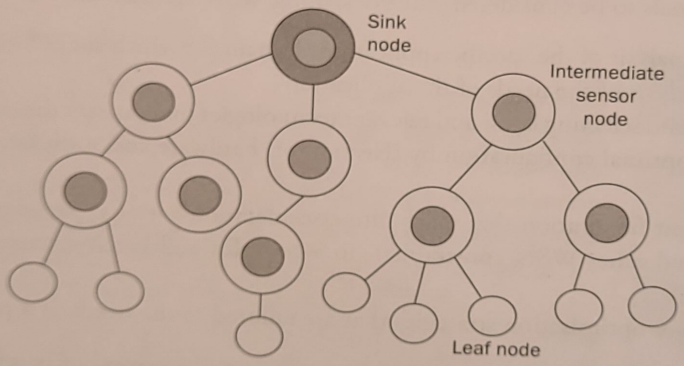


Fig. 3.10 Tree Topology

3.4.1.3 Star Topology

The star network is connected to a centralized hub also called as sink. In this network, the nodes cannot communicate with each other directly as the entire communication is routed through the central communication hub. In this topology, each sensor node acts as a client and the sink node acts as a server (see Fig. 3.11).

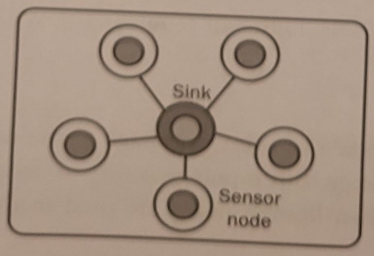


Fig. 3.11 Star Topology

3.1.4.4 Ring Topology

In a ring network, every node has exactly two neighbours for communication (see Fig. 3.12). All messages travel through the ring in a single direction (clockwise or anti-clockwise). A failure in a node breaks the loop. Congestion of traffic is taken care by a variant of ring network as a double path communication (clockwise as well as anti-clockwise).

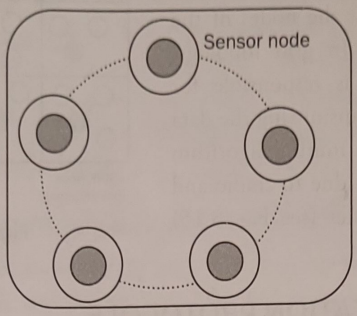


Fig. 3.12 Ring Topology

3.1.4.5 Mesh Topology

In a Mesh network, the message can take multiple paths from source to destination. A mesh in which every node is connected to every other node is known as a full mesh. Mesh network is useful for load balancing, reliable data transmission, and minimizing energy consumption (see Fig. 3.13).

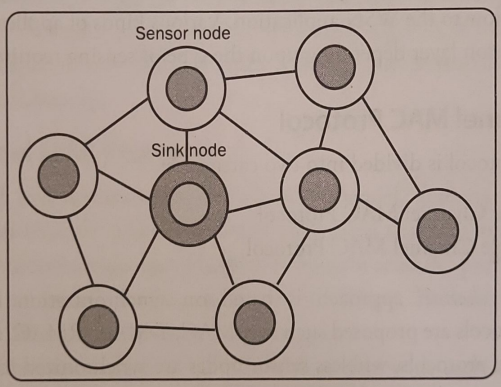


Fig. 3.13 Mesh Topology

3.1.4.6 Circular Topology

In a circular network, there is circular sensing area and every such sensing area has a sink node. The event information is captured by sensor node and sent to the sink node. Distribution of the nodes is random and uniform. This topology is easy to install and maintain. It is an energy-efficient topology. Short beacon messages are sent to the nodes to check the location of the node (see Fig. 3.14).

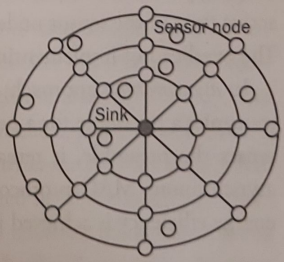


Fig. 3.14 Circular Topology

3.1.4.7 Grid Topology

In the grid network, the sensor network field is divided into non-overlapping square grids of same size. In each grid, at least one node should be in active state. To increase life span of the network and to make it energy efficient the nodes of the grid become active in turns. In grid topology, a node called as head node is responsible for routing the information and transmitting the data packets. Grid-based multi-path routing algorithm is efficient to avoid congestion due to traffic and also good for fast packet transfer (see Fig. 3.15) (Sharma et al., 2013).

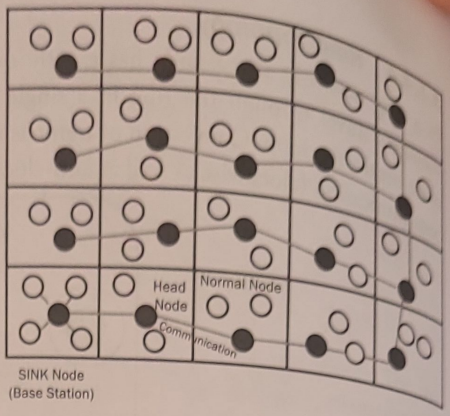


Fig. 3.15 Grid Topology

3.5 WSN COMMUNICATION PROTOCOLS

The primary goal of communication is to optimize the network to work in an energy efficient way so that WSN lifetime is increased. The physical layer of WSN is about the communication hardware. The physical layer is responsible for signal detection, carrier frequency generation, modulation, frequency selection, and data encryption. Medium Access Control (MAC) protocols are applied in data link layer. Routing of data coming from transport layer is managed by network layer. The transport layer is responsible for the data flow to the WSN application. Various kinds of applications and functionalities are supported by application layer depending upon the type of sensing required.

3.5.1 Single Channel MAC Protocol

Single channel MAC protocol is divided into two categories:

- Synchronous Single Channel MAC Protocol
- Asynchronous Single Channel MAC Protocol

The *synchronized single channel* approach is based on synchronization between nodes. Several synchronized MAC protocols are proposed such as S-MAC, T-MAC, RMAC, and DW-MAC (demand wake up MAC). In these protocols, wireless sensor nodes are synchronized to schedule the active and sleeping period of nodes to make the WSN, energy efficient. The exchange of data happens only in the active time of the sensor node. This approach saves energy by reducing idle listening time of each node. The overhead of implementing synchronous protocol is substantial.

In *asynchronous* approach, each sensor node wakes up independently as per its duty cycle. A sender transmits a preamble for a period of sleeping of the receiver node before transmitting data. If receiver senses the preamble, it remains active and becomes ready to receive the data. In receiver initiated asynchronous MAC protocol, a receiver starts a transmission by sending a control message. Thus, energy efficiency is achieved in asynchronous MAC protocol without synchronization.

3.5.2 Asynchronous Single Channel MAC Protocol

Each sensor node in a network periodically checks the availability of the wireless medium according to its own schedule. If the medium is busy, it waits further until it becomes idle or a data packet arrives from the sender. The sender assumes that in such a case the receiver is not ready to receive the data even after receiving long preamble, due to network errors. This problem consumes energy. WiseMAC protocol utilizes the preamble sampling method to overcome this problem. In this method, dynamic length preambles are utilized to reduce idle listening time. In dynamic length preamble, the sensor nodes keep a track of sleep-live schedule of their neighbours. The drawback of WiseMAC is buffering time of the packet increases in a sender buffer. While broadcasting, the sender sends the packets to all its neighbours and that causes waste of energy.

3.5.3 Pseudorandom Asynchronous MAC Protocol

This protocol uses a hash function to decide the next wake up time. The next wake up time decided by the hash function is non-periodic and this way the pseudorandom asynchronous MAC protocol saves energy wasted in idle listening and even collision is reduced.

3.5.4 Medium Reservation MAC (MRMAC)

In this protocol, additional information about next packet arrival time (NPAT) and medium reservation information (MRI) is there in the beacon message to reduce the end-to-end latency in delivery of message. Due to this additional information in the beacon message, every node in WSN knows when the wireless medium is idle, so each node decides its own flexible schedule of transmission and reception, hence idle listening and collision is reduced. MRMAC shows better performance than RI-MAC if the network has periodic traffic pattern.

3.5.5 Multi-channel MAC Protocols

As single channel–based communication uses only one channel, capacity and throughput of the network is less. The multi-channel radio uses several orthogonal channels and divides the bandwidth into multiple channels. Adjacent nodes use different channels to send the packets. Thus, multi-channel MAC protocol improves network throughput and reduces collision. The multi-channel MAC protocol is efficiently designed for WSN by considering following points.

Multi-channel networks are orthogonal and do not interfere with each other. The nodes cannot communicate with each other when they belong to different channels, the nodes should be assigned same channel for communication. There are three methods for channel assignment: fixed, semi-dynamic, and dynamic. In the *fixed* assignment, approach is based on cluster approach. The sensor nodes are divided into several clusters. The cluster communicates through the assigned channel. This approach prevents interference between clusters. In *semi-dynamic* assignment approach, each node is assigned to a certain channel for transmission and reception. While in *dynamic* channel assignment approach, each node is assigned different channel after every wake up schedule of the node. The main advantage of multi-channel protocol is that there is less interference and collision while transmitting and receiving the data. However, in semi-dynamic and dynamic approach, due to switching between channels, there is energy overhead.

3.5.5.1 Flooding

It is a robust algorithm that delivers data packet delivery from source to destination in a network. In flooding, the WSN node that receives the message broadcasts it to all its neighbours. That causes unnecessary retransmission of messages and increased collisions causing waste of limited battery power of sensors. Hence, for dense WSN, flooding algorithms are not useful.

3.5.5.2 Gossiping

Some critical problems of WSN are handled by the gossip protocol. The main goal of gossip protocol is to reduce retransmissions. In gossip protocols, some of the nodes discard messages and do not forward all messages received by it. Gossip protocols are of two kinds: nondeterministic and probabilistic.

In *nondeterministic* behaviour, any probabilistic distribution is not followed for scheduling the wait time, before retransmitting the packet, if the sink node is not active. On the contrary, in *probabilistic* behaviour, based on pre-specific gossip probability p_{gsp}, the packets are forwarded by a node. When a node receives a message rather than forwarding it immediately as in *flooding*, it gives the decision control to p_{gsp}, whether to forward the packet or not. The main advantage of *gossiping* protocol is that it is easy to implement and reduces energy loss overhead due to unnecessary retransmissions. Gossiping protocol suffers from latency, propagation delays. Hence, based on specific application requirement a suitable choice of protocol is used.

3.5.6 Routing Protocols in WSN

Many routing protocols have been developed for WSN. Various challenges and constraints are to be taken into consideration while designing routing protocols. Wireless links are unreliable; consequently, failure of a sensor node may be possible. All major routing protocols are listed in Table 3.1 and the design issues in all these protocols are listed in Box 3.1. They are classified into seven categories.

Table 3.1 Routing Protocols for WSNs

Category	Representative protocols
Location-based protocols	MECN, SMECN, GAF, GEAR, Span, TBF, BVGF, GeRaF
Data-centric protocols	SPIN, directed diffusion, rumour routing, COUGAR, ACQUIRE, EAD, information-directed routing, gradient-based routing, energy-aware routing, information-directed routing, quorum-based information dissemination, home agent based information dissemination
Hierarchical protocols	LEACH, PEGASIS, HEED, TEEN, APTEEN
Mobility-based protocols	SEAD, TTDD, joint mobility and routing, Data MULES, dynamic proxy tree-base data dissemination
Multipath-based protocols	sensor-disjoint multipath, braided multipath, n-to-1, multipath discovery
Heterogeneity-based protocols	IDSQ, CADR, CHR
QoS-based protocols	SAR, SPEED, energy-aware routing

(Source credit: International Journal of Computer Science and Engineering Survey (IJCSES))

3.5.7 Operating Systems for WSN

Operating systems of WSN are less complex as compared to any general purpose OS. WSN OS are like embedded system OS.

- **TinyOS** is an operating system specially designed for WSN. TinyOS software has event handlers and tasks with run-to-completion semantics based on event-driven-programming model.
- **LiteOS** is another newly developed OS for WSN. It has UNIX like environment and uses C programming language.
- **Contiki** is an OS that supports advanced technology such as 6LoWPAN and Prothreads.
- **RIOT** is a state-of-the-art, real-time OS with functionality same as Contiki.
- **PreonVM** OS supports Java programming language specifically designed for WSN and provides 6LoWPAN based on Contiki.

3.5.8 Simulation of WSN

Simulation of WSN is possible by some simulators such as NS, Opnet, and Tetcos NetSim. Refer to Box 3.1 for WSN simulation design steps in NS2 simulator for a sensor network.

BOX 3.1: WSN SIMULATION DESIGN STEPS IN NS2 SIMULATOR FOR A SENSOR NETWORK

1. **ENERGY MODEL**
   Creation of multiple nodes
2. **Communication model using UDP (User Datagram Protocol) and CBR (Constant Bit Rate)**
3. **Simulator instance creation**
   Fixing the coordinate of simulation area
   Define options
   Channel type, Radio-propagation model, Network interface type, MAC type
   Interface queue type, Link layer type, Antenna model, Max packet length
   Number of mobile nodes, Routing protocol,
   X dimension of topography, Y dimension of topography
   Time of simulation, Energy set up
   Setup topography object
   General operational descriptor for storing the hop details in the network
4. **Set up transmission range**
5. **Set up Antenna**
6. **Set up communication and sensing range**
   Default communication range
   Initialize the shared Media interface with parameters
   Sensing range, Communication range
7. **Configure the number of nodes with transmission range**
   Node creation
8. **Configure the remaining nodes with lesser transmission range**

3.6 SECURITY IN WSN

Several threats are encountered by WSN such as eavesdropping, node outrage, message corruption node malfunction, denial of service, and false node. WSN are prone to threats due to infrastructure-less architecture and flaws due to unattended working environment. Hence, security is a primary concern when WSN is to be deployed for special and critical applications. Mitigation methods are needed to avoid attacks due to intrusion and other threats.

Sensor nodes, when deployed in a hostile environment are prone to different malicious attacks. Hence, to ensure security to the sensor network, key management is applied.

Since WSN sensor nodes are distributed, trust in WSN is an important issue. It solves the problems of access control, secure routing, privacy, and reliable communication. Figure 3.16 shows major trust computation steps for WSN.

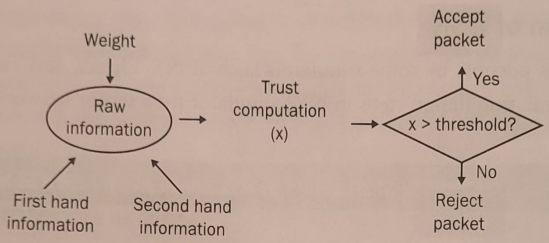


Fig. 3.16 Trust Computation in WSN

3.7 REAL WORLD WSN APPLICATIONS

Wireless sensors are used for measuring various environmental conditions such as temperature, humidity, pressure, and wind speed. Such networks were built initially for military applications and now-a-days they are used for many industrial and consumer applications.

3.7.1 Applications of Wireless Sensor Network

Wireless sensor networks have a wide domain of applications. Some of the major areas are explored below.

3.7.1.1 Healthcare Monitoring

Many types of sensor networks can be used for health care solutions such as wearable, implanted, and environment-embedded devices. The wearable devices are used on the human body surface or placed in close proximity of a human. Implanted devices are inserted in the human body. The sensors contained in the environment are the environment-embedded system sensors. The application areas are body position measurement, finding location of a person, and monitoring of patients in hospital and ill people at home. Environment-embedded system has application such as tracking of a person for continuous health diagnosis. Authenticity and privacy of data are important concerns in health care applications.

Area monitoring It is a common application of WSN where wireless sensor network is deployed over a specified region as per the application, for example, geo-fencing of gas and oil pipelines.

Earth/Environmental sensing For monitoring physical environment around is required in many applications. Some additional constraints as harsh environment and limited power are to be taken care. Some examples are listed as follows:

- **Detection of forest fires:** A network of sensor nodes is installed in a forest area to sense the outbreak of the forest fire. The sensor nodes can also sense temperature, humidity, dryness, solar index, etc., to predict any possibility of the fire based on these climatic conditions and type of vegetation.
- **Air pollution monitoring:** At several cities, concentration of dangerous gases is to be monitored constantly with a certain frequency to decide the air pollution level. This can be done by ad hoc wireless links instead of wired network. This ad hoc wireless network can be made mobile to cover different areas.
- **Water quality monitoring:** Properties of water in rivers, dams, lakes, and oceans are monitored for various reasons. Distributed wireless sensor node network is useful for sensing different parameters to decide water quality. The water samples from unreachable remote places even can be checked with such sensor network.
- **Natural disaster prevention:** Natural disasters such as floods and earthquakes can be predicted from the information collected by sensor nodes. WSN installed in the disaster prone areas for taking precautionary measures that can save loss of different kinds.
- **Landslide detection:** WSN sensor nodes can detect even a slight movement of soil and changes in other environmental parameters to detect a possibility of landslide well before it may happen.

3.7.1.2 Industrial Monitoring

Data centre monitoring In data centre with plenty of server racks cabling, IP addresses become a critical issue. To overcome this problem, wireless temperature sensors are fitted on, as many racks as possible to monitor intake and outtake temperature of racks. Wireless sensor with Mesh network is useful to address this issue effectively.

Data logging The live data feed from different sensors and its logging for statistical analysis is useful to show how systems have been working. This analysis is useful when design changes are needed to evolve the system in future.

Machine health monitoring Maintenance of machinery can be easily planned for best performance and long life of machinery by using the parameters checked by sensor network to decide the machine health.

Wastewater monitoring Wastewater is produced by industrial processes. Quality of such water is required to be monitored before recycling this water for any use.

Structural health monitoring Condition of any civil infrastructure can be monitored by WSN by monitoring different geophysical processes in real time and over long time by logging the data using appropriate sensors.

Wine production Wine production processes can be monitored using WSN for maintaining the desired quality of wine.

3.8 EVOLUTION OF WSN TOWARDS INTERNET OF THINGS

Wireless sensor networks are playing an important role in many applications from various domains such as healthcare, agriculture, smart metering, etc. Furthermore, high heterogeneity is present in WSNs because there are many proprietary and non-proprietary solutions for different applications. Current trend is heading towards IP-based sensor networks using the emerging standard 6LoWPAN/IPv6 instead of proprietary and closed standards. Native connectivity between WSN and Internet enables every day things to become smart to participate in IoT.

Recent research activities are aimed at creating networks of things web technologies such as AJAX, Javascript, Ruby, and PHP can be used to build applications with network of things and can be shared using Web mechanisms. Applications based on RESTful principles are already contributing towards the success of IoT.

THOUGHT EXERCISES

1. Design a suitable Wireless Sensor Network architecture for a Home Automation application using ZigBee communication protocol over mesh topology and mention the advantage and disadvantages of it over Wi-Fi communication.
2. Design a suitable WSN with MAC protocol for broadcast transmission for any suitable application. Choose the suitable routing algorithm for this WSN model and justify the selection.
3. Device the security measures to be taken for data confidentiality and integrity in any WSN based healthcare applications.
4. Suggest two WSN based applications, one using Broadcast protocol and another using Gossip protocol. Justify your selection of the protocol for that application.

SUMMARY

WSN is a backbone of IoT-based applications; hence in this chapter, several characteristics of WSN are explored. While designing the WSN, architecture, the characteristics especially those which are useful for energy efficient and reliable data communication are given prime importance. Various challenges are to be overcome while designing and deploying WSN, useful for different kinds of applications. Hence, different WSN topologies are listed. WSN MAC layer protocols are covered and network layer routing protocols are visited. WSN can be deployed in a wide variety of domains some of which are discussed in this chapter. The importance of data aggregation for energy efficient WSN is described. Finally the evolution of of WSNs towards IoT is presented.

KEYWORDS

Wireless sensor network, WSN topology, MAC protocols, routing protocols, sensor node, base station data aggregation, WSN applications

FURTHER READING

1. Singh, S., Sharma, S. (2015), A survey on cluster based routing protocols in wireless sensor networks. Procedia Computer Science 45: 687–695.
2. Kiran, M., Kant, K., Gupta, N. (2011), Efficient cluster head selection scheme for data aggregation in wireless sensor network. International Journal of Computer Applications 23(9): 10–18.
3. Kong, D., Kong, J., Chen, W., Niu, L. (2014), A design on improved GAF algorithm by wireless sensor network. Microelectronics and Computer 31: 147–152.
4. Chen, Y., Ye, Q. (2014), Summary on security authentication scheme for wireless sensor networks. Journal of Computer and Digital Engineering 42: 261–266.

REVIEW QUESTIONS

1. What are the major challenges while designing WSN?
2. Describe classes of WSN based on the service offered by them.
3. Explain the characteristics of wireless sensor network and their role in design and development of WSN.
4. Explain requirements for designing and developing energy-efficient WSN.
5. What are the objectives of WSN architecture?
6. What are the different WSN topologies? Explain their major strengths and weaknesses.
7. Explain different network layer routing protocols for WSN. Describe their advantages and disadvantages based on specific application.
8. Explain the MAC protocols used in WSN and classify them according to their application areas.
9. Explain duty-cycling technique of MAC protocol for designing energy optimized WSN.
10. Compare the Flooding and Gossiping protocols.
11. Explain any application area for WSN in detail with its requirements.
12. Explain the important goals to be achieved by a good WSN architecture.
13. If data aggregation is applied in WSN then what are the security issues surfaced?
14. What are the major security issues while sending or receiving the data?
15. What are the different applications of mobile WSN?
16. Explain the energy loss happening during idle listening. What are the methods that can reduce this energy loss?

REFERENCES

1. Toor, A., Jain, A. K. (2015), A review on wireless sensor network. *Journal of Mobile Computing Communications & Mobile Networks* 2(2): 10–13.
2. New frontier for wireless sensor networks [ONLINE]. Available at: http://www.networkworld.com/news/2004/0607sensors.html
3. Wireless sensor networks: a survey [ONLINE]. Available at: http://www.ece.gatech.edu/research/labs/bwn/sensornets.pdf
4. Suri, P., Bedi, R., Gupta, S. (2012), Review paper on various clustering protocols used in Wireless Sensor Network (WSN). In Networks (ICON), 2012 18th IEEE International Conference on Electrical, Electronics, Signals, Communication and protocols in Wireless Sensor Networks, pp. 86–91.
5. Akkaya, K., Younis, M. (2005), A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks* 3(3): 325349.

IoT Standards and Protocols

CHAPTER 4

"The protocol of science fiction and the protocol of science are not separate- they'r e woven together."

—Ellen Galagher

OBJECTIVES

- introduce Internet principles and overview of IPv4 TCP/ IP protocol stack
- explain the role of IPv6 in Internet of Things (IoT) applications
- describe low-power wide area network (LPWAN)
- overview of various wireless technologies for IoT

OUTCOMES

- describe IPv4 address mechanism
- explain the contribution of IPv6 in IoT
- explain the developments in low-power wide area network
- elaborate wireless technologies for IoT

REVISION

Chapter 3 covered wireless sensor network and its various types and application domains as it plays major role in industrial Internet of Things (IoT) applications. In Chapter 4, the focus is to explore the Internet principles and various protocol standards suitable for IoT applications.

4.1 AN OVERVIEW OF INTERNET PRINCIPLES

Internet of Things (IoT) focuses on developing suitable network technologies for widespread deployment of smart devices through Internet connectivity using various technologies such as RFID, short-range wireless communication, and sensor networks. IPv6 Internet protocol with its address space expansion can cope with the requirement of huge number of addresses, which will support the increasing demand of the IP addresses. In IoT paradigm, network technologies must be capable of connecting anything to the network. Hence, security, scalability, and compatibility with heterogeneous platforms are essential requirements. Mission critical applications can be managed with reliable and secure wireless communication protocol and ubiquitous sensor networks.

IoT Standards and Protocols

CHAPTER 4

"The protocol of science fiction and the protocol of science are not separate- they'r e woven together."

—Ellen Galagher

OBJECTIVES

- introduce Internet principles and overview of IPv4 TCP/ IP protocol stack
- explain the role of IPv6 in Internet of Things (IoT) applications
- describe low-power wide area network (LPWAN)
- overview of various wireless technologies for IoT

OUTCOMES

- describe IPv4 address mechanism
- explain the contribution of IPv6 in IoT
- explain the developments in low-power wide area network
- elaborate wireless technologies for IoT

REVISION

Chapter 3 covered wireless sensor network and its various types and application domains as it plays major role in industrial Internet of Things (IoT) applications. In Chapter 4, the focus is to explore the Internet principles and various protocol standards suitable for IoT applications.

4.1 AN OVERVIEW OF INTERNET PRINCIPLES

Internet of Things (IoT) focuses on developing suitable network technologies for widespread deployment of smart devices through Internet connectivity using various technologies such as RFID, short-range wireless communication, and sensor networks. IPv6 Internet protocol with its address space expansion can cope with the requirement of huge number of addresses, which will support the increasing demand of the IP addresses. In IoT paradigm, network technologies must be capable of connecting anything to the network. Hence, security, scalability, and compatibility with heterogeneous platforms are essential requirements. Mission critical applications can be managed with reliable and secure wireless communication protocol and ubiquitous sensor networks.

All data networks are based on Open Systems Interconnection (OSI) network standards. Reliable and efficient data communication is the major goal of the network. The OSI model conceptualizes and standardizes the network functions.

The OSI model architecture is of seven layers, physical, data link, network, transport, session, presentation, and application. Provision of each layer is for some network services. The Transmission Control Protocol/Internet Protocol (TCP/IP) model is based on packet switching. The sender sends a packet without any dedicated connection. The message data packets follow different best available routes in any order while reaching to the destination address. The higher layer protocol arranges the packets in order again.

4.1.1 Transmission Control Protocol/Internet Protocol (TCP/IP)

Transmission Control Protocol/Internet Protocol (TCP/IP) is a connection-oriented protocol which sends the data in packets (streams of bytes). The packet header information has sequence number of the packet and acknowledgment messages ensure packet delivery. TCP can retransmit the data packets if the packets have been lost, until reaching the timeout condition. TCP layer flow control mechanism reduces data transfer rate. Packet delivery information is transmitted to the application layer protocols by TCP layer. TCP detects duplicate messages and discards them properly. Due to these properties TCP is a reliable end-to-end protocol.

The TCP/IP model is different from OSI model. OSI model has seven protocol layers while TCP/IP is five-layered standard: application layer, transport layer, network layer, data-link layer, and physical layer. In TCP/IP model the application, presentation, and session layers are presented as application layer. In transport layer of TCP/IP model, the main protocols are TCP and User Datagram Protocol (UDP). This protocol defines the level of service and connection status while transmission of data. The Internet layer of TCP/IP protocol suite divides the messages into packets known as datagrams. The packet contains information of source and destination address to forward the packets to the destination. The Internet layer is responsible for routing packets. The main protocols at Internet layer are IP, Internet Control Message Protocol (ICMP), Address Resolution Protocol (ARP), Reverse Address Resolution Protocol (RARP), and Internet Group Management Protocol (IGMP).

Routers which is the main device of TCP/IP layer hascomponents such as CPU, RAM, flash memory, ROM, and interfaces.

4.1.2 IoT Reference Framework

The IoT reference framework consists of four main levels:

- Device level
- Network level
- Application service platform level
- Application level

IoT network level protocols are mapped with TCP/IP protocol layers as shown in Fig. 4.1. Data packets are formed and routed by IoT Network layer. The Internet layer address protocols such as IP version 4 (IPv4), IP version 6 (IPv6) are explored in section 4.2.1 and in section 4.2.2, respectively.

4.2 IOT NETWORK LEVEL (ADDRESSING PROTOCOL)

Internet Protocol (IP) is responsible for packet routing. In case of erroneous delivery of the packets, IP ensures error reporting, fragmentation and reassembly of data packets called datagrams for transmission over heterogeneous networks. IP addresses are globally unique numbers assigned by Internet Engineering Task Force (IETF).

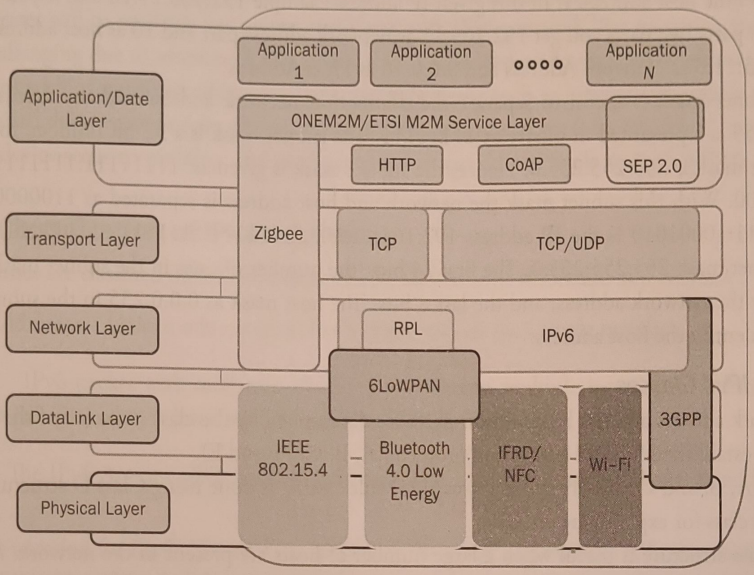


Fig. 4.1 IoT Layers and its Mapping to TCP/IP Layers

4.2.1 IP Version 4 (IPv4) Protocol

IPv4 addresses are represented in dotted-decimal format, for example 192.168.11.15 is a 4-octet (32-bit) IP address. This number uniquely identifies the destination whether it is a host (router, computer, printer, internet-enabled sensor or an output device using a network interface card (NIC) or any other TCP/IP network.

IPv4 addresses are made up of two parts namely network address and host address. A subnet mask extracts an IP address as a host or a network. This subnet mask helps the TCP/IP protocol to identify whether the host is in the local subnet or on a remote network. Because of IPv4 Subnet Mask in TCP/IP networks, routers passing packets from one network to another are unaware of the exact location of the destination address of the host. Routers have only information of the network of which the host is a member and the best possible route is assigned to the packet by checking routing table so that the packet can reach to the destination host in efficient way.

4.2.1.1 IPv4 Addressing

The packet is delivered to the destination network, and then the packet is delivered to the appropriate destination host. To identify network and host on that network, an IP address is made up of two parts namely network address and host address. The IP address such as 192.163.11.10 is represented in binary format as a 32 bit number 11000000.10100011.00001011.00001010. The first three binary octets represent an IP address and used as a network address, in Class C IPv4 addressing and the last octet shows the host address. If in this given IP address example 192.163.11.10 and if you will divide it into two parts two parts you get 192.163.11 as network address part and 10 as host address part host or 192:163:11:0 as Network Address and 0:0:0:10 as Host Address

The subnet mask is useful to separate the destination network address and the destination host address. 255 is represented in binary as 11111111. The subnet mask is a 32-bit number, for example, the subnet mask is 255.255.255.0. Hence, the subnet mask is given as: 11111111:11111111:111111 11:0000000. With this subnet mask the network and host address is separated as 11000000:101001 1:00001011:10001010 is the IP address 192:163:11:10. 11111111:11111111:11111111:00000000 is the subnet mask 255:255:255:0. The first 24 bits (the number of ones in the subnet mask) are used to identify the network address, and the last 8 bits. The host mask as 0.0.0.255 in the subnet mask is useful to identify the host address.

4.2.1.2 IPv4 Classes

The network address and the host address parts are determined by the class structure of the IPv4. Five classes are established in IPv4 addressing protocol (A, B, C, D, and E).

Classes A, B, and C identify actual networks. Multicasting is done using Class D structure. Class E is reserved class for experiment purpose.

The class structure is useful when a large number of hosts are present in the network. As the first bit of the octet is zero always, the octet value ranges from 1 to 127. Class A addresses are ranging from 1.x.x.x to 126.x.x.x and loopback IP addresses are in the range 127.x.x.x. 8-bit address is used in Class A networks for the network address purpose and the remaining 24 bits are used to represent host address. 255.0.0.0 is a default subnet mask for Class A IP address.

In Class B, two octets are for the network addresses and two octets are for host addresses. Hence, the default subnet mask for Class B addresses is 255.255.0.0. Class C allocates three octets for network address field and one octet for host addresses. The default subnet mask of Class C is 255.255.255.0.

4.2.2 IP Version 6 (IPV6) and its Role in IoT

With the advent of IPv6 protocol and gradual replacement of IPv4 with IPv6 protocol, it will have an impact on the future internet communications. IPv6 is an important protocol for IoT as interconnected products have huge demand of IP addresses and could not be met by IPv4.

Security Security is an important issue for smart IoT technology-based products. Also, due to their wide range of applications, privacy is a great concern. IPv6 offers better security solutions than its predecessor IPv4. With IPv6 end-to-end encryption is possible and the attacks such as man in the middle attack and cyber trap are made significantly difficult. IPv6 uses secure name resolution. the SEcure Neighbor Discovery (SEND) protocol is useful for authentication for a host which is secured with cryptography. In IPv4, it is easy to redirect traffic between legitimate users and manipulate the data or at least eavesdrop. However, IPv6 protocol deters such attacks.

Scalability As there is exponential growth expected for IoT objects in next 5 years, IPv6 protocol will become a need for meeting the demand of addresses. A unique identifier is assured for IoT and high scalability in the future.

Connectability The capacity of IPv4 is about 4.3 million addresses which cannot cater to the growing number of IoT devices. Hence, IETF worked on the successor IP protocol IPv6. It is a 128-bit address IP protocol with huge addressing capacity of 340 trillion trillion addresses. A large number of IP addresses will be available per square meter area with IPv6 protocol. The transition from IPv4 to IPv6 is challenging due to interoperability issues as they were designed separately. However, transition mechanisms such as multicast addressing (with simplified way such as hierarchical address allocation) and optimized delivery service are devised for communication between IPv6 and IPv4. Further, issues related to device security, mobility, and configuration, are considered while designing IPv6.

4.2.3 Classification of IPv6 Addresses

IPv6 addresses are classified mainly into three categories: Unicast, Multicast, and Anycast.

Unicast Addresses Unicast address identifies a single address for a single interface.

Multicast IPv6 packet with multicast address is delivered to multiple interfaces. Multicast address identifies a group of address interfaces belonging to different nodes.

Anycast The IPv6 Anycast address delivers the packet to one of the interfaces identified by Anycast address.

Address notation for IPv6: 128-bit-long address of IPv6 with eight blocks of 16 bits each. These 16 bits are converted to four digit hexadecimal number, separated by colons, unlike IPv4 address where the separator is a dot.

4.2.3.1 IPV6 Address Example

Generally IPv6 address is represented in its final form by removing leading zeros. An example of 128 bit IPv6 address in final form is as below:

Final form (simplified by removing the leading zeros):
83DA:D3:0:3F3B:3AA:FF:FE28:9C58

Binary form:
1000001111011010000000001101001100000000000000000011111100111011 0000001110101010000000011111111111111111000101000100111000101100 0

16-bit boundaries form:
1000001111011010 0000000011010011 0000000000000000 0011111100111011 0000001110101010 0000000011111111 1111111000101000 1001110001011000

16-bit block hexadecimal form delimited with colons:
83DA:00D3:0000:3F3B:03AA:00FF:FE28:9C58

4.2.4 IoT Data Link Protocol

In this section, different data link protocols are presented in brief including Ethernet and 802.3 as well as different protocol standards used in IoT and their use in IoT systems. The standards in IoT such as 'Bluetooth' and 'ZigBee' are used very frequently. IEEE 802.11ah uses existing infrastructure of IEEE 802.11, which is widely available.

4.2.4.1 Ethernet and IEEE 802.3

The term Ethernet refers to all Carrier SenseMultiple Access/Collision Detection (CSMA/CD) LANs that conform to Ethernet specifications, including IEEE 802.3. IEEE 802.3 specification is based on Ethernet technology, which was introduced in 1980. Ethernet and IEEE 802.3 maintain the largest market currently for any local area network (LAN) protocol. Both Ethernet and IEEE 802.3 LANs are broadcast networks.

CSMA/CD nodes listen to the network to find if it is in use or free, before sending data. If the network is free, the node transmits the data. In case of collision, both transmissions are corrupted/damaged. CSMA/CD nodes can detect collision and they must retransmit the data again after some wait time. 'Exponential-back-off' algorithm is used to determine the retransmission time to avoid collision.

4.2.4.2 ContikiMAC

ContikiMAC is a protocol that proposes enhancement over X-MAC protocol. ContikiMAC is based on asynchronous mechanisms with no signalling messages and no additional packet headers. ContikiMAC is a radio duty cycling protocol that uses periodical wake-ups to listen for packet transmissions from neighbours. When there is no activity the nodes go to sleep mode for duration of a period of duty-cycle. Channel check rate is an important parameter of this protocol that is to be tracked. This parameter defines the frequency with which the nodes will listen to the medium for receiving the data from neighbours.

ContikiMAC packets are ordinary link-layer messages. When the packet is successfully received, the receiver sends a link-layer acknowledgment. For transmitting a data packet, a sender repeatedly sends the packet until it receives a link-layer acknowledgment from the receiver. If a packet transmission is detected during a wake-up, the receiver is kept 'on' to be able to receive the packet. ContikiMAC uses a fast sleep optimization approach to allow receivers to detect quickly false-positive wake-ups and a transmission phase-lock optimization to allow run-time energy-efficiency of transmissions.

4.2.4.3 IEEE 802.15.4

One of the commonly used protocols IEEE 802.15.4 is used as IoT standard for MAC layer. The formats for and source, destination addresses are as per this standard. The frame formats used for traditional networks are not suitable for most of the resource constrained IoT networks. Hence, low power multi-hop networking protocols are needed for IoT.

IEEE802.15.4e was designed and developed in 2008. For IoT communications time synchronization is one of the requirements. This protocol also supports the feature channel hopping and provides high reliability and low cost solutions. IEEE802.15.4e also supports low power communication.

Slot frame structure For sending and receiving information, scheduling of various activities in a node is needed. The frame structure of the standard IEEE 802.15.4e is designed to satisfy this need. In the sleep mode, radio signalling of the node is turned off as this saves the power. During the turned off period, the node stores the messages that are to be sent in the next transmission. After data transmission it waits for an acknowledgment. In the receiving mode, radio signalling is turned on just before the scheduled receiving time to receive the data and to send the acknowledgement for the received data. Again it turns off its radio signalling while delivering the data to the upper network layers and sleep mode is activated.

Node activity scheduling The protocol standard is not meant for scheduling the nodes for their activities, but mobility issues of the node are handled by this protocol. The central manager node handles the activity scheduling and informs other nodes the schedule so that the nodes can follow for effective synchronization.

Synchronization of nodes Connectivity of nodes with the neighbouring nodes in a network is established by synchronization. Mainly two synchronization methods are used namely: 'acknowledgment-based' and 'frame-based'. Acknowledgement-based synchronization is basically used for guaranteed reliable communication and it is also to maintain connectivity. In frame-based synchronization an empty frame is sent at a pre-defined interval typically of 30 seconds.

Frequency channel hopping The frequency channel hopping is done with a pre-determined random sequence. IEEE802.15.4e has an in-built feature of channel hopping to access the wireless medium in time slots. Thus, frequency diversity is introduced which reduces interference effect. It also takes care of multi-path fading.

Network formation Network formation of nodes is based on advertising and joining by the node to the network. A new device node listens for advertisement command and after receiving such a command, the node sends a join request to the advertising node. Then a join request is sent to the manger node for an action in a centralized system approach. In distributed system approach, the node request for joining is processed locally in a distributed fashion. The 'network formation mode' of a node is deactivated when a device node joins the network. The node now becomes functional for that network, and its network formation node is activated again after receiving any other join request.

4.2.4.4 IEEE 802.11 AH (Wi-Fi)

IEEE 802.11 standards are also known as WiFi. The original WiFi standards have drawbacks such as frame overhead and substantial power consumption. Hence, the original Wi-Fi protocol is not suitable for most of the IoT applications. A low energy version of Wireless Medium Access protocol standard 'IEEE 802.11' is known as 'IEEE 802.11ah'. It is for less overhead requirements that are suitable for IoT. This protocol is used widely for all digital device applications such as digital TVs, mobiles, laptops and tablets. 802.11ah task group standards support low overhead and low power consumption communication suitable to effectively manage sensor network.

Main features of 802.11ah MAC layer are as follows:

Frame synchronization Valid medium information is used by the node to collect the information of the medium and the node uses this information to stop packet exchange by other nodes using the shared media for synchronizing the frame exchange. With the duration field packet, it can gain such information.

Bidirectional packet exchange It saves the power consumption of a sensor device as it communicates both ways between the access point and the sensor, using uplink and downlink. It switches to sleep mode after communication to save power.

Short length MAC frame The frame size is reduced in 802.11ah protocol as it is made shorter than 30 bytes to make it suitable for IoT applications. About 12 bytes frame is used in IEEE 802.11ah to remove the constraints due to large frame size length.

Preamble as a very small signal is used to avoid the overhead due to Acknowledgment (ACK) frames. The sleep time is increased in 802.11ah protocol as this protocol is designed for low-power sensors.

4.2.4.5 IEEE 802.16

IEEE 802.16 is a latest series of wireless broadband standards from the Institute of Electrical and Electronics Engineers (IEEE). The commercially recognized name for this standard is WiMax (Worldwide Interoperability for Microwave Access). The 802.16 family of standards is officially called WirelessMA by IEEE.

4.2.5 Application Layer IoT Protocols

Many Application layer protocols are designed to address the constraints of IoT applications. Various Application Layer data transfer IoT protocols such as traditional HTTP and other data transfer protocols such as File Transfer Protocol (FTP) and Message Transfer Protocols, MQTT, XMPP and AMQP are introduced in this section.

Table 4.1 shows the evolution of application layer IoT protocols.

Table 4.1 Evolution of Protocols

Protocols	Year
FTP	1971
UDP	1980
TCP	1983
HTTP	1989
HTTPS	1994
COAP	1997
MQTT	1999
XMPP	1999
AMQP	2003
LORA (LoRaWAN Protocol)	2009

4.2.5.1 HTTP

The application layer protocol Hypertext Text Transfer protocol (HTTP) is for hypermedia information both for distributed as well as collaborative systems. Initially HTTPS was supported by SSL protocol. After further evolution of Secure Sockets Layer (SSL), Transport Layer Security (TLS) is introduced. The current version of HTTPS is available in RFC 2818.

4.2.5.2 File Transfer Protocol

File transfer protocols such as FTP and SFTP (SSH FTP) are suitable for web applications and messaging protocols are a good option for IoT to reduce chunks of data in a message.

4.2.5.3 Messaging Protocols

Many messaging technologies are emerging to support IoT applications. IoT devices in a distributed network are using these next generation IoT technologies. Message Queue Telemetry Transfer (MQTT) and Constrained Application Protocol (CoAP) enable message management with small message sizes (to reduce message overhead).

CoAP

Constrained Application Protocol (CoAP) is an application protocol of a synchronous request/response type. Low-power small devices are using this lightweight protocol that also supports computation and communication capabilities for RESTful communication. This protocol supports the devices with constrained resources with web service functionalities. CoAP is a HTTP-like web transfer protocol, which enables REpresentational State Transfer (REST) architecture to LoWPANs. The binary nature of COAP is useful for IoT applications and supports integration of devices easily. This protocol runs over connectionless UDP. A thin control layer as messaging sublayer that detects duplicate messages is present in CoAP. An optional, simple stop-and-wait mechanism is used for retransmission for reliable delivery of the messages. Though CoAP is a lightweight protocol for IoT applications, it is not equipped with security features. CoAP protocol architecture has two sublayers, namely messaging and request/response. The request/response sublayer is responsible for communication. CoAP has four messaging modes namely: confirmable (reliable communication), non-confirmable (unreliable communication), piggyback, and separate.

MQTT

IBM introduced Message Queue Telemetry Transport (MQTT) in 1999. In this protocol middleware and application layer connectivity is provided. MQTT was released by IBM for lightweight machine-to-machine communications. The MQTT protocol is a messaging protocol ideal for IoT and Mobile-to-Mobile (M2M) applications and wireless sensor network. In IoT applications, where sensor node and processing node communication is required, MQTT is a suitable lightweight protocol. It is an asynchronous publish/subscribe protocol which runs on TCP protocol stack. MQTT is used to connect embedded devices such as IoT boards and networks.

For transition flexibility, MQTT utilizes publish/subscribe pattern. It is a suitable protocol for networks with low bandwidth for routing low-power and low-memory devices which are vulnerable

to different attacks. In MQTT protocol architecture, there is a broker that works as a server for the topics and a subscriber that receives messages automatically when there is some update on the topic for which it has subscribed. The subscriber receives the update messages about the topic for which it has subscribed. MQTT protocol was designed for remote telemetry applications.

MQTT enables the transfer of data which is originally in the telemetry style to the form of messages to a message broker or server from devices with constrained networks and/or high latency. Different kinds of devices such as sensors, mobile phones, embedded systems, and actuators, are used for M2M communication using MQTT. One of such example is sensors on patient collecting his physical parameters by monitoring equipment.

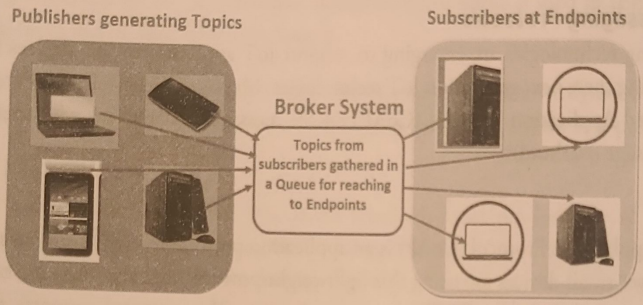


Fig. 4.2 MQTT Architecture

Figure 4.2 shows the MQTT architecture with three main components: publishers, subscribers, and a broker [25]. In IoT applications for low power consumption, the publishers are typically lightweight sensor nodes that connect to the broker to send the data to the subscriber. After data transmission is over it switches to sleep mode to save battery power. Sensor data is subscribed by some applications through the brokers, so that subscribers will receive the information when new data of interest arrives. Quality of service levels offered by MQTT are Level 0: At most one (best effort no acknowledgment), Level 1: At least one (acknowledged and retransmitted if acknowledgment is not received), Level 2: Exactly once (request to send (publish), clear to send (pubrec), message (reply to publish), ack (pubcomp).

SMQTT

Secure MQTT (SMQTT) is an extension of MQTT which is based on lightweight attribute-based encryption technique. The major advantage of this encryption scheme is that, after encrypting one message it can be delivered to multiple nodes. This scenario is very typical in IoT applications. This protocol works in four stages namely: setup, encryption, publish, and decryption. The data to be published is encrypted. This encrypted data is then published by the broker and sent to the subscribers. SMQTT provides enhanced security features as compared to MQTT.

XMPP

Extensible Messaging and Presence Protocol (XMPP) is a messaging protocol that was designed originally for messaging and chatting applications. It is a well-proven protocol, standardized by IETF and used widely all over the Internet. In contrast to CoAP's request/response approach, XMPP supports publish/subscribe as well as request/response architecture which is a favourable feature for IoT applications. It runs over a variety of Internet-based platforms in a decentralized fashion.

Quality of service is not ensured by this protocol, thus it is not suitable for M2M communication. Header and tag formats of XML messages put overhead which is a negative feature that increases power consumption in critical IoT applications. Hence, XMPP is not preferred in IoT applications.

AMQP

Advanced Message Queuing Protocol (AMQP) is a protocol that is designed and developed for applications of financial industry. AMQP uses a reliable transport protocol such as TCP to exchange messages. AMQP provides asynchronous publish/subscribe communication with messaging. Reliable communication is achieved by this protocol by message delivery guarantee primitives namely: at-most-once, at-least-once, and exactly-once delivery. AMQP implementations are interoperable with each other by defining a wire-level protocol.

It runs over TCP and has similar architecture as that of MQTT providing a publish/subscribe architecture. However in this protocol, broker works as an exchange or queues unlike in MQTT as shown in Fig. 4.3.

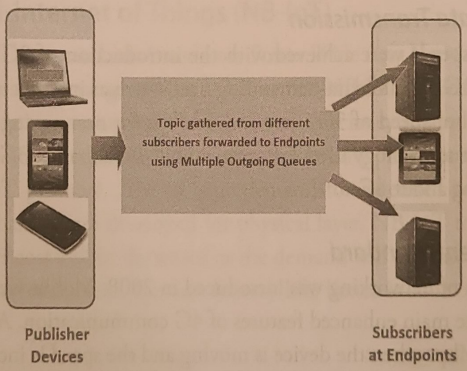


Fig. 4.3 AMQP Architecture

Queues represent topics subscribed by subscribers when messages are available in the queue.

4.2.5.4 DDS

Data Distribution Service (DDS) is also a publish/subscribe protocol. 23 types of quality service levels are offered by this protocol based on the criteria such as priority, security, urgency, reliability, and durability. DDS protocol structure has two sublayers namely data-centric publish-subscribe and data-local reconstruction. Sensor data distribution is carried out by Publisher layer. Subscribers receive the sensor data and supply it to the IoT application as per its requirements. Guaranteed delivery of the message is the most important benefit of this protocol as it follows a broker-less architecture, which is suitable for IoT.

4.2.6 Mobile Communication

Continuous evolution from 1G to 5G is happening in the last three decades in mobile communication. First generation of wireless cellular technology is called as 1G. The second generation technology is named as 2G and so on. 1G was introduced in 1980s. New features are added in the next generations as they are faster than their predecessors. New generation of wireless mobile technology is released after every 10 years. Most wireless carriers support 3G and 4G. Next generation 5G is released in 2020.

4.2.6.1 1G: Voice Only

In 1980s, the 1G technology is introduced for the cell phones. The maximum speed of 1G technology is 2.4 Kbps and this technology supports voice only calls. 1G also supports analog technology with high power consumption.

4.2.6.2 2G: SMS and MMS

Cell phones technology is evolved from 1G to 2G in 1991. GSM networks switched cell phones from analog to digital communications. The maximum speed is increased from 2G speed of 50 Kbps of General Packet Radio Service (GPRS) is to the speed is 1 Mbps with Enhanced Data Rates (EDGE) with the evolution of GSM. The 2G telephone technology has security features such as call and text encryption and data services such as SMS, MMS and picture messages.

4.2.6.3 3G: Faster Data Transmission

Faster data transmission speeds were achieved with the introduction of 3G networks in 1998. High speed was obtained with 3G to fulfil data-demanding needs such as internet access through mobile and video calling. The maximum speed of 3G is around 2 Mbps for nonmoving devices and 384 Kbps in moving mode. 3G cellular technology used the term 'mobile broadband'. 3G also evolved further with additional features to bring about 4G.

4.2.4.4 4G: The Present Standard

4G, the fourth generation of networking was introduced in 2008. Mobile web access and higher speed than the 3G service are the main enhanced features of 4G communication. A 4G network can provide maximum speed of 100 Mbps when the device is moving and the speed is increased to 1 Gbps for low-mobility communication such as when the user is stationary or walking. Video conferencing, gaming services, HD mobile TV, 3D TV are the added features with 4G. Latest smart phone models support both 4G and 3G technologies.

Fifth generation (5G) will be the future wireless technology after 4G, to improve data rates significantly, much lower latency, low energy consumption and higher connection density. The maximum speed of 5G connections is 20 Gbps.

BOX 4.1: COMPARISON OF MESSAGING PROTOCOLS

Table 4.2 summarizes three messaging protocols, MQTT, COAP, and XMPP with respect to mode, transport, and security. Cryptographic protocols such as Transport Layer Security (TLS) and Secure Sockets Layer (SSL) work on the principle of handshaking and use TCP as transport protocol. To encrypt the whole MQTT communication, TLS is used instead of plain TCP. Datagram Transport Layer Security (DTLS) as a security protocol for communication uses CoAP for protecting confidential information. DTLS is also used for authentication of communicating devices.

Table 4.2 Messaging Protocols

	MQTT	COAP	XMPP
Mode	Publish/Subscribe	Client/Server	Client/Server
Transport	TCP or UDP	UDP	TCP or HTTP
Security	SSL/TLS	DTLS	SSL/TLS

4.3 LOW-POWER WIDE AREA NETWORK (LPWAN)

A low-power wide-area network (LPWAN) is designed for low baud rate and long-range communications between connected objects. The objects/things connected to LPWAN such as sensor and actuators are generally powered by battery at remote places, where power supply by traditional way is not possible.

4.3.1 NarrowBand-Internet of Things (NB-IoT)

NarrowBand-IoT (NB-IoT) is a standards-based Low-Power Wide Area Network (LPWAN) technology. This protocol is useful for a wide range of new IoT systems. NB-IoT attains low power consumption of devices and also enhances system capacity and spectrum efficiency in applications which need deep coverage systems. Battery life is improved by 10 years due to low power consumption feature for the applications using NB-IoT protocol. Due to large demand for extended coverage and for ultra-low device complexity, new channels are developed for physical layer. NB-IoT technology is simpler than GSM/GPRS and hence its cost will be decreased as the demand will increase.

Goals of NB-IoT are to reduce cost, increase battery life, better indoor coverage and provide high density connections.

4.3.2 LoRa — LoRaWAN Protocol

Long Range (LoRa) is a low-power, low-bitrate, long-range, and wireless communication system, for the applications of IoT. This system is useful where long battery life and low powered devices are needed. This protocol aims for low energy consumption. For smart sensing technology applications such as environment monitoring and smart metering, LoRa is the best option due to its long range and low-power consumption nature and it is also useful for industrial application. LoRa is a specification based on Chirp Spread Spectrum (CSS) for the physical layer. It is integrated with Forward Error Correction (FEC). Other useful properties of LoRa are high robustness, multipath resistance and Doppler resistance. Also, very long distance communication is achievable with LoRa infrastructure.

LoRa radio in a sensor network having following benefits:

- Relatively large range (maximum 10 km), hence LoRa enabled networks can span large areas with less number of hops. Node to the sink one hop solution is possible in many cases.
- Different spreading factor orthogonal to each other, are used for transmission on the same carrier frequency. Based on this channel division, virtual sub channels are created.
- When multiple transmissions occur at the same time with the same parameters then the high probable transmission is considered as strongest.

This wireless technology is used nowadays in smart home appliances, cars, street lights, and manufacturing equipment.

4.4 WIRELESS TECHNOLOGIES SUPPORTING IOT APPLICATIONS

In IoT applications multiple objects/things are connected in distributed locations. Hence some data link protocols are designed to provide connectivity with low cost and in energy efficient manner. Few protocols such as WirelessHART, Z-Wave, Bluetooth and ZigBee, DASH7 and LTE-A are introduced in this section.

4.4.1 WirelessHART

Wireless Highway Addressable Remote Transfer Protocol (WirelessHART) is a data link protocol which operates in the physical layer. Time Division Multiple Access (TDMA) is used in MAC layer to obtain a secure and reliable MAC protocol. TDMA protocol provides advanced encryption features for data security and integrity while data in transit. It enables different security mechanisms such as per hop, end to end and peer to peer. End to end security mechanisms ensures security from source to destination. In per-hop mechanisms data security is guaranteed only till next hop location.

The WirelessHART architecture is shown in Fig. 4.4. Its different modules such as network manager and gateway connect the wireless network to the wired networks.

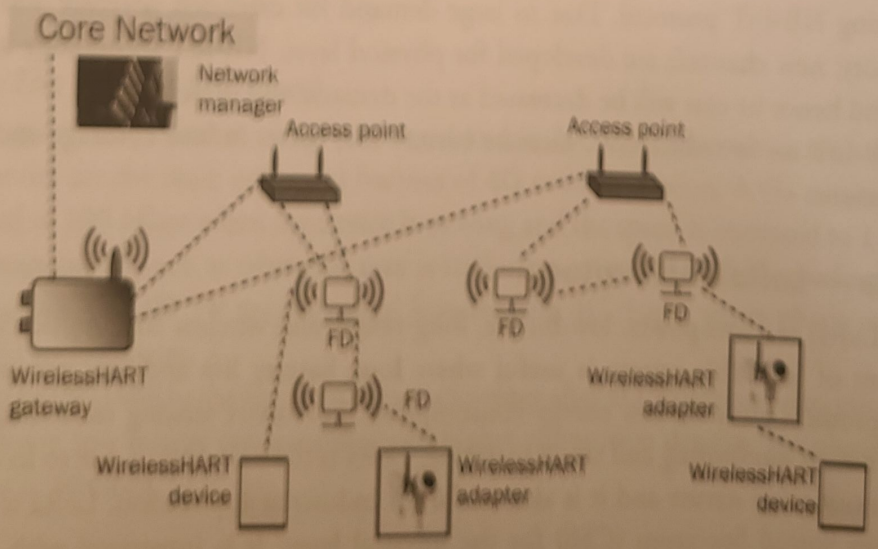


Fig. 4.4 WirelessHART Architecture

4.4.2 Z-Wave

Z-Wave is a MAC layer protocol with low-power consumption with its applications predominantly in home automation. It is frequently used in many application domains as well as in small commercial applications. Master/slave architecture used by Z-Wave where by handling scheduling of the whole network, the master controls the slaves by sending commands. CSMA/CA technology is used for collision detection and ACK messages guarantee reliable transmission. Small messages in IoT applications are efficiently managed by this technology and used in applications such as wearable devices in healthcare control, energy control, etc. Point-to-point communication in a range of 30 m is achievable by this protocol.

4.4.3 Bluetooth Low Energy

Bluetooth low energy also known as Bluetooth smart is a short range MAC layer communication protocol. It is also used in Physical layer. Its major application area is for in-vehicle networking. Contentionless MAC is used by access control with low latency and fast speed transmission. It takes 10 times less time for data communication and improved latency by 15 times. The master/slave architecture follows architecture of two types of frames namely adverting and data frames. Advertising frame is sent by slaves on dedicated channels to discover master node. Master nodes sense these advertisement channels to locate the slave nodes and connect to those nodes. After connection, the wake up cycle schedule is informed to the slave node by the master. For low power consumption, when the nodes are communicating, only for that duration they are in 'awake mode', otherwise they switch to 'sleep mode'.

4.4.4 ZigBee Smart Energy

IoT applications such as smart home, healthcare systems and remote controls use ZigBee smart energy protocol. ZigBee standard has two stack profile: ZigBee and ZigBee Pro. This standard's two stack structure supports full mesh networking for the applications with low memory and low processing power constraints. It also supports many other network topologies such as cluster-tree, star, peer-to-peer etc. The central node of the star topology, works as a coordinator and controls the network. ZigBee Pro stack profile offers features such as symmetric-key exchange based security, scalability and performs better with many-to-one routing mechanisms.

4.4.5 DASH7

DASH7 is a wireless communication protocol suitable for IoT applications which works on active RFID that uses Industrial Scientific Medical (ISM) band. It follows master/slave architecture. It is also useful for applications having asynchronous and transitive traffic and need lightweight technology. The low-cost solution of this protocol supports encryption as well as IPv6 addressing. This protocol is useful to make the application scalable which need a higher data rate as compared to the ZigBee protocol and long range coverage.

Its MAC layer features are as below:

- Filtering of incoming frames
- Validation with Cyclic redundancy check (CRC)
- Link quality assessment using 4-bit subnet mask

The frames are processed further only after passing all the above three checks.

4.4.6 LTE-A

Long-Term Evolution Advanced (LTE-A) is a set of standards developed for M2M and IoT applications. The MAC layer access technology used by LTE-A is OFDMA (Orthogonal Frequency Division Multiple Access) that uses multiple bands for frequency division and each band is then used separately.

Figure 4.5 shows LTE-A architecture which consists of Core network (CN) that controls mobile devices, Radio access network (RAN) for managing and controlling wireless connectivity through radio-access control and mobile nodes. LTE-A protocol is a low cost, scalable protocol. NodeB is a telecommunication node. eNodeB (evolved NodeB) is an element of the LTE network. Two eNodeBs are connected using X2 interface. X2 interface supports control plane and user plane.

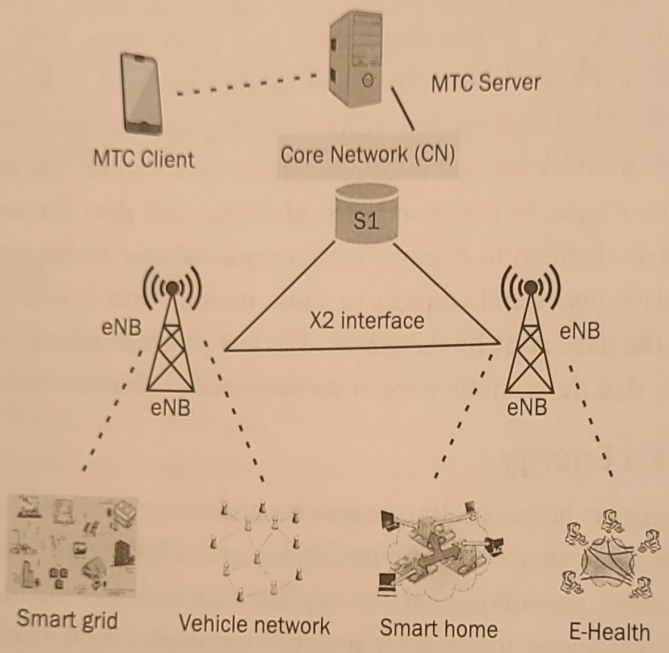


Fig. 4.5 LTE-A Architecture

4.5 NETWORK LAYER ENCAPSULATION PROTOCOLS

IPv6 addresses are very long and are not suitable for most of the IoT data link frames which need smaller frames. IETF is developing a set of standards suitable for data link layer, especially for IoT applications. These standards are briefly introduced below.

4.5.1 6LoWPAN

In Low power wireless Personal Area Network(6LoWPAN), the IPv6 long header is broken efficiently to small packets to avoid long headers of more than 128 byte size. This protocol supports variable length addresses and different topologies such as star and mesh and multi-hop delivery. Applications that requiring low power consumption, low cost, scalable networks and are benefitted by it. Further, transmission overheads are reduced by header compression.

Different types of frames in 6LoWPAN and their use:

- Unless the frame adheres to the 6LoWPAN specification, frame forwarding is not allowed.
- IPv6 header header compression of dispatch header is used for multicast messages.
- Due to frame length limit of 128-bytes of this protocol, frame fragmentation mechanism which is available in IEEE802.15.4 is used for this purpose.

4.5.2 6TiSCH

This standard is developed by IETF working group for IPv6. It integrates various other standards such as IEEE802.15.4 TSCH, 6LoWPAN, RPL and CoAP. The main focus area of 6TiSCH (Time Slotted Channel Hopping standardised in IEEE 802.15.4e) is industrial Internet of Things (IIoT) where it has proven reliability of 99.99% similar to wired technologies. TiSCH is adopted by IEEE as MAC technique which is integrated with IEEE 802.15.4. TiSCH works on the concept of timeslots by partitioning time into slots (10 ms in general) which are used by nodes to communicate (maximum size of 127 B frame) while synchronising with the network. A slotframe is used to group a number of timeslots. A node knows to transmit, whom to transmit, receive or sleep in a timeslot in a slotframe based on a pre-defined schedule.

THOUGHT EXERCISES

- Design the IPv6 class encapsulation mask for a wireless personal area network.
- Design an IPv6 address in binary and 16-bit block hexadecimal and 16-bit block hexadecimal

SUMMARY

The quality of the life is envisaged to improve by fast emerging Internet of Things (IoT), by connecting smart devices, using IoT technologies. The 'Internet' is a backbone of the "Internet of Things." is a focus of this chapter. An overview of the Open Systems Interconnection (OSI) model is presented at the beginning. Followed by the description of TCP/IP model, the basic model for the Internet. This chapter also explains and compares IP version 4 with IP version 6. It showed the limitation of IPv4, especially in the scenario when 20 billion devices for IoT are expected. IPv4 has room for about 4.3 billion addresses, whereas IPv6, with a 128-bit address, has huge number of addresses (340 trillion trillion trillion).

The standards are categorized based on the network layer of operation and the network model for example data link layer, network routing standards, network encapsulation layer. For each layer, some recommended standards and some draft standards are presented. The fast evolving IoT technologies would allow for the automation of almost everything around us. Comparison of data transfer protocol for instant messaging is presented with respect to mode of transport medium and required security.

Network encapsulation protocols IPv6 in IoT MAC frame is explored in this chapter. Many session layer protocols for IoT applications are discussed. MQTT is most frequently used in IoT due to its lightweight nature in low power consumption and low transmission overheads. Other protocols such as XMPP, CoAP, RPL, CORPL, 6LoWPAN are also discussed.

KEY TERMS

IPv4, IPv6, IoT network layer protocols, IoT application layer protocols, LoRA, LoWPAN

FURTHER READING

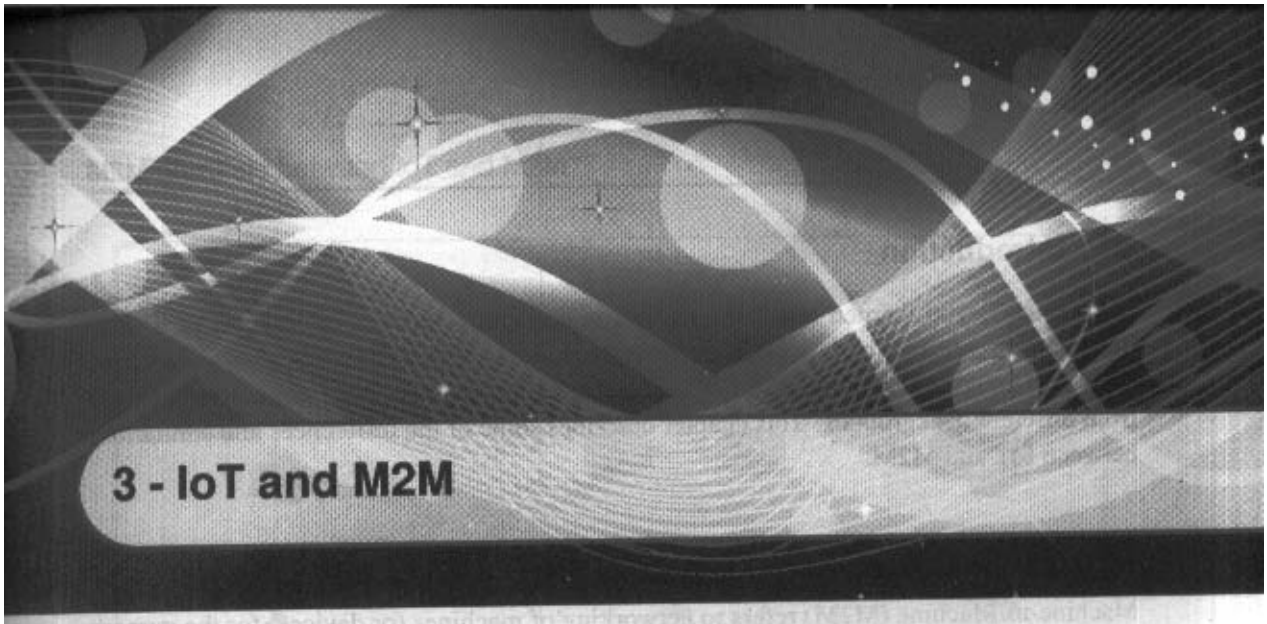
1. http://tools.ietf.org/html/rfc768
2. http://tools.ietf.org/html/rfc4347
3. https://datatracker.ietf.org/wg/roll/RPL
4. http://www.ccnx.org/releases/latest/doc/technical/CCNxProtocol.html
5. http://mqtt.org/
6. http://portals.omg.org/dds/
7. http://openkontrol.org/llap/index.php
8. http://www.bluetooth.com/Pages/Bluetooth-Home.aspx
9. http://www.zigbee.org/
10. https://www.ericsson.com/news/1987789
11. https://www.lora-alliance.org/What-Is-LoRa/Technology
12. http://postscapes2.webhook.org/internet-of-things-technologies
13. http://raml.org/
14. https://tools.ietf.org/html/draft-jennings-senml--senml-08
15. http://www.ipso-alliance.org/wp-content/media/draft-ipso-appframework-04.pdf

REVIEW EXERCISES

1. Why does the Internet require both TCP and IP?
2. How many total IPv4 addresses are available? Explain.
3. Find the ratio of the number of addresses in IPv6 to IPv4. Explain its usefulness in the evolutionary expansion of IoT based applications.
4. IPv6 uses a 128-bit address. Explain, how many IPv6 addresses exist?
5. Are IPv4 and IPv6 protocols interoperable? Which constraints will be faced by an enterprise in transition from IPv4 to IPv6?
6. How many IPv6 address will be available on each square meter of earth?
7. What is a low-power/lossy network? How does that relate to IoT?
8. What is RPL and how does it work (high level)?
9. Explain the application of Z-Wave for home automation.
10. What is the advantage of Bluetooth smart energy over the Bluetooth technology?

REFERENCES

1. Kaukalias, T. Chatzimisios, P. (2014), Internet of Things (IoT) enabling technologies, applications and open issues, Encyclopedia of Information Science and Technology, 3rd Ed., IGI Global Press, USA, 134–136.
2. Colitti, W., Steenhaut, K., De Caro, N., Buta, B., Dobrota, V. (2011). Evaluation of constrained application protocol for wireless sensor networks in Local Metropolitan Area Networks (LANMAN). 18th IEEE Workshop, January, pp. 1–6.
3. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T. (1999), Hypertext Transfer Protocol HTTP, IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), June, pp. 45–46.
4. Sye, K., Kumar, S. S., Hannes, T. (2014), Securing the Internet of Things: A standardization perspective. IEEE Internet of Things Journal, 1(3): 265–275.



3 - IoT and M2M

This Chapter Covers

- M2M
- Differences and Similarities between M2M and IoT
- SDN and NFV for IoT

3.1 Introduction

In Chapter-1, you learned about the definition and characteristics of Internet of Things (IoT). Another term which is often used synonymously with IoT is Machine-to-Machine (M2M). Though IoT and M2M are often used interchangeably, these terms have evolved from different backgrounds. This chapter describes some of the differences and similarities between IoT and M2M.

3.2 M2M

Machine-to-Machine (M2M) refers to networking of machines (or devices) for the purpose of remote monitoring and control and data exchange. Figure 3.1 shows the end-to-end architecture for M2M systems comprising of M2M area networks, communication network and application domain. An M2M area network comprises of machines (or M2M nodes) which have embedded hardware modules for sensing, actuation and communication. Various communication protocols can be used for M2M local area networks such as ZigBee, Bluetooh, ModBus, M-Bus, Wirless M-Bus, Power Line Communication (PLC), 6LoWPAN, IEEE 802.15.4, etc. These communication protocols provide connectivity between M2M nodes within an M2M area network. The communication network provides connectivity to remote M2M area networks. The communication network can use either wired or wireless networks (IP-based). While the M2M area networks use either proprietary or non-IP based communication protocols, the communication network uses IP-based networks. Since non-IP based protocols are used within M2M area networks, the M2M nodes within one network cannot communicate with nodes in an external network. To enable the communication between remote M2M area networks, M2M gateways are used.

Figure 3.2 shows a block diagram of an M2M gateway. The communication between the M2M nodes and the M2M gateway is based on the communication protocols which are native to the M2M area network. M2M gateway performs protocol translations to enable IP-connectivity for M2M area networks. M2M gateway acts as a proxy performing translations from/to native protocols to/from Internet Protocol (IP). With an M2M gateway, each node in an M2M area network appears as a virtualized node for external M2M area networks.

The M2M data is gathered into point solutions such as enterprise applications, service management applications, or remote monitoring applications. M2M has various application domains such as smart metering, home automation, industrial automation, smart grids, etc. M2M solution designs (such as data collection and storage architectures and applications) are specific to the M2M application domain.

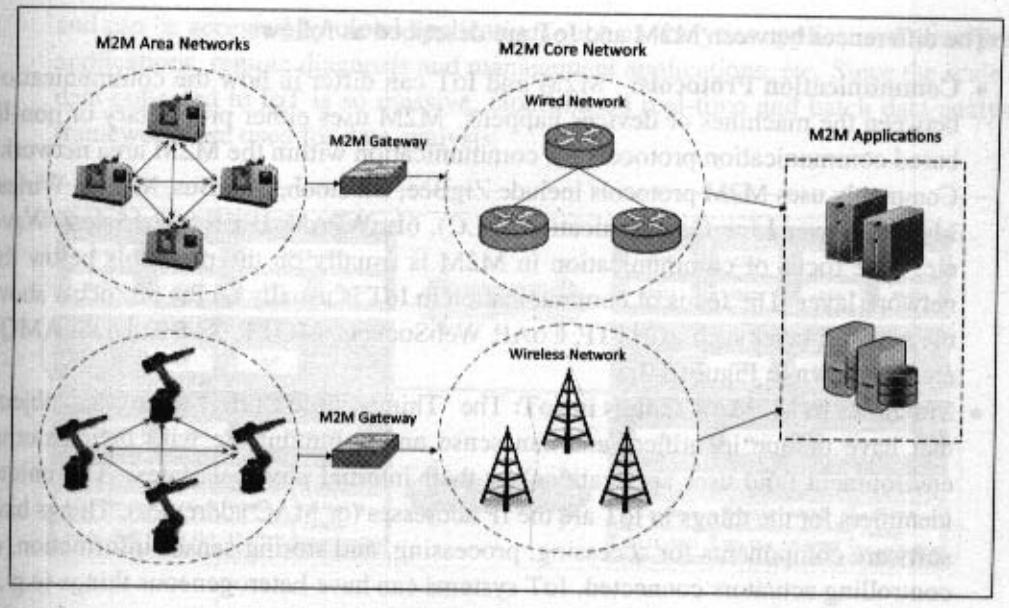


Figure 3.1: M2M system architecture

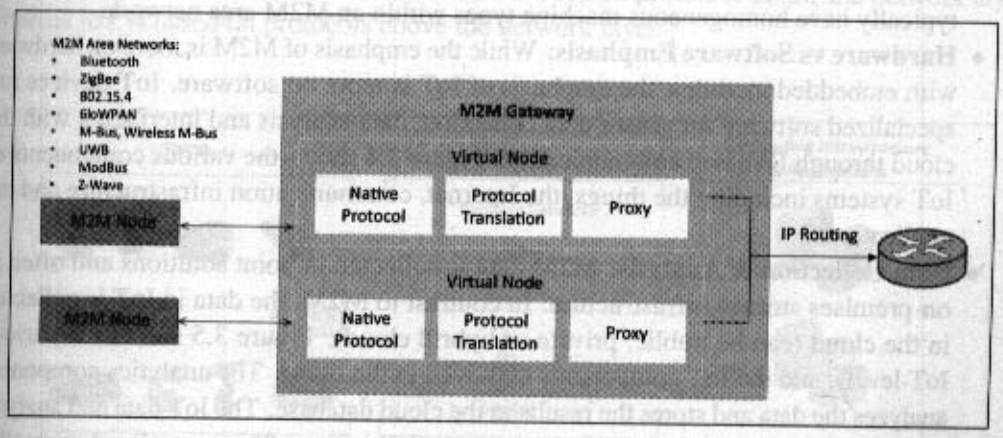


Figure 3.2: Block diagram of an M2M gateway

3.3 Difference between IoT and M2M

Though both M2M and IoT involve networking of machines or devices, they differ in the underlying technologies, systems architectures and types of applications.

The differences between M2M and IoT are described as follows:

- **Communication Protocols:** M2M and IoT can differ in how the communication between the machines or devices happens. M2M uses either proprietary or non-IP based communication protocols for communication within the M2M area networks. Commonly uses M2M protocols include ZigBee, Bluetooh, ModBus, M-Bus, Wirless M-Bus, Power Line Communication (PLC), 6LoWPAN, IEEE 802.15.4, Z-Wave, etc. The focus of communication in M2M is usually on the protocols below the network layer. The focus of communication in IoT is usually on the protocols above the network layer such as HTTP, CoAP, WebSockets, MQTT, XMPP, DDS, AMQP, etc., as shown in Figure 3.3.
- **Machines in M2M vs Things in IoT:** The "Things" in IoT refers to physical objects that have unique identifiers and can sense and communicate with their external environment (and user applications) or their internal physical states. The unique identifiers for the things in IoT are the IP addresses (or MAC addresses). Things have software components for accessing, processing, and storing sensor information, or controlling actuators connected. IoT systems can have heterogeneous things (e.g., a home automation IoT system can include IoT devices of various types, such as fire alarms, door alarms, lighting control devices, etc.) M2M systems, in contrast to IoT, typically have homogeneous machine types within an M2M area network.
- **Hardware vs Software Emphasis:** While the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software. IoT devices run specialized software for sensor data collection, data analysis and interfacing with the cloud through IP-based communication. Figure 3.4 shows the various components of IoT systems including the things, the Internet, communication infrastructure and the applications.
- **Data Collection & Analysis:** M2M data is collected in point solutions and often in on-premises storage infrastructure. In contrast to M2M, the data in IoT is collected in the cloud (can be public, private or hybrid cloud). Figure 3.5 shows the various IoT-levels, and the IoT components deployed in the cloud. The analytics component analyzes the data and stores the results in the cloud database. The IoT data and analysis results are visualized with the cloud-based applications. The centralized controller is aware of the status of all the end nodes and sends control commands to the nodes. Observer nodes can process information and use it for various applications, however, observer nodes do not perform any control functions.
- **Applications:** M2M data is collected in point solutions and can be accessed by on-premises applications such as diagnosis applications, service management applications, and on-premisis enterprise applications. IoT data is collected in the cloud

and can be accessed by cloud applications such as analytics applications, enterprise applications, remote diagnosis and management applications, etc. Since the scale of data collected in IoT is so massive, cloud-based real-time and batch data analysis frameworks are used for data analysis.

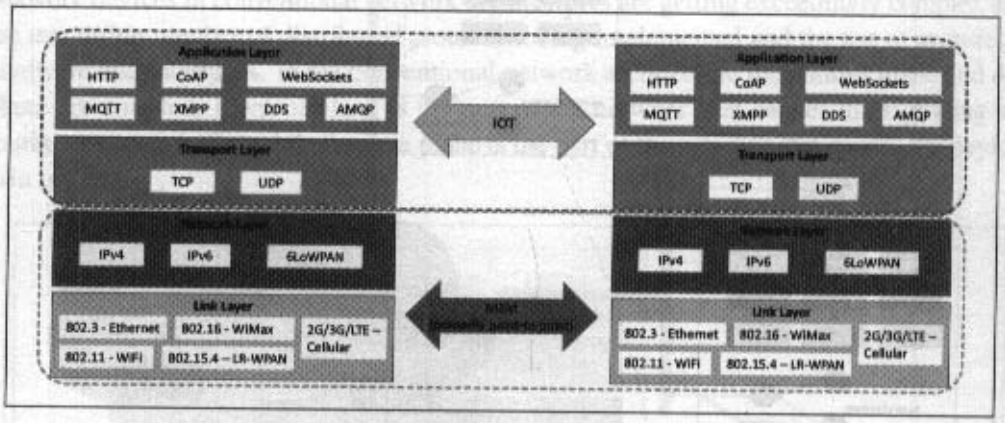


Figure 3.3: Communication in IoT is IP-based whereas M2M uses non-IP based networks. Communication within M2M area networks is based on protocols below the network layer whereas IoT is based on protocols above the network layer.

