ABOUT THE BOOK

Smart World with IoT: Fundamentals, Architecture and Practical Implementation offers a comprehensive journey into the connected world of the Internet of Things. As IoT continues to transform industries, homes, cities, and personal lifestyles, this book equips readers with the knowledge and skills needed to understand, design, and deploy intelligent IoT solutions.

Beginning with the foundations—what IoT is, how it evolved, and how it works—the book guides readers through essential concepts such as IoT architecture, communication models, embedded systems, protocols, and cloud-based integration. It simplifies complex technologies including M2M communication, SDN, NFV, IPv6, device management, analytics, and modern IoT platforms.

Hands-on explanations of physical devices like Arduino and Raspberry Pi, together with real-world case studies in smart homes, smart cities, and environmental monitoring, make this book both practical and insightful. Readers gain exposure to Python for IoT programming, cloud services such as Amazon EC2, data analytics, and system design approaches used in today's smart applications.

Whether you are a student, educator, engineer, or tech enthusiast, this book provides the clarity and depth needed to confidently navigate the evolving IoT landscape. Packed with concepts, illustrations, and step-by-step guidance, it is your essential companion to building a smart, connected world driven by data, intelligence, and innovation.

PhytoElectron TECHNOLOGIES
phytoelectrontechnologies@gmail.com

ISBN: 978-81-988757-9-2
9 788198 875792
INR 699.00

SMART WORLD WITH IOT: FUNDAMENTALS, ARCHITECTURE AND PRACTICAL IMPLEMENTATION

PhytoElectron TECHNOLOGIES

SMART WORLD WITH IOT

FUNDAMENTALS, ARCHITECTURE AND PRACTICAL IMPLEMENTATION

Dr. P. Brundavani | Dr. Shaik Rahamtula | Dr. T. Jaya | Dr. D. Vishnu Vardhan

Smart World with IoT:

Fundamentals, Architecture and Practical Implementation

Smart World with IoT:

Fundamentals, Architecture and Practical Implementation

Authors

Dr. P. Brundavani
Associate Professor
Department of ECE, Ramireddy Subbarami Reddy Engineering College, Kavali, SPSR Nellore.

Dr. Shaik Rahamtula
Associate Professor
Department of ECE, Ramireddy Subbarami Reddy Engineering College, Kadanuthala, Andhra Pradesh, India.

Dr.T. Jaya
Professor
Department of ECE, Vels Institute of Science, Technology & Advanced Studies (VISTAS), Chennai,

Dr. D. Vishnu Vardhan
Principal
JNTUACEA, Pulivendula.

PhytoElectron Technologies
INDIA

Book Title	:	Smart World with IoT: Fundamentals, Architecture and Practical Implementation
Authors	:	**1. Dr. P. Brundavani** **2. Dr. Shaik Rahamtula** **3. Dr. T. Jaya** **4. Dr. D. Vishnu Vardhan**
Book Subject	:	Introduction to Internet of Things
Book Category	:	Authors Volume
Copy Right	:	@ Authors
Second Edition	:	NOV, 2025
Book Size	:	B5
Price	:	Rs.699/-

Published by
PhytoElectron Technologies
INDIA

ISBN Supported by
Raja Ram Mohan Roy National Agency for ISBN
Government of India, Ministry of Human Resource Development,
Department of Higher Education, New Delhi – 110066 (India).

ISBN: 978-81-988757-4-7

9 788198 875747

ACKNOWLEDGMENTS

The author gratefully acknowledges the support and guidance received from all individuals and institutions who contributed to the successful preparation of this textbook.

I express my sincere thanks to the management, principal, and the Department of Electronics and Communication Engineering for providing the academic support, resources, and encouragement needed during the development of this material.

My heartfelt appreciation is extended to all faculty members and subject experts whose valuable insights, review comments, and constructive suggestions significantly enhanced the quality and clarity of the content.

I would also like to thank my colleagues and students for their continuous motivation, feedback, and cooperation throughout the writing process.

Finally, I express deep gratitude to my family for their patience, inspiration, and unwavering support, which enabled me to complete this textbook with dedication and commitment.

— The Authors

FOREWORD

The rapid growth of the Internet of Things (IoT) has transformed the technological landscape, reshaping the way we live, work, and interact with the world around us. Today, billions of interconnected devices collect, communicate, and process data, creating intelligent environments that span across industries such as healthcare, transportation, agriculture, manufacturing, and smart cities. As IoT continues to evolve, the need for a strong foundational understanding becomes essential for students, researchers, academicians, and professionals.

This textbook, Introduction to IoT, offers a comprehensive and structured presentation of IoT concepts, architectures, protocols, embedded systems, data analytics, cloud integration, and real-world applications. It meticulously covers fundamental theories while blending them with practical insights, diagrams, and examples that enhance clarity and understanding. The content is aligned with the academic requirements of undergraduate engineering courses and skill-oriented programs in emerging technologies.

The author's effort in providing a clear, concise, and student-friendly approach is highly commendable. Each chapter is thoughtfully crafted to build conceptual depth while remaining accessible to learners at all levels. The inclusion of M2M communication, IoT protocols, Python programming essentials, analytics, and cloud-based solutions makes this book a valuable resource for academic study as well as self-learning.

I am confident that this textbook will serve as an excellent guide for students and educators, and will inspire further exploration into the dynamic world of IoT. It is my pleasure to recommend this book to all aspiring learners who seek to gain strong foundational knowledge in the Internet of Things.

— The Authors

CHAPTER - 1

INTRODUCTION TO INTERNET OF THINGS

1.1 HISTORY OF IOT

The Internet of Things (IoT) is a network of physical objects or individuals referred to as "things" that are embedded with software, electronics, a network, and sensors that allow these objects to capture and share data. The aim of the Internet of Things is to expand internet access from standard devices such as computers, smartphones, and tablets to comparatively simple devices such as toasters.

Virtually everything becomes "smart" as a result of the Internet of Things, which uses the capacity of data collection, AI algorithms, and networks to improve facets of our lives. A human with a diabetes sensor implant, an animal with tracking devices, and so on are examples of things in the Internet of Things.

This IoT guide for beginners discusses all of the fundamentals of the Internet of Things. Internet of things is shown in Figure 1.1

Figure 1.1: IoT

- 1970- The concept of wired devices was proposed; 1990- John Romkey invented a toaster that could be switched on/off through the Internet; and 1995- Siemens launched the first cellular module designed for M2M.

- 1999- Kevin Ashton coined the phrase "Internet of Things" while working at P&G, and it quickly became famous.
- 2004 - The word was listed in well-known publications such as the Guardian, Boston Globe, and Scientific American.
- In 2005, the International Telecommunications Union (ITU) issued its first study on the subject.
- The Internet of Things was born in 2008.
- Gartner, a consumer intelligence firm, includes "Internet of Things" technologies in its research in 2011. IoT works is shown in Figure 1.2

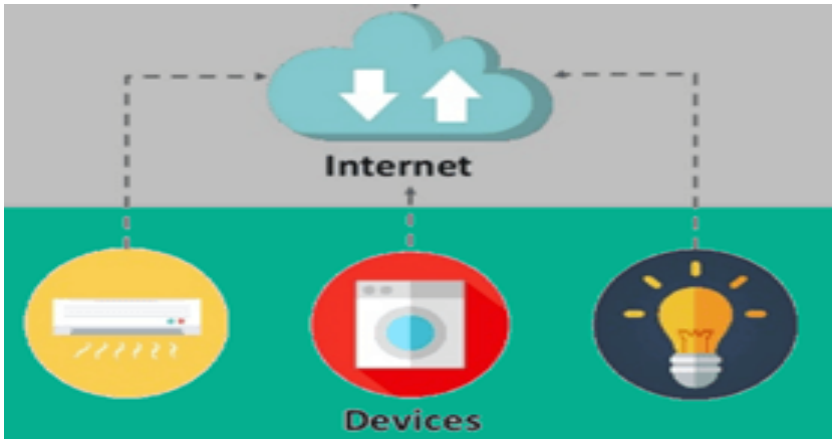

Figure 1.2: IoT Works

1.2 IOT WORKFLOW

The IoT phase begins with the devices themselves, such as smartphones, smartwatches, and electronic appliances such as TVs and washing machines, which enable you to connect with the IoT network four basic components of an IoT system:

1. **Sensors and Devices**: Sensors or devices are an important aspect that allows you to capture real-time data from your surroundings. All of this data can be of varying degrees of complexity. It could be a basic

temperature sensingsensor, or it could be a video stream.

A device can include a variety of sensors that performfunctions other than sensing. A cell phone, for example, has many sensors such as GPS and a camera, but your smartphonecannot detect these things.

2. **Connectivity**: All data gathered is sent to a cloud infrastructure. The sensors should be linked to the cloud through numerous communication channels. Cell or satellite networks, Bluetooth, WI-FI, WAN, and other networking mediums are examples of these.

3. **Data Processing**: Once the data is processed and sent tothe cloud, the software processes it. This method can be as simple as measuring the temperature or reading on devices such as air conditioners or heaters. However, it can also be very nuanced, such as detecting objects on film through computer vision.

4. **User Interface**: The information must be made accessible to the end-user in some manner, which can be done by disabling alerts on their phones or giving them updates by email or text message. The consumer can need an app that actively tests their IoT device from time to time. For instance, the user can have a camera mounted in his house. He needs to use a database server to gain access to video recordings and all streams.

However, contact is not necessarily one-way. Depending on the IoT program and device complexity, the user will also be able to execute an operation that may result in cascading results.

For eg, if a user senses any changes in the temperature of the refrigerator, the user should be able to alter the temperature with their cell phone thanks to IoT technology.

1.3 CHALLENGES OF IOT

At the moment, IoT faces several challenges, including:

- Insufficient monitoring and updating
- Concerns over data storage and privacy
- The difficulty of software
- Volumes of data and their interpretation
- AI and automation integration
- Devices necessitate a continuous power source, which isimpossible to obtain.
- Interaction and close contact

1.4 CHARACTERISTICS OF IOT

There are seven important IoT characteristics:

1. **Interconnectivity**: This doesn't need any clarification.There must be a link between different layers of everything going on in IoT devices and computers, with sensors and otherelectronics and related hardware and control systems.

2. **Particulars:** All that can be tagged or linked in the way that it is intended to be connected. Sensors, kitchen equipment, and tagged livestock are only a few examples. Sensors may be built into devices, or sensing materials may be added to devices and objects.

3. **Details:** Data is the glue that holds the Internet of Thingstogether, and it is the first step toward action and intelligence.

4. **Communicate:** Devices are linked so that they can exchange data, which can then be analyzed. Communication may take place over short distances or over medium to very long distances. Wi-Fi, LPWA network technology such as LoRa or NB-IoT are examples.

5. **Intelligence**: The intelligence factor, such as sensing capabilities in IoT devices and intelligence derived from Introduction to IOT for

Beginners (also artificial intelligence).

6. **Take action:** The product of intelligence. This can bemanual action, action dependent on disputes about phenomena (for example, in smart factory decisions), or automation, which is often the most critical piece. IoT Characteristics are shown in Figure 1.3

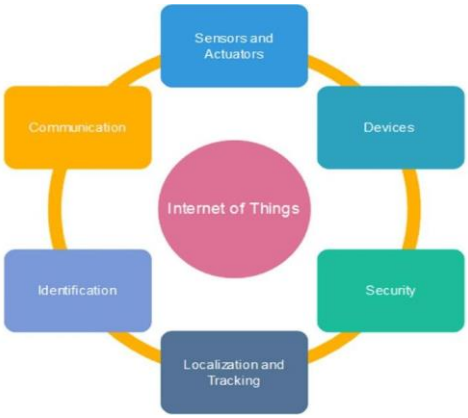

Figure 1.3: IoT Characteristics

7. **The ecosystem:** The Internet of Things' position in relation to other technology, cultures, priorities, and the overall image into which the Internet of Things fits. The Internet of Everything dimension, the network dimension, and the need for strong relationships are all important considerations.

1.5 IOT LIFE CYCLE

IoT development has a very straightforward lifecycle. Deployment is followed by monitoring, servicing, and managing, which is then followed by routine updates and decommissioning.

Iot Product Lifecycle is Described in Figure 1.4

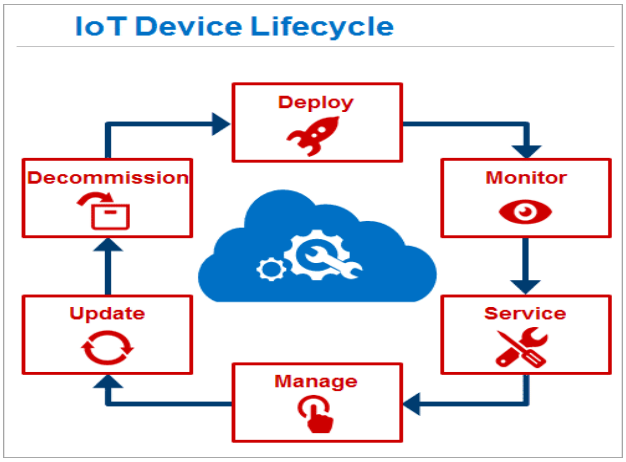

Figure 1.4: IoT Life cycle

Aside from these facts, there are certain benefits and drawbacks of Internet of Things devices that can have a significant effect on the present and future generations of humans.

1.6 PHYSICAL DESIGN OF IOT

1.6.1 THINGS

Things are IoT Devices with special personalities that can do remote sensing, actuation, and monitoring. Things are the most critical component of an IoT program. Sensing Devices, Smart Watches, Smart Electronics Appliances, Wearable Sensors, Automobiles, and Automotive Robots are all examples of IoT Devices. These devices collect data in various ways, which when analyzed by data analytics tools yields valuable information to direct subsequent activities locally or remotely.

Temperature data produced by a Temperature Sensor in the home or other location, for example, when analyzed, may aid in determining temperature and taking action based on user input. The diagram above represents a typical block diagram of an IoT device. It can provide multiple interfaces forconnecting to other devices. Sensors have I/O interfaces, as do Internet connectivity, storage, and audio/video. IoT devices gather data from on-

board or connected sensors, and the data is transmitted to another device or a cloud-based server. There are numerous cloud servers available today, especially for IoT systems. These platforms are known as IoT Platforms. Really, these clouds are specifically designed for IoT purposes. But we can quickly review and process data here.

For example, suppose a relay switch attached to an IoT device will turn on/off an appliance based on commands sent to the IoT device over the Internet. Block Diagram of Iot is Shown in Figure 1.5

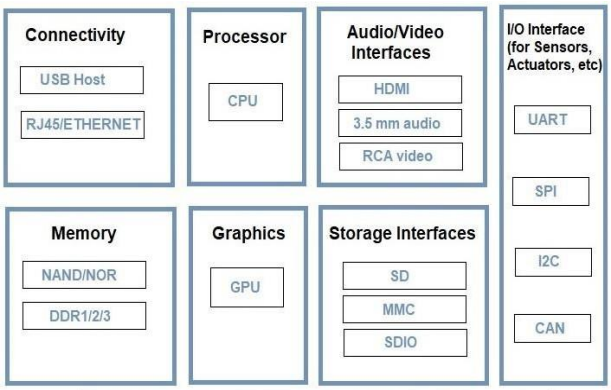

Figure 1.5: Block Diagram of IoT

Temperature data produced by a Temperature Sensor in the home or other location, for example, when analyzed, may aid in determining temperature and taking action based on user input. The diagram above depicts a typical block diagram of an IoT device. It can provide multiple interfaces for connecting to other devices. Sensors have I/O interfaces, as do Internet connectivity, storage, and audio/video. IoT devices gather data from on-board or connected sensors, and the data is transmitted to another device or a cloud-based server. There are various cloud servers available today, especially for IoT systems. These platforms are known as IoT Platforms. Really, these clouds are specifically designed for IoT purposes. But we can quickly analyze and process data here. For example, suppose a relay switch attached to an IoT device will turn on/off an appliance based on commands sent to the IoT device over the Internet.

1.6.2 IOT PROTOCOLS

IoT protocols aid in the establishment of Internet-based communication between IoT devices (Node Devices) and cloud-based servers. It aids in sending commands to IoT devices and receiving data from IoT devices through the Internet. An illustration is shown below. You can tell which protocols were used by looking at this graphic.

1.6.2.1 LINK LAYER

Link layer protocols govern how data is physically transmitted through the network's physical layer or medium (Coaxial cable or other or radio wave). This Layer specifies howpackets are encoded and signaled by the hardware device overthe medium to which the host is connected (eg. coaxial cable).

802.3 – Ethernet: Ethernet is a collection of technologies and protocols that are mainly found in local area networks

(LANs). IEEE 802.3 specification was the first to standardize it in the 1980s. IEEE 802.3 specifies the physical layer as well as the medium access control (MAC) sub-layer of the data link layer for wired Ethernet networks. Ethernet is divided into two types: traditional Ethernet and switched Ethernet.

Ethernet is a collection of applications and protocols that are mainly used in local area networks (LANs). IEEE 802.3 standard was the first to standardize it in the 1980s. IEEE 802.3specifies the physical layer as well as the medium access control (MAC) sub-layer of the data link layer for wired Ethernet networks. Ethernet is divided into two types: traditional Ethernet and switched Ethernet.

Classic Ethernet is the original version of Ethernet, with data speeds ranging from 3 to 10 Mbps. The variants are widely known as 10BASE-X. In this case, 10 denotes the highest throughput, i.e. 10 Mbps, BASE denotes the use ofbaseband communication, and X denotes the medium form. In today's communication setting, most types of traditional Ethernet have become outdated.

Switches are used to link LAN stations in a switched Ethernet network. It replaces the traditional Ethernet repeaters and allows for maximum bandwidth use.

IEEE 802.3 Popular Versions

There are a number of versions of IEEE 802.3 protocol. Themost popular ones are -

- IEEE 802.3: This was the initial 10BASE-5 standard. It made useof a dense single coaxial cable through which a link could be tapped by drilling into the cable to the heart. In this case, 10 represents the highest throughput, i.e. 10 Mbps, BASE represents the use of baseband transmission, and 5 represents the maximum segment length of 500m. layers of IoT is shown in Figure 1.6

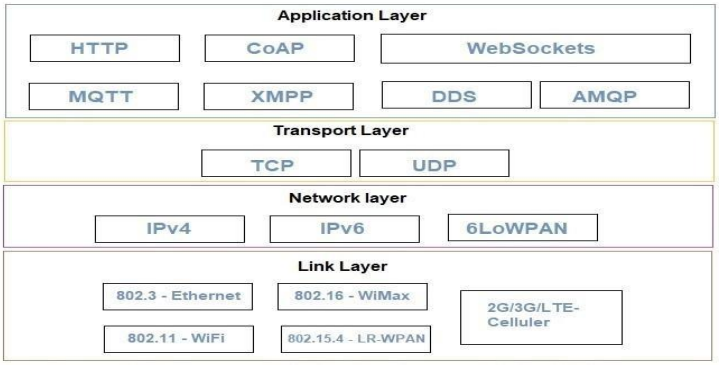

Figure 1.6: Layers of IOT

- IEEE 802.3a: This provided the standard for thin coax (10BASE-2), which is a thinner version in which coaxial cable parts are joined by BNC connectors. The number 2 corresponds to the overall section length of roughly 200m (185m to be precise).
- IEEE 802.3i: This standard established the twisted pair (10BASE- T) standard, which employs unshielded twisted pair (UTP) copper wires as the physical layer medium. IEEE 802.3u included additional variants for 100BASE-TX, 100BASE-T4, and 100BASE-FX.
- IEEE 802.3i: This is the standard for Ethernet over Fiber (10BASE-F),

which uses fiber optic cables as the communication medium. IEEE 802.3 Popular Versions are shown in Figure 1.7

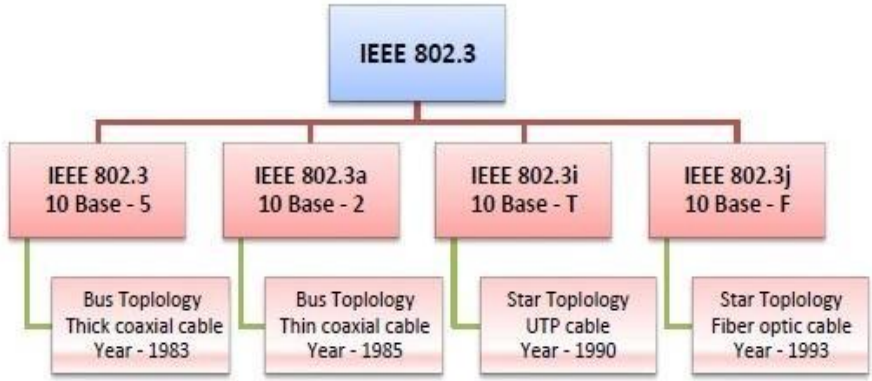

Figure 1.7: IEEE 802.3 Popular Versions

Classic Ethernet Frame Format and IEEE 802.3

The primary fields of a traditional Ethernet frame are as follows:

- Preamble: This is the initial area that contains the alert andtiming pulse for transmission. In the case of traditional Ethernet, it is an 8-byte field, while in the case of IEEE 802.3, it is a 7-byte field.
- Frame Delimiter Start: It is a one-byte field in an IEEE 802.3frame that involves an alternating pattern of ones and zeros ending with two ones.
- Destination Address: This is a 6-byte sector that contains the physical addresses of destination stations.
- Source Address: This is a 6-byte field that includes thesending station's physical address.
- Length: This is a 7-byte field that keeps track of the number of bytes in the data field.
- Data: This variable-sized area contains data from the upperlayers. The actual data field size is 1500 bytes.
- Padding: This is applied to the data to make it the minimum length of 46 bytes.
- CRC: CRC is an abbreviation for cyclic redundancy search. It includes details about error detection. The classis Ethernet Frame format is shown in Figure 1.8

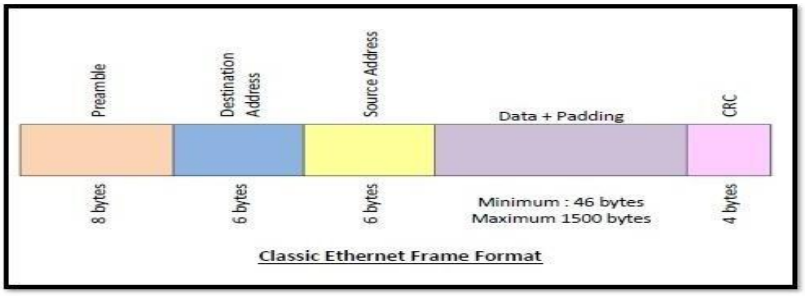

Figure 1.8: Classic Ethernet Frame Format

IEEE 802.11 – Wi-Fi: Part of the IEEE 802 family of LAN protocols, IEEE 802.11 defines a set of media access control (MAC) and physical layer (PHY) protocols for integrating wireless local area network (WLAN) Wi-Fi device communication at different frequencies, including but not limited to 2.4 GHz, 5 GHz, and 60 GHz. IEEE 802.3 Frame Format

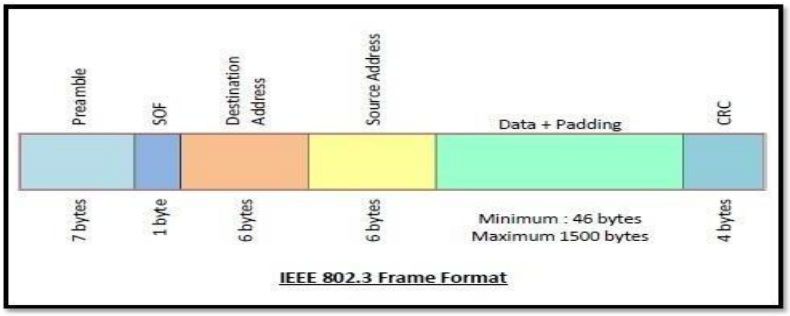

Figure1.9: IEEE 802.3 Frame Format

IEEE 802.11 is a series of medium access control (MAC) and physical layer (PHY) protocols for incorporating wireless local area network (WLAN) device communication that is part of the IEEE 802 set of local area network (LAN) technical standards. The standard and its revisions serve as the foundation for wireless network devices bearing the Wi-Fi name and are the most commonly used wireless computer networking protocols in the world. IEEE 802.11 is used in most home and office networks to allow computers, printers, smartphones, and other devices to communicate with one another and link to the Internet without the use of wires.

The LAN/MAN Standards Committee of the Institute of Electrical and Electronics Engineers (IEEE) develops and maintains the standards (IEEE 802). The standard's base version was published in 1997, and it has since been amended. Although each amendment is legally repealed when it is introduced into the most recent edition of the standard, the business community continues to market to amendments because they concisely denote the capabilities oftheir goods. As a result, each revision continues to become its own standard in the marketplace.

IEEE 802.11 operates at a variety of frequencies, including,but not limited to, 2.4 GHz, 5 GHz, 6 GHz, and 60 GHz. While the IEEE 802.11 requirements list potential networks, the radio frequency spectrum availability allowed differs substantially by regulatory domain.

The protocols, which are usually used in combination withIEEE 802.2, are intended to interoperate easily with Ethernet and are often used to carry Internet Protocol traffic. 802.16 – Wi-Max : The WiMAX infrastructure standard is astandard for Wireless Metropolitan Area Networks (WMANs) established by IEEE 802 working group number 16, whichspecializes in point-to-multipoint broadband wirelessnetworking. WiMAX is a wireless broadband networking technologythat uses the IEE 802.16 standard to provide high-speed data over a wide region.

WiMAX stands for Worldwide Interoperability for Microwave Access (AXess), and it is a point-to-multipoint wireless networking technology. WiMAX technology may address the demands of a wide range of consumers, from those in developing countries who wish to build a modern high-speed data network quite cheaply without the expense and time taken to install a wired network, to those in remote areas who need quick connectivitywhere wired options might not be possible due to distances and costs involved - simply delivering WiMAX broadband. Itis also used by smartphone apps, which provide high-speed data to consumers on the go.

The fundamentals of WiMAX technology

The WiMAX technology standard is a wireless metropolitan area network (WMAN) standard established by IEEE 802 working group number 16, which specializes in point-to-multipoint broadband wireless connectivity. Initially, 802.16a was developed and published, but it has since been refined. The 802.16d standard, also known as the 802.16-2004 standard, was introduced as a streamlined version of the 802.16a standard targeted at fixed implementations. Another variant of the standard, 802.16e or 802.16-2005, targeted at the roaming and smartphone markets, was also announced.

WiMAX broadband infrastructure relies on many main innovations to provide high-speed data rates:

- OFDM (Orthogonal Frequency Division Multiplex): OFDM has been integrated into WiMAX technology to enable it to provide high-speed data without the selective fading and other problems that other signal formats provide.

- MIMO (Multiple Input Multiple Output): WiMAX technology employs MIMO for multipath propagation. MIMO allows for lower signal intensity levels or higher data speeds by using the various signal paths that exist.

The WiMAX Forum is a wireless industry consortium composed of a rising number of industry leaders. It was founded to promote and grow WiMAX technology globally, as well as to bring universal standards around the globe to allow the technology to become a globally established technology. One of the forum's objectives is to facilitate the implementation of a standard that will allow complete interoperability between goods. The WiMAX Forum seeks to avoid the issues associated with previous wireless standards' low interoperability, as well as the effect this has on adoption. Eventually, vendors will be able to have their products approved under the Forum's auspices and then market their products as "Forum Certified." Despite the fact that WiMAX technology can accept traffic dependent on communication technologies such as Ethernet, Internet Protocol (IP), and Asynchronous Transfer Mode

(ATM), the Forum will only validate the IP-related elements of 802.16 items. The focus is on IP operations since this is still the most commonly used protocol.

Evolution of WiMAX

The origins of WiMAX can be traced back to the 1990s, when it was realized that there would be a major rise in data traffic over telecommunications networks. Since wired telecommunications networks are very costly, especially in rural areas, and are not widely available in many countries, methods of delivering wireless broadband were investigated. WiMAX history began with these inquiries into what was known as "last mile connectivity" - means of providing high- speed data to a vast number of people who do not have a wired link.

The promise of low-cost last-mile connectivity, as well as a technology capable of handling backhaul over a cellular link, proved to be a convincing case for developing a modern wireless data link system.The IEEE's development of standards was the next big milestone in WiMAX history. The IEEE 802 LAN/MAN Standards Committee formed the 802.16 standards working group in 1999. In December 2001, the first 802.16 standard was accepted, and it was followed by two revisions to the original 802.16 standard. These revisions, defined as 802.16a and 802.16c, discussed radio spectrum and interoperability problems. A new revision project was launched in September 2003. The aim of this was to match the standard with the European

ETSI HIPERMAN standard. It was also planned to have conformance evaluation requirements as part of the overall standard. The project was finished in 2004, and the standard was introduced as 802.16d, but due to the implementation date, it is sometimes referred to as 802.16-2004. The previous 802.16 manuals, including the a, b, and c amendments, were discontinued with the introduction of the 802.16-2004 standard.

WiMAX variants

New WiMAX applications have been developed since its introduction, and as a result, there are two "flavors" of WiMAX technology available:

The two flavors of WiMAX broadband technologies are used for various implementations, and although they are based on the same standard, each implementation has been optimized for its specific use. 802.16d - DSL substitute The 802.16d version is also known as 802.16-2004, and it is closer to the original version of WiMAX described by 802.16a. It is targeted at fixed applications and provides a wireless counterpart to DSL broadband data, which is commonly referred to as WiMAX broadband. In reality, the WiMAX Forum defines the technology as "a standards-based technology that enables the distribution of last mile wireless broadband service as an alternative to cable and DSL." Because 802.16d can have data speeds of up to 75 Mbps, it is suitable for fixed, DSL replacement applications such as WiMAX broadband. It can also be used for backhaul, with the final data being transmitted to individual users. Cell radii can range up to 75 kilometers.

802.16e - Mobile / Nomadic Although 802.16 / WiMAX was initially intended to be a fixed-only technology, the demand for people on the move who needed high-speed data at a lower cost than wireless networks created a need for a mobile version, and 802.16e was established. This standard is often often referred to as 802.16-2005. It currently allows users to bind to a WiMAX cell from a number of locations, with potential updates to include cell handover. Data speeds of up to 15 Mbps are possible with 802.16e, and cell radius lengths are usually between 2 and 4 km.

Competition

There are some implementations of the IEEE 802.3 protocol. The most well-known are - The rivalry with WiMAX and 802.16 is dependent on the form or variant used. While it was originally believed that WiMAX will be a major competitor to Wi-Fi, there are other places where WiMAX is a challenge.

- Cable DSL bars WiMAX is capable of providing customers with high-speed data connections, and as such, it can pose a challenge to DSL cable operators.

- Cell phone service providers Cell phone providers viewed the smartphone version of WiMAX as a potential challenge as LTE was being established and the first roll-outs began. It was also proposed as the IMT 4G standard, but LTE was chosen as the standard, leaving WiMAX for fixed WiMAX broadband, last mile connections, and a host of other point-to-point applications. WiMAX technology has been used in a variety of applications. Although it was originally seen as a contender for 4G, its use is dwindling, though it is still used for WiMAX broadband and last mile connections.

802.15.4 -LR-WPAN: A collection of Low-rate wireless personal area network standards. The IEEE 802.15.4 standard specifies the MAC and PHY layer used for networking protocols such as Zigbee®, 6LoWPAN, Thread, WiSUN, and MiWiTM, among others. The standards allow low-cost, low- speed communication for devices with limitedpower.2G/3G/4G- Cell Communication: There are separate generations of telecommunications. These specifications allow IoT devices to connect over cellular networks.

1.6.2.2 NETWORK LAYER

In charge of transmitting IP datagrams from the source network to the destination network. The Network layer handles host addressing and packet routing. For Host recognition, we used IPv4 and IPv6. IPv4 and IPv6 are also hierarchical IP address allocation systems.

IPv4: Internet Protocol Version 4.

An Internet Protocol address (IP address) is a numerical mark assigned to every device attached to a computer network that communicates using the Internet Protocol. An IP address performs two primary functions: defining a host or network interface and addressing a particular location. An IP address is specified as a 32-bit number in Internet Protocol version 4

(IPv4). However, due to the proliferation of the Internet and the shortage of valid IPv4 addresses, a new variant of IP (IPv6) was standardized in 1998, using 128 bits for the IP address. Since the mid-2000s, IPv6 implementation has been continuing.

[1] Formalized paraphrase An IP address performs two primary functions: defining a host or network interface and addressing a particular location. An IP address is specified as a 32-bit number in Internet Protocol version 4 (IPv4).

[2] Formalized paraphrase However, due to the proliferation of the Internet and the shortage of valid IPv4 addresses, a new variant of IP (IPv6) was standardized in 1998, using 128 bits for the IP address.

[3] Formal paraphrase Since the mid-2000s, IPv6 implementation has been continuing.

In IPv4, IP addresses are written and displayed in human- readable notation, such as 172.16.254.1, and in IPv6, 2001:db8:0:1234:0:567:8:1. In CIDR notation, the size of the routing prefix of the address is specified by suffixing the address with the number of significant bits, for example, 192.168.1.15/24, which is equal to the traditionally used subnet mask 255.255.255.0.

The Internet Assigned Numbers Authority (IANA) and five regional Internet registries (RIRs) are responsible in their designated regions for assigning IP addresses to local Internet registries, such as Internet service providers (ISPs) and other end users. IANA allocated IPv4 addresses to RIRs in blocks of approximately 16.8 million addresses each, but these have been depleted at the IANA stage since 2011. Just one of the RIRs has a supply of local assignments in Africa left.

[4] Formalized paraphrase Some IPv4 addresses are reserved for private networks and are thus not internationally exclusive. Each device connecting to a network is given an IP address by network administrators. Based on network policies and software features, such assignments can be static (fixed or permanent) or dynamic. An IPv4 address is 32 bits long,

limiting the address space to 4294967296 (232) addresses. Any of these addresses are reserved for specific uses, such as private networks (18 million) and multicast addressing (270 million).

IPv4 addresses are typically expressed in dot-decimal notation, which consists of four decimal numbers separated by dots ranging from 0 to 255, for example, 172.16.254.1. Each component represents an 8-bit (octet) address group. IPv4 addresses can be provided in different hexadecimal, octal, or binary representations in some specialized writing.

History of subnetting

During the early stages of Internet Protocol development, the network number was always the highest order octet (most significant eight bits). Since this approach only allowed for 256 networks, it quickly proved insufficient when new networks emerged that were independent of the existing networks already designated by a network number. With the implementation of classful network architecture in 1981, the addressing specification was updated. [2] Formalized paraphrase

Classful network architecture allowed more individual network assignments and fine-grained subnetwork design. The class of an IP address was specified as the first three bits of the most important octet. For universal unicast addressing, three classes (A, B, and C) were specified. The network recognition was dependent on octet boundary segments of the entire address, depending on the class. Each class used an increasing number of octets in the network identifier, decreasing the available number of hosts in higher order classes (B and C). The table below provides an outline of this now-defunct scheme.

Classy network architecture served its function during the Internet's early days, but it lacked scalability in the face of the 1990s' exponential proliferation of networking. In 1993, the address space's hierarchy structure was replaced by Classless Inter-Domain Routing (CIDR). To make allocation and routing based on arbitrary-length prefixes, CIDR is based on variable- length subnet masking (VLSM). Today, fragments of

classful network principles serve mainly in a small capacity as the default configuration parameters of certain network software and hardware components (e.g., netmask), as well as in technological terms used in network administrator discussions.

Private addresses

When universal end-to-end connectivity was envisioned for communications with all Internet hosts, IP addresses were meant to be internationally exclusive. However, as private networks grew and public address space needed to be conserved, it was discovered that this was not always possible. Computers that are not wired to the Internet, such as factory machines that communicate only through TCP/IP, do not need globally specific IP addresses. Today, such private networks are common, and they usually link to the Internet through network address translation (NAT) when necessary. Three non-overlapping IPv4 address ranges are reserved for private networks.

[8] Formalized paraphrase Since these addresses are not accessible on the Internet, their use would not require coordination with an IP address registry. Any person is free to use the reserved blocks. A network administrator will typically split a block into subnets; for example, many home routers have a default address range of 192.168.0.0 to 192.168.0.255 (192.168.0.0/24).

Internet Protocol version 6 (IPv6) is the successor to IPv4. The Internet Engineering Task Force (IETF) created IPv6 to solve the long-awaited issue of IPv4 address depletion. IPv6 was designated as a Draft Standard by the Internet Engineering Task Force (IETF) in December 1998, and it was adopted as an Internet Standard on July 14, 2017. IPv6 employs a 128-bit address, allowing for a total of 2128, or roughly 3.41038 addresses.

The address size in IPv6 was expanded from 32 bits in IPv4 to 128 bits, allowing for up to 2128 (approximately 3.4031038) addresses. This is thought to be adequate for the near future. The goal of the new design was not only to have a sufficient number of addresses, but also to reinvent Internet routing by enabling more effective aggregation of subnetwork

routing prefixes. This slowed the development of routing tables in routers. The smallest individual allocation available isa subnet for 264 hosts, which is the square of the whole IPv4 Internet. Real address consumption ratios on any IPv6 network segment would be poor at these stages. The new architecture also allows for the separation of a network segment's addressing infrastructure, i.e. the local managementof the segment's usable capacity, from the addressing prefix used to redirect traffic to and from external networks. If the global connectivity or routing strategy changes, IPv6 has facilities that instantly adjust the routing prefix of whole networks, without requiring internal overhaul or manual renumbering. Because of the vast number of IPv6 addresses available, large blocks may be allocated for specific purposes and aggregated for efficient routing where possible. There is no need for complicated address conservation approaches like CIDR with a huge address space.

IPv6 is natively supported by both existing desktop and commercial server operating systems, but it is not yet commonly deployed in other devices such as home networking routers, voice over IP (VoIP) and multimedia appliances, and some networking hardware.

Private addresses

IPv6 addresses are reserved in the same way as IPv4 addresses are reserved for private networks. In IPv6, these areknown as special local addresses (ULAs). This block has the routing prefix fc00: :/7 reserved for it,[9] and is separated intotwo /8 blocks of separate implicit policies. The addresses contain a 40-bit pseudorandom integer, which reduces thepossibility of address collisions if sites merge or packets are misrouted. For this reason, early practices used a separate block (fec0: :), which was called site-local addresses.

Formalized paraphrase However, the concept of what constituted a location remained vague, and the ill-defined addressing strategy generated routing ambiguities. This address form has been deprecated and cannot be used inmodern systems. Formalized paraphrase.

Link-local addresses, which begin with fe80::, are allocated to interfaces for communication over the attached link. The operating system produces addresses for each network interface automatically. This allows for immediate and automated communication between all IPv6 hosts on a link. This functionality is used in IPv6 network administration's lower layers, such as the Neighbor Discovery Protocol. Private and link-local address prefixes are not permitted to be routed on the public Internet.

IPv6 over Low-Power Wireless Personal Area Networks is abbreviated to 6LoWPAN. 6LoWPAN is the name of a completed IETF working group in the Internet region. This protocol enables even the smallest devices with minimal computing power to transfer data wirelessly using an internet protocol. 6LoWPAN is capable of connecting with 802.15.4 devices as well as other types of devices through an IP network connection, such as WiFi. IPv6 over Low-Power Wireless Personal Area Networks is abbreviated to 6LoWPAN.

6LoWPAN is the name of a completed IETF working group in the Internet region. 6LoWPAN is an abbreviation for the current variant of the Internet Protocol (IPv6) and Low-power Wireless Personal Area Networks (LoWPAN). As a result, 6LoWPAN enables even the smallest devices with insufficient computing power to transfer data wirelessly using an internet protocol. 6LoWPAN is capable of connecting with 802.15.4 devices as well as other types of devices through an IP network connection, such as WiFi. The principle of 6LoWPAN arose from the belief that the Internet Protocol might and should be extended even to the smallest devices, and that low-power devices with minimal computing capacities should be able to engage in the Internet of Things.