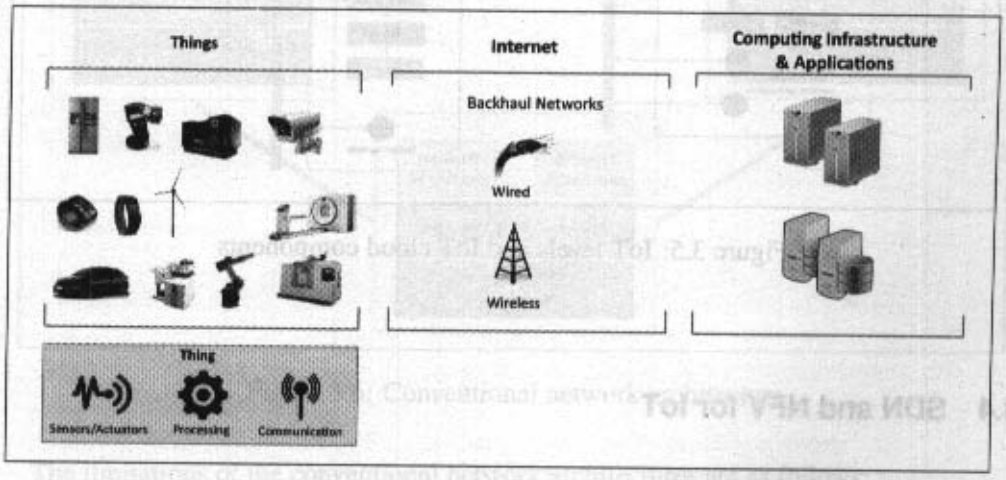


Figure 3.4: IoT components

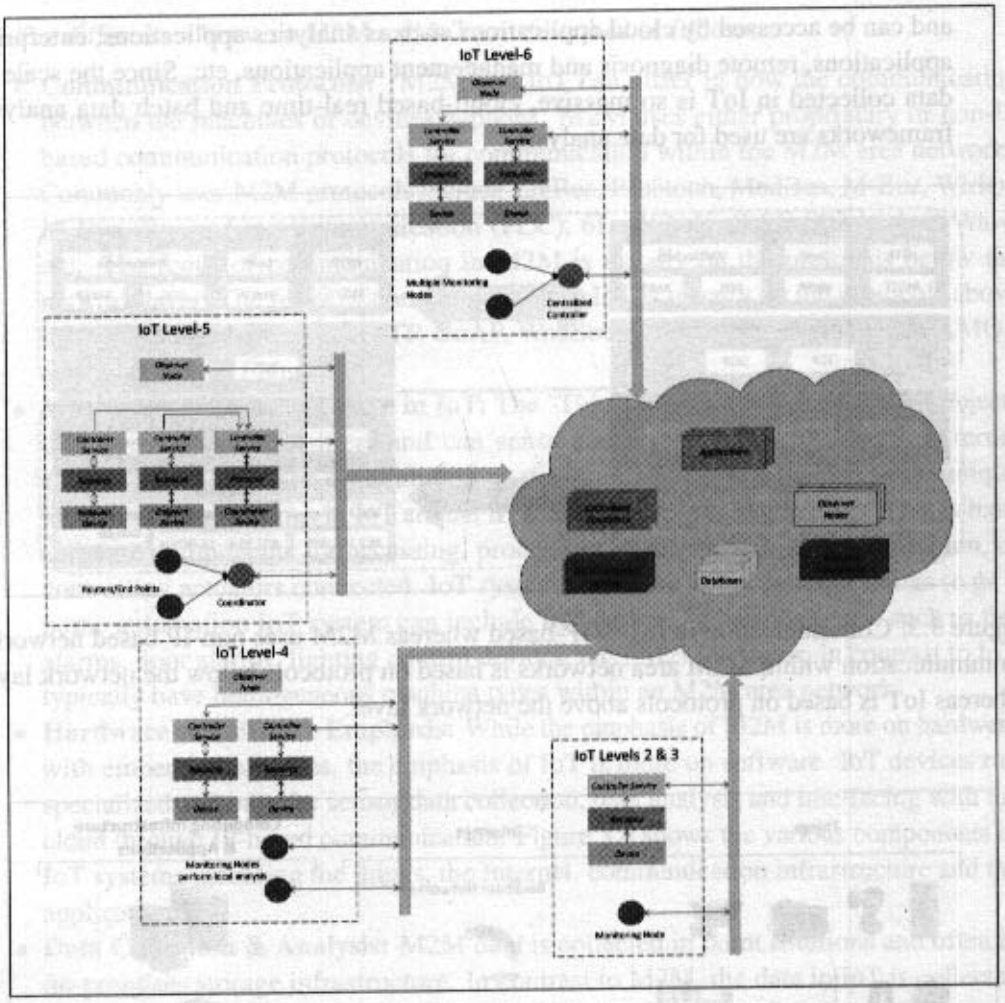


Figure 3.5: IoT levels and IoT cloud components

3.4 SDN and NFV for IoT

In this section you will learn about Software Defined Networking (SDN) and Network Function Virtualization (NFV) and their applications for IoT.

3.4.1 Software Defined Networking

Software-Defined Networking (SDN) is a networking architecture that separates the control plane from the data plane and centralizes the network controller. Figure 3.6 shows the conventional network architecture built with specialized hardware (switches, routers, etc.). Network devices in conventional network architectures are getting exceedingly complex with the increasing number of distributed protocols being implemented and the use of proprietary hardware and interfaces. In the conventional network architecture the control plane and data plane are coupled. Control plane is the part of the network that carries the signaling and routing message traffic while the data plane is the part of the network that carries the payload data traffic.

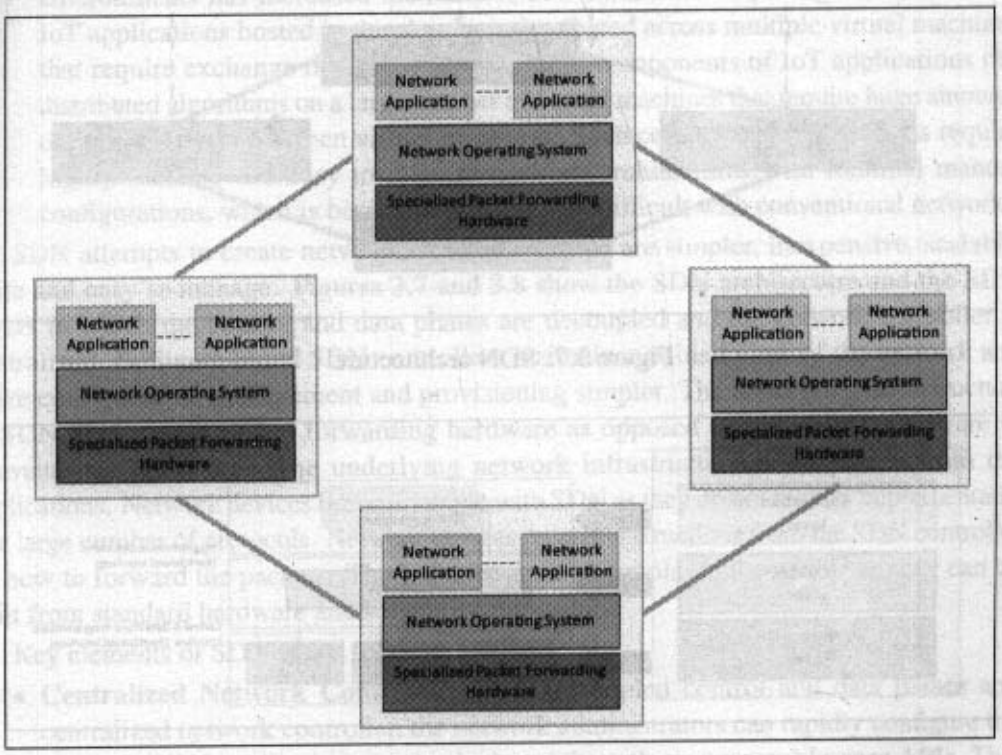


Figure 3.6: Conventional network architecture

The limitations of the conventional network architectures are as follows:

- **Complex Network Devices:** Conventional networks are getting increasingly complex with more and more protocols being implemented to improve link speeds and reliability.

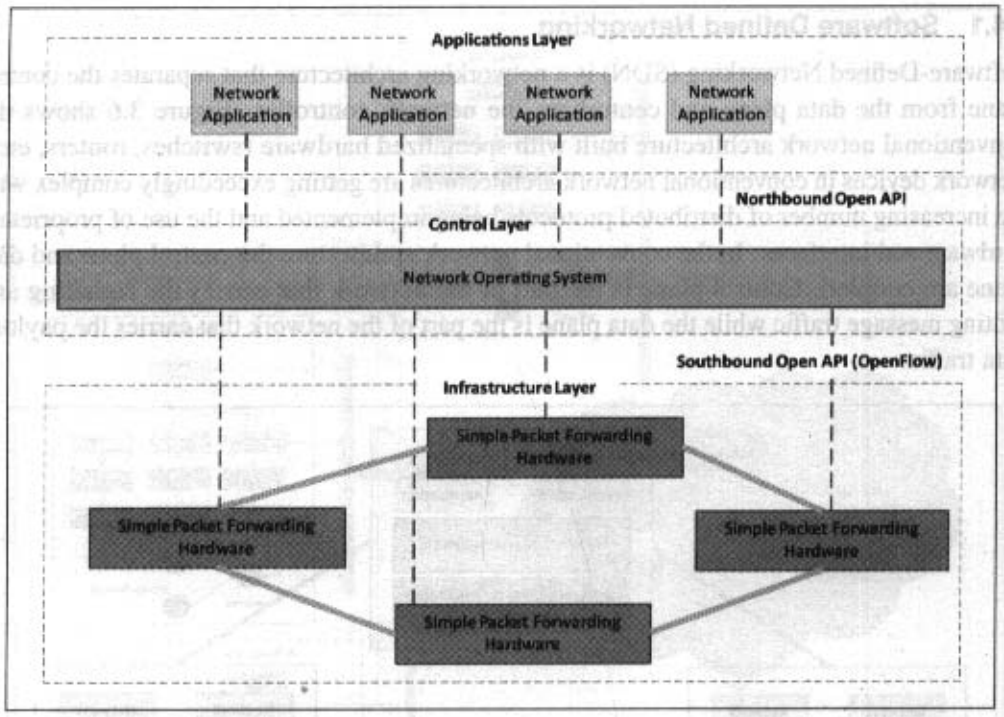


Figure 3.7: SDN architecture

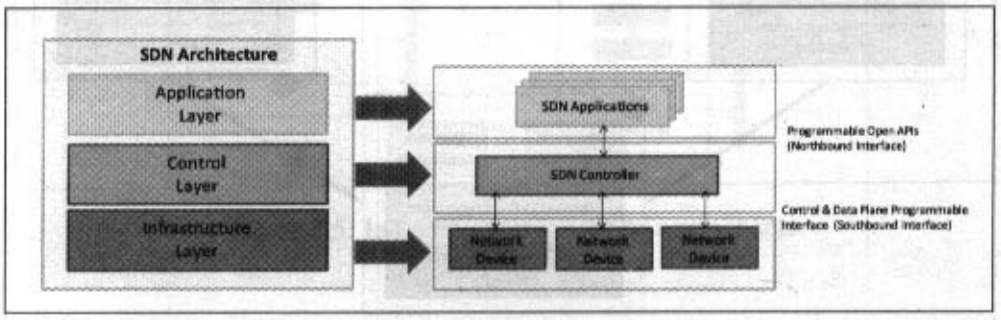


Figure 3.8: SDN layers

Interoperability is limited due to the lack of standard and open interfaces. Network devices use proprietary hardware and software and have slow product life-cycles limiting innovation. The conventional networks were well suited for static traffic

patterns and had a large number of protocols designed for specific applications. For IoT applications which are deployed in cloud computing environments, the traffic patterns are more dynamic. Due to the complexity of conventional network devices, making changes in the networks to meet the dynamic traffic patterns has become increasingly difficult.

- **Management Overhead**: Conventional networks involve significant management overhead. Network managers find it increasingly difficult to manage multiple network devices and interfaces from multiple vendors. Upgradation of network requires configuration changes in multiple devices (switches, routers, firewalls, etc.)
- **Limited Scalability**: The virtualization technologies used in cloud computing environments has increased the number of virtual hosts requiring network access. IoT applications hosted in the cloud are distributed across multiple virtual machines that require exchange of traffic. The analytics components of IoT applications run distributed algorithms on a large number of virtual machines that require huge amounts of data exchange between virtual machines. Such computing environments require highly scalable and easy to manage network architectures with minimal manual configurations, which is becoming increasingly difficult with conventional networks.

SDN attempts to create network architectures that are simpler, inexpensive, scalable, agile and easy to manage. Figures 3.7 and 3.8 show the SDN architecture and the SDN layers in which the control and data planes are decoupled and the network controller is centralized. Software-based SDN controllers maintain a unified view of the network and make configuration, management and provisioning simpler. The underlying infrastructure in SDN uses simple packet forwarding hardware as opposed to specialized hardware in conventional networks. The underlying network infrastructure is abstracted from the applications. Network devices become simple with SDN as they do not require implementations of a large number of protocols. Network devices receive instructions from the SDN controller on how to forward the packets. These devices can be simpler and cost less as they can be built from standard hardware and software components.

Key elements of SDN are as follows:

- **Centralized Network Controller**: With decoupled control and data planes and centralized network controller, the network administrators can rapidly configure the network. SDN applications can be deployed through programmable open APIs. This speeds up innovation as the network administrators no longer need to wait for the device vendors to embed new features in their proprietary hardware.
- **Programmable Open APIs**: SDN architecture supports programmable open APIs for interface between the SDN application and control layers (Northbound interface). With these open APIs various network services can be implemented, such as routing,

quality of service (QoS), access control, etc.

- **Standard Communication Interface (OpenFlow)**: SDN architecture uses a standard communication interface between the control and infrastructure layers (Southbound interface). OpenFlow, which is defined by the Open Networking Foundation (ONF) is the broadly accepted SDN protocol for the Southbound interface. With OpenFlow, the forwarding plane of the network devices can be directly accessed and manipulated. OpenFlow uses the concept of flows to identify network traffic based on pre-defined match rules. Flows can be programmed statically or dynamically by the SDN control software. Figure 3.9 shows the components of an OpenFlow switch comprising of one or more flow tables and a group table, which perform packet lookups and forwarding, and OpenFlow channel to an external controller. OpenFlow protocol is implemented on both sides of the interface between the controller and the network devices. The controller manages the switch via the OpenFlow switch protocol. The controller can add, update, and delete flow entries in flow tables. Figure 3.10 shows an example of an OpenFlow flow table. Each flow table contains a set of flow entries. Each flow entry consists of match fields, counters, and a set of instructions to apply to matching packets. Matching starts at the first flow table and may continue to additional flow tables of the pipeline [83].

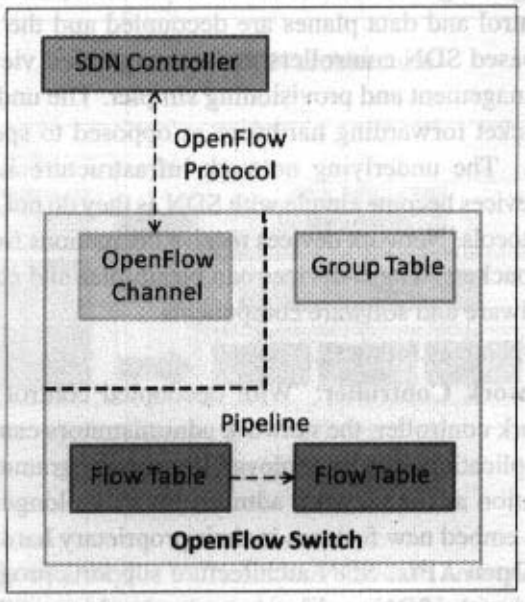


Figure 3.9: OpenFlow switch

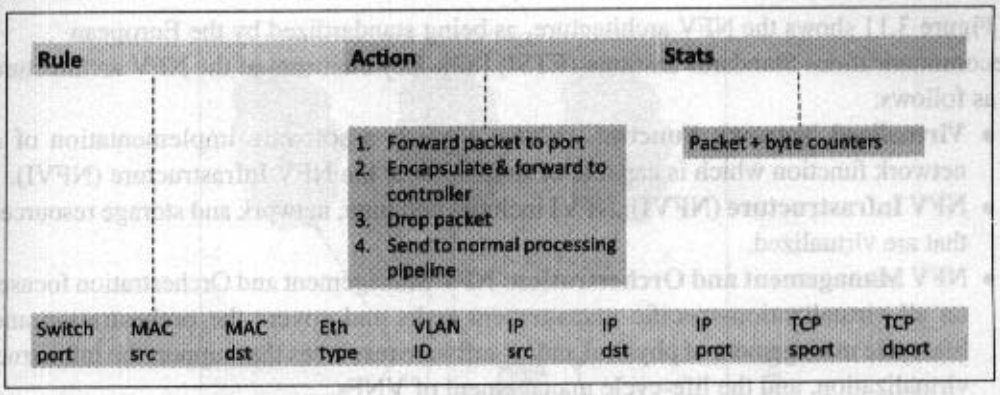


Figure 3.10: OpenFlow flow table

3.4.2 Network Function Virtualization

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

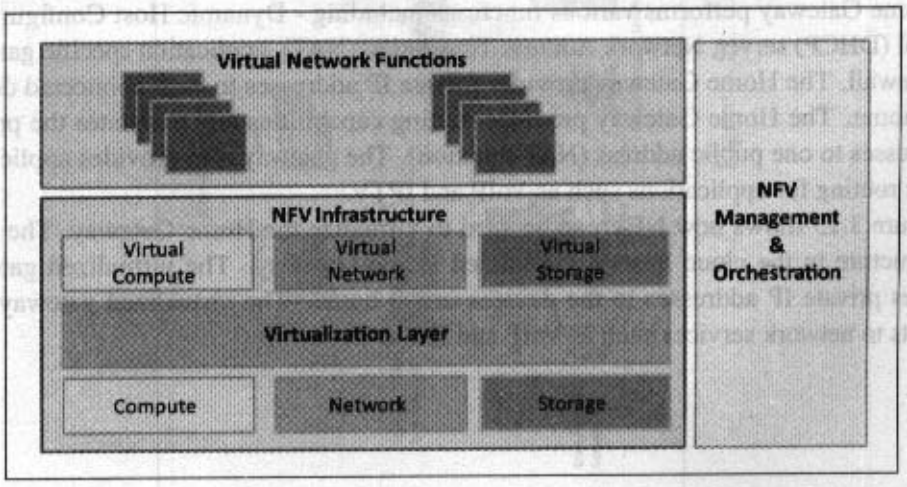


Figure 3.11: NFV architecture

Figure 3.11 shows the NFV architecture, as being standardized by the European Telecommunications Standards Institute (ETSI) [82]. Key elements of the NFV architecture are as follows:

- **Virtualized Network Function (VNF)**: VNF is a software implementation of a network function which is capable of running over the NFV Infrastructure (NFVI).
- **NFV Infrastructure (NFVI)**: NFVI includes compute, network and storage resources that are virtualized.
- **NFV Management and Orchestration**: NFV Management and Orchestration focuses on all virtualization-specific management tasks and covers the orchestration and life-cycle management of physical and/or software resources that support the infrastructure virtualization, and the life-cycle management of VNFs.

NFV comprises of network functions implemented in software that run on virtualized resources in the cloud. NFV enables separation of network functions which are implemented in software from the underlying hardware. Thus network functions can be easily tested and upgraded by installing new software while the hardware remains the same. Virtualizing network functions reduces the equipment costs and also reduces power consumption. The multi-tenanted nature of the cloud allows virtualized network functions to be shared for multiple network services. NFV is applicable only to data plane and control plane functions in fixed and mobile networks.

Let us look at an example of how NFV can be used for virtualization of the home networks. Figure 3.12 shows a home network with a Home Gateway that provides Wide Area Network (WAN) connectivity to enable services such as Internet, IPTV, VoIP, etc. The Home Gateway performs various functions including - Dynamic Host Configuration Protocol (DHCP) server, Network Address Translation (NAT), application specific gateway and Firewall. The Home Gateway provides private IP addresses to each connected device in the home. The Home Gateway provides routing capabilities and translates the private IP addresses to one public address (NAT function). The gateway also provides application specific routing for applications such as VoIP and IPTV.

Figure 3.13 shows how NFV can be used to virtualize the Home Gateway. The NFV infrastructure in the cloud hosts a virtualized Home Gateway. The virtualized gateway provides private IP addresses to the devices in the home. The virtualized gateway also connects to network services such as VoIP and IPTV.

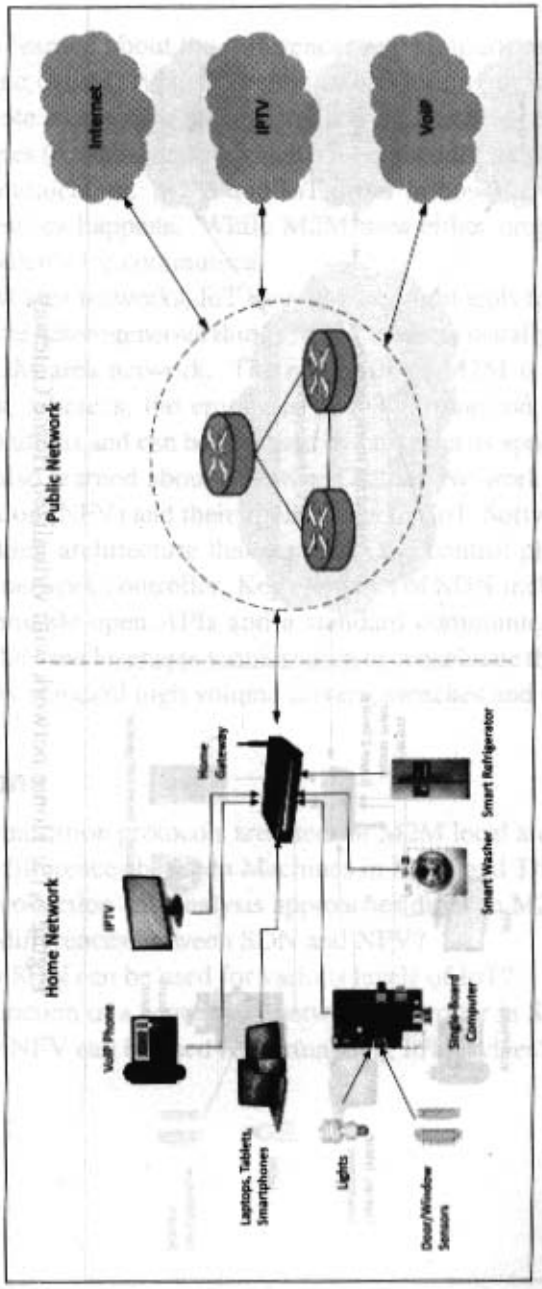


Figure 3.12: Conventional home network architecture

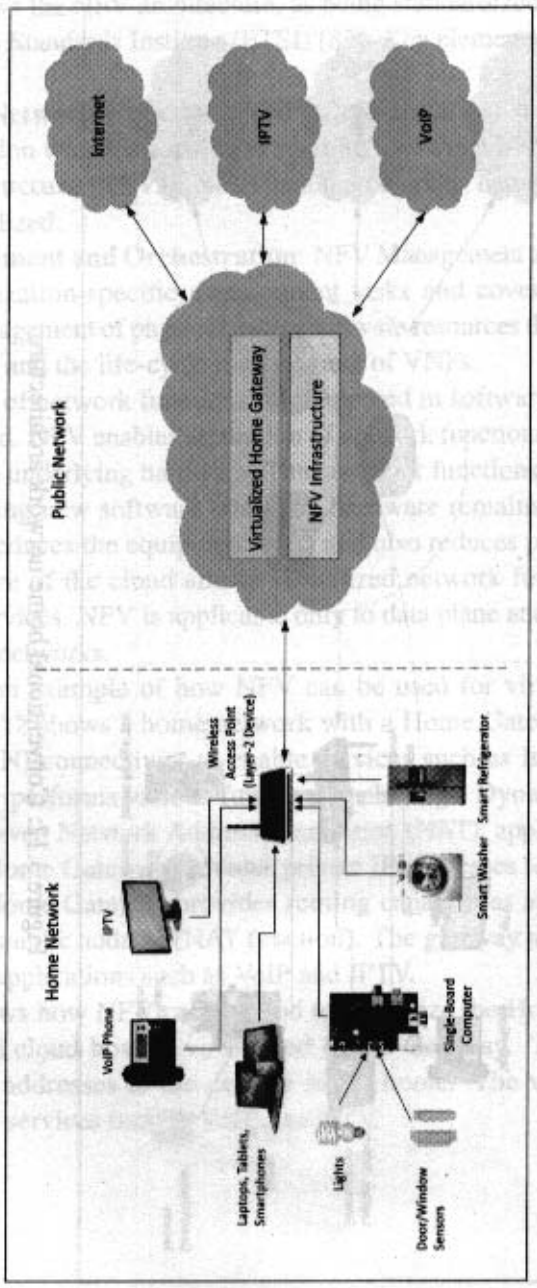
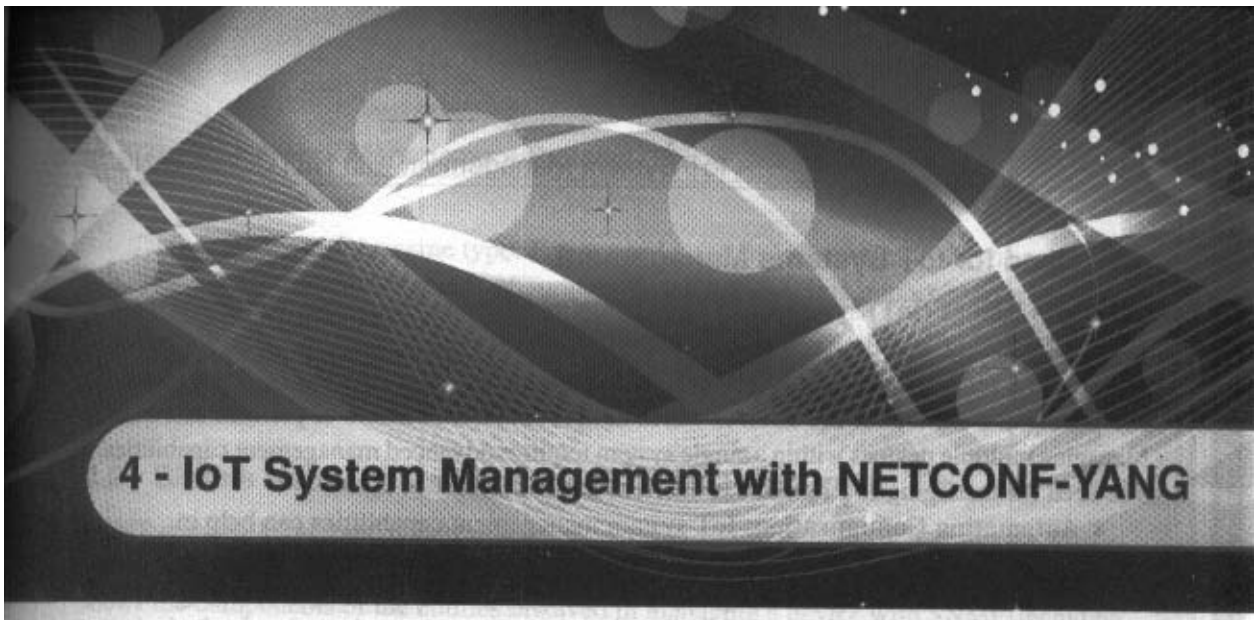


Figure 3.13: Home network with virtualized Home Gateway



4 - IoT System Management with NETCONF-YANG

This Chapter Covers

- Need for IoT Systems Management
- SNMP
- Network Operator Requirements
- NETCONF
- YANG
- IoT Systems Management with NETCONF-YANG

Summary

In this chapter you learned about the differences and similarities between IoT and M2M. Machine-to-Machine (M2M) typically refers to networking of machines (or devices) for the purpose of remote monitoring and control and data exchange. An M2M area network comprises of machines (or M2M nodes) which have embedded hardware modules for sensing, actuation and communication. M2M and IoT differ in how the communication between the machines or devices happens. While M2M uses either proprietary or non-IP based communication protocols for communication within the M2M area networks, IoT uses IP-based protocols for communication. While IoT systems can have heterogeneous things M2M systems usually have the same machine types within an M2M area network. The emphasis of M2M is more on hardware with embedded modules, whereas, the emphasis of IoT is more on software. M2M data is collected in point solutions and can be accessed by on-premisis applications. IoT is collected in the cloud. You also learned about Software Defined Networking (SDN) and Network Function Virtualization (NFV) and their applications for IoT. Software-Defined Networking (SDN) is a networking architecture that separates the control plane from the data plane and centralizes the network controller. Key elements of SDN include centralized network controller, programmable open APIs and a standard communication interface. NFV is complementary to SDN and leverages virtualization to consolidate the heterogeneous network devices onto industry standard high volume servers, switches and storage.

Review 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?

4.1 Need for IoT Systems Management

Internet of Things (IoT) systems can have complex software, hardware and deployment designs including sensors, actuators, software and network resources, data collection and analysis services and user interfaces. IoT systems can have distributed deployments comprising of a number of IoT devices which collect data from sensors or perform actuation. Managing multiple devices within a single system requires advanced management capabilities. The need for managing IoT systems is described as follows:

- **Automating Configuration**: IoT system management capabilities can help in automating the system configurations. System management interfaces provide predictable and easy to use management capability and the ability to automate system configuration. Automation becomes even more important when a system consists of multiple devices or nodes. In such cases automating the system configuration ensures that all devices have the same configuration and variations or errors due to manual configurations are avoided.
- **Monitoring Operational & Statistical Data**: Operational data is the data which is related to the system's operating parameters and is collected by the system at runtime. Statistical data is the data which describes the system performance (e.g. CPU and memory usage). Management systems can help in monitoring operational and statistical data of a system. This data can be used for fault diagnosis or prognosis.
- **Improved Reliability**: A management system that allows validating the system configurations before they are put into effect can help in improving the system reliability.
- **System Wide Configuration**: For IoT systems that consist of multiple devices or nodes, ensuring system-wide configuration can be critical for the correct functioning of the system. Management approaches in which each device is configured separately (either through a manual or automated process) can result in system faults or undesirable outcomes. This happens when some devices are running on an old configuration while others start running on new configuration. To avoid this, system wide configuration is required where all devices are configured in a single atomic transaction. This ensures that the configuration changes are either applied to all devices or to none. In the event of a failure in applying the configuration to one or more devices, the configuration changes are rolled back. This 'all or nothing' approach ensures that the system works as expected.
- **Multiple System Configurations**: For some systems it may be desirable to have multiple valid configurations which are applied at different times or in certain conditions.
- **Retrieving & Reusing Configurations**: Management systems which have the capability of retrieving configurations from devices can help in reusing the configurations for

other devices of the same type. For example, for an IoT system which has multiple devices and requires same configuration for all devices, it is important to ensure that when a new device is added, the same configuration is applied. For such cases, the management system can retrieve the current configuration from a device and apply the same to the new devices.

4.2 Simple Network Management Protocol (SNMP)

SNMP is a well-known and widely used network management protocol that allows monitoring and configuring network devices such as routers, switches, servers, printers, etc. Figure 4.1 shows the components of the entities involved in managing a device with SNMP, including the Network Management Station (NMS), Managed Device, Management Information Base (MIB) and the SNMP Agent that runs on the device. NMS executes SNMP commands to monitor and configure the Managed Device. The Managed Device contains the MIB which has all the information of the device attributes to be managed. MIBs use the Structure of Management Information (SMI) notation for defining the structure of the management data. The structure of management data is defined in the form of variables which are identified by object identifiers (OIDs), which have a hierarchical structure. Management applications can either get or set the values of these variables. SNMP is an application layer protocol that uses User Datagram Protocol (UDP) as the transport protocol.

4.2.1 Limitations of SNMP

While Simple Network Management Protocol (SNMP) has been the most popular protocol for network management, it has several limitations which may make it unsuitable for configuration management.

- SNMP was designed to provide a simple management interface between the management applications and the managed devices. SNMP is stateless in nature and each SNMP request contains all the information to process the request. The application needs to be intelligent to manage the device. For a sequence of SNMP interactions, the application needs to maintain state and also to be smart enough to roll back the device into a consistent state in case of errors or failures in configuration.
- SNMP is a connectionless protocol which uses UDP as the transport protocol, making it unreliable as there was no support for acknowledgement of requests.
- MIBs often lack writable objects without which device configuration is not possible using SNMP. With the absence of writable objects, SNMP can be used only for device monitoring and status polling.
- It is difficult to differentiate between configuration and state data in MIBs.

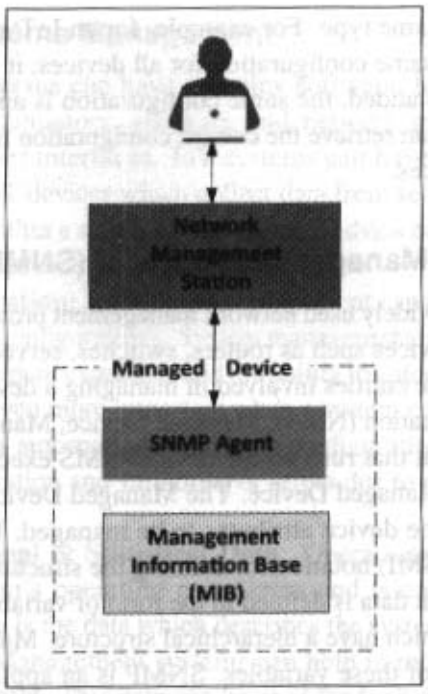


Figure 4.1: Managing a device with SNMP

- Retrieving the current configuration from a device can be difficult with SNMP. SNMP does not support easy retrieval and playback of configurations.
- Earlier versions of SNMP did not have strong security features making the management information vulnerable to network intruders. Though security features were added in the later versions of SNMP, it increased the complexity a lot.

4.3 Network Operator Requirements

To address the limitations of the existing network management protocols and plan the future work on network management, the Internet Architecture Board (IAB), which oversees the Internet Engineering Task Force (IETF) held a workshop on network management in 2002 that brought together network operators and protocol developers. Based on the inputs from operators, a list of operator requirements was prepared [122]. The following points provide a brief overview of the operator requirements.

- **Ease of use**: From the operators point of view, ease of use is the key requirement for

any network management technology.

- **Distinction between configuration and state data**: Configuration data is the set of writable data that is required to transform the system from its initial state to its current state. State data is the data which is not configurable. State data includes operational data which is collected by the system at runtime and statistical data which describes the system performance. For an effective management solution, it is important to make a clear distinction between configuration and state data.
- **Fetch configuration and state data separately**: In addition to making a clear distinction between configuration and state data, it should be possible to fetch the configuration and state data separately from the managed device. This is useful when the configuration and state data from different devices needs to be compared.
- **Configuration of the network as a whole**: It should be possible for operators to configure the network as a whole rather than individual devices. This is important for systems which have multiple devices and configuring them within one network wide transaction is required to ensure the correct operation of the system.
- **Configuration transactions across devices**: Configuration transactions across multiple devices should be supported.
- **Configuration deltas**: It should be possible to generate the operations necessary for going from one configuration state to another. The devices should support configuration deltas with minimal state changes.
- **Dump and restore configurations**: It should be possible to dump configurations from devices and restore configurations to devices.
- **Configuration validation**: It should be possible to validate configurations.
- **Configuration database schemas**: There is a need for standardized configuration database schemas or data models across operators.
- **Comparing configurations**: Devices should not arbitrarily reorder data, so that it is possible to use text processing tools such as $diff$ to compare configurations.
- **Role-based access control**: Devices should support role-based access control model, so that a user is given the minimum access necessary to perform a required task.
- **Consistency of access control lists**: It should be possible to do consistency checks of access control lists across devices.
- **Multiple configuration sets**: There should be support for multiple configurations sets on devices. This way a distinction can be provided between candidate and active configurations.
- **Support for both data-oriented and task-oriented access control**: While SNMP access control is data-oriented, CLI access control is usually task oriented. There should be support for both types of access control.

the role of a NETCONF client. For managing a network device the client establishes a NETCONF session with the server. When a session is established the client and server exchange 'hello' messages which contain information on their capabilities. Client can then send multiple requests to the server for retrieving or editing the configuration data. NETCONF allows the management client to discover the capabilities of the server (on the device). NETCONF gives access to the native capabilities of the device.

NETCONF defines one or more configuration datastores. A configuration store contains all the configuration information to bring the device from its initial state to the operational state. By default a <running> configuration store is present. Additional configuration datastores such as <startup> and <candidate> can be defined in the capabilities.

NETCONF is a connection oriented protocol and NETCONF connection persists between protocol operations. For authentication, data integrity, and confidentiality, NETCONF depends on the transport protocol, e.g., SSH or TLS. NETCONF overcomes the limitations of SNMP and is suitable not only for monitoring state information, but also for configuration management.

4.5 YANG

YANG is a data modeling language used to model configuration and state data manipulated by the NETCONF protocol [137, 124]. YANG modules contain the definitions of the configuration data, state data, RPC calls that can be issued and the format of the notifications. YANG modules defines the data exchanged between the NETCONF client and server. A module comprises of a number of 'leaf' nodes which are organized into a hierarchical tree structure. The 'leaf' nodes are specified using the 'leaf' or 'leaf-list' constructs. Leaf nodes are organized using 'container' or 'list' constructs. A YANG module can import definitions from other modules. Constraints can be defined on the data nodes, e.g. allowed values. YANG can model both configuration data and state data using the 'config' statement. YANG defines four types of nodes for data modeling as shown in Table 4.2.

Let us now look at an example of a YANG module. Box 4.1 shows a YANG module for a "network-enabled toaster". This YANG module is a YANG version of the toaster Management Information Base (MIB). We use the Toaster MIB since it has been widely used as an example in introductory tutorials on SNMP to explain how SNMP can be used for managing a network-connected toaster. A YANG module has several sections starting from header information, followed by imports and includes, type definitions, configuration and operational data declarations, and RPC and notification declarations. The toaster YANG module begins with the header information followed by identity declarations which define various bread types. The leaf nodes ('toasterManufacturer', 'toasterModelNumber' and

Operation	Description
connect	Connect to a NETCONF server
get	Retrieve the running configuration and state information
get-config	Retrieve all or a portion of a configuration datastore
edit-config	Loads all or part of a specified configuration to the specified target configuration
copy-config	Create or replace an entire target configuration datastore with a complete source configuration
delete-config	Delete the contents of a configuration datastore
lock	Lock a configuration datastore for exclusive edits by a client
unlock	Release the lock on a configuration datastore
get-schema	This operation is used to retrieve a schema from the NETCONF server
commit	Commit the candidate configuration as the device's new current configuration
close-session	Gracefully terminate a NETCONF session
kill-session	Forcefully terminate a NETCONF session

Table 4.1: List of commonly used NETCONF RPC methods

'toasterStatus') are defined in the 'toaster' container. Each leaf node definition has a type and optionally a description and default value. The module has two RPC definitions ('make-toast' and 'cancel-toast'). A tree representation of the toaster YANG module is shown in Figure 4.3.

Box 4.1: YANG version of the Toaster-MIB

```
module toaster {

  yang-version 1;

  namespace
   "http://example.com/ns/toaster";

  prefix toast;
```

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

Table 4.2: YANG Node Types

```
revision "2009-11-20" {
 description
   "Toaster module";
}

identity toast-type {
 description
   "Base for all bread types";
}

identity white-bread {
 base toast:toast-type;
}

identity wheat-bread {
 base toast-type;
}

identity wonder-bread {
 base toast-type;
}

identity frozen-waffle {
```

```
 base toast-type;
}

identity frozen-bagel {
 base toast-type;
}

identity hash-brown {
 base toast-type;
}

typedef DisplayString {
 type string {
   length "0 ..  255";
 }
}

container toaster {
 presence
   "Indicates the toaster service is available";
 description
   "Top-level container for all toaster database objects.";
 leaf toasterManufacturer {
   type DisplayString;
   config false;
   mandatory true;
 }

 leaf toasterModelNumber {
   type DisplayString;
   config false;
   mandatory true;
 }

 leaf toasterStatus {
   type enumeration {
    enum "up" {
      value 1;
    }
    enum "down" {
      value 2;
    }
   }
   config false;
   mandatory true;
```

```
    }
  }

  rpc make-toast {
   input {
     leaf toasterDoneness {
      type uint32 {
        range "1 ..  10";
      }
      default '5';
     }

     leaf toasterToastType {
      type identityref {
        base toast:toast-type;
      }
      default 'wheat-bread';
     }
   }
  }

  rpc cancel-toast {
   description
     "Stop making toast, if any is being made.";
  }

  notification toastDone {
   leaf toastStatus {
     type enumeration {
      enum "done" {
        value 0;
        description "The toast is done.";
      }
      enum "cancelled" {
        value 1;
        description
          "The toast was cancelled.";
      }
      enum "error" {
        value 2;
        description
          "The toaster service was disabled or
               the toaster is broken.";
      }
     }
```

```
    }
  }
}
```

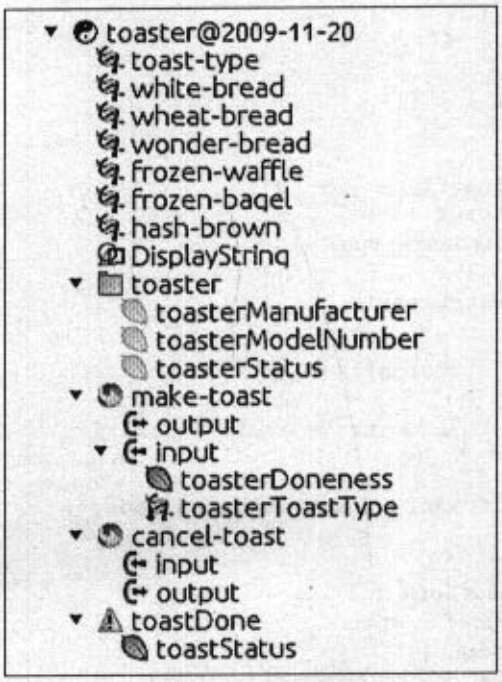


Figure 4.3: Visual representation of the Toaster YANG Module

Let us look at another example of a YANG module. Box 4.2 shows a YANG module for configuring a HAProxy load balancer for a commercial website. The module includes containers for global, defaults, frontend and backend sections of an HAProxy configuration. In the global container the leaf nodes for configuration data such as max-connections and mode are defined. In the defaults container the leaf nodes for configuration data such as retries, *contimeout*, etc. are defined. The front-end port bindings are defined in the frontend container. The backend container has definitions on the servers to load balance. A reusable tree structure called 'server-list' is used for the server definitions. The 'server-list' structure is defined using the 'grouping' construct in the module. A tree representation of the HAProxy YANG module is shown in Figure 4.4.

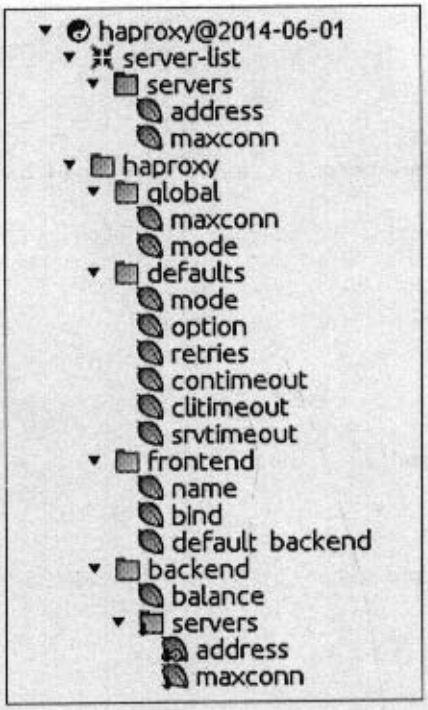


Figure 4.4: Visual representation of HAProxy YANG Module

Box 4.2: YANG Module for HAProxy configuration

```
module haproxy {
   yang-version 1;

   namespace "http://example.com/ns/haproxy";

   prefix haproxy;

   import ietf-inet-types { prefix inet; }

   description
   "YANG version of the ";

   revision "2014-06-01" {
      description
```

```
      "HAProxy module";
   }

container haproxy {
   description
   "configuration parameters for a HAProxy load balancer";

      container global {
         leaf maxconn {
         type uint32;
         default 4096;
         }

         leaf mode {
         type string;
         default 'daemon';
         }
      }

      container defaults {
         leaf mode {
         type string;
         default 'http';
         }

         leaf option {
         type string;
         default 'redispatch';
         }

         leaf retries {
         type uint32;
         default 3;
         }

         leaf contimeout {
         type uint32;
         default 5000;
         }

         leaf clitimeout {
         type uint32;
         default 50000;
         }
```

```
        leaf srvtimeout {
        type uint32;
        default 50000;
        }
    }

    container frontend {

        leaf name {
        type string;
        default 'http-in';
        }

        leaf bind {
        type string;
        default '*:80';
        }

        leaf default_backend {
        type string;
        default 'webfarm';
        }
    }

    container backend {
        leaf balance   {
        type string;
        default 'roundrobin';
        }
        uses server-list;
    }
}

    grouping server-list {
    description "List of servers to load balance";
        list servers {
        key address;
        leaf address {
            type inet:ip-address;
        }

        leaf maxconn {
            type uint32;
            default 255;
```

```
            }
        }
    }
}
```

4.6 IoT Systems Management with NETCONF-YANG

In this section you will learn how to manage IoT systems with NECONF and YANG. Figure 4.5 shows the generic approach of IoT device management with NETCONF-YANG.

Let is look at the roles of the various components:

- **Management System**: The operator uses a Management System to send NETCONF messages to configure the IoT device and receives state information and notifications from the device as NETCONF messages.
- **Management API**: Management API allows management applications to start NETCONF sessions, read and write configuration data, read state data, retrieve configurations, and invoke RPCs, programmatically, in the same way as an operator can.
- **Transaction Manager**: Transaction Manager executes all the NETCONF transactions and ensures that the ACID (Atomicity, Consistency, Isolation, Durability) properties hold true for the transactions. Atomicity property ensures that a transaction is executed either completely or not at all. Consistency property ensures that a transaction brings the device configuration from one valid state to another. Isolation property ensures that concurrent execution of transactions results in the same device configuration as if transactions were executed serially in order. Durability property ensures that a transaction once committed will persist.
- **Rollback Manager** : Rollback manager is responsible for generating all the transactions necessary to rollback a current configuration to its original state.
- **Data Model Manager**: The Data Model manager keeps track of all the YANG data models and the corresponding managed objects. The Data Model manager also keeps track of the applications which provide data for each part of a data model.
- **Configuration Validator**: Configuration validator checks if the resulting configuration after applying a transaction would be a valid configuration.
- **Configuration Database**: This database contains both the configuration and operational data.
- **Configuration API**: Using the configuration API the applications on the IoT device can read configuration data from the configuration datastore and write operational data to the operational datastore.

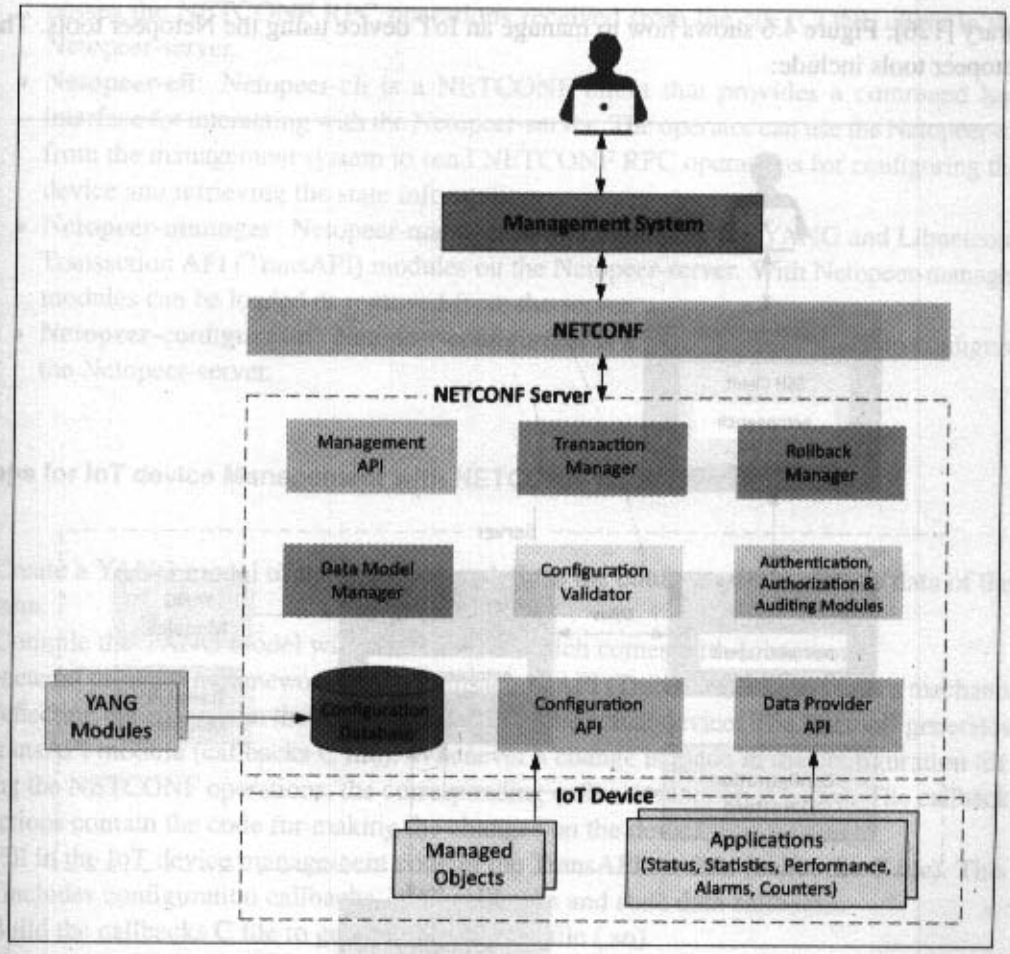


Figure 4.5: IoT device management with NETCONF-YANG - a generic approach

- **Data Provider API**: Applications on the IoT device can register for callbacks for various events using the Data Provider API. Through the Data Provider API, the applications can report statistics and operational data.

4.6.1 NETOPEER

While the previous section described a generic approach of IoT device management with NETCONF-YANG, this section describes a specific implementation based on the Netopeer tools [125]. Netopeer is set of open source NETCONF tools built on the Libnetconf

library [126]. Figure 4.6 shows how to manage an IoT device using the Netopeer tools. The Netopeer tools include:

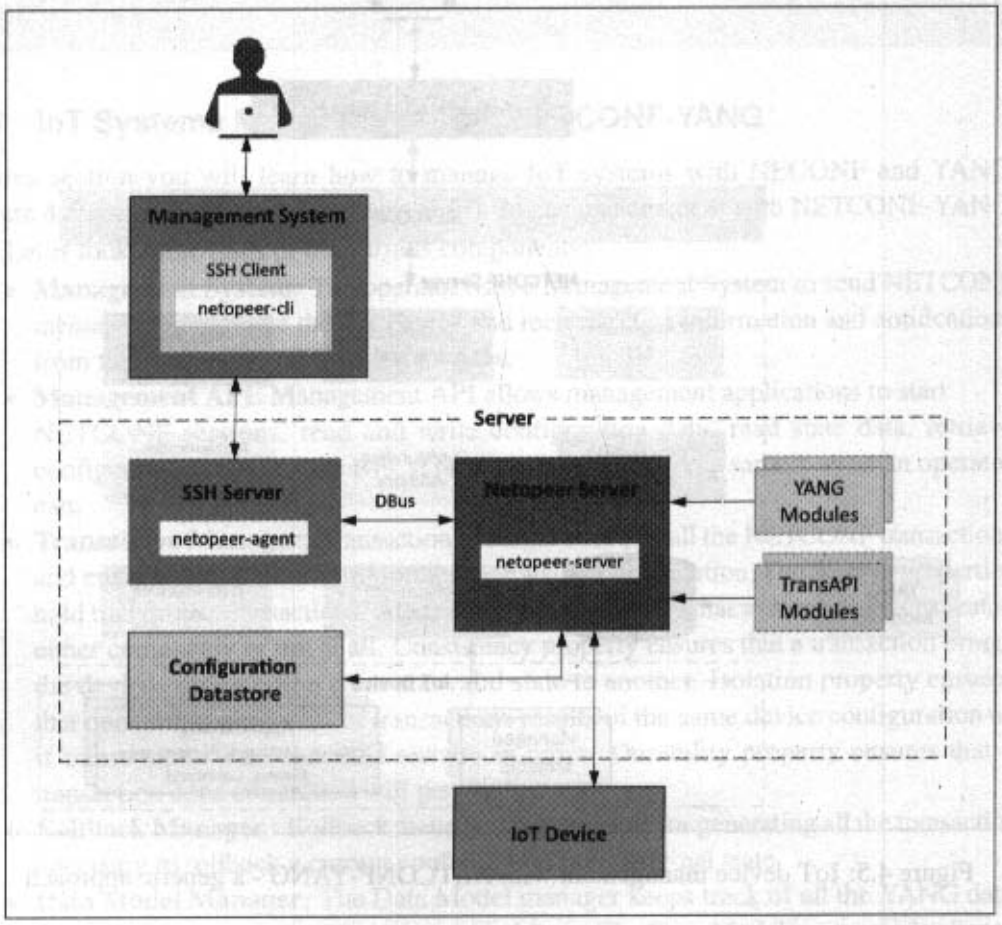


Figure 4.6: IoT device management with NETCONF - a specific approach based on Netopeer tools

- **Netopeer-server**: Netopeer-server is a NETCONF protocol server that runs on the managed device. Netopeer-server provides an environment for configuring the device using NETCONF RPC operations and also retrieving the state data from the device.
- **Netopeer-agent**: Netopeer-agent is the NETCONF protocol agent running as a SSH/TLS subsystem. Netopeer-agent accepts incoming NETCONF connection and

passes the NETCONF RPC operations received from the NETCONF client to the Netopeer-server.

- **Netopeer-cli**: Netopeer-cli is a NETCONF client that provides a command line interface for interacting with the Netopeer-server. The operator can use the Netopeer-cli from the management system to send NETCONF RPC operations for configuring the device and retrieving the state information.
- **Netopeer-manager**: Netopeer-manager allows managing the YANG and Libnetconf Transaction API (TransAPI) modules on the Netopeer-server. With Netopeer-manager modules can be loaded or removed from the server.
- **Netopeer-configurator**: Netopeer-configurator is a tool that can be used to configure the Netopeer-server.

Steps for IoT device Management with NETCONF-YANG

1. Create a YANG model of the system that defines the configuration and state data of the system.
2. Compile the YANG model with the 'lnctool' which comes with Libnetconf.
Libnetconf provides a framework called Transaction API (TransAPI) that provides a mechanism of reflecting the changes in the configuration file in the actual device. The 'lnctool' generates a TransAPI module (callbacks C file). Whenever a change is made in the configuration file using the NETCONF operations, the corresponding callback function is called. The callback functions contain the code for making the changes on the device.
3. Fill in the IoT device management code in the TransAPI module (callbacks C file). This file includes configuration callbacks, RPC callbacks and state data callbacks.
4. Build the callbacks C file to generate the library file (.so).
5. Load the YANG module (containing the data definitions) and the TransAPI module (.so binary) into the Netopeer server using the Netopeer manager tool.
6. The operator can now connect from the management system to the Netopeer server using the Netopeer CLI.
7. Operator can issue NETCONF commands from the Netopeer CLI. Commands can be issued to change the configuration data, get operational data or execute an RPC on the IoT device.

In Chapter 11, detailed case studies on IoT device Management using the above steps are provided.

Summary

In this chapter you learned about the need for IoT systems management. IoT system management capabilities can help in automating the system configurations. Management systems can collect operational and statistical data from IoT devices which can be used for fault diagnosis or prognosis. For IoT systems that consist of multiple devices, system wide configuration is important to ensure that all devices are configured within one transaction and the transactions are atomic. It is desirable for devices to have multiple configurations with one of them being the active and running configuration. Management systems which have the capability of retrieving configurations from devices can help in reusing the configurations for other devices of the same type. SNMP has been a popular network management protocol, however it has several limitations which make it unsuitable for IoT device management. Network Configuration Protocol (NETCONF), which is a session-based network management protocol, is more suitable for IoT device management. NETCONF works on SSH transport protocol and provides various operations to retrieve and edit configuration data from devices. The configuration data resides within a NETCONF configuration datastore on the server. The NETCONF server resides on the network device. The device configuration and state data is modeled using the YANG data modeling language. There is a clear separation of configuration and state data in the YANG models. You learned about a generic approach for IoT device management and the roles of various components such as the Management API, Transaction Manager, Rollback Manager, Data Model Manager, Configuration Validator, Configuration Database, Configuration API and Data Provider API. You learned about the Netopeer tools for NETCONF and the steps for IoT device Management using these tools.

Review Questions

1. Why is network wide configuration important for IoT systems with multiple nodes?
2. Which limitations make SNMP unsuitable for IoT systems?
3. What is the difference between configuration and state data?
4. What is the role of a NETCONF server?
5. What is the function of a data model manager?
6. Describe the roles of YANG and TransAPI modules in device management?

Part II

DEVELOPING INTERNET OF THINGS

5 - IoT Platforms Design Methodology

This Chapter Covers

IoT Design Methodology that includes:

- Purpose & Requirements Specification
- Process Specification
- Domain Model Specification
- Information Model Specification
- Service Specifications
- IoT Level Specification
- Functional View Specification
- Operational View Specification
- Device & Component Integration
- Application Development

5.1 Introduction

IoT systems comprise of multiple components and deployment tiers. In Chapter-1, we defined six IoT system levels. Each level is suited for different applications and has different component and deployment configurations. Designing IoT systems can be a complex and challenging task as these systems involve interactions between various components such as IoT devices and network resources, web services, analytics components, application and database servers. Due to a wide range of choices available for each of these components, IoT system designers may find it difficult to evaluate the available alternatives. IoT system designers often tend to design IoT systems keeping specific products/services in mind. Therefore, these designs are tied to specific product/service choices made. This leads to product, service or vendor lock-in, which while satisfactory to the dominant vendor, is unacceptable to the customer. For such systems, updating the system design to add new features or replacing a particular product/service choice for a component becomes very complex, and in many cases may require complete re-design of the system.

In this Chapter, we propose a generic design methodology for IoT system design which is independent of specific product, service or programming language. IoT systems designed with the proposed methodology have reduced design, testing and maintenance time, better interoperability and reduced complexity. With the proposed methodology, IoT system designers can compare various alternatives for the IoT system components. The methodology described in this Chapter is generally based on the IoT-A reference model [75], but is broad enough to embrace other industry efforts as well. Later chapters in this book describe the implementation aspects of various steps in the proposed methodology.

5.2 IoT Design Methodology

Figure 5.1 shows the steps involved in the IoT system design methodology. Each of these steps is explained in the sections that follow. To explain these steps, we use the example of a smart IoT-based home automation system.

5.2.1 Step 1: Purpose & Requirements Specification

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

Applying this to our example of a smart home automation system, the purpose and requirements for the system may be described as follows:

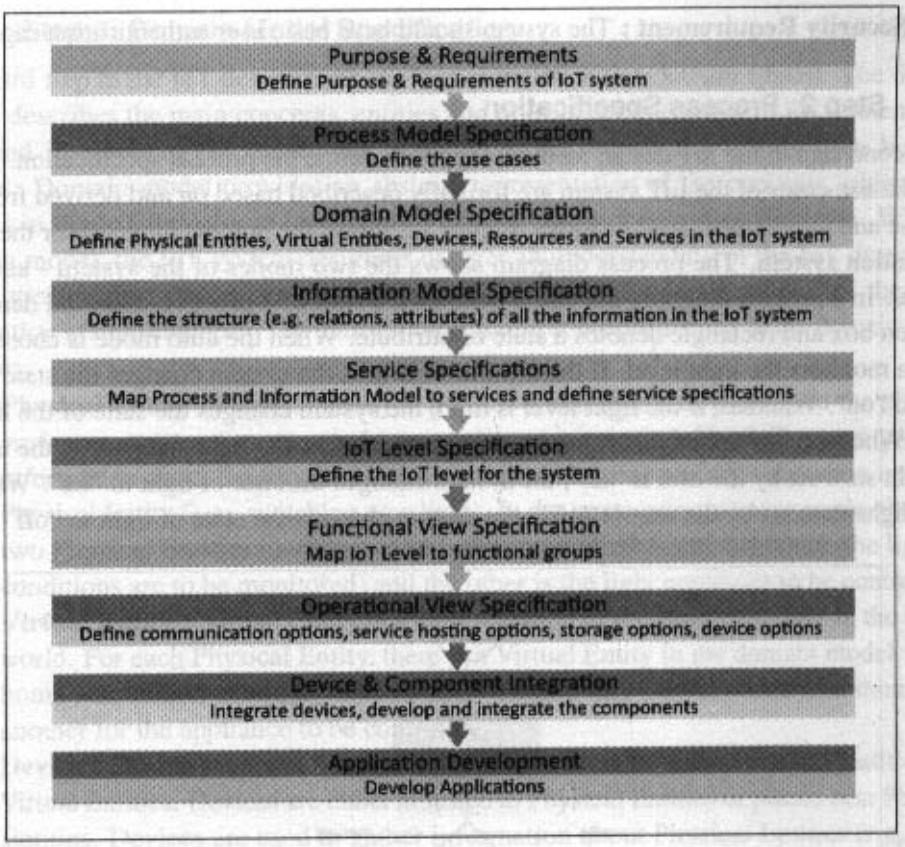


Figure 5.1: Steps involved in IoT system design methodology

- **Purpose :** A home automation system that allows controlling of the lights in a home remotely using a web application.
- **Behavior :** The home automation system should have auto and manual modes. In auto mode, the system measures the light level in the room and switches on the light when it gets dark. In manual mode, the system provides the option of manually and remotely switching on/off the light.
- **System Management Requirement :** The system should provide remote monitoring and control functions.
- **Data Analysis Requirement :** The system should perform local analysis of the data.
- **Application Deployment Requirement :** The application should be deployed locally on the device, but should be accessible remotely.

- **Security Requirement :** The system should have basic user authentication capability.

5.2.2 Step 2: Process Specification

The second step in the IoT design methodology is to define the process specification. In this step, the use cases of the IoT system are formally described based on and derived from the purpose and requirement specifications. Figure 5.2 shows the process diagram for the home automation system. The process diagram shows the two modes of the system - auto and manual. In a process diagram, the circle denotes the start of a process, diamond denotes a decision box and rectangle denotes a state or attribute. When the auto mode is chosen, the system monitors the light level. If the light level is low, the system changes the state of the light to "on". Whereas, if the light level is high, the system changes the state of the light to "off". When the manual mode is chosen, the system checks the light state set by the user. If the light state set by the user is "on", the system changes the state of light to "on". Whereas, if the light state set by the user is "off", the system changes the state of light to "off".

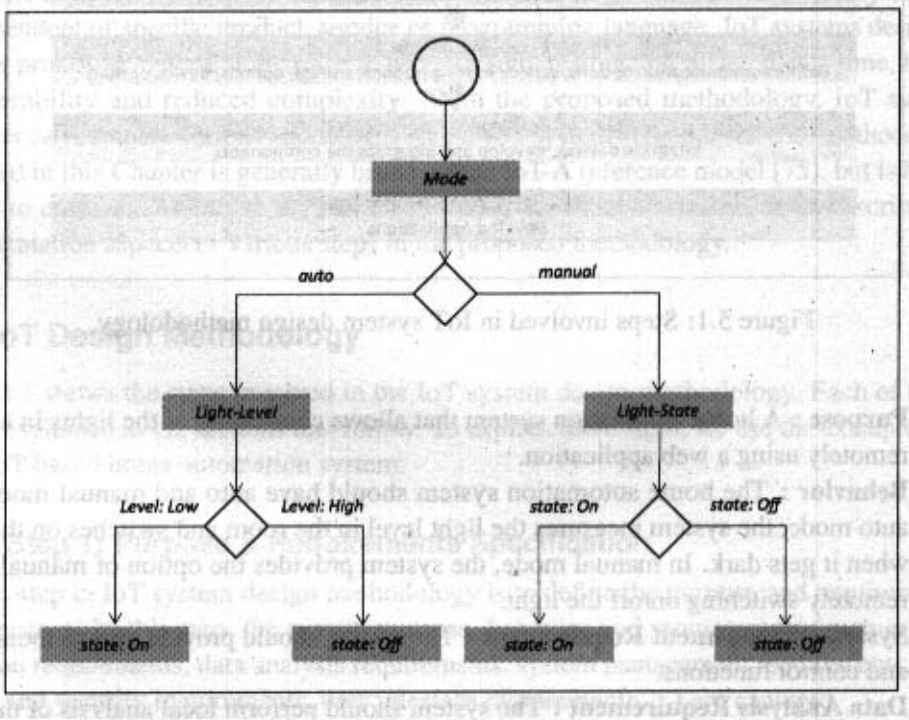


Figure 5.2: Process specification for home automation IoT system

5.2.3 Step 3: Domain Model Specification

The third step in the IoT design methodology is to define the Domain Model. The domain model describes the main concepts, entities and objects in the domain of IoT system to be designed. Domain model defines the attributes of the objects and relationships between objects. Domain model provides an abstract representation of the concepts, objects and entities in the IoT domain, independent of any specific technology or platform. With the domain model, the IoT system designers can get an understanding of the IoT domain for which the system is to be designed. Figure 5.3 shows the domain model for the home automation system example. The entities, objects and concepts defined in the domain model include:

- **Physical Entity :** Physical Entity is a discrete and identifiable entity in the physical environment (e.g. a room, a light, an appliance, a car, etc.). The IoT system provides information about the Physical Entity (using sensors) or performs actuation upon the Physical Entity (e.g., switching on a light). In the home automation example, there are two Physical Entities involved - one is the room in the home (of which the lighting conditions are to be monitored) and the other is the light appliance to be controlled.
- **Virtual Entity :** Virtual Entity is a representation of the Physical Entity in the digital world. For each Physical Entity, there is a Virtual Entity in the domain model. In the home automation example, there is one Virtual Entity for the room to be monitored, another for the appliance to be controlled.
- **Device :** Device provides a medium for interactions between Physical Entities and Virtual Entities. Devices are either attached to Physical Entities or placed near Physical Entities. Devices are used to gather information about Physical Entities (e.g., from sensors), perform actuation upon Physical Entities (e.g. using actuators) or used to identify Physical Entities (e.g., using tags). In the home automation example, the device is a single-board mini computer which has light sensor and actuator (relay switch) attached to it.
- **Resource :** Resources are software components which can be either "on-device" or "network-resources". On-device resources are hosted on the device and include software components that either provide information on or enable actuation upon the Physical Entity to which the device is attached. Network resources include the software components that are available in network (such as a database). In the home automation example, the on-device resource is the operating system that runs on the single-board mini computer.
- **Service :** Services provide an interface for interacting with the Physical Entity. Services access the resources hosted on the device or the network resources to obtain information about the Physical Entity or perform actuation upon the Physical Entity.

In the home automation example, there are three services: (1) a service that sets mode to auto or manual, or retrieves the current mode; (2) a service that sets the light appliance state to on/off, or retrieves the current light state; and (3) a controller service that runs as a native service on the device. When in auto mode, the controller service monitors the light level and switches the light on/off and updates the status in the status database. When in manual mode, the controller service retrieves the current state from the database and switches the light on/off. The process of deriving the services from the process specification and information model is described in the later sections.

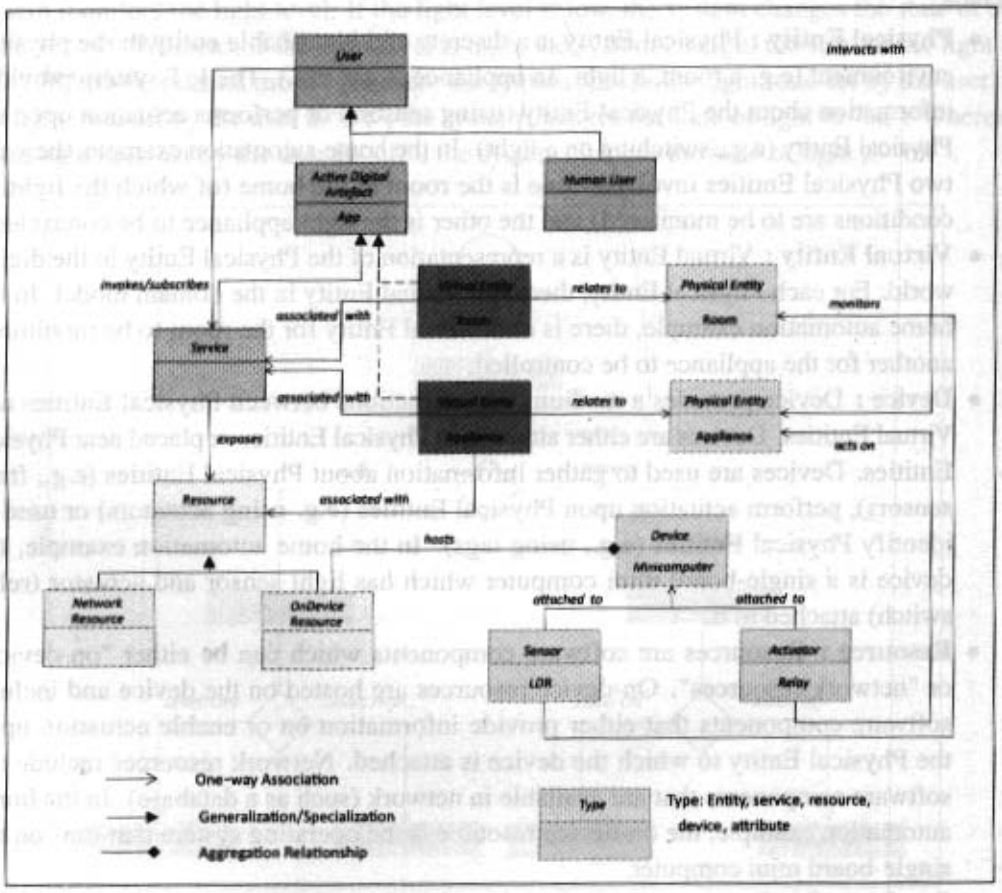


Figure 5.3: Domain model of the home automation IoT system

5.2.4 Step 4: Information Model Specification

The fourth step in the IoT design methodology is to define the Information Model. Information Model defines the structure of all the information in the IoT system, for example, attributes of Virtual Entities, relations, etc. Information model does not describe the specifics of how the information is represented or stored. To define the information model, we first list the Virtual Entities defined in the Domain Model. Information model adds more details to the Virtual Entities by defining their attributes and relations. In the home automation example, there are two Virtual Entities - a Virtual Entity for the light appliance (with attribute - light state) and a Virtual Entity for the room (with attribute - light level). Figure 5.4 shows the Information Model for the home automation system example.

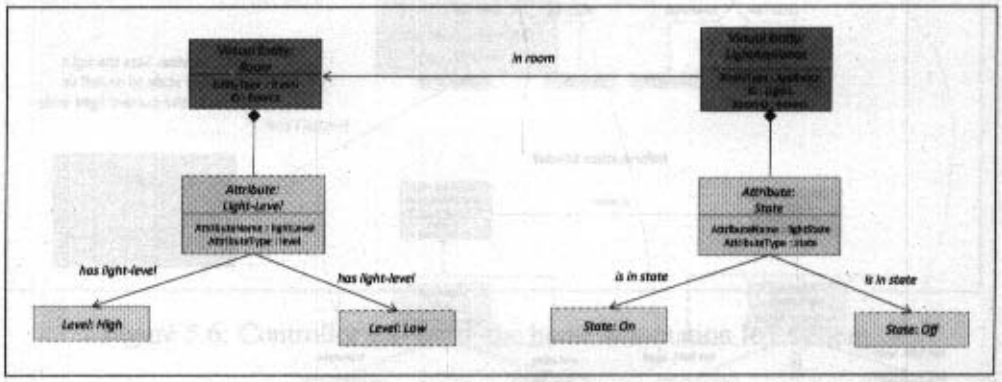


Figure 5.4: Information model of the home automation IoT system

5.2.5 Step 5: Service Specifications

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

You learned about the Process Specification and Information Model in the previous sections. Figure 5.5 shows an example of deriving the services from the process specification and information model for the home automation IoT system. From the process specification and information model, we identify the states and attributes. For each state and attribute we define a service. These services either change the state or attribute values or retrieve the current values. For example, the Mode service sets mode to auto or manual or retrieves the current mode. The State service sets the light appliance state to on/off or retrieves the current light state. The Controller service monitors the light level in auto mode and switches the

light on/off and updates the status in the status database. In manual mode, the controller service, retrieves the current state from the database and switches the light on/off.

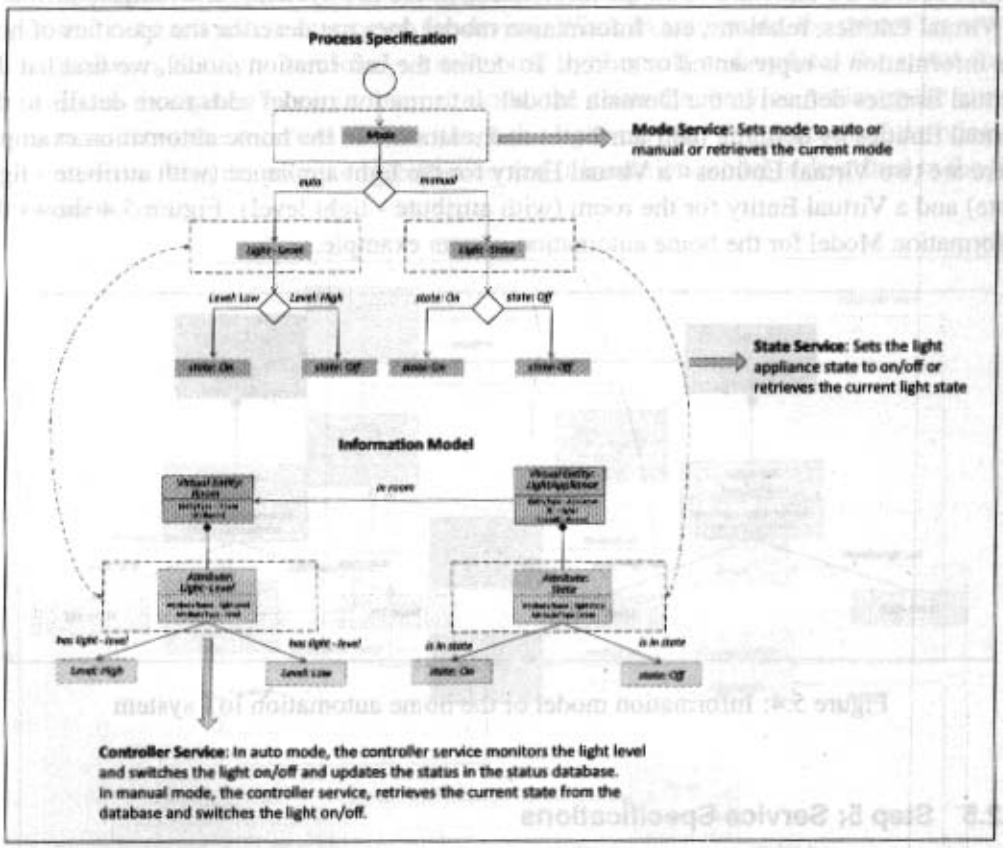


Figure 5.5: Deriving services from process specification and information model for home automation IoT system

Figures 5.6, 5.7 and 5.8 show specifications of the controller, mode and state services of the home automation system. The Mode service is a RESTful web service that sets mode to auto or manual (PUT request), or retrieves the current mode (GET request) . The mode is updated to/retrieved from the database. The State service is a RESTful web service that sets the light appliance state to on/off (PUT request), or retrieves the current light state (GET request). The state is updated to/retrieved from the status database. The Controller service runs as a native service on the device. When in auto mode, the controller service monitors

the light level and switches the light on/off and updates the status in the status database. When in manual mode, the controller service, retrieves the current state from the database and switches the light on/off.

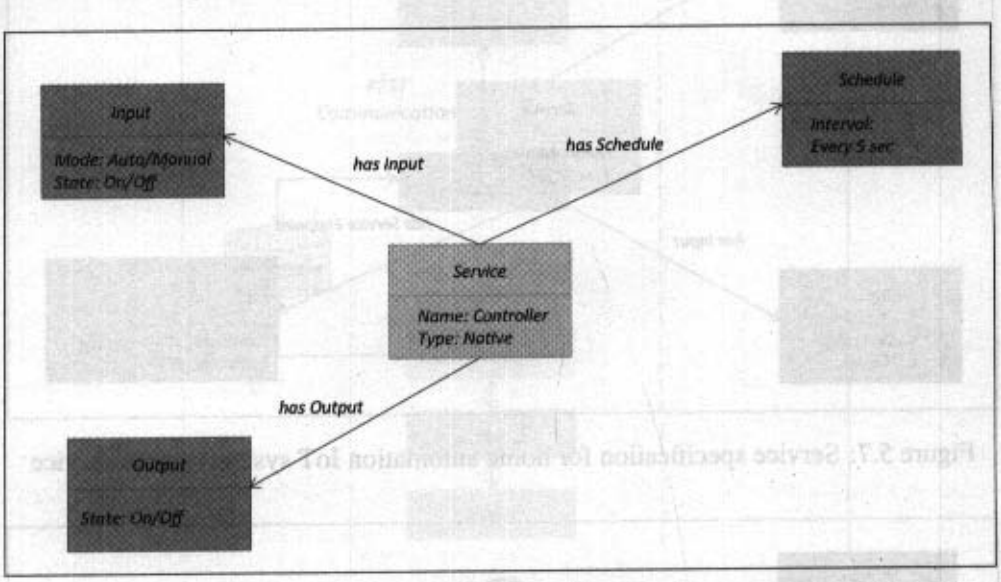


Figure 5.6: Controller service of the home automation IoT system

5.2.6 Step 6: IoT Level Specification

The sixth step in the IoT design methodology is to define the IoT level for the system. In Chapter-1, we defined five IoT deployment levels. Figure 5.9 shows the deployment level of the home automation IoT system, which is level-1.

5.2.7 Step 7: Functional View Specification

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

The Functional Groups (FG) included in a Functional View include:

- **Device :** The device FG contains devices for monitoring and control. In the home automation example, the device FG includes a single board mini-computer, a light

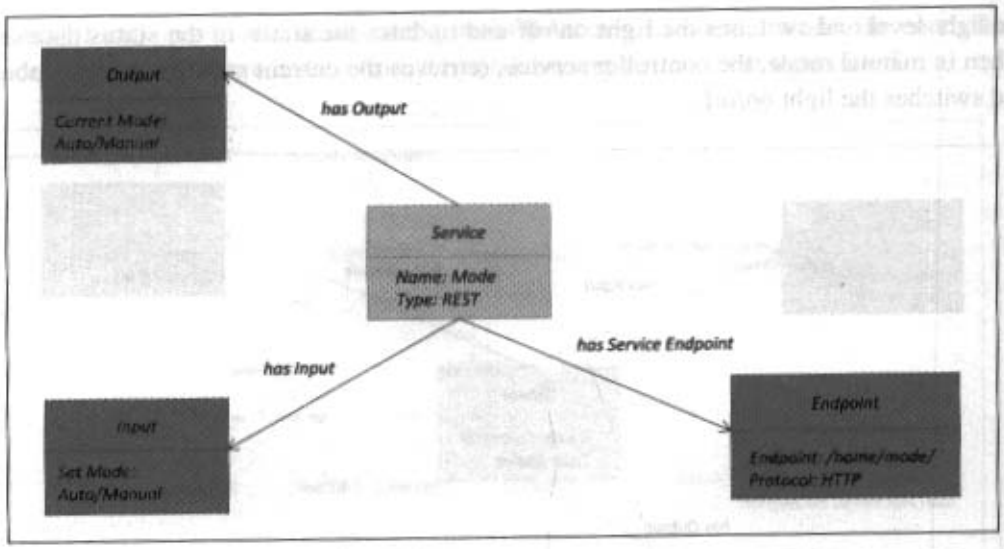


Figure 5.7: Service specification for home automation IoT system - mode service

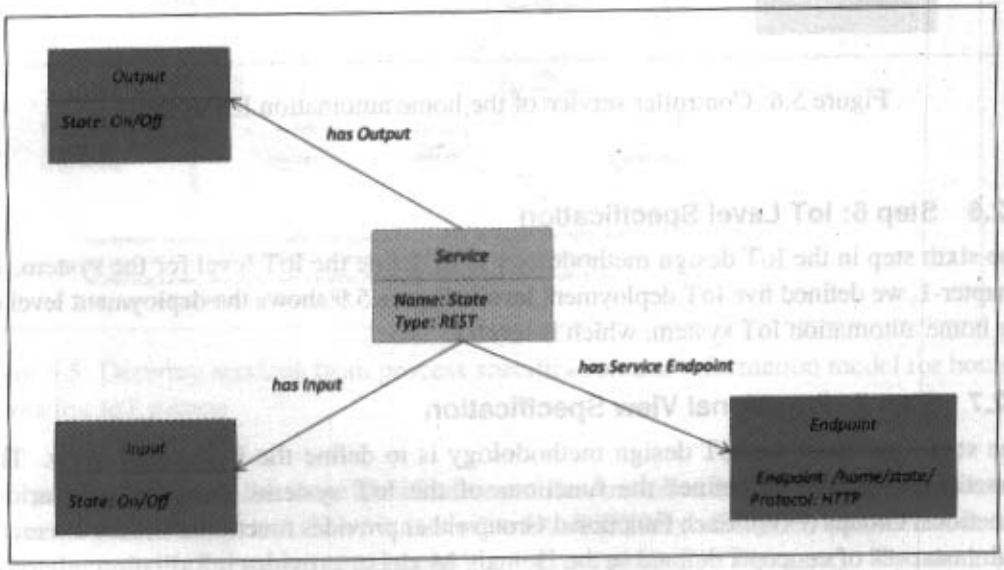


Figure 5.8: Service specification for home automation IoT system - state service

sensor and a relay switch (actuator).