The 6LoWPAN group has established encapsulation and header compression mechanisms that allow IPv6 packets to be sent and received over IEEE 802.15.4 networks. IPv4 and IPv6 are the workhorses for data transmission in both local-area networks and wide-area networks like the Internet. Similarly, IEEE 802.15.4 devices provide sensing communication in the wireless domain. The two networks' underlying characteristics, however, vary.

The IPv6 offers a simple transport mechanism to generate complex control systems and communicate with devices in a cost-effective manner through a low-power wireless network, making the 6LoWPAN suitable for home or building automation. The following are the most popular applications for it:

It is possible to achieve distinct advantages over other IoT platforms by connecting smart home devices via IPv6. The Thread initiative was developed to standardize a protocol that runs over 6LoWPAN to allow home automation. And Open Thread makes it easier for developers to get started with Thread and build smart home solutions.

Smart Agriculture: allowing all types of sensors used in agriculture and farming by linking devices that are far apart in remote areas, the possibilities that this protocol provides for creating mesh networks make it just the right one for this kind of use.

Industrial IoT: Automated factories and industrial plants provide a tremendous opportunity for 6LoWPAN, and using automation will result in considerable savings. The potential of 6LoWPAN to bind to the cloud opens up a wide range of possibilities for data monitoring, analysis, and predictive maintenance.

ZigBee

Like 6LoWPAN, ZigBee is intended for low-data-rate and battery-powered applications. ZigBee is currently the most common, low-cost, low-power wireless mesh networking standard on the market, as well as the more advanced technology of the two (ZigBee, 6LoWPAN). It is commonly used in personal or home-area networks, or in wireless mesh networks for networks that run over longer distances.

The ZigBee IP is based on the IEEE 802.15.4 standard, but, unlike 6LoWPAN, it cannot interface with other protocols easily. However, one advantage of ZigBee is that nodes can remain in sleep mode for the majority of the time, significantly extending battery life.

M2M/IoT Implementations for ZigBee

Wireless Light Switches are a kind of switch that allows you to turn on and off the lights Meters for electricity (Smart grid, demand response, etc)

Monitoring of Industrial Equipment

The name is derived from the wiggly dance performed by honey bees on their way to deposit honey. Data in a ZigBee network, like those wild bees, "hops" through a mesh of transceivers before a route to the host (usually the internet) is discovered. It runs at a fixed data rate of 250 kbit/s and is based on the IEEE 802.15 protocol. Because of this speed and poor transmit capacity, ZigBee has a limited range. Repeaters and/or a dense network of nodes are often needed to achieve the desired coverage.

1.6.2.3 TRANSPORT LAYER

This layer handles things like error management, segmentation, flow control, and congestion control. As a result, these layer protocols allow end-to-end message transmission that is independent of the underlying network.

TCP (Transmission Control Protocol): TCP (Transmission Control Protocol) is a network protocol that specifies how to create and manage a network conversation in which application programs can share data. TCP communicates with the Internet Protocol (IP), which determines how computers transmit data packets to one another.

TCP and IP are the fundamental principles that define the Internet. TCP is defined by the Internet Engineering Task Force (IETF) in Request for Comment (RFC) standards document number 793. UDP is a Transport Layer protocol that stands for User Datagram Protocol. UDP is a component of the Internet Protocol suite, also known as the UDP/IP suite. It is an insecure and connectionless protocol, as opposed to TCP. As a result, there is no need to communicate prior to data transmission.

1.6.2.4 APPLICATION LAYER

Application layer protocols describe how systems communicate with lower layer protocols to transfer data over a network. HTTP (Hypertext Transfer Protocol) is an application-layer protocol used to distribute hypermedia documents such as HTML. It was created to facilitate communication betweenweb browsers and web servers, but it can also be used for other purposes. HTTP follows the traditional client-server model, with a client opening a connection to make a request and then waiting for an answer. HTTP is a stateless protocol, which means that the server does not store any data (state) between requests.

RFC 7252 defines CoAP-Constrained Application Protocol as a specialized Internet Application Protocol for constrained devices. It allows devices to connect with one another over the Internet. The protocol is designed specifically for restricted hardware such as 8-bit microcontrollers, low-power sensors, and other devices that cannot run HTTP or TLS. WebSocket Protocol: The WebSocket Protocol allows two-way communication between a client running untrusted code in a managed environment and a remote host who has agreed to receive messages from that code. The origin-based protection paradigm, which is widely used for web browsers, was used for this.

MQTT is a machine-to-machine (M2M)/Internet of Things (IoT) connectivity protocol. It was created as an incredibly lightweight publish/subscribe messaging transport that can beused for communications with remote locations where a minimal code footprint is needed and/or network capacity is limited.

Extensible Messaging and Presence Protocol (XMPP) is an XML-based communication protocol for message-oriented middleware (Extensible Markup Language). It facilitates the sharing of organized and extensible data in near-real time between any two or more network entities.

DDS: The Object Management Group® (OMG®) Data Distribution Service (DDSTM) is a middleware protocol and API standard for data-

centric connectivity. It connects device elements, offering low-latency data connectivity, extreme stability, and a modular infrastructure needed by enterprise and mission-critical Internet of Things (IoT) applications.

AMQP: The AMQP – IoT protocols are made up of a collection of components that route and save messages within a broker carrier, as well as a set of policies for connecting the components. The AMQP protocol allows patron systems to communicate with the dealer and interact with the AMQP model.

1.7 IOT CONCEPTUAL ARCHITECTURE

In this post, we will look at the logical nature of the Internet of Things. The conceptual architecture of an IoT framework refers to an abstract representation of the entities and processes without delving into the low-level implementation requirements. We define the words below to help you visualize the Logical Design of IoT.

- Functional Blocks for IoT
- Models of IoT Communication
- APIs for IoT Communication

1.7.1 INTERNET OF THINGS FUNCTIONAL BLOCKS

An IoT device is made up of a variety of functional blocks that enable the system to perform functions such as recognition, sensing, actuation, communication, and management.

The following are practical blocks:

An IoT system is made up of devices that perform sensing, actuation, monitoring, and control.
Communication: Manages the IoT system's communication.
Resources provide device monitoring services, device regulation services, data publication services, and device exploration services.
Management: This block includes a number of roles for governing the IoT system.

Protection: This block protects the IoT device by providing functions like authentication, authorization, message and information integrity, and data security.

Application: A GUI from which users can access and manage different facets of the IoT framework. Users may also monitor the device status and view or evaluate the generated data using the application. Functional blocks of IoT is shown in Figure 1.10

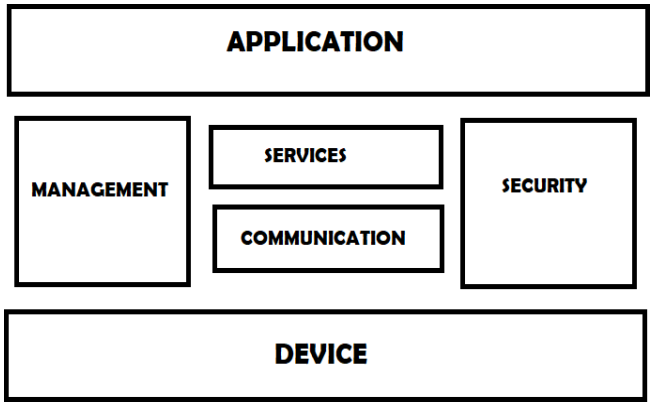

Figure 1.10: IoT Functional Blocks

1.7.2 IOT COMMUNICATION MODELS

Model of Request-Response The request-response model is a communication model in which the client sends requests to the server and the server replies. When a request is sent, the server determines how to respond, retrieves the data, retrieves the resource representation, prepares the response, and then sends the response to the client. Request-response communication is a stateless model in which each request-response pair is independent of the others.

HTTP is a request-response protocol that links a client and a server. The client could be a web browser, and the server could be an application running on a device that hosts a web.

Example: A client (browser) sends an HTTP request to the server, and the server responds with a response. The response includes request status information as well as the requested material.

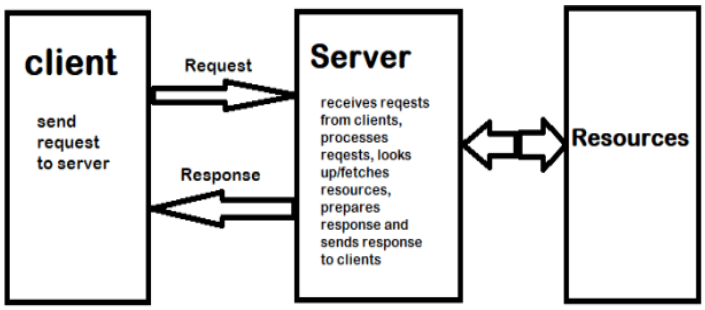

Figure 1.11: Request –Response CommunicationModel

Publish-Subscribe Model

Publish-Subscribe is a communication model in which authors, brokers, and customers are all involved. Data were gathered from publishers. The data is sent by the publishers tothe topics handled by the broker. Publishers are ignorant of their clients. Consumers subscribe to the issues that the broker manages. When the broker receives data for an issue from the publisher, it distributes it to all subscribers**.** Publish Subscribe Model is shown in Figure 1.12

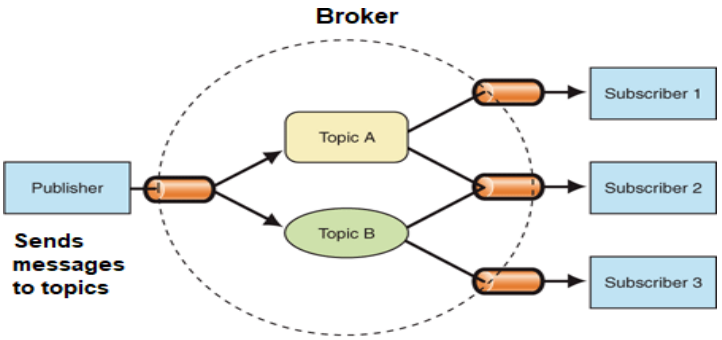

Figure 1.12: Publish-Subscribe Model

Push-Pull Model

Push-Pull is a communication model in which data suppliers push data to queues and users pull data from the queues. Take data from the Queues. Producers are notexpected to be aware of their clients. Queues aid in the separation of messaging between producers and consumers. Queues often act as a buffer, which is useful where there is a difference between the rate at which suppliers push data and the rate at which consumers pull data.

Push – Pull Model is shown in Figure 1.13

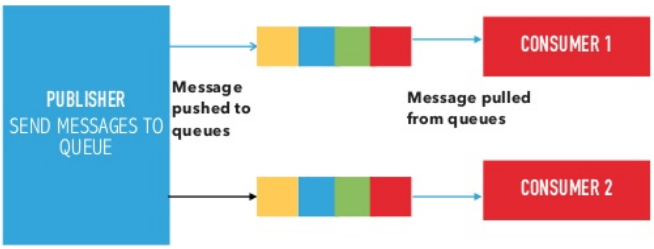

Figure 1.13: Push-Pull Model

Exclusive Pair Model

Exclusive Pair is a bidirectional, entirely duplex communication model that relies on a continuous link between client and server. When a link is created, it stays open until the client sends a request to close the connection. After the connection is established, the Client and server can send messages to one another. Exclusive pair is a stateful communication model in which the server is aware of all open links**.** Exclusive Pair Model is shown in Figure 1.14

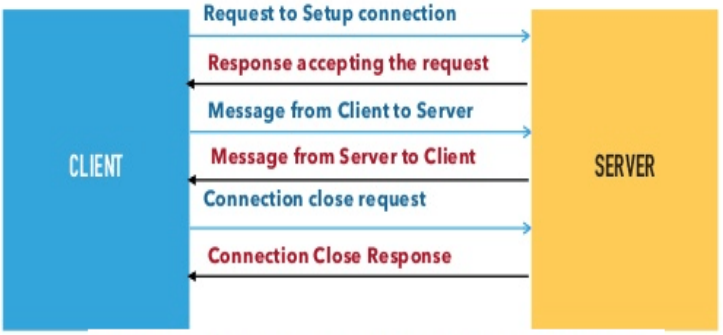

Figure 1.14: Exclusive Pair Model

1.7.3 IOT COMMUNICATION API's

For IoT communication, we usually used two APIs. These are the IoT Communication APIs: REST-based Communication APIs
APIs for WebSocket-based Communication APIs based on REST Representational state transition (REST) is a collection of architecture standards that can be used to design Web services and Web APIs that rely on

the properties of networks and how resource states are handled and transmitted. The rest architectural restriction applies to the modules, connector and data elements, within a distributed hypermedia system that use REST APIs that implement the request response communication model. The below are the remaining architectural constraints:

Client-server architecture – The division of interests is the driving theory behind the client-server constraint. Clients, for example, need not be concerned with data management, which is the responsibility of the provider. Similarly, the server should not be concerned with the user interface, which is the responsibility of the client. Separation enables client andserver to be created and updated separately.

Stateless – Each request from client to server must contain all of the information required to interpret the request and cannot make use of any server-stored context. The session stateis completely stored on the client.

Cache-able – Cache constraints demand that the data in a response to a request be levelled as cache-able or non-cache- able, either implicitly or directly. If a response is cache-able, a client cache is allowed permission to reuse the response data for future, similar requests. Caching will reduce or delete certain orders, improving performance and scalability.

Layered system constraints – layered system constraints limit the action of components such that they cannot see past the immediate layer in which they are communicating. For eg, the client does not know if it is linked directly to the end server or by an intermediary. Allowing intermediaries to respond to request instead of the end server improves systemscalability without requiring the client to do anything different.

Uniform interface – Uniform interface constraints demand that the mode of communication between client and server beuniform. Resources are defined in requests (via URIs in web- based systems) and are distinct from the representations of thedata returned to the client. When a client owns a resource representation, it includes all of the information needed to update or uninstall the resource (provided the client has required permissions).

Each communication contains sufficient information to explain how to process the message.

Code on demand – Servers may have executable code or scripts for clients to run in their context. This is the only restriction that is discretionary.

A RESTful web service is a Web API‖ that is built with HTTP and REST concepts. REST is one of the most commonly used IoT communication APIs.

HTTP METHODS

WebSocket Based Communication API

WebSocket APIs allow full-duplex bidirectional communication between clients and servers. The exclusive pair communication model is used by WebSocket APIs. Unlike the request-response model used by REST, WebSocket APIs support complete duplex communication and do not need a new link to be formed for each message sent. The client sends a link configuration request to the server, which initiates WebSocket communication. The request (known as a WebSocket handshake) is sent over HTTP and is treated by the server as an update request. The server responds to the WebSocket handshake answer whether it accepts the WebSocket protocol. After the link is established, the client and server can send data/messages in full duplex mode to each other. Since there is no overhead for link setup and termination demands for each packet, the WebSocket API eliminates network traffic and latency. WebSocket's are ideal for IoT applications that require low latency or high throughput. As a result, Web sockets are the perfect IoT Communication APIs for IoT Systems. Web socket protocol is shown in below Figure 1.15

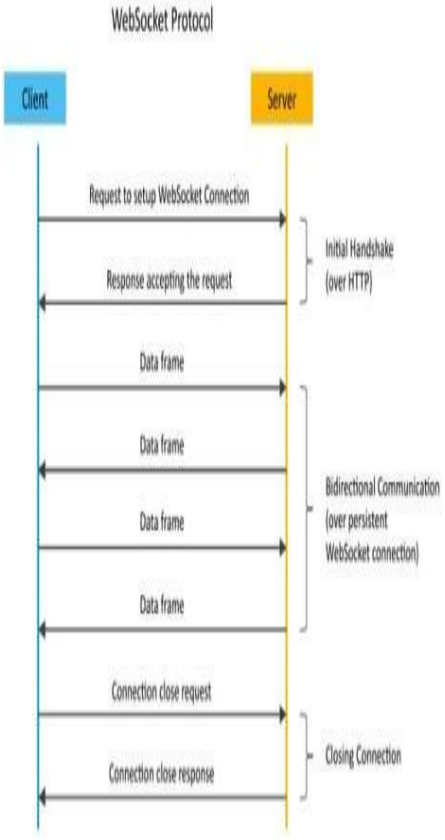

Figure 1.15: Web socket Protocol

1.8 IOT ENABLED TECHNOLOGIES

IoT relies heavily on common protocols and networking technology. RFID, NFC, low-energy IoT, low-energy wireless, low-energy radio protocols, and LTE-A are the main supporting technologies and protocols of IoT. In comparison to a typical uniform network of common networks, these technologies support the unique networking features requiredin an IoT system. In addition to these supporting technologies, the IoT depends on other technologies to maximize the possibilities provided by the IoT. These are as follows:

Figure 1.16: IoT Enabling Technologies

- Wireless Sensor Network
- Colud Computing
- Big Data Analytics
- Communication Protocols
- Embedded Systems

This auxiliary technology mean that data from IoT devices can be obtained, saved, and processed. But first, let's take a closer look at the Internet of Things' supporting technology.

1.9 IOT COMMUNICATION PROTOCOLS

Various communication protocols and technologies are used in the Internet of Things. Bluetooth, Usb, Radio Protocols,LTE-A, and Wi-Fi-Direct are some of the main IoT innovations and protocols (IoT Communication Protocols). This IoT communication protocols are designed to cater to and satisfy the basic functional requirements of an IoT framework. IoT Technologies & Protocols is shown in Figure 1.17

Figure 1.17: IoT Technologies and Protocols

A. Bluetooth

A vital short-range IoT communication protocol / technology. Bluetooth, which has grown in popularity in the computer and consumer goods industries. It is supposed to be critical for wearable devices in particular, which would once again bind to the IoT, but most likely from a smartphone in many situations. The new Bluetooth Low-Energy (BLE) – or Bluetooth Smart, as it is now called – protocol is critical for IoT applications. Importantly, although it has a similar range to Bluetooth, it has been built to use considerably less power.

B. Zigbee wireless technology

ZigBee, which is similar to Bluetooth, is mainly found in commercial environments. It has some major advantages in complex systems, such as low-power operation, high security, robustness, and high efficiency, and it is well placed to take advantage of wireless control and sensor networks in IoT applications. The most recent version of ZigBee is 3.0, which effectively unifies the different ZigBee wireless protocols into a single standard.

C. Z-Wave

Z-Wave is a low-power RF communications IoT technology that was designed specifically for home automation products such as lamp controllers and sensors, among many other applications. Z-Wave uses a simpler protocol than most, which makes for quicker and smoother development, but the only producer of chips is Sigma Designs, as opposed to numerous sources for other wireless technology such as ZigBee and others.

D. Cellular Internet (Wi-Fi)

Wi-Fi connectivity is a common IoT communication protocol that is always an obvious option for many developers, particularly given the availability of Wi-Fi within the home environment within LANs. There is a strong current system, as well as fast data transfer and the capacity to

process large volumes of data. The most popular Wi-Fi protocol currently found in households and many enterprises is802.11n, which provides a spectrum of hundreds of megabits per second, which is good for file transfers but could be too power-consuming for many IoT applications.

E. Molecular

GSM/3G/4G cellular communication features can be used by any IoT program that needs service over longer distances. Although cellular is clearly capable of transmitting significant amounts of data, especially for 4G, the cost and power usage would be prohibitively expensive for many applications. However, it could be suitable for sensor-based low-bandwidth-data ventures that transmit very smallvolumes of data across the Internet.

F. NFC

NFC (Near Field Communication) is an Internet of Things (IoT) technology. It enables easy and secure communications between electronic devices, especially smartphones, enabling consumers to conduct transactions without physically being present. It allows the user to attach mobile devices and view digital information. It basically expands the capability of contactless card technologies by allowing devices to exchange information at a distance of less than 4cm.

G. LoRaWAN

LoRaWAN is a popular IoT technology that is aimed atwide-area network (WAN) applications. LoRaWAN was created to provide low-power WANs with features that are directly needed to enable low-cost mobile secure communication in IoT, smart city, and industrial applications. Meets low-power consumption standards and serves vast networks of millions of millions of users, with data speedsranging from 0.3 kbps to 50 kbps.

1.10 EMBEDDED DEVICES (SYSTEM) IN IOT

It is important to understand embedded devices when learning about IoT or developing IoT projects. The embedded systems are the objects that make up the one-of-a-kind computer machine. This system may or may Embedded System Hardware is shown in Figure 1.18

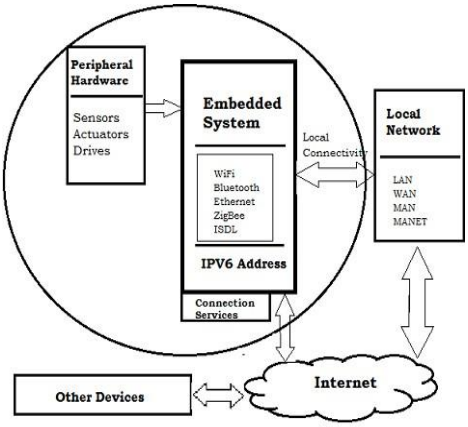

Figure 1.18: Embedded System Hardware

not provide Internet access. In most instances, an embedded device framework operates as a single program. This device, on the other an connect to the internet and communicate with othernetwork devices.

The embedded device may be of the microcontroller or microprocessor variety. An integrated circuit is present in eachof these forms (IC). A RISC family microcontroller, such as the Motorola 68HC11, PIC 16F84, Atmel 8051, and others, is a vital part of an embedded device. The most critical aspect that separates these microcontrollers from microprocessors such as the 8085 is their internal read and write memory. The following parts identify the vital embedded computer modules and system design.

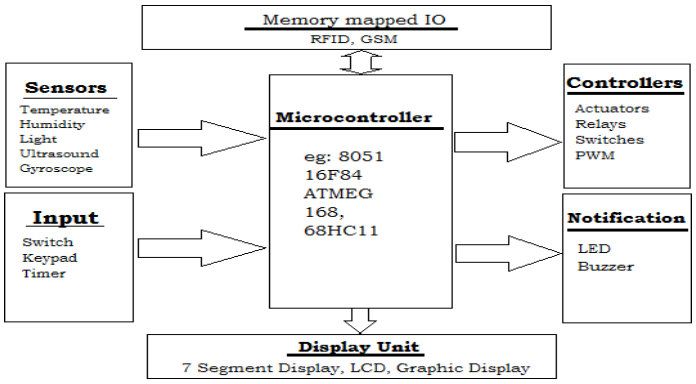

Figure 1.19: Basic Embedded Software for Embedded Systems

The embedded architecture that uses devices for the operating system is based on the language platform, and is essentially where real-time operations are conducted. Manufacturers create embedded applications for electronic devices such as automobiles, telephones, modems, appliances, and so on. Lighting controls run on an 8-bit microcontroller can be used as embedded system applications. It may also be sophisticated applications for missiles, process control systems, and aircraft, among other things.

1.11 IOT LEVELS

The elements of IoT architecture differ depending on the program Based on this fact, different levels of the IoT framework have been identified. Let's take a look at these IoT levels and their modules, as well as several explanations of how they can be used. Consider an air conditioner, the temperature of which must be controlled in order to comprehend IoT levels.

Level 1 IoT

- This level includes an air conditioner, a temperature sensor, a data collection and interpretation app, and a control and tracking app.
- The sensed data is saved locally.
- The data collection is carried out on a territorial level. Monitoring and control was accomplished by the use of a mobile app or a web app.

- The amount of data produced by this level application is notlarge.
- All control activities are carried out via the internet.
- For example, a temperature sensor is used to measure the temperature of a room, and the data is stored and analysed locally. Based on the study, a control operation is activated by a smartphone app, or it may simply assist in status tracking.

Level 2 IoT

- This level includes an air conditioner, a temperature sensor,Internet of Things(larger than level -1, with data processing performed here), a cloud, and a control and monitoring app.
- This level-2 is more complicated than level-1. Furthermore, the rate of sensing is higher than at level 1.
- The data at this level is very large. As a consequence, cloudcomputing is used.
- Data collection is done on a local level. The cloud is mainly used for computing.
- A control operation is initiated using a web interface or asmartphone app Based on data analysis.

Level 3 IoT

- As shown in the figure, this level contains an air conditioner, a temperature sensor, Internet of Things storage (larger than level 1), a cloud (for data analysis), and a control and tracking app.
- The data in this case is massive, i.e. huge data. The data sensing frequency is huge, and the gathered sensed data is processed on the cloud since it is massive.
- Data processing is performed in the cloud, and control actions are initiated using a smartphone app or a web app based on the results of the analysis.
- Examples cover agricultural systems, odour sensor-based space freshening technologies, and so on.

Level 4 IoT

- This level includes various sensors, data processing and interpretation, as well as a control and tracking app.
- At this level-4, multiple sensors that are independent of one another are used.
- The data obtained by these sensors is uploaded to the cloud in its own right. Instead of the need for large amounts of data computing, cloud storage is used at this level.
- Data collection is conducted in the cloud, and control actions are activated either by a web interface or a smartphone app based on the results.

Level 5 IoT

- This level includes several sensors, a coordinator node, data collection and interpretation, and a control and tracking app.
- This level is close to level 4, which also has a vast volume of data, and as a result, it is sensed using multiple sensors at a much faster rate and simultaneously.
- Data processing and analysis was carried out in the cloud.
- Based on the study, the control operation is carried out through a smartphone app or a web app.

1.12 IOT ARCHITECTURE

There is no commonly established special or normative agreement on Internet of Things (IoT) architecture. The IoT architecture differs in terms of functional area and solutions. The IoT architecture technology, on the other hand, is primarily composed of four major components, Architecture of IoT Solutions in Stages is shown in Figure 1.20

IoT Architecture Elements:

- **Sensors and Devices**
- **Networks and Gateways**
- **Layer of Cloud Management/Service**

The Application Layer

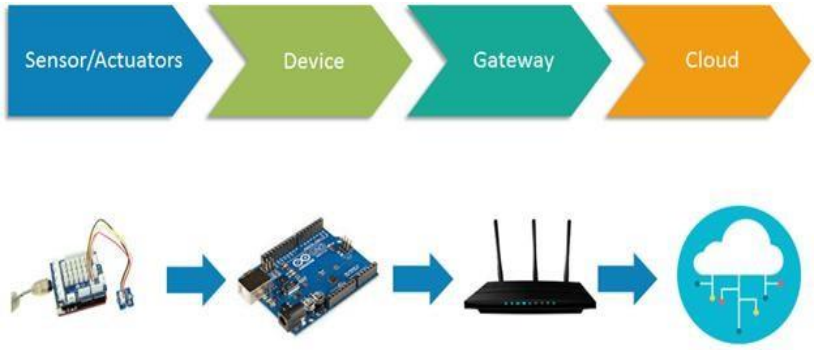

Figure 1.20: Architecture of IoT Solutions in Stages

There are multiple layers of IoT that are based on the capability and performance of IoT components to deliver the right approach to business organisations and end-users. The IoT architecture is a foundational approach for constructing the different components of IoT so that it can distribute services over networks to satisfy future needs.

The primary stages (layers) of IoT that provide the solution for IoT architecture are as follows.

1. **Sensors/Actuators:** Sensors and actuators are devices that can transmit, retrieve, and process data over a network. These sensors or actuators can be linked through wired or wireless connections. GPS, electrochemical, gyroscope, RFID, and other technologies are included. The majority of sensors need communication through sensors gateways. Sensors and actuators may be connected using a Local Area Network (LAN) or a Personal Area Network (PAN).

2. **Gateways and Data Acquisition:** Since these sensors and actuators generate a vast amount of data, high-speed Gateways and Networks are needed to process the data. This network may be of the Local Area Network (LAN) variety (such as WiFi, Ethernet, and so on), or it may be of the Wide Area Network variety (WAN such as GSM, 5G, etc.).

3. **Edge IT:** The hardware and software gateways that evaluate and pre-

process data before moving it to the cloud are referred to as the edge in the IoT Architecture. If the data read from the sensors and gateways is not changed from its previous reading value, it does not pass to the cloud, saving the data.

4. **Data center/Cloud:** The Data Center or Cloud is a Management Service that processes information through analytics, system management, and security measures. Aside from security controls and system management, the cloud transfers data to end-user applications such as Retail, Healthcare, Emergency, Environment, and Energy, among others. IoT Architecture is shown in Figure 1.21

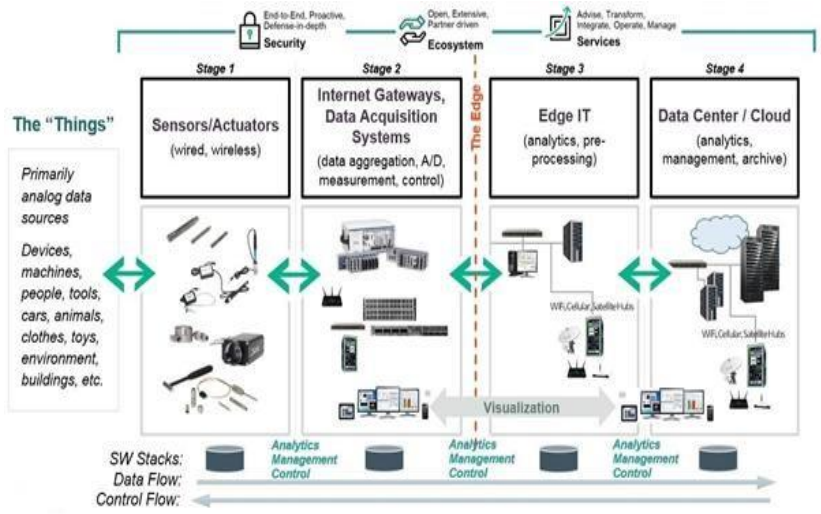

Figure 1.21: IOT Solutions Architecture

1.13 IOT APPLICATIONS

IoT solutions are widely used in numerous companies across industries. Some most common IoT applications are given in Table 1.1

Table 1.1: Applications of IoT

Application type	Description
Smart Thermostats	Helps you to save resource on heating bills by knowing your usage patterns.
ConnectedCars	IoT helps automobile companies handle billing, parking, insurance, and other related stuff automatically.
Activity Trackers	Helps you to capture heart rate pattern, calorie expenditure, activity levels, and skin temperature on your wrist.
Smart Outlets	Remotely turn any device on or off. It also allows you to track a device's energy level and get custom notifications directly into your smartphone.
Parking Sensors	IoT technology helps users to identify the real-time availability of parking spaces on their phone.
Smart City	Smart city offers all types of use cases which include traffic management to water distribution, waste management, etc.
Smart home	Smart home encapsulates the connectivity inside your homes. It includes smoke detectors, home appliances, light bulbs, windows, door locks, etc.
art supplychain	Helps you in real time tracking of goods while they are on the road, or getting suppliers to exchange inventory information.

Figure 1.22: IoT Applications

1.14 ADVANTAGES OF IOT

The below are the key advantages of IoT technology:

- Technological Optimization: IoT technology leads greatly to the advancement and development of technologies. For example, using IoT, a manufacturer may collect data from different car sensors. They are analyzed by the manufacturer in order to enhance their design and make them more effective.

- Enhanced Data Collection: Conventional data collection has drawbacks, as well as being structured for passive use. IoT allows for quick action on data.

- Less Waste: IoT provides real-time information, allowing for more efficient decision making and resource management. For example, if a manufacturer discovers a problem in several car engines, he will monitor the production plan of such engines and resolve the problem using the manufacturing belt.

- Improved Customer Engagement: IoT helps you to enhance customer service by identifying challenges and enhancing processes.

1.15 IOT LIMITATIONS

Now, in this Internet of Things guide, let's look at some of the drawbacks of IoT

- Security: IoT technology produces a network of interconnected devices. However, despite adequate security precautions, the device can have little authentication control during this phase.
- Privacy: Without the user's direct consent, the use of IoT reveals a large volume of sensitive data in extreme detail. This raises a slew of privacy concerns.
- Flexibility: The flexibility of an IoT system is a big problem. It is mostly concerned with integrating with another system, since there are several systems involved in the process.
- Complexity: The IoT system's architecture is often very difficult. Furthermore, deployment and servicing are difficult.
- Compliance: The Internet of Things has its own collection of rules and regulations. However, owing to its complexity, the task of compliance is very difficult.

CHAPTER 2
IOT - M2M

2.1 INTRODUCTION

The Internet of Things (IoT) M2M, or machine to machine communication, is the exchange of information between two computers without the involvement of an individual. This includes serial connections, powerline connections (percent), and wireless networking within the enterprise internet of Things (IoT). Switching to wi-fi has simplified M2M communication and allowed more applications to be linked.

When most people speak about M2M communication, they're usually referring to cellular communication for embedded devices. In this case, M2M communication will be vending machines sending out inventory records or ATM machines receiving permission to dispense cash. Iot M2M is shown in Figure 2.1

Figure 2.1: IoT M2M

When industries recognized the importance of M2M, it was granted a new name: the Internet of Things (IoT). IoT M2M has equivalent guarantees: to radically change the way the market works. M2M, like IoT, allows any sensor to explicitly interact, which opens up the possibilities of systems monitoring themselves and instantly reacting to changes in the environment with a lot less need for human intervention. M2M and IoT

are virtually synonymous M2M can refer to any two machines wireless or stressed out communicating with each other, whereas IoT (the newer time period) mostly corresponds to wireless communications.
Traditionally, M2M has been based on business telematics, ‖ which is a fancy way of saying data transmission with a few industrial benefits. However, many original M2M applications, such as smart meters, continue to exist today. Since the mid-2000s, when 2G cellular networks were introduced, wi-fi M2M has been ruled with the assistance of phones

2.2 IOT - M2M FUNCTIONS

As previously mentioned, device-to-system communication enables the internet of things. According to Forbes, M2M is one of the fastest-growing types of connected tool technologies on the market right now, owing to the fact M2M technology can connect millions of gadgets in a single network. Vending machines, medical devices, cars, and residences are among the numerous wired devices. IoT M2M Working is shown in Figure 2.2

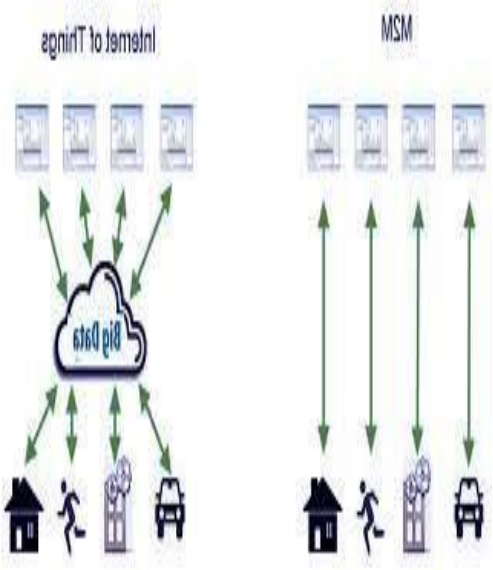

Figure 2.2: IoT M2M – Working of Machine to Machine

This can seem confusing, but the basic idea is very plain. M2M networks,

in principle, are similar to LAN or WAN networks, but they are only used to allow computers, sensors, and controls to communicate. This devices send information they collect back to other devices in the network. This method enables a person to assess what is happening in the whole network and give appropriate orders to member devices.

Figure 2.3: Internet of Things M2M- Working of Machine 2Machine

2.3 IOT - M2M APPLICATIONS

The prospects in the field of IoT M2M can be found in fourmajor use cases, which we have described in Figure 2.4:

Figure: 2.4 IoT M2M Applications

A. Manufacturing

Any manufacturing climate, whether it is food processing or common product production, is dependent on productionto ensure that prices are properly handled and policies are efficiently applied. Automating manufacturing processes in such a fast-paced environment is expected to boost strategies even more. In the manufacturing sector, this can mean highly automated computer renovation and security methods. For example, IoT M2M equipment enables business owners to receive notifications on their devices when a critical piece of equipment requires maintenance, allowing you to address issues as soon as they arise. Modern sensor networks connecting to the internet may choose to order alternate elements on a regular basis.

b. Household Items

Manufacturers such as LG and Samsung are now slowly releasing smart home appliances to help ensure a better quality of living for occupants. For example, an IoT M2M-capable washing machine should send updates to the owners' smart devices after it has done washing or drying, and a smart fridge should automatically order groceries from Amazon when its stock runs low. In cases when a property owner chooses to leave work early, she or he may choose to test the home heating system before leaving to ensure that the temperature at home is comfortable upon arrival.

C. Healthcare Tool Management

One of the most promising uses for M2M technologies is inthe area of health care. With M2M technologies, hospitals can simplify processes to ensure the highest levels of treatment. This is made possible by using devices that can respond faster than a human healthcare provider in an emergency. For example, if a patient's vital signs and conditions fall below average, an M2M-connected life support tool can robotically deliver oxygen and additional treatment before a healthcare provider appears on the scene. IoT M2M also enables patients to be monitored in their own homes rather than in hospitals ornursing centres.

D. Smart Application Management

Automation would soon become the modern norm in the current era of energy conservation. As utility agencies look fornew ways to simplify the metering process, M2M comes to therescue, assisting energy corporations in gathering power usagedata on a regular basis. As a result, we will charge consumers accurately. Smart metres can monitor how much energy a household uses and warn the power provider on a regular basis. This eliminates the need to dispatch a worker to read themetre or require the client to do a reading.

As more buyers, consumers, and business owners expect greater accessibility. Today's generation will still need to be prepared to meet tomorrow's needs and challenges. Statistics may provide information to clients, engineers, data analysts, and key decision-makers in real time, eliminating the need for guesswork.

2.4 SOFTWARE - DEFINED NETWORKING (SDN)

To comprehend software-defined networks, we must first comprehend the different networking planes.

Plane of Data:

This plane covers all operations affecting as well as originating from data packets submitted by the end user. This contains the following:

- Packet forwarding
- Data segmentation and reassembly
- Packet replication for multicasting

Plane of control:

This plane contains all operations that are needed to execute data plane activities but do not include end user data packets. In other words, this is the network's cortex. The control plane's operations include the following:

- Designing routing tables
- Defining packet-handling rules

Each transition in a typical network has its own data plane as well as its own control plane. The control planes of different switches share topology information, resulting in the construction of a forwarding table that specifies where an incoming data packet must be routed through the data plane.

SDN is a networking technique in which we remove the control plane from the switch and delegate it to a centralized device known as the SDN controller. As a consequence, a network administrator can mould traffic from a single console without hitting individual switches. The data plane remains in the switch, and when a packet reaches a switch, its forwarding operation is based by the entries of flow tables that the controller has pre-assigned. A flow table is made up of match fields (such as the input port number and the packet header) and instructions. The packet is first compared to the flow table entries' match fields.

The directions of the preceding flow entry are then carried out. The instructions may be to forward the packet through one or more ports, to drop the packet, or to add headers to the packet. If a packet does not meet a matching entry in the flow table, the switch queries the handler, which sends a new flow entry to the switch. Based on this flow entry, the switch decides whether to forward or drop the packet.

2.4.1 TYPICAL SDN ARCHITECTURE

The application layer includes standard network applications such as intrusion prevention, firewalls, and load balancing.

Control Layer:

It consists of the SDN controller, which functions as the network's brain. It also enables hardware abstraction for software built on top of it.

Infrastructure Layer :

This is made up of physical switches that shape the data plane and carry out real data packet movement.

The layers interact via a series of interfaces known as northbound APIs (between the application and control layers) and southbound APIs (between the application and the data layer) (between control and infrastructure layer). SDN Architecture is shown in Figure 2.5

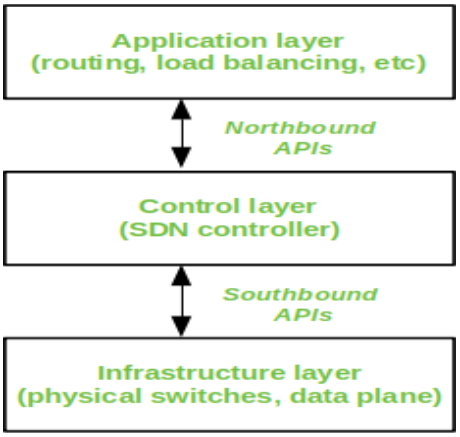

Figure 2.5: SDN Architecture

2.4.2 SDN ADVANTAGES:

- Since the network is programmable, it can be quickly changed via the controller rather than individual switches.
- Switch hardware becomes less expensive when each switch only needs a data plane.
- Since the hardware is abstracted, programmes may be written on top of the controller regardless of the switch provider.
- Increases security because the controller can track traffic and implement security measures. If the controller detects unusual behavior in network traffic, for example, it may reroute or drop the packets.