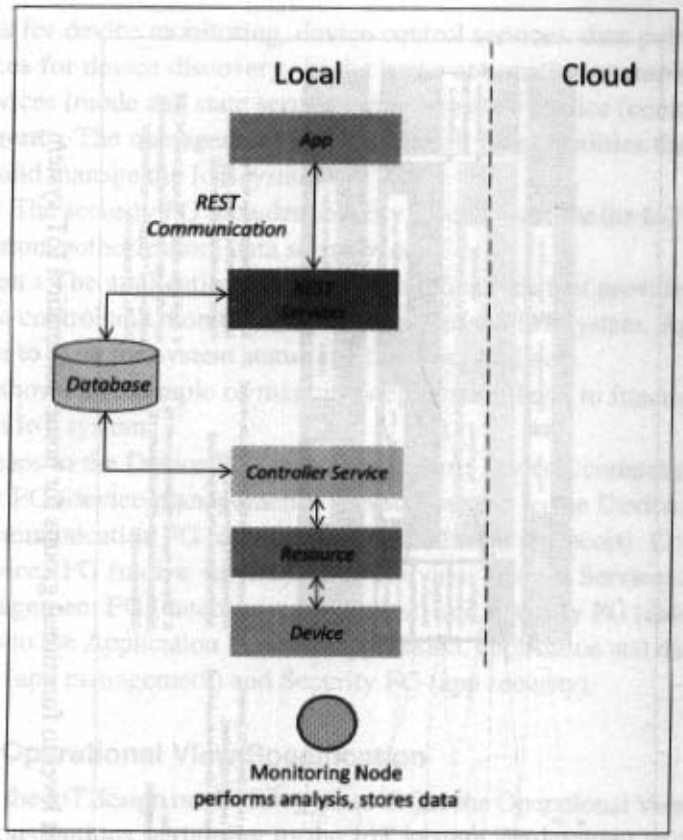


Figure 5.9: Deployment design of the home automation IoT system

- **Communication :** The communication FG handles the communication for the IoT system. The communication FG includes the communication protocols that form the backbone of IoT systems and enable network connectivity. You learned about various link, network, transport and application layer protocols in Chapter-1. The communication FG also includes the communication APIs (such as REST and WebSocket) that are used by the services and applications to exchange data over the network. In the home automation example the communication protocols include - 802.11 (link layer), IPv4/IPv6 (network layer), TCP (transport layer), and HTTP (application layer). The communication API used in the home automation examples is a REST-based API.
- **Services :** The service FG includes various services involved in the IoT system such

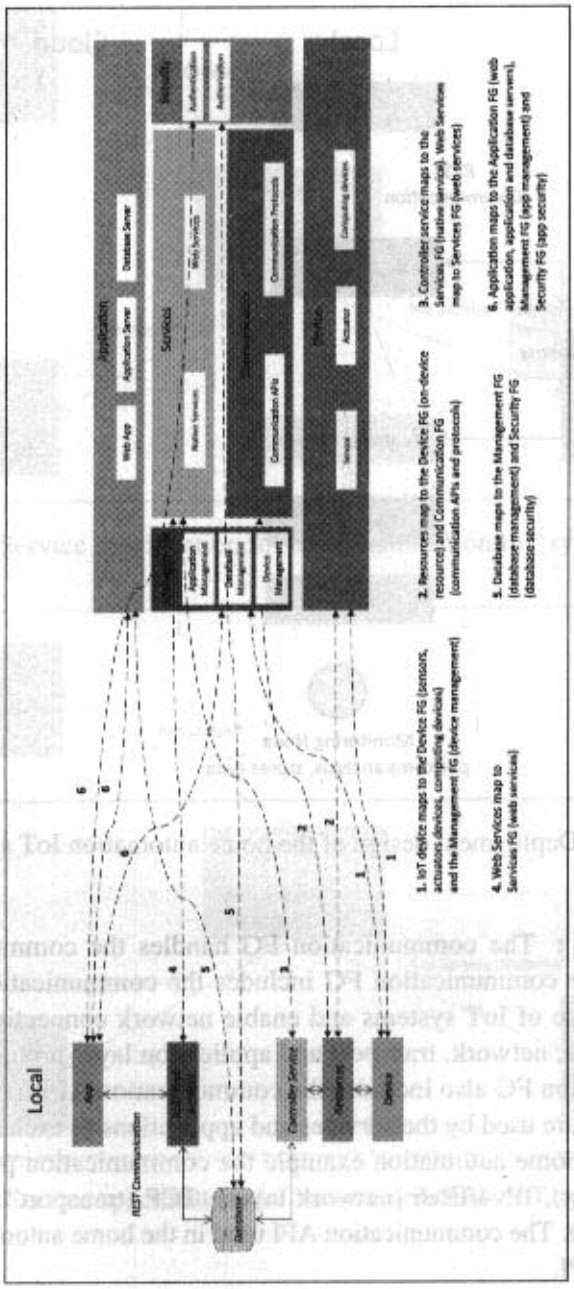


Figure 5.10: Mapping deployment level to functional groups for home automation IoT system

as services for device monitoring, device control services, data publishing services and services for device discovery. In the home automation example, there are two REST services (mode and state service) and one native service (controller service).

- **Management :** The management FG includes all functionalities that are needed to configure and manage the IoT system.
- **Security :** The security FG includes security mechanisms for the IoT system such as authentication, authorization, data security, etc.
- **Application :** The application FG includes applications that provide an interface to the users to control and monitor various aspects of the IoT system. Applications also allow users to view the system status and the processed data.

Figure 5.10 shows an example of mapping deployment level to functional groups for home automation IoT system.

IoT device maps to the Device FG (sensors, actuators devices, computing devices) and the Management FG (device management). Resources map to the Device FG (on-device resource) and Communication FG (communication APIs and protocols). Controller service maps to the Services FG (native service). Web Services map to Services FG . Database maps to the Management FG (database management) and Security FG (database security). Application maps to the Application FG (web application, application and database servers), Management FG (app management) and Security FG (app security).

5.2.8 Step 8: Operational View Specification

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

Figure 5.11 shows an example of mapping functional groups to operational view specifications for home automation IoT system.

Operational View specifications for the home automation example are as follows:

- Devices: Computing device (Raspberry Pi), light dependent resistor (sensor), relay switch (actuator).
- Communication APIs: REST APIs
- Communication Protocols: Link Layer - 802.11, Network Layer - IPv4/IPv6, Transport - TCP, Application - HTTP.
- Services:
  1. Controller Service - Hosted on device, implemented in Python and run as a native service.
  2. Mode service - REST-ful web service, hosted on device, implemented with

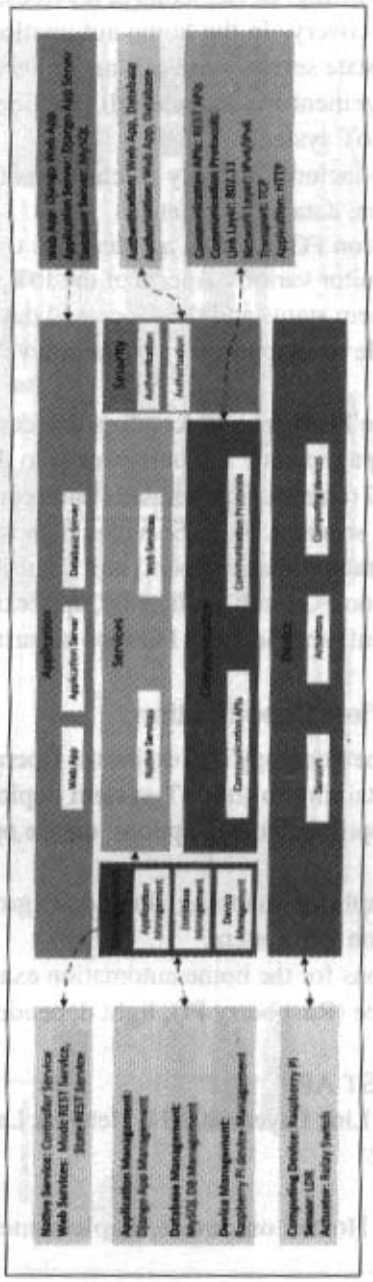


Figure 5.11: Mapping functional groups to operational view for home automation IoT system

Django-REST Framework.
3. State service - REST-ful web service, hosted on device, implemented with Django-REST Framework.

- Application:
  Web Application - Django Web Application,
  Application Server - Django App Server,
  Database Server - MySQL.
- Security:
  Authentication: Web App, Database
  Authorization: Web App, Database

- Management:
  Application Management - Django App Management
  Database Management - MySQL DB Management,
  Device Management - Raspberry Pi device Management.

5.2.9 Step 9: Device & Component Integration

The ninth step in the IoT design methodology is the integration of the devices and components. Figure 5.12 shows a schematic diagram of the home automation IoT system. The devices and components used in this example are Raspberry Pi mini computer, LDR sensor and relay switch actuator. A detailed description of Raspberry Pi board and how to interface sensors and actuators with the board is provided in later chapters.

5.2.10 Step 10: Application Development

The final step in the IoT design methodology is to develop the IoT application. Figure 5.13 shows a screenshot of the home automation web application. The application has controls for the mode (auto on or auto off) and the light (on or off). In the auto mode, the IoT system controls the light appliance automatically based on the lighting conditions in the room. When auto mode is enabled the light control in the application is disabled and it reflects the current state of the light. When the auto mode is disabled, the light control is enabled and it is used for manually controlling the light.

5.3 Case Study on IoT System for Weather Monitoring

In this section we present a case study on design of an IoT system for weather monitoring using the IoT design methodology. The purpose of the weather monitoring system is to collect data on environmental conditions such as temperature, pressure, humidity and light

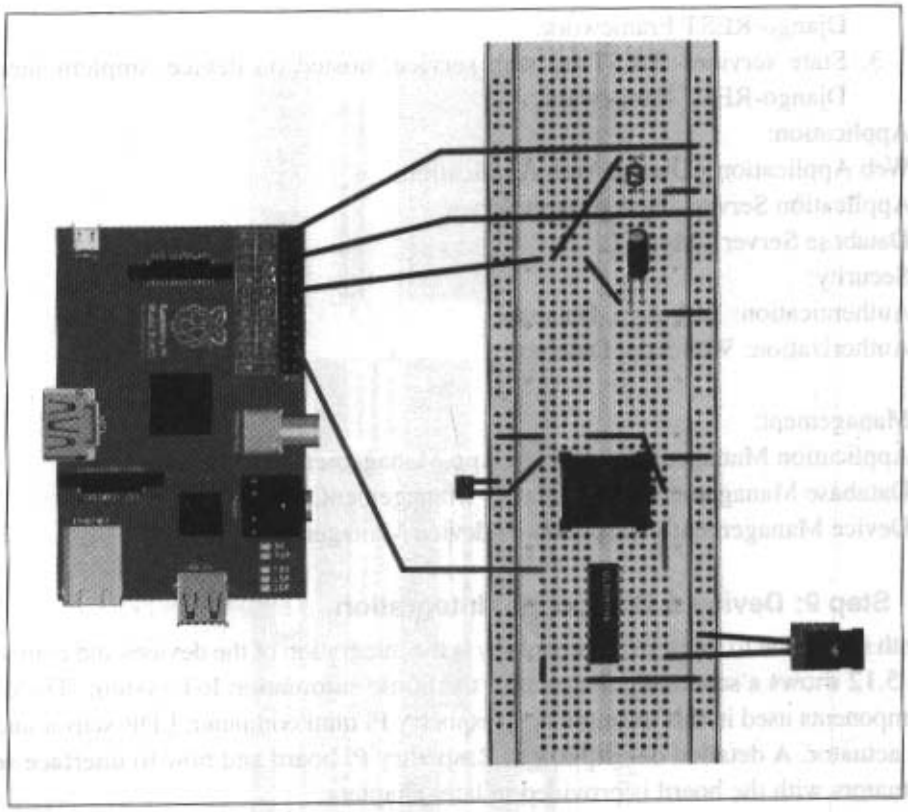


Figure 5.12: Schematic diagram of the home automation IoT system showing the device, sensor and actuator integrated

in an area using multiple end nodes. The end nodes send the data to the cloud where the data is aggregated and analyzed.

Figure 5.14 shows the process specification for the weather monitoring system. The process specification shows that the sensors are read after fixed intervals and the sensor measurements are stored.

Figure 5.15 shows the domain model for the weather monitoring system. In this domain model the physical entity is the environment which is being monitored. There is a virtual entity for the environment. Devices include temperature sensor, pressure sensor, humidity sensor, light sensor and single-board mini computer. Resources are software components which can be either on-device or network-resources. Services include the controller service that monitors the temperature, pressure, humidity and light and sends the readings to the

Figure 5.13: Home automation web application screenshot

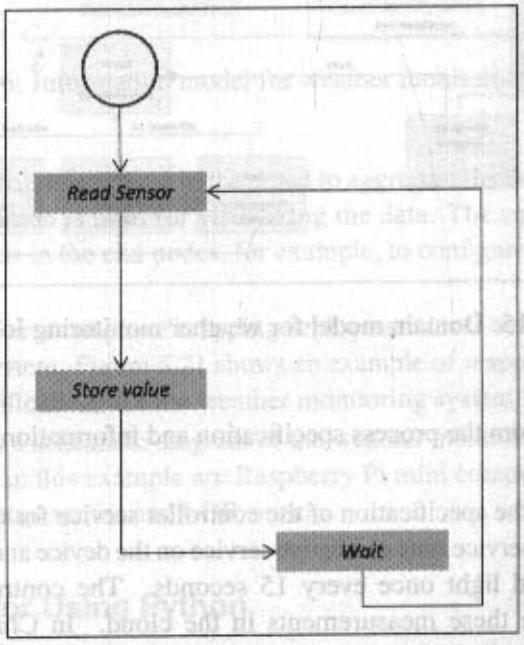


Figure 5.14: Process specification for weather monitoring IoT system

cloud.

Figure 5.16 shows the information model for the weather monitoring system. In this example, there is one virtual entity for the environment being sensed. The virtual entity has attributes - temperature, pressure, humidity and light. Figure 5.17 shows an example of

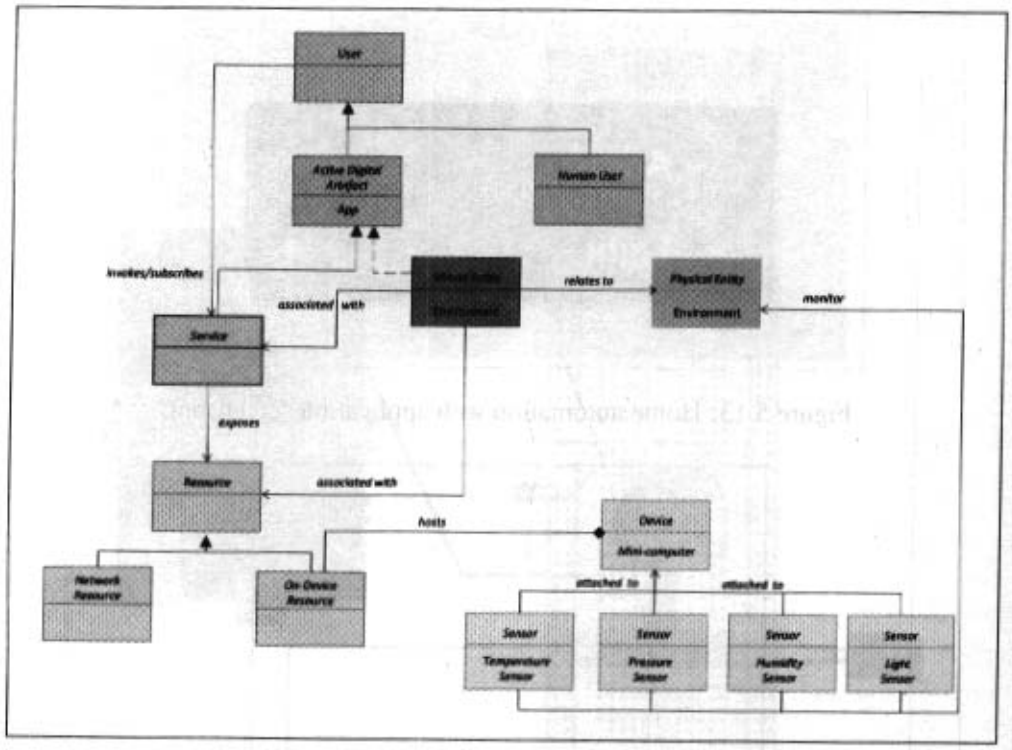


Figure 5.15: Domain model for weather monitoring IoT system

deriving the services from the process specification and information model for the weather monitoring system.

Figure 5.18 shows the specification of the controller service for the weather monitoring system. The controller service runs as a native service on the device and monitors temperature, pressure, humidity and light once every 15 seconds. The controller service calls the REST service to store these measurements in the cloud. In Chapter-8 we describe a Platform-as-a-Service called Xively that can be used for creating solutions for Internet of Things. An implementation of a controller service that calls the Xively REST API to store data in Xively cloud is described in Chapter-9.

Figure 5.19 shows the deployment design for the system. The system consists of multiple nodes placed in different locations for monitoring temperature, humidity and pressure in an area. The end nodes are equipped with various sensors (such as temperature, pressure, humidity and light). The end nodes send the data to the cloud and the data is stored in a cloud

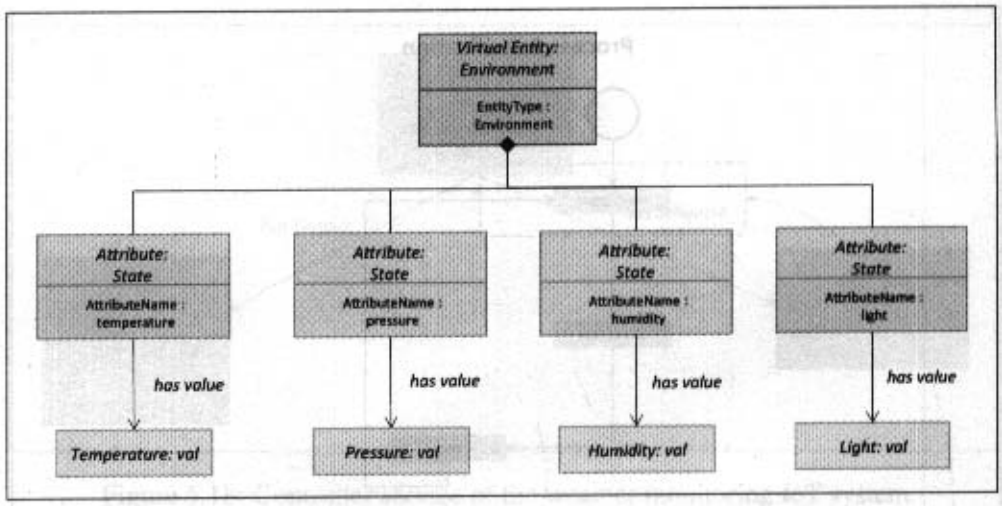


Figure 5.16: Information model for weather monitoring IoT system

database. The analysis of data is done in the cloud to aggregate the data and make predictions. A cloud-based application is used for visualizing the data. The centralized controller can send control commands to the end nodes, for example, to configure the monitoring interval on the end nodes.

Figure 5.20 shows an example of mapping deployment level to functional groups for the weather monitoring system. Figure 5.21 shows an example of mapping functional groups to operational view specifications for the weather monitoring system.

Figure 5.22 shows a schematic diagram of the weather monitoring system. The devices and components used in this example are Raspberry Pi mini computer, temperature sensor, humidity sensor, pressure sensor and LDR sensor.

5.4 Motivation for Using Python

This book uses the Python language for all the examples, though the basic principles apply to other high level languages. In this section we explain the motivation for using Python for developing IoT systems. Python is a minimalistic language with English-like keywords and fewer syntactical constructions as compared to other languages. This makes Python easier to learn and understand. Moreover, Python code is compact as compared to other languages. Python is an interpreted language and does not require an explicit compilation step. The Python interpreter converts the Python code to the intermediate byte code, specific

PART II: UNDERSTANDING THE NUTS AND BOLTS OF IOT HARDWARE, SOFTWARE, AND MIDDLEWARE

Chapter 5: Sensors and Actuators in IoT

Chapter 6: Open Hardware in IoT

Chapter 7: IoT Middleware

Chapter 8: IoT Software Platforms

Chapter 9: Prototyping IoT Applications

CHAPTER 5

Sensors and Actuators in IoT

"Accelerating technology innovations such as ubiquitous sensors, cheap computing power, and 5G networks will open entirely new opportunities and challenges."

— Cathy Engelbert

OBJECTIVES

- understand perception layer of Internet of Things (IoT) architecture
- gain knowledge of various sensors for IoT applications
- conceptualize functions of various commonly used sensors
- understand various actuators for IoT applications
- gain knowledge of environmental sensors

OUTCOMES

- comprehend the concept and importance of perception layer in IoT
- understand the role of various sensors in IoT applications
- acquire knowledge about the functionality of various commonly used sensors
- gain knowledge of various actuators and environmental sensors for IoT applications

REVISION

In Chapter 4, network protocols and standards for IoT applications are discussed. Network is the backbone of an IoT application. Various protocols suitable for IoT applications are designed and developed by considering various constraints such as low power devices and low computing power.

This chapter also presents edge nodes and their network for sensing physical environment as well as actuator devices through which controlling action is carried out.

5.1 INTRODUCTION

Though the concept of Internet of Things (IoT) was introduced in the early 1990s, as the cost of IoT devices is reducing, the practical and economical implementation of IoT applications is possible now. The end device of any IoT system is either a sensing device (sensor) on one end or an actuator at the other end that triggers control action as an output. Hence, it is also known as the edge device and on some occasions can be both a sensor and an actuator.

The edge device/node is connected to the network by wireless or wired network. Data integrity is a major issue as the data is collected from signals from many sensors for intelligent decision making. The device at edge is capable of sensing, pre-processing, measuring, interpreting, and connecting to the cloud through the gateway for providing intelligence. Large IoT applications need a large array of edge nodes to make an array of network of edge devices. For the purpose of such a sensor network, each edge device must be identified with a unique identifier in order to communicate with them as well as for being recognized effectively.

Local intelligence is obtained by edge nodes with processing unit such as one or more microcontroller, hybrid circuit. Edge nodes with a processing unit are larger in size as compared to end nodes. Some examples of edge devices (nodes) – sensors are: light, sound, pressure, temperature, humidity, vibrations, motion, gas, Radio Frequency IDentification (RFID), Near Field Communication (NFC), ultrasonics, altitude, colour fluid, flow meter, cameras, etc. Typical examples of actuators are: switch, relays, PLCs, motors, light, sound, etc.

In IoT systems, a Sensor is an important hardware component and as sensors' prices have decreased over the years, development of IoT applications has gained momentum. Sensors are also described as physical objects that interpret the environmental changes and convert those changes to an electrical signal output. Real-time environmental information is sensed by the sensor devices and this data is converted further into machine-understandable information. Sensors must be uniquely identified with an IP address to identify them easily over a large network of sensors. Interfacing of a sensor with the microcontroller board is relatively easy and the entire unit of both things together makes a smart sensor. Sensors need to be active in nature as to collect real-time data. The sensor nodes are autonomous or made to work by the user depending on the need of a specific application (user controlled). The growth as well as cost effective smart sensors are giving a boost for IoT applications in various domains.

5.1.1 Sensors for Different IoT Applications

Sensors play an important role in almost all IoT applications. Often, IoT applications need things that can react and collect real-time data, using some hardware such as communication modules, to communicate and process the sensed data and some nodes for data collection, sharing, and storage.

The most important aspect of various IoT application domains is the smart sensor (with microcontroller boards). Figure 5.1 shows sensors used for IoT applications such as transportation, industrial, agricultural or environmental home automation, retail and logistics, health care. The domain-wise sensors are listed in Fig. 5.2, for real-time access of data as described below:

Healthcare Various healthcare applications use sensors for developing IoT applications such as smart wearables for monitoring blood pressure and sugar level and also for monitoring conditions of patients inside the hospital. Light sensors, biosensors, motion sensors, and acoustic sensors are commonly used in healthcare applications.

Building automation Building automation applications of IoT are smart lighting, smart waste management, smart parking, energy and water management, etc. A sensor is selected based on specific requirement of the application. Some sensor examples are ultrasonic sensor for smart parking and PIR sensor for smart waste management, and temperature sensor for room temperature monitoring.

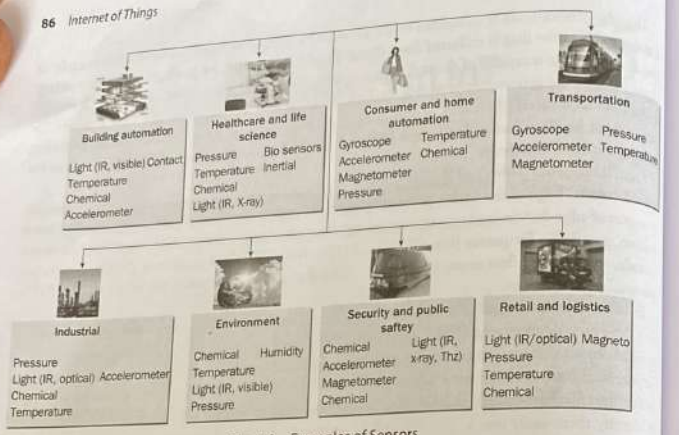


Fig. 5.1 Examples of Sensors

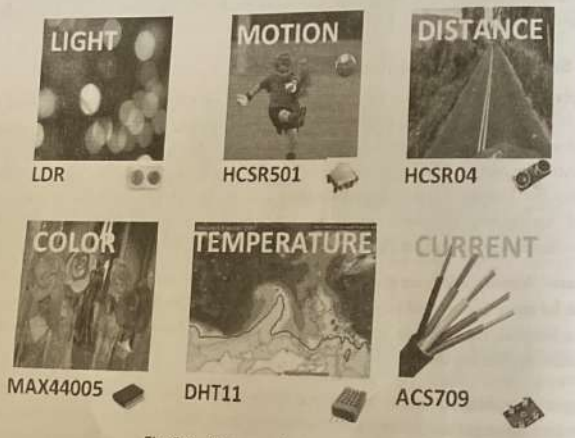


Fig. 5.2 Sensors and IoT Applications
(Note: HCSR501,DT11, etc., are products available in the market)

Agriculture Smart agriculture applications of IoT are soil moisture level detection for best utilization and management of water resources, crop farming management to maximize agricultural production, management of green house, controlling micro-climate, etc. Some commonly used sensors for smart agriculture are soil moisture, PH sensor, solar radiation, soil nutrients, leaf wetness, and weather sensors such as temperature, relative humidity, wind speed/direction, and barometric pressure.

Many other IoT application domains (as shown in Fig. 5.2) are transportation, industry, environment, security, etc. some of which are covered in more details in Chapter 17.

5.2 PERCEPTION LAYER OF IOT

The basic IoT architecture is of three layers namely, perception, network, and application. In the IoT architecture, physical layer is also known as perception layer which consists of sensor devices for sensing and collecting information from the environment. Physical parameters are sensed by these sensors. The task of perception layer is to identify other smart objects in the environment. The discussion in this chapter is mainly focused on the perception layer.

5.2.1 Active Sensors vs Passive Sensors

Sensors detect and respond to the physical environment around them. The sensor which requires external source of energy for its operation is called as active sensor while passive sensor does not need external energy source for its operation. The external energy input for an active sensor can be mechanical, electrical, or any other type. Active sensors are widely used in networking and manufacturing applications and many data centres for the purpose of anomalies detection.

Other sensor-based technologies are GPS, LiDAR, scanning electron microscope, x-ray, sonar, seismic, infrared, etc. Seismic as well as infrared light sensors are both active and passive. Remotely sensed data is used for many applications such as resource exploration, cartography, atmospheric and chemical measurements. In IoT applications, remote sensing is needed so that almost every physical or logical entity can be equipped with a unique identifier and automatic data transfer over a network is possible.

Active sensor is a device with a transmitter, which sends a signal that bounces back from the target. This reflected data is gathered by the sensor. For making observations and measurements from a large distance, remote sensing technology is used in active as well as passive forms. Sensors are also used in hostile environments with extreme conditions and at inaccessible places. Various sensors are to be mounted at different places such as satellite, airplane, boat, submarine, UAV, or a building top based on what is being sensed.

Sensors such as accelerometers, magnetometer, barometer, are passive sensors because they are able to extract a measurement without interacting with the surrounding environment by sending some form of a signal.

5.3 UNDERSTANDING SOME COMMONLY USED SENSORS

In the following subsection, some commonly used sensors are explained.

5.3.1 Light Sensors

Light sensors that convert light energy (photons) into electronic signals are also known as photoelectric devices or photo sensors. The light sensor is a passive device. In a photo sensor, a beam of light is used to detect presence or absence of an object. Visible or infrared light beam is emitted from its light-emitting element. For detecting the light beam reflected from a target, a reflective-type photoelectric sensor is used. The emitted beam of light from the light-emitting element is received by the light receiving element. In a single housing, both the light-emitting and light-receiving elements are installed.

Some common type of light intensity sensors are photoresistors, phototransistors, and photodiodes. When this interfaced circuit, is balanced with potentiometer, the change in light intensity is interpreted as change in voltage. Bright light allows a larger amount of current flow. These sensors are widely used for measuring light intensity as they are simple, cheap, and reliable.

5.3.2 Accelerometers

There are two common types of accelerometers namely piezoelectric accelerometers and seismic mass type accelerometers (Fig. 5.3). For high frequency applications, the piezoelectric accelerometer is preferred due to its compact size. The seismic mass type accelerometer is based on the relative motion between a mass and the supporting, and is used for low- to medium-frequency applications.

Accelerometer is an electromechanical device. The forces are measured by an accelerometer. These forces may be static such as the constant force of gravity pulling some object or they could be dynamic which is caused by moving or vibrating the accelerometer. An accelerometer produces analog or digital outputs. Analog accelerometer's output is a continuous voltage that is proportional to acceleration with some specific calibration. Digital accelerometers commonly use pulse width modulation (PWM) for output. In some accelerometers, piezoelectric effect is used in which microscopic crystal structures get stressed by accelerative forces, which cause a voltage to be generated. Another way is by sensing changes in capacitance. Two microstructures side by side have a certain capacitance between them. If an accelerative force moves one of the structures, then the capacitance will change. Using accelerometer this change in capacitance is measured to convert from capacitance to voltage.

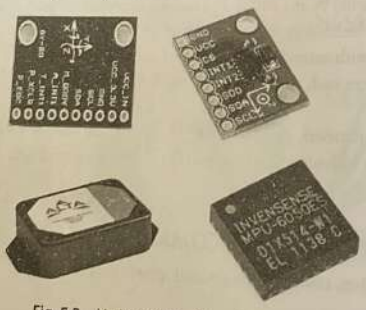


Fig. 5.3 Various Kind of Accelerometers

5.3.3 Gyroscopes

A gyroscope consists of a rotor (rotating disk) mounted at the centre of a larger spinning axis. Angular velocity (change in rotational angle per unit time and expressed in degrees per second) is sensed by gyrosensors. A gyroscope device uses Earth's gravity to determine orientation based on the principles of angular momentum.

There are three basic types of gyroscopes, namely rotary, vibrating structure and optical. Various kinds of gyroscopes can measure rotational velocity in three directions and also tilt and lateral orientation. Gyroscopes are generally implemented with a three-axis accelerometer to provide complete six degrees-of-freedom, which is useful for motion tracking systems that are used in aircrafts, autonomous vehicles, robots, etc.

5.3.4 Magnetometer

Magnetometer measures the strength, direction, or relative change of a magnetic field. Some applications of magnetometers include mobile devices using it as a compass, detecting metal objects under water (for example submarines), various surveys for minerals and oil exploration, locating underground buried objects, equipments that need to maintain precise direction, etc.

5.3.5 Global Positioning System

Satellite-based Global Positioning System (GPS) uses satellites and ground stations to decide and compute a position on earth. GPS receiver collects data from at least four satellites for accurate GPS computation. GPS receivers are used in applications such as smartphones, military applications, fleet management system, etc. for finding or tracking location of a physical entity.

5.3.5.1 Working Principle of GPS

Global positioning system on the satellites transmits information signal to receiver over radio frequency range 1.1–1.5 GHz. With this received information, a ground station or GPS module can compute its own position and time. Accurate location of a system is computed by a GPS receiver on the system, using constellation of satellites and ground stations (refer to Fig. 5.4).

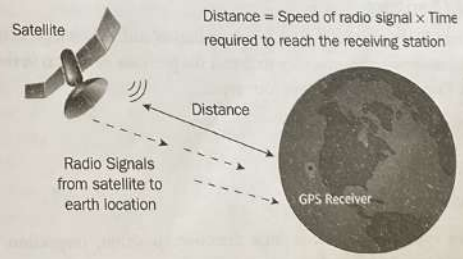


Fig. 5.4 GPS Communication

5.3.5.2 GPS Distance Calculation

Global positioning system calculates its distance from satellites using the time required for the signal to reach to the receiver. To calculate the distance both, satellite and GPS receiver (refer to Box 5.1) generate a synchronized pseudocode signal. The pseudocode signal received from the GPS receiver is transmitted by the satellite. These two signals are compared and the difference between the signals is computed. The receiver computes the distance from the information signals received from three or more satellites and then it can calculate its location by using Trilateration method.

BOX 5.1: GPS RECEIVER

Global positioning system receiver module gives output in standard National Marine Electronics Association (NMEA) string format. Output is received serially on Tx pin with default baud rate of 9600.

GPS receiver module

VCC: Power supply 3.3–6 V

GND: Ground

TX: Information about location, time, etc. is transmitted by TX

RX: Receive data serially

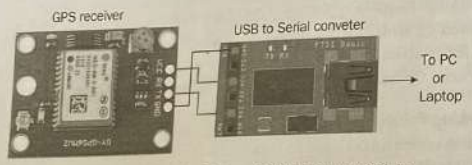


Fig. 5.5 Global Position System (GPS) Receiver Module

5.3.6 Proximity Sensors

A proximity sensor emits a beam of electromagnetic radiation and any change in the field results in a return signal. These sensors have the capability to detect the presence of objects in their vicinity without any physical contact. Proximity sensors are of four types:

- capacitive proximity sensor
- inductive proximity sensor
- photoelectric or optoelectronic sensor
- ultrasonic proximity sensor

Some of the common application areas include detection, position, inspection, and counting on automated machines and in manufacturing systems.

Any change in the dielectric medium surrounding the active face is responded by the capacitive sensor mostly without making physical contact. They can be configured to sense almost any substance. Capacitive sensors can also sense any object beyond a layer of thin carton, glass, or plastic.

Inductive proximity sensors can sense and respond to ferrous and non-ferrous metal objects also. These sensors can detect metal covered in a layer of non-metallic material. Inductive proximity sensors are made of a coil wound around a soft iron core. When a ferrous object is nearby, the change in inductance of the sensor takes place. The change in inductance is then converted to voltage to operate a switch. This concept is used in mobile phones to detect accidental touch when held close to the ear.

5.3.7 Radio Frequency Identification (RFID)

RFID identification technology uses RFID tag (a small chip with an antenna) to embed data in it with an RFID reader. This technology is similar to the bar code technology. Unlike a traditional bar code, line of sight communication is not required between the tag and the reader and RFID tag can be identified from a distance. Radio waves are used to transmit the data stored in the tag. The range of RFID can be varied with the frequency variation and can be extended up to hundreds of meters.

Two types of RFID technologies are used often, namely near and far. In near RFID reader, there is a coil through which alternating current (AC) is passed and a magnetic field is generated. The ambient change in the magnetic field is registered by the tag with a small coil, which generates a potential difference. This voltage is then coupled with a capacitor, that powers up the tag chip. The charge is accumulated by producing a small magnetic field which encodes the signal to be transmitted and it is read by the RFID reader.

Radio frequency identification tags are also classified as active and passive. An external power/energy source is associated with active tags and passive tags do not need any such external power source. Passive can sense the electromagnetic waves emitted by the reader without any external power source and are hence a cheaper option with a long lifetime. The raw data collected from the RFID tags is then pre-processed to make it suitable for IoT applications. Many available user level tools can process the data collected by particular RFID readers and the raw data from RFID tags is then processed and managed. This processed data can be further analysed and inferences are drawn from this analysis for some control action.

A dipole antenna is present in far RFID reader, which is used for propagating EM waves and for sensing the alternating potential difference. The object to be tracked is attached with the RFID tag and the reader detects its presence when the object passes across it. Thus, object motion can be tracked with RFID technology for searching smart things.

Authorized object with RFID tag is useful for access control. For example, small RFID chips are attached in front of vehicles and when the car reaches a (for example a toll plaza) on which an RFID reader is present, the tag data is read for car authorization. If authenticated successfully, it opens automatically. RFID technology is being used in many other applications such as object tracking, supply chain management, identity authentication, and access control.

5.4 ENVIRONMENTAL SENSORS

Various parameters related to the understanding of the physical environment (temperature, humidity, air pressure, etc) are sensed by environmental sensors. For example, to decode the air quality of a particular location, the presence of gases and other particles in the air can be measured using various sensors. Below is a description of a few environmental sensors.

5.4.1 Temperature Sensors

The amount of heat or coldness that is generated by an object is measured by temperature sensors or systems. Sensing or detecting any physical change to temperature is done either by an analog or digital output.

There are two basic physical types of temperature sensors namely contact temperature sensors and non-contact temperature sensors. Contact temperature sensors require physical contact with the object to be sensed and use conduction principle of heat to monitor changes in temperature. Non-contact temperature sensors use convection and radiation heat principles to monitor changes in temperature.

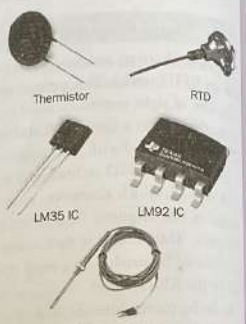


Fig. 5.6 Various Temperature Sensors

The most common devices used to measure temperature are thermistors, Resistance Temperature Detectors (RTDs), infrared, and thermocouples (refer to Fig. 5.6). Thermistors are semiconductor devices which change the resistance with temperature change. Thermistors are useful for measuring temperature in a limited range of up to 100°C with high sensitivity. The RTDs are based on principle that the resistance of a metal changes with temperature. Radiation heat is used by infrared type sensors to sense the temperature from a distance. These non-contact sensors are used to sense and generate a thermal map of a surface.

5.4.2 Pressure Sensors

In many applications, pressure sensors are used for control and monitoring. Pressure sensors are also useful to indirectly measure many other variables such as water level, fluid/gas flow, altitude and speed.

Some pressure sensors work as pressure switches to turn on or off a particular pressure value. For example, a water pump is controlled by a pressure switch using which the water is released from the system to start the pump. Very high pressure changes occurring in dynamic mode are captured by some specially designed pressure sensors, for example, a pressure sensor measuring combustion pressure in a gas turbine. These sensors are manufactured using piezoelectric materials such as quartz. Further applications include altitude sensing (aircraft, rockets, satellites, weather balloons, etc), flow sensing (flow/ depth measurement), and leak testing.

5.4.3 Humidity Sensors

The term 'humidity' refers to the presence of water vapour in air or other gases. Presence of water vapour in air is measured by the humidity sensor (refer to Fig. 5.7). Measurement of water vapour content in the air is important to understand various physical, chemical, biological processes and also has profound affect on certain manufacturing processes.

There are three main types of humidity sensors, namely thermal, capacitive, and resistive. These three types of sensors can monitor minute changes in the atmosphere to calculate the humidity in the air. Based on the humidity of the surrounding air, two thermal sensors conduct electricity. One sensor is encased in dry nitrogen while the other is open in ambient air. Relative humidity is measured by placing a thin strip of metal oxide between two electrodes. Electrical capacity of the metal oxide changes with the relative humidity of atmosphere. Resistive humidity sensors work on ions in salts to measure the electrical impedance of atoms that changes with change in humidity.

Fig. 5.7 HPP801A031 Humidity Sensor

Application areas of humidity sensing include, building and construction, health monitoring and medical applications, fuels, aerospace, indoor, etc.

5.4.3.1 *Relative Humidity*

The Relative Humidity Sensor (RHS) is used for monitoring Agriculture, weather studies, discharges, etc. RHS is encased with an integrated circuit which is used to monitor relative humidity over the range 0–95% (±5%).

5.4.4 Wind Speed and Wind Direction Sensors

Wind vanes measure wind direction and wind speed is measured by anemometers. The manufacturer Campbell Scientific offers variety of anemometer designs such as propeller, cup, ultrasonic, lidar, and sonic.

5.4.5 Soil Moisture Sensors

The water content in the soil is measured by soil moisture sensors indirectly by measuring some properties of soil.

Some soil sensors measure moisture in soil as water potential. Such sensors include tensiometers and gypsum blocks and this property is called as soil water potential. Soil moisture affects reflected microwave radiation. This measure is useful for drawing inferences in remote sensing hydrology and agricultural applications.

5.4.6 Leaf Sensors

Leaf sensors are commonly known as phytometric devices that are mounted on randomly selected leaves on the plant. Generally, the sensor records measurements at 5-min intervals throughout the day.

The sensors determine from these records whether the plant is with enough water content or is water-stressed.

Farmers get correct information and can avoid over-watering or under-watering a crop. Even unnecessary water wastage can be avoided using such sensors, especially in areas of the world where irrigation water-shortage problem is faced.

5.4.7 Lysimeter

A lysimeter is a device which measures the amount of actual evapotranspiration which is released by plants. Record of the amount of precipitation that an area receives, the amount lost through the soil, and the amount of water lost to evapotranspiration can be calculated. Two types of lysimeters are available, weighing and non-weighing.

5.4.8 Rain Gauge

The amount of rainfall of any area in a given time period is measured by the rain gauge. No need to rely on local weather reports if such data is collected using rain gauge. It is like a transparent cylinder with markings on it. The rainfall is measured in inches or millimetres.

Based on the collected information, intelligent decision making is possible about the land and crops.

5.4.9 Chemical Sensors

Chemical and biochemical substances are detected by chemical sensors. Monitoring pollution level in smart cities, testing food, agricultural products, and checking food quality in smart kitchens, medical, automotive, detection of harmful gases indoor such as carbon monoxide are some of the applications of chemical sensors.

These sensors are made of a transducer and a recognition element. Recently, technologies such as electronic tongue (e-tongue) and electronic nose (e-nose) are being developed to sense taste and odour, respectively, and are being integrated with innovative IoT products. Pattern recognition software analyses to identify the stimulus.

5.5 MEDICAL SENSORS

The IoT technology is very useful for healthcare applications. A patient's health can be monitored remotely when they are not in hospital or when they are alone. Many medical parameters in the human body can be measured and monitored using medical sensors for providing real-time feedback to the doctor, patient, or relatives. Currently, a plethora of wearable sensing devices are available in the market such as wristbands, smart watches, monitoring patches, etc. Various parameters such as blood pressure, blood glucose levels, body temperature,etc, are sensed by these wearable sensing devices.

Monitoring patches that are attached on the skin are devices which are stretchable, disposable, and cheap. Such patches when worn by a patient are useful to monitor health parameters more frequently. Companies such as Apple, Samsung, and Sony have brought IoT devices with innovative

features such as smart watches and fitness trackers in the market. These devices help to integrate contextual information (such as brisk walking to heart rate) with health parameters to enable correct inferences regarding an underlying condition in a person.

5.5.1 Heartbeat Sensor

Heart rate is defined as the sound of the valves in a person's heart while contracting or expanding to pump the blood from one region to another. Heartbeat rate is the number of times the heart beats per minute (BPM). There are two methods to measure a heartbeat:

Manual way Manually, heartbeat can be measured by checking a person's pulse in the wrist (radial pulse) and the neck (carotid pulse).

Sensor use for heartbeat measurement Heartbeat can be measured using the optical power variation as light is scattered during its path through the blood when there is change in the heartbeat.

5.5.2 Pulse Sensor

Pulse sensor captures the heart beat or heart rate of a person. The flat side of the sensor is placed on top of the vein and a slight pressure is applied on top of it using a fastening band clip to attain this pressure.

5.5.2.1 Applications

Pulse sensors are used in the following devices:

- Health bands
- Advanced gaming consoles
- Sleep tracking
- Remote patient monitoring/alarm system
- Anxiety monitoring

5.5.3 Blood Glucose Level Sensor

The concentration of blood glucose is measured and monitored using blood glucose level sensor. This is a monitoring device for diabetes patients which can be easily used at home.

Continuous glucose monitoring system The finger-prick test is inconvenient and the chance of contaminating a blood sample is heightened. Alternatively, blood sugar levels can be continuously monitored by a transmitter implanted in the body. The approximate concentration of glucose in the blood is measured by the sensor device, making it simple. The patient is alerted frequently by an alarm system about the fluctuations in the blood sugar level.

5.5.4 Blood Pressure Sensor

For measuring human blood pressure, the blood pressure sensor is used which is a non-invasive sensor. The systolic, diastolic, and mean arterial pressure is measured by the oscillometric technique. Even the

pulse rate is also measured by such a device. The common risk factor for heart attacks is high blood pressure. Hence, diagnosing and monitoring high blood pressure is important.

The new breed of monitoring devices to be worn around wrist and the index finger are similar to traditional cuff devices but simple to use. Such devices are useful to monitor hypertension as well as sleep apnea, which cause breathing to stop many times throughout the night.

5.5.5 Body Temperature Sensor

Body temperature is measured by such sensor devices. The reason is that a number of diseases are accompanied by characteristic changes in body temperature. By measuring body temperature certain diseases can be monitored.

5.6 FLOW AND FLUID MEASURING SENSORS

This section discusses level sensors, stream gauge, and tide guage.

5.6.1 Level Sensors

A level sensor detects the level of a liquid (or semi-solid materials such as slurries) in either an enclosed space or free-flowing (e.g. river).

The level sensing measurement can either be point measurements made at specific locations or continuous, where a range of measurements are recorded. Point-level sensors indicate whether the substance is above or below the sensing level to detect the level above or below the decided threshold.

The selection of a particular method such as acoustic, vibration, mechanical, etc., for level monitoring depends on the application and the domain (industrial, commercial, hydrology, etc.) in which it is used.

5.6.2 Stream Gauge

Terrestrial bodies of water are tested and monitored by stream gauging station also known as stream gauge or gauging station by hydrologists. These network of stations along a river basin are usually shown on a map. The data from these stations are made accessible remotely and are used in various hydrological models.

5.6.3 Tide Gauge

A tide gauge also known as a mareograph is a device for measuring the change in sea level relative to a vertical datum. Tide gauges enable to determine the mean sea level by obtaining hourly measurements at geographically distributed locations for 19 years and averaging them. This mean sea level is usually used as a vertical datum against which height measurements are made. Also, tide gauges are useful in measuring the tide heights during Tsunamis and other ocean related events used in climate modeling.

5.7 RANGE AND MOTION CAPTURE SENSORS

Motion sensors are primarily used to detect movement of objects. Whereas range sensors enable to measure distance of an object. Some sensors combine both these functionality and can do both motion and range sensing.

A Passive Infrared (PIR) sensor (see Fig. 5.8) can sense motion upto 12 metres. It is a pyroelectric device capable of detecting naturally emitted infrared radiation from objects (temperature above absolute zero). The field of view of the sensor is generally around 110° and use a Fresnel lens (Fig. 5.8) to achieve such wide breadth of sensing.

PIR sensors have a wide variety of applications such as

- Motion detection of Humans in indoor spaces such as homes, malls, hotels, etc.
- Vehicles movement detection in parking lots and basements
- Smart automated lighting, AC (indoors), street lights to conserve electricity
- Capture movement of animals in remote areas by integrating with a camera module

Fig. 5.8 PIR Motion Sensor

5.7.1 Distance Sensors

Distance sensors are of two types, namely ultrasonic and laser, which are explained below:

5.7.1.1 Ultrasonic Distance Sensors

Ultrasonic sensors measure distance from ojbects with high accuracy. These sensors come in various sizes to fit into the measuring environment such as small containers, openings, etc. The range of Ultrasonic sensors varies between different manufacturers. The popular low-cost version HC-SR04 distance sensor has a range of 2cm to 400 cm with a measure angle of 15 degrees and accuracy of approximately 3 mm. Some commercially available sensors have a high range of upto 21 meters.

Applications of Ultrasonic sensors include object/people detection, liquid level measurements, monitoring garbage level in trash cans, industrial production lines (objects on conveyor belts).

5.7.1.2 Laser Distance Sensors

Laser sensors are a highly efficient way of object detection as well as measuring distance from an object even under varying lighting conditions and material characteristics of the object. These sensors have many applications in the areas of building and construction, oil and gas, mining, transportation, electronics, etc. In general, it can be used in situations requiring precise contactless measurements of length, width, height, diameter, etc. of objects.

5.7.2 Touch Sensor

A touch sensor is a device that captures and records physical touch on an and sends it to a processing unit or software that processes these records.

Touch sensors are very commonly used on displays of smartphones and tablets. Two types of touch sensors/screens are normally used namely, capacitive and resistive.

5.7.2.1 Capacitive Touch Sensors

A capacitive touch sensor works on the principle of capacitive coupling to sense anything whose dielectric (non-conductors of electricity) is different from the surrounding air. In a touch sensor,

the electrical charge conductor is a human body, when the finger comes in contact with the sensor surface. The exact coordinates (location) is sensed by the change in the electrostatic field around that region (see Fig. 5.9).

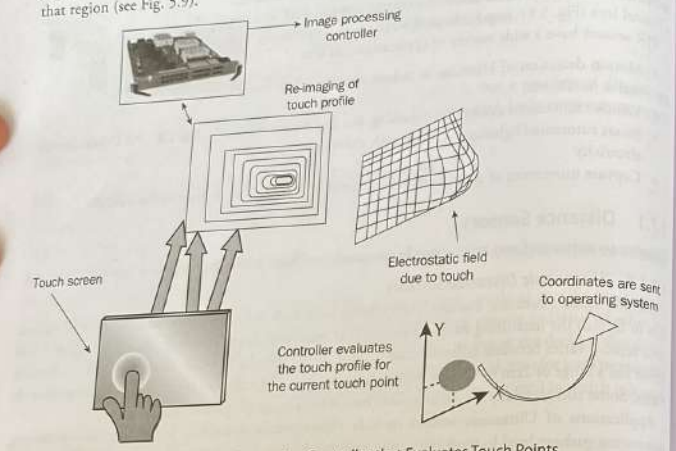


Fig. 5.9 The Image Processing Controller that Evaluates Touch Points

5.7.2.2 *Resistive Touch Sensors*

Resistive touch sensors are usually of two varieties. First type is based on glass (insulating material) as the base support material with film based electrically conductive and resistive layers. The second type is similar to the above, except that the support material is made of polycarbonate. Both the support material and the film are coated with ITO (Indium-Tin-Oxide), which is transparent to light and has high electrical conductivity. An air gap is introduced between these film and support material (glass or polycarbonate) (see Fig. 5.10).

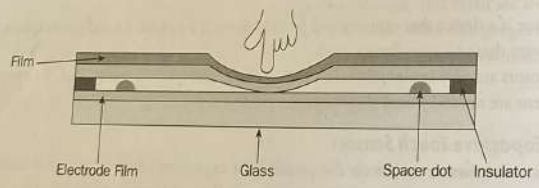


Fig. 5.10 Resistive Touchscreen and its Layers

5.8 ACTUATORS

An actuator is a device that converts the effective change in the environment sensed in the form of electrical energy into some other form of energy. Based on their operation, these are classified into electrical, hydraulic, and pneumatic actuators. Lights, displays, motors, heating or cooling elements, and speakers are some of the examples of electrical actuators. Another example is a smart home system that uses many sensors and actuators.

Pressure of compressed air is used in pneumatic actuators and hydraulic power is used in hydraulic actuators. Actuators are used in home applications such as locking and unlocking doors, switching lights on and off, etc. In this section, some examples of actuators are presented which are useful in IoT applications.

5.8.1 Servo Motor

To push or rotate an object with precision, a servo motor is used. It is an electrical device which can rotate an object at some specific angles or distance. It is a simple motor which uses servo mechanism for its operation. If motor operates on DC power, then it is known as DC servo motor and if it is an AC powered motor, then it is known as AC servo motor. Servo motor is available in small and light weight packages with high torque. Hence, such motors can be used in many applications such as Robotics, RC helicopters and planes, machine toy cars, etc.

5.8.2 Stepper Motor

A stepper motor is an electromechanical device which converts electrical power into mechanical power. It is a brushless, synchronous electric motor and divides its full rotation into an expansive number of steps. If the motor is carefully sized to the application, the motor's position can be controlled accurately without any feedback mechanism.

There are three main types of stepper motors:

- Permanent magnet stepper
- Hybrid synchronous stepper
- Variable reluctance stepper

Stepper motors are used in various applications such as:

- Actuators in fluid pumps, medical scanners, samplers, blood analysis machinery, respirators, etc.
- Focus and zoom functions in digital cameras
- Machine tooling equipment, automotive gauges, and surveillance systems for security industry

5.8.3 DC Motor

Electric machines are used for converting energy. Electrical energy is converted by motors to produce mechanical energy. Many day-to-day applications use electric motors. There are two main classes of electric motors:

- Direct Current (DC) motor
- Alternating Current (AC) motor

A direct current power motor is known as DC motor. Field windings of DC motor are used to provide the magnetic flux and the armature of DC motor acts as the conductor. Its working principle is that when a current carrying conductor is placed in a magnetic field, it experiences a force which causes rotation with respect to its original position.

DC motor mainly consists of two parts. The rotating part is called the rotor and the stationary part is called as the stator. The rotor rotates with respect to the stator.

5.8.4 Linear Actuators

The type of actuator is useful when motion in a straight line (linear motion) between two points is required. These could be mechanical or electro-mechanical (also known has electric linear actuators) devices. The rotation of the electric motor is converted to linear motion by using gears and lead screw in the electric linear actuators.

Applications include valves, automatic windows/doors, agriculture implements, robotics, machine design, material management, industrial equipment, etc.

5.8.5 Solenoid

A coiled wire around a cylinder is called as a solenoid. When current runs through a solenoid, a strong magnet is generated because the magnetic field is concentrated inside the coil. Unlike regular magnets, an electromagnet is particularly useful as it can be switched on and off, and its strength increases by increasing the amount of current flowing through it. This phenomenon is useful in many applications.

5.8.5.1 Electromagnet Solenoid and its Use

Solenoids have a wide range of applications such as valves (air compressors, metres), pistons, home automation devices (door bells, auto locks, AC, heating equipment), automobiles (engine starter), drinks, coffee dispensers, sprinkler system, etc.

5.8.6 Relay

The relay device receives input signal from one side and controls the switching operation on the other side.

Many types of relays are present. A commonly used relay is made of electromagnets and operates as a switch. Relay is a switch that controls the circuit electromechanically. A high-powered circuit is controlled by a relay switch with a low power signal. Generally, a DC signal is used to control a circuit that drives high voltage appliances such as smart home appliances that operate on AC power supply.

5.8.6.1 Different Mechanical Parts of a Relay

- Electromagnet
- Movable armature
- Contacts
- Spring (optional)
- Yoke

Figure 5.11 shows how a relay is internally connected.

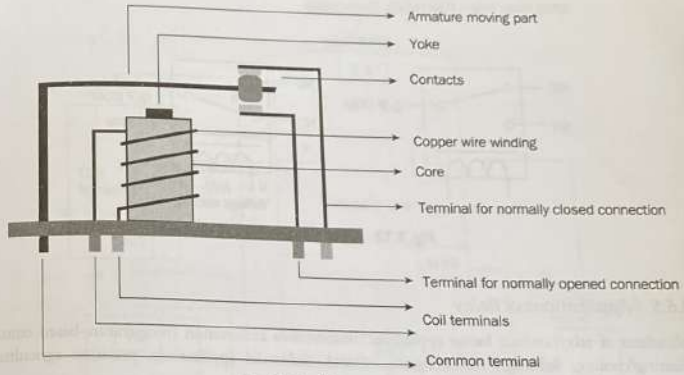


Fig. 5.11 Electromagnetic Relay

When the relay is on, the armature moves and connects the normally opened contact pin of the relay. The circuit is closed and the current flows through the coil and comes back to the original position after it is de-energized.

The circuit representation of the relay is as shown in Fig. 5.12.

5.8.6.2 Different Types of Relay

There are many types of relay other than the electromagnetic relay which works on different principles. The classification of relay is as follows:

5.8.6.3 Based on the Principle of Operation

- Solid state relay
- Hybrid relay
- Electrothermal relay
- Electromechanical relay

5.8.6.4 Based on the Polarity

- Polarized relay
- Non-polarized relay

NC: Normally closed
NO: Normally opened
SPDT relay: Single Pole Double Throw replay

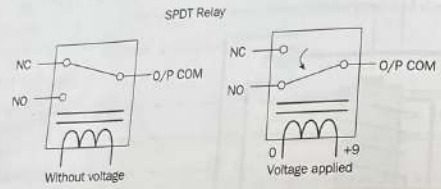


Fig. 5.12 Circuit of a Relay

5.8.6.5 Applications of Relay

Applications of relay include home appliances, home/office automation (temperature-based control of heating/cooling, lighting), traffic signals control, industrial applications, precision agriculture, automotive applications, etc.

5.9 IOT EXAMPLES

Figure 5.13 shows the circuit diagram for the sensor as Light-Dependent Resistor (LDR) and actuators as LED and buzzer.

The text below explains the process for connecting the sensor – LDR and actuators – LED and buzzer:

- Connect the LED, buzzer, and the LDR to the breadboard
- Connect the +ve, that is, VCC of the LDR to 5 V on the Arduino board
- Connect the +ve, that is, VCC of the buzzer to 5 V
- Connect the -ve, that is, GND of the LDR buzzer and LED to the GND on the Arduino board
- Connect resistors of 270 ohms to the +ve of LED and then connect it to pin 13, and resistor of 10 kilo-ohm to the A0 pin of the LDR
- Connect A0 pin of the LDR to the analog pin A0
- Connect I/O pin of buzzer to the digital pin 12
- Push the code into the Arduino board using the Arduino IDE

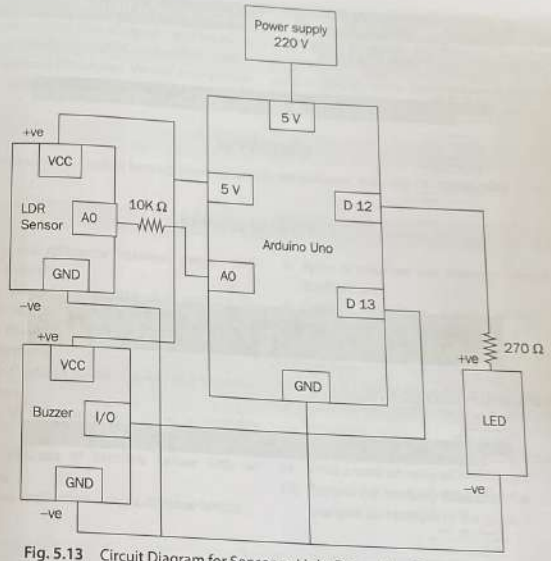


Fig. 5.13 Circuit Diagram for Sensor as Light Dependent Resistor (LDR) and Actuators as LED and Buzzer

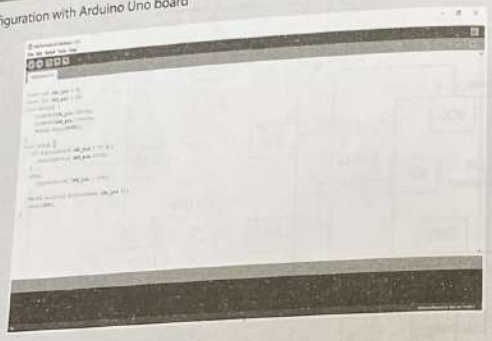
5.9.1 PSEUDO-CODE for the Sketch to Write an Embedded Programming Code Using Arduino IDE

- In the function void setup(), we initialize analog pin A0, as the input, pin 12 and 13 as the output
- We read the input A0, and print it on the serial monitor
- If the ldrStatus is greater than 100. LED blinks and the buzzer beeps or the alarm is deactivated and our LED stops blinking

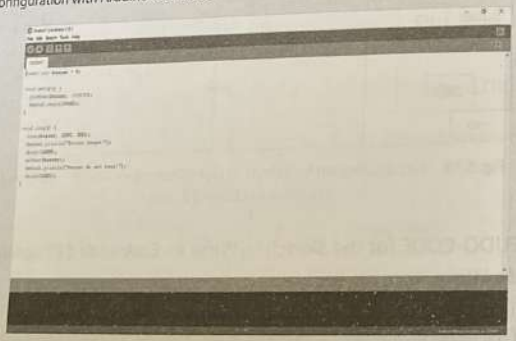
Box 5.1 shows LDR as sensor and buzzer as an actuator.

BOX 5.1: LDR AS SENSOR AND BUZZER AS AN ACTUATOR

LDR configuration with Arduino Uno board



Buzzer configuration with Arduino Uno board



THOUGHT EXERCISES

- Design a sensor and actuating circuit for temperature and humidity sensor.
- Write a code in Python or Embedded C for collecting the information from a temperature and humidity sensor.
- Write a Python code for generating an output for actuating a fan using a relay.

SUMMARY

In this chapter, various kind of sensors and actuators are explored. Initially, the difference between passive and active sensor is introduced. Various environmental sensors and their applications are explained in detail. Medical sensors and their use in IoT applications are also explained. Different actuators such as servo motor, DC motor, linear actuators, and relays are explained along with their application areas.

KEYWORDS

Environmental sensors, medical sensors, linear actuators, servo motor, solenoid, DC motor, relay

REVIEW QUESTIONS

1. What is the difference between passive and active sensors?
2. Write a note on the importance of sensor communication interface.
3. Explain the commonly-used sensor GPS and list its applications.
4. State and explain IoT application of a temperature sensor.
5. Design an IoT application using a humidity sensor.
6. Explain the use of pressure sensor with an example.
7. Design an application using a distance sensor.
8. Write a note on soil moisture sensor and its applications.
9. Explain capacitive touch sensor and its applications.
10. Explain the use of solenoid as an actuator with an example.
11. Write a note on servo motor as an actuator.
12. State some applications of electromechanical actuators.
13. Explain the use of DC motor as an actuator.
14. Write a note on relay as an actuator.
15. Explain the working principle of an RFID sensor and give an example of the same.

REFERENCES

1. Lakowicz, J.R., Geddes, C.D. (2006), In: *Glucose Sensing: Topics in Fluorescence Spectroscopy*. Volume 11, Springer Science and Business Media, Inc., USA.
2. Gault, V., McClenaghan, N. (2009), In: *Understanding Bioanalytical Chemistry. Principles and Applications*, John Wiley & Sons, Ltd, Oxford, UK.
3. McMahon, G. (2007), In: *Analytical Instrumentation: A Guide to Laboratory, Portable and Miniaturized Instruments*, John Wiley & Sons Ltd, West Sussex.
4. Zhang, X., Ju, H., Vang, J. (2008), In: *Electrochemical Sensors, Biosensors and Their Biomedical Applications*, Elsevier Inc., New York.
5. Munden, J., Foley, M. (2007), In: *Diabetes Mellitus: A Guide to Patient Care*, Lippincott Williams & Wilkins, Ambler, Pennsylvania.
6. Porth, C.M. (2011), In: *Essentials of Pathophysiology: Concepts of Altered Health States*, Wolters Kluwer Health, Lippincott Williams & Wilkins, China.
7. http://www.who.int/mediacentre/factsheets/fs312/en/index.html [Last accessed: 30/10/2018]
8. Poretsky, L. (2010), In: *Principles of Diabetes Mellitus*. 2nd Edition. Springer Science and Business Media, LLC, New York, USA.
9. Baura, G. (2012), In: *Medical Device Technologies: A Systems Based Overview Using Engineering Standards*, Elsevier Inc., Oxford, UK.

10. Obtaining Tide Gauge Data. *Permanent Service for Mean Sea Level. PSMSL*. Retrieved 2016-03-07.
11. Other Long Records not in the PSMSL Data Set. *PSMSL*. Retrieved 2015-05-11.
12. Tide gauge history UK National Oceanographic Centre Archived 2015-08-24 at the Wayback Machine.
13. History of tide gauges. *Tide Observation*. Geospatial Information Authority of Japan. Retrieved 2014-04-19.
14. Sclater, N. (2007), In: Mechanisms and Mechanical Devices Source Book, 4th Edition, p. 25, McGraw-Hill, USA.
15. Jafarzadeh, M., Gans, N., Tadesse, Y. (2018), Control of TCP muscles using Takagi–Sugeno–Kang fuzzy inference system, *Mechatronics* 53: 124–133. doi:10.1016/j.mechatronics.2018.06.007.
16. https://www.electronicwings.com/sensors-modules/gps-receiver-module [Last accessed: 07/08/2020]
17. https://circuitdigest.com/article/relay-working-types-operation-applications [Last accessed: 07/08/2020]
18. https://www.arrow.com/en/research-and-events/articles/how-touch-sensors-work

FURTHER READING

1. https://www.ti.com/lit/wp/snia011/snia011.pdf?ICID=I-CT-TP-resources-955 : Temperature sensor solutions for low-voltage systems [Last Accessed: 05/08/2020]
2. https://www.ti.com/lit/an/sbva048/sbva048.pdf?ICID=I-CT-TP-resources-960 : Low-Power Battery Temperature Monitoring [Last Accessed: 05/08/2020]
3. https://www.st.com/resource/en/application_note/cd00174666-increasing-the-resolution-of-analog-temperature-sensors-stmicroelectronics.pdf : AN2648 Increasing the resolution of analogue temperature sensors [Last Accessed: 05/08/2020]
4. https://pdfserv.maximintegrated.com/en/an/AN4679.pdf?ICID=I-CT-TP-resources-963 : Thermal Management Handbook [Last Accessed: 05/08/2020]
5. https://pdfserv.maximintegrated.com/en/an/AN4699.pdf?ICID=I-CT-TP-resources-964 : Overview of Sensor Signal Paths [Last Accessed: 05/08/2020]
6. https://www.st.com/resource/en/technical_note/dm00208005-hts221-digital-humidity-sensor-reference-design-implementation-stmicroelectronics.pdf: Digital Humidity Sensor: Reference Design Implementation [Last Accessed: 05/08/2020]
7. https://www.st.com/content/ccc/resource/technical/document/technical_note/c1/d1/80/37/c3/ad/4b/56/DM00208005.pdf/files/DM00208005.pdf/jcr:content/translations/en.DM00208005.pdf: HTS221 Digital Humidity Sensor: Reference Design Implementation [Last Accessed: 05/08/2020]
8. https://sensing.honeywell.com/white-paper-do-you-know-when-your-proximity-sensor-is-sick.pdf: Healthy Sensing: Do You Know When Your Proximity Sensor is Sick? [Last Accessed: 05/08/2020]
9. https://www.ti.com/lit/an/snoa926a/snoa926a.pdf?ICID=I-CT-TP-resources-926: Capacitive Sensing: Ins and Outs of Active Shielding (Rev. A) [Last Accessed: 05/08/2020]
10. https://www.ti.com/lit/an/snoa927/snoa927.pdf?ts=1596731989011: FDC1004: Basics of Capacitive Sensing and Applications [Last Accessed: 05/08/2020]
11. https://www.ti.com/lit/an/snoa928a/snoa928a.pdf?ICID=I-CT-TP-resources-928: Capacitive Proximity Sensing Using the FDC1004 (Rev. A) [Last Accessed: 05/08/2020]

12. https://www.ti.com/lit/an/snoa943/snoa943.pdf?ICID=I-CT-TP-resources-929: Power Consumption Analysis for Low Power Capacitive Sensing Applications [Last Accessed: 05/08/2020]
13. https://www.st.com/resource/en/technical_note/dm00127867-a3g4250d-supplementary-data-related-to-the-3-axis-digital-output-gyroscope-stmicroelectronics.pdf: TN1189 A3G4250D: Supplementary Data Related to the three-axis Digital Output Gyroscope [Last Accessed: 05/08/2020]
14. https://www.st.com/content/ccc/resource/technical/document/white_paper/c9/a6/fd/e4/e6/4e/48/60/ois_white_paper.pdf/files/ois_white_paper.pdf/jcr:content/translations/en.ois_white_paper.pdf: Optical Image Stabilization (OIS) White Paper [Last Accessed: 05/08/2020]

Open Hardware in IoT

CHAPTER 6

"You don't need anyone's permission to create something great."
—Massimo Banzi—Cofounder of Arduino

OBJECTIVES

- discuss popular Internet of Things (IoT) platforms such as Arduino, Raspberry Pi, Beaglebone, pcDuino, and CubieBoard
- understand hardware configuration of various microcontroller boards and their parameters
- get started with various boards and software installations
- learn interface hardware peripheral devices and perform experiments

OUTCOMES

- comprehend the popular Internet of Things (IoT) platforms and their hardware configuration
- conceptualize the software compatibility of the popular IoT platforms and the interfacing capabilities of various microcontroller boards with hardware and peripherals
- compare popular boards and select a board for a suitable application
- ability to design and develop prototypes with sensors and actuators

REVISION

In Chapters 3 and 4, the middleware technology hardware for wireless sensor network and low power consumption networking protocols are explored, and in Chapter 5, the topics related to sensors and actuators as input and output hardware for the applications of Internet of Things (IoT) are covered. In this chapter, various open electronics hardware boards and devices are explained which can be used for developing IoT applications.

6.1 INTRODUCTION TO INTERNET OF THINGS (IOT) HARDWARE

In the present technology evolution, three fields are are gaining importance namely electronics, data processing, and cloud computing. Their significance from an IoT perspective are:

Electronics Various innovative sensors are being designed and manufactured for capturing information in many of these areas, and is a major contributor to the development of new innovative products surrounding environment, infrastructures, manufacturing sector, supply chain, automotive, etc. A variety of open source embedded development boards are currently available for prototyping in many of these areas, and is a major contributor to the development of new innovative products.

Data Processing Massive Data processing technologies such as those running on various distributed/cluster computing platforms are resulting in gaining new insights from data analytics on real-time as well as archived data.

Cloud Computing Cloud computing Infrastructures such as SaaS, PaaS, IaaS are providing on demand access to hardware, software tools, storage, networking, etc. IT infrastructure and business solutions can be easily configured in cloud using these systems.

The developments in the above mentioned areas are immensely driving the IoT technology in several dimensions and producing many new scientific and business opportunities. In this chapter, the main hardware components of IoT are elaborated.

6.1.1 IoT Hardware and Technology Stack

IoT hardware is a critical component of a system that can have an impact on the cost, functionality, usability, etc. unless it is managed efficiently. The IoT technology stack consists of both the hardware and software. Fig. 6.1 presents the technology stack for an IoT application.

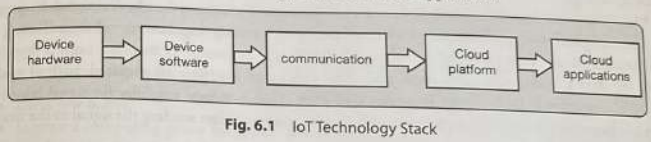


Fig. 6.1 IoT Technology Stack

In this chapter hardware requirements of an IoT application is explored.

6.1.2 IoT Device Hardware Requirements

IoT device framework is explained in four modules as shown in Fig. 6.2.

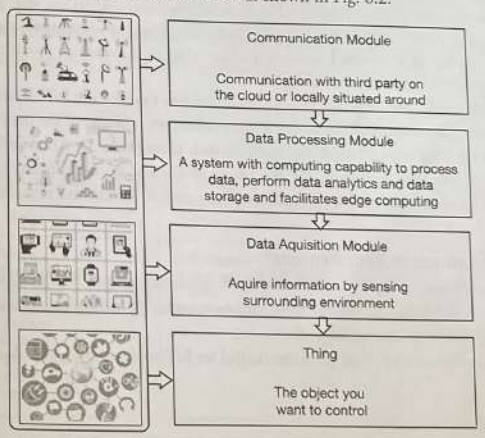


Fig. 6.2 IoT Device Framework

6.1.2.1 Smart Thing Ideation

A smart thing provides some service for controlling some functionality or for monitoring some environmental parameters of an application. Smart IoT device framework constitutes of four modules as shown in Fig. 6.2.

Smart device model functionality is also based on business needs. Sometimes the existing system is modified or sometimes a completely innovative product is required for development, for example smart door lock, autonomous vehicle, etc.

6.1.2.2 Data Acquisition Requirement

While acquiring data from surroundings, necessary hardware is required to sense and acquire the data. Even some additional hardware units such as ADC to convert analog sensor information to digital information.

Major consideration while data collection is right hardware for sensing the physical signal, frequency of generated information from the sensor, information accuracy requirement while collecting it, different sensor requirements specific to the application.

In the data acquisition process, some additional connected hardware are required, such as analog to digital converter (ADC), digital to analog converter (DAC), voltage amplifier for signal boosting, edge hardware for performing preprocessing of the sensor signal before sending the signal to the nearby server or to cloud server.

6.1.2.3 Data Processing Requirements

In data processing stage of IoT framework, end point processing capacity and processed data storage capacity is to be considered while designing and developing the necessary hardware for a specific application. The selection of hardware to fulfill such requirements decides the size, precision, life and cost. The processing power requirement of edge IoT devices for real time application is significantly higher as the number of connected sensors increase. Hence, substantial provision for additional requirement of processing power is needed.

Storage capacity of the end device depends also on off-line time for the IoT device, that is time for which data buffering is needed and frequency of data collection. (volume of the data per unit time).

Many System on Chip (SoC) boards are available such as raspberry Pi, Arduino, Atmel. Choosing the right board is application specific as the processing power and storage capacity of the edge device varies as per the requirements of the application. The network connectivity hardware is supported by SoCs.

6.1.2.4 Communications Requirements

Data transmission capacity is best utilized when interleaving of the frames from different data packets is done. As the data is received from various sensors, by multiplexing the data frames the network bandwidth use can be optimized.

Communication devices and cloud platforms useful for IoT applications are explored in chapter 11.

6.2 PROTOTYPING BOARDS FOR IOT

System on Chip is a mini computer without display monitor and keyboard. It has all functionalities of a CPU such as RAM, input /output ports, limited storage and processing capacity. It also has hardware devices such as ADC and DAC. SoC integrates a programmable micro-controller unit MCU where the application specific program software (firmware) is embedded in a permanent memory space known as SDRAM. Many commercial manufacturers are providing SoCs, for example Raspberry Pi, Beaglebone, Arduino etc.

Arduino board supports programming language known as Embedded C while Raspberry Pi, Beaglebone SoCs support popular programming languages such as Python and Java.

6.2.1 Requirement for Custom Silicon (Customized on Board Chip) in IoT

The semiconductor industry has evolved from emerging technology to enabling technologies in the last few decades. Semiconductor evolution plays major role in the growth of IoT. Devices are usually designed around standard computing components. However, the following factors are driving the change for custom silicon for IoT:

- Ultra-low power
- Increased computing power for better edge analytics
- Hardware level product differentiation
- Customized security features
- Lower BoM (bill of material cost for reducing customized silicon chip cost)

Open-silicon is one of the pioneers and its focus is on custom silicon for IoT applications.

6.2.2 SoC Classification based on Functionality

IoT SoCs are broadly classified in the following categories:

- Edge SoCs
- Gateway SoCs
- Cloud side SoCs

Figure 6.3 represents SoCs at different levels for an IoT application, from endpoint to cloud.

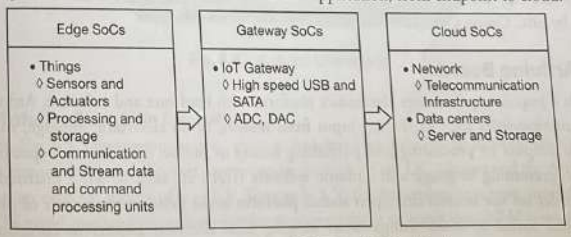


Fig. 6.3 Different Types of SoCs for IoT Applications

6.2.2.1 IoT Edge SoCs

Edge SoCs sense and/or actuate and mainly belong to microcontroller class (CPU based). It has analog and digital interfaces to connect to sensors and actuators and wired or wireless connectivity.

An edge device is an endpoint device, which is a remotely manageable secured device with the ability to manage the following:

- Sensors and actuators
- Local processing of real-time streaming data and local and /or cloud-based
- Send and receive commands for communication with the devices

6.2.2.2 Gateway SoCs

These are built using application class CPUs and wired or wireless connectivity on both sides. These SoCs have high-speed interfaces such as USB and sufficient computing power given by CPUs and DSPs (Digital Signal Processing hardware units such as ADC, DAC) as well as hardware engines.

6.2.2.3 Cloud SoCs

Traditional Application Specific Integrated Circuits (ASICs) are used for cloud SoCs in many networking applications. Cloud side SoCs are LTE (Long-Term Evolution) base stations, satellite modems, networking switches etc.

6.2.2.4 Single-Board Computer (SBC)

A computer built on a single circuit board with a microprocessor(s), memory, input/output (I/O), and other features required for a complete functional computer is known as Single-Board Computer (SBC). Single-board computers are generally without peripheral functions. The first single-board computer was developed in 1976 using Intel C8080A. Other early single-board computers are KIM-1.

Currently, Raspberry Pi and Onion Omega are two most popular SBC families. The Raspberry Pi SBC was developed by the Raspberry Pi Foundation for promoting the basic level computer science in the schools and also in developing countries. Many generations of Raspberry Pi are available. These models have a Broadcom SoC and a CPU compatible with ARM and also have a graphics processing unit (GPU).

The Omega SBC is developed by a start-up company called Onion. This system combines the power efficiency and tiny form factor of Arduino and flexibility of Raspberry Pi. The D-Link routers are powered by SBC Onion Omega.

6.2.3 Arduino Boards

Arduino is a popular open-source electronics platform with hardware and software. Arduino boards support functionalities such as reading input from sensors, or an electronic message, turning such inputs into outputs by processing and publishing locally or online. To perform a specific task, the Arduino programming language and Arduino software (IDE) are used to write instructions to the microcontroller on the board. This open source platform is the brain of thousands of projects over the years, and useful to make things smart ranging from everyday objects to complex scientific instruments.

Arduino is an easy tool for prototyping, which is useful for students from various engineering and non-engineering disciplines. Arduino boards are completely open-source, and can be easily adapted for a particular IoT application. The software supporting the Arduino boards is also open source.

Arduino software is user friendly for beginners and also flexible for advanced users. This software is compatible with different operating systems (OS) such as Mac, Windows, and Linux. Arduino simplifies working with microcontrollers as compared to many other competing platforms.
There are different kinds of Arduino boards:

- Arduino Uno (R3)
- Arduino Mega (R3)
- Arduino Leonardo

More details about these and other Arduino boards are at reference [27].
Advantages of Arduino platform:

- Low cost
- Open-source as well as extensible software and hardware
- Simple and easy-to-use programming environment
- Compatibility with many OS

Figure 6.4 shows the popular Arduino Uno board used for IoT applications and by other DIY products.

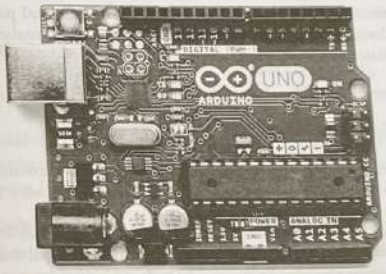


Fig. 6.4 Arduino Uno Board

6.2.3.1 Arduino Board Pin Layout

As shown in Fig. 6.5, power supply can be given to Arduino Uno either by an external source (at X1) or by a USB connection. External power source can be through AC to DC adapter and the Arduino board operates on an external supply of 6 –20 V. To supply 5 V, the board power supply should not be less than 7 V. To avoid overheating, voltage-regulator power supply of more than 12 V is avoided. Hence, the recommended voltage range is 7 to –12 V.

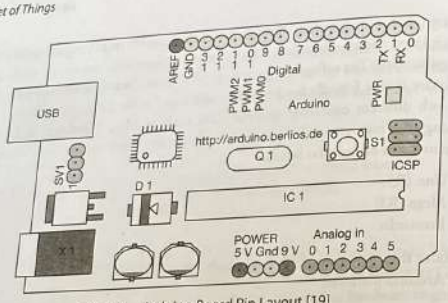


Fig. 6.5 Arduino Board Pin Layout [19]

The power pins are explained as follows:

VIN It represents the input voltage to the Arduino board when the power is supplied from the external source. Other way to supply the power to the Arduino board is using USB connection or any other regulated power supply. The power supply is given through this pin. If the voltage is supplied through power jack then voltage access is through this pin.

5 V The Arduino microcontroller on its board is powered by regulated power supply. This power supply is either given through the VIN pin on the board or by on-board regulator or it is supplied using USB or any other regulated supply.

3V3 3.3 V supply is generated by the on-board voltage regulator with maximum current limit of 50 mA.

GND These are Ground pins.

Memory 32 KB of flash memory is provided in the ATMega328 microcontroller chip for storing the code. 0.5 KB is used for bootloader. The board also has 2 KB of SRAM and 1 KB of EEPROM.

Input output pins Fourteen digital pins on the Arduino Uno board can be used as an I/O using different functions such as pinMode(), digitalWrite(), and digitalRead(). All these pins need 5 V power supply. The maximum current provided or received by each pin is 40 mA.

Some pins that have specialized functions are listed below:

Serial 0 (RX) for receiving and 1 (TX) for transmitting

TTL Series Data These pins are connected to the corresponding pins of ATmega8U2 USB to TTL serial chip.

External interrupts Pin numbers 2 and 3 can be configured to trigger an interrupt on some event such as low value, a rising or falling edge, and change in a value. The function *attachInterrupt()* can be used for this purpose.

PWM Pins 3, 5, 6, 9, 10, and 11 provide 8-bit PWM output with the *analogWrite()* function.

SPI Pin numbers 10 (SS), 12 (MISO), 12 (MISO), and 13 (SCK). These pins are used for Serial Port Interface (SPI) communications, which are facilitated by the microcontroller hardware.

LED The in-built LED is connected to digital pin 13. When the pin is HIGH value the LED is on and for LOW value it is off.

I2C 4(SDA) and 5 (SCL). Support I2C (TWI) communication using the wired library [20].

Reset To reset the microcontroller.

Arduino Uno board has six analog inputs, each of which has 10-bit resolution which means it can take values between 0 and 1023. The value measured will ground means 0–5 V.

6.2.3.2 Arduino Communication

Arduino board has many facilities to communicate with other Arduino boards or computers or other microcontrollers. The serial communication of ATmega328 microcontroller chip is with UART TTL (5 V), with pins 0 (RX) and 1 (TX).

An ATmega8U2 on board channels is for serial communication over USB and works as a virtual communication port to software on the computer. Standard USB COM drivers are used by 8U2 firmware and no external driver is needed. Windows systems require *.inf file for installation of drivers for external devices.

6.2.3.3 Programming

The Arduino Uno can be programmed using the Arduino software. To access the software, select "Arduino Uno w/ATmega328" from the Tools > "Board menu" (with the microcontroller on the board) [21].

The bootloader is preburned on the chip ATmega328 on the Arduino. Hence, new code can be uploaded to the microcontroller chip without the use of external hardware programmer. Even new code can be uploaded using In-Circuit Serial Programming (ICSP) header.

6.2.3.4 Automatic Software Reset

The Arduino software allows uploading of the code just by pressing the upload button in the Arduino environment. A sketch may run on the board to receive one-time configuration or other data when it first starts. It must be taken to ensure that the software with which it communicates waits for a second after opening the connection and before sending the data.

To disable the auto-reset, the UNO contains a trace that can be cut. The pads on either side of the trace can be soldered together to re-enable it. It is labelled as "RESET-EN." An example sketch in embedded programming is in Box 6.1 to represent how to access A5 analog pin on Arduino Uno board to sense temperature sensor output.

BOX 6.1 EXAMPLE SKETCH FOR ACCESSING A5 ANALOG INPUT PIN OF ARDUINO BOARD

```
sketch_temperature | Arduino 1.8.3
File Edit Sketch Tools Help
sketch_temperature §
#include <math.h>
// Sensing Analog pin input at A5 pin and converting it to digital value on the Arduino Uno Board
int sensorPin = A5; // select the arduino board input pin A5

double Thermister(int RawADC)
{
  double Temp = 0.0, adc = 0.0;
  Serial.println(analogRead(sensorPin));
  Serial.println(digitalRead(sensorPin));
  Temp = ((RawADC - 0) * ((125 - (-55)) / (1023 - 0))) + (-55);

  return Temp;
}

void setup()
{
  Serial.begin(9600);
}

void loop() {
  Serial.print(Thermister(digitalRead(sensorPin))); //read pin A5
  Serial.println("°C");
  delay(500);
}

Auto Format finished.
```

6.2.3.5 Arduino Board Other Parameters

USB overcurrent protection The computer's USB ports are protected from short and overcurrent by the resettable poly-fuse of Arduino Uno.

Physical characteristics The UNO PCB is of size 2.7-inch length and 2.1-inch width.

6.2.3.6 Functionality of Arduino

The environment can be sensed by Arduino using various sensors. These sensor inputs are used to control the output devices such as lights, motors, and other actuators. The microcontroller chip is programmed using the Arduino development environment using one of the programming languages supported by Arduino board microcontroller. The Arduino official website is useful for the latest instructions [22]. From this site Arduino IDE can be downloaded and unzipped. The IDE is installed on PC and then you can plug the Arduino board to PC using USB cable.

6.2.3.7 Getting Started with Arduino

Arduino is an open-source hardware and software, platform for making interactive projects. Arduino technology is used to teach and learn the innovative multidisciplinary studies. It is a very good platform to design and develop applications in engineering, IoT, Robotics, and art fields. Arduino technology is useful to develop building skills and problem-solving skills.

6.2.3.7.1 Software (IDE) setup on Arduino board Two options are available to upload the programming Arduino software (IDE). If you have a reliable Internet connection, then even online IDE can be used (Arduino web editor). The Arduino software allows you to write the programs and upload them to the board. If you are working offline, then a latest version of desktop IDE is needed.

There are many retired Arduino boards listed and instructions for retired boards are available on the Arduino website.The instructions for different example shields are on the links provided on the Arduino official website [17].

6.2.4 Raspberry Pi

Raspberry Pi foundation developed first two models of Raspberry Pi. After the first two models of Pi, model B and subsequently several models such as B+, Zero, etc. released. The foundation is an educational charity to promote teaching of the basic computer science in schools and in developing countries.

Raspbian is the official OS of the foundation. The OS can be installed with NOOBS (New Out Of Box software for easy OS installation manager for Raspberry Pi) or by downloading the image Raspbian. The installation guide is available on the Raspberry Pi official website [23]. Raspbian comes preinstalled with plenty of software for education and general use. It supports Python, Scratch, sonic Pi, Java, Mathematica, etc.

Desktop image for Raspbian OS is contained in a ZIP archive and is of a size more than 4 GB. The pin layout for raspberry Pi board is shown in Fig. 6.6.

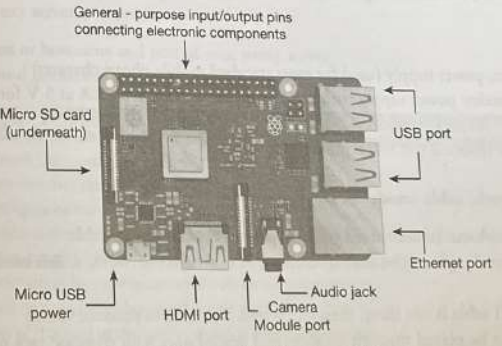


Fig. 6.6 Pin Layout for Raspberry Pi Board

(Source: Copyright © the Raspberry Pi Foundation)

The latest model of Raspberry Pi board

Raspberry Pi 4 Model B+ Model is using the Broadcom chip BCM2711 and a large memory variant up to 16 GB. This chip architecture also has quad-core design. But it has more powerful ARM A72 core for upgrading SoCs. This architecture is capable of addressing more memory and has attached Ethernet controller. Default operating system image is a 32-bit LPAE kernel. Many other options are available for OS such as gentoo-on-rpi-64bit Gentoo. Even the 64 bit beta version OS is released. Both 32 and 64 bit OS images are called as Raspberry Pi OS. This model is 50% faster than Raspberry Pi 3B+ model. Its new 32 bit unit, VideoCore runs at 500 Mhz as additional memory unit. The VideoCore is 32-bit, and has a new Memory Management Unit, which means it can access more memory than previous versions.

6.2.4.1 Set up Procedure for Raspberry Pi

Essential requirements for the setup of Raspberry Pi board are at reference [27]:

SD card

- 8 GB Class 4 SD card (ideally with NOOBS preinstalled)

Connectivity cables and display

- Any DDMI or DVI monitor and any TV can be used as a display device for Raspberry Pi.
- HDMI input will give best result but even other connections are available for compatibility with the older versions.

Keyboard and mouse

- Any standard USB keyboard and mouse
- With wireless pairing, wireless mouse and keyboard can be used
- Keyboard layout and configuration details are available at link raspi-config[32]

Power supply

- USB micro power supply (used for most standard mobile phone chargers)
- A good-quality power supply with current specification at least 2 A at 5 V for 3B model or 700 mA at 5 V for earlier models (Low current power supply works for basic usage but are likely to cause Pi to reboot if too much power is drawn)

Optional (network) cable (model B, B+, 2, 3 only)

- Local network can be connected to Raspberry Pi using Ethernet cable.
- USB wireless dongle can be used to connect Pi to wireless network, which needs configuration.
- Audio lead
  - If HDMI cable is not there, then audio lead is needed to produce sound
  - Audio can be played through speakers and headphones with standard jack of 3.5 mm
  - With HDMI cable, audio can be played by connecting to monitor with speakers. If any other speakers are to be used to play, the audio configuration is required.

Troubleshooting For any setup issues, Raspberry Pi forum can be used to post the query with proper details of the problem.

Raspberry Pi OS image installation steps are available on Raspberry Pi official website at [23].

For more advanced steps, use the system-specific guide for different OS such as Linux, Mac OS, and MS Windows.

Some examples for getting started with software Raspbian are available at [29] for the functionalities as listed below:

- Scratch: Programming tool to create animation
- Python: A general purpose programming language
- Sonic Pi: For writing code to play music with Pi
- Terminal: Linux terminal for using command prompt
- GPIO: General purpose I/O pins for interaction with the surrounding environment
- Minecraft: Programming interface using Python code to access the world around through GPIO
- Python games: Some readymade Python games to play on Raspberry Pi
- WordPress: A content management and blogging system
- Mathemaica: Industry leading computational platform
- Camera module: Raspberry Pi camera
- Webcam: Use of any standard USB webcam
- Codi: Media center software for Raspberry Pi
- Playing audio
- Playing video
- Demo programs: To explore Pi capabilities

Basic guide for Raspberry Pi configuration is available at [28] and they are useful for configuration of Raspberry Pi. Some of the configuration tasks are mentioned in the Box 6.2. If you want your Raspberry Pi to access the internet via a proxy server of a workplace, then you will need to configure your Pi to use the server before you can get online.

For configuring proxy server, you need:

- The IP address or hostname and port of your proxy server
- A username and password for your proxy (if required)

BOX 6.2 RASPBERRY PI SOFTWARE CONFIGURATION TOOLS (RASPI-CONFIG)

1. Change user password for the current user
2. Configure network settings
3. Boot Configure options for start-up
4. Localization options to set up language and regional settings to match your location
5. Interfacing options for configuration of connections to peripherals
6. Overclock option for configuration of overclocking for your Pi
7. Advanced options to configure advanced settings
8. Updation of this tool to the latest version is needed frequently
9. About raspi-config tool more information is available at source mentioned at link below

(Source: Copyright © the Raspberry Pi Foundation)

6.2.5 BeagleBone

Beagle family offers tiny and affordable open-source boards called Beagels with high performance and low-power. Beagels support Android, Ubuntu, and other Linux flavours with flexible peripheral interfaces and a proven ecosystem of "Cape," plug in boards.

6.2.5.1 BeagleBone Black

BeagleBone Black is an open hardware with Linux OS and boots the OS in few seconds. Its microcontroller chip is with 1 GHz AM335x ARM Cortex-A8 processor (refer to Fig. 6.7).

Fig. 6.7 BeagleBone Black Board [24]

6.2.5.2 BeagleBone Board Features

These boards are open software tiny computers. Beagle supports much functionality through small software packages just like a mini-computer. These boards are very user friendly for any DIY project or developing any IoT product.

This board can be connected to USB client for power and communication, USB host, HDMI, Ethernet, and 2 × 46 pin headers. It has software compatibility Ubuntu, Debian, Cloud 9 IDE on node.js, Android, and many other software. BeagleBone Black comes with in-built wireless networking capability with on-board 802.11 b/g/n 2.4 GHz Wi-Fi and Bluetooth.

Pocket BeagleBone is an ultra-tiny yet a complete Linux enabled with scalability, open-source computer. It is low cost with slick design and an ideal development board for DIY projects. You can get its more information on the website https://beagleboard.org/pocket.

6.2.6 pcDuino

pcDuino as shown in Fig 6.8 is a mini PC platform. It runs an OS such as Ubuntu Linux. Any HDMI-enabled TV or monitor screen can be connected to it as an output device. This platform can easily run an OS such as a full-blown OS for a PC. It can also run Android 4.0 ICS.

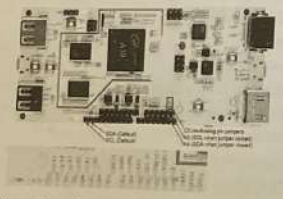


Fig. 6.8 pcDuino Board Source: SparkFun Electronics and Juan Peña (Source: SparkFun Electronics and Juan Peña)

6.2.6.1 Hardware and Software for pcDuino

Hardware specification Components of pcDuino are CPU All Winner A10 SoC, 1 GHz ARM Cortex A8, GPU OpenGL ES2.0, OpenVG 1.1 Mali 400 core DRAM 1GB, Onboard Storage 2 GB Flash, Micro-SD card slot for up to 32 GB, Video Output HDMI, OS Linux3.0 + Ubuntu12.04, Android ICS 4.0, Extension Interface Arduino Headers, Network interface USB Wi-Fi extension (not included), Ethernet 10/100 Mbps, and accessories required are power adaptor with Micro USB port, cable of Micro USB with 2 A current carrying capacity.

The pcDuino boards have very well-exposed hardware peripherals. The tutorial "Getting started with pcDuino" helps to know basics about pcDuino [26]. A fully functional Linux OS makes pcDuino a microcomputer. SPI, a serial peripheral interface, with synchronous clock for communication between chips is at board level. It requires minimum four wires as clock, master-in-slave-out data, master-out-slave-in data, and slave chip select. Each additional chip needs one extra chip select line. IIC or I^2C (interintegrated circuit) TWI (two-wire-interface), SMBus, is used for communication with multiple devices. Serial communication is an asynchronous data interface with minimum two wires (data transmit and data receive). Sometimes, additional signals are added to indicate whether the device is available for sending or receiving the data.

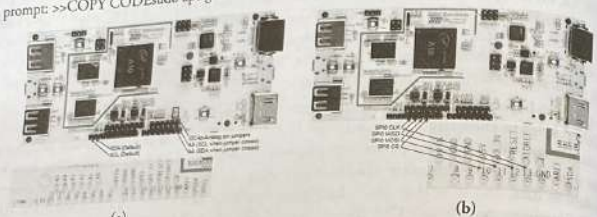
The pcDuino has 18 GPIO pins. The GPIO pins can be accessed easily by opening the file associated with the pin you wish to access and can read and write the file. These files are located at the location: /sys/devices/virtual/misc/gpio/pin/. Each pin has its own file out of the 20 files available and 18 of them are mapped to GPIOs. Some pins can be used for multiple purposes such as serial, SPI, input, and output. "0," "1," and "8" makes the pin "INPUT," "OUTPUT," and "INPUT with pull-up register," respectively. The logic on these pins is 3.3 V and use of 5V devices may cause damage to the pcDuino board. The output current capacity is not as high as Arduino boards though it is sufficient to drive a small output actuator such as LED.

Unlike Raspberry Pi boards, pcDuino has some analog I/O pins. Analog I/O is accomplished using six pins labelled A0–A5 on Arduino-ish headers. The high output of all the pins is limited to maximum 3.3 V. A0 and A1 are 6-bit inputs returning a value between 0 and 63 with a voltage range 0–2 V. A2–A5 are 12-bit inputs operating in the range 0–3.3 V. Using pulse width modulation analog output of pcDuino is accomplished in the Arduino-ish header, 6 PWM enabled pins are given, numbered 3, 5, 6, 9, 10, and 11, same pins like on Arduino boards. Pin numbers 5 and 6 are 520 Hz 8-bit PWM while other pins are having range 0–20 at 5 Hz. The pin configuration details are at the link [30] GPIO read and write example in C++ programming language is at [31].

Serial communications Several built-in serial ports are there on pcDuino board. The A10 processor on the pcDuino board has eight serial ports. Two ports map to the Linux device: /dev/ttyS0 as debugging port and /dev/ttyS1 as the UART, a pinned out on the Arduino-ish header. The mode registers for each pin control the functionality of that pin. For serial I/O for pins 0 and 1, you may need to write "3" to the mode files.

I2C communications There are several I2C bus devices but easily available one is at location /dev/i2c-2. The I2C bus speed is fixed with clock 200 kHz, hence some devices may not work on the I2C

bus of pcDuino. Users cannot change the bus speed as it is compiled with driver software. A set of tools allows to work with I2C bus without writing the code. These tools can be installed at the command prompt: >>COPY CODEsudo apt-get installs i2c-tools.



(a) (b)

Fig. 6.9 pcDuino (a) I2C Pin to Analog Pin Jumper Layout and (b) SPI Pin Layout

(Source: SparkFun Electronics and Juan Peña)

SPI communications By using pcDuino headers, two different SPI buses are used. At a time, only one device is supported on the bus. SPI peripheral SPI0 can be accessed with the Arduino compatible headers using four pins MOSI, MISO, SCK, and CS from the processor to provide hardware support for SPI communications.

6.2.7 CubieBoard Open-Source Hardware

Cubieboard is a single-board computer made in Zhuai, Guangdong, China. It can run Android 4 ICS, Ubuntu 12.04 desktop, Fedora 19 ARM Remix desktop, Armbian, Arch Linux ARM, Debian-based Cubian distribution, or OpenBSD. It uses ALLWinner A10 SoC and is compatible with media PCs and chip tablets. It can run Apache Hadoop computer clustering Lubuntu Linux distribution. The little motherboard uses AllWinner A10 SoC capabilities.

CubieBoard2 CubieBoard2 is released by CubieTech, few months after the release of CubieBoard1. In 2013, the board is upgraded with 2 ARM Cortex-A7 MPCore CPUs. This board also has dual core Mali400 GPU.

CubieTruck (CubieBoard3) The third version is with a larger PCB layout. The Network Interface Card (NIC) RTL8211E achieves a transfer sending rate up to 630–638 Mbit/s (Idle for 5–10%) and receiving rate 850–860 Mbit/s (0–2% Idle). 3.5-inch HDD add-on package can be used to power CubieBoard and to supply necessary 12 V to 3.5-inch HDD. Even LiPo batteries are used as another option to power the CubieTruck.

CubieBoard3/CubieTruck is suitable to make real products. CubieTruck is the third board of Cubieteam. So also named as Cubieboard3. It is with AllWinner A20 main chip but having enhanced features such as 2 GB RAM, on board VGA display interface, on board WiFi, and Bluetooth. It supports Li battery and RTC, SPDIF audio interface.

6.3 CELLULAR IOT HARDWARE

In this section, the cellular IoT devices discussed are Adafruit Feather 32u4 Fona, GOBLIN 2, LinkIt ONE, Hologram Dash, and Arduino GPRS Shield.

6.3.1 Cellular IoT Hardware

Ubiquitous cellular networks are expanding day by day. Leading IoT providers Twilio, Hologram, and Particle have allowed to go beyond the local area network for getting affordable IoT applications. Following are some top cellular enabled hardware devices:

Adafruit Feather 32u4 FONA The all-in-one Adafruit Feather 32u4 FONA is Arduino compatible as well as is a cellular board capable to support audio, SMS, data. The Feather has a Lipo battery that can be used in mobility and micro USB port for powering while stationary. As the name suggests, the feathers are flexible, lightweight, and portable. The GSM module of this board is trusted and tested FONA module.

Particle Electron Full stack of IoT device platform including the device, connectivity hardware, cellular products, SIMs, and cloud is called a particle. Particle Electron is a device solution that integrates hardware, network, and cloud using a global SIM data plan, which has worldwide coverage. The Electron complies with any cellular standard and is certified by FCC/PTCRB/CE/IC.

Hologram Dash Hologram.io It is a cellular connection platform. This platform lets you interact with devices and can easily do routing of incoming and outgoing messages with a secure API.

Hologram also offers the 'Hollogram Dash', a cellular board with Arduino IDE compatible and pre-certified for secure end use. It is also equipped with a networking firmware OTA (over the air) code updates.

Arduino GPRS Shield For cellular IoT application, you need to connect Arduino to the Internet using the GPRS wireless network. This shield is compatible with all the Arduino boards, which have the same pinout and form factor as that of the standard Arduino board. By plugging the GPRS module onto the Arduino board and plug in a SIM card from a service provider that offers GPRS service, the IoT application will be ready to connect to Internet. The GPRS shield acts as human machine interface.

The GPRS shield uses UART and with simple AT commands, the shield is configured and controlled. The IoT application can make or receive calls using the onboard audio or microphone jack. The application can even send or receive SMS messages. The GPRS Shield also has 12 GPIOs, 2 PWMS, and 1 ADC.

Linkit ONE Design and prototype of IoT devices are enabled by Linkit One development platform. It uses hardware and API similar to Arduino boards. Linkit One development kit includes a development board, a WiFi and Bluetooth antenna, and a GSM antenna and a GPS antenna. These modules are powered by Li-battery. This board has all-in-one connectivity.

GOBLIN 2 The GOBLIN 2 are autonomous IoT boards with a module to control the charge of Li-Po battery of 3.7–4.2 V. This board is compatible with Arduino and ATMEL Studio. It has simple connectivity and versatile setup for remote communication.

Important guidelines for getting started with cellular IoT devices is available at the websites in references [34-39].

6.4 INDUSTRIAL MICROCONTROLLER (PLC AND RTU)

Robust and rugged digital computers are required in the applications such as robotic devices, assembly lines, and for any process where high reliability, process fault diagnosis, and easy programming are needed.

6.4.1 Programmable Logic Controller (PLC) and Remote Terminal Unit (RTU)

A gigantic microcontroller is used as a Programmable Logic Controller (PLC) in the last few decades. It does the same thing as a microcontroller can do with higher speed, better performance, and higher reliability.

Many functions of RTUs and PLCs are similar in terms of usage. RTUs are better options for wide area applications while PLCs are best options suited for local area network. Various PLC and RTU units used in last two decades are shown in Fig. 6.10.

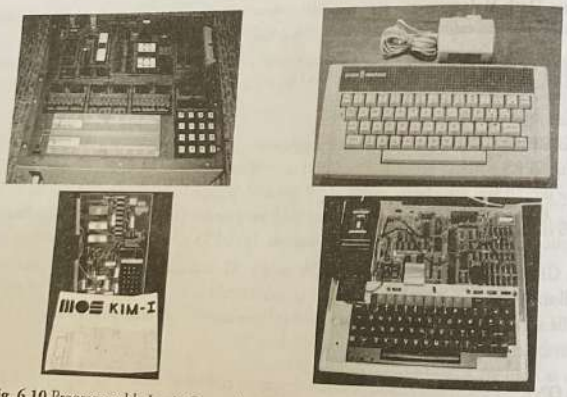


Fig. 6.10 Programmable Logic Controller (PLC) and Remote Terminal Unit (RTU)

6.5 VARIOUS OTHER HARDWARE

IoT hardware includes many devices such as ADC, PWM, PDM, amplifiers, and filters as peripheral devices I/O may not be suitable as a control output or input without conditioning.

6.5.1 Hardware to Convert Continuous Signal to Digital Signal

Hardware ADC is an analog-to-digital converter useful for converting analog voltage received as an input on a pin to a digital number. This digital number is useful for writing a programming logic to perform some function.

The ratiometric value is reported by ADC in a given normalized range. ADC is dependent on the system voltage. Generally, 10-bit ADC is used by IoT Arduino boards.

Pulse-width modulation (PWM) or pulse-duration modulation (PDM) is a way to convert digital signal to analog signal. Modulation encodes the digital information to a pulsing signal.

6.5.2 Hardware for Signal Conditioning, Scaling, and Interpretation

Signal conditioning includes amplification, filtering, converting, range matching, isolation, etc.

Amplifying Amplifying increases the resolution of a signal and its signal-to-noise ratio.

Filtering It is a common signal conditioning function for bandpass limiting the given signal to preserve only desired frequency range of the signal.

Excitation External power is needed for an active sensor. The stability and precision of excitation signal is important.

6.6 COMPARISON OF DIFFERENT HARDWARE PLATFORMS

Various popular hardware microcontroller boards such as Arduino Uno, Raspberry Pi, BeagleBone Black, pcDuino, and CubieBoard are compared for several hardware and software parameters in Tables 6.1 and 6.2.

Table 6.1 Comparison of IoT Platforms: Size, Weight, CPU, Memory, and Power

IoT platform name	Size (mm)	Weight (g)	Processor	RAM and FLASH	Power	LAN (Mbit)	WiFi Module
Arduino (Uno) B Model	75 × 53 × 15	~30	ATMEGA328,	RAM: 2 KB FLASH 32 KB EEPROM 1 KB	7–12 V /USB	–	–
Raspberry Pi R3	85.6 × 53.98 × 17	45	ARM BCM2835	RAM: 256-512 MB SDRAM FLASH: SD card	5 V/ USB	10/100	–
Beagle-Bone Black Rev A5	86.3 × 53.3	39.68	AM335x 1GHz ARM® CortexA8	RAM: 512 MB DDR3 FLASH:	5 V	10/100	–
pcDuino3	121 × 65		CPU:1GHz ARM Cortex A8 GPU: OpenGL ES2.0, OpenVG 1.1 Mali 400 core	RAM: 1 GB On Board FLASH: 2 GB	5 V	10/100	b/g/n (RTL8188)
Cubie-Board3	86.4 × 53.3	40	CPU: ARM Cortex-A7 GPU;	RAM; 1 GB FLASH: 8 GB	5V	GbE	a/b/g/n (BCM4329)

Table 6.2 Comparison of IoT Platforms: Expansion Connectors, Operating System, and Programming Languages

IoT platform name	Analog inputs	Digital I/O pins	Min power	Clock speed	USB ports	Board operating system	Programming language supported
Arduino (Uno)	6	14	42 mA 0.3 W	16 MHz	1	–	Arduino
Raspberry Pi	0	14	700 mA 3.5 W	700 MHz	1–2	Raspbian, Ubuntu, Android, ArchLinux, FreeBSD, Fedora, RISC OS	C, C++, Java, Python
BeagleBone Black	6	14	170 mA 0.85 W	700 MHz	1	Linux Angstrom, Android	Arduino,
pcDuino		14		1 GHz	1–2	Ubuntu Linux, Android	C, C++, Python

HANDS-ON EXERCISES

1. Design and develop a circuit using Arduino Uno board to sense LDR sensor using analog pin input and display the output by writing a sketch using embedded programming using Arduino IDE.
2. Install Raspbian OS on Raspberry Pi board using NOOBS and configure WiFi connectivity.
3. Write a Python program using Raspberry Pi IDE to read input from a digital pin and send it to a remote system using WiFi connectivity.

BEST PRACTICES

- For the beginners, Arduino boards are recommended as tutorials, sample projects are available on the website https://www.arduino.cc/. This board is simplest to interface to external hardware and a wide variety of sensors and actuators.
- For application where minimum board size is needed, Arduino board is recommended as it is a very small, inexpensive, and embedded system on a chip microprocessor of Atmel.
- For applications using the Internet, Raspberry Pi or BeagleBone is recommended. Both the devices are Linux mini computers.
- For applications that use external sensor interface, Arduino is recommended.
- For battery-powered applications, Arduino is recommended as Arduino works on uses least-power consumption. But Arduino cannot work with wide range of input voltages and in terms of computer power per watt BeagleBone performance is much better.
- For applications that use Graphical User Interface (GUI), Raspberry Pi is recommended. Raspberry Pi is an ideal choice when a low-cost web-browsing IoT smart object is to be prepared. It is a small and affordable computer that you can use to learn programming.
- The pcDuino board is similar to the Raspberry Pi board. It has all the features of Raspberry Pi and also some enhanced features such as 2 GB of flash memory and more storage capacity, which makes this board little expensive than Raspberry Pi.
- PcDuino, Beaglebone Black, and CubieBoard have expansion connectors equipped with GPIO and other devices. When power affordability is a major concern, then Cubieboard can be a good choice.

SUMMARY

The Arduino platform is flexible and has great ability for interfacing with hardware peripheral devices. This is usually a better choice for performing mini projects and experiments with best price/performance ratio. Raspberry Pi used when the applications demand higher computing power and GUI interface. It is a popular board with small computer capability to learn programming and design and develop projects slightly complex IoT applications.

The BeagleBone is a great combination to provide best of Arduino interfacing flexibility and full Linux environment as Raspberry Pi.

pcDuino is a mini PC platform that combines the benefit of an ARM-based mini PC and Arduino ecosystem. Hence, pcDuino is a bridging platform powered with open software Linux and open hardware.

Various other platforms are also available for developing IoT applications such as Cubieboard, Particle Photon, Intel Edison, and Tessel. The selection of the board based on the desired application to be developed is a major design decision to make the product with optimum functionality and for making it cost effective.

KEY TERMS

IoT Platforms, Prototyping boards, microcontroller, System on Chip (SoC), Arduino, Raspberry Pi, Beaglebone, pCDuino, Cubieboard.

FURTHER READING

1. Arduino Board. [ONLINE]. Available at: http://arduino.cc/en/Guide/HomePage
2. Raspberry Pi Board. [ONLINE]. Available at: http://www.raspberrypi.org
3. pcDuino. [ONLINE]. Available at: https://s3.amazonaws.com/pcduino/book/Introduction+to+pcDuino.pdf
4. pcDuino User Guide. [ONLINE]. Available at: https://strawberry-linux.com/pub/pcDuino_UserGuide_Rev02.pdf

REVIEW EXERCISES

1. Explain with a diagram the analog and digital pins of Arduino Uno board useful for interfacing input and output.
2. Which programming language is supported by Arduino boards? Explain with its advantages and disadvantages.
3. Explain with a diagram the interfacing pins for input and output on Raspberry Pi board.
4. Describe the steps to write Raspbian OS on the Raspberry pi board using NOOBS.
5. Explain Internet connectivity configuration for Arduino board through WiFi as well as through proxy server of a workplace.
6. Describe steps for WiFi Internet connectivity configuration for Raspberry Pi board.
7. Compare the boards Arduino Uno and Raspberry Pi with respect to its support to speed, memory, and storage.
8. Write a note on Beaglebone board and what are its advantages and disadvantages when compared with Raspberry Pi board?
9. Write a note on pcDuino board and its enhanced features as compared to Arduino board.
10. Compare the boards Beaglebone and pcDuino for their different features.

REFERENCES

1. Arduino vs Raspberry Pi vs Beaglebone vs pcDuino. [ONLINE]. Available at:https://random-nerdtutorials.com/arduino-vs-raspberry-pi-vs-beaglebone-vs-pcduino/
2. Exploring Beaglebone. [ONLINE]. Available at:http://exploringbeaglebone.com/chapter10/
3. Turn your pcDuino into an Apple TV. [ONLINE]. Available at: http://www.linksprite.com/
4. Automate your garage with pcDuino. [ONLINE]. Available at: http://www.pcDuino.com/
5. Single Board Computer. [ONLINE]. Available at: https://www.allaboutcircuits.com/news/single-board-computer-beaglebone-black-raspberry-pi-3-asus-tinker-board/
6. How To Choose Right Platform. [ONLINE]. Available at: https://makezine.com/2014/02/25/how-to-choose-the-right-platform-raspberry-pi-or-beaglebone-black/
7. Adafruit. [ONLINE]. Available at: https://www.adafruit.com/index.php?main_page=category&cPath=17
8. Arduino Final Hand-out. [ONLINE]. Available at: https://dlnmh9ip6v2uc.cloudfront.net/learn/materials/1/Arduino_final_handout.pdf
9. IoT Solutions. [ONLINE]. Available at: https://www.open-silicon.com/solutions/iot-asics/
10. IoT Edge SoC Platforms. [ONLINE]. Available at:https://www.open-silicon.com/iot-edge-soc-platform/
11. IoT Gateway Platform. [ONLINE]. Available at: https://www.open-silicon.com/iot-gateway-platform/
12. Adafruit Feather 32u4 Fona. [ONLINE]. Available at: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-feather-32u4-fona.pdf
13. Hologram.[ONLINE]. Available at: http://hologram.io/dash/
14. Cellular electron datasheet. [ONLINE]. Available at: https://docs.particle.io/datasheets/cellular/electron-datasheet
15. Goblin2. [ONLINE]. Available at: http://verse-technology.com/goblin2/
16. Linkit Documentation. [ONLINE]. Available at: https://docs.labs.mediatek.com/resource/linkit-one/en/documentation
17. GPRS SIM900. [ONLINE]. Available at: https://github.com/Seeed-Studio/GPRS_SIM900
18. https://www.arduino.cc/
19. https://www.arduino.cc/en/Reference/Board
20. https://arduino-esp8266.readthedocs.io/en/latest/libraries.html
21. https://www.arduino.cc/en/Tutorial/HomePage
22. http://arduino.cc/en/Guide/HomePage
23. https://www.raspberrypi.org/
24. http://beagleboard.org/static/ti/
25. https://cdn.sparkfun.com/assets/e/e/f/8/d
26. https://www.generationrobots.com/en/content/84-pcduino-tutorial-getting-started-with-the-pcduino
27. https://www.arduino.cc/en/Main/Productshttps://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up
28. https://www.raspberrypi.org/documentation/configuration
29. https://www.raspberrypi.org/documentation/usage/
30. https://strawberry-linux.com/pub/pcDuino_UserGuide_Rev02.pdf/
31. https://learn.sparkfun.com/tutorials/programming-the-pcduino/all
32. https://pimylifeup.com/raspi-config-tool/
33. https://elinux.org/RPi_raspi-config
34. Connect your Particle devices with Ubidots
35. Connect the Adafruit FONA to Ubidots
36. Hologram and Ubidots cloud-to-cloud integration
37. Connect you Linkit One to Ubidots over GPRS
38. Connect the Arduino UNO with Ubidots using the Arduino-GPRS shield
39. Connect the GOBLIN 2 to Ubidots

IoT Middleware

CHAPTER 7

"In computing terms, children are the 'front end', Mothers are the 'backend' while fathers are the 'middleware."

-Austin Ben

OBJECTIVES

- explain the concept of a middleware
- explain middleware from an Internet of Things (IoT) perspective
- evaluate the need for an IoT middleware
- classify IoT middleware
- discuss various middleware architectures
- outline various existing IoT middleware

OUTCOMES

- identify a middleware and understand its various components
- analyze different IoT middleware architectures
- obtain a broad understanding of various middlewares that are currently popular in IoT

REVISION

Chapter 1 introduced the emergence of Internet of Things (IoT). The basic building blocks of IoT are explained. Further, the highly heterogeneous nature of the IoT systems was highlighted, and specifically the diverse types of device, network, communication, data protocols, storage systems, analytics platforms, etc. were mentioned. To be able to use all these in a unified and standardized way, so that applications can be developed easily on top of these components there is a requirement for a layer of software called the middleware. It can handle these complex interactions between the various system components and provide a uniform way for the application developers to use the underlying infrastructure without having to know their complexity. These concepts will form the background of this chapter and is focused on understanding the need for an IoT middleware, the various requirements of such a middleware, their architectures, and overview of popular middleware platforms.

7.1 INTRODUCTION TO MIDDLEWARE

The function of a middleware is to provide a connection between the applications, the network communications, and the operating system (OS). This is crucial for the proper functioning of the overall system. The primary goal of the system software (OS, system utilities, and drivers) (see Fig. 7.1) is to perform system-related tasks, while the application software is focused on a domain-specific application; its functionality; and the user interfaces. The OS interacts with the hardware through the device drivers.

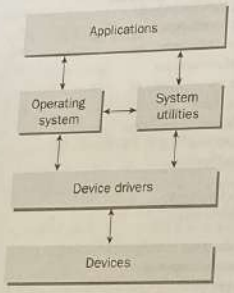


Fig. 7.1 Components of System Software

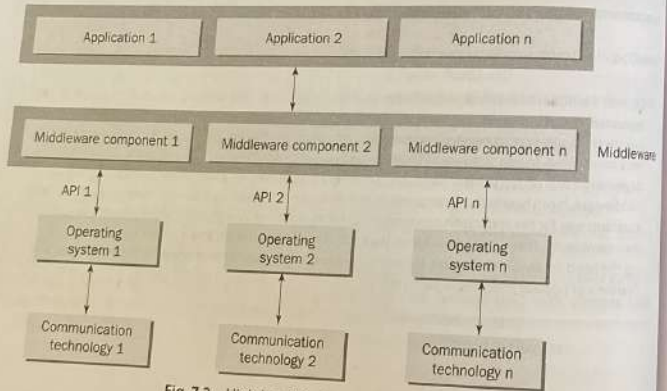


Fig. 7.2 High-level Depiction of Middleware

A middleware layer sits in between the system software and the application software (see Fig. 7.2). There are a number of technologies for communication and several OS's system utilities, device drivers, etc., which makes it hard for the applications to work with the underlying devices (i.e., hardware) in a consistent and uniform way. The middleware layer is precisely meant to address this issue. More specifically, a middleware is a software layer that has the following features:

Transparency Middleware hides from the application, the diverse hardware, OS, and communication protocols. This makes it easier for the application to have a homogeneous way to interact with the various components of the system, without the need to have a separate way to access each one of them. The middleware is designed to hide this complexity.

Flexibility, reusability, and portability The developers of the application do not need to worry about the underlying heterogeneity of the system components, but can instead focus on the application functionality, reusability (ability to change and customize specific components of the Internet of Things (IoT) system and assets to meet specific needs), and portability. This is possible because the middleware is capable of handling various conflicts between applications and things, and provides uniform, standard, and high-level interfaces that can be directly used by the software developers. Further, portability is an important issue that needs to be considered when using commercial off the shelf software, which should be capable of working with any device and in any kind of connectivity environment. Hence, platform-independent software tools have high portability.

Interoperability Various applications built for the same underlying devices (hardware) are expected to have access to a group of common services that perform various common and routine functions to reduce duplication of efforts, and also enable collaboration and interoperability between various applications. Interoperability approaches are also required to overcome the diversity of protocols, data models, configurations, etc. Interoperability is usually achieved by standards at syntactic and semantic levels (see Chapter 14).

Maintainability This characteristic reflects the level of resiliency of the various components (devices, applications, and system processes) of the IoT system. The more resilient it is, the better it will recover from failures and return to normal functioning rapidly. The middleware should be able to identify and isolate the failures and quickly be able to take remedial measures and parallely maintaining the stability of the system.

7.1.1 IoT Middleware

The middleware can be specifically developed for a particular category of applications or it can be very general in nature providing a uniform way to interact with the system. The highly heterogeneous nature of components involved in IoT applications makes it a very complex system. One of the solutions to tackle it is to use standards in each and every component of the IoT system. However, given the extreme variety of devices, manufacturers, vendors, and application domains involved in IoT, enforcing standards is difficult. Hence, there is a need for middleware that is focused on addressing various IoT specific challenges. It should provide a layer of integrating software (aka software glue) for performing all the required tasks rather than having a variety of tools from different vendors. The IoT specific needs for a middleware are described further.

7.1.1.1 Need for Middleware in IoT

IoT consists of a plethora of devices, communication protocols, transport requirements, data types, etc., which makes it extremely difficult to design and maintain a robust IoT system. There is an immense need for middleware that can aid in various functions of IoT systems such as:

Device management An IoT device should have the capability to be self-addressable, that is, making itself identifiable, so that its capabilities can be understood before it can be connected and used for a particular task. This is possible by incorporating information about the device (device name, vendor details, hardware description, software description, etc.) in metadata standard, which makes the device to announce its presence to every other device in the network. In addition, the middleware needed to provide mechanisms for registering the various devices, that is, a device registry and is used to search (syntactic and semantic approaches; refer Chapter 14 for more details) for appropriate devices and retrieve their functions and services.

Analytics The data obtained from IoT devices is highly intensive. This data needs to be analysed in two modes, that is, online and offline. The online module contains capturing, processing, and analysing data streams emanating from the device. This process is accomplished in the IoT edge computing framework (i.e., computing infrastructure at the edge of the network), where various real-time techniques for events detection and processing are provided. Methods such as complex event processing (CEP) and stream data mining techniques (see Chapter 12 for more details) are usually available on this online analytics module either on the gateways or on the cloud. The offline analytics modules are generally made available via a cloud infrastructure where services for data storage (massive repositories), machine learning-based big data analytics are provided. To achieve these functions, an IoT middleware is necessary to bridge the gap between the physical devices layer and the cloud computational infrastructure at a massive scale.

Cloud services Cloud services have emerged as the backbone for deployment of IoT systems. Currently, a variety of cloud offerings for IoT are available such as Google IoT cloud, IBM cloud, AWS IoT, and Azure IoT. Hence, the middleware is required to be flexible and be able to adapt to different cloud platforms by providing a uniform way to work with them.

Security Security and privacy play a crucial role in IoT as these systems are constantly being integrated into our daily lives. As explained in Section 7.1.2.6, it is one of the core functional requirements of the middleware as the security and privacy aspects are integrated in each and every component of the IoT system. The middleware should provide the necessary security modules for authentication and management of access to the IoT system.

The basic functional and non-functional requirements of IoT (Razzaque et al., 2016) are described in the following sections.

7.1.2 Functional Requirements of an IoT Middleware

The functional requirements are related to specific functions or behaviors of the middleware. Some of the core functions of a middleware are described in the sections below.

7.1.2.1 Resource Discovery

The intent of this requirement is to enable automatic discovery of resources based on a given context with minimal human intervention. The IoT middleware service should provide services to detect the various connections of the components and their status in real time. To make this happen, the devices should be made self-descriptive, that is, the ability to announce their presence, capabilities, and the resources it can offer.

7.1.2.2 Resource Management

Resource optimization is a prime need in IoT. Hence, the IoT middleware is required to provide services that can manage the resources and provide some crucial information about the resources such as current availability, how long a particular resource is in use, receive notification when it is released from another task and is available. Resource allocation systems should be provided, for example, a queuing system to manage the demand for a particular resource. The middleware is also responsible to resolve any conflicts that may arise for task that are competing for the same resource.

7.1.2.3 Data Management

The IoT middleware is designed to provide data level management functions such as data acquisition from devices and sensors, data related to network, and system health. Further, the middleware provides interoperability between diverse datasets generated from a variety of devices including gateways having different data protocols. Some middleware are specialized to provide data harmonization capabilities to resolve conflicts in syntactic and semantic representations of various data. The data management middleware is also expected to provide several data processing tools for filtering, aggregation of data from multiple sensor streams and also time-series data, data compression, etc.

7.1.2.4 Event Management

This component of the middleware is required to capture events from the data stream of sensors, that is, the ability to process and capture events that can be used for a particular purpose. For example, a particular pattern of the data in a given time window might represent unusual characteristics of that data and a warning may be sent to application/user for further investigation. These kind of events can be captured either in real time or on archived data based on actual values of the data as well as capturing various patterns (e.g., using CEP) from the streams of data coming from sensors in real time. Chapter 12 (Edge Analytics: Near Real-time Sensor Stream Processing) provides more details on various ways to capture and process events.

7.1.2.5 Code Management

Code management is required for accomplishing a certain pre-defined task by allowing applications and users to command a set of devices/nodes by injecting relevant codes so that they can execute the required task. It also can migrate the code to other devices/nodes and reprogram them.

7.1.2.6 Security and Privacy

The security aspects of IoT need to be incorporated in all the components of IoT system. It is extremely important to ensure that the IoT system is not compromised in any way. At the middleware level, the security- and privacy-related functions ensure that the data reaches only to those that are designated to receive it and ensure that the privacy of individuals and ownership of devices is protected. The middleware should deny access to devices and related services to all external malicious applications.

7.1.3 Non-functional Requirements of IoT Middleware

The middleware should be capable of supporting certain non-functional requirements which can help to evaluate the operation and performance of the middleware. These requirements are discussed in the following sections.

7.1.3.1 Scalability

The IoT system is said to be scalable if it can support the increasing number of devices that are added to an existing system. The middleware should be able to seamlessly integrate the new devices into the existing IoT setup and continue generating the data so that users/applications can use this data without any interruption or loss of the Quality of Service (QoS). This is achieved by loose coupling of hardware and services, so that the middleware can hide the underlying complexity and provide to the users/applications only high-level methods for interaction with it.

7.1.3.2 Reliability

Ensuring reliability implies that all the components and services from various layers of the IoT system (e.g., sensors, communication, data, etc.) are performing their designated functions in a reliable way. The middleware is expected to be resilient and continue to perform in a reliable fashion despite intermittent failures of components in the IoT system.

7.1.3.3 Availability

This requirement is tied with reliability to ensure that at all times the IoT system is available, especially for mission critical applications. It is expected that any failure is quickly recovered (i.e., resilient) and the system comes back to its original state and continues to perform its functions. The uptime of the IoT system is required to be very high and expected to be that way indefinitely.

7.1.3.4 Real-Time/Timeliness

Many IoT systems are designed for real-time dynamic applications (e.g., traffic monitoring, ambulance routing, disaster response activities, healthcare, industrial automation tasks, security, etc.). Hence, timely delivery of data is utmost important to ensure quick response to events.

7.2 ARCHITECTURES OF IOT MIDDLEWARE

The architecture of the IoT middleware is classified into different types based on the functionality and technologies used for its development. The existing middleware architectures were suitably modified to incorporate IoT's specific needs.

7.2.1 Component-based Middleware

The component-based middleware is based on Component-Based Software Engineering (CBSE). In this approach, the software is developed as a set of independent components that can communicate with each other using interfaces. This modeling approach is standardized into a component model, some of the well-known component models are: component object model (COM), COM+, .NET, Enterprise JavaBeans, etc. The main characteristics of the component-based architecture are reusability and loose coupling of the software components. These attributes make it more useful for developing systems that can be modified or extended in the future.

7.2.2 Distributed Middleware

The distributed systems are composed of software and hardware components in a networked environment, that is, these components are independent and interconnected. The middleware architecture is designed to provide distributed computing capabilities and interfaces to access these components and their functionality, but makes it look like a single coherent system (Fig. 7.3).

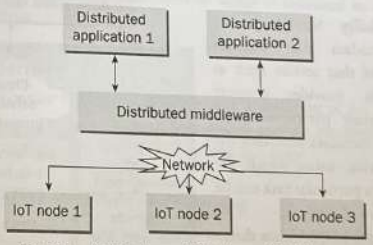


Fig. 7.3 Architecture of Distributed Middleware

This is achieved by developing higher-level programming abstractions (i.e., higher than low-level components such as sockets used for reading and writing data) and paradigms, that characterize the core principles of distributed systems. The Java Remote Method Invocation (RMI—an API that provides a mechanism to create distributed application in java) and the Common Object Request Broker Architecture (CORBA—provides interoperability among distributed objects) are the two most widely used distributed object systems.

7.2.3 Service-Oriented Middleware (SOM)

Service-Oriented Middleware (SOM) is built on the principles of Service-Oriented Architecture (SOA). SOA is a type of software architecture and development model that enables to design the software components in the form of services for interaction with each other's components, and also with external interfaces through communication calls.

Service-oriented architecture

The main characteristics of SOA are:

Loose coupling The services are designed such that they are independent and have minimal dependency on other services, that is, they are self-contained.

Independent of implementation technology The implementation of the components of the services can be done using any technology as long as the communication (request/response, pre-defined contracts) between the components is in a standard form. Also, the person using the service does not need to know the logic or implementation details, but can simply use the service by knowing how to send a request and understand the response from the service.

Service reusability The services can be developed once and used many times in several contexts. Because they are built as components, a software developer can pick and choose a service suitable for a particular task, thus reducing the development time and cost.

Service composability Several related services can be composed together to solve a bigger task. The services can be chained in a workflow kind of environment and triggered based on some other event(s) happening in the system.

Service discoverability Services are described using metadata that encodes several characteristics of that service such as identification, function, provider, request/access methods and their parameters, and response formats. This metadata describes the capabilities of the service, using which the right kind of service for a particular task can be discovered from a registry of services.

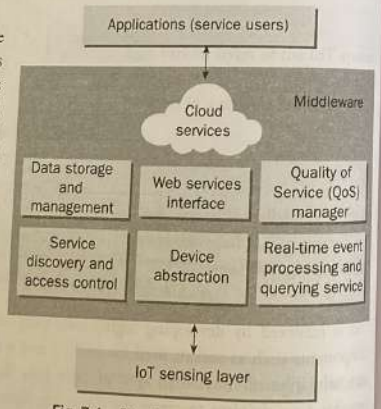


Fig. 7.4 Service-oriented Middleware

As shown in Fig. 7.4, the SOM has a three-layered architecture consisting of the sensing layer (sensors, actuators, tags, etc.), middleware components, and end-user services-based applications. This is a popular middleware architecture but requires high computational resources, and hence is not integrated with devices, but rather deployed on multiple high-end server nodes on cloud infrastructure or on gateways having sufficient computational capacity. Several core services provided by the service-oriented IoT middleware are discussed further.

7.2.3.1 Data Storage and Management

The core functionality of the data storage and management involves data generation, aggregation, preprocessing, filtering and storage, and archival. The data is generated from the sensing devices and sent in a pre-defined time interval for further aggregation. This data may need pre-processing as there can be a variety of formats and data models in which it is generated by various IoT systems. The data may have missing values or outliers due to the malfunctioning of IoT devices. There may be a need to fuse data from multiple devices to generate a new value. All these require a good amount of pre-processing on the raw data. Also, specific applications may require certain filtered data, for example, a traffic monitoring application may require data where the traffic density is above a particular threshold. So filtering techniques and approaches such as CEP (refer Chapter 12) are required. Finally, this data needs to be efficiently stored and organized, so that further querying and analysis can be performed. The difference between normal database management systems and IoT data management systems is that the IoT data management system needs to manage (i) real-time streaming data from sensors, (ii) dynamic queries on streaming data have to be handled to enable edge analytics, and (iii) massive storage and archival of IoT data for offline in-depth and intensive big data analytics.

7.2.3.2 Service Discovery and Access Control

Developing effective service discovery mechanisms in IoT is challenging because of highly heterogeneous nature of devices, communication and computation resources, data and information models, etc. Thus, one of the basic requirements to make services discoverable is to provision them with self-descriptive information, that is, a rich set of metadata. These services are often registered with a registry service, which can be based on either centralized, distributed, or decentralized registry architectures. Further, mechanisms for restricting unauthorized discovery or access to restricted services are expected to be provided by the middleware.

7.2.3.3 Quality of Service (QoS)

The middleware should ensure QoS by providing proper monitoring of the resources that are being used by various components of the IoT system. It should provide mechanisms for proper and fair allocation of resources, resolve conflicts in resource access and usage.

7.2.3.4 Real-Time Event Processing and Querying

The middleware provides data processing and analysis services at the edge of the network where real-time stream data processing is performed. The stream data processing engine can capture and analyse patterns from the streaming data and provide useful insights in real time, which is critical for many dynamic applications. See Chapter 12 for more comprehensive information on real-time event processing methods.

7.2.4 Cloud-based Middleware

The cloud middleware provides various functionalities as shown in Fig. 7.5. The cloud middleware provides a set of APIs that can be used to develop powerful cloud-based applications. The APIs that will

be available on the cloud middleware are vendor specific. Each of them will have an array of offerings ranging from database, storage infrastructure, high-performance computing nodes, powerful analytical engines, security and privacy modules, and other ancillary components. Thus, cloud-based middleware acts as a go-between the actual IoT devices and client applications that require to access them. The cloud enablement of these devices makes them accessible only through the middleware, which provides a uniform way of access and control through web services based on RESTful principles or highly specialized vendor applications (IoT cloud platforms).

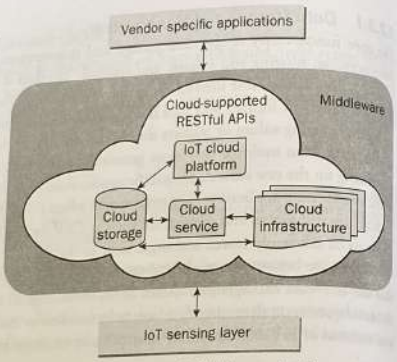


Fig. 7.5 Cloud-based Middleware Architecture

7.2.5 Node-based Middleware

Service-based architecture for IoT is not meant for software components that are embedded in the device. This is due to the low-power constraint of these IoT devices. However, the attractiveness of the service-based architecture lies in its ability to enable the development of big systems by decomposing into logically smaller independent components that are loosely coupled with each other. Therefore, in order to carry these features forward and make them work on mobile and embedded devices such as those that are frequently used in IoT system development is a challenge. Node-based architecture provides a middleware architecture that takes the best of the services-based architecture principles and combines with the

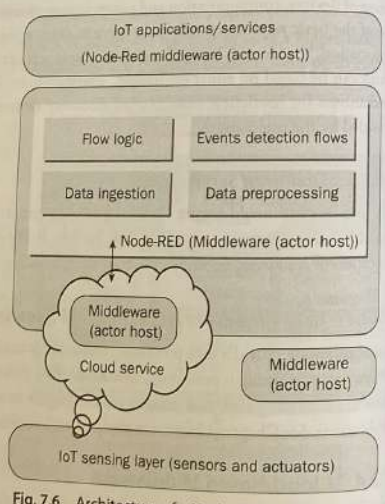


Fig. 7.6 Architecture of a Node-based Middleware. Node-based Middleware called Node-Red (refer to section 7.3.3) is Illustrated as an Example

requirements of the software in mobile/embedded devices. This type of middleware is also called as actor-based middleware. The actor host (i.e., the middleware) can be embedded at the sensing layer, cloud layer, as well as at the application layer. Only those components that are relevant to that particular layer are used in that middleware. Hence, the middleware remains light weight but is able to provide the necessary functionality (see Fig. 7.6). The main characteristics of such a middleware are simplicity (can be easily used by developers), loose coupling, flexibility, lightweight, and dynamic. The main components of the node-based architecture are:

Nodes It is a fundamental component, which can perform some fixed set of operations and can be controlled using an interface. For example, it could be a temperature-measuring node, which can make temperature measurements every 5 min and output the average temperature every 1 h. A node could be a piece of hardware such as a device in IoT or a software component in the middleware and can either reside locally or in a networked environment.

Streams The function of a stream is to act as a conduit for moving data between nodes. The node produces the stream based on some other operations in the middleware which triggers the data stream. It is the responsibility of the middleware to ensure this connection between the nodes via streams. It also determines which node produces the data stream and which consumes it.

7.3 STATE-OF-THE-ART IOT MIDDLEWARE

IoT middleware is a crucial component of IoT systems. Several commercial as well as open source entities are offering sophisticated IoT middleware solutions. Some popular IoT middleware are discussed below.

7.3.1 OpenIoT

OpenIoT is a middleware designed and maintained by the OpenIoT consortium. The middleware is developed to make it easy for deploying IoT solutions, by hiding the complexities of the various IoT system processes and allowing the user/application to connect, acquire, and process data from IoT devices easily. It provides services in a SOA-based architecture on the cloud platform to manage the various components of the IoT system. This middleware is offered as Sensing-as-a-Service (SaaS).

7.3.2 KaaIoT

Kaa is an enterprise-grade IoT platform for device management, data collection, analytics and visualization, remote control, software updates, etc. Kaa middleware is based on SOA for developing IoT solutions. The core modules of KaaIoT are:

Connectivity The popular MQTT and CoAP protocols are supported for connectivity between the platform and devices. Based on these two protocols the Kaa protocol is developed, which is open, asynchronous and has flexibility to process many kinds of messaging formats.

Device management Device management is made easy in Kaa by providing a registry for things, devices, and other components that the Kaa platform supports. The other important feature is the

ability to obtain more information about any device, so that it can be used for specific purposes. This is achieved by storing attributes (metadata) of the devices and enabling to query it. Kaa also allows storing device attributes, which provide more detailed information about any characteristic of the device. The Kaa credential management provides facilities to provision, suspend, or revoke access to devices.

Data collection Kaa has its own protocol optimized for minimal network usage for obtaining data from devices and providing information about the status of the data delivery. It also provides an easy to use protocol for collecting data from connected devices. Functionalities such as batch processing, data buffering, are also available.

Data processing and analytics Kaa provides adapters to facilitate the uploading of data to database engines or analytics systems for further processing. This connection to external systems is made easy in Kaa.

Data visualization Both real-time and historical data can be used for visualization. Kaa provides a number of widgets (charts, maps, tables, etc.) to enable the visualization of a variety of data sets from different domains.

Configuration management Modules are available for setting the parameters for device, data processing, analytics, etc. so that they can be fine-tuned.

7.3.3 Node-RED

According to node-red.org, 'Node-RED is a flow-based programming tool, originally developed by IBM's Emerging Technology Services team and now a part of the JS Foundation. It is a programming tool for wiring together hardware devices, APIs, and online services in new and interesting ways, like for the Internet of Things.' It is developed on node-based middleware architecture that incorporates streams and nodes as explained in Section 7.1.2.3. It can also be used for rapid prototyping. It supports IoT network protocols such as TCP and MQTT. The light-weight programming model of node-red is based on Node.js, which is an asynchronous event driven, non-blocking javascript runtime environment that is highly scalable. Hence, Node-RED is suited for IoT edge applications which run on the device hardware and support concurrent connections. It is also useful for cloud applications, for example, Node-RED is available on IBM cloud platform, amazon web services, etc.

7.3.4 CHOReOS

The CHOReOS middleware provides the ability to easily compose and execute complex large scale web services to solve a particular task. There are several components that CHOReOS middleware provides. Their main characteristics are:

- Facilitate the composition and execution of various services (each meant to do a different task) to solve a complex task. This is called as Xecutable Service Composition (XSC).
- Access and selection of service using the eXtensible Service Access (XSA) component of the middleware. Further, services that were developed for a particular task may be adapted for a different context. For example, a weather monitoring service originally used for precision agriculture can also be adapted for air quality monitoring (i.e., in a different context).

- Integrate heterogeneous services which are offered using different technologies such as client/server, publish/subscribe, and tuple space.
- Discover required services via a service lookup modules termed as eXtensible Service Discovery (XSD).
- Manage the large number of things involved in the IoT system and be able to discover specific things and the corresponding service.
- Provision of high-end computing service to perform intensive big data analytics.

7.3.5 Linksmart

The Linksmart IoT middleware (known earlier as Hydra) is based on the Linksmart historical data store (HDS) middleware platform which is built on REST-based architecture for communication with server. This middleware is completely based on well-known standards for services and systems such as OGC SensorThings, MQTT, REST, TLS, etc. and features such as pub/sub, device abstraction, data storage, live data management, stream mining at the edge or in the cloud, online machine learning, fast built-in visualizations, etc.

7.3.6 CarrIoT

It is an IoT middleware for primarily supporting M2M communication systems. It is designed as a Platform as a Service (PaaS) on the cloud in a SOM architecture. It provides end-to-end functionality ranging from device access and management, data collection, storage, data management, and security. It provides specific methods for development of user interface of a custom application. This middleware enables the exchange of data with other data storage and management systems and APIs (e.g., ERP, Dropbox, etc.). However, the security aspects are not built into this product at the storage level, and also lack data standardization and interoperability.

7.3.7 Oracle Fusion Middleware

It is a middleware stack which provides a highly scalable IoT services infrastructure having the ability to integrate various functions such as monitoring, management, analytics, and security into IoT applications. The cloud middleware also provides the capabilities for mediating and managing the interactions between various applications. In addition, the middleware allows the integration with Business Process Management (BPM) tools to manage business processes that includes devices, applications, and humans in the loop. The Oracle business activity monitoring dashboard provides the visualization of real-time data from devices and event patterns captured by the Oracle event processing engine. The middleware also provides support for authentication, authorization, and identity management to ensure that the communication between devices, gateways, various services, and applications can communicate with each other in a secure way.

7.3.8 Google Cloud IoT

It is based on the SOM architecture and runs as a PaaS on the cloud. It provides functionality for connecting to the devices, acquiring data from the devices and storing it. In addition, the archived data

can be used to perform analytics (e.g., machine learning) in the offline mode on the cloud. The online component of this middleware platform allows capturing and analysing data in real-time using edge analytics. It can be easily integrated with Google Maps to build location aware IoT applications such as logistics and supply chain management, Smart cities, etc. The Xively middleware that was separately developed for IoT was acquired and integrated with Google Cloud IoT platform.

SUMMARY

This chapter introduced the concept of a middleware and put it in the context of IoT. The need for an IoT middleware is presented. The middleware forms an important module in the overall IoT system. The basic functional requirements of the middleware such as resource discovery, resource management, data management, event management, and code management are described. These give an overview of the required characteristics that a middleware architecture should ideally possess. Further, the IoT middleware should also address non-functional requirements such as scalability, reliability, availability, timeliness, security and privacy. Having understood the requirements of the middleware, an overview of various middleware architectures are described highlighting their core technology aspects. Finally, the currently popular middleware platforms (both open source and commercial) are presented.

REVIEW QUESTIONS

1. Define a middleware.
2. What are the main features of a middleware?
3. What is IoT middleware?
4. Explain the need for a middleware in IoT.
5. Describe the functional requirements for IoT middleware.
6. Describe the non-functional requirements of IoT middleware.
7. Explain component, distributed, service-based, cloud-based, node-based architectures of IoT middleware.
8. Describe the various state-of-the-art IoT middleware.
9. Write a comparison between all three types of middleware architectures.

REFERENCES

1. Amaral, L.A., de Matos, E., Tiburski, R.T., Hessel, F., Lunardi, W.T., Marczak, S. (2016), Middleware technology for IoT systems: Challenges and perspectives toward 5G. In: Mavromoustakis C., Mastorakis G., Batalla J. (Eds.) *Internet of Things (IoT) in 5G Mobile Technologies. Modeling and Optimization in Science and Technologies*, vol. 8, Springer, Cham.
2. Ngu, A.H., Gutierrez, M., Metsis, V., Nepal, S., Sheng, Q.Z. (2017), "IoT middleware: A survey on issues and enabling technologies," *IEEE Internet of Things Journal*, 4(1): 1–20.
3. Razzaque, M.A., Milojevic-Jevric, M., Palade, A., Clarke, S. (2016), "Middleware for Internet of Things: A survey," *IEEE Internet of Things Journal*, 3(1): 70–95.
4. Oliveira, A.M., Melo, D.F., Silva, G.M., Gregório, T. (2016), "Comparing IoT platforms under middleware requirements in an IoT perspective an fulfilment analysis of IoT middleware requirements in IoT platforms that uses REST to communicate with external applications", In: *UBICOMM 2016: The Tenth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*.
5. Spiess, P., Karnouskos, S., Guinard, D., Savio, D., Baecker, O., Souza, L., Trifa, V. (2009), "SOA-based integration of the internet of things in enterprise services". In: *IEEE International Conference on Web Services (ICWS)*, pp. 968–975, Los Angeles.
6. Abu-Elkheir, M., Hayajneh, M., Ali, N.A. (2013), "Data management for the internet of things: Design primitives and solution", *Sensors*, 13(11): 15582–15612.
7. Google IoT cloud: https://cloud.google.com/solutions/iot/ [last accessed: 28/08/2019]
8. Linksmart: https://linksmart.eu/ [last accessed: 28/08/2019]
9. Satyadevan, S., Kalarickal, B., Jinesh, M. (2015), Security, trust and implementation limitations of prominent IoT platforms, *Proc. Front. Intell. Comput. Theory Appl. (FICTA'14)*, 328: 85–95.
10. Oracle Fusion Middleware: http://www.oracle.com/us/solutions/machine-to-machine/iot-wp-2190408.pdf [last accessed on 28/08/2019]
11. Da Cruz, M.A.A., Rodrigues, J.J.P.C., Sangaiah, A.K., Al-Muhtadi, J., Korotaev, V. (2018), "Performance evaluation of IoT middleware", *Journal of Network and Computer Applications*, 109: 53–65.
12. OpenIoT Consortium. OpenIoT—Open Source Cloud Solution for the Internet of Things. [Online]. Available: http://www.openiot.eu/ [last accessed: 28/08/2019]
13. CHOReOS: Integrated CHOReOS middleware—Enabling large-scale, QoS-aware adaptive choreographies [https://hal.inria.fr/hal-00912882/document]
14. Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D. (2014), "Context Aware Computing for the Internet of Things: A Survey", *IEEE Communications Surveys & Tutorials*, 16(1): 414–454.
15. Hartikainen, V.-M, Vulli, M, Järvinen, H.-M. (2009), "Node based architecture for lightweight middleware", *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae. Sectio Computatorica*. 30.
16. Criado, J., Asensio, J.A., Padilla, N., Iribarne, L. (2018), "Integrating cyber-physical systems in a component-based approach for smart home," *Sensors*, 18: 2156.
17. Cruz-Piris, L., Rivera, D., Marsa-Maestre, I., de la Hoz, E., Velasco, J.R. (2018), "Access control mechanism for IoT environments based on modelling communication procedures as resources". *Sensors*, 18: 917.
18. Rambold, M., Kasinger, H., Lautenbacher, F., Bauer, B. (2009), "Towards autonomic service discovery: A survey and comparison", In: *2009 IEEE International Conference on Services Computing, Bangalore*, pp. 192–201.
19. Farahzadi, A., Shams, P., Rezazadeh, J., Farahbakhsh, R. (2017), "Middleware technologies for cloud of things—A survey". *Digit. Commun. Networks*.
20. Da Cruz, M.A., Rodrigues, J.J., Sangaiah, A.K., Al-Muhtadi, J., Korotaev, V. (2018), "Performance evaluation of IoT middleware", *Journal of Network and Computer Applications*, 109: 53–65.
21. Al-Jaroodi, J., Mohamed, N., Jiang, H., (2003), "Distributed systems middleware architecture from a software engineering perspective", In: *Proceedings Fifth IEEE Workshop on Mobile Computing Systems and Applications, Las Vegas*, NV, USA, pp. 572–579.
22. Marques, G., Garcia, N., Pombo, N. (2017), "A survey on IoT: Architectures, elements, applications, QoS, platforms and security concepts". In: Mavromoustakis, C., Mastorakis, G., Dobre, C. (Eds.), *Advances in Mobile Cloud Computing and Big Data in the 5G Era. Studies in Big Data*, vol. 22, Springer, Cham. https://nodered.org/ [last accessed: 28/08/2019]

CHAPTER 8

IoT Software Platforms

"Open platforms historically undergo a lot of scrutiny, but there are a lot of advantages to having a open source platform from a security stand point."

- Sundar Pichai

OBJECTIVES

- explain the need for an IoT platform
- outline the fundamental characteristics of IoT platforms
- explain the various functionalities of the IoT platform
- summarize the usefulness of the IoT platform from various perspectives
- outline most popular commercial IoT platforms and their applications
- explain selected open source IoT platforms
- show with an example how open source IoT platform can be used
- explain the selection of an IoT platform for a given goal

OUTCOMES

- justify why an IoT platform is required
- evaluate the characteristics, usefulness, and core functions of an IoT platform
- compare various commercial and open source IoT platforms
- choose the right IoT platform for a specific application

REVISION

Chapter 7 introduced the concept of IoT middleware that can handle complex interactions between the various IoT system components and provide a uniform way for the application developers to use the underlying infrastructure without having to know their complexity. This chapter extends these concepts and introduces the IoT platforms that are currently popular both commercial and open source. The challenges involved in choosing of appropriate platform are discussed. A hands-on tutorial on using an open source IoT platform is presented.

8.1 INTRODUCTION TO IOT SOFTWARE PLATFORMS

The IoT software platforms are designed to ease the deployment and management of IoT applications in various domains. Using these platforms, an IoT solutions' provider can build faster, cheaper, and stable IoT systems.

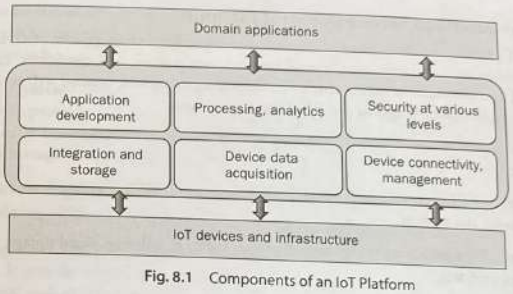


Fig. 8.1 Components of an IoT Platform

8.1.1 Need and Characteristics of IoT Platforms

An IoT platform is a suite of components as shown in Fig. 8.1. Each of these components addresses a particular need for the development of IoT systems. Hence, the essential characteristics of IoT platforms are:

- Device connectivity and management, which involves various functions such as management of large number of diverse types of devices, device registration to a centralized registry service, ability to make and sustain connections on a large scale in a variety of communication and network protocols, management of security and privacy aspects (e.g., access, credentials, certificates, policies, etc.).
- Acquiring the data from a wide range of devices requires some fundamental understanding and transformations, for instance acquisition of physical/analog signal from a sensor and converting it into a digital form, and setting of various acquisition parameters such as sampling rate and resolution.
- Integration of data from devices of various types (data sources) by implementing standards at various layers of the IoT system thus enabling interoperable systems development.
- Requirement of data in the stream mode for real-time analytics. Reading and writing large volumes of data. Storage of data in various data models, databases (e.g., NOSQL), cloud-based storage, etc. (see Chapter 7 for more details), indexing and archiving for offline analytics.
- Data preprocessing, data cleansing to address uncertain and incomplete data, resampling, data transformations, quality of the data.
- Real-time and offline analytics to extract meaningful information and knowledge from the data.
- Support for enabling the development of domain-specific applications, client-user interfaces, visualization environments, etc.

8.2 COMMERCIAL IOT SOFTWARE PLATFORMS

The IoT software platform market is highly competitive, with many commercial vendors offering products that provide sophisticated tools for end-to-end deployment and management of the IoT systems. It is projected that it will grow at a compound annual growth rate (CAGR) of 28.8% between 2019 and 2024. (Reuters, 2019)

8.2.1 Amazon Web Service (AWS) IoT Platform

Amazon IoT platform is a cloud-based platform that enables efficient management of devices remotely. Device metadata such as name, type, manufacturer, access policies, and certificates, can be added quickly and made available as service due to its tight integration with amazon web services (AWS). AWS IoT provides complete end-to-end solutions ranging from:

Device level software Such as IoT operating system for microcontrollers and software for edge connectivity.

Control services These enable device management, connecting devices with web services to develop IoT applications, and security.

Data services Edge processing to capture and process events in real-time, cloud storage, and machine learning-based analytics.

8.2.2 Bosch IoT Suite

The Bosch IoT platform consists of a suite of tools that are offered in the form of Platform-as-a-Service (PaaS). It is built on the eclipse IoT components. The Bosch IoT suite is a comprehensive toolbox for IoT developers, which enables the development of end-to-end IoT solutions in residential, mobility, and industrial domains (Fig. 8.2). The foundation of the Bosch platform is based on the open source technology, which enables it to support easy integration of other platforms and services.

The core components of the Bosch IoT suite are:

Device connectivity The Bosch IoT provides the necessary tools for connecting devices to the cloud in a secured manner. It provides device-to-cloud and cloud-to-device communications such as telemetry, events capture, and command and control of the devices. Further, support for gateways, multiple protocols (MQTT, HTTP, LoRaWAN), identity management, and security are provided.

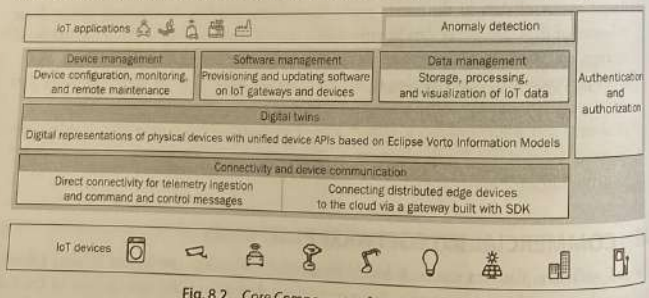


Fig. 8.2 Core Components of Bosch IoT Suite

(Source credit: "Picture: © Bosch.IO")

Digital representations of physical things Digital twins are the digital representation of the things so that they can interact with each other. The applications will easily be able to manage the device assets using this approach.

Device management The IoT remote manager, which is available as a cloud service, provides multiple device protocols and complete management of the devices throughout its lifecycle. Remote configuration, firmware updates, monitoring, diagnostics, backup, etc. are some of the functionalities of the device management component of the platform.

Software provisioning Bosch IoT rollouts provide the necessary functionality both at small and large scales for updating of the software on devices, controllers, and gateways.

Data management and analytics This is supported on the cloud by the Bosch IoT insights component. It provides support for data storage, preprocessing, analytics, and visualization. Third-party analytical tools such as Matlab, Excel, and Tableau can be easily connected.

User and permissions management User management, authorization management, etc. are handled by the Bosch IoT permissions module.

8.2.3 EVRYTHNG

EVRYTHNG is an IoT Software-as-a-Service (SaaS) platform for consumer products. The concept of active digital identities is introduced by this product, which give products digital identity that can be used for developing applications that are driven by the products.

The adaptive analytics component provides machine learning tools that can be applied on real-time product data, thus enabling real-time services. The block-chain integration gives these services robust integrity. Further, the ability to easily integrate with Customer Relationship Management (CRM) and Data Management Platform (DMP) products, makes it easy for businesses to operate and give custom service to the consumers. The digital identity is managed by GS1 link support (standard for web-enabling barcodes) in addition to securing product data.

8.2.4 IBM Watson IoT Platform

The IBM Watson IoT platform built on the IBM cloud enables to communicate and acquire data from connected devices and gateways (Fig. 8.3). The device management component provides functions such as device registration, rebooting or updating firmware, receive device diagnostics and metadata, or perform bulk device addition and removal. The connectivity between devices and applications is provided by the standard MQTT protocol, which is highly efficient for real-time exchanging of data between the devices. In addition, it uses TLS to secure all communication between devices and services. Further, the platform provides cloud-based storage, analytics, and rapid visualization.

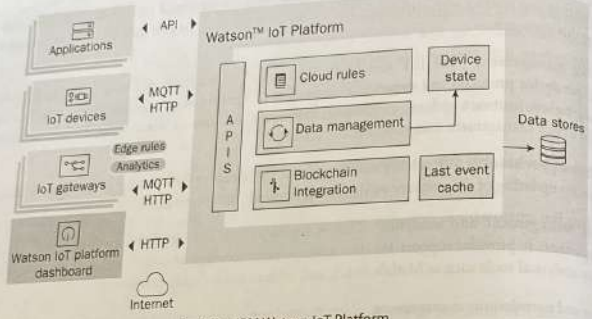


Fig. 8.3 IBM Watson IoT Platform

(Source credit: Courtesy of International Business Machines Corporation, © International Business Machines Corporation)

8.2.5 Cisco Kinetic IoT

The Cisco Kinetic IoT platform is built on three major components (see Fig. 8.4):

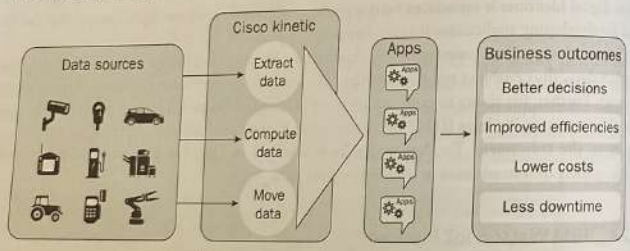


Fig. 8.4 Cisco Kinetic IoT Platform

(Source credit: Cisco)

Extract data This component of the platform can extract data from a variety of devices and is independent of the underlying protocol that the device uses. The data is further processed in a form that is readily usable by applications.

Compute data This component is for making computations at the edge/fog where real-time capture of streaming data and processing for capturing interesting events is performed. This is achieved by running complex rules on streaming data to reduce, compress, normalize, and transmit data in optimal ways. The analytics can also be performed on stored/archived data and on the cloud.

Move data This component facilitates cloud-based applications to consume data when required in a timely and secure manner, programmatically to get the right data to the right applications at the right time. The data can be moved across multiple cloud platforms, locations, etc. at the same time the security of it is not compromised.

8.2.6 Google IoT Cloud

The Google IoT cloud has a set of core IoT components for various functions of the IoT system (see Fig. 8.5).

Through the device manager tool, the configuration of the devices can be performed and also managed securely. The device identity management, role-level access control, etc. are also handled by this component. Further, massive deployment of devices automatically is enabled by REST APIs.

The heterogeneity of the protocols is handled by the protocol bridge component of the platform that allows secure connection of various standardized protocols such as MQTT and HTPP and makes it as a unified system. The publish/subscribe model allows the data to be easily used by other applications in the pipeline.

The platform provides tools for edge processing on both edge devices as well as gateways. These are Machine learning-based tools for developing models that can do real-time predictions on the stored data. In addition, various analytical tools such as Google Big Data Analytics, Dataflow, BigQuery, Bigtable, ML, Data Studio, etc. are available.

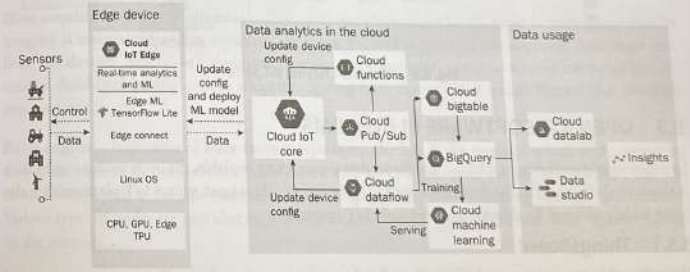


Fig. 8.5 Architecture of Google IoT Cloud

8.2.7 Microsoft Azure IoT Suite

The Microsoft Azure IoT suite consists of a set of Azure services for connecting 'Things' (device connectivity), performing streaming analytics, efficient storage, device metadata management, etc.

As shown in Fig. 8.6, the IoT hub can translate multiple protocols such as HTTP and AMPQ, as well as MQTT via a free, open source MQTT protocol gateway. It also provides security on per device basis.

The stream analytics components support writing queries in a simple SQL-like language so that the data can be analyzed in real time. The various windowing operations are supported to enable capture of specific portion of the data stream and run some logic/rules on it and consequently capture events from data coming from multiple devices.

A web application is provided to interact with the devices and manage the response to events captured by stream analytics by sending text messages based on some preset threshold values on the device measurements.

A logic app is provided to enable the development of workflows. This allows automating a series of steps involved in existing business process (i.e., end-to-end) by connectors that can execute certain rules and perform some actions.

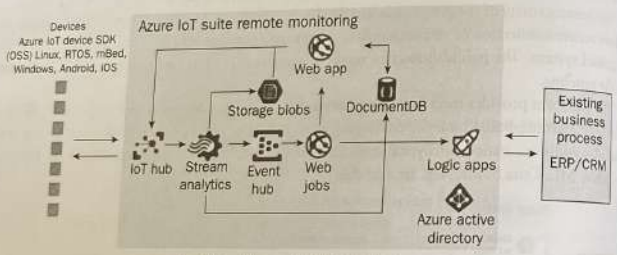


Fig. 8.6 Microsoft AZURE IoT Suite

8.3 OPEN IOT SOFTWARE PLATFORMS

In addition to several commercial IoT platforms that are currently available, the open source community is also actively involved in the development of IoT platforms. These open source IoT platforms are also being widely adopted for developing large scale IoT systems.

8.3.1 ThingsBoard

ThingsBoard is an open source IoT platform for device management, data collection, processing, and visualization. It provides various functionalities as mentioned below:

Device provisioning and management All kinds of IoT entities can be deployed, monitored, and managed using the server side tools available in the component of the platform. Security is ensured in all these processes. Remote Procedure Calls (RPC) are supported, which allow commands to be sent to devices and receive the results. Further, a novel way of connecting device, assets, customers, etc. is available such that a hierarchical arrangement of these based on their relationships can be developed. For example (see Fig. 8.7), air quality and traffic monitoring stations are deployed in five different zones (assets) of a city, and in each zone (asset), between four and eight stations of each are deployed. Further, each station (device) has some sensors associated with it.

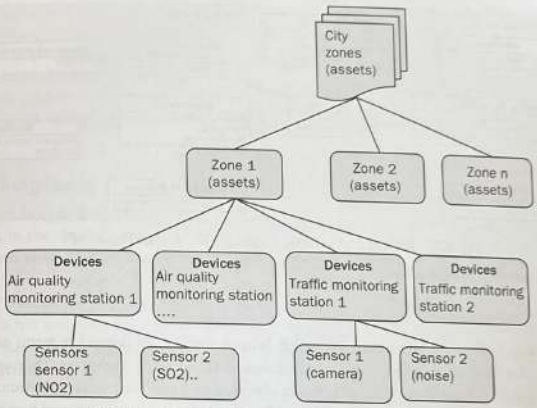


Fig. 8.7 Hierarchy of Assets and Devices in ThingsBoard

Data acquisition and visualization Data from the devices can be acquired in such a way that its integrity is maintained and in a fault-tolerant way. An API is provided for acquiring time-series data from the devices. Several IoT dashboard templates are available, using which the users can quickly create a dashboard having custom widgets (selecting from the provided widget library) that suit their requirements.

Rules engine It consists of three main components: the first is the message, which is any incoming data from various devices, events, REST-based web service-generated event, RPC request, etc. The second component is the rule node that allows the processing of an incoming message by a function. Various types of nodes are provided each of which can execute a specific function (filter, transform, etc.) on the message.