2.4.3 SDN DISADVANTAGES:

The network's core dependence suggests a single point of malfunction, i.e. if the controller becomes compromised, the whole network is affected.

2.5 NFV– SERVICE PROVIDERS ESTABLISHED

Unlike SDN, which was developed by academics and data centre architects, NFV was developed by a group of service providers. When service providers tried to accelerate the rollout of new network services in order to advance their sales and expansion ambitions, they discovered that hardware- based equipment restricted their ability to do so. They investigated standard IT virtualization technologies and discovered that NFV aided in the acceleration of service innovation and provisioning. As a result of this, many providers banded together to form the European Telecommunications Standards Institute (ETSI). The establishment of ETSI resulted in the establishment of the basic specifications and architecture of NFV.

Network Operators' networks are rapidly crowded by a vast range of proprietary hardware equipment. Launching a new network infrastructure often necessitates yet another variation, and finding the room and power to accommodate these boxes is becoming exceedingly difficult; this is exacerbated by rising energy prices, capital spending challenges, and a scarcity of expertise required to build, implement, and run increasingly complex hardware-based equipment. Furthermore, hardware-based appliances quickly hit end of life, necessitating a significant portion of the procure-design-integrate-deploy period to be replicated with little or no sales gain. Network Functions Virtualization seeks to solve these challenges by using standard IT virtualization technologies to combine multiple network infrastructure types into industry standard large capacity servers, switches, and storage that can be installed in Datacenters, Network Nodes, and end user premises. We assume the Network Functions Virtualization can be applied to any data plane packet processing and control plane operation in fixed and mobile network infrastructures.

ETSI is also working on NFV creativity. The Open Forum for NFV Project is another open source reference platform provided by the Linux Foundation (OPNFV). OPNFV collaborates together with ETSI and others to ensure that openpractices are regularly applied.

Network Operations The decoupling of network functions from proprietary hardware equipment and operating them as applications in virtual machines is known as network feature virtualization (NFV) (VMs). The numerous features, such as firewalls, traffic management, and virtual routing, are referred toas virtual network functions (VNFs).

NFV employs virtualized networking elements to enable a hardware-independent architecture. The basic compute, storage, and network infrastructure can all be virtualized and implemented on commercial off-the-shelf (COTS) hardware such as x86 servers. Since virtualized services are available, VMs may be assigned portions of the resources available on the x86 server. As a result, several virtual machines (VMs) will run on a single server and scale to consume the remaining free resources.

This also ensures that resources are less likely to beunused and that data centres with virtualized networks can beused more efficiently. The data plane and control plane inside the data centre and outside networks can also be virtualized using NFV.

The virtualization of network components is referred to as NFV, while SDN is a network architecture that injects automation and programmability into the network by decoupling network control and forwarding functions. To put it another way, NFV virtualizes network architecture, while SDN centralises network access. As SDN and NFV are combined, they build a network that is designed, run, and maintained entirely by software. An SDN usually includes an SDN administrator, northbound and southbound application software interfaces (APIs). The controller enables network managers to view the network and direct the underlying infrastructure's activities and policies. Southbound APIs gather network state information from that infrastructure and send it back to the

controller, which is expected to keep the network running smoothly. Northbound APIs are used by applications and utilities to express their resource requirements to the dispatcher.

2.5.1 BENEFITS OF NFV

Network providers who virtualize their networks can save money, reduce the time it takes to deliver new or upgraded products, and better scale and adapt the capacity available to apps and services. Other benefits include:

Less Vendor Lock-in: When running VNFs on commodity hardware, companies are not locked into proprietary, fixed- function boxes that require truck rolls and considerable time and labour to install and configure.

Greater Resource Efficiency: Since more can be achieved for fewer, a virtualized data centre or other facility is more economical to run. Data centre size, power usage, and cooling needs may all be minimized or retained while workload capability is expanded. This is possible since a single server will run multiple VNFs concurrently, requiring fewer servers to do the same amount of work. When network demand shifts, a company may upgrade its infrastructure through software rather than performing another truck roll. The need for an enterprise to physically upgrade its network and data centres is drastically diminished.

Organizations will take advantage of NFV's versatility to rapidly respond to evolving customer needs and emerging market opportunities. In other words, because the network architecture can be modified to better accommodate the organization's new goods, the time-to-market cycle is shortened. A network that has experienced NFV is therefore capable of rapidly and efficiently responding to changes in resource demand as traffic to the data centre rises or decreases. SDN apps will dynamically scale up and down the amount of VMs and the services available to them.

2.5.2 NFV-RELATED DIFFICULTIES

The NFV problems are embedded in three components of the technology: the NFV manager (NFVM), VNFs, and the NFV infrastructure (NFVI). The three elements are so closely coupled that, rather than theoretically simplifying things for network operators, they really add complexity and difficulty when implementing NFV at scale. The Lean NFV project has sought to resolve these concernsby introducing a new approach to NFV architecture.

The complexity currently impeding NFV stems not from how any of the above components is designed, but rather from how they are woven together into an overall system, the organization claimed in a white paper published in 2019.
More precisely, the complexity occurs when the NFV manager is incorporated with the current computational architecture, when VNFs are integrated with the NFV manager, and when communication between the different components of the NFV manager is required.‖
According to the white paper, the emphasis should be on simplifying the three points of incorporation such that other aspects of NFV designs can be innovated more openly.
One explanation for the complexity of NFV technology components is that several organizations have failed tostandardize them. As a result, there has been no particular solution that has succeeded for the whole market, and no set of principles that has stuck out enough to be heavily engaged in or implemented.

2.5.3 HISTORY OF NETWORK FUNCTIONS VIRTUALIZATION

NFV was created by service providers who wanted to make it simpler and quicker to implement new network functions or applications. As previously stated, various serviceproviders established multiple standards organisations rather than one. In order to create a standard NFV architecture, several service providers chose an open source approach. The European Telecommunications Standards Institute (ETSI) is a prominent standards organisation that was the firstto publish an NFV standard in

October 2013. The ETSI ISGNFV is a standards body that has developed NFV management and network orchestration (MANO) and network orchestration standards (NFV MANO). In collaborative ventures such as OPNFV, ETSI is also crucial.

2.5.4 Compatibility between SDN and NFV

Software-Defined Networking (SDN) is highly complementary to Network Functions Virtualization (NFV),but it is not necessary (or vice-versa). SDN is not necessary to incorporate Network Functions Virtualization, but the two principles and solutions can be combined for greater benefit.

Non-SDN mechanisms, such as the techniques currently in use in many data centres, can be used to achieve Network Functions Virtualization goals. However, methods based on the separation of the control and data forwarding planes, such as those suggested by SDN, can improve efficiency, make current deployments more compatible, and make operation and maintenance procedures easier. By providing the infrastructure on which SDN applications can operate, Network Functions Virtualization can help support SDN. In addition, Network Functions Virtualization is closely aligned with the SDN goals of using commodity servers and switches.

Working Together: NFV and SDN

Let's take a look at how SDN and NFV could complement each other. Figure 2 depicts how a controlled router service is currently deployed at the customer site, using a router. The router function would be virtualized, as seen in Figure3, and NFV would be added to this situation. All that's left at the customer site is a Network Interface Device (NID) that serves as a demarcation point as well as a performance monitor. Finally, as shown in Figure 4, SDN is used to isolate the control from the data. Data packets are now forwarded by an optimised data plane, with the routing (control plane) featurerunning in a virtual machine on a rack mount server.

2.6 IOT SYSTEM MANAGEMENT

Advanced management skills are needed when managing multiple devices within a single system.

1) Configuration Automation: IoT system management capabilities can assist in the configuration automation of the system.

2) Tracking Operational and Statistical Data: Management systems may aid in the monitoring of a system's operational and statistical data. This information may be used to diagnose or forecast an issue.

3) Increased System Reliability: A management system that allows for the validation of system configurations before they are implemented can help to increase system reliability.

4) System-Wide Configurations: For IoT systems with several devices or nodes, maintaining system-wide configuration is crucial to the system's proper operation.

5) Multiple Device Configurations: It might be beneficial for certain systems to have multiple valid configurations that are implemented at different times or under different conditions.

6) Configuration Retrieval & Reuse: Management systems that can retrieve settings from devices can assist in reusing certain configurations for other devices of the same kind.

2.7 NETCONF-YANG FOR IOT SYSTEM ADMINISTRATION

The NETCONF protocol manipulates configuration and state data, and YANG is a data processing language used to model this data. NETCONF-YANG is a generic approach to managing IoT devices. IoT systems Management is shown in Figure 2.6 The following are the roles played by different components: Management System

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

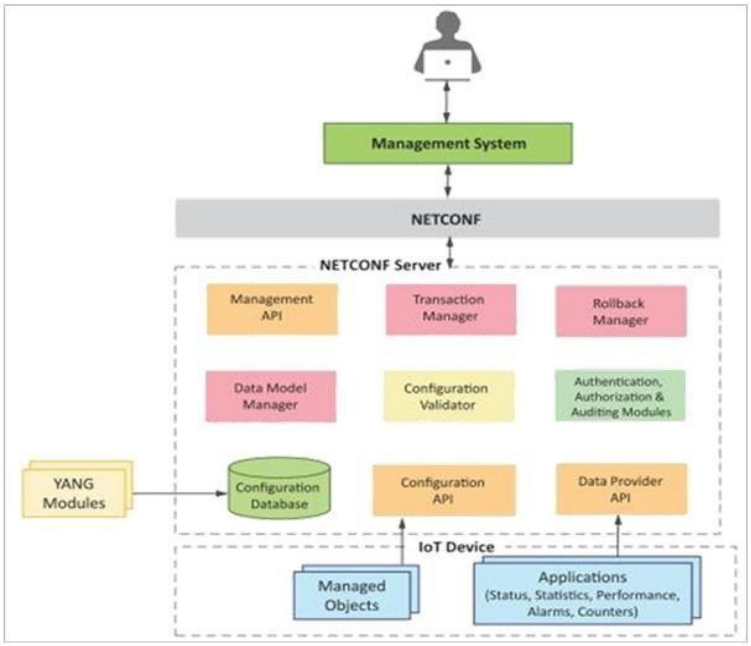

Figure 2.6: IoT Systems Management with NETCONF-YANG

- **Management System**: The operator uses a management system to send NETCONF messages to configure the IoT CHAPTER, and the device responds with NETCONF messages with state information and alerts.

- **Management API:** This API enables management applications to launch NETCONF sessions.

- **Transaction Manager**: this component executes all NETCONF transactions and ensures that their ACID properties are met.

- **The Rollback Manager** is in charge of producing all of the transactions required to restore a current configuration to its previous state.
- **Data Model Manager:** Keeps track of all YANG data models and the controlled objects that go with them. Also keeps track of the applications that supply data for each section of a data model.
- **Configuration Validator:** determines if the resulting configuration after applying a transaction is valid.
- **Configuration Database:** This database holds both configuration and operational information.
- **Configuration API:** The application on the IoT system can read configuration data from the configuration datastore and write operational data to the operation a datastore using the configuration API.
- **Data Provider API:** The Data Provider API allows IoT system applications to register for callbacks for different events. Applications can disclose statistics and operational data using the Data Provider API.

2.7.1 STEPS FOR IOT DEVICE MANAGEMENT WITH NETCONF-YANG

- Create a system YANG model that specifies the system's configuration and state data.
- Use the Inctool that comes with Libnetconf to finish the YANG model.
- In the TransAPImodule, fill in the IoT system management code.
- To generate the libraryfile, create the callbacks C file.
- Using the Netopeer managertool, load the YANG and TransAPI modules into the Netopeer server.

- The operator will now use the NetopeerCLI to connect from the management system to the Netopeer server.

- The Netopeer CLI allows the operator to issue NETCONF commands. On the IoT device, commands can be given to adjust the configuration data, get operational data, or run an RPC.

2.8 SNMP

The Network Architecture Board (IAB) described SNMP as an application–layer protocol for exchanging management information between network devices in RFC1157. The Transmission Control Protocol Internet Protocol (TCPIP)protocol suite includes it.

SNMP is a commonly used network protocol for managing and monitoring network components. The majority of professional–grade network elements have an SNMP agent. Tocommunicate with network monitoring software or a networkmanagement system, these agents must be activated and configured (NMS).

2.8.1 SNMP BASIC COMPONENTS AND THEIR FUNCTIONALITIES

- SNMP consists of
- SNMP Manager
- Managed devices
- SNMP agent
- Management Information Database Otherwise called as Management Information Base (MIB)

SNMP Manager: An SNMP manager, also known as a management system, is a separate agency that communicates with network devices that have SNMP agents installed. One or more network management systems are usually run on this device. Main functions of SNMP Manager

- Agents enquire
- Collects agent answers
- Variables are set in agents.
- Recognizes agents ‘asynchronous events.
- Devices that can be controlled:

A controlled CHAPTER, also known as a network feature, is a component of a network that needs monitoring and management, such as routers, switches, servers, workstations, printers, UPSs, and so on.

Agent: SNMP

The agent is a network element-packaged software. When you enable the agent, it collects the management information database from the system locally and makes it accessible to the SNMP manager when it's needed. These agents could be generic (for example, Net-SNMP) or vendor-specific (e.g. HP insight agent)

Main functions of an SNMP agent

- Gathers data on the local environment for management purposes
- As specified in the MIB, stores and retrieves management data.
- Notifies the manager of an occurrence.
- Serves as a proxy for a network node that isn't SNMP-manageable.
- SNMP Communication Diagram in Its Most Basic Form

SNMP is shown in Figure 2.7

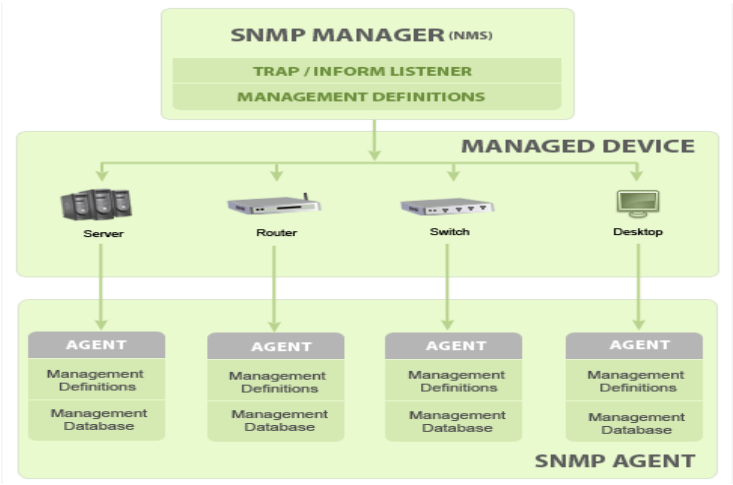

Figure 2.7: SNMP

Management Information Base or Database is a term used to describe a collection of management information (MIB) Every SNMP agent keeps a database of information about the managed device parameters. The SNMP manager consults this database to request specific information from the agent, which is then translated as needed for the Network Management System (NMS). The Management Information Base is a shared database between the Agent and the Manager (MIB).

These MIBs typically contain a standard set of statistical and control values for network hardware nodes. Through the use of private MIBs, SNMP also allows for the extension of these standard values with values specific to a specific agent.

MIB files, in a nutshell, are a set of questions that an SNMP Manager can ask an agent. As specified in the MIB, the agent collects and stores these data locally. As a result, for each type of agent, the SNMP Manager should be aware of these standard and private questions.

SNMP Walk Tool and SNMP MIB Browser

The SNMP MIB Browser in Manage Engine's Free Tools helps to load/unload MIBs and fetch MIB data from SNMP(v1, v2c, v3) agents.

The SNMP MIB Browser is a comprehensive monitoring tool for SNMP-enabled devices and servers. You can load and view multiple MIB modules, as well as perform SNMP operations such as GET, GETNEXT, and SET. This tool allows you to view, configure, and parse SNMP traps and is simple to use. SNMP operations can also be performed on Windows and Linux devices.

2.9 OBJECT IDENTIFIER AND MIB STRUCTURE

The Management Information Base (MIB) is a database of data used to manage network elements. The MIBs are made up of controlled objects with names like Object Identifier (Object ID or OID).

Each Identifier is special and represents a controlled device's particular characteristics. Each identifier's return value, when queried for, could be different, for example, Text, Number, Counter, etc... Scalar and Tabular Managed Objects or Object IDs are the two types of Managed Objects or Object IDs. With an example, they may be more understandable.

The result can only be one if the scalar is the device's vendor name. (A scalar object defines a single object case, according to the definition.)

Tabular: CPU usage of a Quad Processor, which will send me a result for each CPU separately, resulting in four results for that Object ID. (A tabular object describes several related object instances that are grouped together in MIB tables, according to the definition.)

In MIB, each Object ID is arranged in a hierarchical order. Person variable identifiers can be used to represent the MIB hierarchy in a tree structure.

A dotted list of integers is a common object ID. The OID for "sysDescr" in RFC1213, for example, is. 1.3.6.1.2.1.1.1 1.3.6.1.2.1.1.1 1.3.6.1.2.1.

2.9.1 MIB TREE DIAGRAM

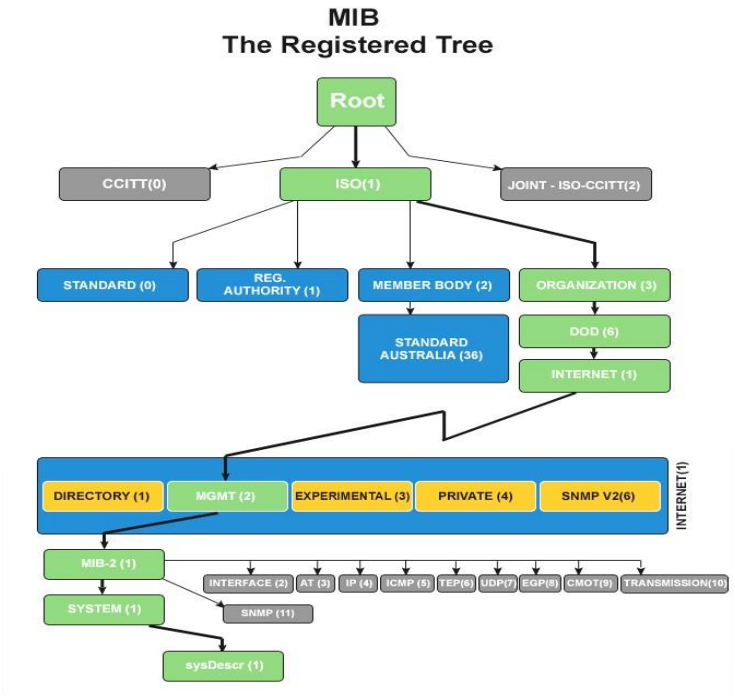

Figure 2.8: MIB Tree Diagram

2.9.2 SNMP BASIC COMMANDS

The SNMP protocol is commonly used due to its ease of use in exchanging data. The main explanation is a short list of commands, which are listed below:

- GET: The GET process sends a request to the managed system from the manager. It is carried out to obtain one or more values from the controlled device.
- GET NEXT: This is a similar operation to GET. The GET NEXT operation retrieves the value of the next OID in the MIB tree, which is a big difference.
- GET BULK: The GETBULK process retrieves a vast amount of data from a MIB table.

- SET: Managers use this operation to change or assign the value of the Managed CHAPTER.
- TRAPS: Unlike the commands mentioned above, which are started by the SNMP Manager, TRAPS are started by the Agents. When an incident occurs, the Agent sends a signal to the SNMP Manager.
- INFORM: This order is identical to the TRAP command issued by the Agent, but INFORM often provides assurance from the SNMP manager that the message has been received.
- RESPONSE: This command is used to bring back the value(s) or signal of the SNMP Manager's behaviour.

Traps for SNMP:

SNMP traps enable an agent to send an unsolicited SNMP message to the SNMP manager in the event of a significant event. The current sysUpTime value, an OID defining the type of trap, and optional variable bindings are all included in SNMP Trap protocols. Trap configuration variables in the MIB are used to specify the destination addressing for SNMP traps in an application-specific manner. In SNMPv2, the trap message's format was modified, and the protocol data CHAPTERs were called SNMPv2-Trap.An example of typical SNMP communication

SNMP messages are packaged as User Datagram Protocol (UDP) and then wrapped and distributed in the Internet Protocol since they are part of the TCP IP protocol set. The four–layer model produced by the Department of Defense is depicted in the diagram below (DoD).

Figure 2.9: SNMP COMMUNICATION

GET/GET NEXT/GET BULK/SET

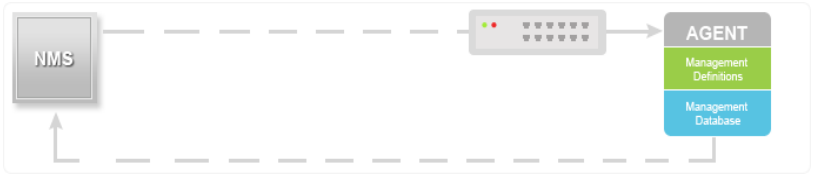

Figure 2.10: GET/GET NEXT/GET BULK/SET

TRAP

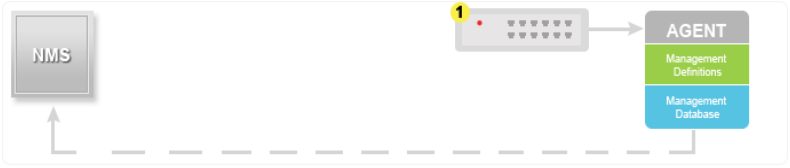

Figure 2.11: TRAP

INFORM

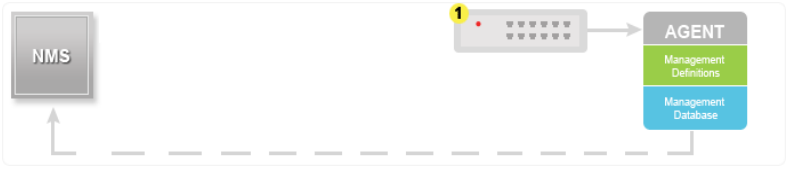

Figure 2.12: INFORM

By default the SNMP port is 161 and TRAP/ INFORM uses SNMP port 162 for communication.

2.9.3 SNMP VERSIONS

Since the inception SNMP, has gone through significant upgrades. However SNMP Protocol v1 and v2c are the most implemented versions of SNMP. Support to SNMP Protocol v3 has recently started catching up as it is more secured when compare to its older versions, but still it has not reached considerable market share.

SNMPv1:

This is the first version of SNMP protocol, which is definedin RFCs 1155 and 1157.

SNMPv2c:

This is the revised protocol, which includes enhancements of SNMPv1 in the areas of protocol packet types, transport mappings, MIB structure elements but using the existing SNMPv1 administration structure ("commonly based" and hence SNMPv2c). It is defined in RFC 1901, RFC 1905,RFC 1906, RFC 2578.

SNMPv3:

SNMPv3 defines the secure version of the SNMP. SNMPv3 protocol also facilitates remote network monitoringconfiguration of the SNMP entities. It is defined by RFC 1905, RFC 1906, RFC 3411, RFC 3412, RFC 3414, RFC 3415. Though each version had matured towards rich functionalities, additional emphasis was given to the security aspect on each upgrade. Here is a small clip on each edition's security aspect. SNMP Versions are shown in Table 2.1

Table 2.1: SNMP Versions

SNMP v1	Commonly–based security
SNMP v2c	Commonly–based security
SNMP v2u	User–based security
SNMP v2	Party–based security
SNMP v3	User–based security

CHAPTER 3
IOT PROGRAMMING LANGUAGE

3.1 INTRODUCTION

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This **tutorial** gives enough understanding on **Python programming** language.

3.1.1 PYTHON HISTORY AND VERSIONS

- Python laid its foundation in the late 1980s.
- The implementation of Python was started in December 1989 by **Guido Van Rossum** at CWI in Netherland.
- In February 1991, **Guido Van Rossum** published the code (labelled version 0.9.0) to alt.sources.
- In 1994, Python 1.0 was released with new features like lambda, map, filter, and reduce.
- Python 2.0 added new features such as list comprehensions, garbage collection systems.
- On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify the fundamental flaw of the language.
- ABC programming language is said to be the predecessor of Python language, which was capable of Exception Handling and interfacing with the Amoeba Operating System.
- The following programming languages influence Python: ABC language.Modula-3

3.2 PYTHON

There is a fact behind choosing the name Python. **Guido van Rossum** was reading the script of a popular BBC comedy series "**Monty Python's Flying Circus**". It was late on-air 1970s.

Van Rossum wanted to select a name which unique, sort, and little-bit mysterious. So he decided to select naming Python after the **"Monty Python's Flying Circus"** for their newly created programming language.

The comedy series was creative and well random. It talks about everything. Thus it is slow and unpredictable, which made it very interesting.

Python is also versatile and widely used in every technical field, such as Machine Learning, Artificial Intelligence, Web Development, Mobile Application, Desktop Application, Scientific Calculation, etc.

Python Version List

Python programming language is being updated regularly with new features and supports. There are lots of update in Python versions, started from 1994 to current release. A list of Python versions with its released date is shown in Table 3.1

Table 3.1: Python Versions and Released Dates

Python Version	Released Date
Python 1.0	January 1994
Python 1.5	December 31, 1997
Python 1.6	September 5, 2000
Python 2.0	October 16, 2000
Python 2.1	April 17, 2001
Python 2.2	December 21, 2001
Python 2.3	July 29, 2003
Python 2.4	November 30, 2004
Python 2.5	September 19, 2006
Python 2.6	October 1, 2008
Python 2.7	July 3, 2010

Python 3.0	December 3, 2008
Python 3.1	June 27, 2009
Python 3.2	February 20, 2011
Python 3.3	September 29, 2012
Python 3.4	March 16, 2014
Python 3.5	September 13, 2015
Python 3.6	December 23, 2016
Python 3.7	June 27, 2018
Python 3.8	October 14, 2019

Tips to Remember While Learning Python

The proper learning method will assist us in learning Python quickly and becoming a competent Python developer.

In this part, we'll go through a few pointers to keep in mind when learning Python.

1. Explain why we want to learn.

Before learning Python, you should have a specific target in mind. Python is a simple but powerful programming language. There are a lot of libraries, modules, built-in functions, and data structures in it. If the aim is vague, learning Python would be a tedious and monotonous experience. You could not get anything done if you don't have a definite target in mind.

So, first find out why you want to learn, which could be anything from learning anything new to developing projects in Python to switching to Python. The following are some of the most popular environments where Python is used. Choose any of them.

Data Processing and Analysis

Artificial Intelligence (AI) is a term that refers to a computer programm

- Playing games
- Robots/Hardware/Sensors
- Applications for Desktop

Choose one or two areas that concern you and begin your Python journey.

2. Learn the Fundamental Syntax

It is the most significant and fundamental step in learning the Python programming language's syntax. We must first master the fundamentals of syntax before moving on to more advanced topics. Python is simple to learn and has a simple syntax, as we discussed in our previous tutorial. It does not have a semicolon or brackets. Its syntax is similar to that of the English language.

As a result, learning its syntax can take very little time. Once we understand the grammar, we will be able to learn more quickly and work on projects.

3. Make Your Own Code

The most efficient and reliable way to learn Python is to write code. Attempt to write code on paper and run it in your head first (Dry Run), then move on to the method. Writing code on paper will assist us in quickly becoming familiar with the syntax and storing the definition in our long-term memory. When writing the code, make an effort to use appropriate functions and variable names.

For Python programming, there are a variety of editors available that automatically highlight syntax issues. As a result, we don't need to pay much attention to these errors.

4. Continue to practice

The practice is the next important step. It must use Python concepts to incorporate them in code. Our regular codingpractice should be consistent. Consistency is essential for success in any endeavor, not just programming. Writing code on a regular basis can aid in the development of muscle memory. We may do a problem-solving exercise involving similar principles or solve at least two or three Python problems. Although it can seem difficult, muscle memory plays an important role in programming. It will put us ahead of those who think Python's reading definition is all that is needed.

5. Make the necessary notes.

Making notes on your own is a great way to learn Python's concepts and syntax. It will help you build the consistency and concentration necessary to become a Python developer. Make short and succinct notes with pertinent details and specific examples of the topic at hand. Keeping track of your own notes will also help you learn more quickly. According to a review published in Psychological Science,

6. Talk about your ideas with others.

While coding appears to be a solitary task, we can improve our abilities by communicating with others. We should talk to an expert or friends who are learning Python about our doubts. This practice will aid in the acquisition of additional knowledge, tips and tricks, and the resolution of coding issues. Python has a fantastic group, which is one of its best features. As a result, we will benefit from enthusiastic Python users.

7. Carry out small projects

After grasping the fundamental concepts of Python, a novice should attempt to work on small projects. It will assist you in better understanding Python and being a more integral part of it. Theoretical expertise isn't enough to master the Python programming language. It doesn't matter what these projects are as long as they teach you something. Start with simple projects like a calculator app, tic-tac-toe game, alarm clock app, to-do list, student or customer management system, and so on.

If you've mastered a small project, you can quickly move on to your more interesting domain (Machine Learning, Web Development, etc.).

8. Others to Teach

"If you want to learn something, teach it to someone else," goes the old adage. In the case of learning Python, this is also valid. Create blog posts, film videos, or enroll in classes at a local training centre to share your knowledge with other students. It will assist us in improving our understanding of Python and uncovering previously unseen gaps in our knowledge. If you don't want to do all of this, enter the online community and answer questions about Python.

9. Investigate Frameworks and Libraries

Python has a large number of modules and frameworks. After becoming acquainted with the fundamental concepts of Python, the next move is to investigate the Python libraries. Working with domain-specific projects necessitates the use of libraries. The following segment provides a brief overview of the major libraries.

- TensorFlow - TensorFlow is an artificial intelligence librarythat allows us to build large-scale AI projects.
- jango - This is an open source platform for creating webapplications. It's easy to use, adaptable, and manage.
- Flask is a web application that is also open source. It's usedto make web applications that aren't too heavy.
- Pandas - Pandas is a Python library for performing scientific computations.
- Keras - Keras is an open source library for working withneural networks.

Python has a plethora of libraries. We've listed a couple ofthem already. Ten Make a contribution to the open source community, as we all know, Python is an open source language, which ensures that everyone can use it. To expand our knowledge,we can also contribute to the Python online community.Contributing to open source projects is the most effective way to broaden one's horizons. We also get reviews, notes, andsuggestions on the work we've submitted. The feedback willencourage us to learn the best Python programming practices and assist us in being a good Python developer.

3.3 PYTHON APPLICATIONS

Python is a high-level programming language that is open source and comes with a range of libraries and frameworks. Python has grown in popularity due to its ease of use, simple syntax, and user-friendly setting. The following is an example of Python use.

- Desktop Applications
- Web Applications
- Data Science
- Artificial Intelligence
- Machine Learning
- Scientific Computing
- Robotics
- Internet of Things (IoT)Gaming
- Mobile Apps
- Data Analysis and Preprocessing

3.4 IMPORTANCE OF PYTHON

Python is a scripting language that is high-level, interpreted, interactive, and object-oriented. Python isintended to be a very readable language. It often uses English keywords instead of punctuation, and it has less syntactical constructions than other languages.

Python is a must-have skill for students and working professionals who want to become great software engineers, particularly if they work in the Web Development field. I'll go through some of the core benefits of learning Python:

- Python is Interpreted Python is processed by the interpreter at runtime. Before running your software, you do not need to compile it. This is analogous to the programming languages PERL and PHP.
- Python is Interactive You can sit at a Python prompt and write your programmes by interacting directly with the interpreter.
- Python is Object-Oriented Python supports the Object- Oriented programming style or methodology, which encapsulates code within objects.
- Python is a Beginner's Language Python is an excellentlanguage for beginning programmers, since it allows for the development of a broad variety of applications, ranging from basic text processing to web browsers and games.

3.5 PYTHON CHARACTERISTICS

The following are some of the most significant features of Python programming:

- It supports OOP as well as functional and formal programming approaches.
- It can be used as a scripting language or compiled into byte-code for large-scale application development.
- It supports dynamic type checking and offers very high-level dynamic data types.
- It allows for garbage disposal to be automated.
- C, C++, COM, ActiveX, CORBA, and Java are all easily integrated.

Example of a programme

Python's version of Hello World.

I'm going to send you a small traditional Python Hello World programme to get you excited about Python. You can check it out using the Demo connection.

("Hello, Python!"); print

3.6 PYTHON APPLICATIONS

Python, as previously said, is one of the most commonly used language languages on the internet. I'll mention a couple of them below:

- Python is easy to learn, with few keywords, a straightforward structure, and a well-defined syntax. This enables the student to easily learn the language.

- Python code that is easy to read is more clearly specified and accessible to the eyes.

- Simple to manage Python's source code is relatively simple to maintain.

- A large standard library The majority of Python's library isportable and cross-platform compatible on UNIX, Windows, and Macintosh systems.
- Interactive Mode Python has an interactive mode that enables interactive debugging and testing of code snippets.Python is portable, meaning it can run on a wide range of hardware platforms and has the same user interface acrossall of them.
- Extendable: The Python interpreter can be extended with low-level modules. These modules enable programmers toimprove the efficiency of their tools by adding to or customizing them.
- Databases Python has interfaces for all of the big commercial databases.
- GUI Programming Python facilitates the development and porting of graphical user interfaces to a variety of system calls, libraries, and operating systems, including Windows MFC, Macintosh, and Unix's X Window system.
- Python is more scalable than shell scripting in terms of structure and support for large programmes.

3.7 PYTHON FEATURES

Python has a lot of features, some of which are mentioned below –

1. Simple to code:

Python is a programming language with a high degree of abstraction. Python is a very simple language to learn compared to other programming languages such as C, C#, JavaScript, and Java. Python is a very simple language to code in, and everyone can learn the basics in a few hours or days. It's also a language that's easy to learn for programmers.

2. **Open Source and Free Software:**

   The Python programming language is freely accessible on the official website, and you can download it by clicking onthe Download Python keyword below. Since it is open-source, the source code is also available to the general public. As a result, you can download it, use it, and share it.

3. **Object-Oriented Programming Language (OOP):**

   Object-Oriented programming is one of Python's most important features. Python supports object-oriented languageand concepts such as classes, encapsulation, and so on.

4. **Support for GUI Programming:**

   Python modules such as PyQt5, PyQt4, wxPython, and Tkmay be used to build graphical user interfaces.

   PyQt5 is the most common Python graphical applicationframework.

5. **Advanced Language:**

   Python is a programming language with a high degree of abstraction. We don't need to recall the machine architecture or manage memory while writing Python programmes.

6. **Function that can be expanded:**

   Python is a language that can be extended. We can convertPython code to C or C++ and then compile it in that language.

7. **Python is a portable programming language:**

   Python is also a portable language. For instance, if we have python code for Windows and want to run it on other platforms like Linux, Unix, or Mac, we don't need to modify it; we can run it on any platform.

8. Python is an integrated programming language:

Python is also an interactive language since it can easily becombined with other programming languages such as c, c++, and others.

9. Language Interpretation:

Python is an Interpreted Language, which means that its code is implemented line by line. Python code does not need compilation, unlike other languages such as C, C++, Java, and others, making it easier to debug. Python's source code is translated into bytecode, which is an immediate representation of the code.

10. Large Standard Library

Python has a wide standard library that includes a large number of modules and functions, allowing you to avoid writing your own code for anything. Regular expressions, CHAPTER testing, web browsers, and other libraries are all available in Python.

Dynamically Typed Language (DTL):

Python is a language of dynamic typing. That is, the typeof a variable (for example, int, double, long, etc.) is determinedat run time rather than in advance, so we don't have to define the type of variable.

3.8 DATA TYPES IN PYTHON

Values may be stored in variables, and each value has a data form. Since Python is a dynamically typed language, we don't need to specify the variable's form when declaring it. Thevalue is implicitly bound to its form by the interpreter.

Data Types

Different types of values may be stored in a variable. A person's name, for example, must be stored as a string, while his or her id must be stored as an integer.

Python offers a number of common data types, each with its own storage system. The following is a list of Python data types. Python data types is shown in Figure 3.1

1. Numbers
2. Sequence Type
3. Boolean
4. Set
5. Dictionary

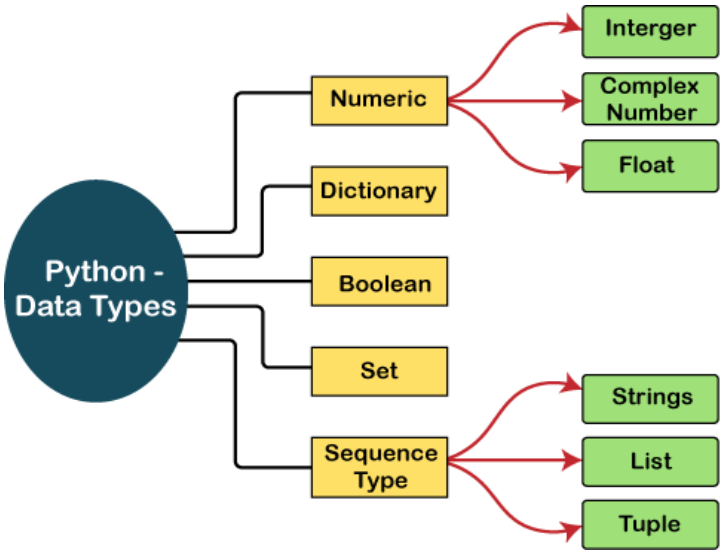

Figure 3.1: Python Data Types

We'll provide a quick overview of the above data types in this section of the tutorial. Later in this guide, we'll go through each of them in depth. Numeric values are stored in Numbers. A Python Numbers data-type contains integer, float, and complex values. The type() function in Python can be used to determine the data type of a variable. The isinstance() function is used to determine if an object belongs to a certain class.

When a variable is assigned a number, Python creates Number objects. As an example;

```
1. a = 5
2. print("The type of a", type(a))
3.
```

```
4.  b = 40.5
5.  print("The type of b", type(b))
6.
7.  c = 1+3j
8. print("The type of c", type(c))
9. print(" c is a complex number", is instance(1+3j,complex))
```

Output:

```
The type of a <class 'int'>
The type of b <class 'float'>
The type of c <class 'complex'>
c is complex number: True
```

Python supports three types of numeric data.

1. **Int -** Integer value can be any length such as integers 10, 2, 29, -20, -150 etc. Python has no restriction on the length of an integer. Its value belongs to **int**

2. **Float -** Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc. It is accurate up to 15 decimal points.

3. **complex -** A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively. The complex numbers like 2.14j, 2.0 + 2.3j, etc.

Sequence TypeString

The string is the set of characters enclosed in quotation marks. A string can be described in Python using single, double, or triple quotes.

Since Python has built-in functions and operators for performing operations on strings, string handling is a simple task.

The operator + is used to concatenate two strings in string handling, as in the operation "hello"+" python," which returns "hello python." Since the procedure "Python" *2 returns 'Python Python,' the operator * is known as a repetition operator.

In Python, the string is shown in the following example.

Example - 1

```
str = "string using double quotes"
print(str)
s = """A multilinestring"'
print(s)
```

Output:

```
string using double quotesA multiline
string
```

Consider the following example of string handling.

Example - 2

```
str1 = 'hello javatpoint' #string str1str2 = ' how are you' #string str2
print (str1[0:2]) #printing first two character using slice operator
print (str1[4]) #printing 4th character of the string
print (str1*2) #printing the string twice
print (str1 + str2) #printing the concatenation of str1 and str2
```

Output:

```
He
o
hello javatpointhello javatpoint
hello javatpoint how are youlist
```

Arrays in Python are similar to lists in C. The list, on the other hand, can contain data of various types. A comma (,) separates the things in the list, which are enclosed in squarebrackets [].

To access the list's data, we can use the slice [:] operators. The concatenation operator (+) and repetition operator (*) operate in the same way with lists as they do with strings.