8.3.2 OGC SensorThings

The OGC SensorThings is a REST-based API that provides 'an open, geospatial-enabled, and unified way to interconnect the Internet of Things (IoT) devices, data, and applications over the Web (OGC SensorThings, 2019).' It is a standardization done at the data and interface of IoT and Web of Things (WoT). The core benefits of such a standardized approach are:

(1) it permits the proliferation of new high value services with lower overhead of development and wider reach, (2) it lowers the risks, time, and cost across a full IoT product cycle, and (3) it simplifies the connections between devices-to-devices and devices-to-applications.

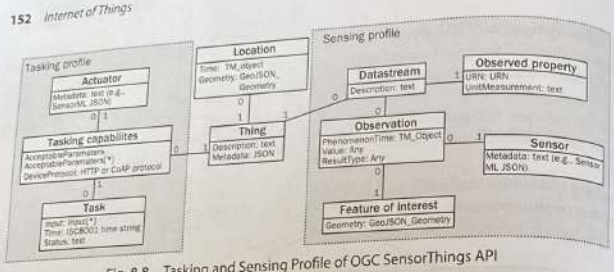


Fig. 8.8 Tasking and Sensing Profile of OGC SensorThings API

(Source credit: Open Geospatial Consortium)

8.3.2.1 *Sensing*

A shown in Fig. 8.8, a thing has a property called *location* that can be defined in terms of space and time. It can also have *data stream(s)* that are understood in terms of *observation* (having a time of observation, value of observation, etc.) of a certain observed property (i.e., sensing environment) such as temperature of a feature of interest (e.g., room). The actual observation is obtained by a sensor (e.g. thermometer). Basically, the goal of the sensing profile is to provide a standardized way to manage and retrieve observations and metadata from heterogeneous IoT sensor systems.

8.3.2.2 *Tasking*

The goal of the tasking profile is to provide multiple tasking capabilities that can be executed by an actuator. The applications can get full control of the IoT devices using which, remote control of sensors is possible. For example, custom acquisition of data can be tasked to sensors remotely based on certain triggers or new requirements/on demand. Before sending a request to execute a particular task on the sensor, it is possible to request for the tasking capabilities of that particular sensor and also the way to request for tasking. This is achieved through the *TaskingCapabilities* component (see Fig. 8.8).

8.3.3 Thinger.io

It provides scalable cloud infrastructure to connect, use, and manage large number of devices and also a REST API for integrating with users' own applications (e.g., WebServices, mobile phones, or desktop applications). Many hardware platforms such as Arduino compatible hardware (Arduino + Ethernet, Arduino + WiFi, ESP8266, NodeMCU, TI CC3200, etc.), Raspberry PI, Sigfox, ARM MBED, can be programmed to connect and use the Thinger.io platform. It provides a mobile app to manage different elements of the Thinger.io platform using a smartphone.

8.3.4 SiteWhere

SiteWhere is an industrial strength open-source application enablement platform for IoT. It provides a multitenant microservice-based infrastructure that includes the key features required to build and deploy IoT applications.

It uses Kubernetes (open-source system for automating deployment, scaling, and management of containerized applications) as the infrastructure platform, allowing it to be deployed in any cloud service platforms such as Google cloud and AWS. The core concept of SiteWhere is microservice, which is a specialized service for each core component (e.g., event ingestion, big-data event persistence, device state management, etc.) of the IoT system. Further, each microservice is a Spring Boot (create stand-alone, production-grade spring-based applications) application wrapped as a Docker (container software) container.

8.3.5 ThingSpeak

It is an Open IoT platform with Matlab analytics. It is a free web service that lets you collect and store sensor data in the cloud. It also provides apps using which various analytics on the sensors data can be performed using Matlab and also perform visualization. It supports data acquisition from several hardware platforms such as Arduino, Raspberry Pi, and BeagleBone Black. The platform uses REST and MQTT APIs for data management functions. Alerts can be obtained based on the events, for example, if the data is not received continuously within a certain time period (preset), an alert is generated and sent to the user called as a *ThingTweet*. Also, using the *TalkBack* function, commands can be sent to the sensors to perform certain task based on the events that are generated.

8.4 CHOOSING AN IOT PLATFORM

Given the availability of IoT platforms both commercial and open source, choosing an IoT platform is rather challenging. Following are some general guidelines for selecting an IoT platform.

8.4.1 Domain of Application

The application in which the intended IoT platform will be deployed is crucial to consider. It is possible that some of the IoT platforms are not general enough to be applicable in any domain, they may be designed specifically for certain application areas, for example, industrial, healthcare, etc. Hence, it is suggested to look at the capabilities in terms of acquisition of data from a large variety of device types, communication, network, and data protocols. The specific needs of the application domain have to be identified and matched with the functionality offerings or the scope of the use cases that the IoT platform supports.

8.4.2 Usability

The ease of use of an IoT platform needs to be assessed beforehand. The user-friendliness of the IoT platform helps in performing essential tasks, using interfaces and functions for various protocols, visualization, decision-making, etc. Further, considerations are in terms of ease of provisioning and management of devices.

8.4.3 Interoperability

The ability of the IoT platform to integrate third-party solutions, ability to assimilate data coming via heterogeneous communication, network and data protocols, open source tools, ERP tools, existing business processes, work flows, etc., makes it a highly interoperable IoT solution.

8.4.4 Scalability

It is the ability to handle a large number of IoT devices/end points. A truly scalable IoT solution can handle an unlimited number of devices and their connections seamlessly by approaches such as load balancing and distributed processing. Hence, it is important to check for the number of IoT devices that can be handled, not only in terms of connectivity, but also further processing at the edge, storage and subsequent deep analytics and visualization capabilities.

8.4.5 Edge Analytics

It is necessary to find out if the IoT platform supports edge analytics, if your intended application is all about providing real-time information to the customers. For example, real-time traffic updates, energy estimations in a building, industrial processes, retail, etc. which need stream data processing capabilities. Hence, choose an IoT platform that provides these capabilities, particularly those that can be set to trigger alerts/warnings based on predefined rules/criteria that run on a subset of the incoming stream using various windowing techniques.

8.4.6 Security

The capability of ensuring high-level of security end-to-end (all levels of the IoT stack) by the IoT platform plays a major role in its selection. The IoT platform needs to be assessed for comprehensive security tools for secure authentication, certification, encryption, identity-based authentication for devices, Cloud-based gateways offering SSL or DTLS encryption, etc.

8.4.7 Recovery

In the event of instability or a crash of the IoT system, how resilient is the system to come back to its original condition is an aspect that needs to be given importance while selecting an IoT platform. The IoT platform needs to provide the ability to have enough redundancy such that failures can be handled smoothly. The data recovery ability is another attribute to pay close attention. The IoT platform should be able to take periodic backups and restore the data on demand.

8.5 HANDS-ON USING AN IOT PLATFORM

In this section, the ThingsBoard IoT platform that is described in Section 8.3.1 is used to demonstrate the usefulness of an IoT platform. Refer to Box 8.1 for step-by-step instructions.

Goal The goal of the hands-on exercise is to be able to connect a soil moisture and soil temperature sensor to a microcontroller (NodeMCU) and use the ThingsBoard IoT platform to send information via Wi-Fi on to the ThingsBoard dashboard for visualization of the live readings from the sensors.

BOX 8.1: SENSORS AND THEIR CONNECTIONS TO A MICROCONTROLLER AND THINGSBOARD IOT PLATFORM

Connecting Soil Moisture and Soil Temperature Sensors to NODEMCU and ThingsBoard Platform

Soil moisture sensor: The DFRobot soil moisture sensor measures soil moisture levels by capacitive sensing. It is made of corrosion resistant material which gives it an excellent service life (Fig. 8.9) .

Fig. 8.9 Soil Moisture Sensor

(Source credit: DFRobot)

Soil temperature sensor: The soil temperature sensor that is used in this exercise is Dallas Semiconductor DS18B20 which is a digital thermometer providing 9- to 12-bit (configurable) temperature readings which indicate the temperature of the device (Fig. 8.10).

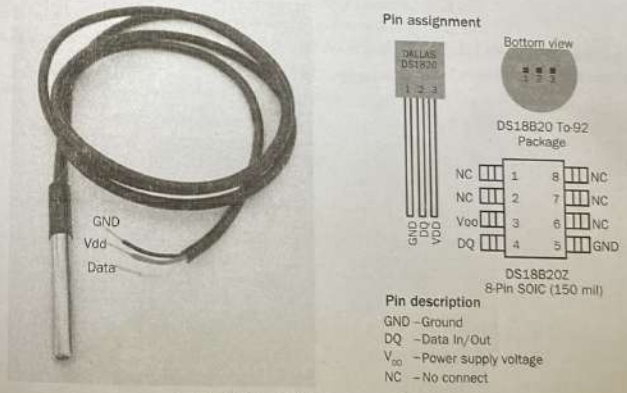


Fig. 8.10 Soil Temperature Sensor

NodeMCU: NodeMCU is an open source LUA-based firmware developed for ESP8266 Wi-Fi chip. It can be programmed directly through USB port using LUA programming or Arduino IDE. NodeMCU has a programmable Wi-Fi module (Fig. 8.11).

(Contd)

Box 8.1 *(Contd)*

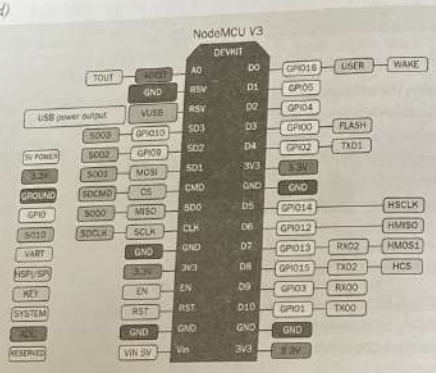


Fig. 8.11 Pinout Diagram of NodeMCU

Wiring schemes: Figures 8.12(a) and (b) show the initial and final wiring schemes of soil moisture and soil temperature sensors with NodeMCU.

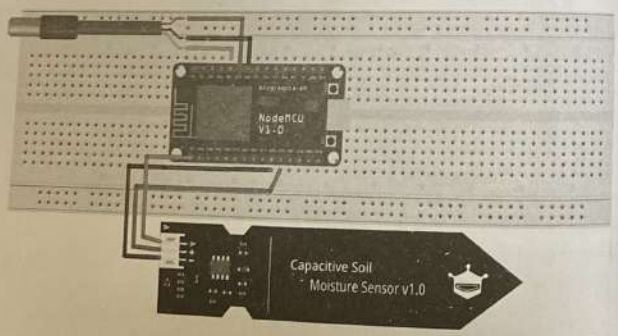


Fig. 8.12(a) Programming/Flashing Schema (Initial)
(Source credit: This image was created by Fritzing)

(Contd)

Box 8.1 *(Contd)*

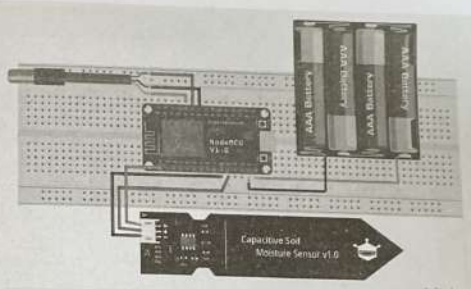


Fig. 8.12(b) Final Schema (Battery Powered)

(Source credit: This image was created by Fritzing)

Step 1: Refer to Section 8.3.1 for a review of the ThingsBoard IoT platform.
Step 2: Complete the setup of the soil moisture and soil temperature using NODEMCU.

Thingsboard Configuration

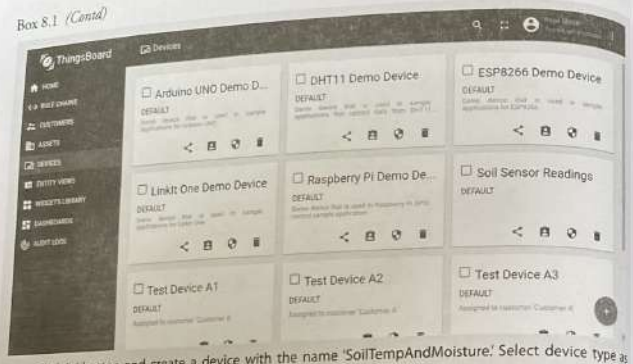
Provision your device: Using ThingsBoard public server: https://demo.thingsboard.io

- Enter login username and password

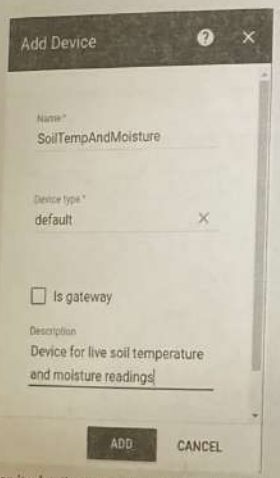
- After logging in, enter 'Devices' section

(Contd)

Box 8.1 *(Contd)*



- Click '+' button and create a device with the name 'SoilTempAndMoisture.' Select device type as 'default'



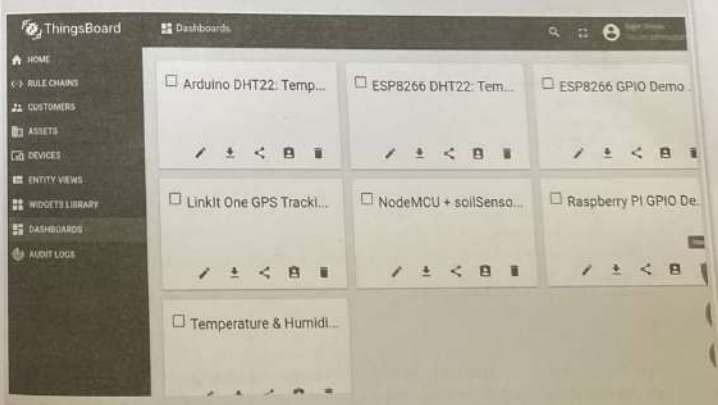
- Once device is created, open its details and click 'Manage credentials'
- Copy the auto-generated access token from the 'Access token' field. Please save this device token. It will be referred to later as $ACCESS_TOKEN

(Contd)

Box 8.1 *(Contd)*

- Click 'Copy Device ID' in device details to copy your device id to the clipboard
- Paste your device id to some place; this value will be used in further steps

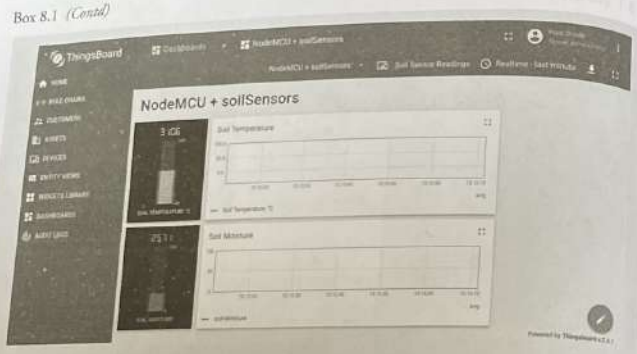
Provision your dashboard:



- Use import/export instructions to import the dashboard to your ThingsBoard instance

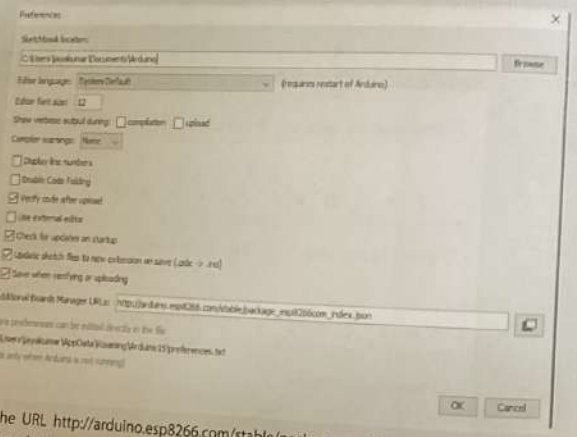
(Contd)

Box 8.1 *(Contd)*



Programming the NodeMCU:

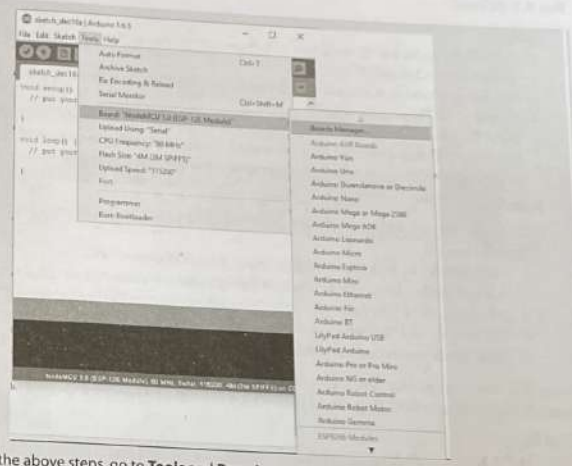
- Firstly, open the Arduino IDE
- Go to files and click on the preference in the Arduino IDE



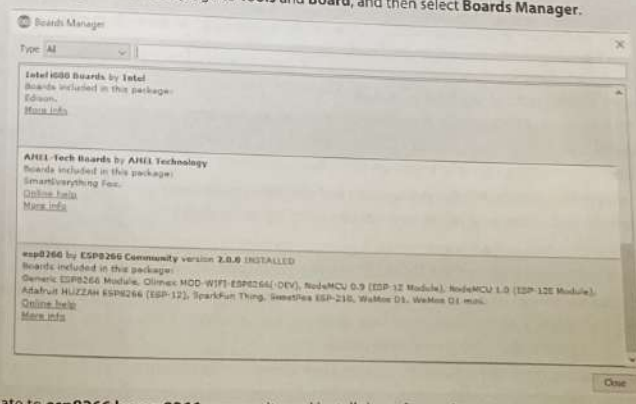
- Copy the URL http://arduino.esp8266.com/stable/package_esp8266com_index.json in the Additional Boards Manager
- Click OK to close the preference tab

(Contd)

Box 8.1 *(Contd)*



After completing the above steps, go to **Tools** and **Board**, and then select **Boards Manager**.



Navigate to **esp8266 by esp8266** community and install the software for Arduino.

Once all the above process has been completed, we are ready to program our **esp8266 with Arduino IDE**.

(Contd)

Box 8.1 *(Contd)*

Step 1:

- In the Tools menu 'Board,' some new boards are added after performing the above steps
- Select 'NodeMCU x.y (ESP12 module)'
- Prepare your hardware according to the programming/flashing schema as shown earlier
- Connect USB-TTL adapter with PC
- In the Tools menu, select the corresponding port of the USB-TTL adapter. Open the serial monitor (by pressing CTRL-Shift-M or from the Tools menu). Set the key emulation to 'Both NL & CR' and the speed to 115200 baud. This can be set in the bottom of terminal screen

Step 2: Install libraries

- Open Arduino IDE and go to Sketch → Include Library → Manage Libraries. Find and install the following libraries:
  - PubSubClient by Nick O'Leary (http://pubsubclient.knolleary.net/)
  - Adafruit Unified Sensor by Adafruit (https://github.com/adafruit/Adafruit_Sensor)
  - DHT sensor library by Adafruit (https://github.com/adafruit/DHT-sensor-library)
  - One Wire by Paul Stoffregen et al. (https://github.com/adafruit/DHT-sensor-library)
  - Dallas Temperature (https://github.com/milesburton/Arduino-Temperature-Control-Library)

Note: This hands-on tutorial was tested with the following versions of the libraries: PubSubClient 2.7.0, Adafruit Unified Sensor 1.0.2, DHT sensor library 1.3.4, One Wire Library 2.3.4, and Dallas Temperature 3.8.0.

Step 3:

Note: For the soil moisture sensor to send proper readings, it has to be calibrated first in air and water. Use the sketch (Code listing shown in Box 8.2) before proceeding with the next steps.

- Copy and paste the soilSensorThingsboard.ino sketch (Code listing as shown in Box 8.3)
- Edit following constants and variables in the sketch:
  WIFI_AP—name of your access point
  WIFI_PASSWORD—access point password
  TOKEN—the $ACCESS_TOKEN from ThingsBoard configuration step
  ThingsboardServer—ThingsBoard HOST/IP address that is accessible within your Wi-Fi network. Specify "demo.thingsboard.io" (if you are using live demo server)
- Connect NodeMCU to PC via a debug cable and select the corresponding port in Arduino IDE
- Compile and upload your sketch to the device using 'Upload' button

After application will be uploaded and started, it will try to connect to ThingsBoard node using MQTT client and upload 'soil Temperature' and 'soil Moisture' time series data once per second.

Autonomous operation: When you have uploaded the sketch, you may remove debug cable required for uploading and connect your NodeMCU, soil moisture and soil temperature sensor directly to the power source according to the final wiring schema as shown above.

Data visualization: Open ThingsBoard Web UI. You can access this dashboard by logging in as a tenant administrator using:

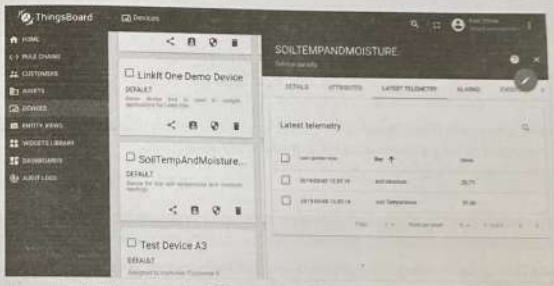
- Local ThingsBoard installation
  - login: tenant@thingsboard.org
  - password: tenant

(Contd)

Box 8.1 *(Contd)*

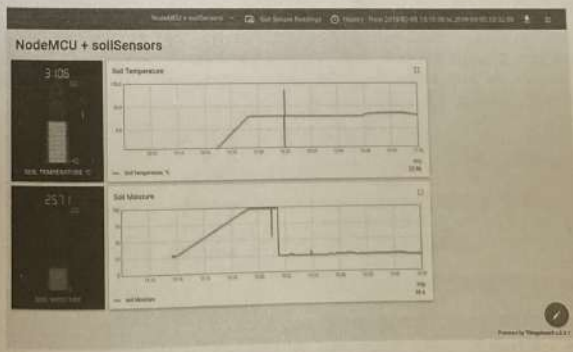
- ThingsBoard Demo Server
  - login: <your ThingsBoard login>
  - password: <your password>

Go to 'Devices' section and locate 'SoilTempAndMoisture,' open device details and switch to 'Latest telemetry' tab. If all is configured correctly, you should be able to see latest values of 'soil Temperature' and 'soil Moisture' in the table.



Note: The key values (soil Moisture and soil Temperature) shown in the above image are directly called from the Device attributes stored in 'Device' menu. These key values are defined in the payload of the MQTT message which is sent to the ThingsBoard instance by the NodeMCU.

Open 'Dashboards' section, then locate and open 'SOILTEMPANDMOISTURE' dashboard. As a result, you will see two digital gauges and two time series charts displaying soil temperature and soil moisture.



BOX 8.2: CALIBRATION OF DFROBOT SOIL MOISTURE SENSOR

Code Listing: Calibration of DFRobot Soil Moisture Sensor

(Source credit: DFRobot)

```
void setup() {
Serial.begin(9600); // open serial port, set the baud rate as 9600 bps
}
void loop() {
int val;
val = analogRead(0); //connect sensor to Analog 0
Serial.println(val); //print the value to serial port
delay(100);
}
```

Note: Run the above code first in Arduino IDE makes sure the soil moisture sensor is calibrated using the above code and also putting it in water and air and noting the readings are correct.

BOX 8.3: CONNECTING A SOIL MOISTURE AND SOIL TEMPERATURE SENSOR TO NODEMCU

Code Listing: Connecting a Soil Moisture and Soil Temperature Sensors to NODEMCU (Arduino Sketch)

```
#include "DHT.h"
#include <PubSubClient.h>
#include <ESP8266WiFi.h>
#include <OneWire.h>
#include <DallasTemperature.h>
// Data wire is plugged into pin 2 on the Arduino
// #define ONE_WIRE_BUS 2
// Data wire is plugged into pin 2..... on the Arduino
#define ONE_WIRE_BUS 2
OneWire oneWire(ONE_WIRE_BUS);

#define WIFI_AP "USERNAME"
#define WIFI_PASSWORD "PASSWORD"

#define TOKEN "9FcooE0NbneIeSZ68ai5"
// DHT
#define DHTPIN 2 //D4 for NodeMCU
#define DHTTYPE DHT11
DallasTemperature sensors(&oneWire);

char thingsboardServer[] = "demo.thingsboard.io";

WiFiClient wifiClient;
```

(Contd)

Box 8.3 *(Contd)*

```
// Initialize DHT sensor.
DHT dht(DHTPIN, DHTTYPE);

PubSubClient client(wifiClient);

int status = WL_IDLE_STATUS;
unsigned long lastSend;

void setup()
{
Serial.begin(115200);
sensors.begin();
delay(10);
InitWiFi();
client.setServer( thingsboardServer, 1883 );
lastSend = 0;
}

float SMpin = A0;
float Po;
double PoY;
float VMC;
float temp;

void loop()
{
if ( !client.connected() ) {
reconnect();
}

if ( millis() - lastSend > 1000 ) { // Update and send only after 1 seconds
getAndSendSoilTemperatureAndMoistureData();
lastSend = millis();
}

client.loop();
}

void getAndSendSoilTemperatureAndMoistureData()
{
Serial.println("Collecting temperature data.");

// Po = analogRead(SMpin);
// PoY = (1.39*0.001*Po*Po)-(1.413*Po)+471.23;
```

(Contd)

Box 8.3 *(Contd)*

```
// VMC = (1.17*0.0000001*PoY*PoY*PoY)-(3.95*0.0001*PoY*PoY)+(4.90*0.001*PoY)-1.9;
float moisture_percentage;
moisture_percentage = ( 100.00 - ( (analogRead(SMpin)/1023.00) * 100.00 ) );
float m = moisture_percentage;

// Read temperature as Celsius (the default)
sensors.requestTemperatures();
temp = sensors.getTempCByIndex(0);
float t = temp;

// Check if any reads failed and exit early (to try again).
if (isnan(m) || isnan(t)) {
Serial.println("Failed to read from Soil sensors!");
return;
}

Serial.print("Soil Moisture: ");
Serial.print(m);
Serial.print(" %\t");
Serial.print("Soil Temperature: ");
Serial.print(t);
Serial.print(" *C ");

String soilTemperature = String(t);
String soilMoisture = String(m);


// Just debug messages
Serial.print( "Sending temperature and humidity : [" );
Serial.print( soilTemperature ); Serial.print( "," );
Serial.print( soilMoisture );
Serial.print( "] -> " );

// Prepare a JSON payload string
String payload = "{";
payload += "\"soil Temperature\":"; payload += soilTemperature; payload += ",";
payload += "\"soil Moisture\":"; payload += soilMoisture;
payload += "}";

// Send payload
char attributes[100];
payload.toCharArray( attributes, 100 );
client.publish( "v1/devices/me/telemetry", attributes );
```

(Contd)

Box 8.3 *(Contd)*

```
Serial.println( attributes );

}

void InitWiFi()
{
Serial.println("Connecting to AP ...");
// attempt to connect to WiFi network

WiFi.begin(WIFI_AP, WIFI_PASSWORD);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
Serial.println("Connected to AP");
}


void reconnect() {
// Loop until we're reconnected
while (!client.connected()) {
status = WiFi.status();
if ( status != WL_CONNECTED) {
WiFi.begin(WIFI_AP, WIFI_PASSWORD);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
Serial.println("Connected to AP");
}
Serial.print("Connecting to ThingsBoard node ...");
// Attempt to connect (clientId, username, password)
if ( client.connect("ESP8266 Device", TOKEN, NULL) ) {
Serial.println( "[DONE]" );
} else {
Serial.print( "[FAILED] [ rc = " );
Serial.print( client.state() );
Serial.println( " : retrying in 5 seconds]" );
// Wait 5 seconds before retrying
delay( 5000 );

}
}
```

SUMMARY

This chapter began with defining an IoT platform and identifying the need for such a platform. The basic characteristics of the IoT platforms are described with examples. Further, various commercial and open source IoT platforms were described in terms of their unique offerings. Some guidelines were identified in terms of usability, domain application, scalability, and security for optimal selection of an IoT platform. To get a feel of an IoT platform, a hands-on tutorial is presented that describes end-to-end steps to make devices enabled via an IoT software platform.

REVIEW QUESTIONS

1. What is an IoT platform?
2. What is the need of an IoT platform?
3. Describe the core characteristics of an IoT platform?
4. Describe various commercial IoT platforms?
5. What is device provisioning and management?
6. Describe open source IoT platforms.
7. What are the important considerations while selecting an IoT platform?
8. What is NodeMCU?
9. Describe the steps involved in connecting ThingsBoard IoT platform with soil moisture sensor and visualize its data.

REFERENCES

1. [Arduino] https://www.arduino.cc/ [last accessed: 09/04/2020]
2. [AWS IoT platform] https://d1.awsstatic.com/IoT/ [last accessed: 09/04/2020]
3. [EVRYTHNG] https://evrythng.com/ [last accessed: 09/04/2020]
4. [Google IoT Cloud] https://cloud.google.com/solutions/iot/ [last accessed: 09/04/2020]
5. [IBM IOT platform] https://www.ibm.com/support/knowledgecenter/en/SSQP8H/iot/platform/iotplatform_overview.html [last accessed: 09/04/2020]
6. [Bosch IoT Suite] https://www.bosch-iot-suite.com/capabilities-bosch-iot-suite/ [last accessed: 09/04/2020]
7. https://developer.cisco.com/docs/kinetic/#!overview/overview [last accessed: 09/04/2020]
8. [OGC SensorThings] http://www.opengeospatial.org/standards/sensorthings [last accessed: 09/04/2020]
9. [ThingsBoard] https://thingsboard.io/ [last accessed: 09/04/2020]
10. [OGC SensorThings Sensing] http://docs.opengeospatial.org/is/15-078r6/15-078r6.html [last accessed: 09/04/2020]
11. [Thinger] https://thinger.io/ [last accessed: 09/04/2020]
12. [ThingSpeak] https://thingspeak.com/ [last accessed: 09/04/2020]
13. [SpringBoot] https://spring.io/projects/spring-boot [last accessed: 09/04/2020]
14. [SiteWhere] https://sitewhere.io [last accessed: 09/04/2020]
15. [NodeMCU] https://www.nodemcu.com/index_en.html [last accessed: 09/04/2020]
16. [Microsoft Azure IoT Suite] https://azure.microsoft.com/en-gb/blog/microsoft-azure-iot-suite-connecting-your-things-to-the-cloud [last accessed: 09/04/2020]
17. https://www.dfrobot.com/wiki/index.php/Capacitive_Soil_Moisture_Sensor_SKU:SEN0193 [last accessed: 09/04/2020]
18. https://cdn.sparkfun.com/datasheets/Sensors/Temp/DS18B20.pdf [last accessed: 09/04/2020]
19. https://thingsboard.io/docs/user-guide/ui/dashboards/#dashboard" [last accessed: 09/04/2020]

Prototyping IoT Applications

CHAPTER 9

"To take the jobs of tomorrow, students must become more than good test takers. They need to become makers who design, build, test and prototype."

—Charles Best

OBJECTIVES

- introduce the concepts of prototyping and its importance and benefits
- explain prototyping from an IoT perspective
- describe the concepts of Logical design
- understand prototyping with API
- explain embedded coding and memory management techniques

OUTCOMES

- ability to prototype an IoT product idea product idea
- use online API for prototyping
- develop a working physical prototype

REVISION

Internet of Things (IoT) open hardware for IoT boards, sensors, and actuators are covered in Chapters 5 and 6. IoT middleware technology and different IoT platforms are explored in Chapters 7 and 8. These chapters give the necessary foundational material to start designing and developing IoT prototype for a new product idea.

9.1 INTRODUCTION

A new product idea usually takes a preliminary shape in the form of a prototype. While building an Internet of Things (IoT) prototype the following points are important:

- What is unique and novel in this device that you have set out to develop?
- How is it different from similar existing devices? What are its major functions?
- How do you interact with it ?

This information helps in deciding different layers in the prototype device to be made, though the IoT device is complex. The full stack prototype of IoT is shown in Fig. 9.1.

Some examples for showing goals of different prototype devices are given as follows:

- Purpose of the physical device
- Functionality of the device
- Device communication interface (as a web application or a mobile app)

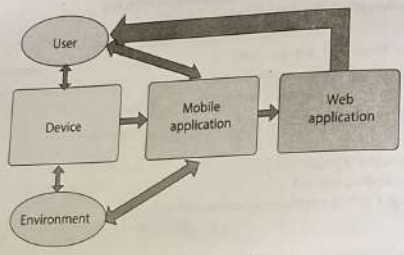


Fig. 9.1 Full Stack Prototype

Understanding and setting proper goals for the new product helps to decide the steps to be followed to implement the proper information flow of the chosen prototype.

9.2 PROTOTYPING AND ITS BENEFITS

Goal setting before prototype design is a very important step as the constraints for the prototyping process can be well understood. Various constraints such as size of the product, specialized/customized hardware, power need, cost, functionality, memory, and security (shown in Fig. 9.2)

Additionally, many other constraints, which are specific to the product to be prototyped, can be determined by asking some questions related to goals of design, need for field testing, extended scope of the product, funding and investors interest.

Understanding primary and secondary constraints for the prototype gives clarity for the design of the prototype.

9.2.1 Prototypes and IoT Product Ideas

A large number of possible applications can be developed using technologies of IoT, such as home automation and security, industrial asset tracking, chronic disease monitoring, and agricultural parameter monitoring. Some IoT products are complementary for the existing business products while others are new business opportunities.

Dimensions: Are the prototype dimensions flexible to support possible changes in the future driven by market needs and technology?
In-built Memory: Is the available device memory sufficient to support all the functions envisaged in the prototype?
Hardware Customization: Is there any requirement to add additional hardware to support specific functionality of the product?
Power Requirements: Is power consumption addressed in the prototype for the required time availability of the battery power?
Expenditure: How much budget is allowed on the prototype development?
Security: Testing the prototype may require certain security protocols to be followed, thus adding additional constraints.

Fig. 9.2 Common Constraints for Prototyping IoT Product

Step 1: Ideation Basically, every IoT product starts with an idea. It is suggested to continue with the components originally conceived for the prototype at the early stage instead of trying to optimize one particular component very early in the prototype development process. For example, to get better battery life, a lower powered processor is selected, but it may not support a particular sensor that you want to include in the prototype.

Step 2: Prototype At early stage, proof of concept creation becomes an expensive task. Rapid prototyping has become easier and cheaper now. It enables to develop an early product and verify its feasibility with minimal time and cost. Also, flexible enough to allow changes and adjustments as the development process evolves.

Step 3: Iterate In this step, various processes, vendors/ suppliers of components used in the product and other essential supporting services are finalised. However, certain components may change at a later point of time due to the evolving nature of technology, for example network connectivity, security, etc. Hence, those aspects have to borne in mind at this step.

Step 4: Deploy In this step, the conditions of the geographical area to be deployed are considered. For example, if the product is to be deployed for long durations and under harsh conditions, then durability of the device is important. Also, the battery life and connectivity issues have to be taken into account. Remote management and frequency of upgrades of the device needs due consideration. Any specific, socio-cultural aspects have also to be kept in mind.

Step 5: Scale Scaling IoT prototypes on a massive scale such as industrial, commercial, residential etc. requires considerations such as connectivity and interfacing with external systems, tweaking the functionality of the application for specific segments of the market, ensuring high performance, ability to capture feedback from the consumers, frequent upgradation of the product, etc. These require the selection of right hardware, software components early on keeping in mind the scalability of the product. Examples of scaling includes increasing the number of devices to cover more geographical area, gateways to support and manage the increased sensors, support for massive volumes of data storage, maintaining connectivity, version control, global vs local needs, etc. All these require several iterations in order to obtain optimal solutions.

9.2.2 Selection of Physical Devices

While building your prototype, keep in mind the key components listed further. You must take into consideration other objects that the prototype will be using such as for a specific task if the users will be wearing masks, gloves, eye protection, or have their hands engaged or gestures. The prototype to be tested should be subjected to the natural environment for which it is defined as close as possible, such as the targeted consumers, similar geographical area or location of its intended use, the entities with which it is going to interact either static or interactive, mode of interaction such as online, offline, real-time, etc.

9.2.3 Sketches and Diagrams

It is one of the oldest methods of prototyping. With very little efforts and without any artistic skills these sketches can be drawn. With such sketches, your ideas can be shared with others to take inputs from team mates.

A system, process, or the structure of your ideas can be illustrated by sketches. Sketches are useful to understand complex use cases, where many factors are interacting with each other. Behaviour maps, journey maps, system flow diagrams, and many other mapping methods are used for understanding complex situations.

9.2.4 Open Source Versus Closed Source Technologies

When deciding software hardware requirements, IoT product developers encounter the dilemma of selecting open versus closed technologies for the development as each choice will have some positive features and some drawbacks.

Open source software (OSS) uses freely available code for any kind of software (system or application). The hardware that supports such software is called open hardware. A product developer can customize the available open source code by modifying and improving to make it suitable for their product idea.

However, most open source software licenses require that such modified software is open to the public for download, modification, update, and improvement.

Closed source software (CSS) means the software developed is proprietary. It can be accessed by the original developers for any kind of changes that are to be done. Any changes to the software are prohibited.

Some basic aspects to be considered while choosing the technology:

Prototype cost and price Open source is often free of cost but it is challenging to integrate it in commercial products. Careful study of the open source licensing is required as different open sources licenses (e.g. GPL, LGPL, BSD, Apache, CC, etc.) provide varying levels of restrictions for use in commercial products. Usually, CSS is a paid software. Its cost depends on the complexity of the software. However, for the higher cost of the paid software, you get full support for the product and better readily available functionality. Even some free trials for limited time period are given by the companies for promoting their software-hardware technologies.

Continuous quality support Between open source and closed source software, CSS provide better support as they are charging you for that service also. The responses from customers are documented and analyzed for improving the services and the product. However for OSS, support options are limited to hired experts, forums, and useful articles.

Source code availability OSS code is available for any kind of change and customization, without any restrictions. CSS is restricted than OSS because the source code cannot be changed or altered.

Security In OSS, bugs are fixed quickly as the code can be modified or updated by anybody. Due to availability of the open source code to anyone, hackers can find the vulnerabilities for the attacks in such OSS. In CSS, the code is not openly available and hence, security threats are less. Good support teams are available for quickly fixing the problems and bugs that a customer faces.

Usability In OSS, user guides are written for developers rather than for novice users. Usability is a major benefit of CSS. It is well documented and also well written with detailed instructions.

OSS is more flexible while CSS provides ease of use and learning is usually more faster.

9.3 PHYSICAL DESIGN CONSIDERATIONS

Once the objectives and constraints to achieve those objectives in a product prototype are identified, then relevant functional modules can be designed for the same. This step provides a better clarity for selection of right tools and technologies. Following discussion is for choosing the right module for a specific prototype.

9.3.1 Different Modules for IoT Prototyping

At least some of the following elements are there in an IoT product. (refer to Fig. 9.3):

Processor modules For executing the device firmware

Sensory modules Sense the surrounding environment for information collection

Power module To supply the power

Input/output module These modules facilitate users to interact with the product physically by providing input or accessing output.

Communication module For device communication, to facilitate the device with the Internet

Action modules Modules to control surrounding objects using the device

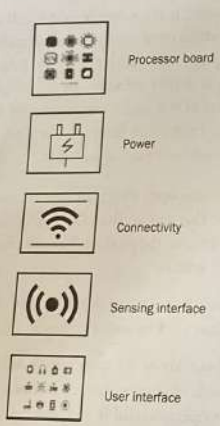


Fig. 9.3 Modules Required in an IoT Device

9.3.1.1 Selection of Embedded Platform: Microcontroller, System on Chips (SOC)

Processor modules: The processor board is the main part of a prototype of any IoT device. An IoT device processor is generally a low-power microcontroller board.

Main things to consider while selecting the processor:

- User-friendly development environment: It should be easy to use and support wide range of software
- Hardware capabilities of the selected processor: It should have suitable communication interfaces
- Memory: Memory size of the processor and its suitability for your design should also be considered

9.3.1.2 Common Open Source Microcontroller Boards for IoT Prototypes

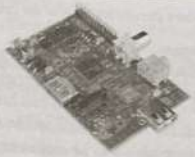
Various microcontroller boards are explained in detail in Chapter 6 (considering various parameters such as memory, requirement, size, weight, I/O analog and digital pins, input/output current, etc).

Popular open source microcontroller boards with their application area are listed in Box 9.1.

BOX 9.1: POPULAR OPEN SOURCE MICROCONTROLLER BOARDS

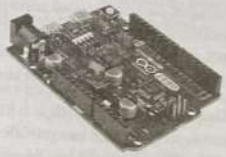
Raspberry Pi:

- Compatible with Android
- Used for smartphone/tablet applications
- Many models and varying in cost
- A Broadcom chip with 700MHz CPU
- Supports at least one HDMI port
- Low cost and wide variety of capabilities



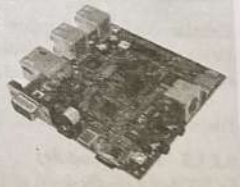
Arduino Board:

- Largest open source platform
- Easy to understand and affordable for small applications
- Microcontroller ATmega328 chip
- Operating voltage varies as per the models, Flash memory, SRAM, EEPROM
- Analog input pins, digital I/O pins
- Different Clock speed with different models



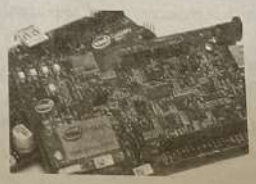
BeagleBoard:

- ARM Cortex-A8 processor, 512MB of DDR3 RAM, 2GB of on-board storage
- 3D graphics accelerator
- USB port and HDMI
- List of common applications of the board
  - Wi-Fi radio alarm clocks
  - Solar powered controllers
  - Retro gaming computers
  - Educational tool
  - Industrial robotics
  - Flying drones
  - 3D printers



Intel:

- Intel's Open Source Hardware namely Edison and Galileo
- Edison board is good for entry level inventors and for entrepreneurs going for mass production
- Galileo board is good for advanced user developing commercial products
- Running the code on an emulator



9.3.1.3 Selection of Sensors and Actuators

Sensor modules Choosing a sensor (for example, a temperature sensor) for an IoT application looks like a simple and straightforward decision. However, selecting the right sensor involves considering many factors into account such as availability, accuracy and precision, measurement range, power consumption,

cost, and supplier. Some examples of environmental sensors are accelerometer, temperature, humidity, camera, etc.

More details on sensors are covered in Chapter 5.

Action modules (actuators) There are several action modules (actuators) that allow the device to control the surroundings. Some examples are: servo motor controllers, DC motor controllers, sound amplifiers, and relays. These and many other actuators are covered in detail in Chapter 5.

9.3.1.4 Communication Modules

Information is to be communicated from the IoT device to the Internet. This is achieved by wireless or wired connectivity.

Commonly used wireless communication modules:

Wi-Fi Wherever coverage is available, internet connectivity is possible of a device. Wi-Fi configuration on an IoT device is somewhat complicated but once set up, it offers very high data rates.

Bluetooth LE (BLE) BLE is low power as compared to Wi-Fi. It allows direct communication with mobile devices. BLE can provide limited distance range of approximately 100 feet and limited bandwidth of less than 100kbps. However, the range issue can often be solved using BLE-Wi-Fi or BLE-Cellular wireless gateway hub.

Cellular Cellular connectivity allows a wider range of network connectivity than BLE or Wi-Fi. Hence, the IoT prototype can communicate with the Internet. However, for cellular connectivity data subscription to the network carrier is required. 4G, and now upcoming 5G are IoT focused.

Other communication modules that are generally in some specific applications are: USB (wired) Ethernet (wired), Zigbee, LoRa, Z-Wave, and Thread.

9.3.1.5 Power Modules

An IoT device usually works on low energy but still it needs to be powered. Many power options are available and selection of appropriate option for your prototype depends on the device characteristics such as battery operated or plugged in, life of the battery, and rechargeable or not.

Commonly used power modules for IoT devices are: USB powered, 9 V battery, AA/AAA batteries, rechargeable LiPo (lithium polymer) batteries, rechargeable/replaceable coin cells, and PoE (Power over Ethernet).

9.3.1.6 Input/Output Modules

Many kinds of Input/Output modules such as LEDs switches, touch- pads, LCD displays, etc.

9.3.2 Explore, Sketch, AND EXPERIMENT

The development of a prototype begins with exploratory analysis of the domain knowledge and then choosing the necessary set of tools and experimenting with them and thus give a basic shape to the idea. The next step is the selection of relevant prototyping tools that can support teams to iteratively work on the design in various stages such as preliminary testing of the idea, layout, device interaction, visualisation, etc. There is no single best tool that can be suggested because it all depends on what you need for a specific IoT application.

9.3.3 Introduction to Mechanical Design and Methodologies

This section presents an overview of different physical prototyping stages.

9.3.3.1 Rapid Prototyping Techniques

The manufacturing of the prototype requires certain experience to decide on the right option. This knowledge could be built over a period of time. Also, the cost of manufacturing and time required to do it could play a major role. Also, the specific properties of the material and number of pieces to be manufactured determines the selection of a particular method. It is suggested that for the initial prototype manufacturing less complicated methods may be used. Below are some approaches:

3D printing A reduced setup can be used for quicker outputs. If specific look and feel are required then it will be more costly and time consuming. For low volumes, it is significantly faster and cheaper than other techniques.

Computerized Numerical Control (CNC) machining Setup time plays a major role in deciding cost. Hence, it is good to make many units for return on investment.

CNC machining technology includes:

Turning The work piece is put into high-speed rotation and a moving cutting tool removes the material.

Milling The cutting tool itself is put into rotation to bring cutting edges to the work piece.

Drilling Holes are drilled by rotating a cutter on the work piece.

Casting Multiple models are created from a first master model that will be used as a reference.

9.3.3.2 Prototype Stage to Production Stage

9.3.3.2.1 Sourcing

Sourcing team checks the availability of required components to build the prototype and the available components are replaced for the specific components in a prototype.

9.3.3.2.2 NPR Release with Tooling and Sampling

After sourcing and quoting stage when the components are available the product goes through a stage named as Non Production Release (NPR). Continuously evolving quality procedure is documented for product specification and then certification by a third party audit. This is an iterative process till the right changes are made in the product.

9.3.3.3 Production Launch

In the product launch stage, various goals are achieved such as making the product available to customer as quickly as possible, improving the production quality by monitoring and obtaining enough sample quantity for the product to assess the quality of the product.

After completion of production phase the final product is inspected for quality control and issues are fixed and reported.

9.4 PROTOTYPING LOGICAL DESIGN

Some important points while developing IoT products and systems for different kinds of clients in consumer and industrial applications are listed as follows:

Unambiguous business strategy Performance/cost trade-off for the device to be developed and value addition by taking inputs from target customers for their expectations is important for business model clarity.

Optimising design Depends on the scale of data transfer as well as the cloud server or other system where the raw data of the sensor is processed. Further, the data capture and actual processing timestamps are to be captured to understand latency in the processing and consider it while improving the design accuracy.

Selecting application specific power model An IoT application determines the kind of battery to be used. Applications that track slowly changing features such as infrastructure, assets, bridges, tunnels etc. require the IoT device to have long battery duration generally few years. Other applications such as wearables, health monitoring devices, etc. require devices that monitor for shorter duration so the battery maybe used accordingly. Hence, the power model has large variations and needs to be carefully understood.

Device security aspects Protecting user's data for security reasons, for preserving privacy and for properly recording critical data (for example, medical condition of a serious patient) and fixing bugs in the device with patches, security practices are needed.

Adapting to network technology evolution It is important to be ready for the fast changing communication technology landscape. As the communication technology is evolving (2G, 3G, 4G, and now 5G), adapting new technology and balancing with existing technologies is needed for the long life of an individual device.

Support for multiple platforms Decision about device compatibility and connectivity to various platforms needs to be thoroughly assessed. For example, which OS (s) will be fully supported by the device software and version control mechanism has to be put in place.

Adopt well proven technologies Many existing large-scale IoT platforms (e.g. Microsoft, Apple, Google, Samsung, Amazon, etc.) are providing a bouquet of services. Hence, it is not necessary to build the infrastructure from scratch as reuse of well proven current technologies is always a good choice.

9.5 PROTOTYPING USING API

Prototyping is an important stage of the design process as it is useful for reviewing concepts and obtaining feedback in the early stages of a project. A prototype helps to identify the drawbacks of the developed product and the limitations can be resolved before mass production. Some of the interactive tools to develop such mockup of your prototype are available, for example InVision, Framer, Marvel,

Origami Studio, Proto.io, etc. InVision is a web based interactive prototyping tool. Framer is a screen design tool for macOS which supports animated and interactive prototypes with your customized code. Marvel is a web based and mobile based tool for simplifying prototyping process. It supports different image formats for uploading such as GIF, TIFF, PSD. Facebook has made available the tool named as Origami Studio for prototyping. Proto.io is also a web based interactive prototype designing tool.

9.5.1 Application Programming Interface (API)

Sometimes a software application needs to send data to or receive data from another software application. Integration between two separate applications built with separate development languages is required in some cases. An Application Programming Interface (API) is a set of routines for providing such an interface. Software components' communication with one another is specified with an API. The applications of APIs are in wide areas. Software applications need not operate in isolation when API is used and the application can use functionalities that already exist in other applications using API calls.

For example, functionalities of YouTube and Twitter can be embedded into other websites with the use of APIs. An API works as a broker that sends messages from one system to another.

9.5.2 API for Real-time IoT Applications

IoT devices can be brought up to speed and scale with real time in different ways. Sensors connected with other IoT objects can provide immediate information to multiple IoT devices on that sensor network. Thus, excellent real-time services can be provided by IoT applications. Real-time data needs to be pushed to multiple devices often. Talend is one such platform where you can stream connected devices with external data such as the weather. Many real-time services for data streaming in medical data on online payment transaction data are benefited with theft or corruption of data with real-time transfer. In addition, the data can be transmitted in real time to multiple devices.

Polling Polling is a mechanism to fetch fresh data. Instead of manually refreshing or reloading a webpage, the new data is refreshed by a web application from the server. It is a synchronous operation. Due to the frequent fetching of data packets, the CPU could be over utilised, leading to higher battery usage. On the other hand if fetching is less frequent, it could lead to higher latency and the application may exhibit lagging. Hence, proper optimisation of polling is recommended using a spooling method.

Push For pushing the data to a browser, when browser is not requesting the data, HTTP requests are used. Web sockets and Server Sent events (SSE) technologies enable a client to receive automatic updates from a server via HTTP.

9.5.3 Packages for IoT in Python

IoT has wide area of applications such as in wireless sensor networks, big data and machine learning, data analytics, and real-time analytics. For such diversified fields, a programming language is needed which spans these diverse fields and is lightweight and scalable. Over the last decade, Python programming language is a popular choice of developers. Some major advantages of Python over other languages are:

- It is a very simple language for programming and for deployment
- It is embeddable, portable, and scalable, irrespective of the operating system
- It provides a good support and libraries for the language

The popular Python packages are mraa (GPIO), opencv (signal, video and image processing) numpy (scientific computing) tensorflow (numerical calculation, machine learning, deep learning) mysqldb (relation database) sockets (networking), pandas (data science), pano-Mqtt (MQTT protocol), matplotlib (data visualisation) tkinter (GUI development).

9.6 EMBEDDED CODE WRITING

Survey of IoT developers shows four preferred languages are used for embedded code writing:

- Java
- C/C++
- Python
- JavaScript

These languages are also preferred for developing desktop applications, mobile apps, and for server applications. In IoT also these languages are used predominantly. IoT applications need some additional functionalities and support.

Three broad areas to be focused when designing IoT architecture for an application are sensors to collect environmental information, local gateway to organize data hubs and centralized server to gather, process and analyze data.

If the sensor has basic construct, it is probably using C so that direct connectivity to RAM can be accomplished. For the rest, developers generally choose the language that best suits them to build the application.

There are many other languages used by developers to build some innovative and interesting smart such as: B#, Go, PHP, parasail, etc.

More than one language may be necessary for providing the required functionality. These days in IoT, the languages listed above are used in a combination to make the things smarter.

9.6.1 Writing Efficient Code

Most IoT devices need careful embedded programming as the nodes need to be managed efficiently due to reasons such as exponential growth of the nodes as the IoT system scales up, need to ensure uptime of the sensors, smart power management of the nodes to ensure longer runtimes and managing other resource constraints of gateways, memory, etc.

9.6.1.1 Memory Management

Memory management in IoT has high requirement due to the limits in memory in the devices, gateways and other resources. Hence, embeddded software has to be written by considering memory handling techniques for reliability and long run time.

From an IoT perspective if the memory issues (arising due to unpredictable usage patterns of the device) are not properly taken care of, it could lead to errors such as fragmentation (due to issues in heap usage, allocation and freeing of memory) mainly due to limited onboard memory and heap failures due to the inability to supply allocated memory even if it is available because of memory leaks and fragmentation. Hence, careful consideration should be given during the testing phase to include as many usage patterns of the device as possible and test the memory aspects.

9.6.1.2 Debugging Syntax

After following the necessary steps of design, testing, iteration, and physical working prototype, you bring up a great, perfectly functioning device and you can move confidently toward your first production run. However, a production run is filled with many bugs and errors. With proper debugging, you can get through.

Different validation tests such as preproduction validation, market requirement validation, Design for Manufacturing (DFM) validation are needed for removing flaws.

9.7 REAL-WORLD PROTOTYPE EXAMPLE: SMART HOME APPLIANCES (LIGHT AND FAN)

The product idea of smart fan and smart tube is as below:
Smart light and fan can be turned on and off automatically depending on the motion of a person and room light (Fig. 9.4).

- The bidirectional visit count is evaluated for each time a person is entering and coming out of the room
- This helps in automatic turning off the light and fan when nobody is present in the room
- This helps in saving energy consumption instantly

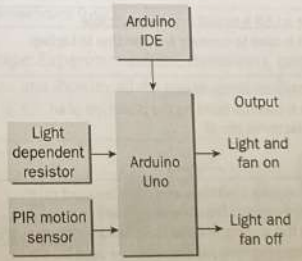


Fig. 9.4 Block Diagram of Automation of Smart Light and Smart Fan

9.7.1 Design Stage

Design stage is comprised of gaining domain knowledge, sketching the prototype design, and iterations of the previous two steps.

9.7.1.1 Functions of a Smart Light

- On/Off operation based on environment light and motion of the body
- If there is enough light of the environment and body motion is detected, then the light will remain OFF
- If there is darkness in the room and motion is detected, then the light will be switched ON automatically
- If there is no motion, then light will be OFF whether there is brightness or darkness in the room

Software and hardware requirements

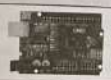
Software: Arduino IDE, Raspbian OS, Python IDE, Arduino Uno and other sensors and actuators, Raspberry Pi, computer system, Embedded C programming, Python programming

Hardware: Arduino Uno and other sensors and actuators, Raspberry Pi, computer system

9.7.1.2 List of Components, Model Name, and Specifications

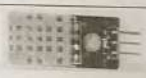
Table 9.1 represents various components and their specifications.

Table 9.1 Components and their Specifications

S. no.	Component name	Description, model name, and specifications	Component image
1.	Arduino Uno	It is an open source microcontroller board It is based on microchip ATmega 328P microcontroller	
2.	Connector cable	It is USB A to USB B connector cable It is used to connect Arduino Uno to Laptop	
3.	Breadboard	It is a solderless device It is used for building the prototype of an electronic circuit	
4.	Jumper wires	These are electronic wires that are used to interconnect the components within a circuit In this project, male to male, female to female, and female to male jumper wires are used	
5.	Resistors	It is a passive electronic component It controls the flow of current in the electronic circuit	
6.	LED	It is an electric light source It is a semiconductor light that emits light when the current flows through it	
7.	Relay module	It is an electrically operated switch It can control the flow of current flowing through the appliance	

(Contd)

Table 9.1 *(Contd)*

S. no.	Component name	Description, model name, and specifications	Component image
8.	Bulb	It is a night lamp and is 0.5 W Less maintenance Light weight and simple design	
9.	Fan	It is a 12 V DC fan Maximum air flow: 49.7 CFM Voltage: 12 V Rated speed: 4000–5000 rpm	
10.	LDR sensor module	LDR light sensor Working voltage: 3.3–5 V LM393 chip is used On board sensitivity adjustment	
11.	PIR motion sensor	Big Dome PIR Input voltage: 5 V DC Sensing angle: 110° Range: 3–5 m	
12.	IR module	IR sensor Input voltage: 5 V Two on-board LEDs	
13.	DHT11 temperature and humidity sensor module	DHT11 humidity + temp Input voltage: 3.3–5 V DC Output: digital Humidity: 20–90% RH Temperature: 0–50°C	

9.7.1.3 Development Stage: Experimentation, Iterations, and Testing

The circuit diagram for Arduino uno showing all the connections with sensors, actuators, and different modules is represented in Fig. 9.5.

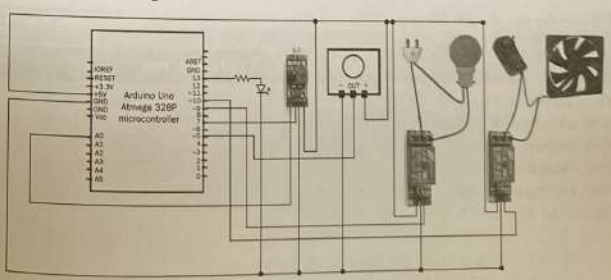


Fig. 9.5 Circuit Diagram for Arduino Uno Connectivity with Sensors, Actuators, and Different Modules

Connections for LDR sensor module

- Connect A0 pin of LDR module to the A0 pin of Arduino Uno
- Connect D0 pin of LDR module to the digital pin 7 of Arduino Uno
- Connect Vcc pin of LDR module to the 5 V pin of Arduino Uno
- Connect GND pin of LDR module to the GND pin of Arduino Uno

Connections for PIR sensor

- Connect Vcc pin of PIR sensor to the 5 V pin Arduino Uno
- Connect OUT pin of PIR sensor to the digital pin 5 of Arduino Uno
- Connect GND pin of PIR sensor to the GND pin of Arduino Uno

Connections for LED

- Connect positive end of LED to the resistor and another end of the resistor is connected to the digital pin 13 of Arduino Uno
- Connect negative end of LED to the GND pin of Arduino Uno

Connections for relay module 1 (Bulb)

- Connect IN pin of relay module 1 to the digital pin 9 of Arduino Uno
- Connect GND pin of relay module to the GND pin of Arduino Uno
- Connect Vcc pin of relay module to the 5 V pin of Arduino Uno
- Connect bulb and its supply pin to the NC and COM connection port of relay module

Connections for relay module 2 (Fan)

- Connect IN pin of relay module 2 to the digital pin 10 of Arduino Uno
- Connect GND pin of relay module to the GND pin of Arduino Uno
- Connect Vcc pin of relay module to the 5 V pin of Arduino Uno
- Connect fan and its supply pin to the NC and COM connection port of relay module

Box 9.1 shows the pseudocode for smart light and fan.

BOX 9.1: PSEUDOCODE: SMART LIGHT AND FAN

```
1. define ldrpin = 7
2. define pirpin = 5
3. define ledpin = 13
4. define relaypin1 = 9
5. define relaypin2 = 10
6. initialize the count = 0
7. setup( )
        i. set the pin mode of ldrpin as INPUT
       ii. set the pin mode of pirpin as INPUT
      iii. set the pin mode of ledpin as OUTPUT
```

(Contd)

Box 9.1 *(Contd)*

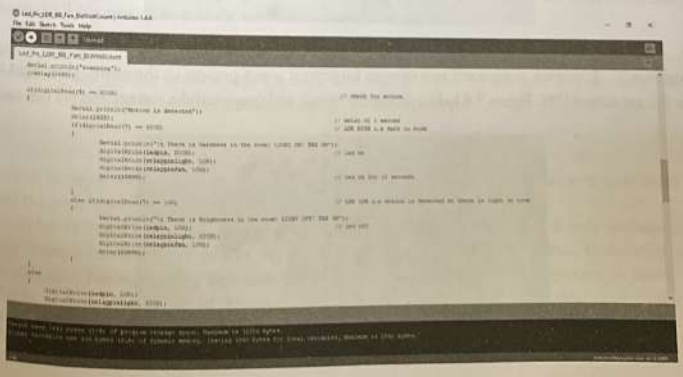
```
       iv. set the pin mode of relaypin1 as OUTPUT
        v. set the pin mode of relaypin2 as OUTPUT
8. loop()
   1. if pirpin is HIGH
       if ldrpin is HIGH
        1. set ledpin HIGH // darkness in the room
        2. set relaypin1 LOW // Bulb ON
        3. set relaypin2 LOW // Fan ON
        else
        1. set ledpin LOW // brightness in the room
        2. set relaypin1 HIGH // Bulb OFF
        3. set relaypin2 LOW // Fan ON
   2. else
        set ledpin LOW
        set relaypin1 HIGH // Bulb OFF
        set relaypin2 HIGH // Fan OFF
```

Steps for loading the code and working of smart light

- Compile the sketch code in the Arduino IDE
- Download software from the website https://www.arduino.cc/en/Main/Software
- Upload sketch from the Arduino to the Arduino Uno microcontroller
- **Case 1:** If brightness is present in the room and motion is detected
- **Case 2:** If brightness is not present in the room and motion is detected
- **Case 3:** If brightness is not present in the room and motion is not detected

9.7.1.4 Embedded Programming using Arduino IDE

Sketch snapshot of the code is shown in Fig. 9.6.



Output sketch

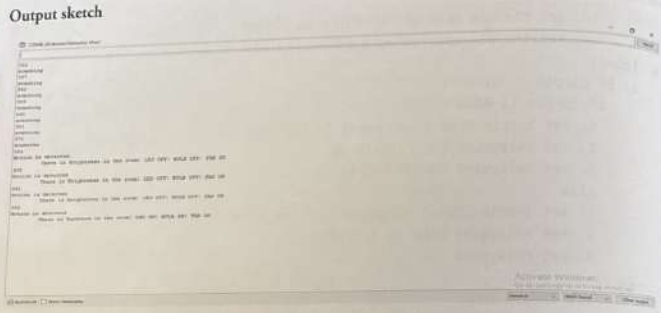


Fig. 9.6 Snapshot of the Code

9.7.2 Test Cases

Test case 1 If motion is detected and brightness is present in the room, then light remains OFF and the fan is turned ON. Figure 9.7 below shows the circuit and output of the connection for test case 1.

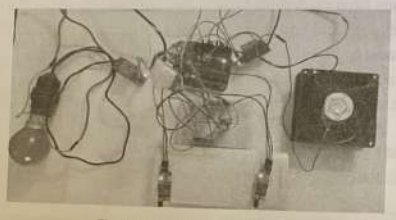


Fig. 9.7 Circuit and Output

Test case 2 If motion is detected and enough brightness is not present in the room, then light and the fan are turned ON. Figure 9.8 below shows the circuit and output of the connection for test case 2.

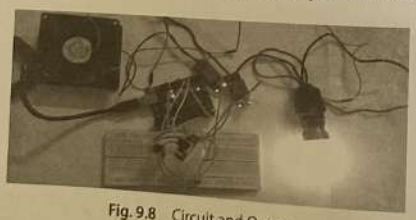


Fig. 9.8 Circuit and Output

Test case 3 If motion is not detected, whether there is brightness or darkness, light and fan are turned OFF. Figure 9.9 below shows the circuit and output of the connection for test case 3.

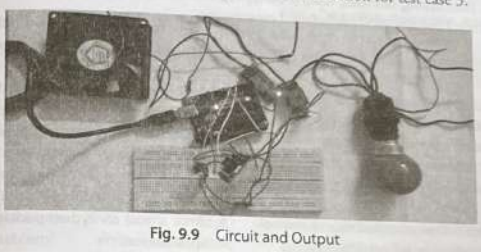


Fig. 9.9 Circuit and Output

THOUGHT EXERCISES

Write prototyping step for a smart house appliance for water sprinkling system for the garden with idea statement, design stage (knowledge, sketch, and experiment stage) with testing and with iterations in design.

9.8 BEST PRACTICES

IoT product should be built around four key elements that is people, objects, location and interactions.

As range of choices is wide, with the right prototype you can take a big step towards ultimately realizing your idea in the form of a user-friendly design.

Some recommendations to help create an excellent prototype are:

Fix right set of goals Based on a clear problem statement, fix a set of unambiguous goals. Ensure proper testing of the goals so that they fully satisfy the stated functionality of the prototype.

Select the right prototyping approach While creating a prototype, consider the stage of the design process, available resources, and the time limit to complete the task.

Explore and select right set of prototyping tools Do an exploratory analysis of the existing tools, by installing and getting familiar with the GUI and tools provided by software. It makes it lot easier to use this knowledge later when actually working on the prototype.

SUMMARY

Selection of right tools, platforms, and hardware makes the prototyping process smooth and fast with less number of iterations. It is possible to build and iterate network-connected IoT devices very quickly with electronic hardware using high-level software packages and libraries and good debugging tools. Many commercial web services can provide storage, communication, and computation for web-based IoT applications.

IoT applications often work with heterogeneous devices, services and protocols. Considering such high heterogeneity it is necessary to realise the importance of design space and prototyping using various tools.

KEY TERMS

IoT prototyping, prototyping modules, rapid prototyping, embedded programming, memory management, representational state transfer (REST), APIs for IoT prototyping

REVIEW EXERCISES

1. Explain what are the benefits of prototyping and its importance.
2. Explain prototype framework with its feasibility considering different parameters such as cost, time, and availability based on geographical area.
3. Compare open source vs closed source technology usage in context to IoT application areas.
4. What are the selection parameters for choosing embedded platforms for IoT device?
5. List popular open source embedded platforms and compare their advantages and disadvantages.
6. Explain mechanical design methodologies while prototyping the physical device.
7. Write a note on use of IDE while writing the embedded code.
8. What are the popular programming languages for IoT applications?
9. Write a note on Python packages useful for IoT applications?
10. Explain the importance and benefits of API?
11. Write a note on real-time API.
12. Explain how to write efficient code with memory management in IoT scenario of constrained memory.

REFERENCES

1. Massimo, B., O'Reilly. 2008. Getting Started with Arduino. ISBN: 978-0-596-15551-3
2. Pollie B. et al. Al.2012. Telematic Dinner Party: Designing for Togetherness through Play and Performance. In Proceedings of Designing Interactive Systems. DIS 2012.
3. Casaleggio A. 2011. The Evolution of Internet of Things. Online at: http://www.casaleggio.it/pubblicazioni/Focus_internet_of_things_v1.81%20-%20eng.pdf [last accessed 30/6/2019]
4. John, G. 2009. The Embedded Internet: Methodology and Findings. IDC.
5. Steve, H., et al. 2013. NET Gadgeteer: Experiences with a new platform for K-12 computer science education. Proceedings of the 44th SIGCSE Technical Symposium on Computer Science Education.
6. Tomas, L., Damith C., Ranasinghe, Mark H., Duncan M., Adding sense to the Internet of Things, An architecture framework for Smart Object systems.
7. Hideaki, K., Saul, G. 1999. Mediating awareness and communication through digital but physical surrogates. In CHI '99 extended abstracts. pp. 11–12.
8. Leonard Richardson and Sam Ruby, RESTful Web Services, O'Reilly, 2007.
9. Constantine A. Valhouli, "The Internet of things: Networked objects and smart devices," The Hammersmith Group Research Report, February 2010.
10. Villar, N., Scott, J., Hodges, S. 2011. Prototyping with Microsoft .NET Gadgeteer. In Proceedings of the fifth International Conference on Tangible, Embedded and Embodied Interaction, TEI.
11. OpenIoT Summit, 3/6/2016 Embedded Programming for IoT. Online at: https://events.static.linuxfound.org/sites/events/files/slides/Embedded%20Programming%20for%20IoT.pdf [last accessed: 30/6/2019]
12. Jakob, N. 2003. Paper Prototyping: Getting User Data Before You Code. Online at: https://www.nngroup.com/articles/paper-prototyping/ [last accessed: 30/6/2019]
13. Whitney, Q., Kevin, B. 2010.Storytelling for User Experience: Crafting Stories for Better Design. Online at: http://rosenfeldmedia.com/books/storytelling-for-user-experience/ [last accessed: 30/6/2019]
14. Tim, B. 2009. Change by Design: How Design Thinking Transforms Organizations and Inspires Innovation.
15. IDEO, Determine What to Prototype. 2016. Online at: http://www.designkit.org/methods/34memory issues in some detail: https://01.org/blogs/2016/heap-allocation-iot [last accessed: 30/6/2019]
16. Rise of the Embedded Internet. Intel embedded processors white paper. 2009. Online at: http://download.intel.com/ embedded/15billion/applications/pdf/322202.pdf. [last accessed: 30/6/2019]
17. Casaleggio, A. 2011.The Evolution of Internet of Things. Online at: http://www.casaleggio.it/pubblicazioni/Focus_internet_of_things_v1.81%20-%20eng.pdf. [last accessed: 30/6/2019]
18. Valhouli, C. A. 2010. The Internet of Things: Networked Objects and Smart Devices. The Hammersmith Group Research Report. Online at: http://thehammersmithgroup.com/ images/reports/networked_objects.pdf [last accessed: 30/6/2019]
19. Richardson, L., Ruby, S. 2007. RESTful Web Services, O'Reilly.
20. Kuzuoka, H., Greenberg, S. 1999. Mediating Awareness and Communication through Digital but Physical Surrogates. Proc. Conf. Human Factors in Computing Systems (CHI 99). ACM. pp. 11–12.

21. P. Barden et al., "Telematic Dinner Party: Designing for Togetherness through Play and Performance," Proc. Designing Interactive Systems Conf. (DIS 12), ACM, 2012, pp. 38–47.
22. S. Hodges et al., ".NET Gadgeteer: Experiences with a New Platform for K-12 Computer Science Education," Proc. 44th SIGCSE Tech. Symp. Computer Science Education, ACM, 2013 (to appear).
23. G. Korteum et al., "Educating the Internet-of-Things Generation," Computer, Feb. 2013, pp. 53–61.
24. Jamal Hadi Salim; Robert Olsson; Alexey Kuznetsov (2001-11-10). Beyond softnet (PDF). 5th Annual Linux Showcase & Conference (ALS '01). pp. 165–172. Retrieved 2011-03-06. The classical NAPI paper.
25. http://arduino.cc/[last accessed 30/6/2019]
26. http://shieldlist.org/ [last accessed 30/6/2019]
27. http://mbed.org/[last accessed 30/6/2019]
28. http://nodejs.org/ [last accessed 30/6/2019]
29. http://www.raspberrypi.org/[last accessed 30/6/2019]
30. http://beagleboard.org/bone [last accessed: 30/6/2019]
31. http://sense.open.ac.uk/[last accessed: 30/6/2019]
32. http://www.iobridge.com/[last accessed: 30/6/2019]
33. http://electricimp.com/[last accessed: 30/6/2019]
34. https://iot.intersog.com/ [last accessed: 30/6/2019]
35. https://www.element14.com/[last accessed: 30/6/2019]
36. https://breadware.com/[last accessed: 30/6/2019]

FURTHER READING

1. Picking The Best Prototyping Software For Your Project: online at: https://www.smashingmagazine.com/2016/06/picking-the-best-prototyping-software-for-your-project/
2. 12 Factors In Selecting A Mobile Prototyping Tool: online at: https://www.smashingmagazine.com/2016/04/factors -selecting -mobile -prototyping -tool/
3. The Skeptic's Guide To Low-Fidelity Prototyping: online at: https://www.smashingmagazine.com/2014/10/the -skeptics -guide -to -low- fidelity-prototyping/
4. Lego Serious Play: online at: http://www.lego.com/en-us/seriousplay
5. d.school, Wizard of Oz Prototyping: online at: http://futureofstuffchallenge.org/download/prototype/bootleg-wizardofoz.pdf

Big IoT Data Science

CHAPTER 10

"You have got to think about the big things while you're doing the small things, so that all the small things go in the right direction."

– Alvin Toffler

OBJECTIVES

- define and expand on the concept of data science
- explore the applicability of AI, machine learning, deep learning for Internet of Things (IoT)
- introduce big data with reference to IoT
- explain the concepts of data swamp/lake
- discuss various IoT specific requirements for analytics
- explain IoT analytics in terms of real-time and offline analytics
- discuss various methods for IoT data analytics
- introduce commercial and open source analytics platforms that are relevant to IoT

OUTCOMES

- gain insight into the unique data science aspects of IoT
- relate big data concepts with IoT
- understand various methods used to perform IoT analytics
- get an overview of IoT data analytics platforms and their applicability in various application domains

REVISION

Chapters 1 and 2 introduced the fundamental concepts of Internet of Things (IoT) and highlighted the huge data generation potential of IoT from multiple devices. Chapters 7 and 8 described middleware and software platforms that enable to quickly get an IoT system up and running and also emphasized IoT analytics as an important component of the middleware. The concept of real-time streaming analytics and also cloud-based offline analytics were also introduced. Building on these concepts, this chapter focuses on analytics in IoT, by first introducing data science and its components. The discussion will focus on IoT specific needs for analytics and unique challenges that it poses for performing real-time streaming analytics. Further, currently popular analytics platforms are explained and how they are actually applicable on power-constrained devices. Hands on exercises for performing IoT analytics are also given.

10.1 FOUNDATIONS AND PRINCIPLES OF BIG DATA SCIENCE

The heterogenous and huge volume of data (having varying characteristics) coming from IoT devices forms the fundamental reason for studying and exploring the area of data science. Data science is the study of data in a scientific manner, which comprises integrating several disciplines (statistics, computer science, linguistics, econometrics, sociology, etc.). This section focuses on the background of data science and its relevance to IoT.

10.1.1 Introduction

Data science as defined by Jeffrey Stanton 'Data Science refers to an emerging area of work concerned with the collection, preparation, analysis, visualization, management and preservation of large collections of information.'

The word 'science' is the key term in data science, and 'data' is like a raw material that is used to perform scientific analysis focused on extraction of knowledge, which is general enough to be applicable on new data.

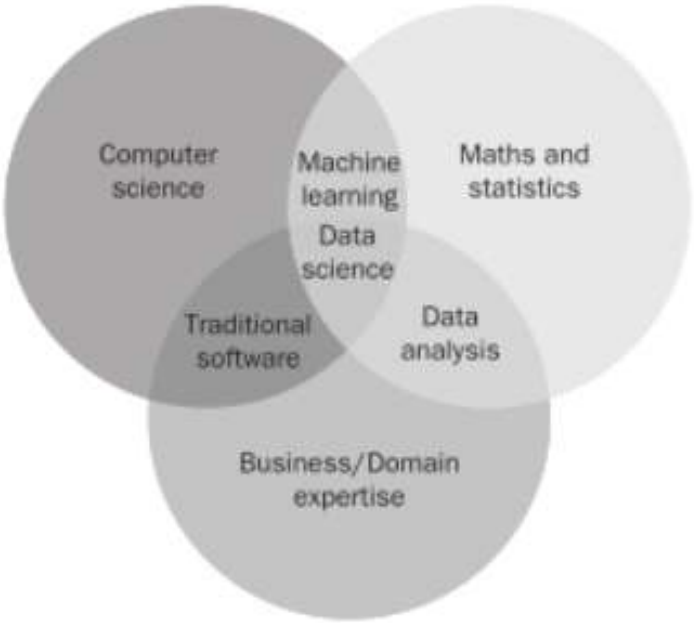


Fig. 10.1 Interdisciplinary Nature of Data Science

As shown in Fig. 10.1, data science includes techniques derived from several scientific disciplines such as mathematics, statistics, machine learning, artificial intelligence, computer science, databases, and optimization and combined with the understanding of the domain in which it is applied. Hence, any problem-solving exercise in data science requires:

- Data scientists, who understand the science of data and know how to formulate the problem that is domain specific; whose main goal is to extract hidden insights from data.
- Domain experts, who can explain the nature of the problem to the data scientists, and also interpret the validity of the results in the context of the domain.

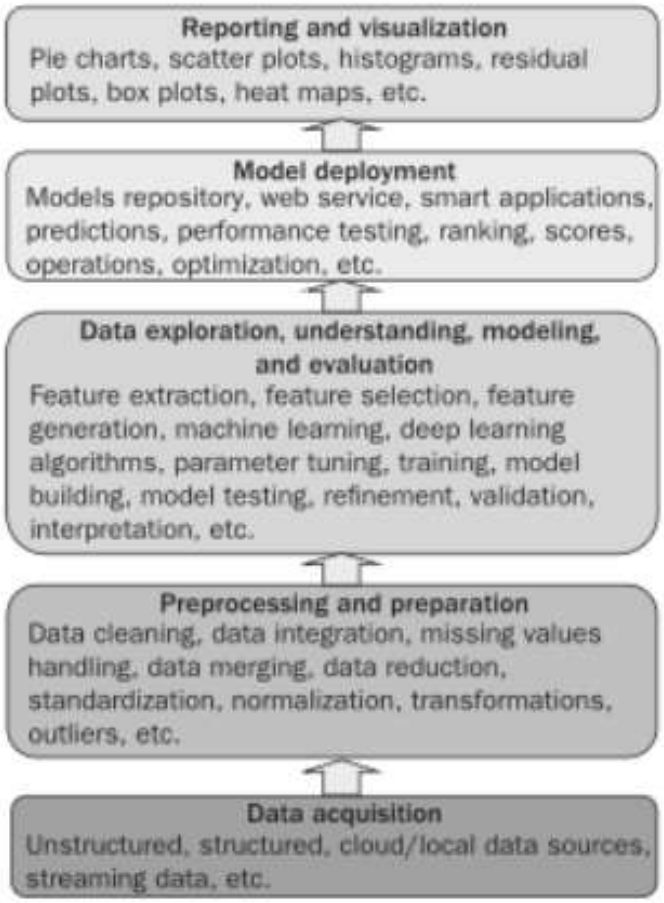


Fig. 10.2 Data Science Process

10.1.1.1 Data Science Process

The data science process involves several steps including (see Fig. 10.2):

Data acquisition The data can be obtained from a variety of IoT devices (such as sensors, actuators, etc.) and ancillary sources (other supporting data such as device health, etc.) either in online mode or data that is acquired from these devices and stored, secured, and managed in local (on premises) or cloud repository or both. The data generated by the IoT devices is highly heterogeneous in nature and can be in several forms such as:

Unstructured This form of data does not fit into a row/column format, which is the standard way of representation in a relational database. There is no predefined data model, schema associated with the data. It can be text, images, videos, discrete sensor readings, time-series data, etc. Due to differing requirements for storage of each of these types of data, data storage is a challenge. Unstructured databases such as NoSQL are usually preferred for this type of data. However, the type of storage is usually determined by the kind of end analytics that will carried out on that data.

Structured This form of data has a predefined record length and associated data model. Rarely, IoT data is directly available in structured form. It goes through a number of preprocessing steps to make it ready for further downstream analytics.

Further, the data can be classified based on its state such as:

Static data This type of data from IoT devices is usually high in volume and communicated using communication protocols such as MQTT or CoAP, and then ingested by IoT services for further processing and storage. Various IoT platforms (as discussed in Chapters 8, 9, and 11) have their own way of storing and managing the data.

Streaming data This type of data is usually captured at the gateways and in some cases on the device (having sufficient memory) to perform edge analytics. Since, this data is in motion, the analytics are tuned to work on a subset (window) of data at a time. This real-time processing is useful to send warnings, alerts, etc. and also trigger other IoT devices. Section 10.4.1 discusses analytics that can be performed on streaming data. In addition, Chapter 12 provides a more comprehensive material on edge analytics.

Data understanding, preprocessing, and preparation This process involves various operations such as:

Importing data Various ways to import the data include reading from tables, excel sheets, clipboard, comma separated values, fixed width formatted data, etc. Listing 10.1 shows an example of reading a CSV file in Python.

LISTING 10.1: IMPORTING DATA

```
# Import the 'pandas' library as 'pd'
import pandas as pd
# Load in the data with 'read_csv()'
PData = pd.read_csv("/Users/user1/datasets/pdata/pollutiondata.csv",
   header=0)
# set the width of the display in the output
pd.set_option('display.width', 300)
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)
#print the first 10 rows of the data
print(PData.head(10))
```

OUTPUT

	ozone	particullate_matter	carbon_monoxide	sulfure_dioxide	nitrogen_dioxide	longitude	latitude	timestamp
0	44	69	57	67	83	10.189355	56.182102	8/1/14 0:05
1	41	66	54	67	78	10.189355	56.182102	8/1/14 0:10
2	46	62	59	68	80	10.189355	56.182102	8/1/14 0:15
3	44	58	58	64	80	10.189355	56.182102	8/1/14 0:20
4	48	55	63	59	82	10.189355	56.182102	8/1/14 0:25
5	47	60	60	62	80	10.189355	56.182102	8/1/14 0:30
6	48	55	55	61	82	10.189355	56.182102	8/1/14 0:35
7	52	52	54	65	86	10.189355	56.182102	8/1/14 0:40
8	56	54	50	67	90	10.189355	56.182102	8/1/14 0:45
9	58	53	54	70	87	10.189355	56.182102	8/1/14 0:50

(Contd)

Listing 10.1 *(Contd)*

Exercise: Practise importing data from other formats. Check Pandas library (https://pandas.pydata.org/) for corresponding functions.