Consider the following illustration.list1 = [1, "hi", "Python", 2] #Checking type of given list **print**(type(list1))

#Printing the list1

print (list1)

List slicing
print (list1[3:])

List slicing
print (list1[0:2])

List Concatenation using + operator
print (list1 + list1)

List repetation using * operator
print (list1 * 3)

Output:
[1, 'hi', 'Python', 2]
[2]
[1, 'hi']
[1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2]
[1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2]

TUPLE

In several ways, a tuple is identical to a list. Tuples, like lists, contain a set of objects of various data types. A comma (,) separates the tuple's elements, which are enclosed in parentheses ().

We can't change the size or value of the things in a tuple, because it's a read-only data structure.

Let's look at a simple tuple example.tup = ("hi", "Python", 2)# Checking type of tup

print (type(tup))

#Printing the tuple
print (tup)

```
# Tuple slicing
print (tup[1:])

print (tup[0:1])

# Tuple concatenation using + operator
print (tup + tup)

# Tuple repatation using * operator
print (tup * 3)

# Adding value to tup. It will throw an error.t[2] = "hi"
```

Output:

```
<class 'tuple'> ('hi', 'Python', 2)
('Python', 2)
('hi',)
('hi', 'Python', 2, 'hi', 'Python', 2)
('hi', 'Python', 2, 'hi', 'Python', 2, 'hi', 'Python', 2)
Traceback (most recent call last):
File "main.py", line 14, in <module>t[2] = "hi";
TypeError: 'tuple' object does not support item assignment
```

Dictionary

A dictionary is an unordered set of items with a key-value pair. It's similar to a hash table or an associative array in that each key stores a unique value. Any primitive data type can bestored in key, while value can be any Python object.

The comma (,) is used to divide the things in thedictionary, which are then enclosed in curly braces.

Consider the following illustration.

```
d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'}
# Printing dictionary
```

```
print (d)
# Accesing value using keys
print("1st name is "+d[1])
print("2nd name is "+ d[4])
print (d.keys()) print (d.values())
```

Output:

```
1st name is Jimmy2nd name is mike
{1: 'Jimmy', 2: 'Alex', 3: 'john', 4: 'mike'}
dict_keys([1, 2, 3, 4])
dict_values(['Jimmy', 'Alex', 'john', 'mike'])
```

Boolean

True and False are the built-in values of the Boolean type. These values are used to decide whether or not the given statement is valid. The class bool denotes it. True is expressed by any non-zero value or the letter 'T,' while false is represented by the letter 'F.' Consider the following illustration.

```
# Python program to check the Boolean type
print(type(True))print(type(False)) print(false)
```

Output:

```
<class 'bool'>
<class 'bool'>
NameError: name 'false' is not defined
```

Set

Python Set is the data type's unordered array. It is iterable, mutable (meaning it can be changed after creation), and has distinct elements. The order of the elements in set is undefined; it may return the element's changed sequence. A sequence of elements is passed in the curly braces and separated by a comma to construct the set, or a sequence of elements is passed in the curly braces and separated by a comma. It may hold a wide range of values. Consider the following illustration.

```
# Creating Empty setset1 = set()
```

```
set2 = {'James', 2, 3,'Python'}
#Printing Set value
print(set2)
# Adding element to the setset2.add(10)
print(set2)
#Removing element from the setset2.remove(2)
print(set2)
```

Output:

```
{3, 'Python', 'James', 2}
{'Python', 'James', 3, 2, 10}
{'Python', 'James', 3, 10}
```

3.9 DATA STRUCTURES IN PYTHON

Python runs on a number of operating systems, including Linux and Mac OS X. Data structures are fundamental computer scienceprinciples that aid in the development of efficient programs in any language. Python is a high-level, interpreted,interactive, and object-oriented scripting language that allowsone to learn data structure fundamentals in a more straightforward manner than other programming languages.

Data Structures Special to Python

These data structures are unique to the Python programming language, and they allow for more flexibility in storing various types of data as well as faster processing in thePython environment.

- **List:** A list is equivalent to an array, except the data elements may be of different data types. A python list can contain both numeric and string data.
- **Tuple:** Tuples are similar to lists, but they are immutable, meaning that the values in a tuple can only be read, not changed.

- **Dictionary**: The dictionary's data elements are key-value pairs.

3.9.1 PYTHON – LISTS

The list is a very flexible Python datatype that can be written as a list of comma-separated values (items) enclosed in square brackets. The most important aspect of a list is that its objects do not have to be of the same kind.

Putting different comma-separated values in square brackets is all it takes to make a list. For instance,

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

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

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

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

Accessing Values in Lists

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example −

```
#!/usr/bin/python

list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5, 6, 7 ]
print "list1[0]: ", list1[0]
print "list2[1:5]: ", list2[1:5]
```

When the above code is executed, it produces the following

result − list1[0]: physics

list2[1:5]: [2, 3, 4, 5]

Updating Lists

You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() method. For example −

```
#!/usr/bin/python

list = ['physics', 'chemistry', 1997,
2000]print "Value available at index 2
: " print list[2]
list[2] = 2001
print "New value available at index 2 :
"print list[2]
```

Note − append() method is discussed in subsequent section.

When the above code is executed, it produces the followingresult −

Value available at index 2 :

1997

New value available at index 2 :

2001

Delete List Elements

To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method if you do not know. For example −

```
#!/usr/bin/python

list1 = ['physics', 'chemistry', 1997,
2000]print list1
del list1[2]
print "After deleting value at index 2 :
"print list1
```

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

['physics', 'chemistry', 1997, 2000]After deleting value at index 2 : ['physics', 'chemistry', 2000]

Note − remove() method is discussed in subsequent section.

Basic List Operations

Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too, except thatthe result is a new list, not a string.

In fact, lists respond to all of the general sequence operations.

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

3.9.2 PYTHON – TUPLES

A tuple is a set of Python objects that cannot be modified. Tuples, like lists, are sequences. Tuples and lists differ in that tuples cannot be modified, whereas lists can, and tuples use parentheses whereas lists use square brackets.

Putting various comma-separated values into a tuple is as easy as that. You

may also place these comma-separated values between parentheses if you like. For instance,

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 );
tup3 = "a", "b", "c", "d";
```

The empty tuple is written as two parentheses containingnothing −

```
tup1 = ();
```

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

```
tup1 = (50,);
```

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

Accessing Values in Tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example −

```
#!/usr/bin/python

tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5, 6, 7 );
print "tup1[0]: ", tup1[0];
print "tup2[1:5]: ", tup2[1:5];
```

When the above code is executed, it produces the following

result − tup1[0]: physics

tup2[1:5]: [2, 3, 4, 5]

Updating Tuples

Tuples are immutable which means you cannot update or change the values of

tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates −

```
#!/usr/bin/python

tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');
```

```
# Following action is not valid for
tuples# tup1[0] = 100;

# So let's create a new tuple as
followstup3 = tup1 + tup2;
print tup3;
```

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

(12, 34.56, 'abc', 'xyz')

Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the **del** statement. For example –

```
#!/usr/bin/python

tup = ('physics', 'chemistry', 1997,
2000);print tup;
del tup;
print "After deleting tup :
";print tup;
```

This produces the following result. Note an exception raised, this is because after **del tup** tuple does not exist anymore ('physics', 'chemistry', 1997, 2000)After deleting tup :
Traceback (most recent call last): File "test.py", line 9, in <module>print tup;
NameError: name 'tup' is not defined

Basic Tuples Operations

Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except thatthe result is a new tuple, not a string.

In fact, tuples respond to all of the general sequence operations

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

3.9.3 PYTHON – DICTIONARY

Dictionary uses a colon (:) to distinguish each key from its value, commas to separate the objects, and curly braces to enclose all. With just two curly braces, an incomplete dictionary with no things is written as follows:

Values may not be unique inside a dictionary, but keys are. A dictionary's values may be any data type, but the keys must beimmutable data types like strings, numbers, or tuples.

Using the Dictionary to Find Values

To get the value of a dictionary element, you can use the familiar square brackets with the key. Here's an easy illustration:

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']
```

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

```
dict['Name']: Zara
dict['Age']: 7
```

If we attempt to access a data item with a key, which is not part of the dictionary, we get an error as follows –

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print "dict['Alice']: ", dict['Alice']
```

When the above code is executed, it produces the following result − dict['Alice']:

```
Traceback (most recent call last):
 File "test.py", line 4, in <module> print "dict['Alice']: ", dict['Alice'];
KeyError: 'Alice'
```

Updating Dictionary

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example –

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry

print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

When the above code is executed, it produces the following result − dict['Age']: 8
dict['School']: DPS School

Delete Dictionary Elements

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the **del** statement. Following is a simple example −

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
del dict['Name']; # remove entry with key 'Name'
dict.clear(); # remove all entries in dict
del dict ; # delete entire dictionary

print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

This produces the following result. Note that an exception is raised because after **del dict** dictionary does not exist anymore − dict['Age']:

Traceback (most recent call last):

File "test.py", line 8, in <module>print "dict['Age']: ", dict['Age'];
TypeError: 'type' object is unsubscriptable

Note − del() method is discussed in subsequent section.

Properties of Dictionary Keys

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys. There are two important points to remember about dictionary keys −

(a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins. For example –

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}
print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result − dict['Name']: Manni

(b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed. Following is a simple example –

```
#!/usr/bin/python

dict = {['Name']: 'Zara', 'Age': 7}
print "dict['Name']: ", dict['Name']
```

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

```
Traceback (most recent call last):
 File "test.py", line 3, in <module> dict = {['Name']: 'Zara', 'Age': 7};
TypeError: list objects are unhashable
```

3.10 PYTHON CONTROL STRUCTURES

Any computer programme can be written using the basic control structures, according to the structure theorem. A control structure (also known as a flow of control) is a programming block that analyses variables and chooses a course of action based on specified parameters. In simple terms, a control structure is simply a decision made by the machine. As a result, flow of control is the fundamental decision-making mechanism in programming, and it decides how a computer programme will react when given certain conditions and parameters.

Data and instructions are the two most fundamental elements of computer programming. Working with data necessitates an understanding of variables and data types, whereas working with instructions necessitates an understanding of control systems and statements. Three specific types of control structures are used to execute control flow in any given programme: sequential, selection, and repetition. Python control structures is shown in Figure 3.2

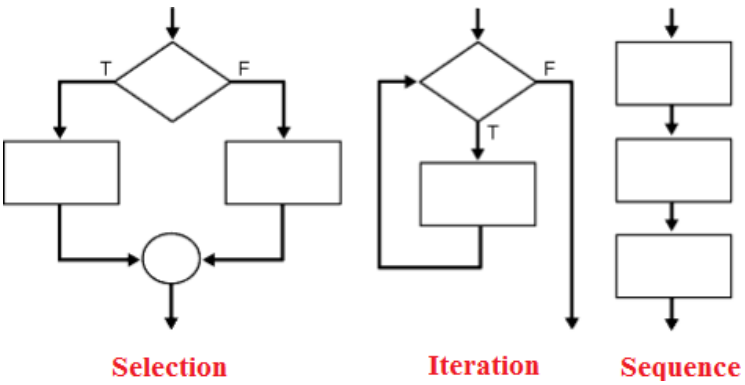

Figure 3.2: Python control structures

Sequential

When sentences are executed sequentially, they are executed one after the other. There's nothing else you need to do to make this happen.

Selection

Selection used for decisions, branching - choosing between 2 or more

alternative paths.

- if
- if...else
- switch

Repetition

Repetition used for looping, i.e. repeating a piece of codemultiple times in a row. while loop do, while loopfor loop

Python if...else Statement

In Python, each value has a datatype. Data types are simply classes, and variables are instances (objects) of these classes, since everything in Python programming is an object. If we want to run a programme only if a certain condition is met, we must make a decision.

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

Python if StatementSyntax

if test expression:

statement(s)

The programme tests the test expression and only executes the statement(s) if the text expression returns True.

The statement(s) are not executed if the text expression is False. The indentation in Python indicates the body of the if sentence. The body begins with an indentation and ends with the first unindented segment. Non-zero values are interpretedas True in Python. False is translated as None and 0.

Python if...else StatementSyntax

if test expression:

Body of ifelse:

Body of else

The if..else statement evaluates test expression and will execute body of if only when test condition is True.
If the condition is False, body of else is executed.
Indentation is used to separate the blocks.

Python if..else Flowchart

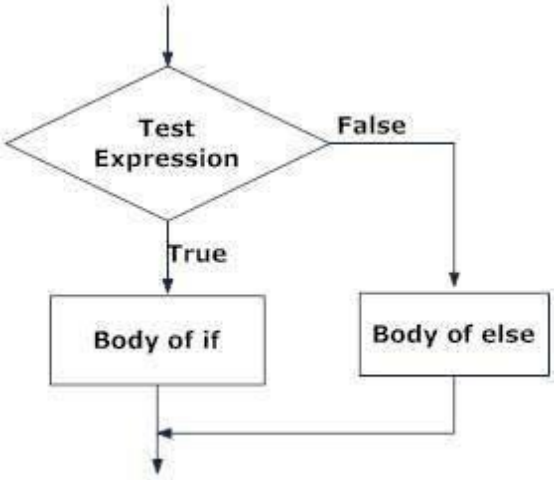

Figure 3.3: Python if...elif...else Statement

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

Elif stands for "else if." It enables us to look for multiple expressions at the same time. If the if condition is False, the next elif block's condition is verified, and so on. The body of else is executed if any of the conditions are False. According to the situation, only one of the many if...elif...else

blocks is executed. There can only be one else block in the if block. It can, however, have multiple elifblocks.

Flowchart of if...elif...else

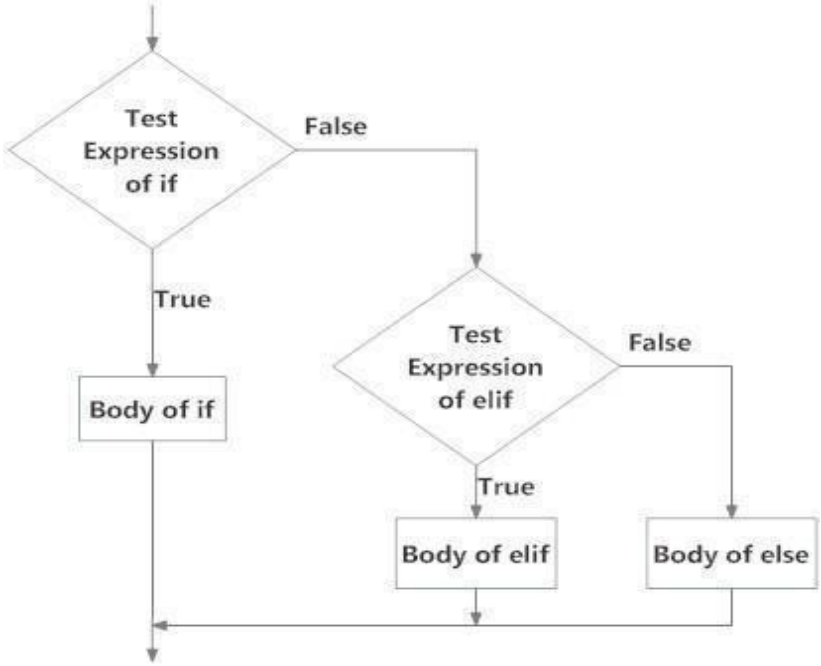

Figure 3.4: If elif Else Statement

Python Nested if statements

A if...elif...else statement may be nestled inside another if...elif...else statement. In computer programming, this is referred to as nesting. These statements can be nested within each other in any order. The only way to determine the extent of nesting is to use indentation. This can be perplexing, so we should avoid it if at all possible.

Python Nested if Example

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

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

else:

print("Positive number")else:
print("Negative number")

Output 1

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

Python for Loop

Python's for loop iterates over a sequence (list, tuple,string) or other iterable items. Traversal refers to iterating overa sequence. Loop's Syntax
in the sequence of val:

Foreword

Val is the variable that, for each iteration, takes the value ofthe item in the sequence.
The loop is repeated until the last component in thesequence is reached. The for loop's body is broken up.
Using indentation, separate the rest of the code.Loop Flowchart

Syntax

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

for val in numbers:

sum = sum+val

Output: The sum is 48 print("The sum is", sum)

when you run the program, the output will be:The sum is 48

The range() function

The range() function can be used to create a number sequence. Range(10)

generates a range of numbers from 0 to 9. (10 numbers). The start, end, and step size can also be described as a range (start,stop,step size). If no value is defined, the phase size is set to 1. Since it would be impractical to store all of the values in memory, this function does not do so. As a result, it remembers the start, end, and phase size, and produces the next number in real time.

To force this function to output all the items, we can use the function list(). The following example will clarify this.# Output: range(0, 10)
print(range(10))
# Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] print(list(range(10)))
Output: [2, 3, 4, 5, 6, 7] print(list(range(2, 8)))
Output: [2, 5, 8, 11, 14, 17] print(list(range(2, 20, 3)))

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

Program to iterate through a list using indexinggenre = ['pop', 'rock', 'jazz']
iterate over the list using index for i in range(len(genre)):print("I like", genre[i])
When you run the program, the output will be:I likepop I likerock I likejazz

While loop in Python

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

Syntax of while Loop in Python

while test_expression:
Body of while

The test expression is tested first in a while loop. Only if the test expression evaluates to True is the body of the loop entered. The test expression is tested again after one iteration. This procedure is repeated until the test

expression returns False. The body of the while loop in Python is determined by indentation. The body begins with an indentation and ends with the first unindented segment. Any non-zero value in Python is interpreted as True. False is translated as None and 0.

3.10.1 PYTHON BUILT IN FUNCTIONS

Python has a set of built-in functions.

Table 3.2: Python Built in Functions

Function	**Description**
abs()	Returns the absolute value of a number
all()	Returns True if all items in an iterable object are true
any()	Returns True if any item in an iterable object is true
ascii()	Returns a readable version of an object. Replaces none-ascii characters with escape character
bin()	Returns the binary version of a number
bool()	Returns the boolean value of the specified object
bytearray()	Returns an array of bytes
bytes()	Returns a bytes object
callable()	Returns True if the specified object is callable, otherwise False
chr()	Returns a character from the specified Unicode code.
classmethod()	Converts a method into a class method
compile()	Returns the specified source as an object, ready to be executed
complex()	Returns a complex number
delattr()	Deletes the specified attribute (property or method) from the specified object

dict()	Returns a dictionary (Array)
dir()	Returns a list of the specified object's properties and methods
divmod()	Returns the quotient and the remainder when argument1 is divided by argument2
enumerate()	Takes a collection (e.g. a tuple) and returns it as an enumerate object
eval()	Evaluates and executes an expression
exec()	Executes the specified code (or object)
filter()	Use a filter function to exclude items in an iterable object
float()	Returns a floating point number
format()	Formats a specified value
frozenset()	Returns a frozenset object
getattr()	Returns the value of the specified attribute (property or method)
globals()	Returns the current global symbol table as a dictionary
hasattr()	Returns True if the specified object has the specified attribute (property/method)
hash()	Returns the hash value of a specified object
help()	Executes the built-in help system
hex()	Converts a number into a hexadecimal value
id()	Returns the id of an object
input()	Allowing user input
int()	Returns an integer number
isinstance()	Returns True if a specified object is an instance of a specified object
issubclass()	Returns True if a specified class is a subclass of a specified object
iter()	Returns an iterator object
len()	Returns the length of an object
list()	Returns a list

locals()	Returns an updated dictionary of the current local symbol table
map()	Returns the specified iterator with the specified function applied to each item
max()	Returns the largest item in an iterable
memoryview()	Returns a memory view object
min()	Returns the smallest item in an iterable
next()	Returns the next item in an iterable
object()	Returns a new object
oct()	Converts a number into an octal
open()	Opens a file and returns a file object
ord()	Convert an integer representing the Unicode of the specified character
pow()	Returns the value of x to the power of y
print()	Prints to the standard output device
property()	Gets, sets, deletes a property
range()	Returns a sequence of numbers, starting from 0 and increments by 1 (by default)
repr()	Returns a readable version of an object
reversed()	Returns a reversed iterator
round()	Rounds a numbers
set()	Returns a new set object
setattr()	Sets an attribute (property/method) of an object
slice()	Returns a slice object
sorted()	Returns a sorted list
@staticmethod()	Converts a method into a static method
str()	Returns a string object
sum()	Sums the items of an iterator
super()	Returns an object that represents the parent class
tuple()	Returns a tuple
type()	Returns the type of an object
vars()	Returns the __dict__property of an object
zip()	Returns an iterator, from two or more iterators

3.11 PYTHON MODULES

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

Create a Module

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

Example

Save this code in a file named mymodule.pydef greeting(name):

print("Hello, " + name)

Use a Module

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

Example

Import the module named mymodule, and call the greetingfunction:
import mymodule mymodule.greeting("Jonathan")

Note: When using a function from a module, use the syntax: module_name.function_name. **Variables in Module**

The module can contain functions, as already described, butalso variables of all types(arrays, dictionaries, objects etc): **Naming a Module**
You can name the module file whatever you like, but it musthave the file extension .py

Re-naming a Module

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

Example

Create an alias for mymodule called mx:

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

Built-in Modules

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

Example

Import and use the platform module:

import platform

x = platform.system()print(x)

Using the dir() Function

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

Example

List all the defined names belonging to the platform module: import platform

x = dir(platform)print(x)

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

Import from Module

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

Example

The module named mymodule has one function and one dictionary:

def greeting(name):

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

Example
Import only the person1 dictionary from the module: from mymodule import person1
print (person1["age"])

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

3.12 PYTHON PACKAGES

We don't normally keep all of our files in the same folder on our computer. For easier access, we use a well-organized hierarchy of folders. Similar files are stored in the same directory; for example, all of the songs could be kept in the "music" directory. Python has packages for folders and modules for files, similar to this. When our framework software increases in size and contains more modules, we group related modules together in one package and separate modules into separate packages. This makes it simple to plan and understand a project (programme).

A Python programme can have sub-packages and modules, much as a directory can have sub-directories and files. In order for Python to recognise a directory as a package, it must contain a file named init.py. This file can be left blank, but we usually include the package's initialization code here. Here's an illustration. If we were making a game, one potential kit and module organisation would be as shown in the diagram below.

Package Module Structure in Python Programming Importing module from a package. We can import modules from packages using the dot (.) operator. For example, if want to import the start module in the above example, it is

done as follows.

import Game.Level.start

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

Game.Level.start.select_difficulty(2)

If this construct seems lengthy, we can import the module without the package prefix as follows. from Game. Level import start
We can now call the function simply as follows. start.select_difficulty(2)
Yet another way of importing just the required function (or class or variable) form a module within a package would be as follows. from Game.Level.start import select difficulty Now we can directly call this function. select_difficulty(2)
Although easier, this method is not recommended. Using the full namespace avoids confusion and prevents two same identifier names from colliding. While importing packages, Python looks in the list of directories defined in sys.path, similar as for module search path.

3.13 FILES HANDILING IN PYTHON

A file is a named location on disc where relevant data is stored. It's a form of non-volatile memory that stores data indefinitely (e.g. hard disk). We use files for potential use of the data because random access memory (RAM) is unreliable and loses its data when the device is switched off. We must first open a file before we can read or write to it. When we're done, we'll need to shut it down so that any resources associated with the file can be released. As a result, a file operation in Python is performed in the following order.
Open a file
Read or write (perform operation)
Close the file.

Open a file Using Python

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

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

When we open a file, we can specify the mode. We decide whether to read 'r', write 'w', or add 'a' to the file in mode. We can also choose whether to open the file in text or binary format. Reading in text mode is the default setting. When reading from the file in this mode, we get strings. Binary mode, on the other hand, returns bytes and is the mode to use when dealing with non-text files such as images or executables. Python File Modes are shown in Table 3.3

Table 3.3: Python File Modes

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

```
f = open("test.txt",'w') # write in textmode
```

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

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

Close a file Using Python

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

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

This method is not entirely safe. If an exception occurs when we are performing some operation with the file, the code exits without closing the file.
A safer way is to use a try...finally block.try:
f = open("test.txt",encoding = 'utf-8')
# perform file operations

finally:
f.close()

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

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

Write the File Using Python

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

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

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

Read the files in Python

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

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

Time Functions in Python

Python has defined a module, time‖ which allows us to handle various operations regarding time, its conversions and representations, which find its use in various applications in life. The beginning of time is started measuring from 1 January, 12:00 am, 1970 and this very time is termed as **epoch**‖ in Python.

3.14 OPERATIONS ON TIME :

1. **time()** :- This function is used to count the numberof **seconds elapsed since the epoch**.

2. **gmtime(sec)** :- This function returns a **structure with 9 values** each representing a time attribute in sequence. It converts **seconds into time attributes(days, years, months etc.)** till specified seconds from epoch. If no seconds are mentioned, time is calculated till present. The structure attribute table is given below.

Index Attributes Values0 tm_year 2008

1.tm_mon 1 to 12
2.tm_mday 1 to 31
3.tm_hour 0 to 23
4.tm_min 0 to 59
5.tm_sec 0 to 61 (60 or 61 are leap-seconds)
6.tm_wday 0 to 6
7.tm_yday 1 to 366
8.tm_isdst -1, 0, 1 where -1 meansLibrary determines DST

3. asctime("time") :- This function takes a time attributed string produced by gmtime() and returns a **24 character string denoting time**.
4. ctime(sec) :- This function returns a **24 character time string** but takes seconds as argument and **computes time till mentioned seconds**. If no argument is passed, time is calculated till present.
5. sleep(sec) :- This method is used to **hault the program execution** for the time specified in the arguments.

3.15 DATE MANIPULATIONS

Date manipulation can also be performed using Pythonusing datetime‖ module and using date‖ class in it

Operations on Date :

1. **MINYEAR** :- It displays the **minimum year** that can berepresented

using date class.

2. **MAXYEAR** :- It displays the **maximum year** that can berepresented using date class.

3. **date(yyyy-mm-dd)** :- This function returns a string withpassed arguments in order of year, months and date.

4. **today()** :- Returns the **date of present day** in the formatyyyy-mm-dd.

Output:

5. **fromtimestamp(sec)** :- It returns the **date calculated fromthe seconds** elapsed since epoch mentioned in arguments.

6. **min()** :- This returns the **minimum date** that can berepresented by date class.

7. **max()** :- This returns the **maximum date** that can berepresented by date class.

Python Classes/Objects

A class is a blueprint or prototype that is specified by theuser and used to build objects. Classes allow you to group data and features together. A new class introduces a new type of object, allowing for the creation of new instances of that type. Each instance of a class may have attributes attached to it to keep track of its state. Class instances may also have methodsfor changing their state (defined by their class).

Consider the following scenario: you want to keep track of the number of dogs with various characteristics such as breed and age. The first element of a list could be the dog's breed,while the second element could be its age. What if there are 100 different dogs? How can you know which one is supposedto be which? What if you were to give these dogs more abilities? This lacks structure, which is precisely why classes are needed. A class creates a user-

defined data structure with its own data members and member functions that can be accessed andused by creating a class instance. A class is similar to an object's blueprint.

Some points on Python class:

- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using thedot (.) operator.

Eg.: Myclass.Myattribute

Create a Class

To create a class, use the keyword class:

Example

Create a class named MyClass, with a property named x:class MyClass:
x = 5

Create Object

Now we can use the class named MyClass to create objects:Example
Create an object named p1, and print the value of x:p1 = MyClass()
print(p1.x)

The __init__() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in init_() function.

All classes have a function called_init_(), which is always executed when the class is being initiated.

Use the_init_() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Example

Create a class named Person, use the __init_() function toassign values for name and age:

class Person:

def __init_(self, name, age):

self.name = nameself.age = age

p1 = Person("John", 36)

print(p1.name)print(p1.age)

Class Objects

A Class's instance is an Object. A class is similar to a blueprint, while an instance is a copy of the class that containsactual data. It's no longer a concept; it's a living being, such as a seven-year-old pug. You may have a lot of dogs to make a lot of different scenarios, but without the class to help you, you'd be confused and have no idea what knowledge is needed.
An entity is made up of the following elements:

- **State:** It is represented by the attributes of an object. It alsoreflects the properties of an object.
- **Behavior:** It is represented by the methods of an object. It alsoreflects the response of an object to other objects.
- **Identity:** It gives a unique name to an object and enables oneobject to interact with other objects.

Object Methods

Objects can also contain methods. Methods in objects arefunctions that belong to the object.
Let us create a method in the Person class:

Example

Insert a function that prints a greeting, and execute it on the p1object:
class Person:
def __init__(self, name, age):
self.name = nameself.age = age
def myfunc(self):
print("Hello my name is " + self.name)
p1 = Person("John", 36)p1.myfunc()
The self Parameter

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.
It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:
Example
Use the words *mysillyobject* and *abc* instead of *self*:class Person:

def _____init_(mysillyobject, name, age): mysillyobject.name = name mysillyobject.age = age
def myfunc(abc):
print("Hello my name is " + abc.name)
p1 = Person("John", 36)p1.myfunc()
Modify Object Properties
You can modify properties on objects like this:

Example
Set the age of p1 to 40:
p1.age = 40
Delete Object Properties
You can delete properties on objects by using the del keyword:Example
Delete the age property from the p1 object:
del p1.age Delete Objects
You can delete objects by using the del keyword:Example
Delete the p1 object:
del p1
The pass Statement
class definitions cannot be empty, but if you for some reason have a class definition with no content put in the pass statement to avoid getting an error.
Example class Person:
pass

3.16 EXCEPTIONS IN PYTHON

Python has many built-in exceptions that are raised when your program encounters an error (something in the program goes wrong).

When these exceptions occur, the Python interpreter stops the current process and passes it to the calling process until it is handled. If not handled, the program will crash.

For example, let us consider a program where we have a function A that calls function B, which in turn calls function C. If an exception occurs in function C but is not handled in C, the exception passes to B and then to A. If never handled, an error message is displayed and our program comes to a sudden unexpected halt.

Catching Exceptions in Python

In Python, exceptions can be handled using a try statement.

The critical operation which can raise an exception is placed inside the try clause. The code that handles the exceptions is written in the except clause.

We can thus choose what operations to perform once we have caught the exception. Here is a simple example.

import module sys to get the type of exception import sys

randomList = ['a', 0, 2] for entry in randomList:

try:
print("The entry is", entry) r = 1/int(entry)
break except:
print("Oops!", sys.exc_info()[0], "occurred.") print("Next entry.")
print()
print("The reciprocal of", entry, "is", r) Output

The entry is a

Oops! <class 'ValueError'> occurred.Next entry.

The entry is 0

Oops! <class 'ZeroDivisionError'> occured.Next entry.

The entry is 2

The reciprocal of 2 is 0.5

In this program, we loop through the values of the randomList list. As previously mentioned, the portion that can cause an exception is placed inside the try block.

If no exception occurs, the except block is skipped and normal flow continues (for last value). But if any exception occurs, it is caught by the except block (first and second values).

Here, we print the name of the exception using the exc info() function inside sys module. We can see that a causes Value Error and 0 causes Zero Division Error.

Since every exception in Python inherits from the base Exception class, we can also perform the above task in the following way:

```
# import module sys to get the type of exception import sys
randomList = ['a', 0, 2] for entry in randomList:
try:
print("The entry is", entry)r = 1/int(entry)
break
except Exception as e:
print("Oops!", e._class_, "occurred.")print("Next entry.")
print()
print("The reciprocal of", entry, "is", r)
```

This program has the same output as the above program.

Catching Specific Exceptions in Python

In the above example, we did not mention any specific exception in the except clause.

This is not a good programming practice as it will catch all exceptions and

handle every case in the same way. We can specify which exceptions an except clause should catch.
A try clause can have any number of except clauses tohandle different exceptions, however, only one will be executed in case an exception occurs.
We can use a tuple of values to specify multiple exceptionsin an except clause. Here is an example pseudo code.

try:
do somethingpass

except ValueError:
handle ValueError exceptionpass

except (TypeError, ZeroDivisionError):
handle multiple exceptions
TypeError and ZeroDivisionErrorpass

except:
handle all other exceptionspass
Raising Exceptions in Python
In Python programming, exceptions are raised when errors occur at runtime. We can also manually raise exceptions usingthe raise keyword.
We can optionally pass values to the exception to clarify why that exception was raised.

>>> raise Keyboard Interrupt Traceback (most recent call last):
...
Keyboar Interrupt

>>> raise Memory Error("This is an argument")Traceback (most recent call last):
...
Memory Error: This is an argument

>>> try:
... a = int(input("Enter a positive integer: "))

... if a <= 0:
... raise ValueError("That is not a positive number!")
... except ValueError as ve:
... print(ve)
...
Enter a positive integer: -2 That is not a positive number!

Python try with else clause

In some situations, you might want to run a certain block of code if the code block inside try ran without any errors. For these cases, you can use the optional else keyword with the trystatement.

Note: Exceptions in the else clause are not handled by the preceding except clauses.

Let's look at an example:

program to print the reciprocal of even numbers

try:
num = int(input("Enter a number: "))assert num % 2 == 0

except:
print("Not an even number!")else:
reciprocal = 1/numprint(reciprocal) Output

If we pass an odd number:

Enter a number: 1 Not an even number!
If we pass an even number, the reciprocal is computed anddisplayed.

Enter a number: 40.25
However, if we pass 0, we get Zero Division Error as the codeblock inside else is not handled by preceding except.

Enter a number: 0
Traceback (most recent call last):

File "<string>", line 7, in <module>reciprocal = 1/num
ZeroDivisionError: division by zero

Python try...finally

The try statement in Python can have an optional finally clause. This clause is executed no matter what, and is generally used to release external resources.

For example, we may be connected to a remote data centerthrough the network or working with a file or a Graphical User Interface (GUI).

In all these circumstances, we must clean up the resource before the program comes to a halt whether it successfully ran or not. These actions (closing a file, GUI or disconnecting from network) are performed in the finally clause to guarantee the execution.

Here is an example of file operations to illustrate this.

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

This type of construct makes sure that the file is closed even ifan exception occurs during the program execution.

Python Internet modules

A list of some important modules in Python Network/Internet programming is shown in Table 3.4

Table 3.4 Python Internet Modules

Protocol	**Common function**	**Port No**	**Python module**
HTTP	Web pages	80	httplib, urllib, xmlrpclib
NNTP	Usenet news	119	nntplib
FTP	File transfers	20	ftplib, urllib
SMTP	Sending email	25	smtplib

POP3	Fetching email	110	poplib
IMAP4	Fetching email	143	imaplib
Telnet	Command lines	23	telnetlib
Gopher	Document transfers	70	gopherlib, urllib

CHAPTER 4

IOT PHYSICAL DEVICES AND END POINTS

4.1 INTRODUCTION

Device for the Internet of Things

Any object with a unique identifier and the ability to send/receive data (including user data) over a network qualifies as a "Thing" in the Internet of Things (IoT) (e.g., smartphone, smart TV, computer, refrigerator, car, etc.).

- Internet-connected IoT devices transmit information about themselves or their surroundings (e.g., information sensed by connected sensors) over a network (to other devices or servers/storage) or enable remote actuation of physical entities/environments.

Examples of IoT Devices

A home automation device that allows you to remotely track and manage the status of your appliances.

- An industrial computer that sends data to a server about its activity and health monitoring.
- A vehicle that transmits data about its position to a cloud-based service.
- A wireless-enabled wearable device that collects and sends data about a person, such as the number of steps taken, to a cloud-based service.

4.2 IoT BUILDING BLOCKS

Sensors, processors, gateways, and applications are the four fundamental building blocks of the IoT framework. To form a useful IoT structure, each of these nodes must have its own set of characteristics. Basic Building blocks of IoT is shown in Figure 4.1

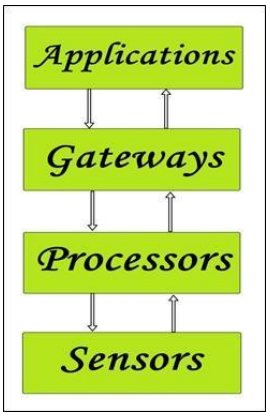

Figure 4.1: Simplified Block Diagram of theBasic Building Blocks of the IoT

Sensors:

These are the IoT devices' front ends. These are the system's so-called "Stuff." Their primary function is to gather data from the environment (sensors) or to transmit data to the environment (actuators).

- These must be individually recognisable devices with their own IP address in order to be easily identified over a vast network.
- They must be interested in nature, which means they must be able to gather data in real time. This can either function on their own (autonomous in nature) or be programmed by the user to meet their specific requirements (user-controlled).
- Gas sensors, water quality sensors, moisture sensors, and other sensors are examples of sensors.

Processors:

Processors are the IoT system's brain. Their primary purpose is to process the data collected by the sensors in orderto derive useful information from the massive data of raw datacollected. In a nutshell, it provides intelligence to the data.

- Processors are often real-time devices that can be easily operated by software. These are also in charge of data security, which includes data encryption and decryption.

- Embedded hardware devices, such as microcontrollers, process data because they have processors connected to them.

Gateways:

Gateways are in charge of routing processed data anddirecting it to the appropriate locations for proper (data) use.

- In other words, a gateway facilitates data exchange between two points. It allows data to be accessed over a network. Any IoT machine that wants to communicate needs to be connected to the internet.
- Network gateways include things like LANs, WANs, and PANs.

Applications:

An IoT system's applications are another end. Applications are needed for the correct use of all collected data.

- These cloud-based systems are in charge of giving the data obtained a meaningful value. Users manage applications, which serve as a delivery point for specific services.
- Home automation software, security systems, industrial control hubs, and other applications are examples.

In Figure 4.2 the extreme right block forms the application end of the IoT system.

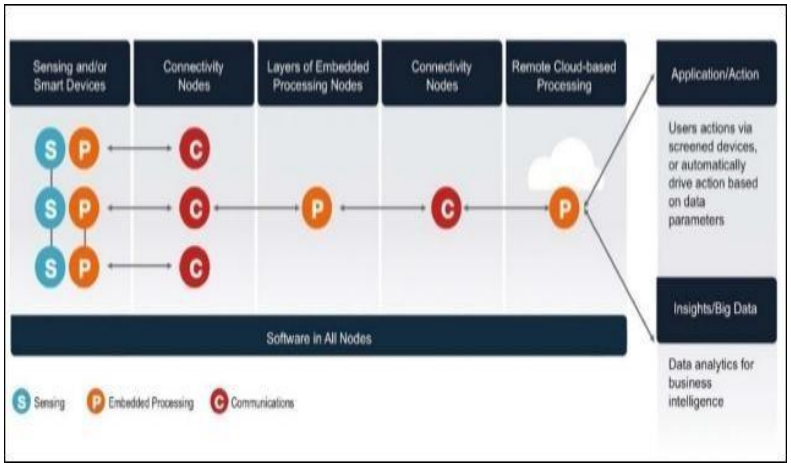

Figure 4.2: Basic Building Blocks of IoT

In a nutshell, the information gathered by the sensing node(end node) is processed first, then transmitted via connectivityto the embedded processing nodes, which can be any embedded hardware devices, and

processed there as well. It then passes through the networking nodes again, arriving at remote cloud-based processing, which can be any software, before being sent to the application node for proper application of the data collected as well as Internet of Things analysis.

4.3 IoT WORKS

It's easy to understand how the Internet of Things functions. In other words, sensors, RFID tags, and all other uniquely identifiable objects or "things" acquire real-time information (data) by virtue of acquiring basic resources (names, addresses, and so on) and related attributes of objects through automatic identification and perception technologies such as RFID, wireless sensor, and satellite positioning.

Second, it incorporates object-based information into the information network and realises intelligent indexing and aggregation of information related to masses of objects by relying on fundamental resource resources, thanks to a variety of communications technologies (similar to the resolution, addressing and discovery of the internet).

Finally, it analyses and processes information related to masses of objects using intelligent computing technologies like cloud computing, fuzzy recognition, data mining, and semantic analysis in order to achieve intelligent decision and control in the physical world.

Let's take a look at the diagram below. Layers of IoT is shown in Figure 4.3

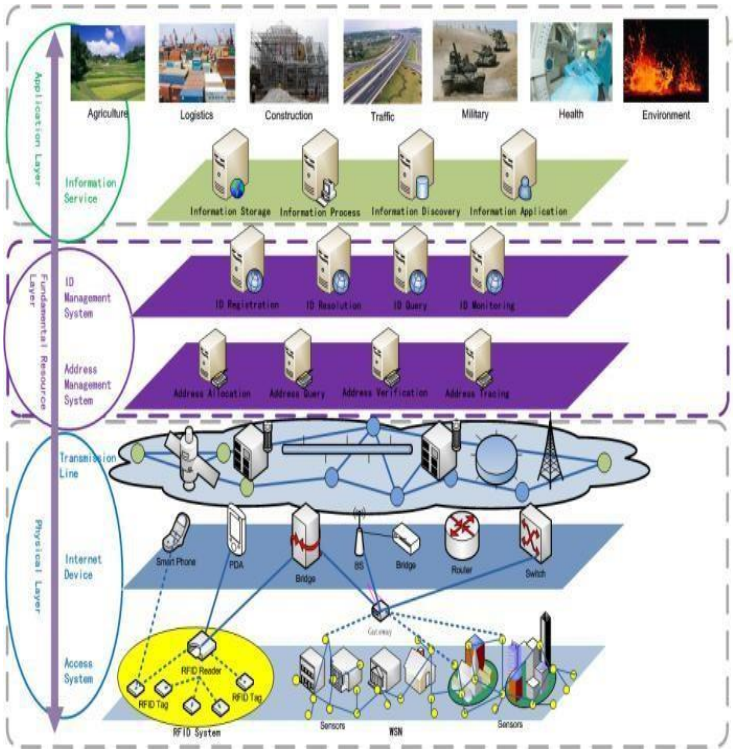

Figure 4.3: Layers of the Iot

All data collected by the access system (uniquely recognizable "things") is collected in the Physical layer and sent to internet devices (like smartphones). The data is then sent to the management layer through transmission lines (such as fiber-optic cable), where it is separated from the raw data and handled separately (stream analytics and data analytics). The application layer receives all of the managed data and uses it appropriately.

4.4 IoT ARCHITECTURE LAYERS

There are four major layers.

The Sensors and Connectivity network, which collects data, is at the very bottom of the IoT architecture. The Gateway and Network Layer are the next two layers. Over that, there's the management service layer, and finally,

there'sthe application layer, where the data is processed to meet the needs of different applications. IoT Architecture Layers is Shown in Figure 4.4

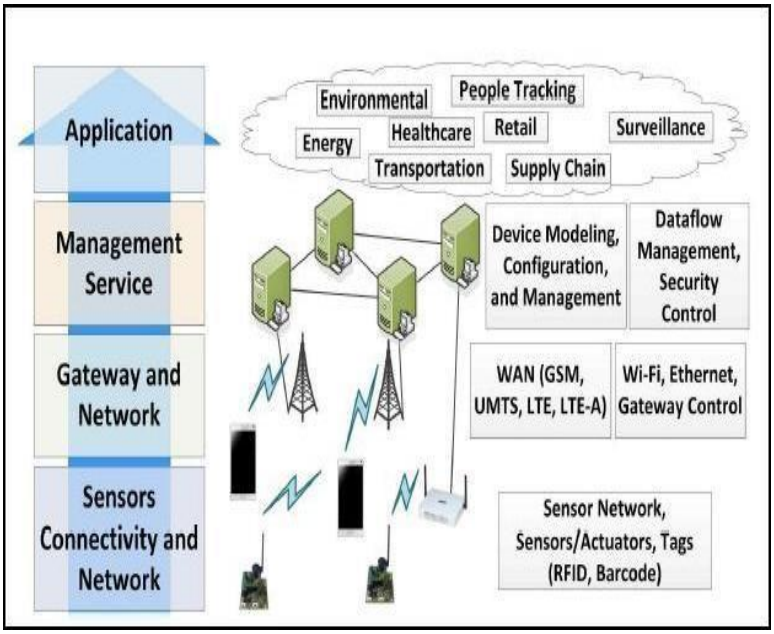

Figure 4.4: IoT Architecture Layers

Let's look at the characteristics of each of these architectural layers individually.

Layer of Sensors, Connectivity, and Networking

- RFID tags and sensors are included in this layer (which are an essential part of an IoT system and are responsible for collecting raw data). These are the ‖things‖ that make up an IoT scheme.
- Wireless Sensor Networks are made up of sensors, RFID tags, and other wireless devices (WSN).
- Since sensors are active in nature, they must capture and process data in real time.
- This layer also includes network access (such as WAN, PAN, and others) for transmitting raw data to the next layer, the Gateway and Network Layer.

The computers that make up a WSN have limited storage space, limited communication bandwidth, and slow processing speeds. We have various sensors for various applications, such as a temperature sensor for collecting temperature data, a water quality sensor for analysing water quality, and a moisture sensor for measuring the moisture content of the atmosphere or soil, among others.

The tags, which are RFID tags or barcode readers, are at the bottom of this layer, followed by sensors/actuators, and finally communication networks, as shown in Figure 4.5.

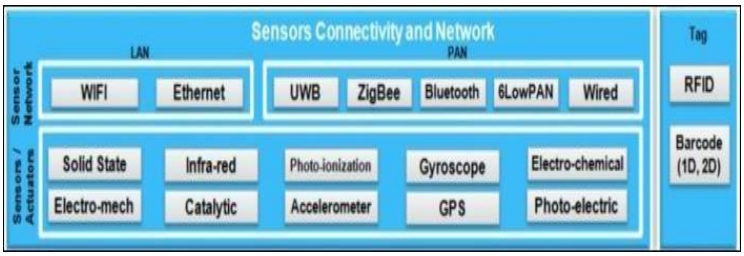

Figure 4.5: Sensor, Connectivity and Network layer

Gateway and Network Layer

Gateways are in charge of routing data from the Sensor, Connectivity, and Network layers and passing it on to the next layer, the Management Service Layer.

This layer necessitates a huge storage space in order to store the massive amounts of data generated by sensors, RFID tags, and other devices. This layer must also provide consistent, reliable performance across public, private, and hybrid networks.

Various types of network protocols are used by different IoT devices. Both of these protocols must be consolidated into a single layer. This layer is in charge of integrating different network protocols.

The gateway, which is comprised of the embedded OS, Signal Processors and Modulators, Micro-Controllers, and other components, is shown at the bottom of the diagram. The Gateway Networks, which include LAN (Local Area Network), WAN (Wide Area Network), and others, are

located above the gateway.

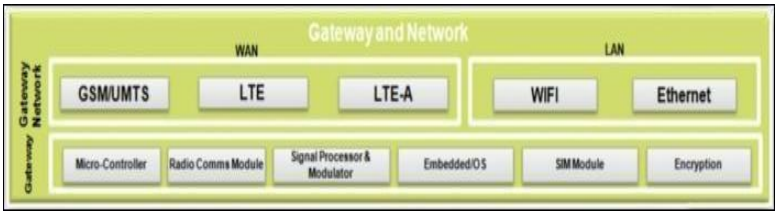

Figure 4.6: Gateway and Network layer

Management Service Layer

This layer is used to manage Internet of Things (IoT) services. Security Analysis of IoT devices, Information Analysis (Stream Analytics, Data Analytics), and Device Management are all handled by the management Service layer. Data management is needed to extract the appropriate information from the massive amounts of raw data obtained by sensor devices in order to produce a useful result from all of the data. This layer is where this operation is carried out.

In addition, a specific circumstance necessitates an urgent response. By abstracting data, extracting knowledge, and managing the data flow, this layer aids in this endeavour.

This layer is also in charge of data mining, text mining, and service analytics, among other things.

The management service layer has Operational Support Service (OSS), which includes Device Modeling, Device Configuration and Management, and more, as shown in the diagram below. The Billing Support System (BSS) also assists in billing and reporting.
We can also see from the diagram that there are IoT/M2M Application Services, which include Analytics Platform; Data – which is the most important part; Security, which includes Access Controls, Encryption, Identity Access Management, and so on; and finally, BRM and BPM (BPM). Management service layer is shown in Figure 4.7

Figure 4.7: Management Service layer

Application Layer

The application layer is the topmost layer of IoT architecture, and it is in charge of making efficient use of the data collected.

- Home automation, e-health, e-government, and other IoT applications are only a few examples.
- As seen in the diagram below, there are two types of IoT applications: horizontal market applications such as fleet management and supply chain, and sector-specific applications such as oil, healthcare, and transportation. Application layer is shown in Figure 4.8

Figure 4.8: Application Layer

Smart Environment Application Domains

	Smart Home	Smart Office	Smart Retail	Smart City	Smart Agriculture	Smart Energy & Fuel	Smart Transportation	Smart Military
Network Size	Small	Small	Small	Medium	Medium /Large	Large	Large	Large
Network Connectivity	WPAN, WLAN, 3G, 4G, Internet	WPAN, WLAN, 3G, 4G, Internet	RFID, NFC, WPAN, WLAN, 3G, 4G, Internet	RFID, NFC, WLAN, 3G, 4G, Internet	WLAN, Satellite Comm., Internet	WLAN, 3G, 4G, Microwave links, Satellite Comm.,	WLAN, 3G, 4G, Satellite Comm.	RFID, NFC, WPAN, WLAN, 3G, 4G, Satellite Comm.
Bandwidth Requirement	Small	Small	Small	Large	Medium	Medium	Medium~Large	Medium~Large

Figure 4.9: Smart Environment Application domains in which

WLAN stands for Wireless Local Area Network, which encompasses Wi-Fi, WAVE, IEEE 802.11 a/b/g/p/n/ac/ad, and other wireless technologies.

Bluetooth, ZigBee, 6LoWPAN, IEEE 802.15.4, UWB, and other wireless personal area networks are examples of WPAN.

Service Domain	Services
Smart Home	Entertainment, Internet Access
Smart Office	Secure File Exchange, Internet Access, VPN, B2B
Smart Retail	Customer Privacy, Business Transactions, Business Security, B2B, Sales & Logistics Management
Smart City	City Management, Resource Management, Police Network, Fire Department Network Transportation Management, Disaster Management
Smart Agriculture	Area Monitoring, Condition Sensing, Fire Alarm, Trespassing
Smart Energy & Fuel	Pipeline Monitoring, Tank Monitoring, Power Line Monitoring, Trespassing & Damage Management
Smart Transportation	Road Condition Monitoring, Traffic Status Monitoring, Traffic Light Control, Navigation Support, Smart Car Support, Traffic Information Support, ITS (Intelligent Transportation System)
Smart Military	Command & Control, Communications, Sensor Network, Situational Awareness, Security Information, Military Networking

Figure 4.10: Smart Environment application domains: Service domain and their servicesclassified

4.5 IOT PROCESSORS

In the Internet of Things (IoT) ecosystem, processors act as the brain of devices by managing data collection, processing, communication, and control. The choice of processor depends on factors such as computing power, energy efficiency, and application requirements.

IoT processors are mainly of two types: Microcontrollers (MCUs) and Microprocessors (MPUs**).** Microcontrollers are compact, low-power chips that integrate a CPU, memory, and input/output ports on a single board. They are ideal for simple tasks like sensing and controlling devices. A popular example is the Arduino**,** which uses processors such as the ATmega328P. Arduino boards are easy to program, affordable, and widely used in applications like home automation, sensor networks, and wearables.

Microprocessors, in contrast, are more powerful and capable of running full operating systems. They are suitable for advanced tasks such as multimedia processing, machine learning, and real-time analytics. The Raspberry Pi**,** built on ARM Cortex-A series processors, is a well-known microprocessor-based

board. It functions like a small computer and is used in projects like IoT gateways, smart surveillance, and industrial automation.

Together, MCUs and MPUs complement each other in IoT systems: microcontrollers handle simple, energy-efficient sensing and control tasks, while microprocessors manage complex processing and communication with the cloud.

4.5.1 ARDUINO

INTRODUCTION

In the world of the Internet of Things (IoT), choosing the right development platform is crucial for building efficient, reliable, and scalable systems. Two of the most popular platforms used by engineers, researchers, and hobbyists are **Arduino** and **Raspberry Pi**. Both play significant roles in IoT projects, but they differ in architecture, capabilities, and applications.

Arduino UNO Board

The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converters. Arduino Uno is shown in Figure 4.11

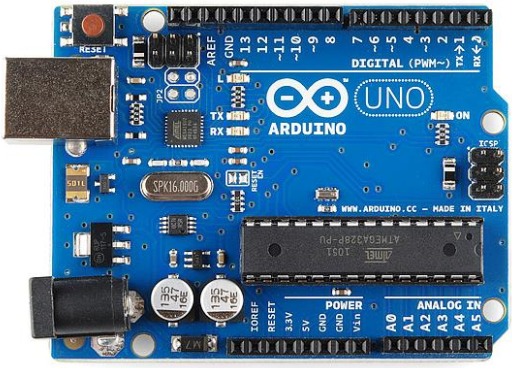

Figure 4.11**:** Arduino Uno

Arduino Architecture:

Arduino's processor basically uses the Harvard architecture where the program code and program data have separate memory. It consists of two memories- Program memory and the data memory.The code is stored in the flash program memory, whereas the data is stored in the data memory. The Atmega328 has 32 KB of flash memory for storing code (of which 0.5 KB is used for the bootloader), 2 KB of SRAM and 1 KB of EEPROM and operates with a clock speed of 16MHz. Architecture of Arduino UNO is shown in Figure 4.12

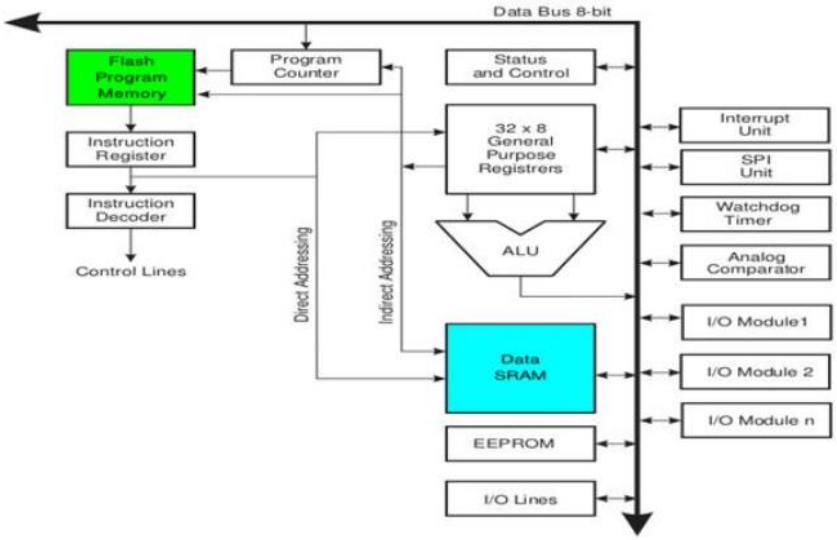

Figure 4.12: Architecture of Arduino UNO

Arduino UNO Pin Diagram:

A typical example of Arduino board is Arduino Uno. It consists of ATmega328- a 28 pin microcontroller. Pin diagram of Arduino Uno is shown in Figure 4.13

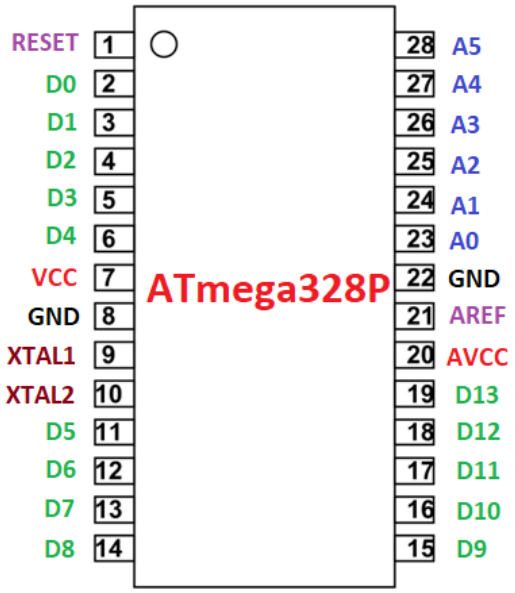

Figure 4.13: Pin Diagram of Arduino UNO

Digital Pin 0 (RX): Digital pin 0 serves as the UART receive (RX) pin. It is used to receive serial data from external devices, such as computers or other microcontrollers, and can also act as a general digital input when not used for serial communication.

Digital Pin 1 (TX): Digital pin 1 is the UART transmit (TX) pin. It sends serial data from the Arduino to external devices or computers. When not used for serial communication, it can function as a standard digital output.

Digital Pin 2: Digital pin 2 is a general-purpose input/output pin. It can read signals from buttons, switches, or sensors, and it can also control devices like

LEDs or relays. This pin can also serve as an external interrupt pin for detecting events.

Digital Pin 3: Pin 3 is a general-purpose digital pin that supports **PWM (Pulse Width Modulation)**. PWM allows it to simulate analog output, controlling motor speed, LED brightness, or audio signals. It can also be used as a standard digital input or output.

Digital Pin 4: Pin 4 is a general-purpose digital pin used for input or output. It can interface with buttons, sensors, LEDs, and other low-power devices.

Digital Pin 5: Pin 5 is a general-purpose digital pin with PWM capabilities. It is commonly used for controlling motor speed, dimming LEDs, or generating variable signals, along with normal digital input/output operations.

Digital Pin 6: Pin 6 is a digital pin that supports PWM. It is used for tasks like motor control, LED brightness modulation, and other applications requiring variable output signals.

Digital Pin 7: Pin 7 is a simple general-purpose digital I/O pin. It can read sensor inputs or drive output devices like LEDs, buzzers, or relays.

Digital Pin 8: Pin 8 is a general-purpose digital pin for input or output operations. It can detect button presses, read sensor signals, or drive devices like LEDs.

Digital Pin 9: Pin 9 is a digital pin with PWM capability. It can be used to control motor speed, LED brightness, or other devices requiring analog-like output.

Digital Pin 10: Pin 10 serves as a general-purpose digital pin and can also act as a **PWM output**. Additionally, it is often used as the **SS** (Slave Select) pin when using SPI communication with other devices.

Digital Pin 11: Pin 11 is a digital pin that supports PWM and SPI communication as MOSI (Master Out Slave In)**.** It can send data to SPI devices like displays or ADCs and also act as a regular digital output.

Digital Pin 12: Pin 12 is a general-purpose digital pin that can serve as SPI MISO (Master In Slave Out) when communicating with SPI devices. It can also be used as a regular input or output pin.

Digital Pin 13: Pin 13 is a digital I/O pin often connected to the onboard LED. It is widely used for testing and simple LED control. This pin can also interface with external circuits or sensors.

Analog Pin A0 – A5: Pins A0 through A5 serve as analog input pins, allowing the Arduino to read varying voltage levels from sensors, such as potentiometers, temperature sensors, or light sensors. These pins can also be used as digital I/O pins if needed.

Power Pins: The Arduino Uno includes several power pins. 3.3V and 5V pins supply power to sensors and modules. GND pins provide ground references, and the **VIN pin** allows an external power source to supply the board.

AREF Pin: The AREF (Analog Reference) pin is used to provide an external reference voltage for the analog inputs. It allows precise voltage measurements from sensors.

RESET Pin: The RESET pin can reset the microcontroller when pulled LOW. It is often used for restarting the board programmatically or manually.

Applications of Arduino

Arduino finds its applications in various fields due to their ability to perform different things. Let us see some of its applications:

- Arduinos are used in 3D printing where they perform the task of selecting how the printing will be performed.
- Arduinos are used for creating basic designs by makers, designers, hackers, and creators across the globe to create some great projects. Some

of the projects are Laser Turret Midi Controller, Retro Gaming With an OLED Display, and Traffic Light Controller.

- Arduinos are used by college students to understand programmable electronics and to explore their interest in programming.
- Arduinos are used in the field of robotics for programming robots and adding basic features like sensing and responding to environmental conditions.
- Arduino is used in IoT(Internet of Things) since it can collect information using sensors. The collected data is then processed and transmitted for developing various smart devices.

Use of Arduino Boards

Arduino board has been used for making different engineering projects and different applications. The Arduino software is very simple to use for beginners, yet flexible adequate for advanced users. It runs Windows, Linux, and Mac. Teachers and students in the schools utilize it to design low-cost scientific instruments to verify the principles of physics and chemistry. There are numerous other microcontroller platforms obtainable for physical computing. The Netmedia's BX-24, Parallax Basic Stamp, MIT's Handyboard, Phidget, and many others present related functionality.

Arduino also makes simpler the working process of microcontroller, but it gives some advantages over other systems for teachers, students, and beginners.

- Inexpensive
- Cross-platform
- The simple, clear programming environment
- Open source and extensible software
- Open source and extensible hardware

The Function of Arduino Board

The flexibility of the Arduino board is enormous so that one can do anything they imagine. This board can be connected very easily to different modules such as obstacle sensors, presence detectors, fire sensors, GSM Modules GPS modules, etc. The main function of the Arduino board is to control electronics

through reading inputs & changing it into outputs because this board works like a tool. This board is also used to make different electronics projects in the field of electronics, electrical, robotics, etc..

Different Types of Arduino Boards

The list of Arduino boards includes the following such as

- Arduino Uno (R3)
- Arduino Nano
- Arduino Micro
- Arduino Due
- LilyPad Arduino Board
- RedBoard Arduino Board
- Arduino Mega (R3) Board etc..

Arduino Nano

This is a small board based on the microcontrollers like ATmega328P otherwise ATmega628 but the connection of this board is the same as to the Arduino UNO board. This kind of microcontroller board is very small in size, sustainable, flexible, and reliable. Arduino Nano is shown in Figure 4.14

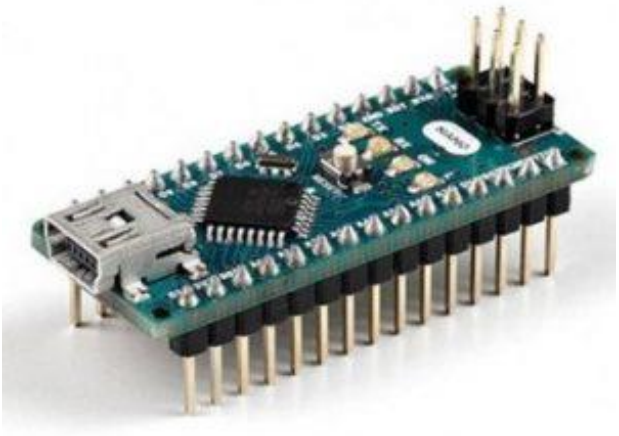

Figure 4.14: Arduino Nano

Arduino Nano Architecture

An Arduino Nano is a very tiny and simple microcontroller.
Specifications:

- Microcontroller: ATmega328
- Operating Voltage: 5 V
- Flash Memory: 32 KB of which 2 KB used by bootloader
- SRAM: 2 KB (Static Random Access Memory)
- Clock Speed: 16 MHz
- Analog IN Pins: 8 (10 bits of resolution, from ground to 5 volts)
- EEPROM: 1 KB (Electrically Erasable Programmable Read-Only Memory)
- Digital I/O Pins: 22 (6 of which are PWM)
- PWM Output: 6 (Pulse with Modulation)
- Power Consumption: 19 mA

As any other microcontroller, Arduino Nano has a set of GPIO pins (General Purpose Input(Output pins) that we can use to “control” external sensors.

Our Arduino Nano has 14 digital pins that can be used as an input or output. They operate at 5 volts. However, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX) to receive (RX) and transmit (TX) serial data.
- PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM (Pulse With Modulation) output.
- Internal LED (13): There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it’s off.

The Arduino Nano also has 8 analog inputs (to convert a voltage level into a digital value that can be stored and processed in the Arduino Nano).

Arduino Nano Pinout

Arduino nano pin configuration is shown below and each pin functionality is discussed in Figure 4.15

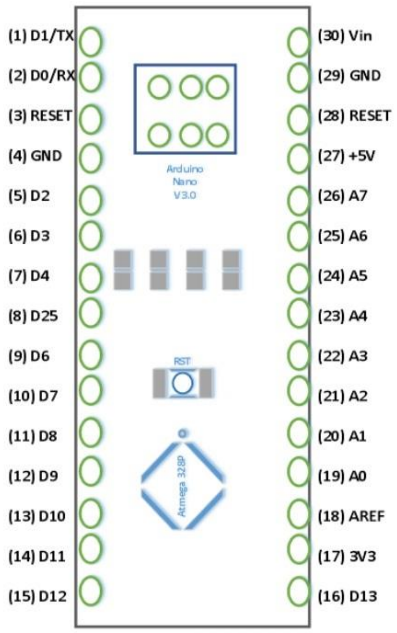

Figure 4.15: Arduino-nano-pinout

Digital Pin D0 (RX): Digital pin D0 serves as the UART receive (RX) pin. It receives serial data from external devices, such as computers or other microcontrollers, and can also function as a general-purpose input when serial communication is not in use.

Digital Pin D1 (TX): Digital pin D1 is the UART transmit (TX) pin, sending serial data from the Arduino Nano to external devices or a computer. When not used for UART, it can act as a standard digital output.

Digital Pin D2: Pin D2 is a general-purpose digital I/O pin. It can read input signals from sensors, switches, or buttons, or act as an output to control LEDs, buzzers, or relays. It also supports external interrupts for detecting events.

Digital Pin D3: Pin D3 is a digital pin with PWM (Pulse Width Modulation) capability. PWM allows the pin to simulate analog output, making it suitable for controlling motor speed, LED brightness, or audio signals, in addition to general digital input/output use.

Digital Pin D4: Pin D4 is a standard digital input/output pin, useful for connecting switches, LEDs, sensors, or other simple digital devices.

Digital Pin D5: Pin D5 supports PWM and general digital I/O. It is often used for applications that require variable output, such as dimming LEDs or controlling motor speed.

Digital Pin D6: Pin D6 is a PWM-enabled digital pin. It can generate analog-like outputs for LEDs, motors, or tone generation, along with general digital input/output tasks.

Digital Pin D7: Pin D7 is a general-purpose digital I/O pin, suitable for reading sensors, controlling relays, or signaling LEDs.

Digital Pin D8: Pin D8 functions as a standard digital pin for input or output operations, such as reading a button press or turning on an LED.

Digital Pin D9: Pin D9 supports PWM, making it ideal for controlling LED brightness or motor speed. It also functions as a normal digital input/output pin.

Digital Pin D10: Pin D10 serves as a digital I/O pin with PWM capability. It is commonly used as the **SS (Slave Select)** pin for SPI communication when connecting SPI devices.

Digital Pin D11: Pin D11 is a digital pin with PWM capability and serves as **MOSI (Master Out Slave In)** for SPI communication. It can transmit data to SPI devices such as displays or ADCs.

Digital Pin D12: Pin D12 is a general-purpose digital I/O pin and also functions as MISO (Master In Slave Out) for SPI communication, receiving data from SPI devices.

Digital Pin D13: Pin D13 is a digital I/O pin connected to the onboard LED. It is widely used for testing, LED control, or signaling purposes.

Analog Pins A0 – A7: Pins A0 through A7 serve as analog inputs, allowing the Arduino Nano to read varying voltages from sensors like potentiometers, temperature sensors, or light sensors. They can also function as digital input/output pins if required.

Power Pins: The Nano provides 5V and 3.3V power pins for powering sensors or modules, multiple GND pins for ground reference, and a VIN pin for supplying external voltage to the board.

AREF Pin: The AREF (Analog Reference) pin allows connection of an external reference voltage for more precise analog input measurements.

RESET Pin: The RESET pin can restart the microcontroller when pulled LOW, useful for resetting the program either manually or programmatically.

Special Pins: Some pins on the Nano, like D2 and D3, support external interrupts, allowing the board to respond to hardware events immediately. Additionally, pins D3, D5, D6, D9, D10, and D11 support PWM output for analog-like control.

Applications of Arduino Nano

These boards are used to build Arduino Nano projects by reading inputs of a sensor, a button, or a finger and gives an output by turning motor or LED ON, or and some of the applications are listed below.

- Samples of electronic systems & products
- Automation
- Several DIY projects
- Control Systems
- Embedded Systems
- Robotics & Instrumentation

Arduino Micro

The Arduino Micro is a microcontroller board based on the ATmega32u4 (datasheet), developed in conjunction with Adafruit. It has 20 digital input/output pins (of which 7 can be used as PWM outputs and 12 as analog inputs), a 16 MHz crystal oscillator, a micro USB connection, an ICSP header,

and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a micro USB cable to get started. It has a form factor that enables it to be easily placed on a breadboard. The Micro is similar to the Arduino Leonardo in that the ATmega32u4 has built-in USB communication, eliminating the need for a secondary processor. This allows the Micro to appear to a connected computer as a mouse and keyboard, in addition to a virtual (CDC) serial / COM port. It also has other implications for the behavior of the board. Arduino Micro is shown in Figure 4.16

Figure 4.16: Arduino Micro

Arduino Micro Architecture

The Arduino Micro is a compact microcontroller board based on the ATmega32U4 chip, which integrates a microprocessor, memory, and peripherals on a single package. At its core, the ATmega32U4 is an 8-bit AVR RISC microcontroller that operates at 16 MHz**,** providing sufficient processing power for a wide range of embedded and IoT applications. The architecture includes 32 KB of Flash memory for storing the user program, 2.5 KB of SRAM for runtime data, and 1 KB of EEPROM for non-volatile storage. The microcontroller features multiple digital and analog I/O pins, including 20 digital pins and 12 analog input pins, some of which support PWM (Pulse Width Modulation) for analog-like output control, useful in controlling motors, LEDs, or audio signals.

The Arduino Micro integrates communication peripherals such as UART (serial), SPI, and I2C, allowing seamless interfacing with sensors, displays, and other microcontrollers. Unlike other Arduino boards, the Micro has native

USB support, enabling it to act as a USB device like a keyboard, mouse, or MIDI controller. The board's power architecture allows operation via the USB port or an external voltage source through the VIN pin, with onboard regulators providing stable 5V and 3.3V outputs for external components. Architecture of Arduino Micro is shown in Figure 4.17

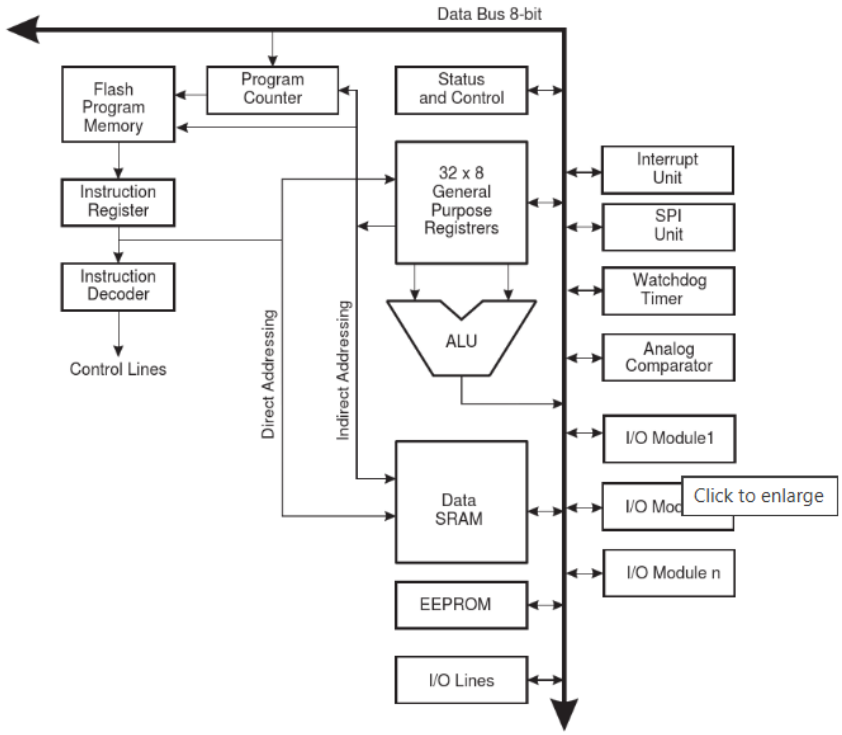

Figure 4.17: Architecture of Arduino Micro

Ground pins and multiple power pins are included to ensure proper current flow and stable operation of attached peripherals.

The internal architecture of the ATmega32U4 also incorporates timers, counters, analog-to-digital converters (ADC), and interrupt capabilities, which enable precise control of hardware and responsive interaction with external events. The layout of the Arduino Micro is designed for compactness, yet it provides easy access to all pins for prototyping on a breadboard or integrating into embedded systems. Overall, the Arduino Micro's architecture combines processing capability, flexible I/O, communication interfaces, and USB

integration**,** making it ideal for small-scale IoT projects, wearable devices, and interactive electronics. Pin Diagram of Arduino Micro is shown in Figure 4.18

Pin Diagram of Arduino Micro:

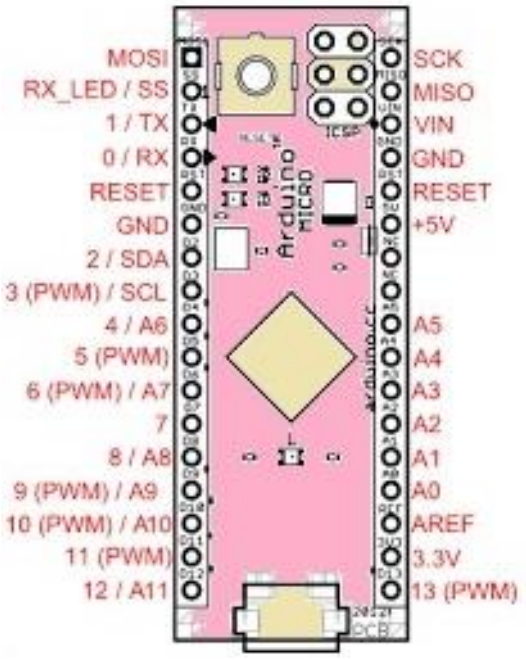

Figure 4.18: Pin Diagram of Arduino Micro

Digital Pin D0 (RX): Pin D0 serves as the UART receive (RX) pin, allowing the Arduino Micro to receive serial data from external devices, such as computers or other microcontrollers. It can also be used as a standard digital input pin when serial communication is not in use.

Digital Pin D1 (TX): Pin D1 is the UART transmit (TX) pin, sending serial data to computers or other devices. When not used for UART, it functions as a general-purpose digital output.

Digital Pin D2: Pin D2 is a general-purpose digital I/O pin that also supports external interrupts. It can read input from buttons or sensors and control output devices such as LEDs, buzzers, or relays.

Digital Pin D3: Pin D3 supports both general-purpose digital I/O and PWM **(Pulse Width Modulation)** output. It is useful for dimming LEDs, controlling motor speed, or generating audio signals.

Digital Pin D4: Pin D4 is a digital I/O pin that can function as input or output for connecting switches, LEDs, or sensors.

Digital Pin D5: Pin D5 is a digital I/O pin with PWM capability, allowing analog-like output control for LEDs, motors, or other devices.

Digital Pin D6: Pin D6 also supports PWM output in addition to general digital I/O functions. It is used for tasks that require variable voltage simulation, such as controlling LED brightness.

Digital Pin D7: Pin D7 is a simple digital input/output pin, suitable for reading switches, controlling relays, or signaling LEDs.

Digital Pin D8: Pin D8 is a standard digital I/O pin for connecting input devices like buttons or output devices such as LEDs.

Digital Pin D9: Pin D9 supports PWM and general-purpose digital I/O, making it suitable for motor or LED control.

Digital Pin D10: Pin D10 is a digital I/O pin with PWM capability and is commonly used as the SS (Slave Select) pin for SPI communication when connecting SPI devices.

Digital Pin D11: Pin D11 supports PWM and functions as MOSI (Master **Out** Slave In) for SPI communication. It can transmit data to SPI devices like displays or ADCs.

Digital Pin D12: Pin D12 is a digital I/O pin and can also serve as MISO (Master In Slave Out) for SPI devices, receiving data from peripherals.

Digital Pin D13: Pin D13 is a digital I/O pin connected to the onboard LED, commonly used for testing and simple LED control.

Analog Pins A0 – A11: Pins A0 through A11 serve as analog inputs to read varying voltages from sensors such as potentiometers, temperature sensors, or light sensors. Many of these analog pins can also function as digital I/O pins, increasing flexibility in projects.

Power Pins: The Arduino Micro provides 5V and 3.3V power pins to supply sensors and modules. Multiple GND pins provide ground reference, and the VIN pin allows the board to be powered from an external voltage source.

AREF Pin: The AREF (Analog Reference) pin allows the connection of an external reference voltage for precise analog measurements.

RESET Pin: The RESET pin can restart the microcontroller when pulled LOW. This is useful for manually or programmatically restarting the Arduino Micro's program.

Special Pins: The Arduino Micro has native USB support, allowing it to act as a USB device (keyboard, mouse, or MIDI) in addition to standard GPIO functions. Certain pins like D2 and D3 support external interrupts, and pins D3, D5, D6, D9, D10, and D11 support PWM for analog-like control.

Arduino mega 2560

The microcontroller board like "Arduino Mega" depends on the ATmega2560 microcontroller. It includes digital input/output pins-54, where 16 pins are analog inputs, 14 are used like PWM outputs hardware serial ports (UARTs) – 4, a crystal oscillator-16 MHz, an ICSP header, a power jack, a USB connection, as well as an RST button. This board mainly includes everything which is essential for supporting the microcontroller. So, the power supply of this board can be done by connecting it to a PC using a USB cable, or battery or an AC-DC adapter. This board can be protected from the unexpected electrical discharge by placing a base plate.

The SCL & SDA pins of Mega 2560 R3 board connects to beside the AREF pin. Additionally, there are two latest pins located near the RST pin. One pin is the IOREF that permit the shields to adjust the voltage offered from the

Arduino board. Another pin is not associated & it is kept for upcoming purposes. These boards work with every existing shield although can adjust to latest shields which utilize these extra pins. Arduino Mega is shown in Figure 4.19

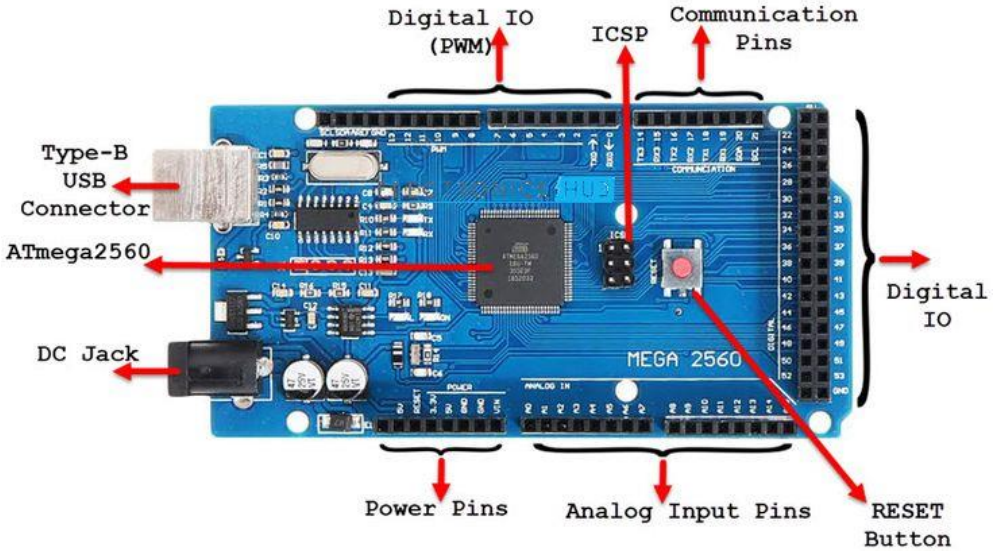

Figure 4.19: Arduino Mega 2560

Arduino Mega Architecture

The Arduino Mega 2560 is a high-performance microcontroller board designed for projects that require a large number of input/output pins and more memory. It is based on the ATmega2560 microcontroller, which is an 8-bit AVR RISC processor running at 16 MHz, offering significantly higher I/O capacity and memory than boards like the Arduino Uno. The ATmega2560 includes 256 KB of Flash memory for storing programs, 8 KB of SRAM for runtime data, and 4 KB of EEPROM for non-volatile data storage, making it suitable for complex programs with multiple sensors, actuators, and communication modules.

The board has 54 digital I/O pins, of which 15 support PWM (Pulse Width Modulation) for analog-like output control, and 16 analog input pins, allowing

precise sensor measurements. It includes multiple UART serial communication interfaces (four in total), enabling simultaneous communication with multiple serial devices like GPS modules, Bluetooth modules, and other microcontrollers. The Mega also supports SPI and I2C communication protocols, allowing high-speed data transfer to peripherals like displays, memory devices, and sensors.

Power management is an important aspect of the Arduino Mega architecture. The board can be powered via the USB port or an external voltage source (VIN)**,** with onboard voltage regulators providing stable 5V and 3.3V outputs for connected sensors and modules. Ground pins ensure proper electrical reference and safe current flow. The ATmega2560 microcontroller includes timers, counters, analog-to-digital converters (ADC), digital-to-analog converters (DAC), and interrupt capabilities, enabling precise timing, measurement, and real-time responses to external events.

The board's architecture also integrates a USB-to-serial interface, which simplifies programming and communication with a host computer. The layout of the Mega is designed for accessibility, with clearly labeled pins and multiple headers to facilitate breadboard prototyping or embedded system integration. Its combination of high memory, multiple communication ports, numerous I/O pins, and PWM outputs makes the Arduino Mega ideal for robotics, complex IoT systems, 3D printing control, and other advanced embedded applications.

Pin Diagram of Arduino mega 2560

Digital Pin 0 (RX0): Pin 0 serves as the UART receive (RX) line for serial communication. It receives data from external devices or a computer, and when not used for serial, it can function as a standard digital input.

Digital Pin 1 (TX0): Pin 1 is the UART transmit (TX) line. It sends serial data from the Arduino Mega to other devices or a host computer and can also act as a digital output pin when UART is not used.

Digital Pins 2–13: These are general-purpose digital I/O pins. Pins 2 and 3 support external interrupts**,** allowing the board to respond immediately to

external events. Pins 3, 5, 6, 9, 10, and 11 also support PWM output**,** enabling analog-like control for LEDs, motors, or other devices. Pins 2–13 can read inputs from switches, buttons, or sensors and drive output devices such as LEDs, buzzers, and relays. Pin Diagram of Arduino Mega is Shown in Figure 4.20

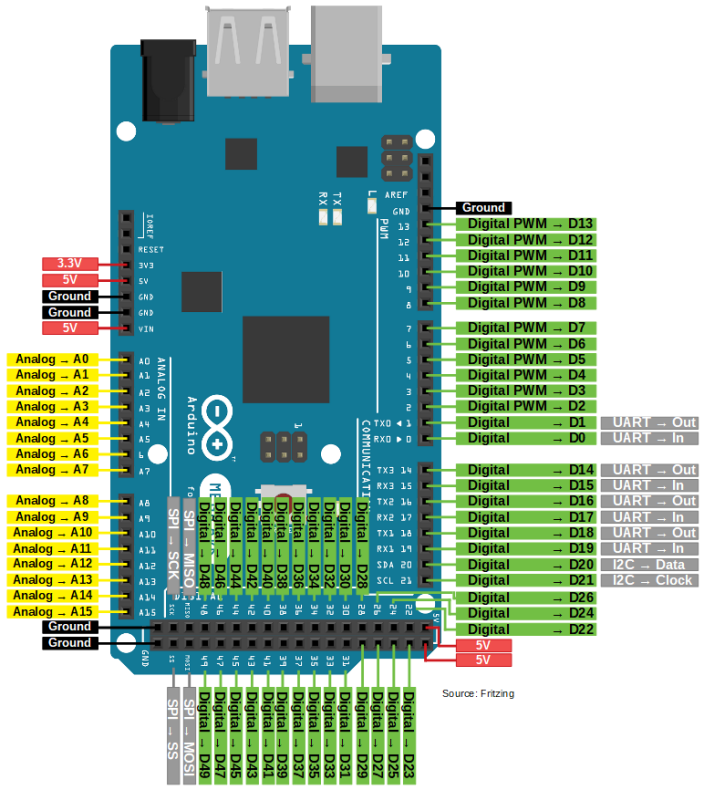

Figure 4.20: Pin Diagram of Arduino mega 2560

Digital Pins 14–15 (TX3/RX3): These pins serve as the third UART interface (Serial3), enabling communication with additional serial devices. They can also be used as general-purpose digital I/O pins.

Digital Pins 16–17 (TX2/RX2): These pins function as the second UART interface (Serial2), allowing connection to other serial peripherals while also serving as general digital I/O pins.

Digital Pins 18–19 (TX1/RX1): Pins 18 and 19 provide the first additional UART interface (Serial1) for multiple serial device communication. They can also be used for general-purpose digital I/O.

Digital Pins 20–21 (SDA/SCL): These pins are used for I2C communication**.** Pin 20 serves as the I2C data line (SDA), and Pin 21 is the I2C clock line (SCL), allowing the Mega to communicate with multiple I2C devices such as displays, sensors, and memory modules.

Digital Pins 22–53: These pins are standard general-purpose digital I/O pins. Several pins in this range support PWM output for controlling LEDs, motors, or generating analog-like signals. They are used extensively in projects requiring many inputs or outputs, like robotics or 3D printer control.

Analog Pins A0–A15: These pins read analog signals from sensors, converting voltages from 0–5V into digital values using the 10-bit ADC. They can also be used as digital I/O pins if needed. Analog pins are commonly used for sensors like potentiometers, temperature sensors, light sensors, or any device that outputs a varying voltage.

Power Pins: The Mega includes multiple 5V and 3.3V pins for powering external modules, GND pins for ground reference, and a VIN pin for supplying external voltage to the board. The RESET pin allows manual or programmatic restarting of the board.

AREF Pin: The Analog Reference (AREF) pin allows connection of an external voltage reference for more precise analog input readings.

Special Function Pins: Several pins support external interrupts, PWM output, SPI communication (pins 50–53 for MISO, MOSI, SCK, SS), and UART interfaces**.** This makes the Mega highly versatile for projects requiring multiple sensors, actuators, and communication channels.

Arduino Due

Arduino Due is the most powerful Arduino development board in the Arduino series. This Arduino board is a beginner board including many features with excellent processing speed, so used in advanced applications. This board was developed on an ARM series controller whereas other Arduino boards were developed based on an ATMEGA series controller. Arduino's due board is based on the 32-bit ARM core microcontroller. This board is available with 54 digital I/O pins where 12 pins are used as PWM o/ps, 12-analog inputs, UARTs -4, an 84 MHz CLK, DAC -2, TWI-2, an SPI header, a power jack, a JTAG header, an USB OTG connection, and a RESET button & can ERASE button. The Arduino Due board can be simply connected to any computer by a micro-USB cable & power through a battery or an AC-to-DC adapter to get started. This board is well-suited with all types of Arduino shields which work at 3.3V. Block diagram of Arduino Due is shown in Figure 4.21

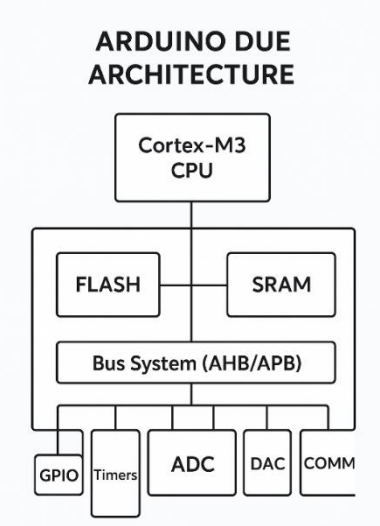

Figure 4.21: Block Diagram of Arduino DUE

Architectural Overview

The Arduino Due is built around the Atmel SAM3X8E microcontroller**,** which is based on the 32-bit ARM Cortex-M3 processor. This architecture provides significantly higher processing power compared to 8-bit Arduino boards,

operating at 84 MHz with a rich set of peripherals suitable for complex IoT and embedded applications. The SAM3X8E includes 512 KB of Flash **memory** for program storage, 96 KB of SRAM for runtime data, and a set of dedicated peripherals, including timers, counters, PWM controllers, ADCs (analog-to-digital converters), DACs (digital-to-analog converters), and communication interfaces. The board features 54 digital I/O pins, of which **12** support PWM**,** and 12 analog inputs, enabling precise control and measurement in sensor and actuator applications.

The Arduino Due architecture supports multiple communication protocols**,** including UART, SPI, and I2C, allowing seamless interaction with external devices like sensors, displays, and other microcontrollers. Unlike previous Arduino boards, the Due operates at 3.3V logic levels**,** which ensures compatibility with modern low-voltage modules but requires caution to avoid damage from 5V signals. Its USB interface serves both for programming and direct communication with a host computer, leveraging the ARM Cortex-M3's native USB controller.

Internally, the SAM3X8E microcontroller is organized around a Harvard architecture, with separate buses for instructions and data, allowing faster processing. The microcontroller also integrates advanced features such as nested vectored interrupt controllers (NVIC), multiple timers, and analog-to-digital and digital-to-analog converters, making the Arduino Due suitable for precise real-time control applications. Its compact board design provides easy access to all I/O pins while maintaining stability and high performance. Overall, the Arduino Due's architecture combines high-speed processing, extensive I/O capabilities, versatile communication interfaces, and native USB support, making it ideal for advanced embedded systems, robotics, IoT projects, and real-time sensor applications.

Specifications

The specifications of Arduino Due include the following.

- Microcontroller is SAM3X8E 32-bit ARM controller.
- The operating voltage is 3.3V.

- The maximum current throughout every I/O pin is 3mA and 15mA.
- The maximum current drawn from all I/O pins is 130mA.
- Flash memory is 512K bytes.
- 16Kbyte EEPROM.
- 96Kbytes Internal RAM.
- The internal Clock Frequency is 12 Mhz.
- The external Clock Frequency is 84 Mhz.
- Operating temperature ranges from -40ºC to +85ºC
- Recommended i/p voltage ranges from 7V to 12V.
- Input voltage ranges from 6 to 20V
- Digital I/O Pins – 54.
- Analog i/p pins – 12.
- Analog o/p Pins – 2.

Arduino Due Pin Configuration

The pin configuration of Arduino Due is shown Figure 4.22.

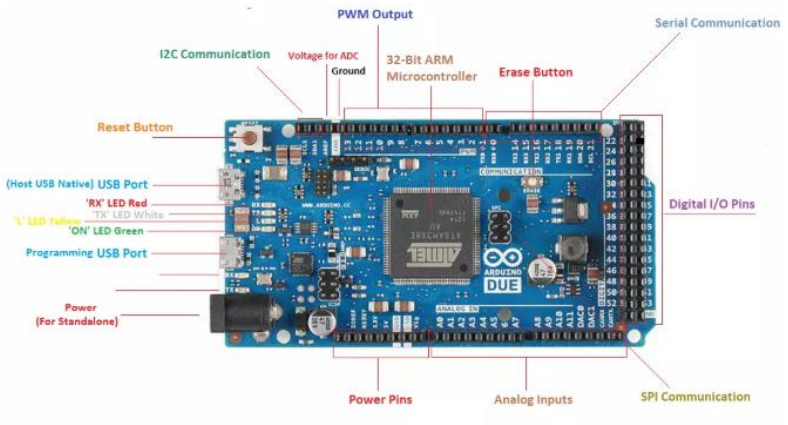

Figure 4.22: **Arduino Due**

Digital Pin 0 (RX0): Digital pin 0 on the Arduino Due serves as the UART receive (RX) pin for serial communication. It receives data from other

microcontrollers, computers, or serial devices. When not used for UART, it can act as a standard digital input pin.

Digital Pin 1 (TX0): Pin 1 functions as the UART transmit (TX) pin. It sends serial data from the Arduino Due to external devices or computers. This pin can also act as a general-purpose digital output when serial communication is not in use.

Digital Pin 2: Pin 2 is a general-purpose digital I/O pin. It can read input signals from buttons, switches, or sensors and output signals to LEDs, relays, or other actuators.

Digital Pin 3: Pin 3 is a digital pin that **s**upports PWM (Pulse Width Modulation)**.** It can be used to generate analog-like signals for controlling LED brightness, motor speed, or audio tones.

Digital Pin 4: Pin 4 is a standard digital input/output pin, suitable for connecting sensors, switches, or output devices.

Digital Pin 5: Pin 5 is a PWM-enabled digital pin, useful for tasks that require variable voltage output, such as controlling the speed of a DC motor or dimming LEDs.

Digital Pin 6: Pin 6 supports PWM output in addition to standard digital I/O functions. It can be used for analog-like control in IoT projects.

Digital Pin 7: Pin 7 is a general-purpose digital I/O pin, which can read digital inputs or control output devices.

Digital Pin 8: Pin 8 serves as a standard digital I/O pin for input or output tasks.

Digital Pin 9: Pin 9 supports PWM and general-purpose digital I/O, enabling fine control of motors, LEDs, or other actuators.

Digital Pin 10: Pin 10 is a digital I/O pin with PWM capability and is commonly used as SS (Slave Select) for SPI communication.

Digital Pin 11: Pin 11 supports PWM and acts as MOSI (Master Out Slave In) for SPI communication, transmitting data to SPI-enabled devices.

Digital Pin 12: Pin 12 is a digital I/O pin and serves as MISO (Master In **Slave** Out) for SPI devices, receiving data from peripherals.

Digital Pin 13: Pin 13 is a digital pin often connected to the onboard LED. It can be used for testing, signaling, or simple LED control.

Digital Pins 14 – 53: These pins function as general-purpose digital I/O pins, with several supporting PWM output. Pins 14–21 are also capable of interfacing with the SPI bus. They allow the Due to connect with a wide range of sensors, actuators, and modules for IoT or embedded applications.

Analog Input Pins A0 – A11: The Arduino Due provides 12 analog input pins that can read voltages from 0 to 3.3V, allowing precise measurement from sensors such as potentiometers, temperature sensors, or light sensors. These analog pins can also function as digital I/O pins if required.

Power Pins: The Arduino Due includes 3.3V and 5V power pins to supply external modules and sensors. Multiple GND pins provide grounding, while the VIN pin allows powering the board from an external voltage source. The Due operates natively at 3.3V logic, so care must be taken not to exceed voltage limits on its pins.

AREF Pin: The AREF (Analog Reference) pin allows an external reference voltage to be applied for analog inputs, improving measurement accuracy for sensors.

RESET Pin: The RESET pin can restart the microcontroller when pulled LOW, enabling manual or programmatic resetting of the Arduino Due.

Special Pins: Some pins support external interrupts, which allow the Arduino Due to respond immediately to external events. PWM-enabled pins enable analog-like control, while communication pins support UART, SPI, and I2C protocols for connecting multiple sensors, displays, and devices in complex IoT projects.

Features of Arduino Due:

- **High performance** → suitable for real-time data processing.
- **Multiple communication protocols** → connects to sensors, networks, and external devices.
- **Large memory** → allows complex programs.
- **Analog + Digital interfacing** → enables IoT devices to sense and control environments.
- **Low power modes** → suitable for portable/battery-operated IoT systems.

Applications in IoT

- Environmental monitoring systems.
- Industrial automation and control.
- Robotics and real-time control systems.
- Smart healthcare devices (sensor-based).
- IoT gateways when combined with communication modules (Wi-Fi, GSM, LoRa).

LilyPad Arduino Board

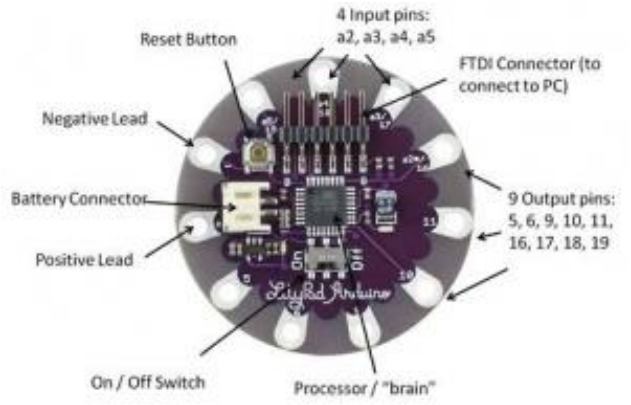

Figure 4.23: Lily Pad Arduino

The LilyPad Arduino is a specially designed microcontroller board for wearable electronics and e-textiles, developed by Leah Buechley and the MIT Media Lab. Its architecture is based on the ATmega328V microcontroller, which is an 8-bit AVR processor operating at 8 MHz, providing sufficient processing power for sensor interfacing, LED control, and simple signal processing in wearable projects. The LilyPad features 16 general-purpose input/output (GPIO) pins, some of which support pulse-width modulation (PWM) for controlling LED brightness or small motors, and 6 analog input pins, enabling the reading of sensors such as accelerometers, temperature sensors, or flex sensors. The board's low-profile design allows it to be sewn directly into fabric using conductive thread, which connects sensors, actuators, and power sources in a flexible, wearable format.

The power architecture of the LilyPad Arduino is optimized for wearable applications, supporting a 3.3V regulated supply from a small coin-cell or LiPo battery. Ground pins are included to complete electrical circuits safely. The microcontroller integrates standard features of AVR architecture, including timers, counters, analog-to-digital converters (ADC), and interrupt support, allowing responsive and precise control of connected devices. Communication interfaces such as I2C and UART are also available for

connecting additional modules like displays or wireless transmitters, although space constraints often limit extensive peripheral use in wearable designs.

The board's compact, circular form factor and large conductive pads make it easy to connect with conductive threads or snap-on modules, providing flexibility for innovative wearable designs. Its architecture combines **low-**power consumption, compact size, flexible I/O, and standard Arduino programming support, making it ideal for interactive clothing, soft robotics, health monitors, and other embedded textile applications. By leveraging the familiar Arduino IDE and standard libraries, designers can easily program the LilyPad to read sensor data, control actuators, and interact with the wearer in real time, all while maintaining a lightweight and flexible form factor suitable for textiles. Architecture of Lily Pad Arduino is shown in Figure 4.24

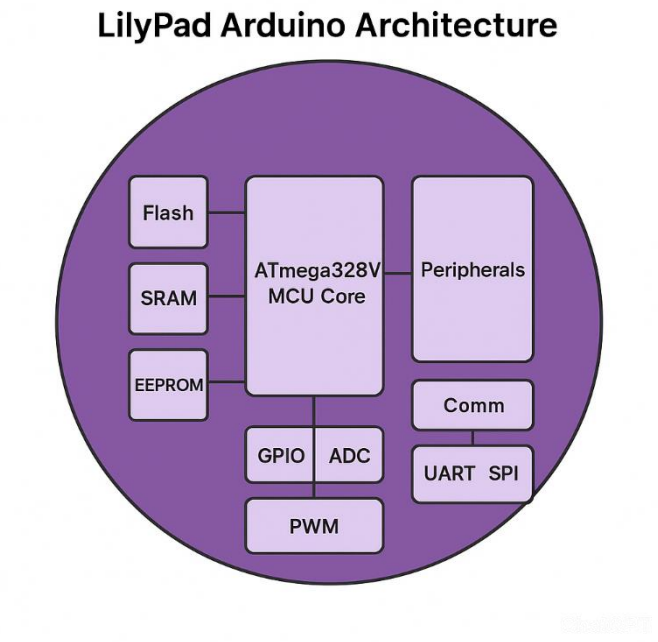

Figure 4.24: Architecture of Lily pad Arduino

Specifications

- **Microcontroller**: ATmega328V or ATmega168V (low-power AVR series).
- **Operating Voltage**: 2.7V – 5.5V (commonly powered by coin-cell batteries or LiPo).
- **Clock Speed**: 8 MHz (lower than Uno's 16 MHz, for power efficiency).
- **Digital I/O Pins**: 14 (6 with PWM).
- **Analog Inputs**: 6.
- **Flash Memory**: 16 KB (ATmega168) / 32 KB (ATmega328).
- **SRAM**: 1 KB (ATmega168) / 2 KB (ATmega328).
- **EEPROM**: 512 bytes (ATmega168) / 1 KB (ATmega328).
- **Unique Form Factor**: Circular board with sewable pads instead of pin headers.

Pin Diagram of Lily pad Arduino:

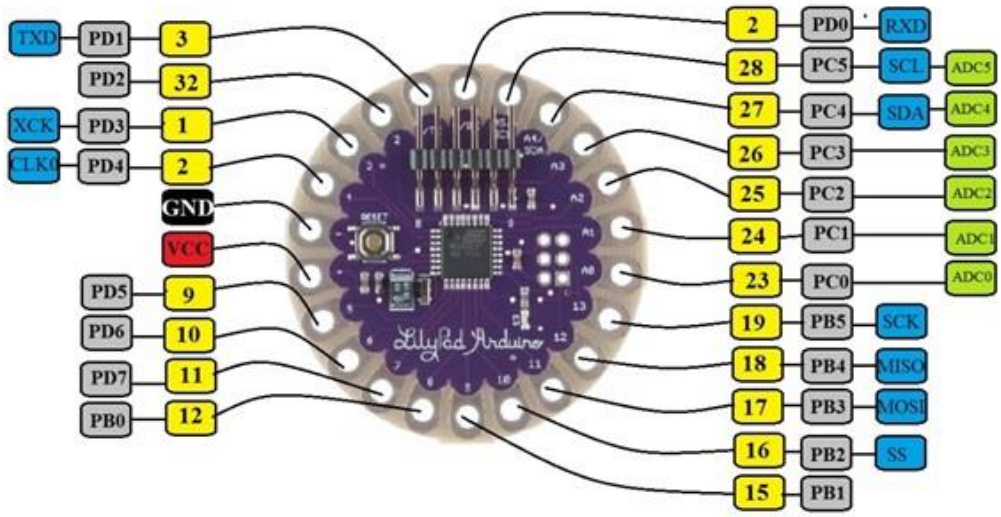

Figure 4.25: Pinout diagram of Lilypad Arduino

Power Pins: The LilyPad Arduino features a 3.3V regulated power pin**,** designed to power small sensors and actuators in wearable projects. It also

includes a GND (ground) pin**,** which provides a reference point for all electrical circuits and ensures safe current flow. The VCC pin can be connected to an external battery source, such as a LiPo or coin-cell battery, supplying voltage to the board and connected components.

Digital Pins 0–13: The LilyPad Arduino has 14 digital input/output pins**.** These pins are used to read digital signals from sensors or switches and to control output devices such as LEDs, buzzers, or small motors. Some of these digital pins also support PWM (Pulse Width Modulation)**,** which allows them to simulate analog output. This feature is useful for controlling LED brightness or motor speed in wearable projects. Pins 2 and 3 also support external interrupts, which allow the microcontroller to respond immediately to external events, such as button presses or sensor triggers.

Analog Input Pins A0–A5: The board includes 6 analog input pins. These pins can read varying voltages from analog sensors, such as potentiometers, temperature sensors, or light sensors. The analog-to-digital converter (ADC) of the ATmega328V microcontroller converts these analog signals into digital values that the Arduino can process. These pins can also be used as general-purpose digital I/O if needed.

RESET Pin: The RESET pin allows the LilyPad Arduino to be restarted when pulled LOW. This is useful for programmatically or manually restarting the board during development or in wearable applications.

AREF Pin: The AREF (Analog Reference) pin provides a reference voltage for the analog inputs. By connecting a stable voltage source to AREF, users can improve the accuracy of analog measurements from sensors.

Special Function Pins: Some pins on the LilyPad support serial communication (UART)**,** enabling communication with external devices such as Bluetooth modules or serial sensors. The microcontroller also has built-in timers and PWM channels, which are accessible through certain digital pins, allowing smooth control of actuators and LEDs.

Conductive Pads: Unlike standard Arduino boards, the LilyPad's pins are designed as large conductive pads, making it easy to sew the board directly into fabric using conductive thread. This allows multiple sensors, LEDs, and actuators to be connected in a flexible, wearable manner without traditional wiring.

General Notes: The LilyPad Arduino is designed for low-power, wearable applications**,** and all pins are optimized for 3.3V logic levels. Its compact, circular design and large sewing pads make it ideal for e-textiles, interactive clothing, soft robotics, and other embedded textile projects. Despite its small size, the LilyPad retains full Arduino programmability via the Arduino IDE, allowing users to easily control sensors, actuators, and wearable electronics.

Applications in IoT

The LilyPad Arduino is widely used in wearable IoT devices such as:

- Smart clothing with embedded LEDs and sensors.
- Fitness and health monitoring wearables (e.g., heart-rate, body temperature).
- Interactive fashion (light-up dresses, sound-reactive clothing).
- Assistive technology (wearables for visually/hearing-impaired).
- Educational projects (STEM kits for introducing IoT in fashion/healthcare).

Software and Coding

Introduction to Arduino IDE

Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programmed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

The key features are:

- Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.
- You can control your board functions by sending a set of instructions to the microcontroller on the board via Arduino IDE (referred to as uploading software).
- Unlike most previous programmable circuit boards, Arduino does not need an extra piece of hardware (called a programmer) in order to load a new code onto the board. You can simply use a USB cable.
- Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program.
- Finally, Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.

After learning about the main parts of the Arduino UNO board, we are ready to learn how to set up the Arduino IDE. Once we learn this, we will be ready to upload our program on the Arduino board.

Arduino data types: Data types in C refers to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in the storage and how the bit pattern stored is interpreted.

The following table provides all the data types that you will use during Arduino programming.

Void:

The void keyword is used only in function declarations. It indicates that the function is expected to return no information to the function from which it was called.

Example:

```
Void Loop ( )
{
```

```
// rest of the code
}
```

Boolean:

A Boolean holds one of two values, true or false. Each Boolean variable occupies one byte of memory.

Example:

Boolean state= false ; // declaration of variable with type boolean and initialize it with false.

Boolean state = true ; // declaration of variable with type boolean and initialize it with false.

Char: A data type that takes up one byte of memory that stores a character value. Character literals are written in single quotes like this: 'A' and for multiple characters, strings use double quotes: "ABC".

However, characters are stored as numbers. You can see the specific encoding in the ASCII chart. This means that it is possible to do arithmetic operations on characters, in which the ASCII value of the character is used. For example, 'A' + 1 has the value 66, since the ASCII value of the capital letter A is 65.

Example:

Char chr_a = ‘a’ ;//declaration of variable with type char and initialize it with character a.

Char chr_c = 97 ;//declaration of variable with type char and initialize it with character 97

Unsigned char:

Unsigned char is an unsigned data type that occupies one byte of memory. The unsigned char data type encodes numbers from 0 to 255.

Example:

Unsigned Char chr_y = 121 ; // declaration of variable with type Unsigned char and initialize it with character y

Byte:

A byte stores an 8-bit unsigned number, from 0 to 255.

Example:

byte m = 25 ;//declaration of variable with type byte and initialize it with 25

int:

Integers are the primary data-type for number storage. **int** stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).

The **int** size varies from board to board. On the Arduino Due, for example, an **int** stores a 32-bit (4-byte) value. This yields a range of -2,147,483,648 to 2,147,483,647 (minimum value of -2^31 and a maximum value of (2^31) - 1).

Example:

int counter = 32 ;// declaration of variable with type int and initialize it with 32.

Unsigned int:

Unsigned ints (unsigned integers) are the same as int in the way that they store a 2 byte value. Instead of storing negative numbers, however, they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1). The Due stores a 4 byte (32-bit) value, ranging from 0 to 4,294,967,295 (2^32 - 1).

Example:

Unsigned int counter= 60 ; // declaration of variable with type unsigned int and initialize it with 60.

Word:

On the Uno and other ATMEGA based boards, a word stores a 16-bit unsigned number. On the Due and Zero, it stores a 32-bit unsigned number.

Example

word w = 1000 ;//declaration of variable with type word and initialize it with 1000.

Long:

Long variables are extended size variables for number storage, and store 32 bits (4 bytes), from 2,147,483,648 to 2,147,483,647.

Example:

Long velocity= 102346 ;//declaration of variable with type Long and initialize it with 102346

Unsigned long: Unsigned long variables are extended size variables for number storage and store 32 bits (4 bytes). Unlike standard longs, unsigned longs will not store negative numbers, making their range from 0 to 4,294,967,295 (2^32 - 1).

Example:

Unsigned Long velocity = 101006 ;// declaration of variable with type Unsigned Long and initialize it with 101006.

Short:

A short is a 16-bit data-type. On all Arduinos (ATMega and ARM based), a short stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).

Example:

short val= 13 ;//declaration of variable with type short and initialize it with 13

Float:

Data type for floating-point number is a number that has a decimal point. Floating-point numbers are often used to approximate the analog and continuous values because they have greater resolution than integers.

Floating-point numbers can be as large as 3.4028235E+38 and as low as 3.4028235E+38. They are stored as 32 bits (4 bytes) of information.

Example:

float num = 1.352;//declaration of variable with type float and initialize it with 1.352.

Double:

On the Uno and other ATMEGA based boards, Double precision floating-point number occupies four bytes. That is, the double implementation is exactly the same as the float, with no gain in precision. On the Arduino Due, doubles have 8-byte (64 bit) precision.

Example:

double num = 45.352 ;// declaration of variable with type double and initialize it with 45.352.

In this section, we will learn in easy steps, how to set up the Arduino IDE on our computer and prepare the board to receive the program via USB cable.

Step 1: Selection of Arduino Board

First you must have your Arduino board (you can choose your favourite board) and a USB cable. In case you use Arduino UNO, Arduino Duemilanove, Nano, Arduino Mega2560, or Diecimila, you will need a standard USB cable (A plug to B plug), the kind you would connect to a USB printer as shown in the Figure 4.26

Figure 4.26: USB Cable

Step 2: Download Arduino IDE Software

You can get different versions of Arduino IDE from the Download page on the Arduino Official website. You must select your software, which is compatible with your operating system (Windows, IOS, or Linux). After your file download is complete, unzip the file.

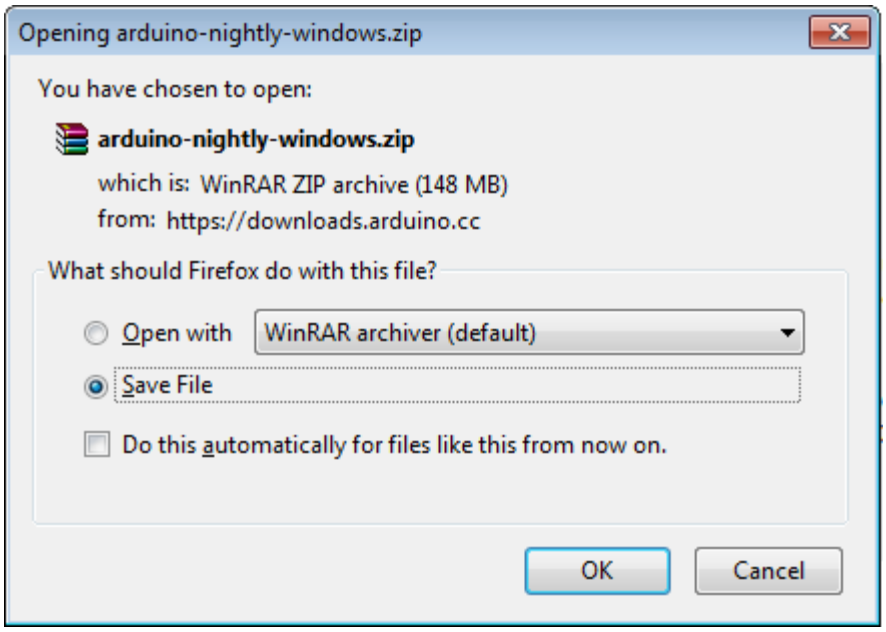

Figure 4.27: Arduino Installation File Download Confirmation

Step 3: Power up your board.

The Arduino Uno, Mega, Duemilanove and Arduino Nano automatically draw power from either, the USB connection to the computer or an external power supply. If you are using an Arduino Diecimila, you have to make sure that the board is configured to draw power from the USB connection. The power source is selected with a jumper, a small piece of plastic that fits onto two of the three pins between the USB and power jacks. Check that it is on the two pins closest to the USB port. Connect the Arduino board to your computer using the USB cable. The green power LED (labeled PWR) should glow.

Step 4: Launch Arduino IDE.

After your Arduino IDE software is downloaded, you need to unzip the folder. Inside the folder, you can find the application icon with an infinity label (application.exe). DoubleClick the icon to start the IDE.

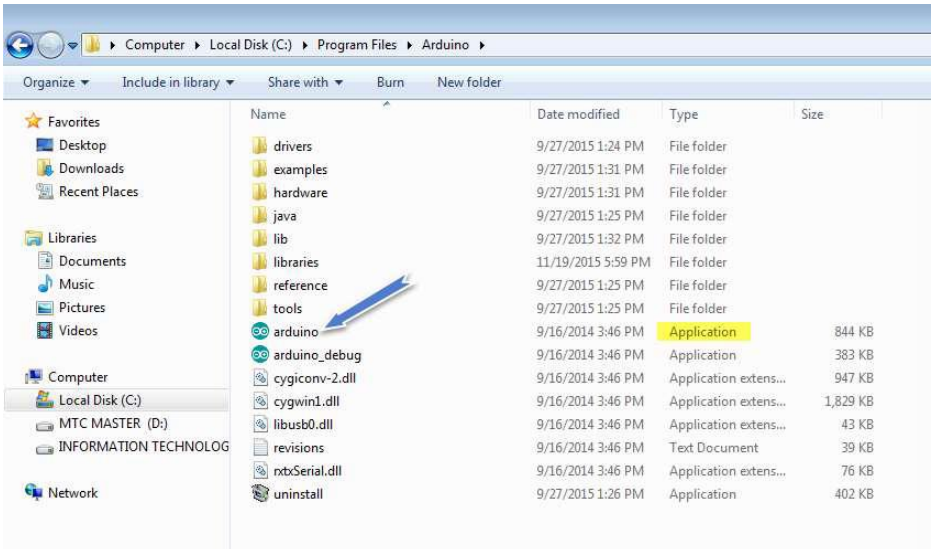

Figure 4.28: Arduino IDE Executable File Location

Step 5: Open your first project.

Once the software starts, you have two options:

- Create a new project.
- Open an existing project example.

To create a new project, select File --> New.To open

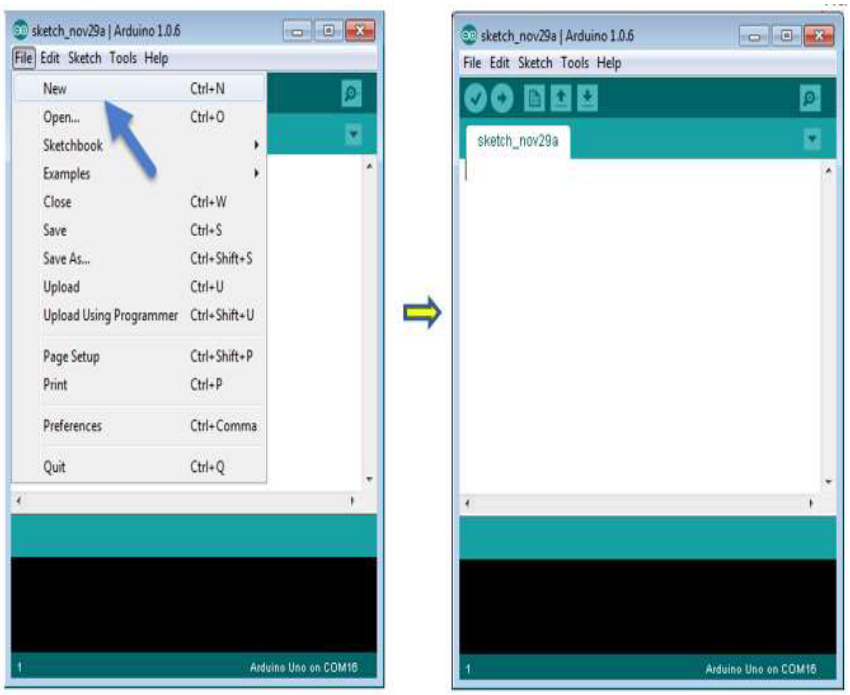

Figure 4.29: Creating a New Sketch in Arduino IDE

To open an existing project example, select File -> Example -> Basics -> Blink.

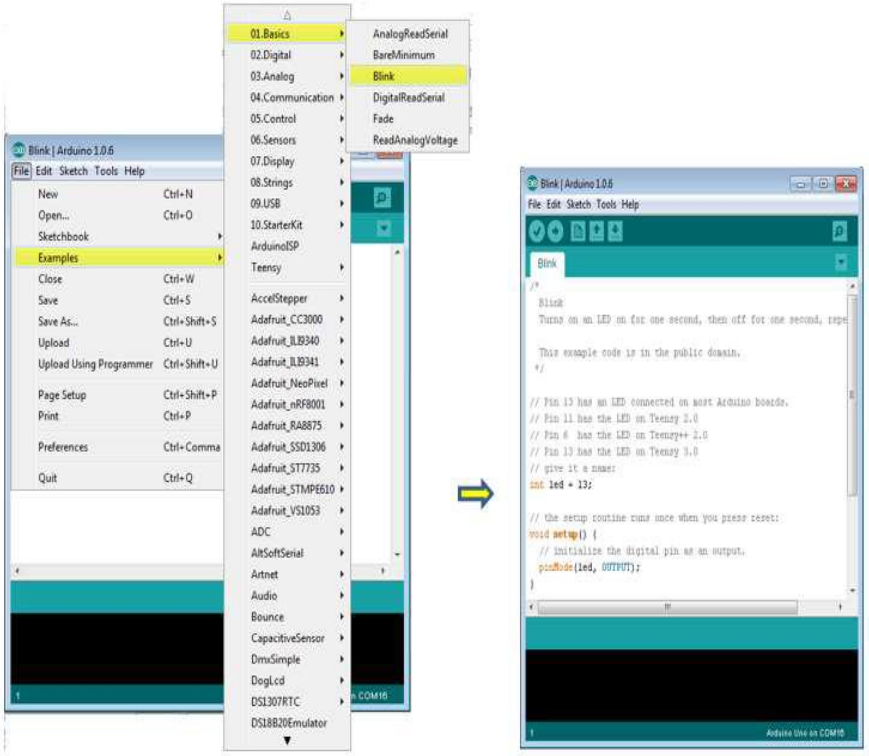

Figure 4.30: Opening the Blink Example Sketch in Arduino IDE

Here, we are selecting just one of the examples with the name **Blink**. It turns the LED on and off with some time delay. You can select any other example from the list.

Step 6: Select your Arduino board.

To avoid any error while uploading your program to the board, you must select the correct Arduino board name, which matches with the board connected to your computer.

Go to Tools -> Board and select your board

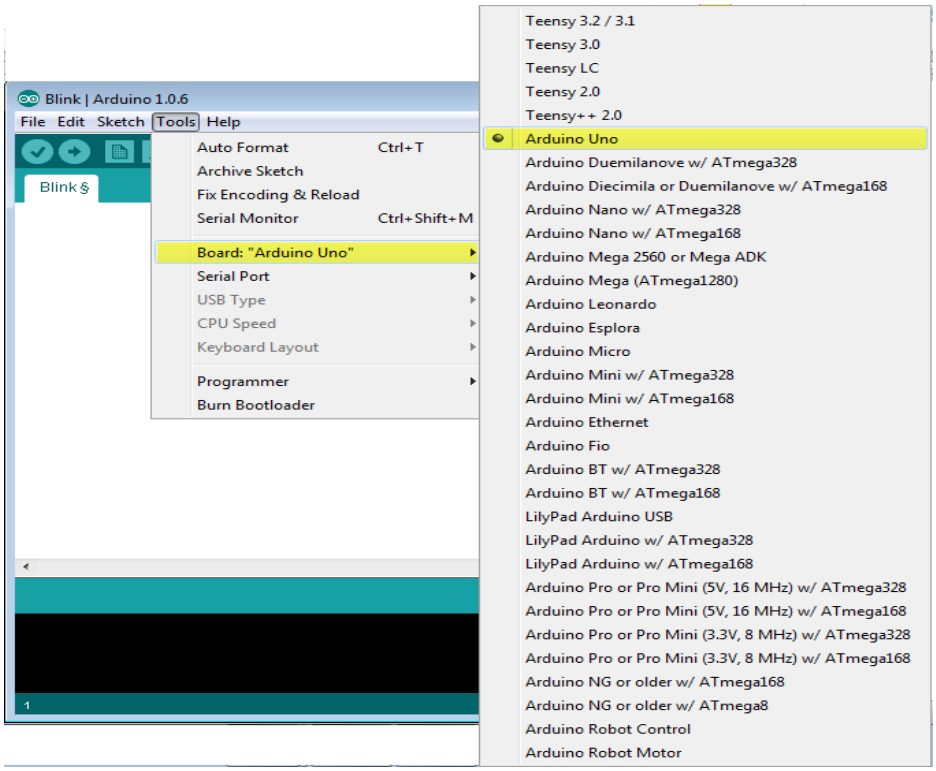

Figure 4.31: Selecting the Arduino Board Type in Arduino IDE

Here, we have selected Arduino Uno board according to our tutorial, but you must select the name matching the board that you are using

Step 7: Select your serial port.

Select the serial device of the Arduino board. Go to **Tools** ->**Serial Port** menu. This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu, the entry that disappears should be of the Arduino board. Reconnect the board and select that serial port.

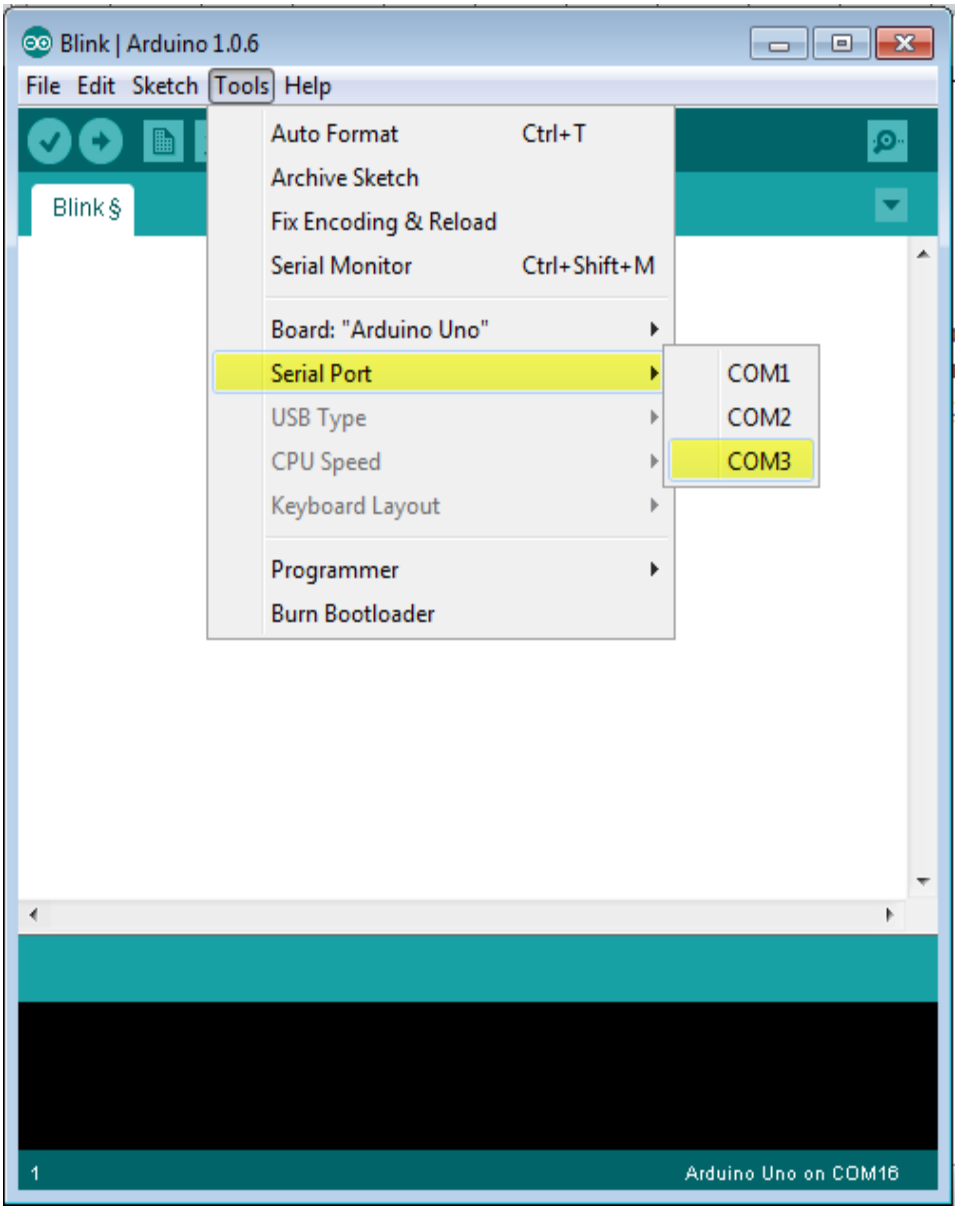

Figure 4.32: Selecting the Serial Communication Port in Arduino IDE

Step 8: Upload the program to your board.

Before explaining how we can upload our program to the board, we must demonstrate the function of each symbol appearing in the Arduino IDE toolbar.

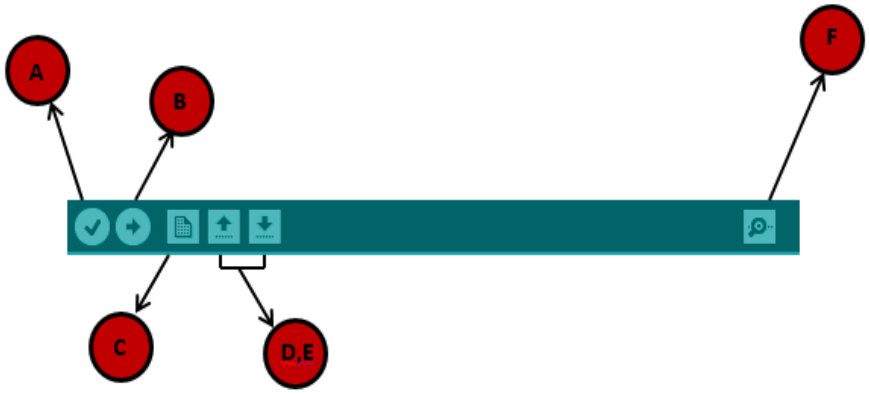

Figure 4.33: Arduino IDE Toolbar Icons and Their Functions

A- Used to check if there is any compilation error.

B- Used to upload a program to the Arduino board.

C- Shortcut used to create a new sketch.

D- Used to directly open one of the example sketch.

E- Used to save your sketch.

F- Serial monitor used to receive serial data from the board and send the serial data to the board.

Now, simply click the "Upload" button in the environment. Wait a few seconds; you will see the RX and TX LEDs on the board, flashing. If the upload is successful, the message "Done uploading" will appear in the status bar.

Note: If you have an Arduino Mini, NG, or other board, you need to press the reset button physically on the board, immediately before clicking the upload button on the Arduino Software.

Arduino programming structure

In this chapter, we will study in depth, the Arduino program structure and we will learn more new terminologies used in the Arduino world. The Arduino software is open-source. The source code for the Java environment is released under the GPL and the C/C++ microcontroller libraries are under the LGPL.

Sketch: The first new terminology is the Arduino program called "**sketch**".

Structure

Arduino programs can be divided in three main parts: **Structure**, **Values** (variables and constants), and **Functions**. In this tutorial, we will learn about the Arduino software program, step by step, and how we can write the program without any syntax or compilation error.

Let us start with the **Structure**. Software structure consist of two main functions:

- Setup( ) function
- Loop( ) function

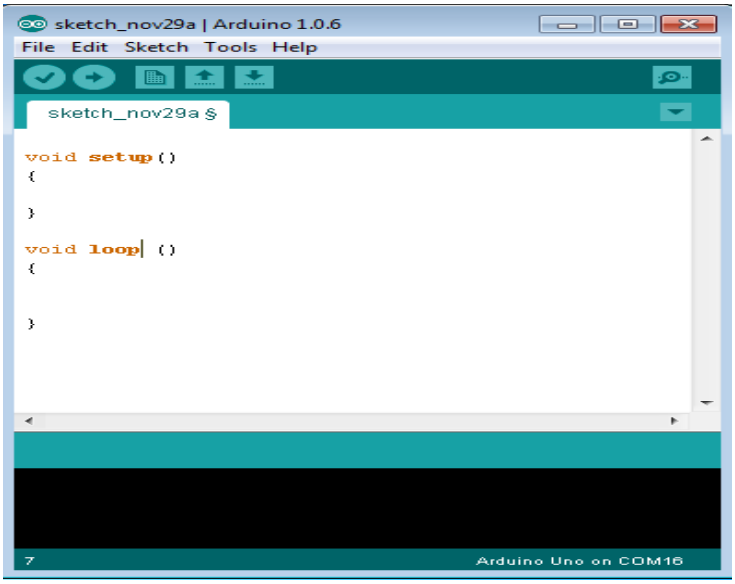

Figure 4.34: Basic Arduino Sketch Structure - setup() and loop() Functions

PURPOSE:

The **setup()** function is called when a sketch starts. Use it to initialize the variables, pin modes, start using libraries, etc. The setup function will only run once, after each power up or reset of the Arduino board.

INPUT

OUTPUT

RETURN