Data cleaning This is a major step in the preprocessing operations. The data cleaning process helps to make the data in a form that is usable for further processing. It involves rectifying improper data having missing values, formatting issues, malformed records, outliers, etc. Listing 10.2 shows ways to perform data cleaning using various operations. The missing values can be filled using data imputation techniques as shown in Listing 10.3.

LISTING 10.2: MISSING VALUES IDENTIFICATION AND FILLING

Missing values

```
# Import the `pandas` library as `pd`
import pandas as pd
# Load in the data with `read_csv()`
PData = pd.read_csv("/Users/user1/datasets/pdata/pollutiondata_missing.csv",
   header=0)
# set the width of the display in the output
pd.set_option('display.width', 300)
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)
#print the first 10 rows of the data
print(PData.head(10))
```

OUTPUT

	ozone	particullate_matter	carbon_monoxide	sulfure_dioxide	nitrogen_dioxide	longitude	latitude	timestamp
0	44.0	69.0	57.0	67.0	83.0	10.189355	56.182102	8/1/14 0:05
1	41.0	66.0	54.0	67.0	78.0	10.189355	56.182102	8/1/14 0:10
2	46.0	62.0	59.0	68.0	80.0	10.189355	56.182102	8/1/14 0:15
3	44.0	58.0	NaN	64.0	80.0	10.189355	56.182102	8/1/14 0:20
4	NaN	55.0	63.0	NaN	82.0	10.189355	56.182102	8/1/14 0:25
5	47.0	60.0	60.0	62.0	NaN	10.189355	56.182102	8/1/14 0:30
6	48.0	NaN	55.0	61.0	82.0	10.189355	56.182102	8/1/14 0:35
7	52.0	52.0	NaN	65.0	86.0	10.189355	56.182102	8/1/14 0:40
8	56.0	54.0	50.0	67.0	90.0	10.189355	56.182102	8/1/14 0:45
9	56.0	53.0	54.0	70.0	87.0	10.189355	56.182102	8/1/14 0:50

Note: Notice the NaN values in the output. Some data is missing.

LISTING 10.3: DATA IMPUTATION APPROACH TO FILL MISSING DATA

```
# Imputation is the process of replacing missing data with substituted values. You can either fill #in
the mean, the mode or the median or choose an interpolation technique using the #interpolate()
function to perform interpolation which gives an approximate value based on #neighborhood values.
```

(Contd)

Listing 10.3 (*Contd*)

```
# Import the `pandas` library as `pd`
import pandas as pd
import numpy as np
# Load in the data with `read_csv()`
PData = pd.read_csv("/Users/user1/datasets/pdata/pollutiondata_missing.csv",
   header=0)
# set the width of the display in the output
pd.set_option('display.width', 300)
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)
#print the first 10 rows of the data
print(PData.head(10))
# Identify missing values
s=PData.isnull()
# Prints True or False depending on whether there is data value present or not
print(s)
# calculate Mean
mean_ozone=np.mean(PData.ozone)
# Replace missing values with the mean
filled_data = PData.ozone.fillna(mean_ozone)
print(filled_data)
```

OUTPUT

	ozone	particullate_matter	carbon_monoxide	sulfure_dioxide	nitrogen_dioxide	longitude	latitude	timestamp
0	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False
3	False	False	True	False	False	False	False	False
4	True	False	False	True	False	False	False	False
5	False	False	False	False	True	False	False	False
6	False	True	False	False	False	False	False	False
7	False	False	True	False	False	False	False	False
8	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	False
10	False	False	False	False	False	False	False	False

```
0     44.000000
1     41.000000
2     46.000000
3     44.000000
4     51.229167
5     47.000000
6     48.000000
7     52.000000
8     56.000000
9     56.000000
10    57.000000
```

(*Contd*)

Listing 10.3 (*Contd*)

Note: Notice that the value in the fourth row (see output of listing 10.2), which was originally 'NaN', is replaced with the values 51.229.

Exercise: Modify the above code so that the missing values in all the columns are replaced at once.

Data merging This involves combining datasets from different sources to create a unified dataset which is further used for processing. This process is useful for data integration purposes when there is heterogeneity in the datasets in terms of data models, formats, etc. The merging technique is demonstrated in Listing 10.4 where data from two sensors is merged based on the timestamp of their acquisition.

LISTING 10.4: DATA MERGING TECHNIQUE USING PANDAS IN PYTHON

Data merging

```
# Import the `pandas` library as `pd`
import pandas as pd
import numpy as np
# Load in the data with `read_csv()`
ozonedata = pd.read_csv("/Users/user1/datasets/pdata/ozone.csv",
   header=0)
patriculatedata = pd.read_csv("/Users/user1/datasets/pdata/particulate_matter.csv",
   header=0)
# set the width of the display in the output
pd.set_option('display.width', 300)
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)
#print the first 10 rows of the data
print(ozonedata.head(10))
print(patriculatedata.head(10))
mergeddata=pd.merge(ozonedata,patriculatedata)
print(mergeddata)
```

OUTPUT

```
     timestamp  ozone
0  8/1/14 0:05     44
1  8/1/14 0:10     41
2  8/1/14 0:15     46
3  8/1/14 0:20     44
4  8/1/14 0:25     48
5  8/1/14 0:30     47
6  8/1/14 0:35     48
7  8/1/14 0:40     52
8  8/1/14 0:45     56
9  8/1/14 0:50     56
```

(*Contd*)

Listing 10.4 (*Contd*)

```
     timestamp  particullate_matter
0  8/1/14 0:05                   69
1  8/1/14 0:10                   66
2  8/1/14 0:15                   62
3  8/1/14 0:20                   58
4  8/1/14 0:25                   55
5  8/1/14 0:30                   60
6  8/1/14 0:35                   55
7  8/1/14 0:40                   52
8  8/1/14 0:45                   54
9  8/1/14 0:50                   53
```

Merged data:

```
     timestamp  ozone  particullate_matter
0  8/1/14 0:05     44                   69
1  8/1/14 0:10     41                   66
2  8/1/14 0:15     46                   62
3  8/1/14 0:20     44                   58
4  8/1/14 0:25     48                   55
5  8/1/14 0:30     47                   60
6  8/1/14 0:35     48                   55
7  8/1/14 0:40     52                   52
8  8/1/14 0:45     56                   54
9  8/1/14 0:50     56                   53
```

Data standardization Data standardization is required because the variables are measured at different scales, for example, in Listing 10.5, the data is measured by different sensors (light, temperature, humidity, etc.) at different scales; hence, it is possible that the analysis can be skewed. Hence, there is a need to standardize these sensor observations. It is also a general requirement for many machine learning algorithms. The common way to do standardization is to bring the data into a normal distribution, that is, Gaussian with zero mean and unit variance.

LISTING 10.5: DATA STANDARDIZATION BY A NORMAL DISTRIBUTION

```
from sklearn import preprocessing
import numpy as np
import pandas as pd
# set the width of the display in the output
pd.set_option('display.width', 300)
```

(*Contd*)

Listing 10.5 (*Contd*)

```
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)

# Read the CSV training and testing data using pandas
dataset=pd.read_csv("/Users/user1/datasets/Occupancy_data/datatraining.txt")

print(dataset.head(10))

# convert to numpy arrays and select the first 10 rows of data
Xtrain=np.array(dataset.head(10))

X_train=Xtrain[:,1:6]

print(X_train)

X_scaled = preprocessing.scale(X_train)
print(X_scaled)

xscaledmean=X_scaled.mean(axis=0)
print(xscaledmean)

# standard deviation tells you how much the individual numbers tend to differ from the mean

xscaledstd=X_scaled.std(axis=0)
print(xscaledstd)
```

OUTPUT

Snippet of raw data:

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
1	2015-02-04 17:51:00	23.180	27.2720	426	721.250000	0.004793	1
2	2015-02-04 17:51:59	23.150	27.2675	429.5	714.000000	0.004783	1
3	2015-02-04 17:53:00	23.150	27.2450	426	713.500000	0.004779	1
4	2015-02-04 17:54:00	23.150	27.2000	426	708.250000	0.004772	1
5	2015-02-04 17:55:00	23.100	27.2000	426	704.500000	0.004757	1
6	2015-02-04 17:55:59	23.100	27.2000	419	701.000000	0.004757	1
7	2015-02-04 17:57:00	23.100	27.2000	419	701.666667	0.004757	1
8	2015-02-04 17:57:59	23.100	27.2000	419	699.000000	0.004757	1
9	2015-02-04 17:58:59	23.100	27.2000	419	689.333333	0.004757	1
10	2015-02-04 18:00:00	23.075	27.1750	419	688.000000	0.004745	1

Scaled:

```
[[ 1.8519298   1.78624382  0.79240582  1.7058527   1.88909147]
 [ 0.91818368  1.64283441  1.67285672  0.98681595  1.22642022]
 [ 0.91818368  0.92578739  0.79240582  0.93722721  0.95034791]
 [ 0.91818368 -0.50830667  0.79240582  0.41654543  0.3982138 ]
 [-0.63805985 -0.50830667  0.79240582  0.04462987 -0.60933159]
 [-0.63805985 -0.50830667 -0.968496   -0.30249132 -0.60933159]
 [-0.63805985 -0.50830667 -0.968496   -0.236373   -0.60933159]
 [-0.63805985 -0.50830667 -0.968496   -0.50084629 -0.60933159]
 [-0.63805985 -0.50830667 -0.968496   -1.45956195 -0.60933159]
 [-1.41618161 -1.30502559 -0.968496   -1.5917986  -1.41741545]]
```

(*Contd*)

Listing 10.5 (*Contd*)

```
Zero mean:
[ 1.10134124e-14 -4.44089210e-17 -5.79536419e-15 4.48530102e-15
 -6.06181771e-15]

Unit variance:
[1. 1. 1. 1. 1.]
```

Data scaling It is the process of scaling the features (e.g., temperature, humidity, light, etc.) between a predefined maximum and minimum. In Listing 10.6, the data is scaled between 0 and 1, which is a very common way of scaling.

An alternative standardization is scaling features to lie between a given minimum and maximum value, often between 0 and 1, so that the maximum absolute value of each feature is scaled to unit size.

LISTING 10.6: SCALING DATA IN A GIVEN RANGE OF VALUES

```
from sklearn import preprocessing
import numpy as np
import pandas as pd
# set the width of the display in the output
pd.set_option('display.width', 300)
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)

# Read the CSV training and testing data using pandas
dataset=pd.read_csv("/Users/datasets/Occupancy/Occupancy_data/datatraining.txt")

print(dataset.head(10))
# convert to numpy arrays and select the top 10 rows
Xtrain=np.array(dataset.head(10))

X_train=Xtrain[:,1:6]

print(X_train)
min_max_scaling = preprocessing.MinMaxScaler()
X_train_minmax = min_max_scaling.fit_transform(X_train)

print(X_train_minmax )
```

(*Contd*)

Listing 10.6 (*Contd*)

OUTPUT

```
[[1.         1.         0.66666667 1.         1.        ]
 [0.71428571 0.95360825 1.         0.78195489 0.79958571]
 [0.71428571 0.72164948 0.66666667 0.76691729 0.71609206]
 [0.71428571 0.25773196 0.66666667 0.60902256 0.54910795]
 [0.23809524 0.25773196 0.66666667 0.4962406  0.24439201]
 [0.23809524 0.25773196 0.         0.39097744 0.24439201]
 [0.23809524 0.25773196 0.         0.41102757 0.24439201]
 [0.23809524 0.25773196 0.         0.33082707 0.24439201]
 [0.23809524 0.25773196 0.         0.04010025 0.24439201]
 [0.         0.         0.         0.         0.        ]]
```

Data normalization It is the used for scaling the sample data values so that it has unit norm. The normalization is usually performed using either L1 or L2 norm. Listing 10.7 shows the Python code for normalization.

LISTING 10.7: NORMALIZATION USING L2 NORM

```
from sklearn import preprocessing
import numpy as np
import pandas as pd
# set the width of the display in the output
pd.set_option('display.width', 300)
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)
# Read the CSV training and testing data using pandas
dataset=pd.read_csv("/Users/user1/datasets/Occupancy/Occupancy_data/datatraining.txt")
print(dataset.head(10))
# convert to numpy arrays
Xtrain=np.array(dataset.head(10))
X_train=Xtrain[:,1:6]
print(X_train)
# Normalization using L2 norm
X_normalized = preprocessing.normalize(X_train,norm='l2')
print("Normalized data(L2 Norm):",X_normalized )
```

(*Contd*)

Listing 10.7 (*Contd*)

OUTPUT

```
Normalized data( L2 Norm): [[2.76470340e-02 3.25276061e-02 5.08094757e-01 8.60242590e-01
  5.71664827e-06]
 [2.77579995e-02 3.26950865e-02 5.14991827e-01 8.56121454e-01
  5.73558322e-06]
 [2.78323112e-02 3.27555645e-02 5.12162616e-01 8.57812270e-01
  5.74615621e-06]
 [2.79837279e-02 3.28793693e-02 5.14948945e-01 8.56132841e-01
  5.76780149e-06]
```

Data exploration, modeling, and evaluation

Exploratory data analysis (EDA) It is performed to gain a basic intuition and understanding of the data and prepare it for further modeling and analysis. The core tasks involved in EDA are:

Data description Describe the data and understand its various characteristics, so that you get a feel of the data. This is achieved by extracting summary statistics from the data such as count, mean, standard deviation, minimum and maximum values (see Listing 10.8).

LISTING 10.8: EXTRACTING SUMMARY STATISTICS FROM THE DATA

Exploratory data analysis: Extracting summary statistics from the dataset

```
# Import the `pandas` library as `pd`
import pandas as pd
import numpy as np
# Load in the data with `read_csv()`
PData = pd.read_csv("/Users/user1/datasets/pdata/pollutiondata.csv",
   header=0)
# set the width of the display in the output
pd.set_option('display.width', 300)
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)
#print the first 10 rows of the data
five_variables=np.array(PData)
print(PData.head(10))
# Put the data in a DataFrame using Pandas and Convert the data array to numpy float type
df=pd.DataFrame(five_variables[:,0:5].astype(float))
# use the describe function to show the summary statistics of the data
print(df.describe(include=[np.number]))
```

(*Contd*)

Listing 10.8 (*Contd*)

OUTPUT

	0	1	2	3	4
count	49.000000	49.000000	49.000000	49.000000	49.000000
mean	51.163265	53.428571	46.877551	66.040816	79.714286
std	4.579606	4.699291	6.978039	6.928080	5.078550
min	41.000000	45.000000	31.000000	47.000000	68.000000
25%	48.000000	50.000000	42.000000	64.000000	76.000000
50%	50.000000	53.000000	47.000000	67.000000	79.000000
75%	54.000000	55.000000	51.000000	70.000000	84.000000
max	63.000000	69.000000	63.000000	80.000000	90.000000

Sampling the data This is useful for quickly seeing the data samples. Random sample rows or columns can be extracted from the dataset as shown in Listing 10.9.

LISTING 10.9: RANDOM SAMPLING OF THE DATASET

Data sampling

```
# Import the `pandas` library as `pd`
import pandas as pd
import numpy as np
# Load in the data with `read_csv()`
PData = pd.read_csv("/Users/user1/datasets/pdata/pollutiondata.csv",
   header=0)
# set the width of the display in the output
pd.set_option('display.width', 300)
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)
#print the first 10 rows of the data
print("Sampled Data:")
# Sample 10 rows from the total 50 rows of data
print(PData.sample(10))
```

OUTPUT

```
Sampled Data:
    ozone  particullate_matter  carbon_monoxide  sulfure_dioxide  nitrogen_dioxide  longitude   latitude     timestamp
35     50                   50               44               69                72  10.189355  56.182102  8/1/14 3:00
19     55                   54               46               76                74  10.189355  56.182102  8/1/14 1:40
25     50                   53               36               75                84  10.189355  56.182102  8/1/14 2:10
21     52                   53               40               73                74  10.189355  56.182102  8/1/14 1:50
16     47                   59               48               73                77  10.189355  56.182102  8/1/14 1:25
28     50                   49               40               69                84  10.189355  56.182102  8/1/14 2:25
33     46                   56               47               64                68  10.189355  56.182102  8/1/14 2:50
38     50                   50               48               66                76  10.189355  56.182102  8/1/14 3:15
29     55                   51               43               67                81  10.189355  56.182102  8/1/14 2:30
46     57                   47               40               53                84  10.189355  56.182102  8/1/14 3:55
```

Further, it is possible to extract a predefined percentage of random sample data from the dataset.

Listing 10.10 shows how a predefined percentage of sample data can be extracted from the dataset.

LISTING 10.10: SAMPLING BASED ON A PREDEFINED QUANTITY OF SAMPLES

```
# Import the `pandas` library as `pd`
import pandas as pd
import numpy as np
# Load in the data with `read_csv()`
PData = pd.read_csv("/Users/user1/datasets/pdata/pollutiondata.csv",
   header=0)
# set the width of the display in the output
pd.set_option('display.width', 300)
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)
#print the first 10 rows of the data
print("25% Sampled Data:")
samples=PData.sample(frac=.25)
print(samples)
```

OUTPUT

```
25% Sampled Data:
    ozone  particullate_matter  carbon_monoxide  sulfure_dioxide  nitrogen_dioxide  longitude   latitude     timestamp
11     55                   54               51               68                88  10.189355  56.182102  8/1/14 1:00
17     50                   57               47               77                78  10.189355  56.182102  8/1/14 1:30
6      48                   55               55               61                82  10.189355  56.182102  8/1/14 0:35
48     63                   48               44               47                81  10.189355  56.182102  8/1/14 4:05
26     48                   50               40               70                88  10.189355  56.182102  8/1/14 2:15
10     57                   54               54               69                87  10.189355  56.182102  8/1/14 0:55
4      48                   55               63               59                82  10.189355  56.182102  8/1/14 0:25
27     46                   50               42               69                89  10.189355  56.182102  8/1/14 2:20
35     50                   50               44               69                72  10.189355  56.182102  8/1/14 3:00
18     51                   57               45               80                79  10.189355  56.182102  8/1/14 1:35
0      44                   69               57               67                83  10.189355  56.182102  8/1/14 0:05
28     50                   49               40               69                84  10.189355  56.182102  8/1/14 2:25
```

Data querying The data can be further explored by making specific queries. It enables to make selections of the data based on some conditions. This is also a filtering technique to select specific subsets of the data (see Listing 10.11).

LISTING 10.11: QUERYING AND RETRIEVING SPECIFIC SUBSETS OF THE DATA

```
# Import the `pandas` library as `pd`
import pandas as pd
import numpy as np
# Load in the data with `read_csv()`
```

(Contd)

Listing 10.11 (*Contd*)

```
PData = pd.read_csv("/Users/user1/datasets/pdata/pollutiondata.csv",
    header=0)
# set the width of the display in the output
pd.set_option('display.width', 300)
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)
#Query the data and show only those rows where the values of ozone are greater than sulfurdioxide
print(PData.query('ozone > sulfure_dioxide'))
```

	ozone	particullate_matter	carbon_monoxide	sulfure_dioxide	nitrogen_dioxide	longitude	latitude	timestamp
43	56	49	46	54	81	10.189355	56.182102	8/1/14 3:40
44	59	46	41	50	84	10.189355	56.182102	8/1/14 3:45
46	57	47	40	53	84	10.189355	56.182102	8/1/14 3:55
47	62	45	40	52	86	10.189355	56.182102	8/1/14 4:00
48	63	48	44	47	81	10.189355	56.182102	8/1/14 4:05

Data reduction The data reduction approach is used when the dataset has high dimensionality. It would be easier to reduce the number of dimensions using a transformation, which extracts a new reduced set of features from the original dataset. This process can also be termed as feature extraction through transformations. Approaches such as principal component analysis, kernel principal component analysis, linear discriminant analysis (LDA), and multidimensional scaling transform the original features in the dataset and create new set of features based on their combinations, which can aid in creating better predictive models by increasing the information content in the features. Listing 10.12 shows the PCA analysis on a data having a dimensionality of 562. It is reduced to five components, which can be further used for various other analyses such as classification.

LISTING 10.12: DATA REDUCTION USING FEATURE TRANSFORMATION

Dimensionality reduction using principal component analysis

```
# Import the `pandas` library as `pd`
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
# set the width of the display in the output
pd.set_option('display.width', 300)
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)
# Read the CSV training and testing data using pandas
train_data = pd.read_csv("/Users/User1/datasets/activity/train.csv", header=0)
```

(*Contd*)

Listing 10.12 (*Contd*)

```
test_data=pd.read_csv("Users/User1/datasets/activity/test.csv", header=0)
print(train_data.head(10))
# convert to numpy arrays
Xtrain=np.array(train_data)
Xtest=np.array(test_data)
# subset the data so that features and labels are assigned to X_train and y_train
X_train=Xtrain[:,0:561]
y_train=Xtrain[:,562]
print(y_train)
# test data
X_test = Xtest[:,0:561]
# true values of the test labels
y_test=Xtest[:,562]
#preprocess the data
# Standardize the data
X_standardized_data=StandardScaler().fit_transform(X_train.astype(float))
#print("Standardized Data:",X_standardized_data)
# PCA projection
pca=PCA(n_components=5)
# Do the PCA Transformation
PrincipalComponents=pca.fit_transform(X_standardized_data)
pcomponents_df=pd.DataFrame(data=PrincipalComponents,columns=['Principal Component 1','Principal Component 2','Principal Component 3','Principal Component 4','Principal Component 5'])
#pcomponents_df=pd.DataFrame(data=PrincipalComponents,columns=['Principal Component 1','Principal Component 2','Principal Component 3'])
# display the variance explained by each component
print("Explained Variance: ",pca.explained_variance_ratio_)
# Concatenate the labels with the principal components and display
finalDf = pd.concat([pcomponents_df, pd.DataFrame(y_train)], axis = 1)
print(finalDf)
```

```
Explained Variance:  [0.58701172 0.0658068  0.02980437 0.02582953 0.01888285]
     Principal Component 1  Principal Component 2  Principal Component 3  Principal Component 4  Principal Component 5         0
0                -16.138544               2.152624               3.144729              -0.272487               6.709680  STANDING
1                -15.296194               1.387144              -0.682232               2.813658               4.266427  STANDING
2                -15.137819               2.473351              -1.756667               3.717915               4.181649  STANDING
3                -15.354884               3.915681              -1.796296               2.567528               3.205163  STANDING
4                -15.544814               4.598737              -2.188562               2.897649               3.898899  STANDING
5                -15.359102               4.725042              -2.434422               2.527158               2.232916  STANDING
6                -14.986628               3.157020              -1.968583               2.529863               3.410365  STANDING
7                -15.228728               2.888404              -0.419245               2.802432               4.014855  STANDING
```

Feature selection The goal of this approach is to select those features (attributes) that contribute maximum to the estimators' (e.g., classification algorithms) accuracy. Feature selection is a preprocessing technique, which is useful for building robust predictive models. The main contribution of feature selection process is:

- Dimensionality reduction and at the same time increasing the performance of the estimators. This is due to the reason that high dimensional data is complex, increases the training time in the model building process and can also sometimes lead to overfitting.
- Speeding up the learning process of the classification/regression algorithms.
- Reduce the model complexity and make it easy to understand as the number of features is reduced.
- It can result in models with higher accuracy than those without feature selection as only those subset of the features that are relevant to the problem are used in the model and thus have the highest impact on the overall accuracy of the model.

Approaches for feature selection

The main approaches for feature selection are filter-based, wrapper-based, and embedded.

Filter-based methods These methods filter/select the features based on a predefined performance criteria or metric. The subset of features that result from the selection can then be used for further analysis such as classification and regression (see Listing 10.13). The features that are selected by these filter methods can be ranked based on their relevance. Some of the performance metrics that are used by the filtering methods for feature subset selection include information gain, gain ratio, variance threshold, chi-square, correlation, fisher score, etc.

Wrapper-based methods These feature selection methods are called wrapper-based because they consider the actual induction or modeling algorithm that will be actually used for the final modeling task as the performance metric. This is in contrast to the filter-based methods, which do not care about where the selected subset of the features will be used. Examples of wrapper-based approach are using a classifier such as Naive bayes and SVM, which will be used every time a subset of features are selected by the feature selection algorithm (e.g., Genetic algorithm) and tested with the classifier to see if its accuracy has improved for that subset selection. Similarly for feature subset selection for clustering a clustering algorithm (e.g., K-means) will be used as the performance criterion and for a regression task a regression algorithm (e.g., support vector regression, logistic regression, etc.) will be used. High accuracy can be achieved using these methods. See Listing 10.14 for an example of wrapper-based feature selection.

Embedded methods These are the methods where the feature selection is embedded in the model construction process itself. Hence, no separate feature selection process is required. It combines the advantages of the filter- and wrapper-based approaches. Examples of such methods include CART, C4.5, random forests, multinomial logistic regression, etc. Both high accuracy and efficiency can be achieved using these methods as it performs the filtering operation first to obtain several potential subsets of features and then a wrapper-based approach is employed to actually select the most relevant features for the task at hand.

LISTING 10.13: FEATURE SELECTION USING A FILTERING TECHNIQUE

Feature selection (Filter-based approach using Chi-squared test)

```
# importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# set the width of the display in the output
pd.set_option('display.width', 300)
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)
# Read the CSV training and testing data using pandas
dataset=pd.read_csv("/Users/user1/datasets/Occupancy/Occupancy_data/datatraining.txt")
# convert to numpy arrays and select the top 10 rows
Xtrain=np.array(dataset)
X_train=Xtrain[:,1:6].astype(float)
Y_train=Xtrain[:,6].astype(int)
print("X_Train:",X_train)
print("Y_Train:",Y_train)
## Select three features with highest chi-squared statistics
kbest=SelectKBest(score_func=chi2,k=3).fit(X_train,Y_train)

#summarize the scores
np.set_printoptions(precision=3)
features=kbest.transform(X_train)
scores=kbest.scores_
# summarize Selected Features
print("Scores:",scores)
print("Selected Features:")
print (features)
```

OUTPUT

```
Scores: [1.183e+02 1.711e+02 2.127e+06 6.728e+05 1.381e-01]
Selected Features:
[[ 27.272 426.    721.25 ]
 [ 27.267 429.5   714.   ]
 [ 27.245 426.    713.5  ]
 ...
 [ 36.095 433.    798.5  ]
 [ 36.26  433.    820.333]
 [ 36.2   447.    821.   ]]
```

Note: The score indicates the strength of a particular feature out of the five features (temperature, humidity, light, CO_2, and humidity ratio and in the same order). The selected features (3) and their values are displayed for the whole dataset.

LISTING 10.14: FEATURE SELECTION USING A WRAPPER-BASED TECHNIQUE

Feature selection using recursive feature elimination and logistic regression

```
# importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

# set the width of the display in the output
pd.set_option('display.width', 300)
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)

# Read the CSV training and testing data using pandas
dataset=pd.read_csv("/Users/user1/datasets/Occupancy/Occupancy_data/datatraining.txt")

print(dataset)

# convert to numpy arrays and select the top 10 rows

Xtrain=np.array(dataset)

X_train=Xtrain[:,1:6].astype(float)
Y_train=Xtrain[:,6].astype(int)

print("X_Train:",X_train)
print("Y_Train:",Y_train)

RModel=LogisticRegression()
## Select the features using recursive feature elimination with the logistic regression model as the decision function.
rfe=RFE(RModel,4)
fit= rfe.fit(X_train,Y_train)

print("No. of Features:", fit.n_features_)
print("Selected Features:", fit.support_)
print("Feature Ranking:", fit.ranking_)
```

OUTPUT

```
 No. of Features: 4
Selected Features:  [ True  True  True  True False]
Feature Ranking: [1 1 1 1 2]
```

Note: Out of the five features, that is, temperature, humidity, light, CO_2, and humidity ratio, the wrapper-based algorithm chose the first four features.

The clustering and classification approaches, and their evaluation methods, which also come under the modeling and evaluation component of the data science process (refer Fig. 10.2) are explained in Section 10.4.

Model deployment The models that were developed by the processing steps as outlined in Fig. 10.2 and described in the above sections are now ready to be deployed into the production environment. This is the last mile task after which these models will be used for real-world tasks or address a business problem and are expected to perform well and highly scalable. The process of deployment of the models is often related to the cloud infrastructure and the tools and environment that it provides.

The machine learning models and other related tools and computational requirements are all integrated in a platform called the DataOps (Data operations). It provides automation, data access, integration modules and model deployment, and management functionality.

Reporting and visualization Reporting is of three types, that is, static or canned reports, dashboards, and alerts. These are based on the purpose of the report and its reach or coverage of the topic. The canned reports are those that can be generated by analysis tool itself, and extracted by the users of the tools by themselves on a regular basis and sent to other end users based on their requirements. These reports generally have the same structure and could be routine for some people, while, they are specifically useful for certain set of users. The dashboards on the other hand are more focused and can have a very specific set of information shown to a specialized group of people. These dashboards can have several different views and each view can show a different perspective of the analysis for different people, that is, company top management, operational staff, etc. Various visualization tools are built into these dashboards for enhanced understanding of the information. The real-time information is usually reported in the form of warnings, alerts, etc. These are normally brief and are triggered based on a set of predefined criteria. The alerts are usually graded to provide the level of urgency for response activities. For example, during flood disasters, the IoT sensors can be used to capture the water levels and based on the predefined thresholds, warnings and alerts can be sent on a regular basis.

10.2 CONCEPT OF A DATA LAKE/SWAMP

A data lake consists of data that is in its raw and unprocessed form and the data is gathered irrespective of its quality, that is, it retains all the data (both current and historical). The reason behind this is to enable the use of data in ways that were not originally intended or perceived, that is, to go beyond traditional ways of looking at data. The big data approaches usually are aligned with data lake concept and go further than the traditional relational database, which is the foundation of data warehouse (a central repository that integrates business data from heterogeneous sources and optimized for performing analytics). Hadoop, NoSQL kind of approaches are more relevant in this context. The main characteristics of a data lake are:

- Retaining all data to ensure that in some future time, it may be necessary to use that data to gain some unforeseen insights. It also helps to rewind and go back in time to perform analysis.
- Support for various data types/formats, data in the form of web logs, images, videos, sensor data, social network data, etc. is all supported by data lakes. Some of the above kinds of data were

usually not stored in traditional transactional type of data repositories earlier, but the recent acknowledgement of usefulness of such data and the understanding that can be obtained from them resulted in their acquisition, storage in their raw forms, and transformation into a usable format whenever it is necessary.

- Support for various kinds of users, that is, (i) those that just need structured data, (ii) those that go beyond the already transformed data and seek the raw data and combine with data from other sources, and (iii) those that perform more in-depth analysis by integrating various types of data from a variety of relevant sources and creating a new data source and then performing analytics on top it.
- Adaptability to the changing conditions that require a different way of answering questions than it was originally designed for. Since, the data lakes store raw, untransformed data, new novel ways of extracting information from it can be done easily without the need for major rewriting of the analysis tools. This provides the opportunity to obtain more useful insights than was possible earlier using data warehouses, which are more structured and were built for specific end goals in mind.

BOX 10.1: SOME KEY DIFFERENCES BETWEEN DATA WAREHOUSE AND DATA LAKE

Data Warehouse vs Data Lake

Feature	Data warehouse	Data lake
Data storage	Preprocessed data is stored for predefined uses. Takes less storage, but more costly storage infrastructure.	All the data is stored, no specific use is preconceived. Needs a lot more storage, but available at low-cost (i.e., commodity off-the-shelf servers and cheap storage).
Data model	Structured data with well-defined data models and methodology.	Data is stored as it comes in raw form, unstructured, structured, nontraditional data, etc.
Types of users	Applicable to users that have been trained to work with specific data sources and data structures. Cannot go beyond what is offered by the warehouse. Usually, business professional are typical users.	Since data is available in raw form, it is up to the user to curate it to their needs. Hence, more flexibility in transforming data based on the need. Data scientists are typical users.
Accessibility	Because the data is preprocessed, it is easier to understand readily by users, but is limited to using the data for specific purposes only. Changes to the data are not permitted.	Data is in raw form, hence certain level of processing before it can be put to use. Data repositories are more easily accessible due to its unstructured nature, so changes can be made quickly, that is, configured and reconfigured as necessary.
Purpose and adaptability	Well defined purpose for using the data. Cannot go beyond what it was intended to do. Adaptability to new situations, use cases is limited.	No predefined purpose. Can be moulded for use with any tool, application, etc. High adaptability.
Security	The security aspects are more mature and sophisticated in data warehouses.	Security aspects are still evolving. Recently, many robust approaches are in operation.

However, the data lakes over a period of time can become data swamps and unmanageable if care is not taken to ensure data quality and good data governance practices (See Box 10.1).

10.3 RELATION BETWEEN IOT AND BIG DATA

IoT systems in various application domains are generating tremendous amounts of data at a rate that is unprecedented. This data has characteristics of big data in terms of:

Volume/Scale: Millions of devices are being connected to the Internet, connecting people, devices, and applications in a way that is massive in scale.

Velocity: The extremely high rate at which the data is being generated by the IoT devices.

Variety: The data from diverse types of IoT devices is generated in a variety of data models: structured, unstructured, semistructured, etc.

Heterogeneity: The data used for IoT-based solutions is usually gathered from heterogeneous data sources having multiple characteristics and structures.

In addition, other requirements such as big data storage, big data security, and big data analytics are required to efficiently process and manage IoT data. Hence, big data technologies combined with IoT specific technologies related to devices and connectivity are emerging as the solutions for developing IoT applications. In this chapter, the analytics for IoT are specifically discussed from a big data viewpoint.

10.4 BIG DATA ANALYTICS IN IOT

Big data analytics provides a means for analyzing and visualizing data from IoT sensors, actuators, devices, and other connected components of the IoT system. The analytics are useful to understand, summarize, and obtain useful insights from large volumes of data coming at very high speed in the form of streams.

IoT data analytics are useful for:

- Automating many decision-making processes so that human intervention is minimized and IoT devices and applications can autonomously perform actions.
- Increasing the efficiency with which processes can be executed. For example, supply chain operations can be made highly efficient by deploying IoT-based solutions.
- Condition-based monitoring and predictive maintenance of equipment, which is critical in many areas such as industries, manufacturing, healthcare, and transportation.
- Service efficiency that encompasses remote management, service chain, material management, etc.
- Analysis of the product usage by customers and accordingly customize the product thus enabling competitive advantage in the market.
- Reducing overall operational expenditure and increasing revenue.

The data from IoT can take various forms such as structured/semistructured/unstructured, time-series, spatiotemporal (e.g., mobile sensors), etc. Due to such high diversity and having the prime attributes of big data such as volume, velocity, variety, and veracity, the data from IoT requires a different way of processing as compared to traditional data, which is primarily seen as a transactions database and is static to a certain extent. Several data analysis methods that are traditionally used such as preprocessing,

transforming, and filtering, are still applicable on the IoT data albeit the data in the continuous streaming form requires some adaptation of these methods. The analytics can be in the form of:

Descriptive analytics Descriptive analytics provides summaries in the forms of reports, charts, figures, etc. The data from the IoT devices is continuously monitored for providing feedback and enabling situational awareness. It can answer questions such as: What is the current status? What happened earlier or from x days till now? These analytics are based on descriptive statistics that are techniques for summarizing and organizing the information of the data.

Diagnostic analytics Diagnostic analytics can provide the causative factors for a particular problem. The IoT data is deeply examined to extract the reasons for the occurrence of unexpected events, anomalies, etc. In general, it can answer questions such as: Why is it happening that way? What processes led this event to happen?

Predictive analytics Predictive analytics are developed based on the models that are generated using IoT data in conjunction with various methods for building predictive models using AI, machine learning, and deep learning techniques (see Box 10.2) that take into consideration the historical data and can provide future projections. In real-time systems, these analytics can quickly trigger automated reconfigurations, start processes that can recover the system from failure, reduce risk, etc. It can answer 'what if' kind of questions, that is, what will happen in X amount of time, steps, etc.

Prescriptive analytics Prescriptive analytics provides the steps or rules for action so that a particular task can be accomplished in a more efficient manner. It assists in the process of decision making. It can answer questions such as: What are my options? What steps do I need to take? What actions will address this issue?

BOX 10.2: AI, MACHINE LEARNING, AND DEEP LEARNING

Artificial intelligence (AI): According to John McCarthy who first coined the term Artificial Intelligence, it is 'the science and engineering of making intelligent machines, especially intelligent computer programs.'

The goal is to make a computer-controlled system think and possess intelligence similar to a human being. AI techniques have proven to be successful in many domains such as natural language processing (NLP), computer vision systems, robotics, medical domain, industrial applications, intelligent transportation systems, etc.

Machine learning: Machine learning is a field of study that applies the principles of computer science and statistics to create statistical models, which are used for future predictions (based on past data or big data) and identifying (discovering) patterns in data. Machine learning is itself a type of artificial intelligence (Fig. 10.3) that allows software applications to become more accurate in predicting outcomes without being explicitly programmed.

(*Contd*)

Box 10.2 (*Contd*)

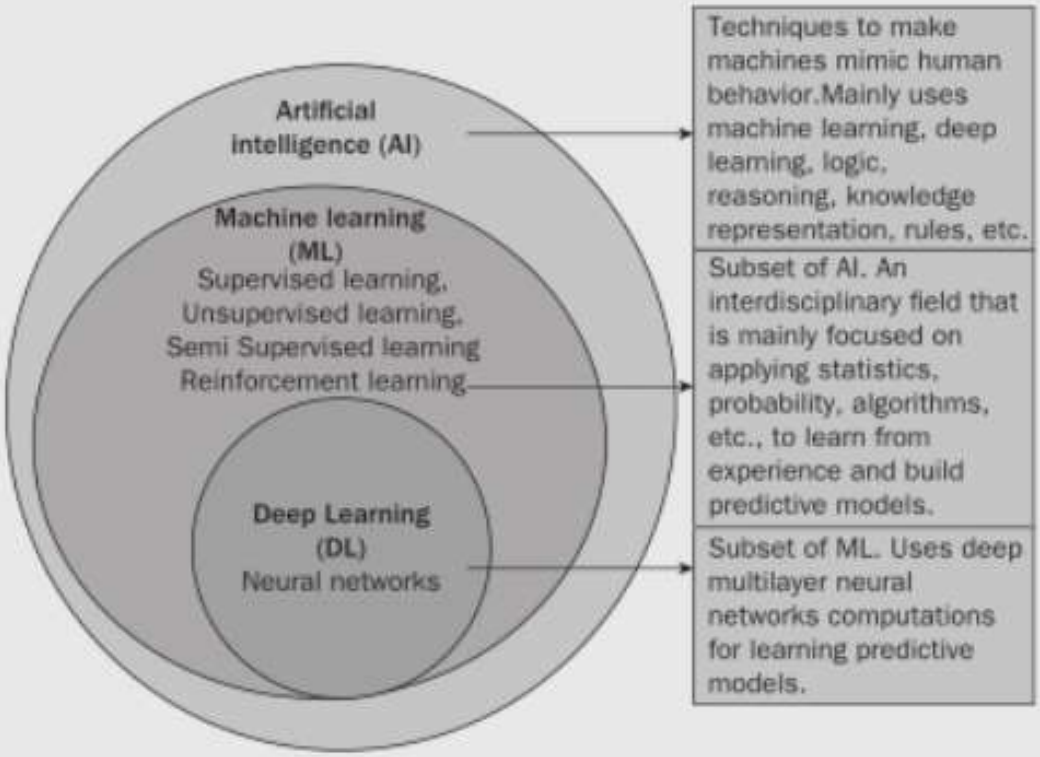


Fig. 10.3 AI, Machine Learning, and Deep Learning

Deep learning: Deep learning is a form of machine learning that is used for supervised, unsupervised, and reinforcement learning. These algorithms are based on deep artificial neural networks and are showing unprecedented new levels of classification accuracy in many domains such multimedia image recognition, audio/video processing, medical images, etc. It is also used commonly for feature extraction and data visualization.

A neural network is a system that tries to mimic the human brain by using layers of connected units (called neurons) to learn relationships based on training data (Fig. 10.4). When there are many hidden layers (deep) with increasing complexity in the network hierarchy, the approach is called deep learning.

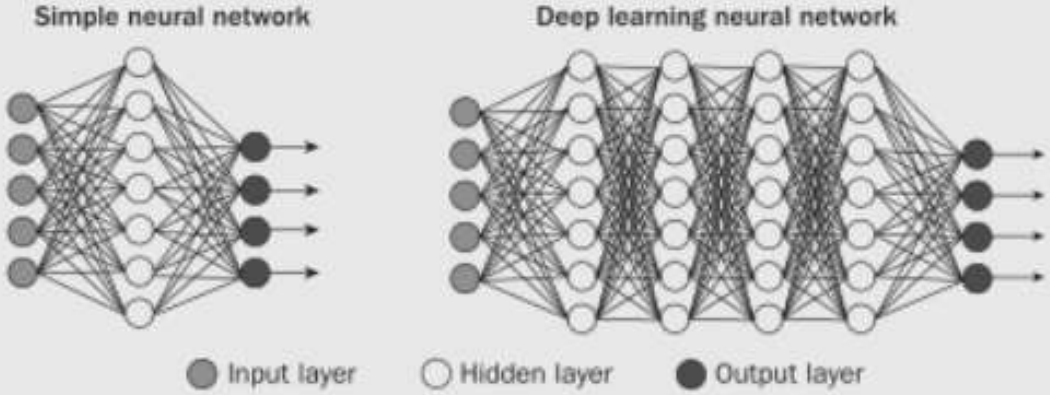


Fig. 10.4 Neural Network Versus Deep Learning

Several deep learning methods are available as shown in Fig. 10.5.

(*Contd*)

Box 10.2 (*Contd*)

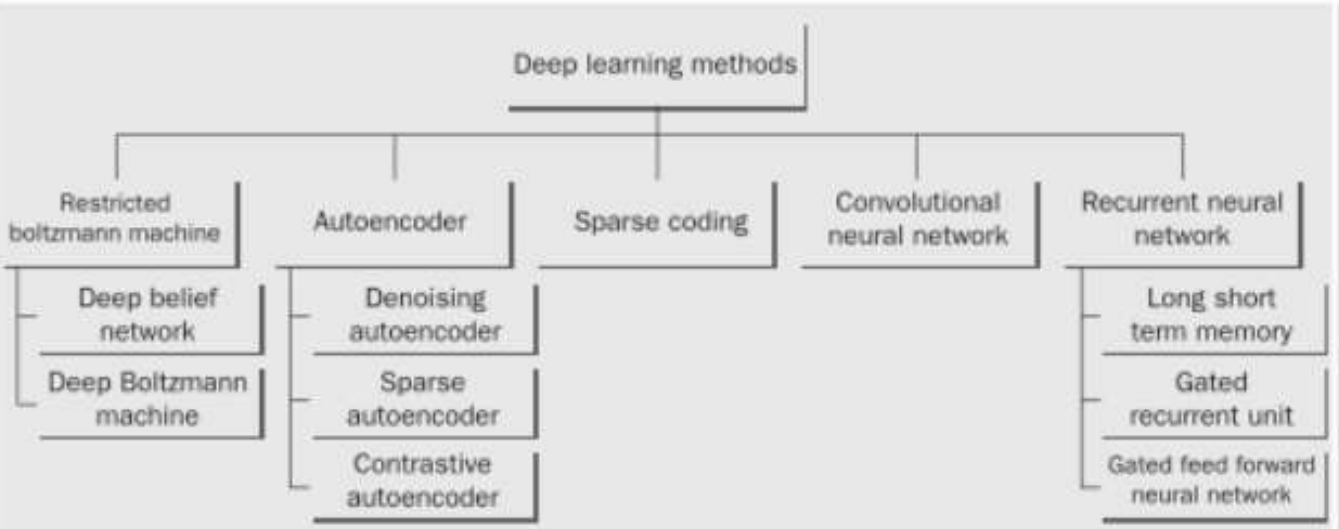


Fig. 10.5 State-of-the-art Deep Learning Methods

The broad classification of analytics in IoT based on the mode of IoT data acquisition (i.e., streaming or static mode) is: (i) real-time/edge analytics and (ii) offline analytics or analytics on the cloud. These are discussed below.

10.4.1 Real-time Analytics

This main characteristic of this type of analytics is its application on continuous streaming data, that is, data that is in motion. The approaches for doing analytics on this type of data can be mainly divided into:

10.4.1.1 Event Processing-based Approaches

These are based on methods such as event stream processing (ESP) and complex event stream processing (CEP). The goal is to capture interesting patterns from data coming from a single or multiple IoT devices and able to send alerts, warnings, etc. in real time. This requires understanding several filters that operate on streaming data using various kinds of predetermined window sizes and their configurations. The reader is referred to Chapter 12 for more in-depth information into these topics.

10.4.1.2 Data Stream Mining Approaches

In this approach, hidden knowledge/patterns are extracted from streaming data. This is different from classical data-mining techniques, where the dataset is static and the algorithm can iterate through the data many number of times to build the model. However, the stream mining approaches cannot do that due to the huge volume and continuously changing nature of the data. The key challenges of data stream mining as compared to traditional data mining are (Fig. 10.6):

Memory bounded The streaming data is continuous and can arrive indefinitely; therefore, the system cannot store the entire stream, but only a small fraction, in some form of summary (using data reduction approaches such as random sampling, histograms, and wavelets) may be stored and the rest discarded. The computations need to be performed (i.e., running the main algorithm) using limited amount of memory; due to this limitation, approximate answers/results are allowed.

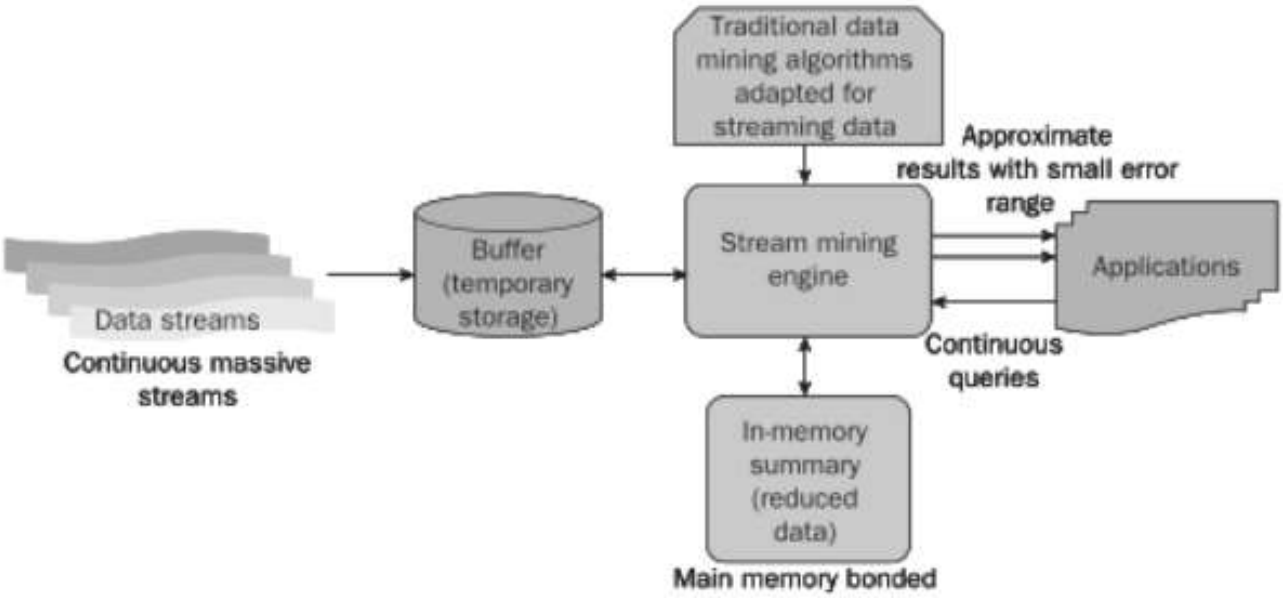


Fig. 10.6 Data Stream Mining Process

Single pass Each record (tuple) is examined only once. Since, it cannot be stored, the possibility of rewinding and looking back at the same data is not possible.

Real-time response The time taken for processing each record should be minimum, that is, rapid processing is a requirement.

Concept drift The patterns that were modeled using the data stream mining algorithms may not remain consistent over a period of time due to the arrival of new data and the possibility of underlying data distribution of that data is different than the previous data stream that was used to build the model. This phenomenon is termed as concept drift, so the stream data-mining algorithms need to be adaptable to these changing conditions and have the capability for anytime predictions.

Stream data-mining applications There are several stream data-mining applications from an IoT viewpoint. In general, any application that requires real-time information obtained from IoT device data streams is a candidate for data stream mining. A few examples are listed below.

- Industrial processes particularly in manufacturing; industrial IoT (IIoT) is an example for the application of data stream mining
- Real-time security monitoring using IoT devices (e.g., video streams)
- Traffic monitoring (continuous traffic-related data transmitted by IoT sensors)
- Real-time disaster monitoring using IoT sensors
- Real-time weather monitoring applications
- Streaming data from sensor networks/social networks
- Live stream monitoring of Smart Agriculture devices

Algorithms for stream data mining

Stream frequent pattern analysis This is a form of stream mining focused on capturing frequently occurring patterns (set of items, subsequences, substructures, etc.) in various types of data

(i.e., structured, unstructured, semistructured, etc.). Windowing is a technique that is most common for frequent pattern analysis. Several kinds of windows are available such as:

Landmark window It captures the frequent item sets from a given starting point to the current point of time.

Sliding window The length of the window is kept constant. At a current time point, the frequent pattern time is captured and moved along with the current time point. Only those patterns are captured that arrived from the current time points are retained.

Damped window This type of window is useful to capture more recently arrived data and is given more weight than the historical data.

Stream clustering The objective of clustering of the streaming data is to find groups of data items that are similar in some way and separate them from other dissimilar data items. These groups or clusters are homogeneous and have distinct characteristics. The clustering algorithms have been developed outside the stream data-mining domain a few decades back and many are being developed currently. Some of these algorithms are being extended/adapted to the streaming data in such a way that they address the challenges as outlined in Section 10.4.1.2. They are classified into (i) partitioning methods, (ii) hierarchical methods, (iii) density-based methods, and (iv) grid-based methods. Figure 10.7 shows some of the currently available stream clustering algorithms.

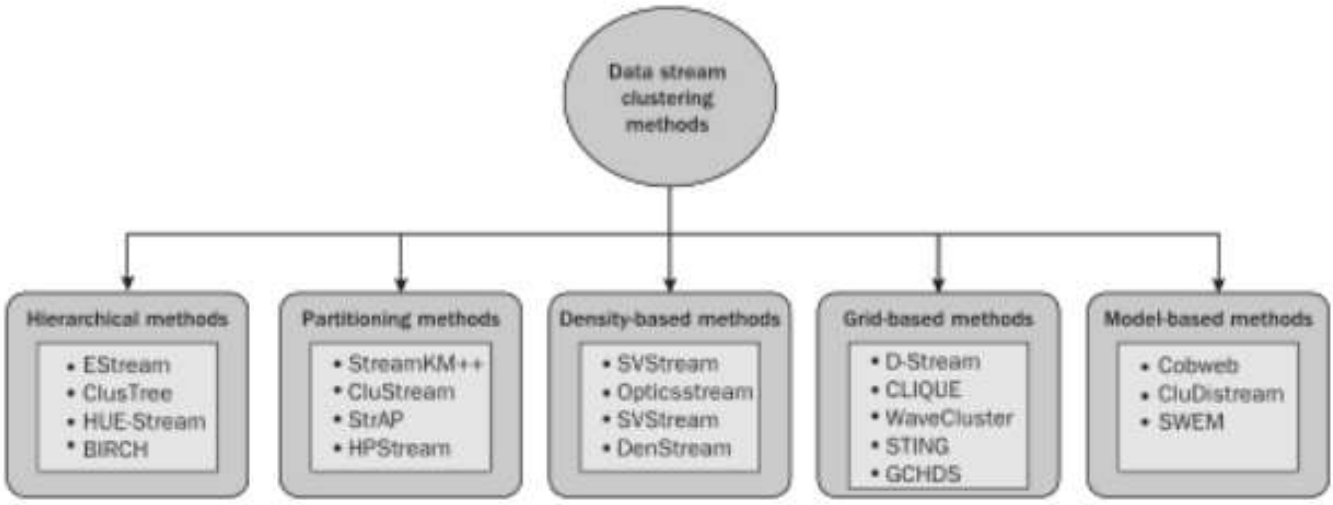


Fig. 10.7 Selected Stream Clustering Algorithms

Stream classification The objective of classification is to assign data to distinct predefined categories called classes. This is achieved by developing a model based on data with known classes (also called class labels) and applying it on new data to automatically assign class labels. This process is basically divided into two steps, that is, training and testing.

In normal data, that is, data that is persistent and residing in a repository, the classification models are built by creating a training dataset, which is used by the classification algorithm to learn the class labels (also termed as batch learning). This learned model is then applied on a test dataset, which assigns a class label for the unlabeled data.

As described earlier in Section 10.4.1.2, several constraints of stream data mining such as memory boundedness, single pass, real-time response, and concept drift makes the development of stream classification challenging. The need for processing huge volumes of streaming data sequentially, the need for high-speed computationally-intensive processing capability, and the restriction on access of the data items only once due to the sequential nature and inability to store the huge volume of data in memory, makes stream classification a unique process. Figure 10.8 lists some state-of-the-art data stream classification algorithms.

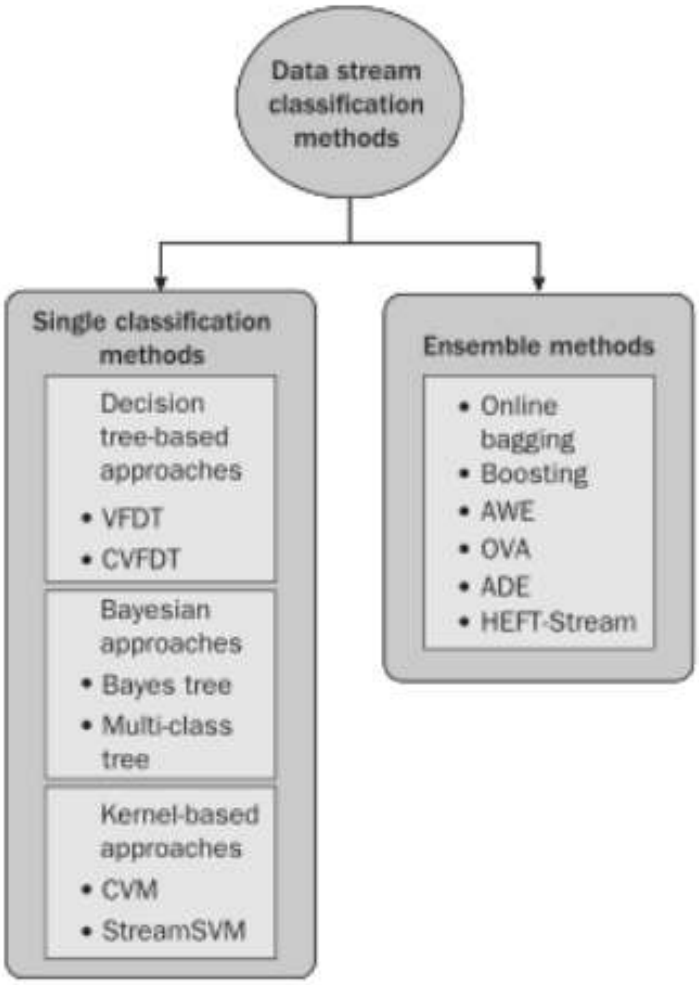


Fig. 10.8 Selected Data Stream Classification Methods

10.4.2 Offline Analytics/Analytics on the Cloud

Offline analytics are those that usually performed on highly scalable computing infrastructures such as cloud computing platforms. These are required for processing large volume of data, which is mainly stored in a repository on the cloud. The various cloud computing platforms that are currently popular are described in Chapter 11. This section is focused on the various analytics that can be performed on the big data obtained from IoT systems and is stored on the cloud. Hence, the assumption is that the data is static and is at rest (the next section describes the analytics on data in motion). The main classes of algorithms for offline analytics which are normally performed on the cloud are described in the subsequent sections :

10.4.2.1 Clustering

Clustering is an unsupervised classification technique (see Fig. 10.9 and Box 10.3) that separates an unlabeled dataset into a number of distinct groups. The clustering techniques can be categorized based on the principle on which the cluster model is built i.e. the way they form clusters or groups.

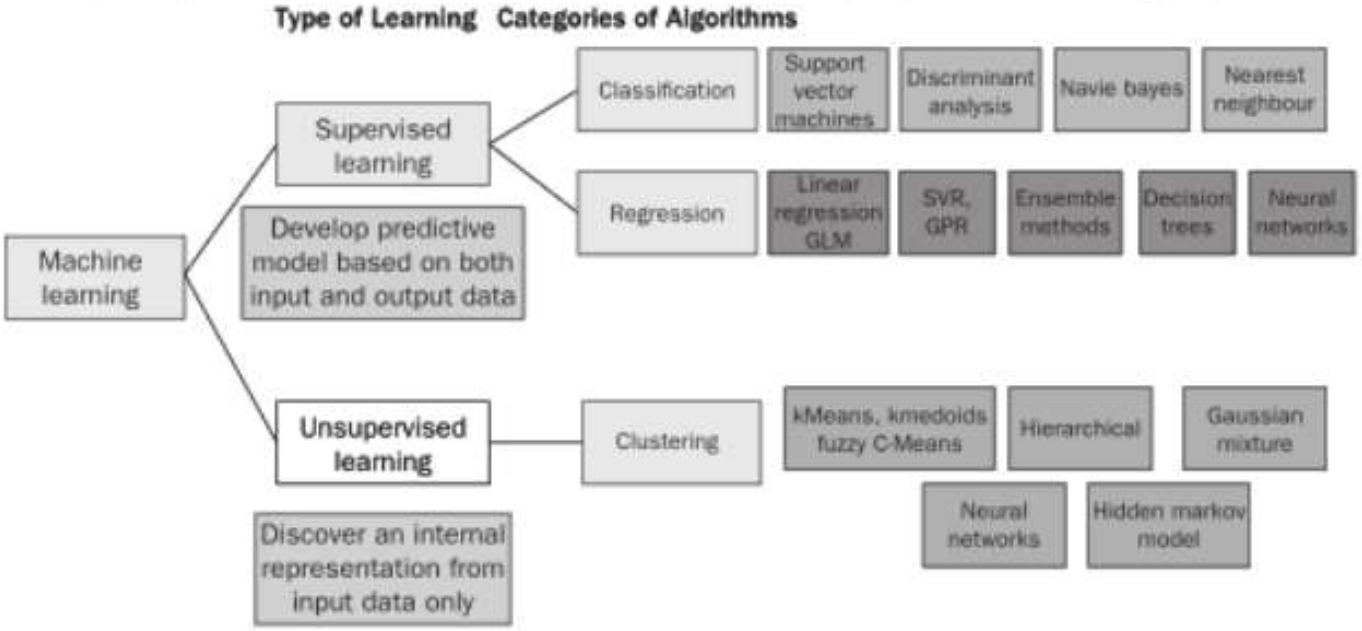


Fig. 10.9 Main Classes of Algorithms for Supervised and Unsupervised Learning

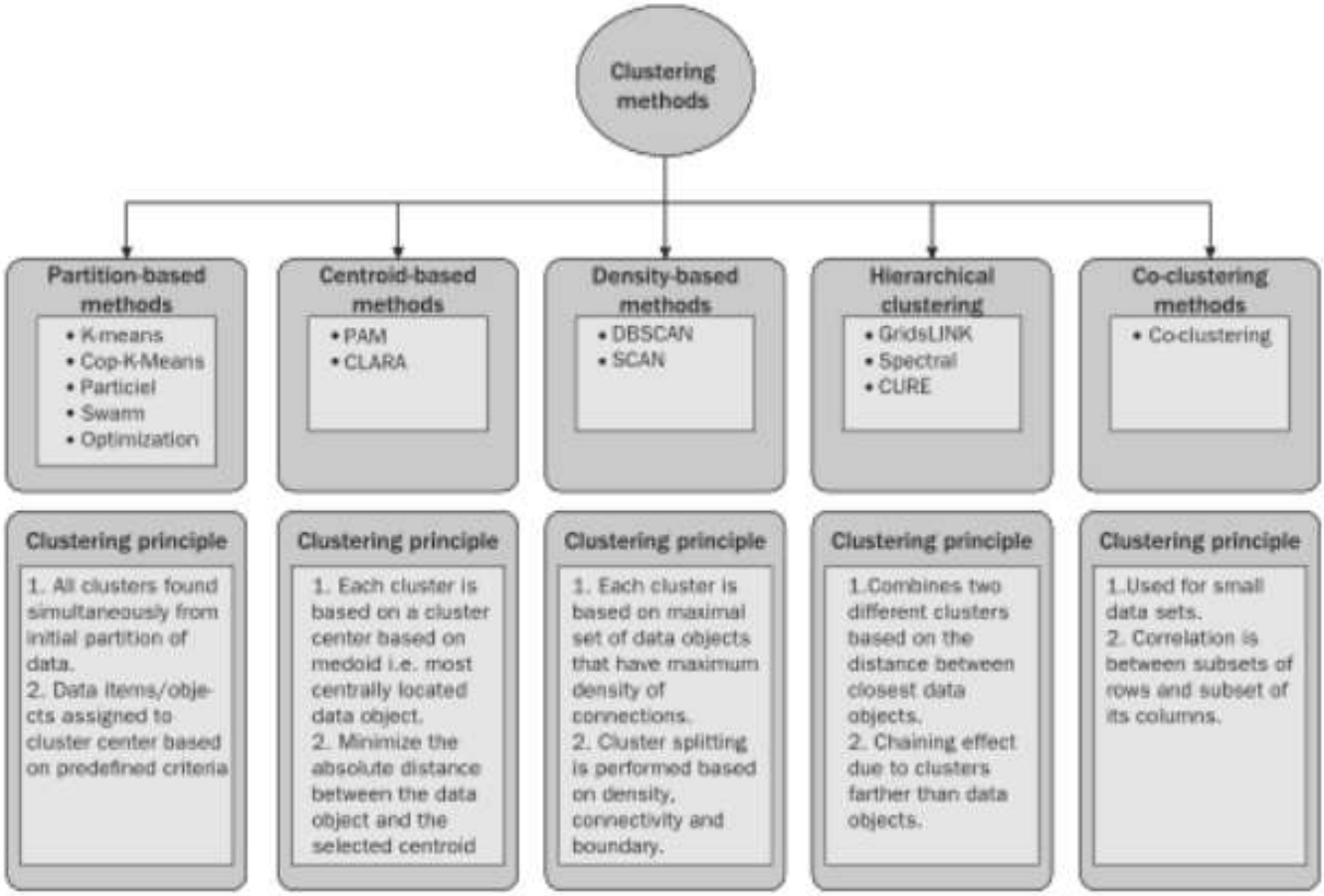


Fig. 10.10 Various State-of-the-Art Clustering Techniques

The K-means algorithm is one of the popular techniques for clustering. Figure 10.10 gives a list of various clustering algorithms and their main principle of clustering.

BOX 10.3: UNSUPERVISED LEARNING

In unsupervised learning, the training data does not have any labels or classes, which tell what that training sample is about. It can be raw data or data with features extracted. The main goal of unsupervised learning is to find some interesting structure and pattern in the data, which is not readily apparent by just looking at the data in its unprocessed form. The unsupervised learning algorithm is designed to find clusters in data that exhibit some form of homogeneity and are separable from each other. It is an approach most commonly employed for visualization of high-dimensional data by projecting it into a lower dimension. It can also be used for assigning unknown data points to existing cluster so that its characteristics can be better understood. Figure 10.11 shows three clusters from the air-quality variables particulate matter and ozone. It can be seen that the clusters are not well defined but rather have a partial overlap.

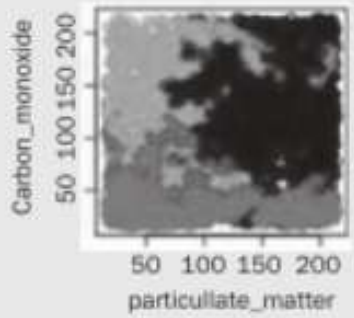


Fig. 10.11 Clustering of Two Air-quality Variables

In IoT, clustering is typically done for data coming from various sensors and there is a requirement to capture some form or structure of the data in terms of their natural groupings. For example, sensor data from various air-quality monitoring sensors is clustered using the K-means algorithm (see Figs 10.12,10.13) as shown in Listing 10.15.

LISTING 10.15: CLUSTERING AIR-QUALITY DATA

K-means clustering using R

Install packages:

Download and install R:
The R project for statistical computing
https://www.r-project.org/

(Contd)

Listing 10.15 (*Contd*)

Install factoextra package as follow:

```
if(!require(devtools)) install.packages("devtools")
devtools::install_github("kassambara/factoextra")
```

The remaining packages can be installed using the code below:

```
pkgs <- c("cluster", "fpc", "NbClust")
install.packages(pkgs)

# Read the data
citypulse <- read.csv(file="/Users/user1/data/pollutionData204273.csv", header=TRUE, sep=",")
str(citypulse)
summary (citypulse)
# Show the top few columns of the variables
head(citypulse)
citypulse_new <- citypulse[,c(1,2,3,4,5)]
#citypulse_new<-scale(citypulse_new)
result <- kmeans(citypulse_new,3)
result$size
result$centers
result$cluster
# prepare for plotting
par(mfrow=c(3,3), mar=c(5,4,2,2))
# Plot 2 variables at a time
plot(citypulse_new[c(1,2)], col=result$cluster)
plot(citypulse_new[c(2,3)], col=result$cluster)
#plot cluster centers
plot(citypulse_new[c(1,2)], col=result$centers)
# plot all possible combinations of the variables
plot(citypulse_new[,], col=result$cluster)
```

OUTPUT

```
> result$centers
      ozone particullate_matter carbon_monoxide sulfure_dioxide nitrogen_dioxide
1 151.78810           168.44307       130.94273       122.76009         119.2661
2  88.56335            77.68161        43.69736        83.72753         147.8420
3  76.14906            80.56786       123.31419       144.81624          62.1788
```

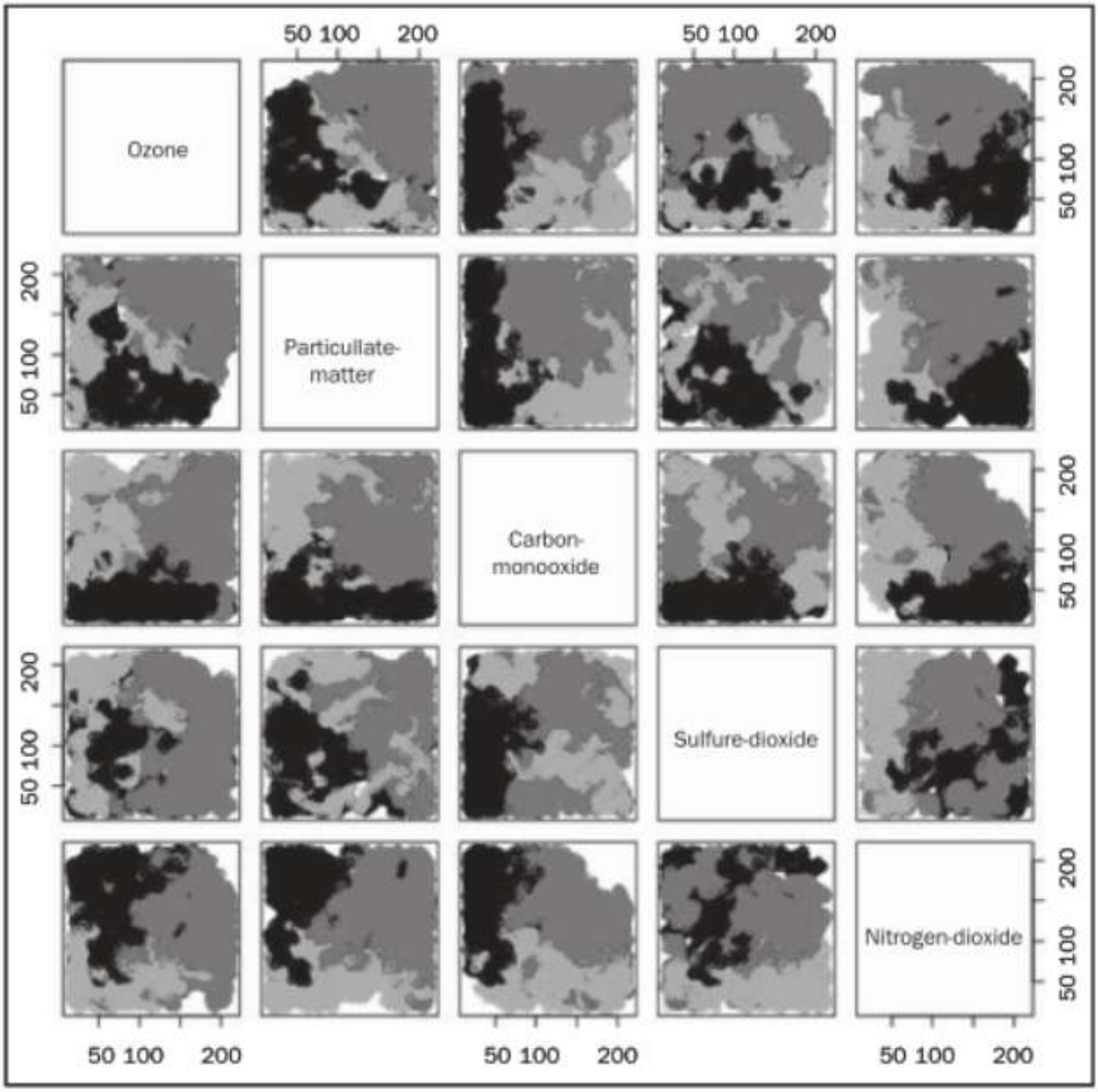


Fig. 10.12 K-means Clustering with $K = 3$

10.4.2.2 Classification

Classification is a supervised learning approach (see Box 10.4) in which a set of labeled data also called as training data is used to learn (by a learning algorithm) a model (hypothesis), which has the capability to predict the class label(s) of unlabeled data.

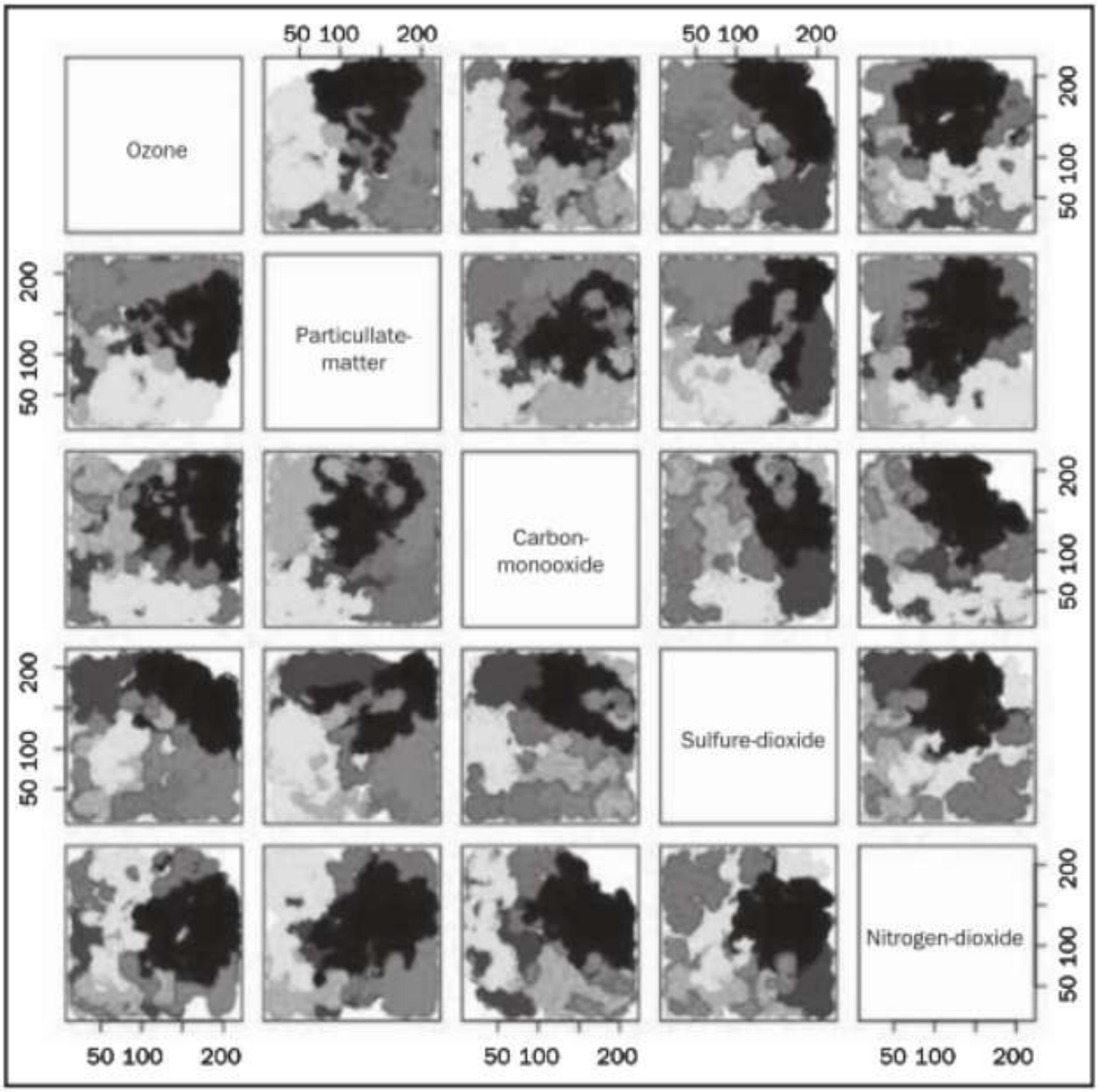


Fig. 10.13 K-means Clustering with $K = 6$

BOX 10.4: CONCEPT OF SUPERVISED LEARNING

Supervised learning: Supervised learning is based on making the learning algorithm learn by giving examples. Once it has seen sufficient number of samples, the algorithm is able to form a hypothesis and is ready to make predictions on data samples that are unlabeled and drawn from similar distribution.

Given a set of input variables (x), an output variable (Y), a learning algorithm is used to learn the mapping function from the input to the output $Y = f(x)$. Once the mapping function is obtained with sufficient accuracy, new input data (x), can be used to predict the output variables (Y) for that data.

Training sample A training sample is a training dataset that can be used in the predictive modeling task. For example, if we are interested in classifying occupancy of a room based on the attributes temperature, humidity, CO_2, the dataset will contain several rows of such data, and the corresponding outcome or class label. Each row in such a training data is called a training sample or training instance or training example. Table 10.1 shows an example of a training data. The goal is to predict the occupancy of the room.

Table 10.1 Sample Training Data Obtained Using Various Sensors in a Room. Each Row is a Training Sample

Temperature	Humidity	CO_2	Room Occupancy (Class Label)
21.1	36.2	821	Yes
20.9	35.68	712.6	Yes
21.84	35.95	1513	Yes
23	27.2	681.5	No
20.2	33.06	448	No

There are many classification algorithms that are available such as naive bayes classifier, support vector machines, decision trees, boosted trees, random forest, neural networks, nearest neighbors, etc. Listings 10.16–10.18 show the application of three classification algorithms (support vector classification, naive bayes, and decision trees) on human activity dataset.

LISTING 10.16: CLASSIFICATION USING SUPPORT VECTOR CLASSIFICATION (SVC)

Classification of human activity using data from mobile phone

Data set description: The Human Activity Recognition database was put up from the recordings of 30 study applicants performing activities of daily living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors. The objective is to classify activities into one of the six activities (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING) performed.

Python code for performing classification using support vector classification (SVC) method

```
# importing necessary libraries
from sklearn.metrics import confusion_matrix
import pandas as pd
import numpy as np
# importing necessary libraries
from sklearn.metrics import confusion_matrix
import pandas as pd
```

(*Contd*)

Listing 10.16 (*Contd*)

```
import numpy as np
from sklearn.metrics import average_precision_score, auc, roc_curve, precision_recall_curve

# Read the CSV training and testing data using pandas
train_data = pd.read_csv("/Users/user1/datasets/activity/train.csv", header=0)
test_data=pd.read_csv("/Users/user1/datasets/activity/test.csv", header=0)
# convert to numpy arrays
Xtrain=np.array(train_data)
Xtest=np.array(test_data)
# subset the data so that features and labels are assigned to X_train and y_train
X_train=Xtrain[:,0:561]
y_train=Xtrain[:,562]
# test data
X_test = Xtest[:,0:561]
# true values of the test labels
y_test=Xtest[:,562]
# training a linear SVM classifier
from sklearn.svm import SVC
# learn the model with a linear kernel, hyperparamter C=1
svm_model_linear = SVC(kernel='linear', C=1).fit(X_train, y_train)
#predictions
svm_predictions = svm_model_linear.predict(X_test)
y_pred = svm_model_linear.predict(X_test)
# model accuracy for X_test
accuracy = svm_model_linear.score(X_test, y_test)
print("Accuracy:",accuracy)
# Set width and number of columns to display in the output
desired_width = 500
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 10)
# Select the true labels that are unique from the test data
unique_label = np.unique(y_test)
# creating a confusion matrix
print(pd.DataFrame(confusion_matrix(y_test, y_pred, labels=unique_label),
    index=['true:{:}'.format(x) for x in unique_label],
    columns=['pred:{:}'.format(x) for x in unique_label]))
```

(*Contd*)

Listing 10.16 (*Contd*)

OUTPUT

```
Accuracy: 0.9640312181879878
                    LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  WALKING_UPSTAIRS
LAYING                 537        0         0        0                   0                 0
SITTING                  0      435        54        0                   0                 2
STANDING                 0       16       516        0                   0                 0
WALKING                  0        0         0      492                   3                 1
WALKING_DOWNSTAIRS       0        0         0        4                 410                 6
WALKING_UPSTAIRS         0        0         0       18                   2               451
```

Note: You can experiment with other kernels such as 'poly' and 'rbf'. Below is an example of using a 'rbf' kernel.

svm_model_l=rbf = SVC(kernel='rbf',gamma=0.2, C=1).fit(X_train, y_train)

Data sourced from UCI machine learning repository : http://archive.ics.uci.edu/ml/datasets.php

LISTING 10.17: NAIVE BAYES CLASSIFICATION

Classification using Naive Bayes

```
# importing necessary libraries
from sklearn.metrics import confusion_matrix
import pandas as pd
import numpy as np
from sklearn.metrics import average_precision_score, auc, roc_curve, precision_recall_curve

# Read the CSV training and testing data using pandas
train_data = pd.read_csv("/Users/user1/datasets/activity/train.csv", header=0)
test_data=pd.read_csv("/Users/user1/datasets/activity/test.csv", header=0)
# convert to numpy arrays
Xtrain=np.array(train_data)
Xtest=np.array(test_data)
# subset the data so that features and labels are assigned to X_train and y_train
X_train=Xtrain[:,0:561]
y_train=Xtrain[:,562]
# test data
X_test = Xtest[:,0:561]
# true values of the test labels
y_test=Xtest[:,562]
# training a Naive Bayes classifier
from sklearn.naive_bayes import GaussianNB
gaussianNB = GaussianNB().fit(X_train, y_train)
gnb_predictions = gaussianNB.predict(X_test)
```

(*Contd*)

Listing 10.17 (*Contd*)

```
#predictions
y_pred = gaussianNB.predict(X_test)
# model accuracy for X_test
accuracy = gaussianNB.score(X_test, y_test)
print("Accuracy:",accuracy)
# Set width and number of columns to display in the output
desired_width = 500
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 10)
# Select the true labels that are unique from the test data
unique_label = np.unique(y_test)
# creating a confusion matrix
print(pd.DataFrame(confusion_matrix(y_test, y_pred, labels=unique_label),
    index=[format(x) for x in unique_label],
    columns=[format(x) for x in unique_label]))
```

OUTPUT

```
Accuracy: 0.7702748557855447
                    LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  WALKING_UPSTAIRS
LAYING                 323      211         0        0                   0                 3
SITTING                  5      368       111        0                   0                 7
STANDING                 8       54       455        0                   0                15
WALKING                  0        0         0      416                  42                38
WALKING_DOWNSTAIRS       0        0         0       80                 257                83
WALKING_UPSTAIRS         0        0         0        9                  11               451
```

*Datasourcedfrom*www.kaggle.com;www.github.com;www.nbviewer.jupyter.org;www.datacamp.com

LISTING 10.18: DECISION TREE-BASED CLASSIFICATION

Classification using decision tree

```
# importing necessary libraries
from sklearn.metrics import confusion_matrix
import pandas as pd
import numpy as np
from sklearn.metrics import average_precision_score, auc, roc_curve, precision_recall_curve

# Read the CSV training and testing data using pandas
train_data = pd.read_csv("/Users/user1/datasets/activity/train.csv", header=0)
test_data=pd.read_csv("/Users/user1/datasets/activity/test.csv", header=0)
# convert to numpy arrays
```

(*Contd*)

Listing 10.18 (*Contd*)

```
Xtrain=np.array(train_data)
Xtest=np.array(test_data)
# subset the data so that features and labels are assigned to X_train and y_train
X_train=Xtrain[:,0:561]
y_train=Xtrain[:,562]
# test data
X_test = Xtest[:,0:561]
# true values of the test labels
y_test=Xtest[:,562]
# training Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
decisiontree_model = DecisionTreeClassifier(max_depth = 8).fit(X_train, y_train)
decisiontree_predictions = decisiontree_model.predict(X_test)
#predictions
y_pred = decisiontree_model.predict(X_test)
# model accuracy for X_test
accuracy = decisiontree_model.score(X_test, y_test)
print("Accuracy:",accuracy)
# Set width and number of columns to display in the output
desired_width = 500
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 10)
# Select the true labels that are unique from the test data
unique_label = np.unique(y_test)
# creating a confusion matrix
print(pd.DataFrame(confusion_matrix(y_test, y_pred, labels=unique_label),
    index=[format(x) for x in unique_label],
    columns=[format(x) for x in unique_label]))
```

OUTPUT

```
Accuracy: 0.8741092636579573
                    LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  WALKING_UPSTAIRS
LAYING                 537        0         0        0                   0                 0
SITTING                  0      372       119        0                   0                 0
STANDING                 0       55       477        0                   0                 0
WALKING                  0        0         0      469                  17                10
WALKING_DOWNSTAIRS       0        0         0       13                 349                58
WALKING_UPSTAIRS         0        0         0       70                  29               372
```

Data sourced from UCI machine learning repository : http://archive.ics.uci.edu/ml/datasets.php

10.4.2.3 Regression

The regression problem is similar to the classification problem, where given a set of input variables (x) and an output variable (Y). A learning algorithm is used to learn the mapping function from the input to the output $Y = f(X)$ except the output variable (Y) is a real or continuous value, S. The target value S is usually predicted based on a set of independent variables. A linear regression is the most common form of regression where there is one independent and one dependent variable as shown in equation below.

$$Y = mX + c + e$$

Where X is the independent variable and Y is the dependent variable. As shown in Fig. 10.14, the relationship between X and Y is established by a best-fit straight line also called the regression line or regression model. The slope and intercept, that is, m and c are found such that the error is minimum between the observed and the predicted values. This approach is called the least-squares approach used for fitting the best-fit line for the observed data by minimizing the sum of the squares of the vertical deviations from each data point to the line.