```
Void Loop ( )
{
}
```

PURPOSE:

After creating a **setup()** function, which initializes and sets the initial values, the **loop()** function does precisely what its name suggests, and loops secutively, allowing your program to change and respond. Use it to activelycontrol the Arduino board.

INPUT

OUTPUT

RETURN

4.5.2 Raspberry-Pi in IOT

Architecture of Raspberry Pi

Architecture of Raspberry Pi is shown in Figure 4.27

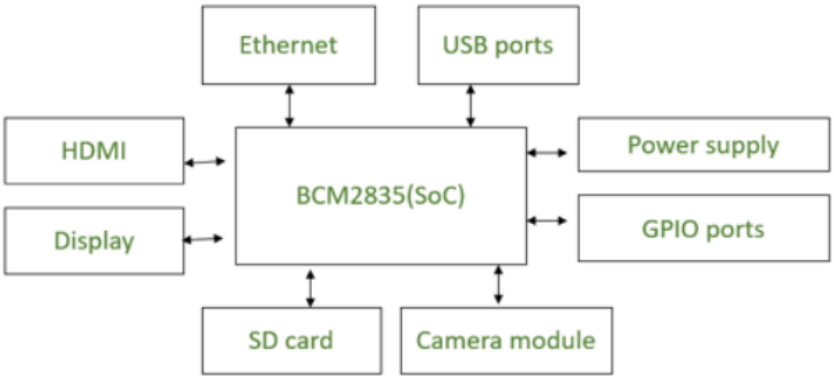

Figure 4.35: Architecture of Raspberry Pi

Raspberry Pi mainly consists of the following blocks:

• Processor: Raspberry Pi uses Broadcom BCM2835 system on chip which is an ARM processor and Video core Graphics Processing Unit (GPU). It is the heart of the Raspberry Pi which controls the operations of all the connected devices and handles all the required computations.

• HDMI: High Definition Multimedia Interface is used for transmitting video or digital audio data to a computer monitor or to digital TV. This HDMI port helps Raspberry Pi to connect its signals to any digital device such as a monitor digital TV or display through an HDMI cable.

• GPIO ports: General Purpose Input Output ports are available on Raspberry Pi which allows the user to interface various I/P devices.

- Audio output: An audio connector is available for connecting audio output devices such as headphones and speakers.
- USB ports: This is a common port available for various peripherals such as a mouse, keyboard, or any other I/P device. With the help of a USB port, the system can be expanded by connecting more peripherals.
- SD card: The SD card slot is available on Raspberry Pi. An SD card with an operating system installed is required for booting the device.
- Ethernet: The ethernet connector allows access to the wired network, it is available only on the model B of Raspberry Pi.
- Power supply: A micro USB power connector is available onto which a 5V power supply can be connected.

Camera module: Camera Serial Interface (CSI) connects the Broadcom processor to the Pi camera.

- Display: Display Serial Interface (DSI) is used for connecting LCD to Raspberry Pi using 15 15-pin ribbon cables. DSI provides a high-resolution display interface that is specifically used for sending video data. Raspberry pi is shown in Figure 4.28

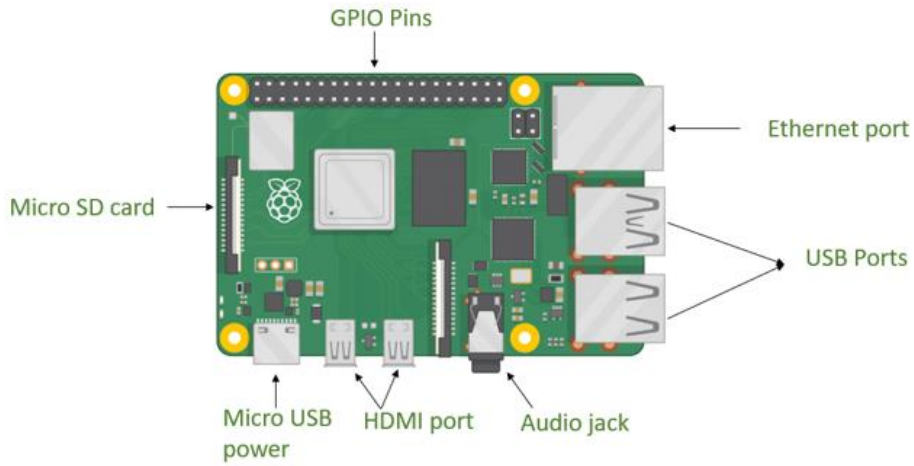

Figure 4.36: Raspberry Pi

Types of Raspberry Pi

Since its launch in 2012, the **Raspberry Pi Foundation** has released several models, each with improvements in processing power, memory, and

connectivity. These models cater to **IoT projects, robotics, education, multimedia, and industrial automation**.

List of Raspberry pi models and releases year is shown in Table 4.1
1. pi 1 model B - 2012
2. pi 1 model A - 2013
3. pi 1 model B+ - 2014
4. pi 1 model A+ - 2014
5. Pi 2 Model B - 2015
6. Pi 3 Model B - 2016
7. Pi 3 Model B+ -2018
8. Pi 3 Model A+ -2019
9. Pi 4 Model A – 2019

Table 4.1 Types of Raspberry Pi

Model	Processor	RAM	Special Features	Best Use Case
Raspberry Pi 1	ARM11 (700 MHz)	256–512MB	First-generation Pi	Basic IoT, learning
Raspberry Pi 2	Quad ARM Cortex-A7 (900MHz)	1GB	Faster than Pi 1	Robotics, IoT hub
Raspberry Pi 3	Quad ARM Cortex-A53	1GB	Wi-Fi & Bluetooth onboard	Smart homes, IoT
Raspberry Pi 4	Quad ARM Cortex-A72	2–8GB	Dual HDMI, USB 3.0, faster CPU	AI IoT, desktops
Raspberry Pi Zero/Zero W	ARM11 (1GHz)	512MB	Ultra-small, Zero W has Wi-Fi	Wearables, compact IoT
Raspberry Pi Pico	RP2040 (Dual Cortex-M0+)	264KB	Microcontroller, ultra-low power	Sensor nodes, robotics
Raspberry Pi 400	Quad Cortex-A72 (1.8GHz)	4GB	Built into a keyboard	Education, IoT gateway

Raspberry Pi Pinout:

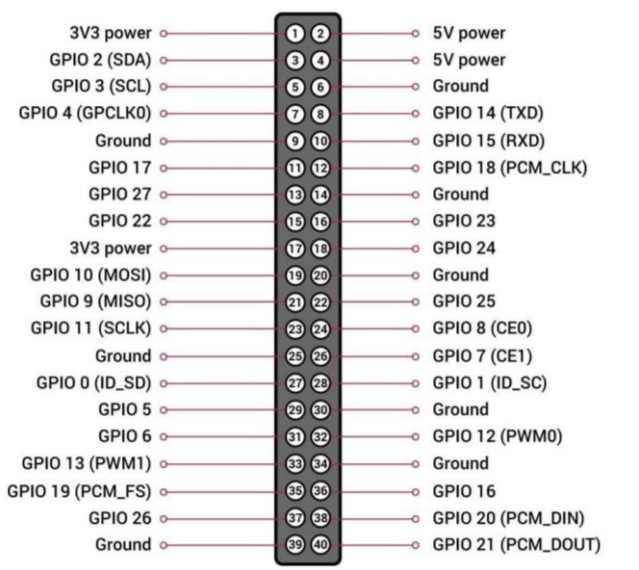

Figure 4.37: Raspberry Pi Pin Diagram

Pin 1 (3.3V Power): Pin 1 supplies a constant 3.3 volts directly from the Raspberry Pi. It is typically used to power small sensors, modules, or circuits that operate at 3.3V. It is important to monitor the current draw from this pin, as exceeding its limit can damage the board.

Pin 2 (5V Power): This pin delivers a 5-volt supply from the Pi's onboard regulator or USB power. It is used to power devices or components that require 5 volts, such as certain displays or small motors. It provides a higher voltage than Pin 1, suitable for slightly more demanding components.

Pin 3 (GPIO2 / SDA1): Pin 3 functions as a general-purpose input/output (GPIO) pin and as the I2C data line (SDA1). This pin enables communication with I2C devices like temperature sensors, OLED displays, or EEPROMs. When used as a GPIO pin, it can be programmed as either input or output.

Pin 4 (5V Power): Pin 4 is another 5-volt power supply pin, identical in function to Pin 2. It is often used in parallel with Pin 2 to power multiple

components simultaneously or provide extra current for more demanding modules.

Pin 5 (GPIO3 / SCL1): Pin 5 serves as the I2C clock line (SCL1) and a general-purpose GPIO. It works together with Pin 3 to enable I2C communication. Additionally, it can act as a regular GPIO pin when I2C is not in use.

Pin 6 (Ground): Pin 6 is a ground (GND) pin. Ground pins provide the reference point for electrical circuits and are essential for completing any connection between the Pi and external components.

Pin 7 (GPIO4): Pin 7 is a versatile GPIO pin that can serve general input/output functions, act as a 1-Wire interface for certain sensors, or generate a clock signal (GPCLK0). It is commonly used in DIY electronics projects for both digital signaling and sensor communication.

Pin 8 (GPIO14 / TXD0): This pin functions as the UART transmit line (TXD0) for serial communication and can also act as a general GPIO. It is used to send data from the Raspberry Pi to another microcontroller or serial device, such as GPS modules or serial displays.

Pin 9 (Ground): Another ground pin, Pin 9 serves as the return path for current in electrical circuits connected to the Raspberry Pi. It ensures proper functioning and stability of all connected components.

Pin 10 (GPIO15 / RXD0): Pin 10 is the UART receive line (RXD0) and a general GPIO pin. It receives serial data from external devices, such as microcontrollers or sensors, enabling two-way communication when paired with Pin 8.

Pin 11 (GPIO17): Pin 11 is a general-purpose input/output pin commonly used to control LEDs, buttons, or relays. It can be configured as an input to read signals or as an output to control other devices.

Pin 12 (GPIO18 / PWM0): Pin 12 supports hardware Pulse Width Modulation (PWM0) and general GPIO functions. It is frequently used to control the brightness of LEDs, the speed of motors, or generate audio signals.

Pin 13 (GPIO27): Pin 13 is a flexible GPIO pin used for general digital input or output tasks. It is suitable for connecting switches, sensors, or other low-power devices in IoT projects.

Pin 14 (Ground): Pin 14 is a ground pin that provides a stable reference for connected devices. Multiple ground pins are included in the Pi to ensure easy connectivity on the 40-pin header.

Pin 15 (GPIO22): Pin 15 is a general-purpose GPIO pin. It is often used to interface with buttons, LEDs, or relays, providing digital control in IoT and embedded systems projects.

Pin 16 (GPIO23): Pin 16 functions as a general-purpose digital input/output pin. It can read signals from sensors or send output signals to actuators and indicators.

Pin 17 (3.3V Power): This pin supplies an additional 3.3 volts for powering low-voltage sensors or modules. It is identical in function to Pin 1 and provides flexibility when multiple devices require 3.3V power.

Pin 18 (GPIO24): Pin 18 is a general-purpose GPIO pin, often used to read sensor data or control output devices. It is compatible with a wide range of sensors and modules.

Pin 19 (GPIO10 / MOSI): Pin 19 serves as the Master Out Slave In (MOSI) line for SPI communication and also functions as a general GPIO pin. It is used to send data from the Raspberry Pi to SPI-compatible devices like displays and ADCs.

Pin 20 (Ground): Pin 20 is a ground pin providing electrical stability for SPI devices or other circuits connected to the Pi.

Pin 21 (GPIO9 / MISO): Pin 21 acts as the Master In Slave Out (MISO) line for SPI communication and can serve as a general-purpose GPIO pin. It receives data from SPI devices back to the Pi.

Pin 22 (GPIO25): Pin 22 is a general-purpose GPIO pin suitable for digital input or output tasks, such as reading sensors or controlling actuators.

Pin 23 (GPIO11 / SCLK): Pin 23 functions as the SPI clock line (SCLK) and a general GPIO pin. It synchronizes data transfer between the Raspberry Pi and SPI devices.

Pin 24 (GPIO8 / CE0): Pin 24 is the SPI chip enable line (CE0) and also a GPIO pin. It selects the first device on the SPI bus, enabling communication with multiple SPI peripherals.

Pin 25 (Ground): Another ground pin that ensures stable electrical operation for SPI devices or any other connected components.

Pin 26 (GPIO7 / CE1): Pin 26 acts as the second SPI chip enable line (CE1) and a general-purpose GPIO. It allows the Pi to communicate with a second SPI device on the same bus.

Pin 27 (GPIO0 / ID_SD): Pin 27 is used as the I2C ID EEPROM data line (ID_SD) but can also function as GPIO0 in custom projects. It helps the Pi detect connected HAT boards.

Pin 28 (GPIO1 / ID_SC): Pin 28 is the I2C ID EEPROM clock line (ID_SC) and can act as GPIO1. It works alongside Pin 27 for HAT identification or custom use.

Pin 29 (GPIO5): Pin 29 is a general-purpose GPIO pin suitable for input or output tasks in embedded systems, such as triggering LEDs, reading switches, or interfacing with sensors.

Pin 30 (Ground): Pin 30 is a ground pin used to complete circuits and maintain electrical stability for connected devices.

Pin 31 (GPIO6): Pin 31 is a general-purpose GPIO pin for digital input or output, allowing connection with a variety of sensors or small devices.

Pin 32 (GPIO12 / PWM0): Pin 32 supports hardware PWM (Pulse Width Modulation) and general GPIO functions. It is commonly used to control motor speed, LED brightness, or audio tones.

Pin 33 (GPIO13 / PWM1): Pin 33 provides hardware PWM support (PWM1) and general GPIO capabilities, useful for smooth control of actuators or generating variable signals.

Pin 34 (Ground): Pin 34 is a ground pin providing a stable reference point for electrical circuits.

Pin 35 (GPIO19 / PCM_FS / PWM1): Pin 35 can be used for hardware PWM (PWM1), as a digital audio interface (PCM_FS), or as a general-purpose GPIO pin.

Pin 36 (GPIO16): Pin 36 is a general-purpose GPIO pin for digital input/output. It is commonly used for connecting switches, LEDs, or other devices.

Pin 37 (GPIO26): Pin 37 is a flexible GPIO pin suitable for input or output tasks, making it ideal for a variety of IoT sensors and actuators.

Pin 38 (GPIO20 / PCM_DIN): Pin 38 can serve as a general-purpose GPIO or as a digital audio input (PCM_DIN). It is often used for audio projects or general signaling.

Pin 39 (Ground): Pin 39 is a ground pin, providing the return path for electrical circuits and ensuring safe operation.

Pin 40 (GPIO21 / PCM_DOUT): Pin 40 can be used as a general GPIO or digital audio output (PCM_DOUT). It enables communication with audio devices or can function as a regular GPIO pin.

Linux on Raspberry Pi

1. Raspbian: Raspbian Linux is a RaspberryPi-optimized Debian Wheezy port.
2. Arch: Arch is an AMD-optimized version of Arch Linux.
3. Pidora: Pidora Linux is a RaspberryPi-optimized version of Fedora Linux.
4. RaspBMC: RaspBMC is a RaspberryPi-based XBMC media centre distribution.
5. OpenELEC: OpenELEC is an XBMC media centre distribution that is fast and easy to use.
6. RISC OS: RISC OS is a small, fast operating system.

Once you've decided on a Raspberry Pi, you'll need to learn a few things about it, beginning with an understanding of the various parts on the PCB and what - one does.

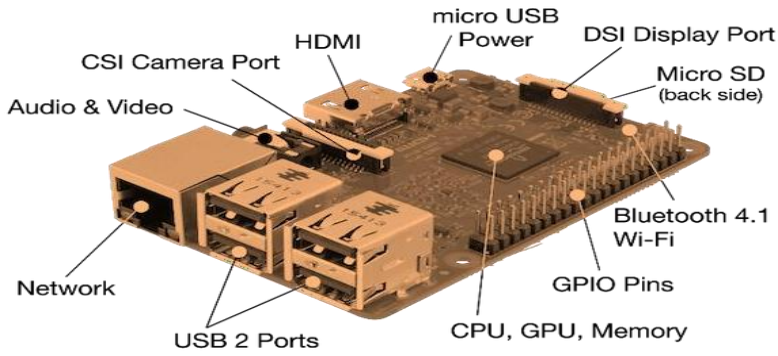

Figure 4.38: Parts of The Raspberry Pi

GPIO

GPIO is the Raspberry Pi's most critical function, and it's the Arduino's equivalent of GPIO pins. These pins can be used to read electrical signals from circuits and provide electrical signals for controlling circuits in

programmes. When using GPIO, be cautious since they are easily damaged and use 3.3V logic. A driver circuit (see 3.3 connecting I/O) should be used to operate external devices that draw more than 20mA current.Relays, inductors, and high-brightness LEDs are examples of such instruments.

DSI Display Port

The Raspberry Pi will connect to a serial display similar to those found in tablets using the DSI display port. These touch- screen monitor modules come in a variety of sizes, including 7inches.

CSI Camera Port

The CSI camera port is a connector that connects a Raspberry Pi camera module to the Raspberry Pi. Generic web cameras may not function since they usually only have a USB port.

MicroSD Slot

This slot is used to store the Raspberry Pi operating systemon a microSD card. The Pi does not come with a microSD card. This SD card also stores all of the user's files, directories, notes, and photos. It is simply the computer's hard drive.

HDMI / USB / Network

These ports link the Pi to an HDMI display, USB devices including mice and keyboards, and an ethernet connection for internet access. However, since the Raspberry Pi 3 has built-inWi-Fi, the ethernet port is seldom used.

Micro USB Power

Power can be supplied to the Raspberry Pi via a micro USB cable connected to the micro USB connector (recommended) or by directly feeding 5V into the 5V GPIO pin. Now that you're familiar with the components that make up the Raspberry Pi, you might discover that when it's turned on, it does nothing. This is due to the fact that we must first instal an operating system! In the following post, I'll show you how to put Raspbian, the

Raspberry Pi's most common operating system, on a microSD card.

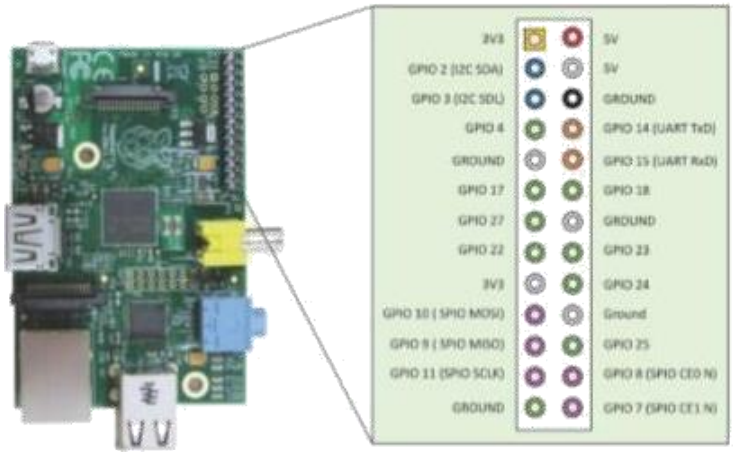

Figure 4.39: Raspberry Pi Gpio

4.5.2 Raspberry Pi Interfaces

For data transfer, the Raspberry Pi has Serial, SPI, and I2C interfaces. Receive (Rx) and transmit (Tx) pins on the Serial interfaceon the Raspberry Pi are used to communicate with serial peripherals.

Serial Peripheral Interface (SPI) is a synchronous serial data protocol that allows one or more peripheral devices to communicate with one another. In an SPI link, the Raspberry Pi has five pins for the SPI interface:

MISO (Master in slave out) – Master line for data transmission to peripherals.

Slave line for sending data to the owner, MOSI (Master out slave in).

SCK (Serial Clock) – Clock generated by the master fordata transmission synchronization.

To allow or disable devices, use CE0 (Chip Enable 0).

CE0 (Chip Enable 1) – This command is used to enable or disable devices.

I2C (Inter-Industry Communication):

The Raspberry Pi's I2C interface pins allow you to attach hardware

modules. The I2C interface allows for synchronous data transfer using only two pins: SDA (data line) and SCL (control line) (Clock Line).

Python is an excellent language for programming the Raspberry Pi because of its ease of use and accessibility to hardware, such as GPIO. We'll look at how to run Python programmes on the Raspberry Pi in this tutorial.

Hardware needed:

- Pi (Raspberry Pi)
- SD card with Raspbian
- OSDisplay with HDMI input
- Mouse and keyboard

A Python programme that requires external libraries couldnot operate on Raspbian using any of the methods mentioned below. PIP can be used to instal additional libraries, but it appears that it only does so for programmes that run in the terminal window. This could be because the Python IDE programmes have their own copy of Python installed locally. As a result, it is recommended that you run Python programmes that require external libraries installed using PIPfrom the terminal window.