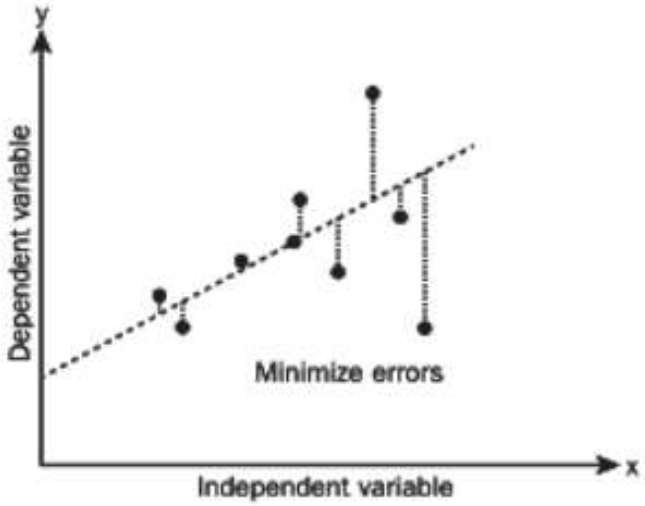


Fig. 10.14 Linear Regression

Various other forms of regression techniques are: logistic regression, polynomial regression, stepwise regression, ridge regression, lasso regression, support vector regression, etc. The regression technique is mainly useful for time-series data from IoT sensors such as air-quality data over a period of time, weather monitoring data, traffic monitoring, and infrastructure monitoring. In all these cases, the output to be predicted is a continuous value and not a categorical variable.

10.4.2.4 Correlation and Pattern Analysis

This type of analytics is often exploratory in nature and mainly focused on the identification of patterns in the data. Correlation analysis is a frequently used approach that gives an understanding about the relationships between various variables (data attributes). The correlation coefficient is a metric that quantifies/measures the strength of the relationship between pairs of attributes. This is obtained by the statistical measure called covariance (see Listing 10.19) that explains the association between two variables X and Y. There are some well-known approaches for correlation analysis such as Pearson's

correlation coefficient, Spearman's correlation coefficient, and Kendall's Tau coefficient. Listing and 10.20 provides examples of measuring the strength of the relationship between air pollution variables using the above-mentioned correlation measures.

LISTING 10.19: COVARIANCE MATRIX

```
#The covariance and covariance matrix are used to characterize the relationships between two or more variables.

# Import the `pandas` library as `pd`
import pandas as pd
import numpy as np

# Load in the data with `read_csv()`
PData = pd.read_csv("/Users/user1/datasets/pdata/pollutiondata.csv",
   header=0)
# set the width of the display in the output
pd.set_option('display.width', 300)
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)
#print the first 10 rows of the data
five_variables=np.array(PData)
print(PData.head(10))
#print(five_variables[:,0:5])
ozone=five_variables[:,0:1].astype(float)
print("Ozone:",ozone)
particulate=five_variables[:,1:2].astype(float)
print("Particulate:",particulate)
co=five_variables[:,2:3]
print("CO:",co)
so2=five_variables[:,3:4]
print("SO2:",so2)
no2=five_variables[:,4:5]
print("NO2:",no2)

cov_oz_no2 = np.cov(ozone.astype(float).T, no2.astype(float).T,bias=True)
print("Covariance (Ozone,NO2:",cov_oz_no2)

cov_oz_particulate = np.cov(ozone.astype(float).T, particulate.astype(float).T,bias=True)
print("Covariance (Ozone,Particulate:",cov_oz_particulate)
```

(*Contd*)

Listing 10.19 (*Contd*)

OUTPUT

```
Covariance (Ozone,NO2: [[20.54477301  6.74052478]
 [ 6.74052478 25.26530612]]
Covariance (Ozone,Particulate: [[ 20.54477301 -13.33527697]
 [-13.33527697  21.63265306]]
```

Note: Ozone and NO2 show positive correlation but Ozone and particulates show negative correlation.

Exercise: One important assumption in generating the covariance matrix is that the data is normally distributed. To make the data distributed use the function preprocessing. PowerTransformer (method='box-cox', standardize=False) in 'sklearn'. Perform it on each of the variables and then do the covariance matrix estimation.

Pearson's correlation coefficient measures the linear association between continuous variables. In other words, this coefficient quantifies the degree to which a relationship between two variables can be described by a line commonly used in linear regression and has been used to measure the strength of relationships between the four air pollutants. Spearman's correlation coefficient can be used to find the strength between two data variables when they have a nonlinear relationship and also non-gaussian Gaussian distribution (See Listing 10.20).

LISTING 10.20: PEARSON'S AND SPEARMAN'S CORRELATION COEFFICIENTS

Pearson's correlation coefficient

```
#Pearson's correlation coefficient = covariance (X, Y) / (std(X) * std(Y))
#Provides the strength of the linear relationship between two data samples.
# Import the 'pandas' library as 'pd'
import pandas as pd
import numpy as np
from scipy.stats import pearsonr
# Load in the data with 'read_csv()'
PData = pd.read_csv("/Users/user1/datasets/pdata/pollutiondata.csv",
   header=0)
# set the width of the display in the output
pd.set_option('display.width', 300)
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)
#print the first 10 rows of the data
five_variables=np.array(PData)
print(PData.head(10))
```

(*Contd*)

Listing 10.20 (*Contd*)

```
#print(five_variables[:,0:5])
ozone=five_variables[:,0:1].astype(float)
print("Ozone:",ozone)
particulate=five_variables[:,1:2].astype(float)
print("Particulate:",particulate)
co=five_variables[:,2:3].astype(float)
print("CO:",co)
so2=five_variables[:,3:4].astype(float)
print("SO2:",so2)
no2=five_variables[:,4:5].astype(float)
print("NO2:",no2)
# Calcluate the Pearson's Correlation Coefficient
corr, _ = pearsonr(ozone, particulate)
print('Pearsons correlation (Ozone,Particulate Matter): %.3f' % corr)
corr, _ = pearsonr(ozone, no2)
print('Pearsons correlation(Ozone,NO2): %.3f' % corr)
```

OUTPUT

```
Pearsons correlation (Ozone,Particulate Matter): -0.633
Pearsons correlation(Ozone,NO2): 0.296
```

Note: There is a positive correlation between ozone and NO2 but it is very weak as it is 0.296. Any value above 0.5 can be considered as having some significant linear relationship.

Spearman's correlation coefficient

```
#Spearman's correlation coefficient = covariance(rank(X), rank(Y)) / (std(rank(X)) * std(rank(Y)))
# Import the `pandas` library as `pd`
import pandas as pd
import numpy as np
from scipy.stats import pearsonr
from scipy.stats import spearmanr
# Load in the data with `read_csv()`
PData = pd.read_csv("/Users/user1/datasets/pdata/pollutiondata.csv",
   header=0)
# set the width of the display in the output
pd.set_option('display.width', 300)
# set the number of columns to show in the output
pd.set_option('display.max_columns', 10)
```

(*Contd*)

Listing 10.20 (*Contd*)

```
#print the first 10 rows of the data
five_variables=np.array(PData)
print(PData.head(10))
#print(five_variables[:,0:5])
ozone=five_variables[:,0:1].astype(float)
print("Ozone:",ozone)
particulate=five_variables[:,1:2].astype(float)
print("Particulate:",particulate)
co=five_variables[:,2:3].astype(float)
print("CO:",co)
so2=five_variables[:,3:4].astype(float)
print("SO2:",so2)
no2=five_variables[:,4:5].astype(float)
print("NO2:",no2)
# Calcluate the Pearson's Correlation Coefficient
corr, _ = pearsonr(ozone, particulate)
print('Pearsons correlation (Ozone,Particulate Matter): %.3f' % corr)
corr, _ = pearsonr(ozone, no2)
print('Pearsons correlation(Ozone,NO2): %.3f' % corr)
# Calculate Spearmans correlation coefficient
corr, _ = spearmanr(ozone, particulate)
print('Spearmans correlation(Ozone,Particulate Matter): %.3f' % corr)
corr, _ = spearmanr(ozone, no2)
print('Spearmans correlation(Ozone,NO2): %.3f' % corr)
```

OUTPUT

```
Spearmans correlation(Ozone,Particulate Matter): -0.517
Spearmans correlation(Ozone,NO2): 0.277
```

10.4.3 Big Data Analytics Platforms for IoT

The most popular analytics platforms that are currently available are:

- Microsoft Azure Stream Analytics
- AWS IoT Analytics
- IBM Watson Analytics
- Cisco Data Analytics

- Oracle Stream Analytics and Oracle Edge Analytics
- Google Cloud IoT

For more information on these platforms please refer to Chapter 8 on IoT Software Platforms.

10.5 MACHINE LEARNING AND DEEP LEARNING TOOLS

IoT data analytics can be performed using many ready-to-use tools that are freely available. The most popular among them are described below.

10.5.1 Tensorflow

TensorFlow is an open source software library for numerical computation using data flow graphs. Graph nodes represent mathematical operations, while graph edges represent the multidimensional data arrays (tensors) that flow between them. This flexible architecture enables you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device without rewriting code. TensorFlow also includes TensorBoard, a data visualization toolkit.

10.5.2 Theano

Theano is a Python library that allows defining, optimizing, and evaluating mathematical expressions involving multidimensional arrays efficiently. It can use GPUs and perform efficient symbolic differentiation. It is tightly integrated with the Python package NumPy and can use GPU for significantly speeding up computations than on a CPU.

10.5.3 Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Keras is highly modular, that is, the various components (neural layers, cost functions, cost function, etc.) are available as stand-alone modules and it is up to the user to select those that are required and connect them together to create new models. It supports convolutional networks as well as recurrent networks.

10.5.4 Scikit-learn

It is a machine learning library written in Python programming language. It provides various classification, regression, and clustering algorithms. It is tightly integrated with the Python numerical and scientific libraries NumPy and SciPy.

SUMMARY

The IoT analytics forms one of the most important modules of the IoT solution. It enables to gain insights and enables to solve a business problem. IoT analytics are based on the big data paradigm since the data generated by IoT has similar characteristics. The whole data analytics comes under the umbrella of data science; hence, this chapter focused on introducing the data science and the end to processes involved in it.

IoT in the Cloud

CHAPTER 11

"Cloud is about how you do computing, not where to do computing."

—Paul Martiz

OBJECTIVES

- discuss the background of cloud computing and Internet of Things (IoT)
- explain the integration of cloud computing and IoT
- study the cloud services for IoT
- review selected cloud service providers for IoT applications
- learn about open source cloud platform ThingSpeak for IoT

OUTCOMES

- gain knowledge about cloud services for Internet of Things (IoT)
- explain the integration of cloud computing and IoT
- list various cloud services useful for IoT applications
- outline popular cloud platforms and how to integrate with complex IoT applications
- describe RESTful APIs useful for real-time IoT analytics

REVISION

In Chapters 8, 9, and 10, different Internet of Things (IoT) platforms, programming concepts, and prototyping concepts are explained. Further in this chapter, we will explore the use of different cloud services and communication models required for large and complex IoT applications.

11.1 INTRODUCTION: CLOUD COMPUTING AND IOT

The computing power required these days has increased considerably for processing large volumes of data. Earlier, mainframe computers were rented by businesses and organisations for their computing needs, but with the advent of the PCs, the way people/ organisations/ businesses have used computers

for various needs has drastically changed. However, the recent big data era has ushered in new requirements for computing to cater to the needs of processing high volume, velocity and variety of data. It led to the shifting of the computing infrastructure from desktop to remote services that can provide highly scalable computing infrastructure and can be charged by a subscription plan.
Majority of IoT applications produce high frequency and volume of data. These data are processed at various levels such as:

- Edge/fog computing, where streaming raw data from IoT sensors is processed for real/ near real-time applications. These computations usually happen on an specialised edge device integrated with an IoT gateway or on nearby available servers.
- Cloud comma after computing where the data is further sent to a remote location hosting a large network of servers and capable of providing various services such as hardware, software, storage, processing, security, etc.

Chapter 12 provides more details about edge computing for IoT. This chapter is focused on the second approach (cloud) for IoT data processing.
Some major advantages of using cloud for IoT includes:

- On demand processing of high volumes of data
- Availability of latest computing hardware and software that enables more efficient processing
- High scalability and availability that allows aggregation and processing of data from multiple IoT applications and distributed IoT sensors
- Payment of the cloud services such as data storage, processing, etc. on the basis of "pay as you go" approach
- No maintenance is required at the user end for the computing infrastructure as it is managed by the cloud service provider
- Several options for choosing off-the-shelf (commercial and open source) IoT platforms to quickly setup the required infrastructures for integrating the IoT sensors with the processing hardware/ software and begin gaining useful insights in a particular domain.

All the above is possible assuming that sufficient internet bandwidth is available with the IoT data producers. Also, some latency and occasional disruption of services is expected in the cloud service.

11.1.1 Evolution of Cloud-based Novel IoT Applications

IoT is used for developing objects such as physical devices, vehicles, etc. with embedded hardware, software, sensors, and network connection to collect, send, and receive data.

The market expansion for IoT applications is displayed in Fig. 11.1.

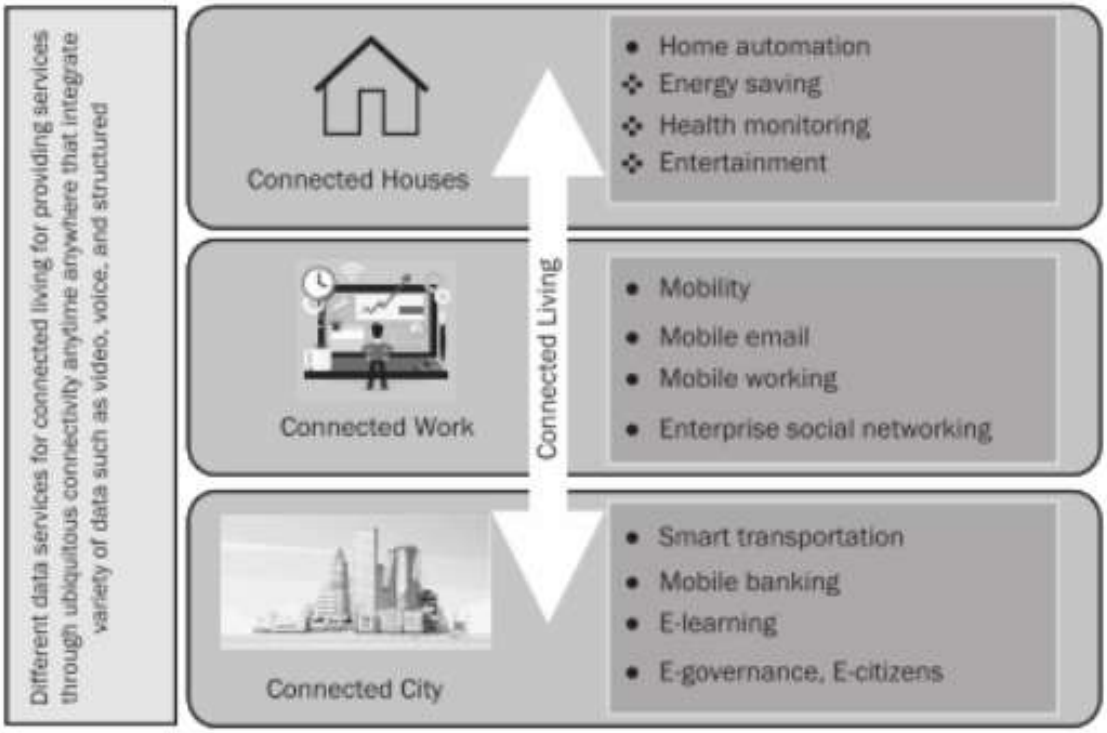


Fig. 11.1 Connected Living with Smart Things

High volumes of data generated by various IoT applications can cause heavy stress to the local computing and internet infrastructure. Consequently, several organisations are migrating their data to the cloud to reduce the load on the local devices, thus enabling secure storage and for availing various functionalities offered by cloud services. Generally, cloud providers allow for data transfer by the Internet or by a dedicated data link for secure transmission of the data, thus providing high quality of service.

Cloud computing provides on-demand service for computing power, data storage, software and applications, and other IT resources. Using the various resources in such a way on the cloud organisations resembles and works like a virtual machine, reducing significant investment on the infrastructure. In general, organisations decide on the need for a cloud service based on the application's computing and data requirements. Based on it, the organisation can decide on a subscription plan.

"Pay as you grow (PAYG)" is another model that is relevant to small and medium enterprises (SMEs). As the business grows and more IoT system deployments and aggregation of data from multiple domains is required, the need perceived for more computing resources can be easily scalable by the PAYG cloud service model.

Cloud computing plays an important role in IoT as it allows massive data generated by IoT to be handled efficiently without the need for expensive local infrastructure. Hence, cloud computing and IoT are inseparable. IoT has gradually changed the way daily tasks are performed. In smart home applications, people can start their cooling devices through their mobile phones from a remote place before reaching home. Cost-effective solutions are possible due to cloud integration which enables cities to host these data and applications. In a smart city, many IoT applications are likely to be deployed, such as applications for smart transport management, smart energy management, urban mobility of the citizens, smart water management, and more (refer to Fig.11.2).

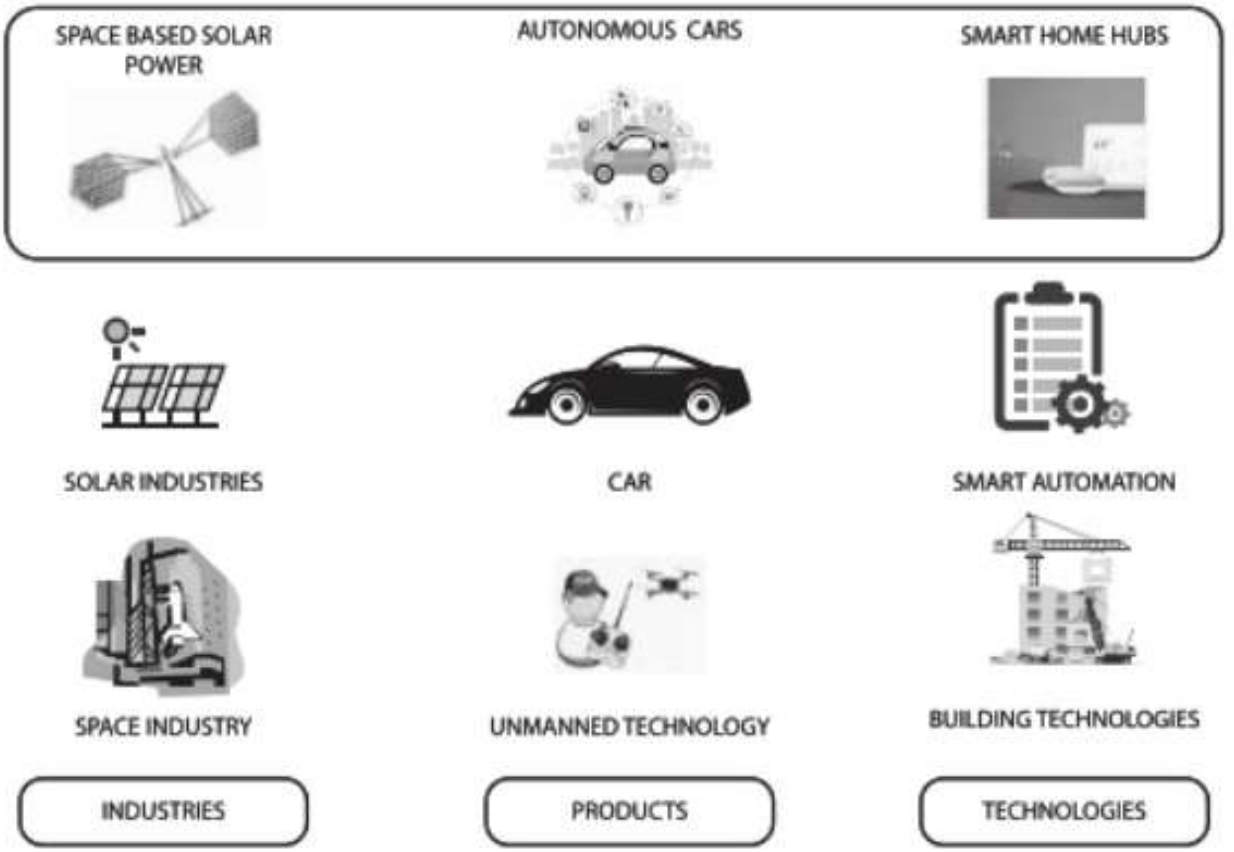


Fig. 11.2 IoT Application Domains

11.1.2 Cloud Computing Service Models

For many applications, cloud computing needs to be integrated with another platform of the cloud or even another cloud provider as shown in Fig. 11.3. Cloud computing services are categorized by different levels at which they are provided and by underlying APIs. Following are some of the common services.

IaaS APIs (Infrastructure-level) Infrastructure-as-a-Service (IaaS) help in management of cloud resources and their distribution. Infrastructure APIs are useful for provisioning or de-provisioning of cloud resources. In addition, network configurations as well as workload management are provided by these APIs. Examples of IaaS include AWS EC2, Google compute engine, etc.

PaaS APIs (Service-level) Platform-as-a-Service (PaaS) provide access and functionality in a cloud environment. Storage, database integration, portals, messaging systems, and many other components are available through these APIs. Examples of PaaS include Google App engine, AWS elastic, Azure, Apache Stratos, etc.

SaaS APIs (Application-level) SaaS consists of cloud-hosted applications which can support multiple users simultaneously. Software-as-a-Service APIs are useful to connect the application layer with the cloud Infrastructure. Examples of Saas include Google Apps, DropBox, etc.

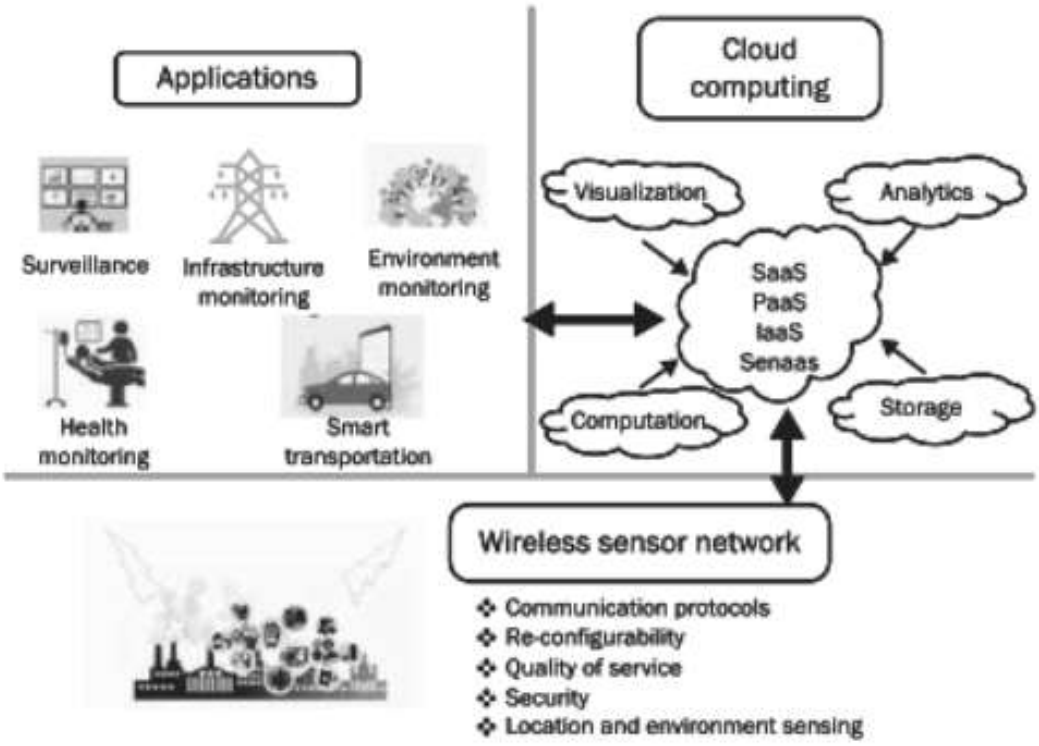


Fig. 11.3 Cloud Computing Service Model

Cloud provider and cross-platform APIs Many cloud providers are providing services to a variety of environments. Hence, cross-platform compatibility is needed. It can enable the usage of resources and workload across multiple cloud services.

An organisation's cloud services have to be assessed properly before selecting one or multiple could services, as sometimes the requirements are spread across multiple cloud providers to satisfy the resource needs of a particular application. However, often lack of interoperability between various cloud services is a big issue that is hampering these efforts of working across multiple cloud services. Hence, compatibility should be checked for software, scalability, redundancy, and a particular functionality available for a specific version and geographical area.

The need to integrate with different services and platforms has generated new market opportunities and such distributed IoT platforms are increasingly becoming popular. Many market leaders have emerged as the cloud API providers. Some are listed below:

- Simple Cloud
- Apache (Citrix) CloudStack
- Amazon Web Services API and Eucalyptus
- Google Compute Engine
- VMware vCloud API
- OpenStack API

Though every platform has its own benefits and challenges, some functionalities and services in them are common. For example, some common API models such as OpenStack API and AWS API are supported by the CloudStack model and VMware vCloud API. The cloud environments with secure and multiple-user support help to create a robust infrastructure that enables scalability.

11.2 INTEGRATING CLOUD COMPUTING WITH IOT

IoT systems require storage of huge volume of data as well as high computational power for processing and analysis. Cloud services can cater to such needs and can significantly increase the efficiency of many IoT tasks and provide tremendous support for IoT systems (refer to Fig. 11.4).

Cloud can also facilitate to work on collaborative tasks with several developers engaged remotely in a common cloud environment to develop applications. The big data storage facility of the cloud allows various stakeholders to access and retrieve relevant data on demand. Both established and IoT startups can leverage the pay-as-you-use model of the cloud to optimise their return on investment. Thus, the cloud can act as an important middleware between the things in the IoT and IoT based applications. The next section discusses some cloud based tools for big IoT data analytics.

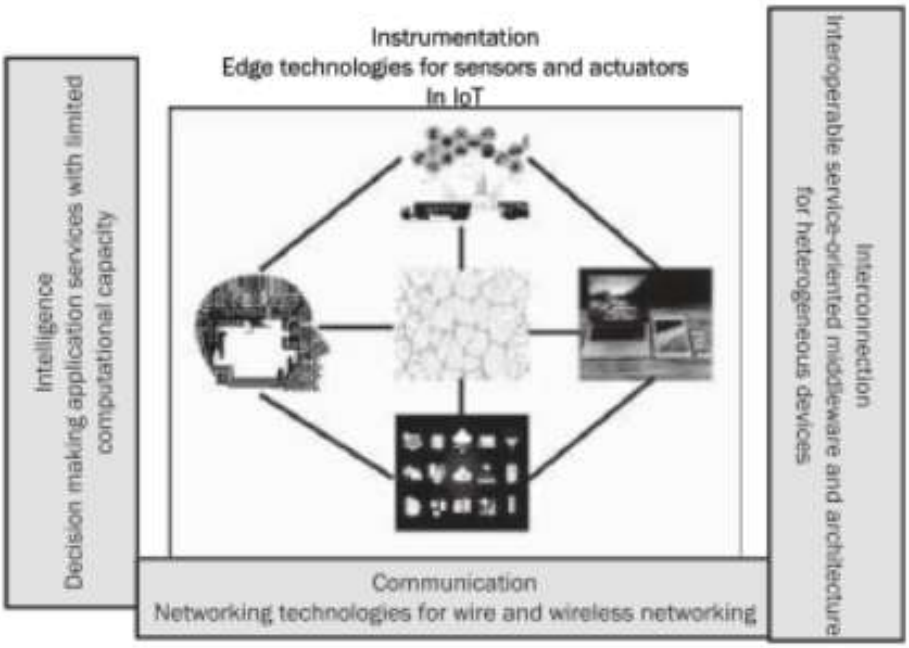


Fig. 11.4 Integration of IoT and Cloud

11.2.1 Apache Hadoop

Hadoop is an open source platform from Apache consisting of an ecosystem of tools, libraries and services. For example, it provides services for storage, NoSQL, batch as well as streaming data processing, etc. Hadoop is composed of different modules that create the Hadoop framework as mentioned below:

- Hadoop Common
- Hadoop Distributed File System (HDFS)
- Hadoop MapReduce
- Hadoop YARN

Several other modules such as Avro, Cassandra, Hive, Pig, Oozie, Flume, and Sqoop, are also useful to enhance Hadoop's power further and are useful for processing large data. Hadoop can run on commodity hardware cluster or in the cloud. The Hadoop Distributed File System (HDFS), allows access of IoT application data at high-speeds through parallelization approaches.

11.2.2 Apache Spark

Apache Spark is a general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python, and R. It also has optimized engine that supports general execution graphs. Different high level tools supported are SparkSQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming. Spark can also perform batch processing; however, it is used generally for streaming workloads, machine-based learning, and interactive queries. Spark possesses real-time data-processing capability and it is much faster than MapReduce's batch-processing engine. Spark has good compatibility with Hadoop and its modules. It also has a standalone mode and can work independently. Spark is a cluster-computing framework; hence, its functionality is supported by MapReduce. Spark does not have its own distributed filesystem but it uses HDFS. Spark uses Resilient Distributed Datasets (RDDs) and the MapReduce uses persistent storage. Spark is well known for its user friendly environment and good performance as it has many user friendly APIs for Spark SQL, Java, Scala (which is its native language), and Python. Spark provides an interactive mode while MapReduce does not have this feature. Hence, Spark developers as well as users can have immediate feedback for queries and other actions. Spark offers greater speed, agility, and ease of use while MapReduce provides low cost of operation. The property of real time processing makes Spark more suitable for real-time edge analytics in IoT applications.

11.3 CLOUD SERVICES

The following sections discuss the various cloud services.

11.3.1 SEaaS: Sensing-as-a-Service

SEaaS is related to cloud services that provide sensing data collected by cloud enabled sensors on demand. This data can be shared by different applications. From a business standpoint, this service model acts as a market place for sensing data, that can be bought for a price. Hence, a passive secondary source of income for a business which has already deployed IoT systems. Some areas in which this is useful include transportation, healthcare, social networking, environment/weather monitoring, etc. where the sensing data can be made available as a service either free or for a price. Several commercially available wearables also have this kind of model for sharing the sensing data. An example query such as find all people who have gone for a walk between 6 and 7 am at a particular location. Assuming that walkers having wearables are uploading data to a particular sensing-as-a-service cloud provider, it is easy to answer such questions.

11.3.2 SAaaS: Sensing and Actuator-as-a-Service

It is related to cloud-enablement of sensors and actuators through virtualisation so that they can be offered as a service. From an IoT stand point, this is vital in creating a more ubiquitious and pervasive

computing model of IoT. Many innovative services can be developed that integrate clouds with sensor networks and actuators. An example scenario of the usefulness of SAaaS is forest fires monitoring. IoT sensors related to forest fires monitoring are made available as a cloud service and the alarms and notifications to fire departments as well as public are triggered based on a certain threshold of the fire intensity. These alerts and warnings could be in the form of sirens, alarms, message notifications which act as actuators and are also cloud enabled. Thus both sensors and actuators are available on demand, so developers can create new applications for authorities as well as citizens, that will help them to gain situational awareness in real-time about the disaster.

11.3.3 SEaaS: Sensor Event-as-a-Service

It is the triggered based on the events that are captured from IoT data. These could be events based on real-time sensor data such as temperature, traffic density, water levels, etc. The events could also have been generated by multiple sensors and a pattern that emerged out of their data. Chapter 12 discussed about event processing of streaming IoT data using complex event processing (CEP). The detected events can be used by a cloud-enabled service to generate messages and notifications. Further, users can register sensors for events capture and subscribe and receive the alerts based on the events. Some examples include events generated in supply chain, air quality, traffic monitoring, etc., and the subsequent notification of important alerts to the personnel/ users in these domains.

11.3.4 SenaaS: Sensor-as-a-Service

The IoT sensors, both physical and virtual (e.g., models), can be enabled through virtualisation approaches as a service that exposes the high level functions and capabilities of the sensors and hiding the complex technical aspects (e.g., communication protocols) of the sensors. The cloud middleware is responsible for the ubiquitous sensor management, authorisation, privacy, discovery of sensors that are useful for a particular task/application.

11.4 SELECTED CLOUD SERVICE PROVIDERS

When it comes to selecting cloud platforms for IoT, scalability, cost, and connectivity are the important parameters to be considered. There are many IoT cloud platforms such as Salesforce IoT Cloud, OpenIoT, Thingworx 8 IoT Platform, Microsoft Azure IoT Suite, among many others. Two open-source platforms, Kaa IoT and ThingSpeak, and two more popular platforms such as Google and Oracle Cloud are explored below.

11.4.1 Kaa IoT Platform

Kaa is an open-source, multipurpose middleware platform for end-to-end IoT application development as displayed in Fig. 11.5. Being open source, it reduces the cost, market time, and risk. For different use cases of IoT, Kaa IoT offers a variety of IoT tools.

Fig. 11.5 Kaa IoT Open-Source Cloud Computing Platform

It has many unique features such as:

- Open source
- Handles millions of devices
- Reduced development time
- Ease of implementation
- Reduced marketing time

Advantages:

- Data security
- Easy to use
- Third-party integration

Limitation Application based deployment is not possible.

11.4.2 ThingSpeak IoT Platform

ThingSpeak is an open-source platform for collecting and storing sensor data on the cloud. It provides you with apps to analyze and visualize your data in MATLAB. Arduino, Raspberry Pi, and Beaglebone boards can be used to send sensor data. In addition, a separate channel can be created to store data. Python code for client to upload data on the ThingSpeak in JSON format is shown in Box 11.1.

BOX 11.1 PYTHON CODE FOR CLIENT TO UPLOAD DATA ON THE THINGSPEAK CLOUD IN JSON FORMAT

```
import random
import urllib.request
import requests
import threading
import json

def client_data_send():
    d = {}
    for i in range(10):
        d[i] = {}
        for j in range(10):
            d[i][j] = random.randint(0,30)

    print("Data:-")
    print(d)

    val = json.dumps(d)
    URL = "https://api.thingspeak.com/update?api_key="
    KEY = "4044IAQ2S4DUEO7G"
    HEADER ="&field1={}&field2={}".format(val,val)
    NEW_URL = URL+KEY+HEADER
    #print(NEW_URL)
    data = urllib.request.urlopen(NEW_URL)
    print(data)
    print("Data sent")


def client_data_get():
    CHANNEL_ID = 763230
    READ_API_KEY = "4044IAQ2S4DUEO7G"
 conn=urllib.request.urlopen("http://api.thingspeak.com/channels/"+str(CHANNEL_
ID)+"/feeds/last.json?api_key="+str(READ_API_KEY))
    response = conn.read()
   print("http status code=%s" % (conn.getcode()))
    data = json.loads(response.decode())
    print(data["field1"])
    conn.close()

"""
client_data_send()
"""
client_data_get()
```

Fig. 11.6 Display of Downloaded and Saved Data from the ThingSpeak Cloud Server in JSON Format

Features of ThingSpeak

- App integration
- Collection data from private channels
- Feature supporting scheduling of event
- Visualization and analytic functionality of MATLAB

Advantages:

- Easy visualization
- Free channel hosting
- Python, Node.js, and Ruby support

11.4.3 Google Cloud

Google's cloud platform is one of the most popular cloud platforms available. Google has an end-to-end platform for IoT solutions. It facilitates easy connection, storage, and management of IoT data. Their main focus is on making smart things in an easy and fast way. Per minute use pricing is offered by Google Cloud and it is cheaper than other platforms.
Google Cloud IoT platform's main features:

- Efficient and scalable
- Very large storage capacity
- Server maintenance cost is less
- IoT data with full protection and with frequent response
- Analyses big data

Advantages:

- Lesser access time than many other platforms
- Fastest input/output
- Provides integration with other Google services

Limitations:

- Limited programming language choices
- Many components are using Google technologies

Google Cloud IoT provides many tools for edge analytics application to process, store, connect, and analyze data. This platform provides scalable cloud services managed by integrated software stack for edge computing with machine learning capabilities.

Businesses can gain by harnessing the diverse IoT cloud devices made available via Google cloud IoT. Results can be visualized with rich reports and dashboards in Google Data Studio. Various machine learning modules are integrated with Google cloud to run advanced analytics on the IoT data and gain meaningful insights. Users also take advantage of Tensor Processing Units (TPUs) to significantly increasing the computational efficiency of AI algorithms when processing big IoT data. The platform also supports many embedded operating systems.

11.4.4 Oracle Cloud

Oracle IoT platform offers real-time IoT data analysis, endpoint management, and high-speed messaging where users can get real-time notifications on their devices. Oracle IoT cloud service is a PaaS. Cloud-based offering helps in making critical business decisions.
Features offered to users:

- Real-time feedback
- Security and scalablity
- Analytics
- Reduced marketing time

Advantages:

- High-speed messaging
- Event storing capability
- Device visualization

11.5 REST-BASED WEB SERVICES FOR IOT

In resource-centric infrastructures such as IoT, REST(Representational State Transfer) style of web services are preferred due to lightweight, simple, portability, reliability and ability to directly transmit data via HTTP. The representation of the state of a IoT resource anemometer (measuring wind speed) is the value '20 mts/sec'. The interaction between client and server is stateless as opposed to Stateful web services such as those based on simple object access protocol (SOAP).

Advantages of using RESTful web services including the usage of URIs for the IoT resources (becomes web resource using URI), for example each device in a large IoT system can be given an URI, thus making it unique and universally addressable and also easy to locate because of the hyper media representation that is the use of web links that contain the metadata of the resource.

Further, the uniform interfaces enable the standard HTTP methods as applicable. Also since REST is based on Service Oriented Architecture (SOA) it inherits the basic features of loose coupling, reusability, scalability, integrity, etc. The Constrained Application Protocol (CoAP) is a specialized web transfer protocol that works with constrained nodes and constrained networks in the Internet. It is used to connect low powered devices to IoT. CoAP is based on the architectural principles of REST and runs over Universal Datagram (UDP) protocol.

THOUGHT EXERCISES

- Write a cloud client program in Embedded C for Arduino Uno and server program for Kaa IoT cloud platform to save the file in JSON format.
- Write the client program in Python language to send and store data collected in CSV format to ThingSpeak IoT cloud and write a server program on the ThingSpeak cloud server.

SUMMARY

For the real-time processing and with new emerging technologies, the need of connecting smart devices with cloud through Internet is gaining more and more importance. To work with vast number of IoT enabled gadgets there is a need for high computing facility, coupled with efficient storage infrastructure and provision of supporting software. An IoT cloud platform offers these facilities and with the added flexibility of subscription plans to suit various application domain specific requirements.

Hence, integration of cloud and IoT can bring new business opportunities and innovative smart applications. For efficient management of smart IoT applications, cloud computing provides a very good solution through its elasticity and facility to access shared resources and common infrastructure.

KEY TERMS

Cloud IoT service models, IoT Cloud platforms, ThingSpeak, SaaS, SEaaS, SenaaS, Google Cloud, Kaa IoT, Restful web API

FURTHER READING

1 https://www.edureka.co/blog/spark-tutorial/
2 GoGrid Storage Services, http://www.gogrid.com
3 iCloud, http://www.icloud
4 Amazon Web Services (AWS), http://aws.amazon.com/
5 The Pachube Feed Cloud Service, http://www.pachube.com
6 Internet of Things—ThingSpeak service, http://www.thingspeak.com

REVIEW QUESTIONS

1. List steps to set up a Hadoop cluster.
2. Compare Spark and Flink with respect to their merits and demerits.
3. Explain the benefits of integration of cloud services with IoT.

Edge Analytics: Near Real-Time Sensor Stream Processing

CHAPTER 12

"Our minds work in real time, which begins at the Big Bang and will end, if there is a Big Crunch - which seems unlikely, now, from the latest data showing accelerating expansion. Consciousness would come to an end at a singularity."

-Stephen Hawking

OBJECTIVES

- explain the concept of stream and its characteristics and relate it in the context of IoT
- outline the fundamentals of stream processing
- explain edge analytics
- explain the concepts of Complex Event Processing (CEP)
- classify various CEP operators
- apply open source stream processing tool, (Apache Flink)
- build an IoT application that performs edge analytics for real-time decision-making

OUTCOMES

- compare and contrast real-time processing and batch processing
- summarize the attributes of real-time processing
- make use of CEP in various scenarios to enable real time warnings and alerts
- experiment with Apache Flink to develop CEP applications

REVISION

To increase your understanding of the material in this chapter, you can spend a few minutes reviewing key concepts in chapters 10 and 11

- concepts related to Big data analytics need to be reviewed
- the earlier chapter on IoT platforms described various approaches for working with data coming from IoT devices
- the chapter on IoT Protocols explained the various network and communication protocols for low powered devices. The data latency issues were also discussed.

12.1 INTRODUCTION

A variety of Internet of Things (IoT) sensor devices are enabling to gather high-frequency continuous streams of data resulting in unprecedented amounts of data. The quantity of data generated depends on the temporal resolution of the sensor and the purpose of measurement pertaining to a particular application. A weather station comprising of sensors such as temperature, relative humidity, wind speed/direction, and precipitation could generate a few kilobytes of data, if collected at an hourly rate (refer to Table 12.1). However, sensors measuring traffic for estimating traffic flow characteristics may collect gigabytes of data in few minutes. Similarly, sensor networks in industrial plants could generate multiple gigabytes of data in a second. Many other examples of continuous streams of data include the following:

- Data generated by disaster monitoring network stations. For example, water level monitoring stations for floods, seismographs for earthquakes, buoys network for Tsunamis forecast, Forest fire monitoring networks, landslides detection, etc.
- Network security monitoring and management systems.
- Healthcare systems that provide continuous status of the patients in Intensive Care Units (ICU). Remote health assistive systems that continuously monitor elderly patients.
- Home security systems that continuously monitor inside the home to detect smoke/fire and outside for detecting intruders.
- Fraud detection using behavioural patterns in financial transactions through real-time tracking and analytics.
- Companies that provide e-commerce applications track the customers' behaviour on the website, and accordingly place relevant products and advertisements at strategic locations on the webpage.
- Real-Time analysis and prediction of stock market.
- Industrial IoT applications where there is a need to react to a process based on its status- and trigger-related processes.

The aforementioned applications require real-time or online processing (the gap between the data generation and processing is minimal), also known as low latency and high throughput requirement. In reality, however, this is very challenging to achieve. An end-to-end stream processing system should have the capabilities to ingest big data sets in an optimal way. In addition, transformation, processing, querying, and analysis are other important attributes of such a system.

Table 12.1 Data from an IoT Device Measuring Various Weather Parameters

Time Stamp	Temperature (°C)	Humidity (%)	Precipitation (mm)	Wind Speed (km/h)
9:00	26.4	53	10.2	17.7
9:02	26.3	52	10.4	17.6
Similar to above, thousands of records arriving at tremendous velocity				
10:15	26	50	10.9	17.5

12.1.1 Stream Processing Workflow

IoT data ingestion The incoming data from IoT devices arrive at the data ingestion layer. The capabilities of this layer depend upon how the data is produced. For example, data coming from a traffic flow monitoring sensor may come at a high rate, and in addition to gathering this data, it is also required to provide specialized services (by a middleware application) that enables to warn the users of an imminent traffic jam. Since the data produced is at a very high rate, and the decisions also have to be taken in real time based on the incoming data, there is a requirement for new ways for ingestion of the time-series data. Hence, this layer is devised to acquire and preprocess the data as it arrives and pass it on for further analytics. The data that is once passed on will no longer be available, that is, there is no persistence in the data ingestion layer.

Preprocessing and transformation The incoming data from the IoT devices are in a variety of syntactical and structural models, and also may have noise (due to disturbing events close to the sensors) associated with them. It is necessary to preprocess these data streams to make them amenable for consumption by other layers of IoT. For example, the data may need to be converted to a format that is acceptable to a particular application in the processing layer. Recently, messaging systems such as Apache Flume (Apache Flume, 2018) and Apache Kafka (Apache Kafka, 2018) are able to convert data in a form that is easily consumed by the various stream processing platforms. In addition, the data may need to be normalized, to remove variability in the ranges of values coming from different sensors. Further, sometimes it is necessary to add additional information to the data stream to make it more context specific such as adding the geographic coordinates (i.e., Latitude and Longitude) to give the data value a location (see Fig. 12.1).

Stream processing This layer consumes the streaming data either directly or buffered in between by the ingestion and preprocessing layers, and sends the output or intermediate results to the storage layer. The heart of a stream processing system is this layer, where a variety of processing approaches can be applied on the data stream in real time. It facilitates the capture of specific predetermined events or patterns of events coming from diverse IoT streaming data sources, which are actionable immediately.

Storage At this layer, persistent storage of the data occurs. The data that is stored could be either selected time windows of data from the intermediate buffer for long-term archival purpose or the captured events that will be analysed or used again for a different purpose such as understanding the

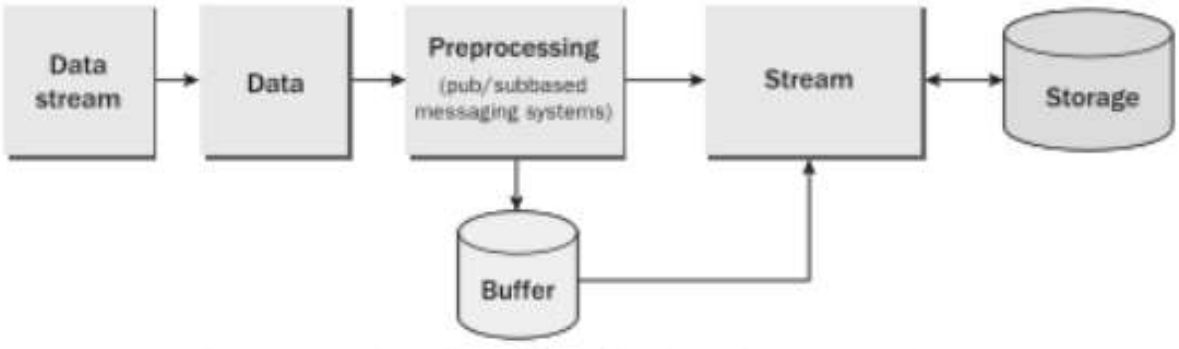


Fig. 12.1 A Typical Workflow of a Stream Processing System

limitations, faults, or combining them with other types of data. The archived data is generally used in a mini batch-processing mode for applying various techniques such as Machine Learning (ML)-based approaches for clustering and classification of the events. This topic is covered in Chapter 10 on Big IoT Data Science.

12.2 WHAT IS STREAMING DATA?

Data that resides persistently somewhere is no longer interesting than data that is in perpetual motion. The value of a data can decrease over time, especially those kinds of data that is used for real-time understanding of a phenomenon. For example, disaster monitoring requires continuously updated information to provide rapid response services, which can mitigate the loss of life and property. Some of the data that is used in such scenarios is related to weather conditions provided by a variety of meteorological sensors. The usefulness of a particular weather prediction for the next couple of hours expires as soon as that period has passed. Therefore, the information is useful only in that time window. Several such examples exist in other domains such as Traffic monitoring, stock markets analysis, and industrial IoT.

A *stream* can be defined as: 'a real-time, continuous, ordered (implicitly by arrival time or explicitly by time stamp) sequence of items. It is impossible to control the order in which items arrive, nor is feasible to locally store a stream in its entirety (Lukasz and Özsu, 2003).'

Typical characteristics of a data stream are as follows:

Unbounded data The data is arriving continuously and unbounded, that is, possibly an infinite stream.

Unordered data The data is coming in haphazard fashion without any particular order. This could be due to data buffering at the device level/gateway or coming from multiple devices and each device having its own frequency of sending the data. This results in a stream whose time stamps are not in a sequence, resulting in disparities between data acquisition time and data processing time.

Unsaved data Data is usually discarded after being processed for certain events. The extracted events may be stored for future use.

Data deviation The data distribution can deviate from the original after some time due to drift in the phenomenon being measured (e.g., rapidly changing weather), device issues and new requirements for acquisition resulting in a noisy data stream.

12.2.1 Bounded vs Unbounded Data

The general notion of streaming versus batch processing is confusing because there are systems that perform mini batch processing to mimic stream processing. Ideally, such systems are batch processors with an ability to perform very fine-grained batch operations that look like a stream process. Hence, to overcome such confusion, it is better to think in terms of bounded and unbounded data.

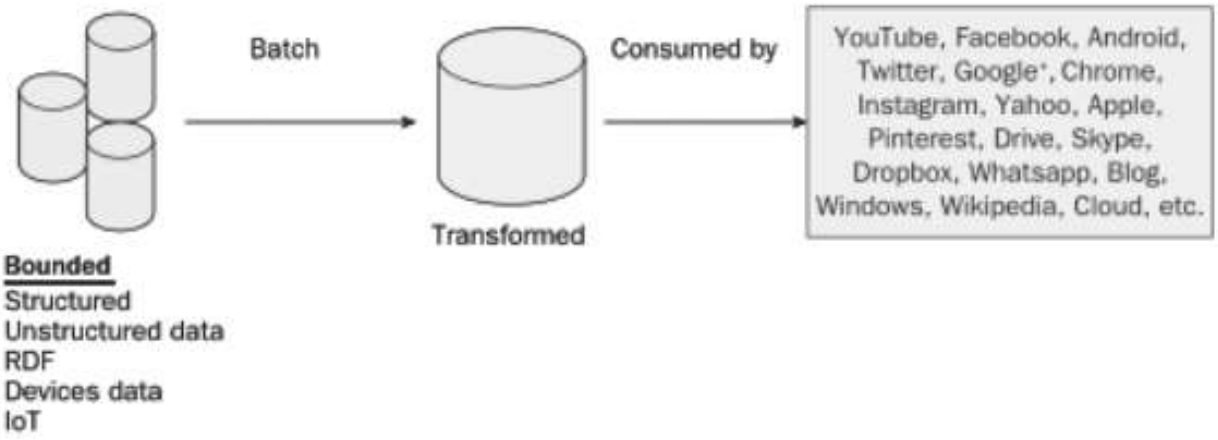


Fig. 12.2 Batch Processing on Bounded Data

12.2.1.1 Bounded Data

When data possesses distinctive starting and end points, that is, in terms of its time stamp, it is called as a bounded data set. The processing on bounded data set usually is in the batch mode (batch processing). In this approach (Fig. 12.2), a dataset in a repository (stored data), which could be in various formats (structured, unstructured, RDF triples, etc.) is processed in chunks (chunking is done based on some predefined criteria) and given to a batch-processing engine. The data transforms into a structured form at the end of the batch processing, which can be consumed by various applications.

The performance of a batch processing system can be measured in terms of *throughput* and *latency*. Theoretically, *throughput* is the amount of data that can be processed in a unit time over a known bandwidth; however, in reality this varies due to various hurdles in the network connection such as transmission and processing delays. *Latency* is the delay that occurred in processing over a network, which directly affects the *throughput*.

12.2.1.2 Unbounded Data

Unbounded data is data that is unending (infinite) and coming continuously in some ordered or unordered fashion (Fig. 12.3).

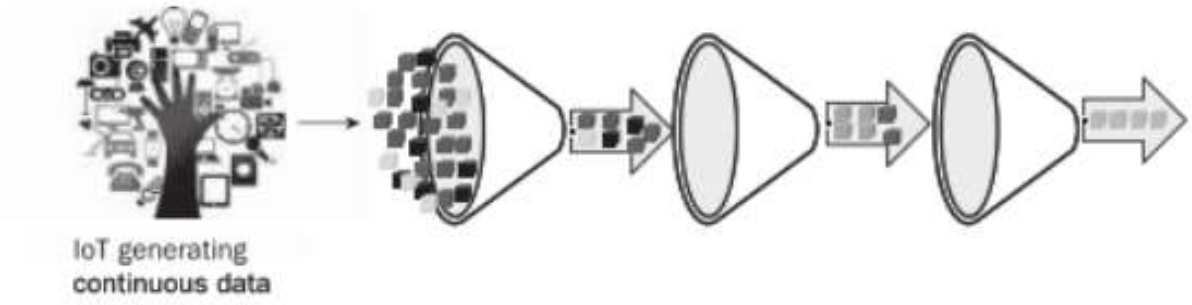


Fig. 12.3 Unbounded Data Streams from a Variety of IoT Sensors (High speed and volume streams are reduced by applying filters designed to capture specific events or patterns)

12.2.2 Time-bound Processing of Streaming Data

Streaming data processing capability is measured by time in terms of latency and throughput. It is useful to differentiate between two types of time measures, that is, event time and processing time.

12.2.2.1 Event Time

This is when the actual event has occurred. For example, if a sensor measured temperature at 15:30 Hours, the event time is the time of measurement of the phenomenon. It has a time stamp and a value associated with it.

12.2.2.2 Processing Time

This is time at which the event has arrived at the event stream-processing platform and is one of the events out a stream of events coming in sequentially.

It is quite possible that the time of processing (system time of the machine-based on the system clock) of the event and the time of the actual event happening are different. This could result in the disruption of the time-synchronized arrival of the incoming streams. Hence, it is important to keep a check on the order of the arrival of events.

This discrepancy of event time and processing time is crucial for certain applications. In the context of IoT, this is a very important consideration if there are time critical events that are being detected and acted upon.

For example, in a real time disaster application such as floods, the precipitation measurements at various geographically distributed measuring stations have an event timestamp associated with each precipitation measurement. But when some of these measurements are unable to reach the processing engine due to connectivity issues and are queued and sent later, it leads to latency issues. The processing engine meanwhile processes the other available measurements from other stations and they have a different processing time stamp than those measurements that are subsequently processed from the delayed stream of data. This can lead to a disruption in the right sequence of processing of the precipitation measurements, which are used in a model for predicting possible inundation area.

12.3 DATA STREAM MANAGEMENT SYSTEMS

Data Stream Management Systems (DSMS) are specialized systems that deal with continuously arriving data streams as against Database Management Systems (DBMS), which deal with stored data (Babcock, 2002). As shown in Fig. 12.4, the data streams are processed as they arrive, and optionally the incoming streams may be buffered in an intermediate storage (at the IoT gateway) and formatted in such as a way that is suitable for processing in the stream processing module. DSMS normally store the important events that were derived from the data streams for further analysis and also for combining them with other sources of information.

The data is stored first and then analysed in the normal data processing approaches. The data stream is immediately processed as soon as it arrives and the intermediate results are stored in a buffer for further processing.

The main differences between a Database Management System (DBMS) and Data Stream Management System (DSMS) at various levels in data processing pipeline are shown in Table 12.2.

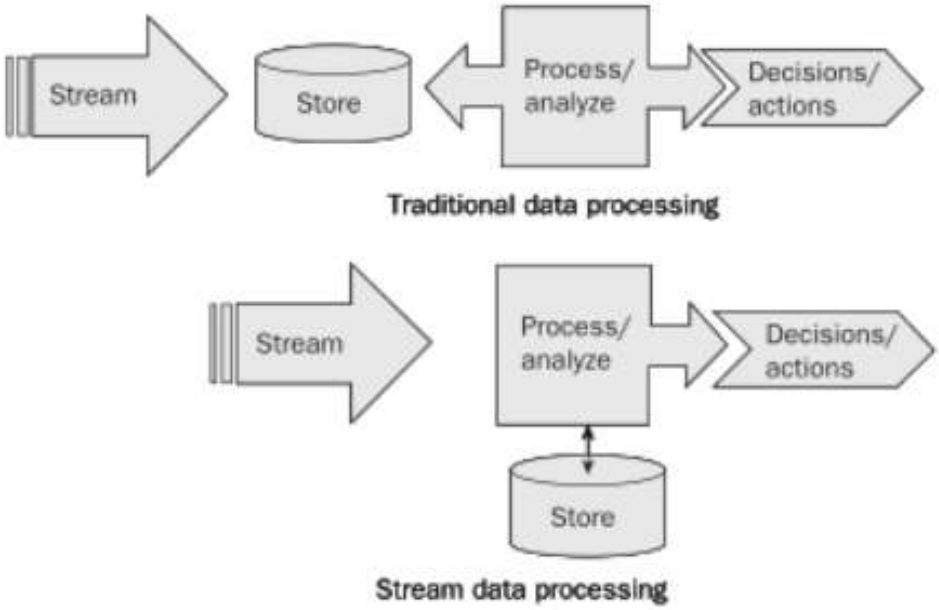


Fig. 12.4 Traditional Data Processing vs Stream Data Processing

Table 12.2 Differences between Database Management System (DBMS) and Data Stream Management Systems (DSMS)

Database Management Systems (DBMS)	Data Stream Management Systems (DSMS)
Data level	
Static/stored data, time of capture or storage of data is unimportant	Continuously changing data, time of event capture and time of processing are important
Access to the data can be either sequential or random	Data access is sequential and is usually based on the timestamp of the incoming events
Data updation is relatively slow, usually in relatively small increments	High frequency and volume of data updation
Data is consistent and has high precision	Data is noisy due to imperfections in the data capturing process
Processing level	
Processing on data available currently in the database	Processing on current as well as historical data, which could be data that arrived few seconds, hours, or days before.
Asynchronous data processing (e.g., batch processing), no need to process or deliver results in real time	Synchronous, real-time processing of data. Requirement for delivery of results rapidly
Processing on relatively large portions of data	Incremental processing approach
Data value remains consistent over time	Data can become stale and no longer be usable after a set period of time

(Contd)

Table 12.2 *(Contd)*

Database Management Systems (DBMS)	Data Stream Management Systems (DSMS)
Query level	
Queries can be dynamic (i.e., querying whenever the situation demands) and complex	Queries are pre-determined and run on time varying data. Complexity is in the capturing and processing of event patterns.
Queries are precise	Queries can be approximate and can return approximate results
Multiple rounds of data access may be done to answer a query	One time access to data, after which the data is no longer available

12.3.1 Background of DSMS Development

Various DSMSs were developed during the last decade. Here a few selected DSMS are briefly reviewed to understand the progress in the development of more sophisticated systems that are available in today's market.

Gigascope It is based on an SQL-like language, called GSQL. It was developed specifically for network domain applications such as intrusion detection and traffic monitoring (Cranor et al., 2003).

TelegraphCQ It is a system that can enable continuous querying over big streams that change over time. It is based on SQL-based language called StreaQuel. It can work on clusters (Chandrasekaran et al., 2003).

Aurora It is based on a language called SQuAl, which can be used to visually define various rules with the help of various GUI widgets (Abadi et al., 2003).

Borealis It is the successor to Aurora and a second-generation stream processing engine. It addresses two important areas of Stream processing such as dynamic revision of query results, which enables to process the data that was corrected and re-injected into the stream. If such ability is missing then the results will be imprecise. The second functionality of the system is that it enables to change the query attributes at runtime (Abadi et al., 2005).

TESLA/T-Rex T-Rex system performs events processing incrementally whenever events incoming event streams enter the system. Detection automata based algorithm is developed to perform such incremental processing (Cugola & Margara, 2010a).

Further, several other stream processing engines such as Stream Mill (Bai et al., 2006), STREAM (Arasu et al., 2003), StreaMon (Babu & Widom, 2004), Tibco-StreamBase (Tibco-Streambase, 2018), OracleCEP (OrcaleCEP, 2018), Esper (Esper, 2018), IBM WebSphere (IBM WebSphere, 2018) are available. However, some of these systems have challenges in satisfying the current big data requirements of volume and velocity. Scalability implies that, based on the requirement of the processing task the number of worker nodes (or processing units) can be increased without any damage to the system.

BOX 12.1: EIGHT REQUIREMENTS OF A STREAM PROCESSING SYSTEM

Strohbahach, M., et al., 2005, provided eight requirements that an ideal stream processing system should adhere to:

1. Keep the data moving
2. Query using SQL on streams (StreamSQL)
3. Handle stream imperfections (delayed, missing, and out-of-order data)
4. Generate predictable outcomes
5. Integrate stored and streaming data
6. Guarantee data safety and availability
7. Partition and scale applications automatically
8. Process and respond instantaneously

Eight requirements of stream processing systems are shown in Box 12.1. Recently, several streaming platforms have been developed that are highly scalable and have the attributes of low latency and high throughput. These platforms have the ability to run on High Performance Computing (HPC) systems so that real-time processing can be achieved.

12.3.2 Stream Processing Platforms

Recently, several stream processing platforms were developed to support large-scale applications. Among others, these systems differ in their delivery semantics as shown in Box 12.2.

In this section, a brief review of the most popular platforms is presented.

Apache Storm It is a stream processing platform that has proven to process a million records in the fraction of a second. The architecture is based on the concept of Directed Acyclic Graph (DAG), where the vertices represent either a data source or a computational unit, this form of representation is referred to as a topology in storm. Storm carries out stream processing using micro-batches of the incoming events, so a latency of a few milliseconds can happen. It supports at least once and almost once semantics (Toshniwal et al., 2014).

BOX 12.2: DELIVERY SEMANTICS OF IOT EDGE ANALYTICS PLATFORMS

At-most-once: The events are dropped if they are not processed correctly; here there is a possibility for data loss.

At-least-once: In this case, each input event is monitored to see if the processing is successful within a time window. This is achieved by keeping a record in-memory. In case the event was not processed successfully, it is resent again for processing. Therefore, here the data loss issue is resolved; however, there is a possibility of having duplicates.

Exactly once: Events are processed at-least once and the duplicates are detected and discarded.

MillWheel Developed at Google, MillWheel is a streaming system that uses the concept of DAG processing. The nodes of the DAG are computational units as well as persistent storage, while the vertices carry the messages. In MillWheel, it is possible to enforce a time-based watermark that makes sure that all the events data before a certain time (called the *low watermark*) is processed by the system, irrespective of the order of their arrival. This ensures that all data within a particular range of time (time stamps) are guaranteed to be processed. It supports exactly once semantics (Akidau et al., 2013).

Apache Samza Samza uses Kafka for handing the streams. In Kafka a stream of data is treated as a topic, which is a stream of related events and can be subscribed by a user. Samza can store a state as a local key-value store and supports at-least-once delivery semantics (Apache Samza, 2018).

Apache Spark Spark is a hybrid stream processing system, which can handle both steam and batch processing. Spark's streaming module is based on stream buffering at very small intervals (milliseconds) and uses them in the batch mode for processing. Due to the buffering mechanism used in Spark, there is a need to flush it in between, which can lead to latency issues. It supports exactly once delivery semantics (Apache Spark, 2018).

Apache Flink It is a hybrid stream processing approach based on the Kappa architecture, where streams are first class citizens, and are used for all kinds of processing (both batch and streams). The stream-first philosophy of Flink results in high throughput and low latency. In Flink, the *event time*, that is, when an event has actually occurred can be used to guarantee ordered streams. It guarantees exactly once record delivery semantics (Apache Flink, 2018).

The edge analytics (filters, rules, triggers, warning, etc.) are performed on live stream of data as it arrives. Complex event processing (CEP) techniques are applied to these types of event streams. The cold analytics are performed on data streams that are in a buffer or in a persistent storage; stream data mining approaches are relevant in this mode.

12.4 EDGE ANALYTICS

Edge analytics are performed at the location (or close to) of the devices from which the data is collected. It is an approach, where the incoming data is processed and analysed as it arrives, in contrast to approaches, which rely on the analysing data that is stored in a repository. Edge analytics assume a prime importance in the context of IoT because it allows capturing those events that are interesting and important to a particular task and which has a time constraint for delivering the results. The recent explosion in the volume, velocity, and variety (3 vs of Big data) demands for approaches that are able to make sense of the data in real time, not only for real-time response activities, but also for storing only those events that are interesting and discarding the rest. Some examples that tremendously benefit from edge analytics are mentioned in Section 12.1. The ability to perform real-time analytics is possible due to the techniques, methods, and tools that are developed specifically for stream processing. In general, there are two major approaches for performing analytics on IoT data streams. As described in Box 12.3, the cold analytics approach is applicable when the data is stored in some buffer or in persistent storage. Machine Learning (ML) approaches are relevant in this module. The next chapter (Chapter 10) on IoT

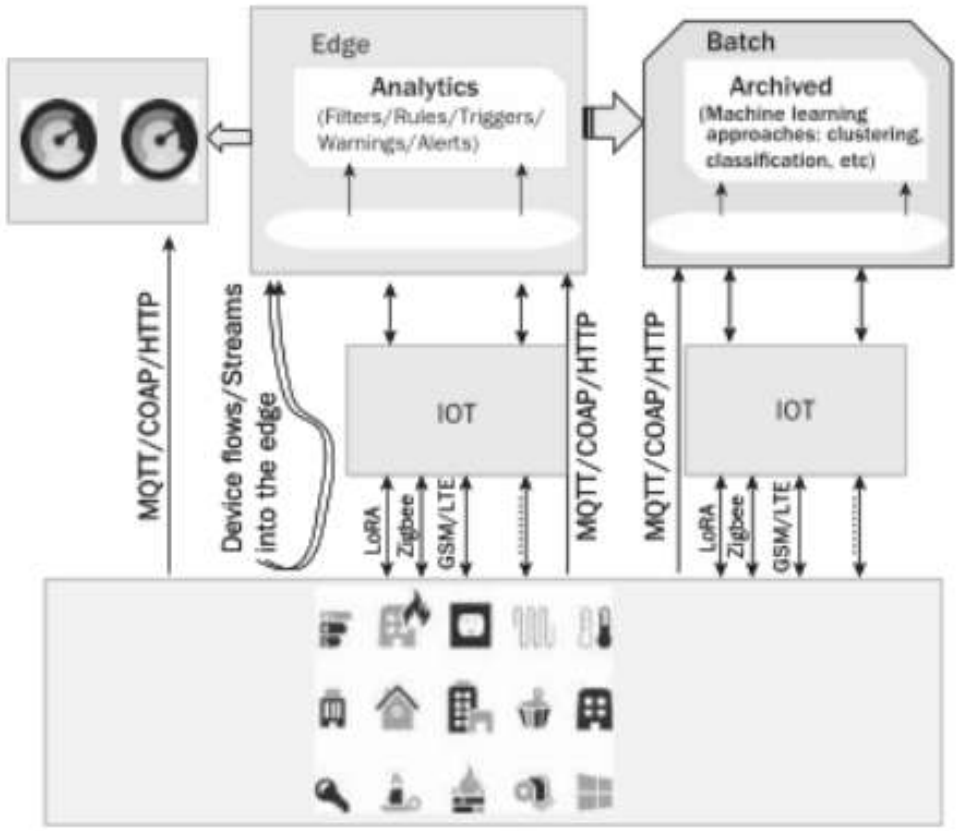


Fig. 12.5 Framework for Analytics in IoT

Data Science covers topics relevant to ML-based techniques. The analytics covered in this chapter are geared towards techniques that are applied on the fly in real time (see Fig. 12.5); hence, they are called hot analytics, which can facilitate rapid analysis to gain situational awareness.

Complex event processing is one of the major techniques to harness multiple streams of data coming from IoT sensors, which is described in Section 12.4.2.

BOX 12.3: TYPES OF IOT ANALYTICS

Cold analytics approach: In the cold analytics approach, the incoming data stream is archived first and then processed usually in a batch mode. The amount and duration of data that is processed is predetermined. This is discussed in the next chapter on data stream mining, which covers various IoT stream data mining methods such as stream learners for classification, regression, clustering, and frequent pattern mining.

Hot analytics approach: A hot analytics approach (see Fig. 12.6) which requires the stream to be processed as and when they arrive and characterized based on a predefined criteria. Stream processing and CEP approaches belong to the hot analytics approach, where the incoming data streams are continuously matched against a pattern(s) so as to capture the essence of the data stream and gain useful insights. The events that are detected are called complex event patterns.

12.4.1 Event Processing

Event An event is a record of the change of state of an entity at a given point of time. It is usually something that is of interest. A reading from an IoT sensor is an example of an event.

Atomic event An atomic event is an event that happens at a specific point in time having a time stamp and carries unprocessed information. It may have a state and value corresponding to phenomenon that is measured. Raw readings from a sensor are an example of atomic event streams and are processed in a stream processing engine; its goal is to dynamically process the data streams by some mathematical transformations.

Complex event Also called as a composite event which is based on the composition of atomic events in some predetermined form (Figs 12.6 and 12.7). It could be based on various operators that are applied on a single or multiple events to form more complex set of events. For example, the change in the air quality from good to worse, which is based on streams of events coming from different air quality measuring parameters. CEP goes beyond simple aggregation of events; it is designed to obtain insights into the relationships between streams.

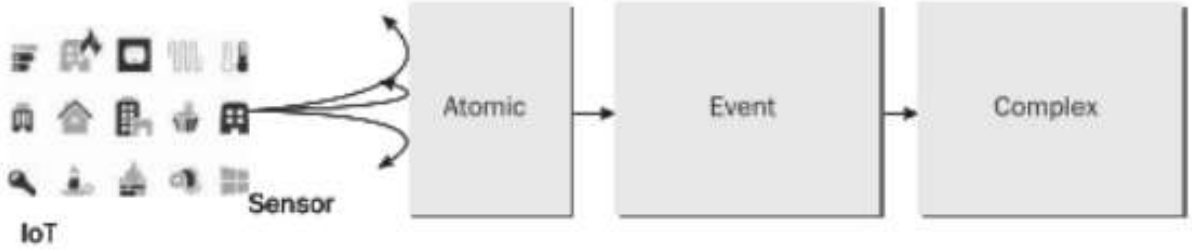


Fig. 12.6 Relation between Atomic Events, Event Streams, and Complex Events

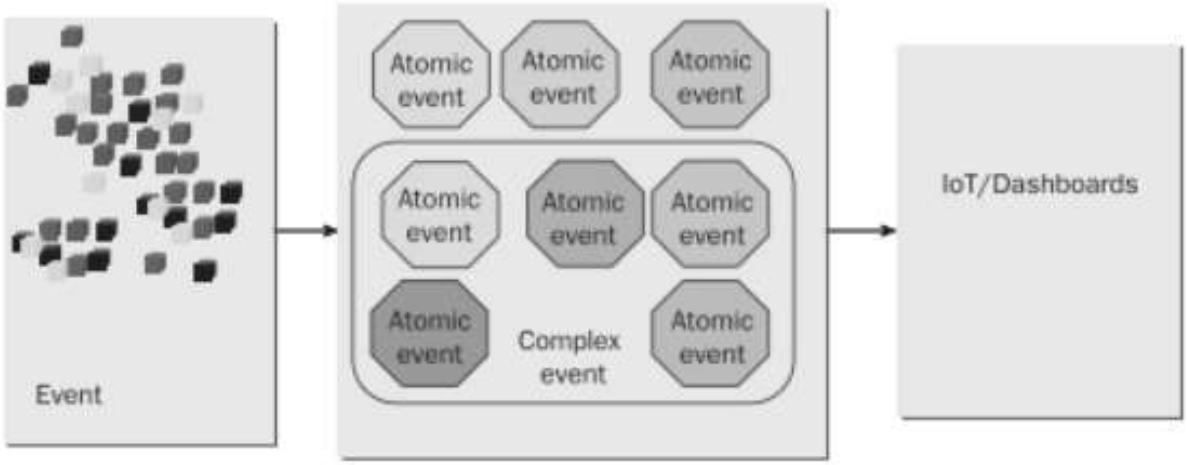


Fig. 12.7 Illustration of Atomic and Complex Events

Event pattern An event pattern captures an event's properties and relationships (if applicable) between different (possible from multiple sources) events of a certain type. An event instance is a realization of an event pattern. Examples of such event patterns are: temporal pattern of an event such as traffic jam, weather front, and stock market. Various complex event patterns are discussed in Section 12.4.2.1.

Event stream processing (ESP) The goal of ESP is to process big event streams in a continuous fashion. In ESP, the order of arrival of the events is not important. The earlier version of DSMS systems are based on the ESP concept. ESP is a subset of CEP.

12.4.2 Complex Event Processing

Complex event processing enables to understand, detect, correlate, and process the relationship between events. It is the principal technology solution for processing real-time data streams (Suhothayan et al., 2011). CEP was first introduced by Luckham (2002). Several authors have defined CEP in various terms such as:

Luckham (2007) 'CEP consists of principles for processing clouds of events to extract information, together with technologies to implement those principles.'

Eckert and Bry (2009) 'A technology which encompasses methods, techniques, and tools for processing events while they occur, i.e., in a continuous and timely fashion.'

BOX 12.4: DIFFERENCES BETWEEN DSMS AND CEP

DSMS were designed specifically for querying continuously varying time series of data and produce a result to the query at any point of time. However, the order of the arrival of the events has not been dealt with in these systems. Newer variants of DSMS, address this issue. In general, they are not capable of working with multiple data streams.

CEP is focused on the processing of multiple related events coming from a variety of sensors belonging to a particular domain in order to detect interesting time varying events and send continuous alerts (notifications) related to the complex patterns. The patterns are similar to those described in Section 12.4.1.2, which are usually a composition of the simple patterns to form a composite or complex pattern to deal with a more complicated scenario. The ordering of the events is important. It is possible to capture a set of complex patterns and the results may be converted to a stream of events, which can then be combined with another set of complex events to answer an advanced query. For example, warnings from a stream of events can be converted to set of event streams and another pattern can be created to capture those critical warnings (e.g., consecutive warnings of the same type, within a particular time window) that should be treated as alerts and notified to the user.

Many current systems are a hybrid of the aforementioned systems.

CEP has taken the DSMS systems to the next level, the key differences between CEP and DSMS are shown in Box 12.4.

A CEP processor captures the state change of the events, and uses such information to carry out analytics, which can predict future states. Thus, events and their state are what triggers a CEP system. Events form components which are composed together to form higher-level complex events (Fig. 12.8). The way the components are combined is based on which CEP operators (see Section 12.4.2.1) are used in the sequence to derive a query that is pertinent to a particular situation.

CEP is now being quickly adopted in the IoT systems as they are designed to respond and provide solutions in real-time to enable predicting events, obtain awareness about critical situations, conditions, and imminent threats. Some of the main reasons for developing CEP systems for IoT are mentioned in Box 12.5.

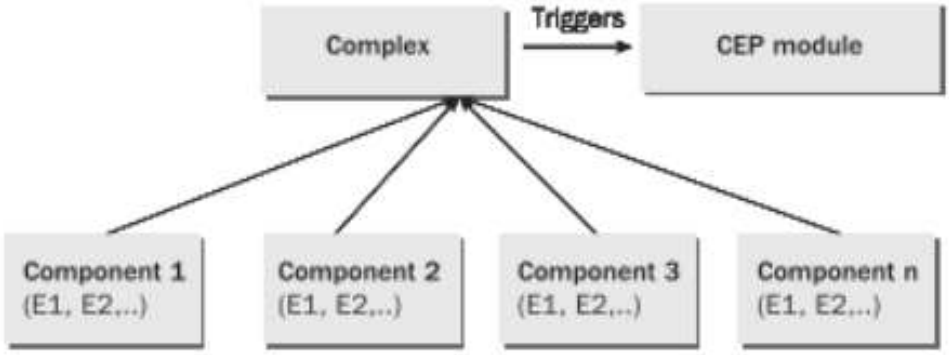


Fig. 12.8 Components Consist of Events and Encapsulated into a Higher-level Complex Event

BOX 12.5: MAIN REASONS FOR ADOPTION OF COMPLEX EVENT PROCESSING FOR IOT APPLICATIONS

Complex event processing technologies are becoming quite popular for IoT use cases, the main reasons are as:

1. Ability to provide situational awareness for events that are happening in a particular domain: This is a crucial requirement in IoT applications as they are designed to provide feedback or response rapidly, so that the user is aware of the current situation and status of a particular scenario.
2. Ability to capture and process high throughput big data stream coming from IoT sensors.
3. Ability to handle time varying IoT events in various forms (ordered, unordered, drifting, etc.).
4. Ability to integrate and gain understanding of complex interactions between event streams coming from a variety of sources, through various forms of filtering on the time series, and developing a variety of complex patterns.
5. Ability to obtain approximate answers to queries when time is of essence.

12.4.2.1 CEP Operators

The complex event processing engine is capable of processing multiple streams of events and understand the patterns and relationships between streams of a certain type (also called as keyed streams, e.g., all streams that originated from a station based on its stationID), but also between event streams that are independent. To enable an intuitive way of expressing the behaviour of the events, most of the currently available CEP engines allow representing events in a declarative language similar to SQL. Many stream query languages were developed, the structure of these languages are similar to SQL but they are extended with special operators for continuous stream processing (Lukasz and Özsu, 2003; Gyllstrom et al., 2007; Wu et al., 2006; Sqlstream, 2018; Arasu et al., 2006).

Below are the most often used fundamental operators (Cugola and Margara, 2010). In general, the CEP patterns are based on a combination of the below mentioned operators. Usually the pattern is in a nested form containing a sequence of the basic operators.

12.4.2.2 Single-item Operators

Selection The selection operators are similar to filtering, in which a specific condition or criteria is tested. The conditions are based on the values of the event (e.g., a value measured by a sensor). The most common criteria is a threshold. As an example, consider the following pattern, which selects the events from a ground level ozone sensor, which are between 90 and 130 parts per billion (PPB).

Select OzoneSensor (Ozone>= 90 and Ozone<=130)
From OzoneDataStream

Elaboration operators These operators are used for transformation of events. Projection and renaming are elaboration operators.

Projection Extracts a part of the attributes and processes the events related to it. For example, the location and height at which a measurement is made is extracted. Only those events that match these criteria are further examined.

Select OzoneSensor (Height, Location (Lat, Long))
From OzoneDataStream

Renaming operators These operators are used to change the name of the property of a certain event.

Select OzoneSensor ("Measurement")
From OzoneDataStream

Rename ("Ozone ground level measurement")

12.4.2.3 Logical Operators

Conjunction A conjunction of events EV1, EV2, ..., EVn implies that all of the events EV1, EV2, ..., EVn have occurred. It is applicable to two or more events occurring together. For example, it can be used to capture events related to air pollution where both Particulate Matter (PM) events and ozone at ground level events are notified within a time window of 30 min.

OzoneSensor(Ozone>130) AND PMSensor(Particulate Matter > 75)
From AirqualityStationData
(Within 30m)

Disjunction In a predefined window, the occurrence of either of the selected events.

OzoneSensor(Ozone>130) OR PMSensor(Particulate Matter > 75)
From AirqualityStationData
(Within 30m)

Repetition This pattern is useful to detect repeat occurrence of an event. The repetition can be predefined. For example, it would be useful to know how many times the air quality has deteriorated to poor quality in the last 8 h. In addition, the same event can repeat between two time periods (m, n)

Select PMSensor (Particulate Matter > 85) as PM AND
Select OzoneSensor (Ozone > 125) as OZ
From AirQualtiyData
Where count (PM and OZ) > 3

Aggregation The aggregate values from a particular event are subjected to a condition such as min, max, average, and std. For example, the 8 h average of the ozone value is used in the calculation of air quality index.

Select average (ozonesensor.Ozone)
From AirqualityData
(Within 8 Hrs)

Negation operator A negation of event EV means that the event E has not occurred. For example, it is safe to drive if:

PMSensor(PM <33) and OzoneSensor(Ozone<60) and NOT Dusty().

Sequence operator The sequence of occurrence of events is captured by the sequence operator. The order of the events (e.g., the sequence in which a measurement is made over a time interval) is important.

Quantification It is a measure of the occurrence of a number of events (n) of a particular type.

12.4.2.4 Iteration

Iterations are conditional and the sequence of events must conform to the iteration condition.

Typical steps involved in building a pattern query is shown in Box 12.6. These steps can also be compared with those from the example given in Section 12.5.

12.4.2.5 Windowing

Windows are useful to make some boundaries on the unbounded data streams, so that only a portion of the stream of events can be processed correctly in a predefined time interval or a set of events. The various operators as discussed earlier are used with the windows.

BOX 12.6: TYPICAL STEPS IN BUILDING A COMPLEX PATTERN QUERY

1. **Begin:** Starting of a pattern definition:
   ```
   Pattern<TrafficEvent, ?> pattern = Pattern.<TrafficEvent>begin("start"). Here we have begun a pattern definition for capturing events from traffic data
   ```
2. **Next:** New pattern that matches the previous matching pattern
   ```
   Pattern<TrafficEvent, ?> next = start.next("next");
   ```
3. **Followed by:** Appends a new pattern state, but here other events can occur between two matching events: Pattern<TrafficEvent, ?> followedBy = start.followedBy ("next");
4. **Where:** Filter condition:
   ```
   start..where(new FilterFunction <TrafficEvent>() {
     @override
       public boolean filter(TrafficEvent value) throws Exception
         {
           return ... // some condition
         }
   });
   ```
5. **Or:** Add new filter condition with an existing one. Example is using two where conditions with an OR in between them.
6. **Within:** Time interval for matching the pattern: patternState.within(Time.minutes(5));

Note: The example complex event patterns show above are based on the representation using Apache Flink and assuming event streams from a traffic IoT sensor.

Tumbling window In these windows, the upper and lower bounds are predetermined and they remain constant. These windows do not overlap and are adjacent to each other. This creates non-overlapping adjacent windows in a stream. The tumbling windows are bounded by either time (e.g., all traffic related events that are in a fixed interval of time, i.e., 15:00 to 15:30 Hours) or by the number of events that will be aggregated together before doing some analysis on it (e.g., five traffic events (e.g., traffic jam) on particular road, aggregated, and then analysed (refer to Fig. 12.9).

Sliding window Most commonly used windows, that are useful for setting an upper and lower bound and both of them move with the same frequency. As shown in Fig. 12.10, the windows can overlap similar to the tumbling windows but here the windows can overlap. We can use it in scenarios where we need to quantify or summarize the events in a particular time frame and at the same time send the results in another time frame. For example, if we want to know the traffic density every 10 min, it should contain that information for the last 30 min.

Windows can overlap and have a predefined upper and lower bound when can be different. Each stream can be identified by its key value.

Session window It is based on how long events are occurring in close proximity to each other (refer to Fig. 12.11). A session gap is predefined, which tells how long the activity should occur together, and

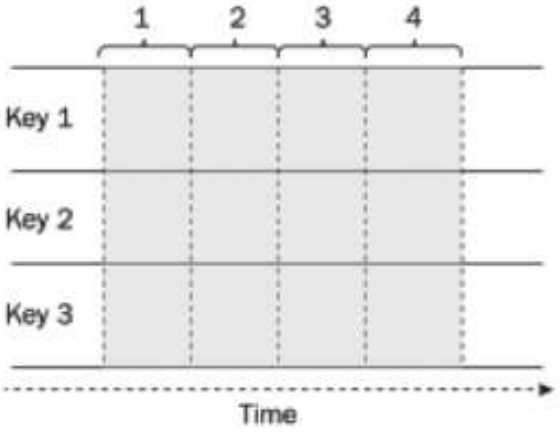


Fig. 12.9 Tumbling Windows are Non-overlapping and Their Size is Predetermined Based on Some Understanding of the Phenomenon Being Studied

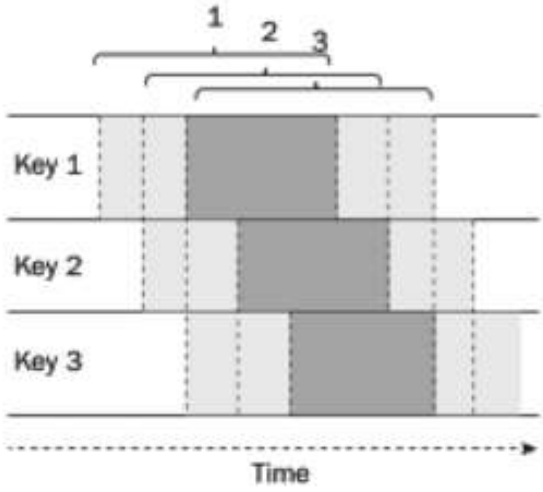


Fig. 12.10 Sliding Window Approach

how wide a gap of inactivity can be tolerated. This kind of processing is useful in situations where we are expecting activity in a particular order with an approximate time gap between them, but due to some other influence this session gap has become wider, which needs to be flagged.

For example, the peak traffic monitoring is based on sessions, i.e. morning session when the traffic is very high during office time, then again in the evening session when the traffic is at its peak from

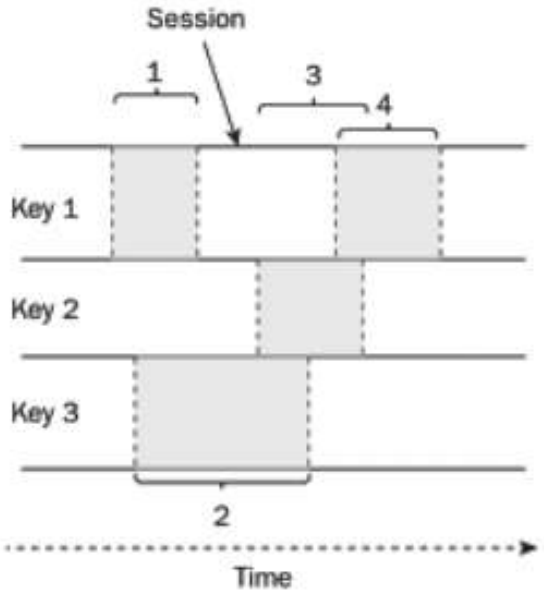


Fig. 12.11 Session Windows Capture the Activity of Events that are in Close Proximity and are Occurring in a Certain Time Period. If There is No Activity Happening for a Time Period, which is More Than What is Predefined, the Session May Be Closed.

returning office goers and in between there is an approximate time gap. Although this is a usual phenomenon, sometimes it could so happen that the people returning from office may come back at an unusual time than their regular time. This could be due to unusual traffic because of some procession or other road blockages. The session window is useful in such scenarios to flag the unusual gap between the peak traffic in the morning session and the peak traffic in the evening session.

Global window A global window can be applied when we need to process the events in their entirety. So all the events are enclosed in a global window and some analysis is done (e.g., statistical operators such as min/max and average) and then a trigger may be used to flag any anomaly.

For example, the traffic data for the whole day between 8 am to 10 pm can be processed in their entirety, so that all the events (e.g. accidents pattern, traffic congestion patterns, etc.) can be analysed.

12.5 EDGE ANALYTICS WALKTHROUGH

Imagine that you are on a road trip and read in the newspaper that yesterday the haze and smog levels in the city are alarmingly high. Which lead you to think, what should I do to avoid exposure to this pollution as I travel to work today? I need to stay away from:

- Areas and roads having pollution levels, which are classified as "Poor Air Quality"

Thankfully, the roads that you need to travel have a network of IoT sensors that are measuring the air pollution *parameters* such as:

- ground level Ozone (O3)
- particle pollution (PM10)
- carbon monoxide (CO)
- sulphur dioxide (SO_2)

Assuming the standard pollutant level for each of these pollutants are: Ozone = 100 ppb, NO_2 = 120 ppb, sulphur dioxide = 200 ppb, carbon monoxide = 9 ppm, PM10 = 50 $\mu g/m^3$

(Note: Values are based on the information obtained from this site: http://www.epa.vic.gov.au/your-environment/air/air-pollution/air-quality-index/calculating-a-station-air-quality-index)

$$\text{AQIndex} = \frac{\text{Observed value of the parameter}}{\text{Standard value of the parameter}} \times 100$$

Suppose there is monitoring station whose AQI is as below:

PM10 particles	Ozone
33	56

Then the AQI summary for that station at that time would be 56, which corresponds to the 'Good' category (refer to Table 12.3). (Note: For the sake of illustration only two parameters are used in this example.)

Table 12.3 Air Quality Category Based on the Index

Air Quality Category	Index Range
Very good (VG)	0–33
Good (G)	34–66
Fair (F)	67–99
Poor (P)	100–149
Very poor (VP)	150 or greater

Adapted from www.epa.vic.gov.au; www.epa.sa.gov.au; www.soe.environment.gov.au

Now let us develop an IoT application that is able to provide air quality-related information in real time.

12.5.1 Problem Background

This example program generates a stream of monitoring events, which are analysed using Flink's CEP library. More details about Flink is shown in Box 12.6.

12.5.1.1 Scenario

Consider that there are a number of IoT-enabled air quality monitoring stations along the road network that you are travelling. For each of these stations, among different quality parameters that are being measured, let us take two of them, that is, NO_2 and PM10. These are measured every 1 h, which means two events (NO_2 and PM10) are generated. This will be a continuous stream and based on it we can detect on which roads the air quality is becoming poor. Knowing such information in real time, you can plan to reroute the travel so as to avoid those places which are highly polluted.

12.5.1.2 Goal

The goal is to detect when the status of the air quality has changed. To accomplish this goal, we will first keep track of NO_2 and PM10 events (refer to Fig.12.12).

We create a CEP pattern which generates a NO_2 and PM10 warning whenever it sees two consecutive NO_2 and PM10 events in a given time interval whose values are higher than a given threshold value. A warning itself is not critical but if we see two warnings for the same station whose NO_2 or PM10 values are rising, we want to generate an alert. This is achieved by defining another CEP pattern, which analyses the stream of generated NO_2 and PM10 warnings.

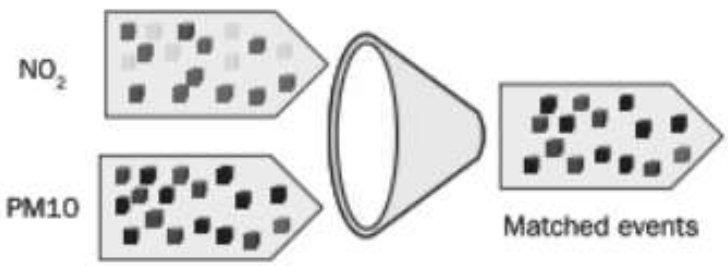


Fig. 12.12 Conceptual Map of Stream of Events that are Interesting or Useful

BOX 12.7: ARCHITECTURE OF STREAM PROCESSING USING APACHE FLINK

In the hands-on exercise in this chapter, we will use Apache Flink. It has hybrid architecture as it has both stream processing and batch processing. In Flink, streams are used for both stream processing as well as batch processing. As shown in the Fig. 12.13, it is useful for developing distributed, high-performing, always-available, and accurate data streaming applications. In this hands-on exercise, we will use the CEP library of Flink, which is capable of detecting event patterns in an unending stream of events and capture those events that are useful and interesting to a particular application that are useful and interesting to us.

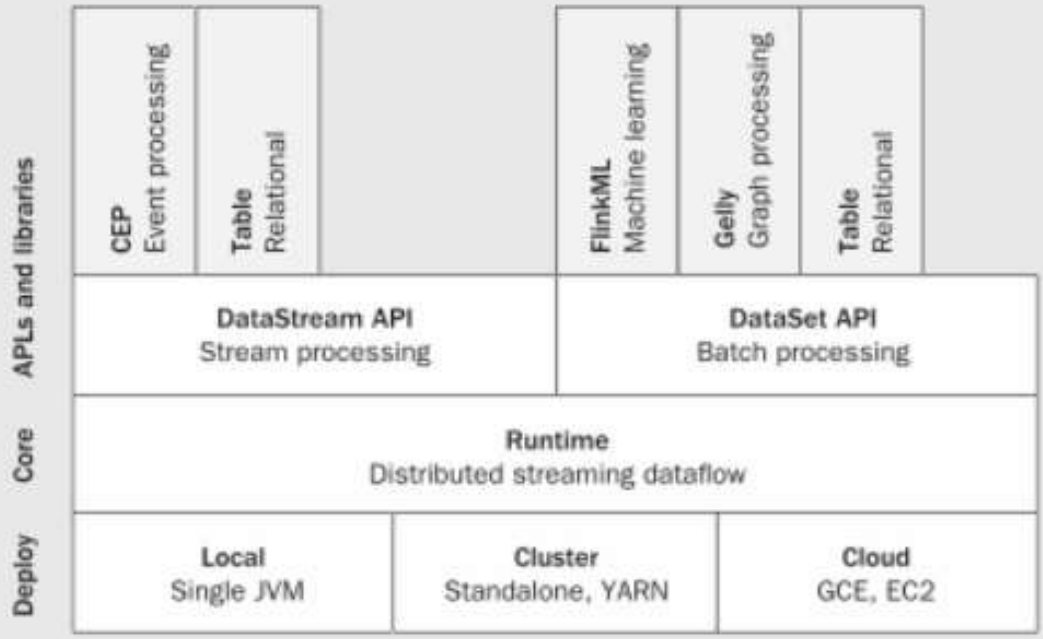


Fig. 12.13 Architecture of Flink

Adapted from www.ci.apache.org; www.oreilly.com; www.data-flair.training.com

For the purpose of this walkthrough, the input event stream consists of simulated NO_2 and PM10 measurements that are used to assess the air quality at a place. These are continuous measurements and the data is received at the IoT gateway, which are then processed by the stream processing engine.

The hands-on example is based on Apache Flink 1.4.0. A brief description of the architecture is shown in Box 12.7.

12.5.2 Steps to Develop the Air Quality Monitoring Applications

1. Create the sources of stream data for both NO_2 and PM10. In this example, we will create synthetic data streams for both of these pollutants, that is, simulate stream as if they were coming from a real IoT sensor measurement.

```
public void run(SourceContext<NO2_PM10_MonitoringEvent> sourceContext) throws Exception
{
    while (running) {NO2_PM10_MonitoringEvent NO2_PM10_monitoringEvent;
  while (random.nextDouble() >= 0.5)
    {
      /* consider the Mean and Standard Deviation of for each of the sensor measurements,
      and assume a Gaussian distribution of the values that are measured */
      NO2 = random.nextGaussian() * NO2Std + NO2Mean;
      PM10 = random.nextGaussian() * PM10Std + PM10Mean;
        NO2_PM10_monitoringEvent = new NO2_PM10_Event(stationId, NO2,PM10);
    }
  sourceContext.collect(NO2_PM10_monitoringEvent);
    Thread.sleep(pause);
}
```

CONSIDER THIS...

In order to simulate the IoT data, you may wish to use Apache Kafka, specifically Kafka producers for producing a stream of events. More information on Kafka producers is available here:
https://kafka.apache.org/0100/javadoc/org/apache/kafka/clients/producer/KafkaProducer.htm

2. Create an input stream of monitoring events. Here, a *sliding window* is used to get every 10 min a window that contains the events that arrived during the last 30 min.

```
// Input stream of monitoring events
  DataStream<NO2_PM10_MonitoringEvent> inputEventStream = env
    addSource(new NO2_PM10_MonitoringEventSource(
      SENSOR_ID,
      PAUSE,
      NO2_MEAN,
      NO2_STD,
      NO2_RATIO,
      PM10_MEAN,
      PM10_STD,
      PM10_RATIO
      ))
    assignTimestampsAndWatermarks(new IngestionTimeExtractor<>());
  inputEventStream
  keyBy("stationID")
  window(SlidingEventTimeWindows.of(Time.minutes(30), Time.minutes(10)));
```

In the aforementioned code, "keyBy" is used to access the items with the same key. In this example, we are accessing items based on the StationID (we have a total of 10 stations). This helps to generate warnings for each station. More about keyed and non-keyed streams is shown in Box 12.8.

BOX 12.8: KEYED AND NON-KEYED STREAMS IN APACHE FLINK

- **Keyed stream:** Partitions a single stream into multiple independent streams by a key (e.g., StationID of the generating event stream).
- **Non-keyed stream:** In this case, all elements in the stream will be processed together and our user-defined function will have access to all elements in a stream.

For more information, see:
https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/stream/operators/windows.html.

3. Create a warning pattern for our streams

```
Pattern<NO2_PM10_MonitoringEvent, ?> warningPattern = Pattern.<NO2_PM10_
MonitoringEvent>begin("first")
  subtype(NO2_PM10_Event.class)
  where(new IterativeCondition<NO2_PM10_Event>() {
    private static final long serialVersionUID = 3017L;
    @Override
    public boolean filter(NO2_PM10_Event value, Context<NO2_PM10_Event> ctx) throws Excep-
    tion {
      return value.getNO2() >= NO2_THRESHOLD;
    }
  })
    next("second")
    subtype(NO2_PM10_Event.class)
    where(new IterativeCondition<NO2_PM10_Event>()
    {
      private static final long serialVersionUID = 1392L;

    @Override
    public boolean filter(NO2_PM10_Event value, Context<NO2_PM10_Event> ctx) throws
    Exception {
      return value.getNO2() >= NO2_THRESHOLD;
    }
  })
    followedBy("third")
    subtype(NO2_PM10_Event.class)
    where(new IterativeCondition<NO2_PM10_Event>() {
      private static final long serialVersionUID = 1239L;
      @Override
      public boolean filter(NO2_PM10_Event value, Context<NO2_PM10_Event> ctx) throws
      Exception
```

```
      {
        return value.getPM10() >= PM10_THRESHOLD;
      }
  })
    followedBy("fourth")
    subtype(NO2_PM10_Event.class)
    where(new IterativeCondition<NO2_PM10_Event>() {
      private static final long serialVersionUID = 928L;
      @Override
      public boolean filter(NO2_PM10_Event value, Context<NO2_PM10_Event> ctx) throws Excep-
      tion
      {
        return value.getPM10() >= PM10_THRESHOLD;
      }
  })
within(Time.minutes(30));
```

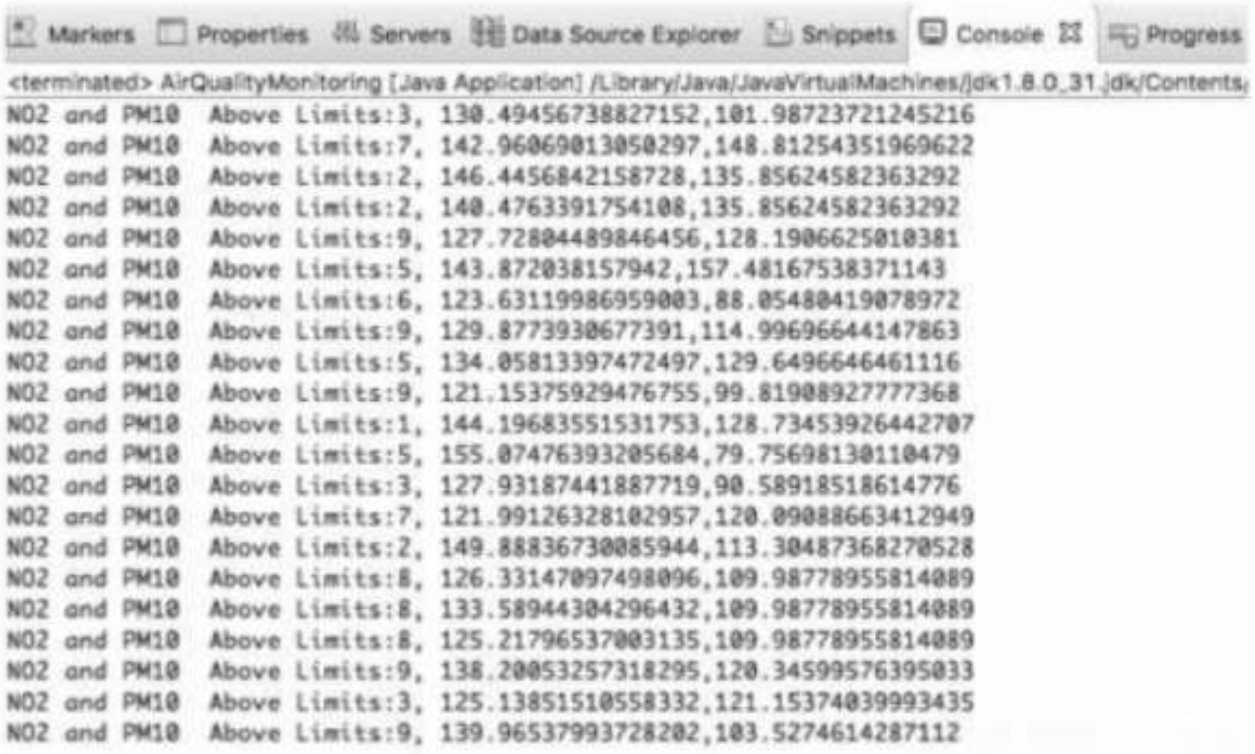
4. Create a pattern stream from our warning pattern.

```
// Create a pattern stream from our warning pattern
  PatternStream<NO2_PM10_MonitoringEvent> NO2_PM10_PatternStream = CEP.pattern(
    inputEventStream.keyBy("stationID"),
    warningPattern);
```

5. Air quality above predefined limits as set by the user. In this example, the values of NO_2 and PM10 were set at 120 and 50, respectively.

```
//AirQuality Warnings for each matched warning pattern

  AQwarning = NO2_PM10_PatternStream.select(
  (Map<String, List<NO2_PM10_MonitoringEvent>> pattern) ->
  {
    NO2_PM10_Event first = (NO2_PM10_Event) pattern.get("first").get(0);
    NO2_PM10_Event second = (NO2_PM10_Event) pattern.get("second").get(0);
    NO2_PM10_Event third = (NO2_PM10_Event) pattern.get("third").get(0);
    NO2_PM10_Event fourth = (NO2_PM10_Event) pattern.get("fourth").get(0);

    NO2_PM10_Warning=new NO2_PM10_Warning(first.getStationID(), (first.getNO2() + second.
    getNO2()) / 2,(third.getPM10()+fourth.getPM10()/2));
    System.out.println("NO2 and PM10  Above Limits:"+NO2_PM10_Warning.toString());

    return  NO2_PM10_Warning
  }

  ).returns(NO2_PM10_Warning.class);
```

6. The results from the above matched warning pattern for NO_2 and PM10 are shown in the following text.



7. Now let us do it for air quality monitoring based on Air Quality Index, which was defined earlier. Next shown is the inclusion of Air Quality Index calculator.

```
//AirQuality Warnings for each matched warning pattern

  AQwarning = NO2_PM10_PatternStream.select(
  (Map<String, List<NO2_PM10_MonitoringEvent>> pattern) ->
    {
      NO2_PM10_Event first = (NO2_PM10_Event) pattern.get("first").get(0);
      NO2_PM10_Event second = (NO2_PM10_Event) pattern.get("second").get(0);
      NO2_PM10_Event third = (NO2_PM10_Event) pattern.get("third").get(0);
      NO2_PM10_Event fourth = (NO2_PM10_Event) pattern.get("fourth").get(0);

      NO2_PM10_Warning=new NO2_PM10_Warning(first.getStationID(), (first.getNO2() + second.
      getNO2()) / 2,(third.getPM10()+fourth.getPM10()/2));

      double AQINO2=((((first.getNO2() + second.getNO2()) / 2)/120))*100;
      double AQIPM10=((((third.getPM10() + fourth.getPM10()) / 2)/50))*100;
      double AQIvalue=Math.max(AQINO2,AQIPM10 );

      if(AQIvalue >=0  &&   AQIvalue<=33){
      System.out.println(NO2_PM10_Warning.toString()+"="+"Very good (VG) air quality");
    }
```

```
    if(AQIvalue >33  &&   AQIvalue<=66){
      System.out.println(NO2_PM10_Warning.toString()+"="+"Good (G) air quality");
    }

    if(AQIvalue >66  &&   AQIvalue<=99){
      System.out.println(NO2_PM10_Warning.toString()+"="+"Fair (F) air quality");
    }

    if(AQIvalue >99  &&   AQIvalue<=149) {
      System.out.println(NO2_PM10_Warning.toString()+"="+"Poor (P) air quality");
    }

    if(AQIvalue >149  ) {
      System.out.println(NO2_PM10_Warning.toString()+"="+"Very poor (VP) air quality");
    }
      return  NO2_PM10_Warning;
      }
      ).returns(NO2_PM10_Warning.class);
      env.execute("Air Quality Monitoring Job");
    }
```

CONSIDER THIS...

Usually event processing is done on huge unending streams of events, but only a small percent of those events are really useful or of interest. Find out what is interesting that you want to capture in the domain of your choice and make a conceptual map of it before beginning to write the code.

8. The results based on the Air Quality Index for each station are given as follows:

CyberSecurity and Privacy

CHAPTER 15

"The good we secure for ourselves is precarious and uncertain, until it is secured for all of us and incorporated into our common life."

— Jane Addams

OBJECTIVES

- understand the introduction to IoT security challenges and security issues, privacy preservation, and security architecture in the various layers of IoT stack
- gain knowledge of various types of attacks on IoT devices pertaining to different layers, which affect the confidentiality, integrity, and availability (CIA) aims of IoT security
- understand different control methods to avoid security threats
- explore selected IoT case studies focussing on the security aspects and best practices for developing IoT systems

OUTCOMES

- assess security issues for a given IoT system application
- design IoT system with security controls for protecting CIA principles
- apply lightweight elliptic-curve cryptography (ECC) for data confidentiality and integrity in the IoT system application
- use best practices for IoT system security

REVISION

To increase your understanding of the material in this chapter, you can spend a few minutes reviewing key concepts in the following chapters:

Chapter 3: Concept and characteristics of sensors
Chapter 6: Open Hardware
Chapter 5: Understand the general framework for IoT platforms
Chapter 6: Understand the low-power IPv6 stack and related protocols
Chapter 7: Review the section on WSN communication patterns

15.1 INTRODUCTION

Internet of Things (IoT) technology is opening opportunities in various application domains. Billions of smart things will be connected in near future for data sharing, data analysis, and information retrieval. A huge

amount of data will be generated and shared by different IoT devices, containing private and sensitive information. Hence, data privacy and security has prime importance in an IoT system. An IoT can be a physical thing, or software, or any virtual object that has the ability to communicate with the outside world through the Internet to provide a smart service. In addition to data security, to develop a reliable application suitable for any business process, which is protected from all kinds of threats (e.g., interruption, interception, modification, and fabrication), it is necessary to implement hardware, software, and network security at different network communication layers. The major cause of the ever-growing security risks to IoT devices is due to the diversity of new IoT applications that are being developed in a variety of domains.

Three main classes of IoT systems are as follows:

Customer friendly These devices are sold to customer. Any special expertise is not needed to avail the service from these devices. For example: IoT based Smart Bulb, Smart cooking appliances, etc.

Always live These devices are connected to the Internet permanently and are always on. Example: Surveillance devices.

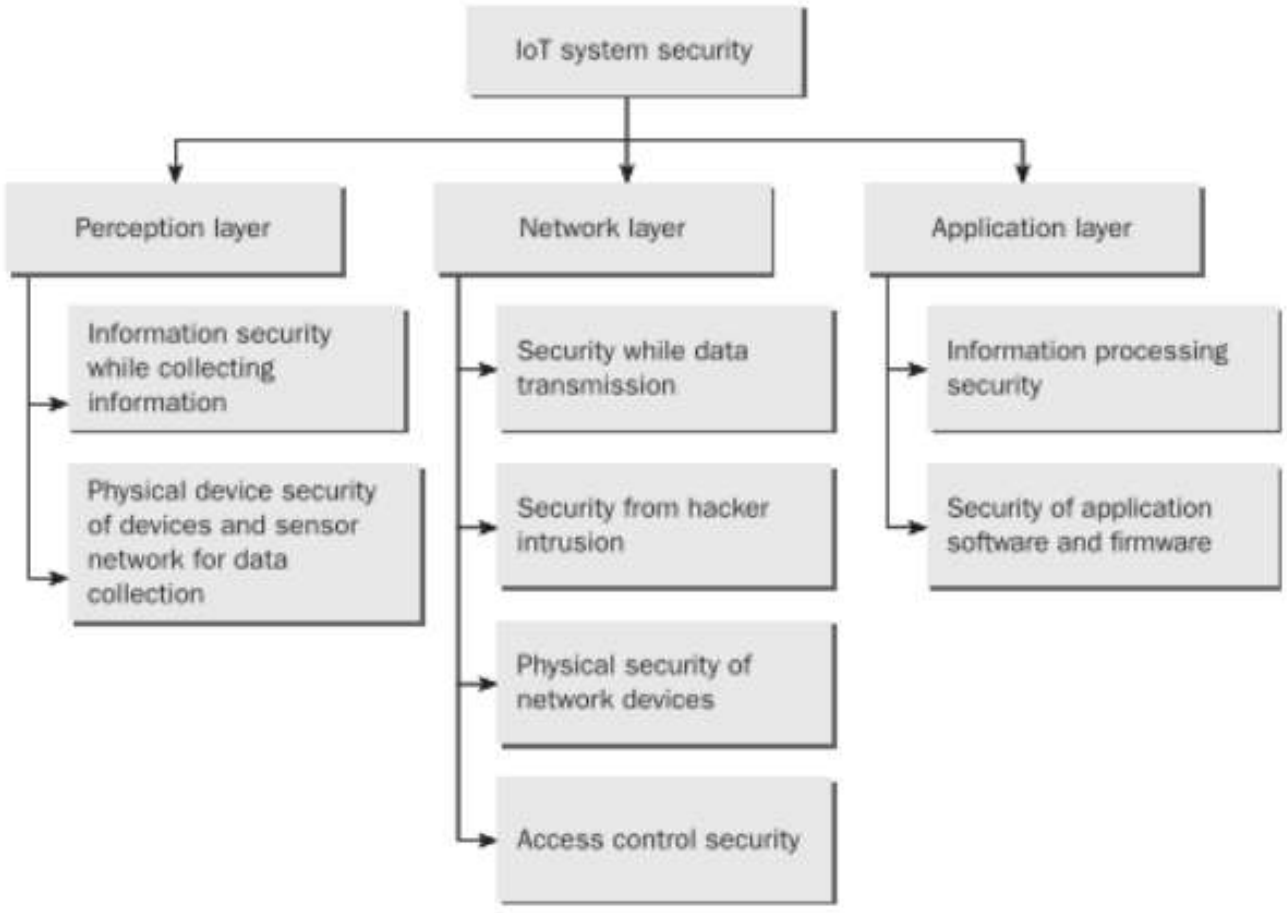


Fig. 15.1 IoT System Security Architecture

Devices interacting with the physical environment These devices have the capability to measure various phenomenon on Earth using specialized sensors to run models and make predictive analysis. For example: IoT-based weather station, water level monitoring stations, etc.

Each of the above class of IoT systems has different types of security requirements. While considering security measures for these IoT devices, there is a need to incorporate best practices so that the threats can be minimized. There are unique security challenges for IoT devices specifically as their characteristics deviate from normal devices. Figure 15.1 depicts the high level architecture for IoT system security in three different layers such as Perception layer, Network layer, and Application layer.

15.1.1 IoT Security Challenges

Device size and processing power IoT devices are small in size with limited CPU power for providing efficient security enforcing computations. As these devices have limited processing power, it is necessary to design lightweight encryption techniques.

Power consumption In general, various sensors in IoT system need to measure an environmental phenomenon with a certain temporal frequency, specific to the application, which can consume power constantly.

Hence, another challenge is to provide uninterruptible power supply for considerably long duration to keep the sensors working to enable continuous measurements, and at the same time ensure that malicious attacks do not artificially increase the power consumption.

Access control Development of key exchange management and identity management for providing authorized access control for securing confidentiality for a variety of IoT devices is challenging. Unless this is accomplished in a robust way, various IoT applications based on these devices are very vulnerable for attacks.

Edge security Endpoint device security is another challenge for securing various new IoT devices such as new firmware, new embedded software, and OS.

Exploding network traffic As a large number of IoT devices are expected to be connected through the Internet in the next few years, issues related to huge network traffic are required to be addressed. As network security is complex, increased network traffic will make it more complex while solving traffic issues and the traffic analysis will become a challenging task.

Based on the aforementioned text (see also Fig. 15.2), it is also important to understand the security aspects from a domain perspective. It will enable to clearly highlight the various issues in a particular domain and the remedial measures that are required to address them.

Fig. 15.2 Security Components in IoT Systems

15.1.2 IoT System Security Domains

IoT system security is required in the following security domains:

Information security Implemented by end-to-end encryption, secure network protocols, network access control to authentic users.

Physical security Enabled by providing proper casing of the hardware, controlling physical device access to legitimate users, etc.

Information technology security Ensuring authentic softwares with proper certification of authenticity with less vulnerability to avoid attacks.

Operational security Enforced through the development of secured software with layered security to avoid backdoors.

It is envisaged that an IoT system will combine all these domains; hence, an integrated approach is necessary to detect the issues and provide countermeasures. In addition, as IoT systems are going to massively invade the private space both of an individual and organizations; there is a lot of interest in developing ways to handle privacy issues. For example, intruders gaining access to homes through IoT-enabled home security systems, baby monitors, accessing healthcare-related information through smart medical devices, intruding into industrial process that are managed by IoT-enabled devices, etc.

15.2 SECURITY ISSUES IN IOT SYSTEMS AND PRIVACY PRESERVATION

Privacy is a very sensitive issue while providing IoT services. Through IoT devices, which generate large amount of data, users are gaining access to many personalized services and data. Few examples of IoT services are energy consumption control, smart parking, remote-monitoring of patients, production chain, and inventory management. In all these services, users need protection of their personal data related to their movement, habits, and interaction with other people. Figure 15.3 shows the various

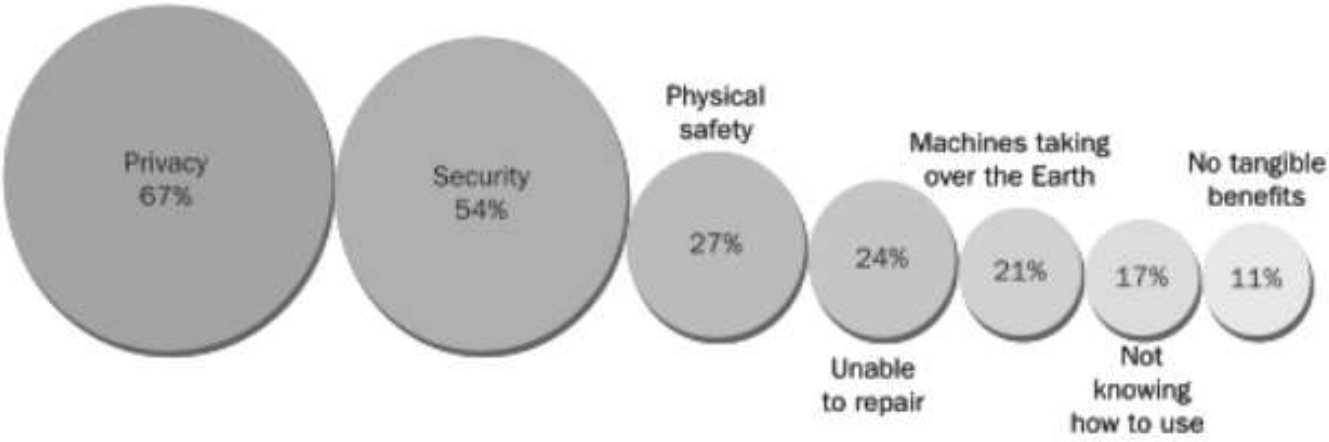


Fig. 15.3 Various Concerns While Adapting IoT Services

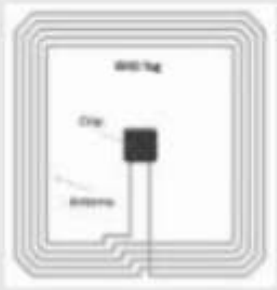
(Source credit: Ericsson White papers, 2017)

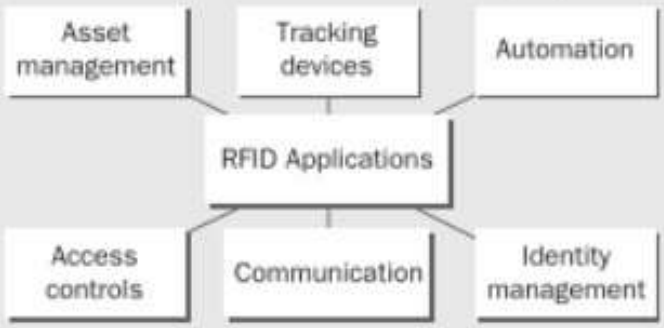
concerns that users of IoT-based applications face. It is clear that privacy is of utmost concern and needs to be incorporated in most of the IoT services.

Further, the wide usage or adaption of an IoT service depends on how secure it is from various perspectives. To further understand these aspects, it is pertinent to look at the three-layered IoT architecture and the detailed functionality of each of its components. Box 15.1 explains the RFID technology.

BOX 15.1: RFID TECHNOLOGY

Radio Frequency Identification (RFID) technology is used in information tags for automatic communication with each other. In a RFID tag an antenna is embedded in a microchip. The RFID tag also consists of a memory chip which embeds a unique identifier known as Electronic Device Code (EDC).





Different types of RFID tags are:

Active tag These types of tags have in-built battery for sensing the information from a limited distance.

Passive tag In this type of tag the information is read from this tag by activation of the tag by a transceiver from a specific distance. No internal battery is present in this type of tag.

Radio transceiver (RFID reader) RFID reader interacts with RFID tags to read the information by using unique identifier of the tag under scan using its Electronic Product Code (EPC) as shown below. The products with EPC can capture unique identifiers at very high rates using radio frequencies, without the need for line of sight. A distance of 10 meters is usually preferable.

15.2.1 Three-layer IoT Architecture and Security Issues in Each Layer

IoT system architecture has three main layers, which are as follows:

- Perception layer
- Network layer
- Application layer

15.2.1.1 Perception Layer

Main functionality of the perception layer in the IoT system is collection of information using different devices such as Smart card, RFID tag, and other sensors in domains such as healthcare, transportation, retail, and industrial IoT. The main security issues in perception layer are terminal security, sensor network security by proper authentication and access control mechanism to ensure confidentiality, and availability of IoT device services to the legitimate users. Figure 15.4 shows specific security issues that are related to the perception layer. Node capture, fake node and malicious data, denial of service attack, timing attack, routing threats, and replay attacks are some of the threats in this layer. More details of these threats are given in Section 15.3.

15.2.1.2 Network Layer

Various components of wireless and wired network components in the network layer of the IoT system need protection from threats due to different vulnerabilities present in the network components. Main functionality of the network layer of an IoT system is the transmission of data while preserving the confidentiality and integrity of the data. In the network layer, many vulnerabilities are present at the entry points and can be exploited by attackers such as eavesdropping, tampering the data by modification, insertion, or fabrication. Physical security of network layer devices, such as interfacing devices, is important to deliver uninterrupted service of IoT applications. By ensuring data encryption method suitable for IoT applications and access control to the networking devices, security goals can be achieved.

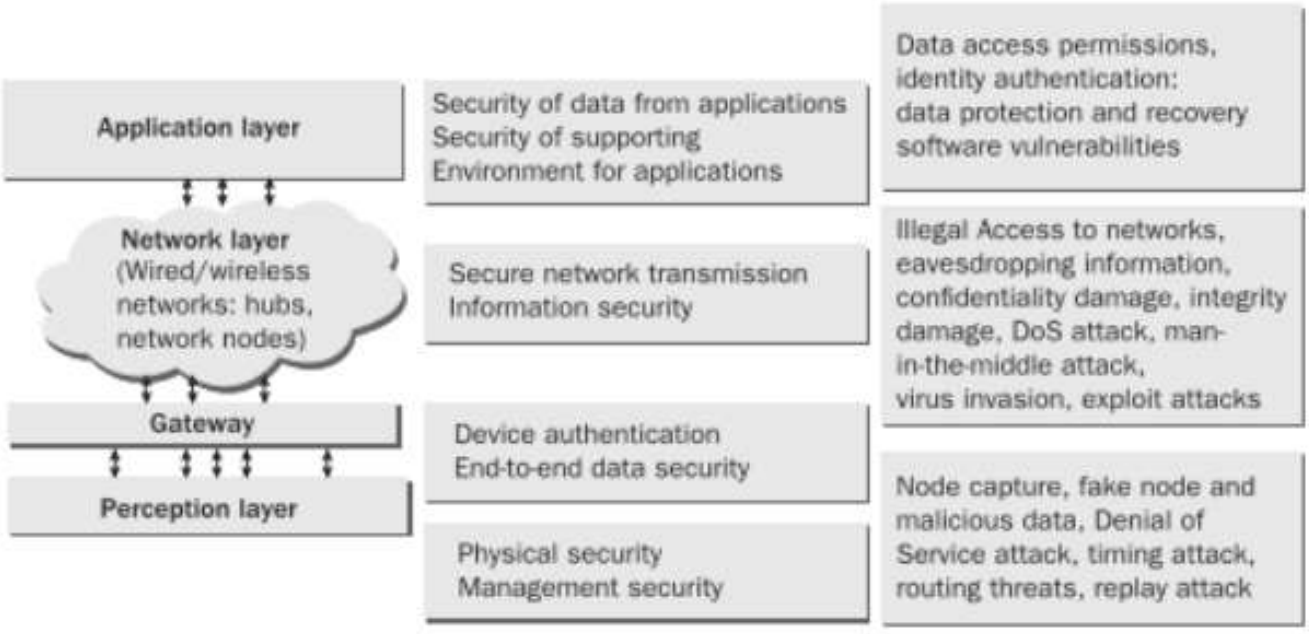


Fig. 15.4 Three-layer Architecture of IoT and Corresponding Security Aspects

15.2.1.3 Application Layer

The application layer, sometimes known as processing layer of the IoT system, has intelligent devices performing the data analysis task for information retrieval and for deriving inferences to support the decision making for further control. It provides a user interface for interacting with the IoT system. Intelligent computing technologies are used to process the information to control different objects and devices. These smart controls decide, what, when, and how to accomplish it. The main security goal that is to be achieved in this layer is privacy while processing and transmitting the data, which can be obtained by implementing end-to-end encryption.

15.3 IOT SECURITY REQUIREMENTS BASED ON CIA PRINCIPLES

IoT system security has to address basic security goals as defined by CIA (Confidentiality, Integrity, and Availability). It has to incorporate security parameters such as privacy and authentication that supports the security goals, confidentiality, and availability.

Confidentiality Any sensitive information should not be leaked to illegitimate users through the data reading devices. Confidentiality must be maintained while collecting, processing, transmitting, and storing the information.

Integrity While transferring the information from the IoT device to IoT application through various layers, ensuring the data security is necessary to focus on threats such as interruption, interception, modification, and fabrication so as to avoid leakage and tampering of the information, and also to ensure the availability of the data to the legitimate user.

Availability Availability and access to different kinds of IoT enabled smart devices to legitimate users is a primary goal in IoT security. Balancing is needed while achieving two goals, that is, confidentiality and availability. As while securing confidentiality, availability of services to legitimate users and objects, may get affected.

Section 15.3.1 describes various attacks that are possible on IoT systems.

15.3.1 Denial of Service (DoS) Attacks in Physical Layer

In this layer, the actual transmission and reception of the data is carried out with selection and generation of carrier frequency signals, modulation and demodulation, encryption, and decryption techniques. Various attacks that are possible in this context are as follows:

Jamming In this DoS attack, the intruder exhausts the bandwidth of the channel by creating fake traffic to prevent the nodes of WSN from communicating with each other.

Node tampering In this attack, the WSN node is tampered with by eavesdropping to steal and tamper some important information. Thus, the availability of correct information is affected.

Additional security goals are listed in Box 15.2.

BOX 15.2: ADDITIONAL SECURITY GOALS

In addition to the CIA-based security measures, IoT systems need four more such as:

Authenticity: For preserving confidentiality, the information reading devices should receive information from legitimate users, and the processed information should reach to the authentic concerned person to preserve privacy of information and to ensure non-repudiation.

Reliability: It refers to the ability to provide continuous service to the end user without interruption and on demand. In addition, service is trustworthy, precise, and meets the users' requirements.

Resilience: The IoT system should be able to quickly recover from any attacks, accidents, and other kinds of incidents, adapt to continuously changing environmental conditions, resist perturbations, and get back to normalcy within a tolerance limit. An example is the recovery of a communication system after a disaster and it continues to operate on other modes albeit in a limited fashion.

Safety: Absence of conditions that could cause physical damage to people and property. The IoT system should not pose any risk to the physical well-being of a person(s) who interact with it on a daily basis. A typical example of such an environment is Industrial IoT.

15.3.2 DoS Attacks in Link Layer

Various data streams are multiplexed in the Link layer of IoT system. The data frame detection and error control and medium access control functionalities are also provided by Link layer. Some of the attacks on Link layer are as follows:

Collision When two packets are transmitted on the same frequency, the collision of the packets happens due to sharing the transmission medium simultaneously and the data packets are corrupted due to the collision. Hence, retransmission of the data packets is required to remove the error.

Battery exhaustion When a DoS attack generates large amount of data traffic, the transmission channel is available only for a few devices in an IoT network; hence, large number of requests (request to send) are sent by many other devices for data transmission continuously, which ends up in exhausting the battery of some of the IoT devices.

15.3.3 DoS Attacks in Network Layer

While routing the packets in WSN, some specific attacks take place such as the following:

Spoofing Replaying and misdirection of the traffic is done in this attack by sending useless messages to exhaust the bandwidth of the channel to affect the network availability of a legitimate user.

Selective forwarding In this attack, the compromised node is used for forwarding the messages to only selected nodes to enable the intention of the attacker to perform the malicious activity (Fig. 15.5).

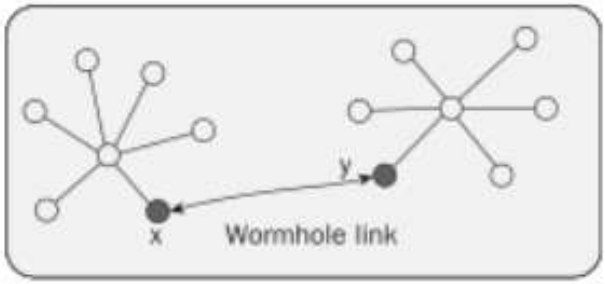


Fig. 15.6 Wormhole Attack

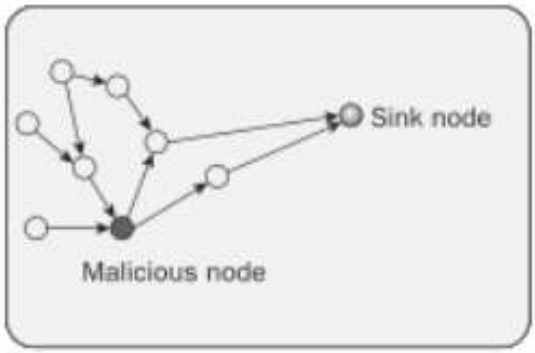


Fig. 15.5 Malicious Node in the IoT Network Performs Selective Forwarding to Divert the Traffic for Some Attack

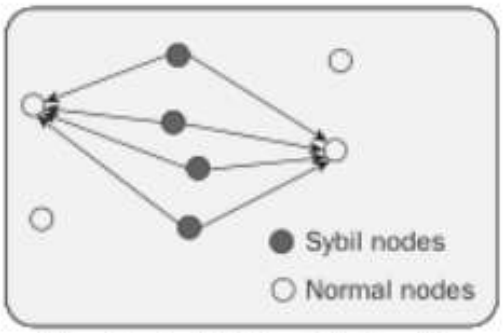


Fig. 15.7 Malicious Node with Multiple Identities

Homing In this attack, search is done in the traffic to find out nodes, which do key management and search cluster heads for taking control for malicious activities.

Wormhole In this attack (Fig. 15.6), through tunnelling of bits of data on a low latency link, a relocation of the data packet is carried out by changing the data position on the network.

Sybil An attacker replicates a single node with multiple identities to other nodes in the Sybil attack (Fig. 15.7).

Acknowledgement flooding In this attack, intruder spoofs the wrong acknowledgment to the sensor network node.

15.3.4 DoS Attacks in Transport Layer

Transport layer of WSN due to its architecture avoids congestion by reducing traffic and provides reliable transmission. The attacks on this layer are as follows.

Desynchronizing In this attack, fake messages are generated at endpoint nodes asking for retransmission of message, though no error exists in the message. Such messages increase traffic and consume battery power at node to execute the additional frequent instructions for retransmission.

Flooding In this attack, traffic is generated by useless messages to create congestion for affecting availability of transmission facility to authentic user.

15.3.5 DoS Attacks in Application Layer

In application layer of IoT, while carrying out software services, this layer also provides traffic management service. Path-based DoS attacks are dominant in this layer by simulating the IoT sensor nodes for creating huge traffic towards the IoT gateway.

Some other DoS attacks are node subversion, node outage, node malfunction, false node, message corruption, neglect and greed, black holes, and interrogation.

15.4 SECURITY TECHNOLOGIES

For secure communication between IoT devices to avoid open inbound network ports, various network connectivity technologies are needed as shown in Table 15.1.

Table 15.1 IoT System Layers and its Specifications for Security Controls

IoT Layer	Components	Working of Layer	Security Issues	Security Parameters	Counter-measures
Perception layer	Smart card, RFID tag, sensors	Collection of information	Terminal security, sensor network security	Authentication, confidentiality	Certification and access control
Network layer	Wireless or wired network, computer, components	Transmission of information	Information transmis-sion security	Integrity, availability, confidentiality	Hop-by-hop data encryp-tion
Application layer	Intelligent devices	Analysis of information, control decision-making	Information processing	Safety of IoT, privacy	End-to-end encryption

15.4.1 Network Connectivity Technologies and IoT Device Security Issues

To take care of different security issues, a well-planned and designed security model is needed for implementing IoT systems. Internet connectivity is a primary requirement for IoT devices. Hence, best practices for IoT devices are discussed in this section, considering network security as primary concern.

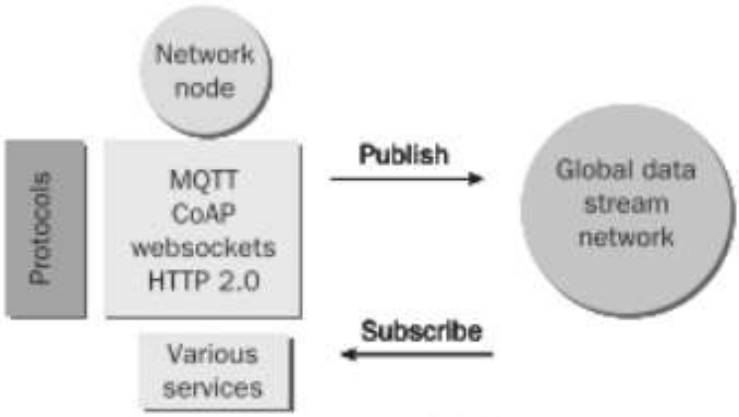


Fig. 15.8 Connectivity Technologies

Protocols for secure and reliable communication For secure and reliable communications protocols for low powered devices such as, MQTT, CoAP, Web Socket, and HTTP 2.0 facilitates Publish-Subscribe secure communication between IoT devices with no open ports (see Fig. 15.8). To cater to the needs of IoT scale, distributed high-performance servers replicated at multiple points are needed to handle data streams.

Table 15.2 lists various IETF standards designed specifically for low-powered devices such as those that are usually implemented in IoT systems.

Table 15.2 IETF Standards that are Useful in Implementing the IoT System

Standard	Purpose	URL
6LowPAN	IP connectivity	http://datatracker.ietf.org/wg/6lowpan/
ROLL	IP connectivity	http://datatracker.ietf.org/wg/roll/
CoAP	Generic web protocol definition	http://datatracker.ietf.org/wg/core/
CoRE	Lightweight REST web service architecture	http://datatracker.ietf.org/wg/core/

Further, various security aspects of these standards based protocols are compared in Table 15.3.

Table 15.3 Comparison of Security Aspects in Various Application Layer Protocols

Protocol	Security Goal	Purpose	Features	Application Area
Message Queue Telemetry Transport (MQTT)	Confidentiality, data integrity	Machine-to-machine communication	Small footprint and minimal bandwidth consumption	Mobile applications and useful for connection with remote location
Constrained Application Protocol CoAP	Confidentiality, data integrity	Multiplexing responses for sending parallel responses	Simple, low overhead and multicasting capability	Resource constraint, Internet devices such as wireless sensor node
Web socket	Integrity, confidentiality	Monitoring systems and for quick and/or constant updates	Bidirectional transactions	Secure financial transactions
HTTP 2.0	Data integrity, confidentiality, and availability	Data streaming protocol intended to speed up browser-side and server-side transactions	Bidirectional and full duplex messaging, compact transmissions and low overhead	Realtime bidirectional transactions
Low-Power Wireless Personal Area Networks (6LoWPAN)	Integrity, confidentiality	Low-power devices with limited processing capabilities	Low-power consumption	Home, office, factory

15.5 IOT SYSTEM SECURITY CONTROLS

CIA goals for data security in an IoT system are described in this section.

15.5.1 IoT Security for Data Access, Integrity, Availability, and Data Communication

15.5.1.1 Data Confidentiality

As in the sensor network, many hops of data through intermediate nodes take place. Hence, there is a high threat of data leakage. For securing the data while data transmission, data is encrypted and only recipient can decrypt the data.

15.5.1.2 Data Integrity

The data received by receiver should be in its original form without any alteration through modification or fabrication is called as data integrity. Intruders change the original data for some malicious purpose affecting data integrity.

Secure and reliable communication requirements

Strong encryption and secure protocols In IoT systems, while transmitting the data over the network different encryption techniques are used to achieve data integrity. Various attacks on data integrity such as modification and fabrication can be avoided with strong encryption techniques.

End-to-end encryption Transportation Layer Security (TLS) is an industry standard layer for communication to send encrypted data over wide area network (see Fig. 15.9). Added layer security is given with AES encryption to provide end-to-end encryption. TLS/SSL protects the data streaming at top level. Advanced Encryption Standard (AES) is used for data encryption. AES is paired with TLS for key management. The message body is encrypted with AES, while the envelope that holds the key (has to be used at midstream) is encrypted at the endpoint with TLS to provide true end-to-end encryption in IoT systems.

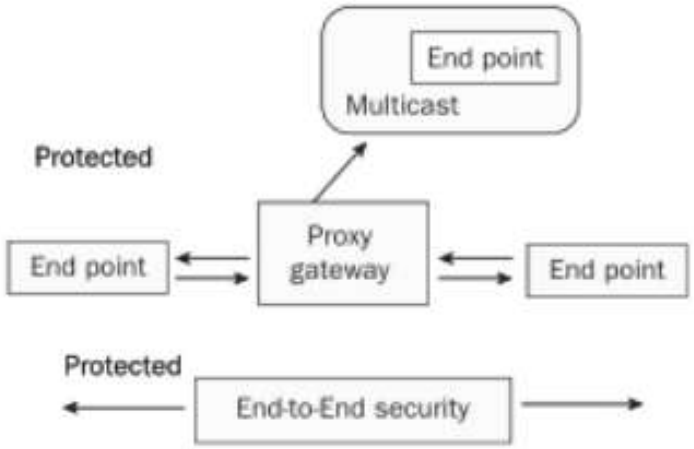


Fig. 15.9 End-to-End Encryption

Lightweight cryptography It is challenging to implement cryptographic functions in a constrained computation environment. A lig tailored to work in a constrained

environment, which is useful for hardware devices, which are having limitations in RAM size, energy consumption, and chip size (e.g., RFID tags, different kinds of sensors, contactless smart card healthcare devices, etc.). Lightweight cryptography provides adequate security, though it is not always good to exploit the trade-off between security and efficiency.

Symmetric-key cryptography Broadly, there are two kinds of cryptographic systems— symmetric-key and public key cryptosystems. In symmetric-key schemes, the communicating entities first agree upon keying material that is both secret and authentic. Subsequently, they may use a symmetric-key encryption scheme such as the Data Encryption Standard (DES), RC4, and Advanced Encryption Standard (AES), etc. The major advantage of symmetric-key cryptography is its high efficiency; however, there are significant drawbacks to these systems. The major drawback is the key distribution problem, i.e. the need for a channel that is both secret and authenticated for the distribution of keying material.

Block cipher In AES, many block ciphers with lightweight properties such as CLEFIA and PRESENT are proposed. These ciphers are ready for practical use.

Stream cipher ECRYPT II eSTREAM project held from 2004 to 2008 selected a set of effective new stream ciphers.

Hash function NIST's new cryptographic hash algorithm 'SHA-3' is one of the popular hash algorithms as a general purpose hash algorithm, but most of its versions do not have lightweight properties. Research on lightweight dedicated hash functions based on lightweight block ciphers is an ongoing effort.

Public key cryptography For key management, lightweight public key cryptographic protocols are preferred in the IoT network system. The resources requirement for public key primitives is much larger than that of symmetric key primitives. There are no trustable primitives, which can meet the balance between security and efficiency; however, if the focus is on lightweight properties (as compared to RSA), some public key primitives (e.g., ECC) can be implemented with relatively small footprint.

15.5.2 Need for Lightweight Cryptography for IoT Systems

To achieve end-to-end security and secure and efficient end-to-end communication where end node-points perform processing, implementation of symmetric key algorithm can be done. However, for IoT devices with constraints on the resources, cryptographic solutions having less energy consumption is needed. In such applications, lightweight cryptographic algorithms are suitable.

Lightweight cryptographic solutions having smaller footprint when used for secure communication consume less bandwidth to provide optimum use of network resources for more IoT devices.

15.5.2.1 Elliptic Curve Cryptography (ECC)

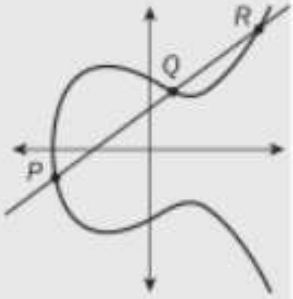
Elliptical Curve Cryptography (ECC) is used as a lightweight cryptosystem. Certain criteria are needed to be considered, while choosing public key cryptosystem for specific applications, such as capability of the public key cryptosystem, security provided by the crypto system protocols and performance objectives of public key cryptosystem method to meet the required level of security (refer to Box 15.3).

BOX 15.3: ELLIPTIC CURVE KEY GENERATION

The equation of an elliptic curve is:

$$y^2 = x^3 + ax + b$$

- Let E be an Elliptic Curve defined for a finite field F.
- If P is a point on this elliptic curve and if the prime order of point P is n then the cyclic subgroup of the elliptic curve over the finite field F generated by P is $|p|=\{\infty, P, 2P, 3P, \ldots, (n-1)P\}$.
- The elliptic curve E, the point P and its order n, the prime p, are the parameters for the public domain.
- A private key is an integer d and it is selected from the interval $[1, n-1]$ randomly and the corresponding public key is $Q = dP$.
- Using elliptic curve discrete logarithm (ECDL), d is determined when the domain parameters and Q are given.



$y^2 = x^3 + ax + b$

Advantages of using ECC for IoT systems:

- Achieving a certain level of security with shorter keys. Thus, making it a lightweight cryptosystem, which is useful for the IoT systems that are resource constrained to handle computational complexity.
- Very few attacks are developed so far for ECC cryptosystems due to the unexplored mathematics of arithmetic operations to form Abelian groups to add points on the elliptical curve.

ECC is based on mapping plain text as points on an elliptic curve, which also adopts the idea of public key cryptosystem for key management as well as providing security in the process of key transmission and management.

15.5.2.2 Elliptical Curve Groups

Consider the example, where p is a prime number and let F_p be the field of integers modulo p. The elliptical curve $E(F_p)$ is defined by an equation of the form

$$y^2 = x^3 + ax + b \qquad 15.1$$

Where, $a,b \in F_p$ satisfy $4a^3 + 27b^2 \equiv 0 \pmod{p}$. A point $P(x, y)$, where $P(x,y) \in F_p$ is a point on the elliptical curve if $P(x, y)$ satisfies Equation 15.1.

Let us consider a field for prime number $p = 7$. Let $E(F_p)$ is the elliptical curve over field $F_p = F_7$.

For example, let the elliptical curve has defining equation,

$$y^2 = x^3 + 2x + 4 \qquad 15.2$$

Then the points (as shown below) that are on the elliptical curve are:

$E(F_7) = \{\infty, (0, 2), (0, 5), (1, 0), (2, 3), (3, 3), (3, 4), (6, 1)\}$

Let $P(x, y) = P(0, 2)$

$x = 0$ and $y = 2$ satisfy the elliptical curve Equation 15.2.

$$x^3 + 2x + 4 \equiv 4 \bmod 7 = 4$$

$$y^2 \equiv 4 \bmod 7 = 4$$

Hence point $P(0, 2)$ is on the elliptical curve $E(F_7)$, and satisfies the Elliptical curve Equation 15.2.

15.5.2.3 Elliptic Curve Key Pair Generation

Elliptic curve encryption method Here a procedure for the elliptical curve encryption and decryption scheme is explained using the basic ElGamal encryption scheme. A plain text m is presented by a point M and this point is encrypted by adding it to kQ, where k is an integer selected randomly and Q is a public key for the recipient of the message. The sender transmits the encrypted ciphertext {C1, C2} where, C1 = kP. P is a point on the elliptical curve to be made public to the recipient for key generation and C1 = kP who decrypt the received ciphertext using the private key d, to compute dC1 = $d(kP)$ = $k(dP)$ = kQ and then M is computed by the equation M= C2 – kQ. An eavesdropper who is listening to the transmission and wish to recover M must know kQ. The computation of kQ using the domain parameters Q and C1 = kP is the elliptical curve analogue of Diffie-Hellman key exchange algorithm.

Pseudocode

```
Input: Elliptic curve domain parameters
(p, E, P, n)
Output: Public key Q and private key d
generated from the public point P
        Select d ∈ R [1, n-1]
        Compute Q = dP
        Return(Q, d)
```

15.5.2.4 Basic ElGamal Elliptic Curve Encryption

The basic ElGamal is public-key cryptography which uses an asymmetric key encryption algorithm based on the Diffie–Hellman key exchange. ElGamal encryption consists of three steps namely the key generator, the encryption algorithm, and the decryption algorithm.

Pseudocode

```
Input: Elliptic Curve domain public key Q,
plaintext m, and parameters (p, E, P, n)
Output: Cipher text (C1, C2).
Represent the message m as a point M in E(F).
        Select k ∈ R [1, n-1]
        Compute C1 = kP
        Compute C2 = M + kQ
        Return (C1, C2)
```

15.5.2.5 Basic ElGamal Elliptic Curve Decryption

Pseudocode

```
Input: Domain parameters (p, E, P, n),
private key d, cipher text (C1, C2)
Output: Plaintext m
   Compute M = C2 - dC1, and extract m
   from M
   Return (m)
```

Solved Exercise

Exercise: If some plaintext m is representing a point $M(3, 3)$ on an Elliptical curve, $E(F_7)$, with equation, $y^2 = x^3 + 2x + 4$, then using Elliptical Curve Cryptographic method generate the public and private keys using public point $P(2, 3)$. Using these keys find the ciphertext, using ECC encryption algorithm. Apply ECC decryption algorithm to recover the point M. Given prime number p with prime order n = 101.

Solution

Elliptical curve key pair generation

Given: In the Elliptical curve $E(F_7)$, where, the prime number $p = 7$, Public key Q, private key d.
Given prime order $n = 101$
Primary key is an integer d randomly selected from the interval $[1, \ldots, (n\text{-}1)]$
In this case, the range is $[1, 100]$ as $n = 101$
Let private key, $d = 99$
Public key $Q = dP$, where P is the point on the elliptical curve which is public and to be used for key generation.
Here P is $P(x, y) = P(2, 3)$ and elliptical curve is $E(Fp) = E(F_7)$
Public key $Q = (2d, 3d)$
$Q = (2 \times 99, 3 \times 99)$
$Q = (q_1, q_2) = (198, 297)$

ECC encryption

Given (p, P, d, Q)
$p = 7$, Message point: Plaintext point: $M(3, 3)$, $d = 99$, $Q(198, 297)$
Let $k = 10$, be a randomly chosen integer
$C_1 = kP = \{10 \times 2, 10 \times 3) = \{20, 30\}$ // P is a public point
$C_2 = M + kQ$
$C_2 = \{3 + kq_1, 3 + kq_2\}$
$C_2 = \{3 + (10 \times 198), 3 + (10 \times 297)\}$
$C_2 = \{1983, 2973\}$

ECC decryption

Given $C_2 = \{1983, 2973\}$, $d = 99$, $C_1 = \{20, 30\}$
$dC_1 = \{99 \times 20, 99 \times 30\}$
$dC_1 = \{1980, 2970\}$
$M = C_2 - dC_1$
$M = \{(1983 - 1980), (2973 - 2970)\}$
$M = (3, 3)$
Hence point M is decyphered from the encrypted cyphertext as $M(3, 3)$.

15.5.4 Token-based Access Control

Though AES and TLS/SSL are used for data encryption, even fine-grained access control, over what to be transmitted and to whom it should be sent is required and it is quite challenging to implement fine-grained access control.

Token can be distributed to devices for granting access in publish/subscribe structure. A fine-grained control on the access of data is achieved in this paradigm and centralized control can be established for revoking the access permissions of the devices. In this way, network security architecture gains capability to manage the devices speaking to and listening to specific devices, which have the required token for the access.

15.5.5 Device Status Monitoring

It is difficult to monitor online and offline status of the IoT devices of various applications. When a device stops sending and receiving data, the reason could be local tampering or it may be due to Internet or power outage.

A separate devoted channel is needed for sending the data streams to easily track the IoT metadata such as the online/offline status of the device in IoT systems. Such a dedicated channel is useful for sending an alert to the owner about the change in the lock status if the owner's phone is not in the decided range when the lock will be opened. If a sensor network goes offline, a maintenance person can be sent to the site immediately to take the appropriate action.

Real-time and highly reliable IoT applications will gain the confidence of both consumers as well as manufacturers. As per predictions, 50 billion new IoT devices will be there in next 5 years. Without the trust in the system devices, their adoption will be difficult.

15.5.6 User-friendly Set-up and Upgrades

For IoT devices to be up in the working state always, it is necessary to keep them updated by upgrading the software and the firmware of the devices.

Most of the times instead of receiving reliable service from the vendor for the IoT device, the customer is held responsible for failure of service due to some IoT device set up. For example, a Wi-Fi-enabled camera used as a part of the IoT system for house security is held responsible for blocking their home firewall or broadcasting the packets to the wrong port. Hence, a user-friendly set-up is an important factor for success of any IoT system.

15.5.7 Access Control for Availability

Fine-grained access control is an essential requirement for IoT devices. As in near future, billions of devices will be trying to listen from the assigned port, it is insecure and inefficient to filter out unwanted messages at the endpoint device for listening the correct one. Instead, the network should carry out the bulk of the task.

A token-based access control can be used under publish/subscribe paradigm. In this approach, the end IoT devices will be given access to certain ports based on the token distributed to them with specific grant permission for the data use. This fine-grained control will assign permissions to the end devices to access for listening and speaking to specific ports on the network.

15.5.8 IoT Security Controls for Middleware Platforms

In IoT paradigm, various heterogeneous technologies are present at different layers of the IoT system architecture. Several types of middleware layers are developed for providing integration and security

to the devices. Different middleware platforms during communication, must respect the security of IoT devices. Hence, for communication between different middleware technologies, different communication mediums need to be considered in the design, development, and deployment of IoT systems.

15.6 OTHER SECURITY CONTROLS FOR IOT SYSTEMS

Few more security controls for IoT systems are discussed below.

15.6.1 Fault Tolerance

As billions of smart objects will be added through IoT systems to Internet, many vulnerabilities will be introduced which will help to perpetrate attacks. Hence, fault tolerance is extremely important to provide reliable IoT services. To achieve fault tolerance, cooperative efforts are needed. All objects must be made secure from different threats by default. Development of secure protocols and high-quality software is of prime importance, to reduce the software patch requirement to upgrade the software to remove bugs and loopholes.

IoT objects should be designed to provide capability to know network status and should be able to give feedback to the network and also should be able to defend themselves from network failures and attacks.

15.6.2 Privacy Preservation Methods

Privacy by design This is one way for providing privacy where design of the system is such that it incorporates certain tools, which facilitates the users to protect and manage their own data. Whenever user produces some data fragment, control access is given to outside world through some dynamic consent tools.

Data management It is very important who manages the data and how the secrecy and access control tools are deployed. It is not feasible for some systems to incorporate encryption and access control mechanism. In such scenarios, establishment of data management policy is mandatory.

Transparency Users must know which objects are managing their data, when and for what purpose that data is used. Hence, transparency is necessary. Stakeholders such as service providers should execute licence agreement for the data usage by considering duration and usage type for different functionalities.

15.6.3 Identity Management

Considering variety of identities and different relationships associated with those identities of various objects, in IoT system, identity management is required. According to object identity principles,

- An object can have a main identity and many other temporary identities
- An object identity can be different than the identity of the underlying mechanism
- An object can be identified using its identity or by using its specific feature
- Objects know the identity of the owner

Identity management in IoT offers both challenges and opportunities in practical use of security.

15.6.4 Trust and Governance

The term 'Trust' is used in different contexts with various connotations. Here, while mentioning trust in IoT devices, as most of the smart objects/devices are human-operated, malicious nodes try to attack basic functionality of IoT by trust-related attacks such as self-promoting, bad mouthing, good mouthing, and so on. For trust evaluation, various parameters that are judged are: honesty, cooperativeness, and community interest. These attributes are useful to develop trust management protocols to cope with the need of the changing environment and also to adapt to the changes in the trust parameters.

Trust management system for IoT has ability to assess truth level of a node from its behavioural patterns based on various cooperative services. Various trust models are suggested in the literature to develop trust management system for IoT. However, following issues related to trust management are yet to be explored for further advancement of trust management systems for IoT:

- Supporting semantic interoperability in IoT context, for introducing well-defined trust negotiation language
- Defining smart objects or devices management systems
- Developing models for trust negotiation mechanism for controlling data stream access

Governance helps in strengthening trust in the IoT system. A proper governance mechanism and a good framework can reduce liability. Yet, there are many challenges while implementing governance for IoT system.

THOUGHT EXERCISES

- Design a suitable IoT system security model for assuring best practices for automated transportation of goods.
- Design the layer-wise security technologies using layer model of IoT system for healthcare monitoring.
- Design an IoT-based network considering both the CIA principles of security and also IoT-specific security measures for air quality monitoring considering the parameters such as humidity, temperature, particulate matter, and ozone.

15.7 BEST PRACTICES FOR SECURING IOT DEVICES

Some best practices for securing IoT systems are listed as follows.

15.7.1 Tamper-resistant Hardware

The IoT devices should be kept in isolation and only designated persons should get the physical access of the IoT devices, especially for the devices, which are unattended continuously. This provides safety to such devices and makes them tamper resistant.

Physical endpoint security can be ensured by camera covers, small plastic devices, port locks to provide secure cover to webcams, USB, and Ethernet ports. Implementing strong boot password for hardware/software is one of the best practices for device security. TCP/UDP ports, places for code injections such as web servers, communications without encryption, open serial ports, and open password prompts are the open vulnerabilities susceptible for attacks very often. Such known vulnerabilities should be protected.

15.7.2 Firmware Updates/Patch Updates

Even after taking care of known vulnerabilities to provide security measures, some vulnerabilities still remain, which can be exploited by attackers. Hence, firmware-upgrading patches are needed to be developed from time to time to upgrade the firmware to remove the loop holes. This maintenance service enables the IoT systems to be more efficient and reliable.

At present, most of the IoT devices are not developed to support the patch updates by OTA (Over the Air). Hence, most of the IoT devices are vulnerable to attacks. There is a need by manufacturers and vendors to supply the information of potential risks, and details about patches and upgrades and information, related to policy and functionality in a transparent way to the user of the IoT system.

15.7.3 Dynamic Testing

For assigning minimum baseline security, testing of IoT system devices is essential. Static testing is not useful to find vulnerabilities in the off-the-shelves products. Dynamic testing is useful for finding vulnerabilities in the code as well as weakness in the hardware, which is not visible with static testing. For checking the hardware and software items, used in the IoT systems, manufacturers should perform dynamic testing.

15.7.4 Strong Authentication Technique Practices

Strong username and passwords credentials should be used by IoT devices for authentication purpose. Default credentials that are generally used by multiple devices are big threats as attackers may try these common credentials for unauthorized access. Unique username/password should be maintained by each IoT device and the password should follow the strong password rules so as to deter the attacker from guessing the password.

Even the IoT system is recommended for opting, two-factor authentication where one more level of security is provided by another authentication technique such as OTP code through SMS is employed.

For IoT applications, even adaptive authentication such as context-aware authentication (CAA) is advised in which a background machine-learning algorithm is constantly checking the contextual information, to avoid illegitimate use of IoT devices.

15.7.5 Use of Secure Protocols and Encryption

Though strong authentication is provided for access control through password, still IoT systems can be hacked. In the heterogeneous environment of the IoT systems, many protocols are used such as Zigbee, Z-Wave, 6LoWPAN, NFC, Wi-Fi, Cellular, Neul, Sigfox, and Bluetooth. Depending upon the kind of protocol and the device computation capability, the use of encryption technique is enforced before data transmission. Hence, strength of encrypted data to resist the attacker is doubtful.

15.7.6 Network Division into Segments

Division of a larger network into domain specific smaller networks using VLANs, IP address ranges, or a combination of them should be promoted. Such division of the network creates different security zones with different security policies for each zone. This can be established based on the kind of application, information, and threat level, resulting in preventing internal and external unauthorized access.

15.7.7 Sensitive Information Protection

IoT smart devices provide services that are discoverable by other devices. In addition, most of the protocols leak personally identifiable information (PII). This information can be used by other information sources for targeting some attacks.

15.7.8 Encouraging Ethical Hacking and Discouraging Safe Harbour for Unethical Practices

Legislation should be useful for promoting safe IoT ecosystems for everyone. The offenders should be punished by the law, and strongly discourage safe harbours for harmful activities. Provisions in legislation should be made for ethical hacking. Some check and bounds for manufacturers should be in place, which will deter manufacturing of harmful and insecure products for some financial gain.

15.7.9 Need for IoT Security and Privacy Certification Board

There is need for professional bodies/organisations to proactively develop and provide professional certification programs for designers, manufacturers, and providers of IoT technologies for guiding in the creation of new devices. There should be some oversight to verify whether recommended engineering practices are adhered to while providing the new products.

15.8 IOT SECURITY WITH BEST PRACTICES FOR HOME AUTOMATION APPLICATION

Following is a list of best practices tailored for a smart home application (see Fig. 15.10):

- Tamper resistant hardware by providing proper casing, covers for webcams, Ethernet port locks.
- Wi-Fi Internet connectivity for firmware updates for the smart objects such as smart door lock, fridge, and air conditioners.

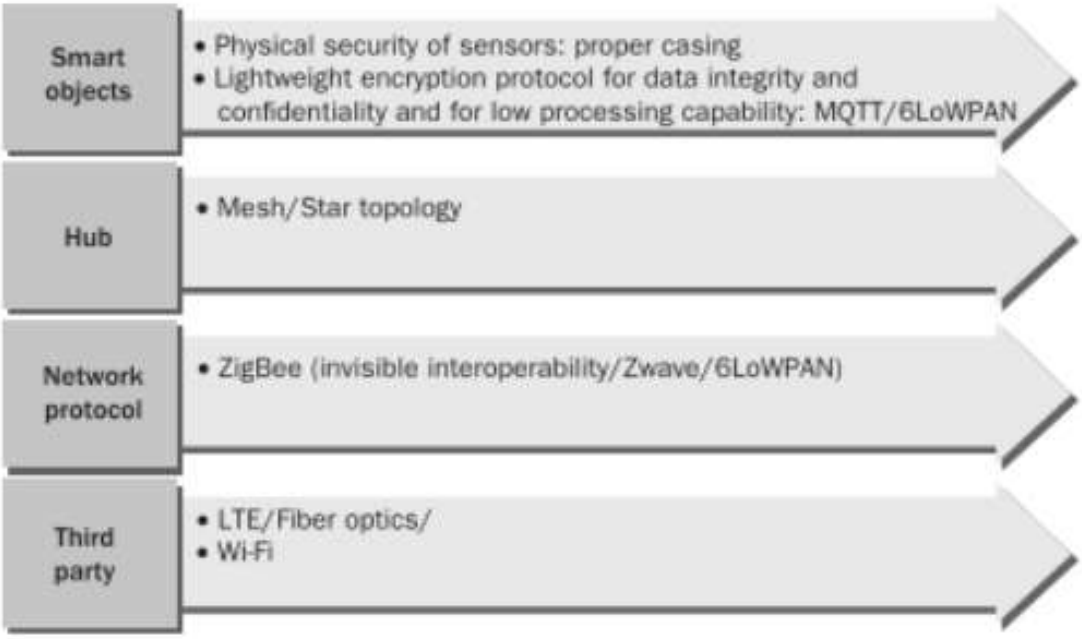


Fig. 15.10 Secured Smart Home Management

- Dynamic testing before adding new smart object for checking interoperability, data privacy, and access control.
- Secure network protocols selection for data communication to third party and for data encryption.
- Smart objects maintenance contract with the IoT system provider for regular maintenance to remove security flaws.
- Two level strong authentication for access control.
- Network segmentation to isolate private smart home network from third party to provide access control through isolation.
- Secure data disposal from server when the smart object is discarded or changed.
- Manufacturer should provide security certification for ensuring best practices implementation while designing manufacturing and deploying the smart home system for privacy preservation and trust building.

SUMMARY

In the Internet of Things, every physical object has a virtual component, which produces and/or consumes the service. Novel approaches must be used to provide secure and trustable IoT systems, which can be easily adopted by consumers for the safe and ethical use. The manufacturers gain reputation by providing such trustworthy IoT devices.

For strengthening the security, strong security enforcement is needed at in-house, device-by-device, and for the complete lifecycle of every product. Different security threats, issues, and attacks in different layers of IoT system architecture are to be considered while designing, developing, and deploying the security controls to IoT devices to create an innovative, reliable, and user-friendly environment which will enforce the fundamental tenets of security.

KEYWORDS

IoT system security architecture, IoT system layers, Security issues in IoT system, Vulnerabilities, Security threats, Attacks, Security controls, WSN security, Best practices

REVIEW QUESTIONS

1. Design a IoT-driven network for monitoring atmospheric conditions such as temperature, humidity, solar index, and soil parameters such as moisture content to analyse the need for watering the plants of a garden with sprinkle irrigation.
2. If a plain text message character is mapped to Elliptic Curve Point is $M(6, 1)$, for the elliptical curve defined by equation $y^2 = x^3 + 2x + 4$. Find the public and private key using public point $P(2, 3)$ and encrypt the message to find ciphertext ($C1$, $C2$) and decrypt the ciphertext using the keys to recover the message point M, using ECC with elliptical curve of the field $F_p = F_7$ for prime number $p = 7$.
   Given: $d = 199$ for prime order
   $n = 201$, $k = 10$
3. What are the advantages of elliptical curve cryptosystem? Explain with reference to some IoT system considering the resource constraints of that system.
4. Select a suitable application layer protocol for the smart energy grid IoT system to protect CIA principles of security. Justify your selection.