```
rom time import sleeP

 import RPi.GPIO asGPIO

GPIO.setmode(GPIO.BCM)
#Switch Pin GPIO.setup(25,GPIO.IN)
#LEDPin
GPIO.setup(18,GPIO.OUT)
state=false
deftoggleLED(pin):
state = not state GPIO.output(pin,state)
whileTrue:
try:
if (GPIO.input(25) ==True):
toggleLED(pin)
```

sleep(.01) exceptKeyboardInterrupt:
exit()

Other Devices

1. pcDuino
2. BeagleBone Black
3. Cubie board pcDuino

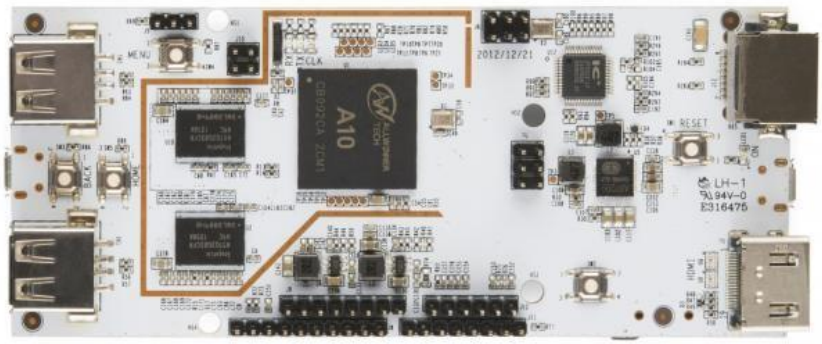

Figure 4.40: PCDUINO

The pcDuino's elegance lies in its exceptionally well- exposed hardware peripherals. Using these peripherals, on the other hand, is more difficult than using them on an Arduino- compatible circuit.

- **pcDuino** - some familiarity with the pcDuino basics is needed before proceeding. Before continuing, please check our Getting Started with pcDuino tutorial.
- Linux - The Linux operating system is the most important thing you should be familiar with. Keep in mind that pcDuino isn't an Arduino; it's a modern microcomputer with a completely functional, albeit compact, operating system.
- SPI is a synchronous (clocked) serial peripheral interface that allows chips on a board to communicate with one another. A minimum of four wires are required (clock, master-out-slave-in data, master-in-slave-out data, and slave chip select), with one additional chip select line required for each additional chip added to the bus.
- I2C - also known as IIC (inter-integrated circuit), SMBus, or TWI (two-wire interface), I2C communicates with different devices using just two wires

(bidirectional data and clock lines).

- Serial Communication - a data interface of at least two wires that is asynchronous (no transmitted clock) (data transmit and data receive; sometimes, additional signals are added to indicate when a device is ready to send or receive).
- Pulse Width Modulation (PWM) is a digital-to-analog conversion technique that employs a fixed frequency square wave with varying duty cycle that can be easily converted to an analogue signal ranging from 0V to the maximum amplitude of the digital IC driving the signal.
- Analog-to-Digital Conversion is the process of measuring an analogue voltage and converting it to a digital value.
- BeagleBoneBlack is a character in the game BeagleBoneBlack

The BeagleBone Black is the BeagleBoard family's newest member. It's a low-cost, high-expansion BeagleBoard powered by a Texas Instruments Sitara XAM3359AZCZ100 Cortex A8 ARM processor. It's similar to the Beaglebone, but it has certain features that the Beaglebone doesn't. The distinctions between the BeagleBone and the BeagleBone Black are summarised in the table 4.2

Table 4.2: Comparison of BeagleBone Black and BeagleBone Boards

	BeagleBone Black $55	BeagleBone $89
Processor	AM3358BZCZ100, 1GHZ	AM3359ZCZ72, 720MHz
Video Out	HDMI	None
DRAM	512MB DDR3L 800MHZ	256MB DDR2 400MHz
Flash	4GB eMMC, uSD	uSD
Onboard JTAG	Optional	Yes, over USB
Serial	Header	Via USB
PWR Exp Header	**No**	**Yes**
Power	210-460 mA@5V	300-500 mA@5V

BeagleBone Black Features

The following table lists the key features of the BeagleBoneBlack

Table 4.3: Detailed Hardware Specifications of the BeagleBone Black

	Feature	
Processor	Sitara AM3358BZCZ100 1GHz, 2000 MIPS	
Graphics Engine	SGX530 3D, 20M Polygons/S	
SDRAM Memory	512MB DDR3L 800MHZ	
Onboard Flash	4GB, 8bit Embedded MMC	
PMIC	TPS65217C PMIC regulator and one additional LDO.	
Debug Support	Optional Onboard 20-pin CTI JTAG, Serial Header	
Power Source	miniUSB USB or DC Jack	5VDC External Via Expansion Header
PCB	3.4" x 2.1"	6 layers
Indicators	1-Power, 2-Ethernet, 4-User Controllable LEDs	
HS USB 2.0 Client Port	Access to USB0, Client mode via miniUSB	
HS USB 2.0 Host Port	Access to USB1, Type A Socket, 500mA LS/FS/HS	
Serial Port	UART0 access via 6 pin 3.3V TTL Header. Header is populated	
Ethernet	10/100, RJ45	
SD/MMC Connector	microSD , 3.3V	
User Input	Reset Button Boot Button Power Button	
Video Out	16b HDMI, 1280x1024 (MAX) 1024x768,1280x720,1440x900 ,1920x1080@24Hz w/EDID Support	
Audio	Via HDMI Interface, Stereo	
Expansion Connectors	Power 5V, 3.3V , VDD_ADC(1.8V) 3.3V I/O on all signals McASP0, SPI1, I2C, GPIO(69 max), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 4 Serial Ports, CAN0, EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)	
Weight	1.4 oz (39.68 grams)	
Power	Refer to Section 6.1.7	

In the box is (1)BeagleBone Black board, (1)USB cable, and(1)card that should be read.

BeagleBone Black Picture

Here is a picture of the Rev A5A board.

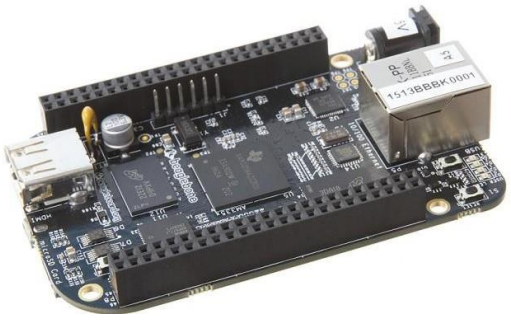

Figure 4.42: Beaglebone Black Picture

BeagleBone Black Key Component Locations

Here are the locations of the key components on the Rev A5A.

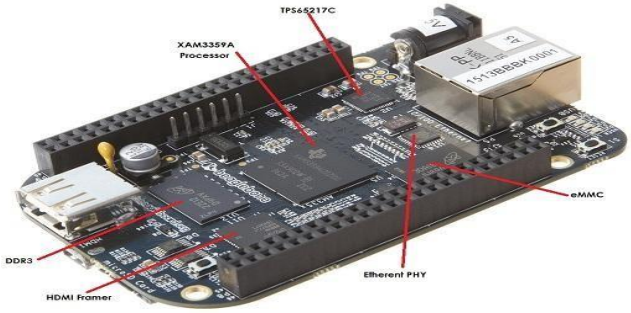

Figure 4.43: Beaglebone Black Key ComponentLocations

BeagleBone Black Connector and Switch Locations

Below is the location of the connectors and switches on theRev A5A board. The Power Button and Battery Connections are new additions to the Rev A5A.

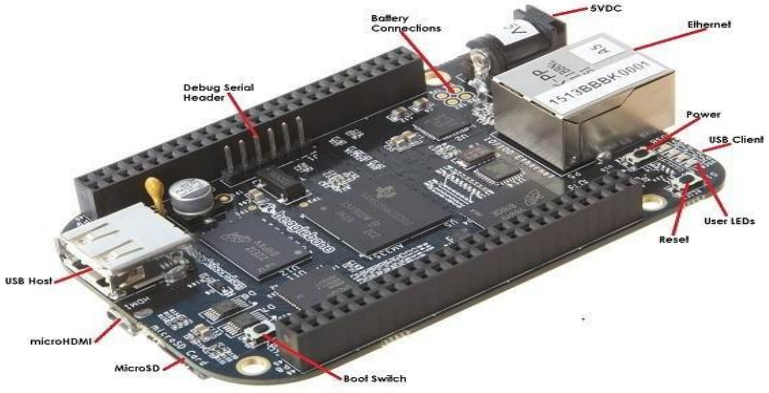

Figure 4.44: Beaglebone Black Connector AndSwitch Locations

Cubieboard

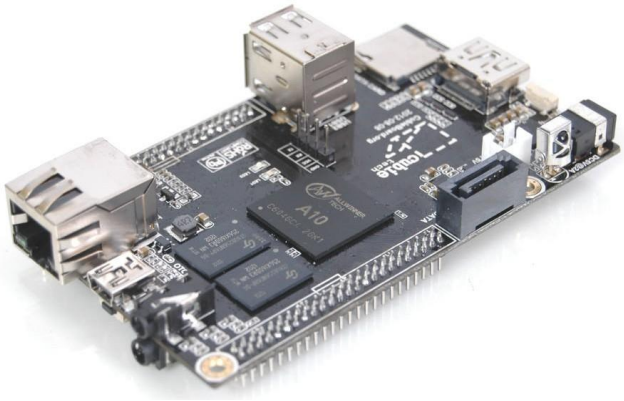

Figure 4.45: Cubieboard

Cubieboard is a Chinese single-board machine manufactured in Zhuhai, Guangdong. In September 2012, the first short run of prototype boards was sold internationally, and in October 2012, the production version was released. [1] It can run Android 4 ICS, Ubuntu 12.04 desktop, and other operating systems. [2] Remixed Fedora 19 for ARM [3] Armbian, Arch Linux ARM, desktop [4] a Cubian distribution based on Debian, [5] FreeBSD,[6] or OpenBSD are two options.[nine]

It is powered by the AllWinner A10 SoC, which is commonly used in low-cost tablets, phones, and media PCs. Developers of the lima driver, an open-source driver for the ARM Mali GPU, use this SoC. [eight] It ran ioquake 3 at 47 frames per second in 1024x600 at the 2013 FOSDEM demo. [nine]

Using the Lubuntu Linux distribution, the Cubieboard team was able to run an Apache Hadoop computer cluster.

CHAPTER 5

DATA ANALYTICS AND SUPPORTING SERVICES

5.1 INTRODUCTION

Traditional data management systems are clearly unprepared for the demands of what has come to be known as "big data." As explored in this book, the true importance of IoT lies not in connecting things but in the data generated by those things, the new technologies that can be enabled by those linked things, and the business insights that the data can show. However, in order for the data to be usable, it must be treated in an ordered and managed manner. As a result, a new approach to data analytics is needed for the Internet of Things. The generation of massive amounts of data from sensors is common in the world of IoT and one of the most difficult challenges not only from a transportation standpoint, but also from a data management standpoint. The commercial aviation industry and the sensors installed in an aircraft are a perfect example of the deluge of data that can be provided by IoT. Modern jet engines are outfitted with thousands of sensors that produce a whopping 10GB of data per second.

For eg, modern jet engines, such as the one depicted in Figure 1, may be outfitted with about 5000 sensors. As a result, a twin-engine commercial aircraft with these engines running on average 8 hours per day can produce over 500 TB of data per day, and this is just the data from the engines! Thousands of other sensors are now connected to the airframe and other equipment on modern aircraft. In fact, a single wing of a modern jumbo jet is outfitted with 10,000 sensors.

A petabyte (PB) of data a day per commercial Aeroplan is not out of the question and this is only for one Aeroplan. Every day, approximately 100,000 commercial flights take place all over the world. The amount of IoT data generated by the commercial airline industry alone is staggering. This is only one of the examples that illustrate the Internet of Things dilemma that is being compounded by IoT. Analyzing this volume of data

in the most reliable way possible comes under the purview of data analytics. Data analytics must be able to provide actionable observations and information from data, regardless of volume or format, in a timely manner, or the full benefits of IoT will not be realised.

Until delving further into data analytics, it is critical toidentify a few core data principles. For starters, not all data is the same; it can be classified and thus analysed in a thing of ways. Various data analytics software and analysis techniquescan be used depending on how the data is classified. From the standpoint of IoT, two essential distinctions are whether the data is organised or unstructured, and whether it is in motion or at rest.

5.1.1 STRUCTURED VS UNSTRUCTURED DATA

Structured data and unstructured data are important classifications because they usually necessitate separate tool sets in terms of data analytics. Figure 2 compares organised and unstructured data at a high level. Structured vs unstructured is shown in Figure 5.1

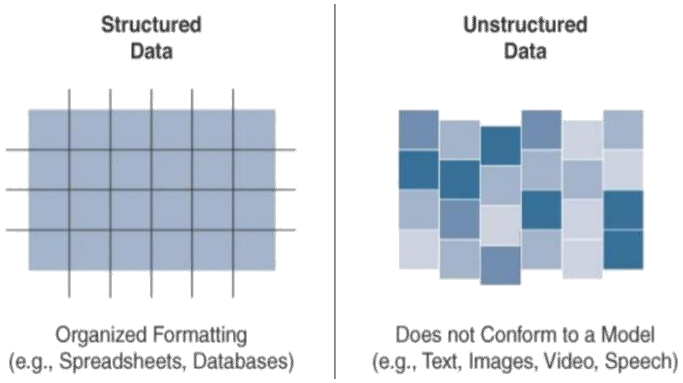

Figure 5.1: Structured Vs Unstructured Data

Structured data is described as data that adheres to a model or schema that specifies how the data is interpreted or structured, implying that it is compatible with a standard relational database management system (RDBMS). In certain instances, organised data can be found in a simple tabular format, such as a spreadsheet where data occupies a certain cell

and can be clearly specified and referenced. Structured data can be used in almost all computing environments and contains everything from financial transactions and invoices toserver log files and router settings. Structured values, such as temperature, pressure, humidity, and so on, are often used in IoT sensor data and are all sent in a known format. Structured data is efficiently formatted, saved, queried, and processed; as a result, it has long been the primary category of data used to make business decisions. Since structured data is highly organized, a broad range of data analytics resources are readily available for analyzing this sort of data. Most people are experienced and at ease dealing with organized data, whether by custom scripts or commercial applications such as Microsoft Excel and Tableau.

Unstructured data lacks a logical format that can be used to interpret and decode the data using standard programming methods. Text, voice, photographs, and video are examples of this data form. Unstructured data is described as any data that does not neatly fit into a predefined data format.

According to some figures, about 80% of a company's data is unstructured. As a result, data analytics approaches that can be applied to unstructured data, such as semantic computing and deep learning, are receiving a lot of interest. You can decipher speech using deep learning applications such as natural language processing (NLP). You can retrieve vital knowledge from still photographs and video using image/facial recognition software.

Later, we'll go over how to handle unstructured IoT data with machine learning techniques.
In IoT networks, smart objects generate both organized and unstructured data. Because of its well-defined organization, structured data is easier to handle and process. Unstructured data, on the other hand, can be more difficult to manage and usually necessitates the use of very different analytics softwareto analyze the data. It is important to be familiar with all of these data classifications because understanding which data classification you are dealing with makes interacting with the right data analytics solution much simpler.

Data in motion versus data at rest

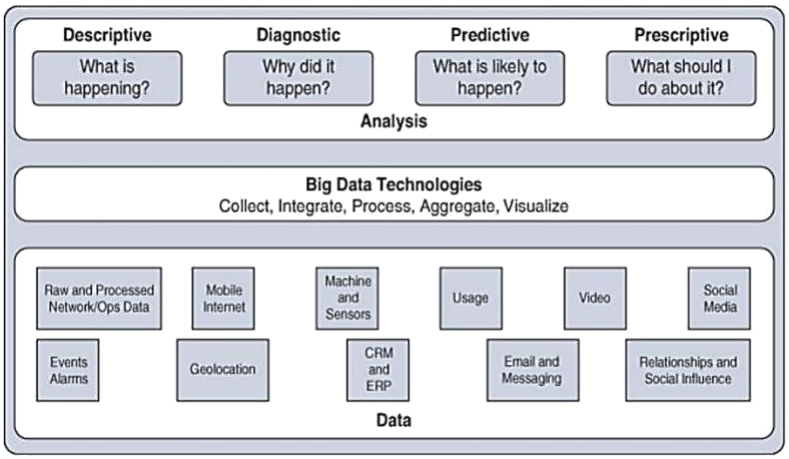

Figure 5.2: Data in Motion Versus Data at Rest

Data in IoT networks is either in transit (data in motion‖) or kept or preserved (data at rest‖), as it is in most networks. Standard client/server communications, such as online access and file transfers, as well as email, are examples of data in motion. Data at rest is data saved on a hard disc, storage array, or USB drive. From the standpoint of the Internet of Things, data from smart objects is called data in motion as it travels across the network on the way to its final destination. This is frequently handled at the edge through fog computing. As data is stored at the edge, it may be filtered and discarded, or it may be transferred to a fog node or a data Centre for further processing and future storage. At the edge, Data does not come to rest.

When data arrives at the data Centre, it can be processed in real-time, much as it is at the ground, while it is still in motion. Tools with this feature, such as Spark, Storm, and Flink, are only in their infancy when opposed to tools for processing stored data. Later parts of this chapter offer further detail on the Hadoop ecosystem's real-time streaming research tools. Data at rest in IoT networks is usually found in IoT brokers or in a storage array at the data Centre. From the standpoint of data analytics, there are numerous

tools available, especially tools for organized data in relational databases. Hadoop is the most well-known of these tools. Hadoop aids not only in data collection but also in data storage. It is explored in more depth later.

Descriptive data processing teaches you what is going on either now or in the past. A thermometer in a truck engine, for example, records temperature values per second. From a descriptive research standpoint, you can use this data at any time to obtain insight into the truck engine's actual working state. If the temperature is too high, there could be a cooling issue or the engine could be under too much pressure.

Diagnostic: If you want to know why anything is the way it is, diagnostic data analysis will help. Continuing with the temperature sensor in the truck engine case, you might be wondering why the truck engine failed. Diagnostic analysis can reveal that the engine's temperature was too high, causing it to overheat. Diagnostic research applied to data provided by a diverse set of smart objects may give a good image of whether a condition or incident happened.

Predictive forecasting seeks to foresee challenges or conflicts before they arise. For example, using historical temperature values for the truck engine, predictive analysis may provide an estimation of the remaining life of such engine components. These modules may then be substituted proactively until they malfunction.
Alternatively, if the temperature levels of the truck engine begin to steadily increase over time, this could mean the need for an oil change or some kind of engine cooling maintenance.
Prescriptive research moves beyond predictive analysis when recommending alternatives to potential problems. A predictive analysis of temperature data from a truck engine may generate a number of cost-effective maintenance options for our truck. Both estimates could include the cost of more regular oil changes and cooling repairs, as well as the installation of new cooling equipment on the engine or the upgrade to a lease on a model with a more powerful engine. Prescriptive review considers a number of considerations before making the required recommendation.

As in most networks, data in IoT networks is either in transit (data in motion‖) or being held or stored (data at rest‖). Examples of data in motion include traditional client/server exchanges, such as web browsing and file transfers, and email. Data saved to a hard drive, storage array, or USB drive is data at rest. From an IoT perspective, the data from smart objects is considered data in motion as it passes through the network en route to its final destination. This is often processed at the edge, using fog computing.

When data is processed at the edge, it may be filtered and deleted or forwarded on for further processing and possible storage at a fog node or in the data center. Data does not come to rest at the edge.

When data arrives at the data center, it is possible to process it in real-time, just like at the edge, while it is still in motion. Tools with this sort of capability, such as Spark, Storm, and Flink, are relatively nascent compared to the tools for analyzing stored data. Later sections of this chapter provide more information on these real-time streaming analysis tools that are part of the Hadoop ecosystem.

Data at rest in IoT networks can be typically found in IoT brokers or in some sort of storage array at the data center. Myriad tools, especially tools for structured data in relational databases, are available from a data analytics perspective. The best known of these tools is Hadoop. Hadoop not only helps with data processing but also data storage. It is discussed in more detail later. Prescriptive is shown in Figure 5.3

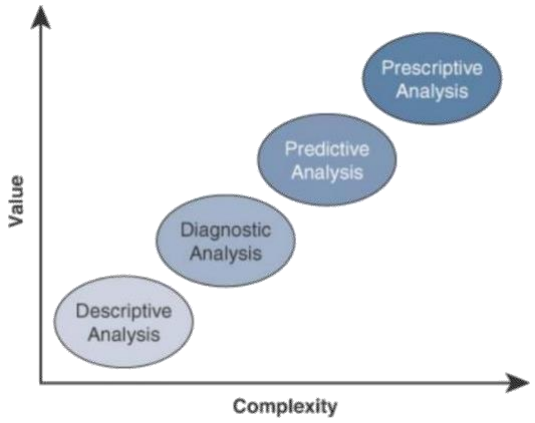

Figure 5.3: Prescriptive

5.2 IOT DATA ANALYTICS CHALLENGES

As the Internet of Things (IoT) grew and matured, it became apparent that conventional data analytics tools were not necessarily sufficient. Traditional data analytics, for example, usually hires a basic RDBMS and matching software, but the world of IoT is much more demanding. Although relational databases continue to be used for specific data types and implementations, they often struggle with the complexities of IoT data. IoT data presents two distinct threatsto a relational database:

Problems of scaling: Relational datasets can grow very large quite quickly due to the large number of smart objects in most IoT networks that constantly transmit data. This can leadto performance problems that are expensive to fix, frequently necessitating further hardware and architecture improvements.

Data Volatility: When it comes to relational databases, it is important that the schema is constructed correctly from the start. Changing it later will cause the database to run slowly or not at all. Because of the schema's lack of versatility, revisions must be kept to a minimum. IoT data, on the other hand, is volatile in that the data model is expected to shift and adapt over time. A dynamic schema is often needed in order for data model adjustments to be made on a regular or even hourly basis.

To address issues such as scaling and data volatility, a newkind of database known as NoSQL is being used. Structured Query Language (SQL) is the

programming language used to interface with an RDBMS. A NoSQL database, as the name suggests, is one that does not use SQL. It is not set up in the conventional tabular format of a relational database. NoSQL databases do not have rigid schema requirements and can accommodate a dynamic, changing data model. This database are also generally much more scalable.

In addition to the relational database issues that IoT imposes, with the high amount of smart object data that varies on a regular basis, IoT also introduces problems with the live streaming nature of its data and with data management at the network level. Streaming data, which is created when smart objects relay data, is difficult to manage since it is typically of very high volume, and it is only useful if it can be analyzed and responded to in real-time. Real-time data analysis helps you to spot patterns or deviations that may signify a problem or a condition that requires an urgent response. To have a chance of influencing the outcome of this dilemma, you must be able to filter and interpret the data as it happens, as close to the edge as possible. The market for real-time data analysis is rapidly expanding.

Streaming analytics services are available from major cloud analytics vendors such as Google, Microsoft, and IBM, and a variety of other solutions can be used in-house. (Edge streaming analytics is covered in greater detail later in this chapter.) Another problem that IoT introduces to analytics is network data, which is referred to as network analytics. With too many smart objects interacting and streaming data in IoT networks, it can be difficult to ensure that these data flows are efficiently handled, tracked, and protected.

5.3 IOT NETWORK ANALYTICS APPLICATIONS

Flexible NetFlow and IPFIX are network analytics applications that can spot unusual trends or other issues in the flow of IoT data across a network. More information on network analytics, including Flexible NetFlow and IPFIX, is given.

Acquiring Data

We can now understand the functions needed for software, utilities, and business processes at the server-support and application levels after learning about computers, device- network data, communications, and packet connectivity to the Internet. These functions include data acquisition, data collection, data transactions, analytics, performance visualization, IoT technology integration, services, procedures, intelligence, information creation, and knowledgemanagement.

Let us first go through the following terminology and their definitions as they are found in IoT implementation layers.

Computer software or a collection of software modules is referred to as an application. An programme allows a person to carry out a series of organised operations, roles, and duties. Control and supervision of streetlights is one example of an application. Tracking and inventory management software aretwo further examples of applications. RFID tags and positioning data are used in tracking applications.

An programme allows a customer to borrow cash from an ATM (ATM). Other examples of IoT implementations include an umbrella transmitting weather alert notices (Example 1.1), waste container management, health tracking, traffic light control, synchronisation, and monitoring.

The term "service" refers to a mechanism that allows accessto one or more capabilities to be provided. The gateway to capabilities is provided by a service gui. Access to each resource is compliant with the limitations and policies specified in the service definition. Examples of service technologies include vehicle repair service capabilities and service capabilities for Automatic Chocolate Vending Machines (ACVMs) for timely filling of chocolates into the machines.

A service is made up of a group of similar programme modules and their functionalities. The package is reused for one or more uses. The set's use is compatible with the controls, restrictions, and policies defined in the

service description for each service. A service is often associated with a Service Level Agreement (SLA).

A service is composed of self-contained, discrete, and reusable elements. It offers logically clustered and encapsulated functionalities. Examples of IoT services include traffic light synchronisation, vehicle monitoring, system positioning, identification, and monitoring, home security- breach detection and management, waste containerreplacement, and health-alerts.

Service Oriented Architecture (SOA) is a software architecture paradigm that consists of utilities, messages, procedures, and processes. In a high-level business CHAPTER, SOA components are spread around a network or the Internet. An SOA can be used to create new software systems and technology integration architecture in an enterprise.

A message is an agent or object that communicates

The term "operation" refers to an action or series of actions. For eg, actions taken during a bank transaction.

The term transaction (trans + action) refers to two interconnected sets of operations, acts, or orders. For example, a contract may be gaining access to sales data in order to pick and receive the annual sales for a given year in exchange. The first operation is access to revenue data, and the second is annual revenues in exchange. A query transaction with a Database Management System is another example of a transaction (DBMS).

A query is a command that retrieves select values from a database and then returns the response to the query after it has been processed. A query example is a command to the ACVMs database that returns ACVM sales data on Sundays near city gardens during a particular festival time of the year. Another example is a question to a service centre database to include a list of car parts in need of repair that have reached the end of their planned service life in a particular vehicle. Query processing is a collection of organised tasks performed to obtain responses from a data store based on the query.

The term "Key Value Pair" (KVP) refers to a group of two related entities, one of which is the key, which is a specific identifier for a linked entity, and the other which is either the entity that is named or a reference to the position of that entity. A birthday-date pair is an example of a KVP. KVP's birthday is July 17, 2000. A table's key is the birthday, and the value is the date July 17, 2000. KVP programmes generate look-up tables, hash tables, and network or system configuration files. A hash table (also known as a hash map) is a data structure that maps KVPs and is used to implement an associative array (for example array of KVPs). A hash table can employ an index (key) that is computed using a hash function, and the key maps to the value. The index is used to obtain or point to the desired value.

Bigtable converts two arbitrary string values into an arbitrary byte array. One is used as a row key, and the other as a column key. Time stamps are used in three-dimensional imaging. Mapping, unlike a relational database, can be thought of as a fragmented, distributed multi-dimensional sorted map. The table will scale up or 100s to 1000s of distributed computing nodes, with the ability to easily add more nodes.

In database theory, a business transaction (BT) is a (business) operation that requests information from or modifies data in a database.

A BT process is the command _connect,' which binds a DBMS and client, which in turn connects with the DBMS. Similarly, BTs are processes that use the commands _insert,' _delete,' _append,' and _modify.'

A process is described as a collection of organised activities or tasks that lead to a specific purpose (or that interact to achieve a result). For example, streetlights may monitor the process of purchasing an airline ticket. A process defines operations with relevant rules based on the process's data.
A Process Matrix is a multi-element entity in which each element connects a series of data or inputs to an operation (or subset of activities).
A business process (BP) is an operation or sequence of operations, or a set of organised events, functions, or processes that are interconnected. A BP serves a clear goal, outcome, service, or commodity. The business process (BP) is a representation, process matrix, or flowchart of a series of operations with

interleaving decision points; interleaving means adding in between. A decision point is a point in a sequence of events where choices are made about subsequent activities.
According to the web description, a business process (BP) is "a particular occurrence in a sequence of organised businessprocesses that usually alter the state of data and/or a product and produce some kind of output." Seeking annual revenue increase and handling stocks are two examples of BPs. Another definition2 of BP is "a business operation is an action or group of activities that achieves a particular operational purpose." According to another definition3, BP ia collection of logically linked activities or tasks (such as preparation, manufacturing, or sales) conducted together to generate a given collection of results.

5.4 INTELLIGENCE IN BUSINESS

Business intelligence (BI) is a mechanism that allows a business service to derive new information and insights in order to make smarter decisions. This new information and insights are the product of previous data collection,aggregation, and interpretation.

Obtaining Data

Data is generated at devices and then sent to the Internetthrough a gateway.

Data is produced as follows:

1. **Data generated by passive devices**: Data generated at the interface or machine as a result of interactions. A passive system lacks its own power supply. An external source assists such a system in generating and transmitting data. RFID tags and ATM debit cards are two examples. The computer may or may not have a microcontroller, memory, or transceiver. A contactless card is one example of the former, and a sticker or barcode is another.

2. **Data generated by active devices**: Data generated at the computer or machine as a result of encounters. An operational computer is self-

contained in terms of fuel. Responsive RFID, streetlight sensors, and wireless sensor nodes are a few examples. An working computer is also equipped with a microcontroller, memory, and a transceiver.

3. **Event data:** A system can only produce event data once. For example, when traffic is detected or when the environment is dark, the incident is signalled. The event on darkness communicates the need for a common of streetlights to be illuminated (Example 1.2). A security camera device may produce data in the event of a security violation or the detection of an intruder. When a waste container with an associated circuit is filled to 90% or higher, it may produce data. The components and instruments in a car produce data about their performance and operation. For example, as a brake lining wears out, there is a play in the steering wheel and less air conditioning. The data were sent to the Internet. The contact occurs as the vehicle approaches a Wi-Fi connection point.

4. **Device real-time data:** An ATM produces data and transmits it to a server through the Internet in real time. This initiates and facilitates real-time Online Transaction Processing (OLTP).

5. **Event-driven computer data**: A device data can only be produced once in response to an event. Here are some examples: I a computer receives a command from a Controller or Monitor and then executes an action(s) with the aid of an actuator. When an operation is completed, the system sends an acknowledgement; (ii) when an application requests a device's status, the device expresses the status.

1. **Data collection**

Data acquisition refers to the collection of information from IoT or M2M computers. Following contacts with a data acquisition device, the data interacts (application). The programme interacts and connects with a variety of devices in order to obtain the required data. The machines transmit data on demand or at predetermined intervals. System Data communicates through the network, transport, and security layers.

When computers have the ability to configure themselves, an application may configure them for data. The framework, for example, will configure machines to transmit data at predefined intervals. The frequency of data generation is determined by the system configuration. For example, the machine may configure an umbrella package to obtain weather data from an Internet weather service once a week. Every hour, an ACVM can be programmed to communicate system sales data and other information. The ACVM framework can be programmed to interact instantly in the event of a malfunction or when a particular chocolate flavour requires the Fill service.

At the data-adaptation layer, the application may customize data sending after filtering or enrichment at the gateway. The portal that sits between the application and the hardware will perform one or more of the following functions: transcoding, data management, and system management. Data management can include the provision of privacy and protection, as well as data aggregation, compaction, and fusion.

Device-management programme includes features for determining the ID or address of a device, activating it, configuring it (managing device parameters and settings), registering, deregistering, attaching, and detaching it.

2. **Data verification**

Data obtained from devices does not imply that the data is accurate, relevant, or reliable. Data integrity refers to data thatis within the intended range or pattern, or data that was not manipulated during transmission. As a result, data must be validated. The validation checks on the acquired data are performed by data validation software. Logic, guidelines, and semantic annotations are used for validation tools. Valid data is needed for the applications or services. Only analytics, forecasts, prescriptions, diagnosis, and judgments should thenbe considered permissible.

A large amount of data is collected from a large number of devices, especially machines in industrial plants, embedded components data from a large number of vehicles, health devices in ICUs, wireless sensor networks, and so on. As a result, validation programme requires a large amount of energy. A suitable approach must be implemented. In industrial

networks, for example, the implemented technique could be filtering out invalid data at the gateway or at the device itself, or monitoring the pace of receiving or cyclically scheduling the collection of devices. The adaptation layer enriches, aggregates, fuses, or compacts Data.

3. **Storage Data Categorization**

Data is used in services, corporate operations, and business intelligence. True, useful, and important data can be processedin three ways: data alone, data plus results of processing, and just the results of data analytics. The following are three storage cases:

Data that must be accessed, referenced, or audited regularly in the future, and thus data itself must be stored. Data that only has to be processed once, and the reports are used later using analytics, and both the data and the results of processing and analytics are saved. The benefits in this case include fast visualisation and report generation without the need for reprocessing. In addition, the data is archived for future reference or auditing.

Online, real-time, or streaming data must be analysed, and the outcomes of this collection and interpretation must besaved.

Data from a wide range of computers and sources is classified as Internet of Things in a fourth group. As Big data, data is stored in files on a disc, in a data centre, or in theCloud.

5.5 GATHERING SOFTWARE FOR EVENTS

Events may be produced by a computer. When thetemperature exceeds a predetermined value or falls below a threshold, for example, a sensor will cause a case. As pressure in a boiler reaches a critical point, a pressure sensor produces an incident that requires treatment. An ID may be allocated to each case. A logic value either sets or resets the state of an occurrence. Logic 1 applies to an incident that has been createdbut has not yet been acted upon. Logic 0 applies to an incidentthat has been created and operated on, or that has not yet been generated. In applications, a

software component can assemblethe events (logic value, event ID, and system ID) and even add a date and time stamp. Program is used to compile events from IoTs and logic-flows.

Data storage

A data storage is a data archive that contains a collection ofitems that are integrated into the store.

The below are the characteristics of a data store:

Classes, which are described by the database schemas, are used to model objects in a data-store.

A data store is a broad term. Databases, relational databases, flat files, spreadsheets, mail servers, email servers,directory utilities, and VMware are examples of data warehouses.
A data store can be spread across several nodes. A distributed data store is an example of Apache Cassandra.
A data store can be made up of several schemas or of data in a single scheme. A relational database is an example of a single scheme data store. In English, a repository is a collection of people who work together to find needed items, unique material, or skills. For eg, a collection of artist's paintings. A database is a collection of data that can be used for reporting. Analytics, method, information exploration, and intelligence are all examples of intelligence. Another kind of archive is a flat disc.

Management of Data Centers

A data centre is a building that houses several banks of computers, servers, massive memory facilities, a high-speed network, and Internet access. The centre offers data security and safety through the use of specialized technologies, full data backups with data recovery, redundant data connectivity links, and full system power and energy supply backups.
Data centres are used by large manufacturing CHAPTERs, banks, railways, airlines, and other organizations where data isimportant. A dust-free, heating, ventilation, and air conditioning (HVAC), cooling,

humidification, and dehumidification devices, pressurization system, and physically stable atmosphere are also available in Data centers. The data centre manager is in charge of both operational and IT concerns, device and server processes, data entries, data protection, data quality control, network quality control, and the management of data processing resources and applications.

Server Management

Database management entails overseeing the provision of services, as well as the installation and upkeep of all devices connected with the server. A server must be available 24 hours a day, seven days a week. Server management entails overseeing the following:

Short response times when the device or network fails High protection requirements are achieved by regular system servicing and updating.

Device upgrades on a regular basis with cutting-edge setups

Enhanced results

All essential services are monitored and alerted via SMS and email.

Device security and privacy Maintaining data protection and privacy

At the organisation, there is a high level of confidentiality and ethics, as well as efficient preservation of data, archives, and databases.

Protection of customer data or enterprise internal records from attackers such as spam emails, unauthorised server access, viruses, malware, and worms

Both operations must be meticulously recorded and audited.

9. Storage Space

Consider RFID-tagged products. When items are moved from one location to another, monitoring or inventory management applications include the IDs of the goods as well as the locations. Spatial storage is storage in the form of a spatial database that is optimised for storage and later accepts requests from applications. Assume a city has a wireless map for parking spaces. Spatial data is data that describes structures in a geometric space. Geometric structures that can be represented in spatial databases include points, lines, and polygons. Databases for 3D objects, topological coverage, linear networks, triangular irregular networks, and other

complex structures can all be represented by spatial databases. Spatial databases with additional features allow for more effective retrieval. RFIDs, ATMs, vehicles, ambulances, traffic signals, streetlights, and waste bins are only a few examples ofhow spatial databases are used.

Spatial databases are ideal for spatial searches. A spatial database can execute standard SQL queries, such as pick statements, as well as a wide range of spatial operations. The following characteristics are found in a spatial database:

Geometry constructors may be used. Creating new geometries, for example.

The vertices can be used to describe a form (points or nodes)

Can execute observer functions by using queries that return precise spatial information, such as the position of a geometric object's centre.

Can conduct spatial calculations, such as calculating distances between geometries, line lengths, polygon regions, and other parameters.

Can use spatial functions to change current features to new ones and can use true or false style queries to predicate spatial relationships between geometries.

5.6 COMPUTING FOR IOT/M2M APPLICATION

Below are a few examples of traditional data processing and storage methods:

- Device data is saved at a local registry for the device nodes.Communicating with the computers and storing their data in folders on portable media such as micro SD cards and machine hard discs.
- Communicating with and storing data and computation outputs in a dedicated data store or organising node on a local level.
- Communicating and storing data at a local node of a distributed DBMS

- Communicating and saving in a distributed DBMS at a remote node
- Internet communication and data storage in a web or business cloud data store
- Communication over the Internet and cost savings at a data centre for a business

The cloud is a next generation method for collecting, storing, and computing data. Cloud computing is a data processing, storage, networking, and service delivery paradigm. describes cloud infrastructure service models in a software architectural term known as "all as a service" Describes IoT-specific cloud-based applications, including Xively and Nimbits. AWS IoT, Cisco IoT, IOx and Fog, IBM IoT Foundation, and TCS Connected Universe are examples of platforms (TCS CUP). Figure depicts various data processing, storage, and computational processes. The diagram depicts I data collection from devices or sensor networks at the system web site, (ii) local files, (iii) dedicated data store at the coordinating node,

(iii) local node in a clustered DBMS, (iv) Internet-connected data centre, (v) Internet-connected server, Internet-connected distributed DBMS nodes, and (vii) cloud storage and facilities.

The cloud computing model represents a significant advancement in information and communications technology (ICT). The new paradigm collects, stores, and computes data using XAAS in Internet-connected clouds. The following are the main terminology and definitions that must be known before learning about the cloud computing platform.A resource is one that can be read (used), written (created or altered), or executed (processed). A route specification is a resource as well. The resource is atomic (not more divisible) data that can be used during computations. A resource can have several instances or just one instance. A resource may also be a data point, pointer, data, object, data store, or process. Cloud platform for IoT/ M2M is shown in Figure 5.4

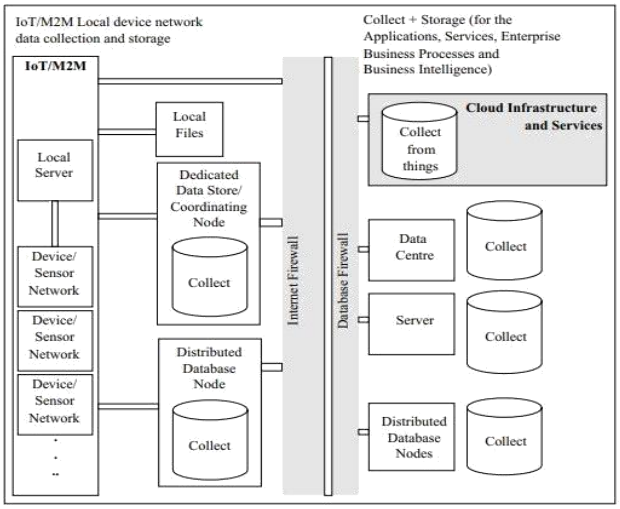

Figure 5.4: Cloud Platform for Iot/M2M

Sensors or devices Collection of network data at a computer Local-server, local archives, dedicated data store, ata coordinating node, a distributed DBMS local node, an Internet-connected data centre server, server or distributed database nodes, or a cloud service

An operating system (OS), memory, network, computer, applications, or application are all examples of machine resources. The term "environment" refers to a setting forprogramming, programme execution, or both. For eg, cloud9 online offers an open programming framework for theBeagle Bone board for the development of IoT devices; a Windows environment for application development and execution; and a Google App Engine environment for the creation and execution of web applications written in Python or Java. The term platform refers to the specific hardware, operating system, and network that are used for software applications or services that allow programmes to be run or built.

A framework can provide a browser and APIs that can be used as a foundation for running or developing other applications. Edge computing is a method of computing that shifts the boundary of computing applications, data, and resources away from centralised nodes and toward

IoT data generating nodes, i.e. at the network's logical extremes.

2 IoT system nodes are pushed by incidents, triggers, notifications, and communications, and data is gathered from remote centralised database nodes for enrichment, storage, and computations. Pushing computations from centralized nodes allows for the use of energy at system nodes,

which could be needed in low power lossy networks. The computation may also be categorised as local cloud edge computing, grid computing, or mesh computing. The nodes may be handheld, part of a wireless sensor network, or distributed cooperatively in peer-to-peer and ad-hoc networks. Spread computing is the processing and use of resources that are distributed through various computing environments via the Internet. The tools are logically connected, which means they communicate with each other using message passing and openness terms, are cooperating with each other, are movable without disrupting computations, and can be thought of as a single computing machine (location independent).

A service is a piece of software that offers capabilities as well as logically clustered and encapsulated functionalities. An programme invokes a service in order to make use of its resources. A service has a definition and methods of experimentation, such as direct use or by a service broker. The service is bound by the Service Level Agreement (SLA) that exists between the service (provider end point) and the application (end point). One provider may make use of another.

According to the W3C, a Web Service is an application defined by a URI that is represented and discovered using the XML-based Web-Service Description Language (WSDL). A web server communicates with other providers and applications using XML messages, and objects are exchanged via Internet protocols.
Service-oriented architecture is made up of modules that are deployed as individual services that can be dynamically bonded and coordinated and have loosely coupled implementations, with messages used to communicate between them. Orchestrating is a mechanism that determines the order in which programmes are called (in sequence and in parallel) as

well as the data and message transfers.
Online computing is the use of tools at the computing environment of a web server(s) or web servers over the Internet.
Grid computing is the use of a pooled, integrated grid of computing services and ecosystems instead of web servers.
Utility computing is defined as computing that focuses on service levels with the optimum amount of resources allocated when needed and uses shared resources and environments to host applications. The utilities are used by the applications.
Cloud computing is the use of a range of resources accessible over the Internet to provide computational capabilities on a service provider's platform for linked networks, enabling distributed grid and utility computing.
A Key Performance Indicator (KPI) is a range of values that typically consists of one or more raw monitored values, including minimum, average, and maximum values that define the size. A service must be fast, dependable, and safe. The KPIs monitor the achievement of these goals. A collection of values, for example, may be associated with Quality of Service (QoS) characteristics such as bandwidth efficiency, data backup capabilities, peak and average workload handling power, ability to manage specified volume of demand at various times of the day, and ability to provide defined total volume of service. A cloud provider should be able to meet the minimum, average, and maximum KPI values specified in the SLA.
Localization refers to the monitoring of cloud computing content use by assessing the localisation of the QoS level and KPIs.

When the service usage switches to a position with equivalent QoS level and KPIs, the content usages and computations resume without interruption. For example, if the app developer changes, continue to use the same cloud model. Elasticity means that an application can instal both local and remote programmes or utilities and only release them after the application has finished using them. The expenses are borne by the customer based on consumption and KPIs.

Measurability (of a good or service) is something that can be calculated

for managing or tracking and allows for reporting on resource or service quality.

The homogeneity of various computing nodes in a cluster or clusters corresponds to convergence with the kernel that allows for the automated transfer of processes from one homogeneous node to another. Each computing node's system software can ensure the same storage representation and processing results.

Resilient computing refers to the capacity to provide and sustain agreed QoS and KPIs in the face of established problems, specified and acceptable resilience metrics, and service protection. 4 The problems can range from minor to major, such as a misconfigured computing node in a network to natural disasters. The strength or desire to revert to the original shape, place, etc. after being bent, compressed, or extended, according to an English Cambridge dictionary.

Scalability in cloud computing refers to an application's ability to instal smaller local infrastructure as well as remotely accessible servers and resources, and then raise or decrease utilisation while incurring costs on increasing scales.

The term "maintenance" in cloud systems refers to the upkeep of databases, applications, computing resources, services, data centres, and servers, which are the responsibility of remotely linked cloud networks at no cost to the customer.

XAAS is a software architecture concept that allows for the deployment and creation of applications as well as the provision of services through the web and SOA. A computing model is to combine dynamic systems and resources (Section 5.3.5) and to deploy a cloud network using the XAAS concept.

The term "multitenant cloud model" refers to the ability of multiple users to access a cloud network and computing environment while paying according to the agreed-upon QoS and KPIs, which are specified in

separate SLAs by each user. The users pool resources, but each user pays separately. The subsections that follow explain the cloud computing paradigmand implementation models.

5.7 CLOUD COMPUTING TRENDS AND SERVICES

Cloud computing refers to a set of resources that are accessible over the Internet. The computing capability is provided by the cloud. Cloud storage is the deployment of a cloud-service provider's technology. The technology is built on a utility or grid computing or webservices environment, which involves a network, device, grid of computers or servers, or data centres. Much like we, as energy consumers, do not need to know about the origins and underlying infrastructure for electricity supply provision, a user of a computing service or programme does not need to know about how theinfrastructure deploys or the specifics of the computing environment. Much as the user does not need to understand the Intel processor inside a device, the user does not need to understand the data, computing, and information in the cloud as part of the services. Similarly, the cloud systems are used as a utility.

The cloud infrastructure has the following features:

1. Wide data storage infrastructure for computers, RFIDs, industrial plant equipment, vehicles, and device networks

2. Computing resources such as analytics, IDE, and so on (Integrated Development Environment)

3. Collaborative networking and the exchange of data stores

5.8 CLOUD PLATFORM APPLICATIONS

Cloud platforms are used to link computers, data, APIs, apps, and services, as well as people, organisations, companies, and XAAS.

The below is a basic philosophical structure of the InternetCloud:

Internet Cloud + Clients = End-user apps and utilities with "no boundaries and no partitions."

A platform, which contains the operating system (OS), hardware, and network, is where an application or service runs. Initially, many programmes can be built to run on various platforms (OSs, hardware and networks). Applications and utilities must coexist on a single network and operating environment. The cloud storage and networking systemprovides a virtualized environment, which refers to a running environment that appears to all software and utilities as one, but in reality can include two or more running environments and platforms.

Virtualization is a term used to describe the process

A feature of a virtualized architecture is that it allows programmes and utilities to run in their own execution environment (heterogeneous computing environment). Any ofthem stores and executes in isolation on the same network, despite the fact that it can directly execute or access a collection of data centres, servers, distributed services, and computing systems. Remotely hosted applications or services that are accessible via the Internet can be conveniently installed at a user application or service in a virtualized environment, providing the Internet or other communicationsare available.

Applications do not need to be aware of the platform; allthat is needed is Internet access to the platform, which isreferred to as a cloud platform. The computing is known ascloud storage. Cloud computing refers to this kind ofcomputing. In the same way as online applications are hosted on web servers, these services are referred to as cloud services. Storage virtualization occurs when a user programme or utility accesses physical storage through an abstract databaseinterface, file system, logical drive, or disc drive, despite thefact that storage may be available via several interfaces orservers. For example, Apple iCloud provides storage to an individual or user community, allowing them to sharealbums, songs, images, data stores, edit files, and collaborate
with other members of the user group.

Network Function Virtualisation (NFV) refers to a user programme or utility accessing services that appear as if they are on a single network, despite the fact that the network access to the resources could be through various resources and networks. Server virtualisation ensures that a user programme can control several servers rather than just one.

The term "virtualised desktop" refers to the ability of a user programme to modify and deploy multiple desktops, even though the user accesses them from their own computer platform (OS), which can in turn be multiple OS and platforms or remote computers.

5.8.1 CLOUD COMPUTING FEATURES AND ADVANTAGES

Essential features of cloud storage and computing are:

1. On demand self-service to users for the provision of storage, computing servers, software delivery and server time
2. Resource pooling in multi-tenant model
3. Broad network accessibility in virtualised environment to heterogeneous users, clients, systems and devices
4. Elasticity
5. Massive scale availability
6. Scalability
7. Maintainability
8. Homogeneity
9. Virtualisation
10. Interconnectivity platform with virtualised environment for enterprises and provisioning of in-between Service Level Agreements (SLAs)
11. Resilient computing
12. Advanced security
13. Low cost

5.8.2 CLOUD COMPUTING CONCERNS

Concerns in usage of cloud computing are:

1. Requirement of a constant high-speed Internet connection
2. Limitations of the services available
3. Possible data loss
4. Non delivery as per defined SLA specified performance
5. Different APIs and protocols used at different clouds
6. Security in multi-tenant environment needs high trust and low risks
7. Loss of users control

5.9 CLOUD DEPLOYMENT MODELS

The four cloud deployment models are as follows:

Public cloud: This model is made available to the public by educational organisations, companies, government institutions, or corporations or enterprises.

Private cloud: This model is only available to institutions, sectors, corporations, or organisations, and is intended for private use within the organisation by staff and associated users.

Private cloud: This model is intended solely for use by a comprised of organisations, companies, corporations, or corporations, as well as within the same organisation, staff, and related users. Security and enforcement considerations are defined by the participating community members.

A hybrid cloud is a set of two or more distinct clouds (public, corporate) with distinct data stores and applications that connect to implement proprietary or common technologies.

Cloud platform architecture is a virtualized network architecture that consists of a cluster of linked servers distributed through data centres and Service Level Agreements (SLAs) between them. A cloud infrastructure handles

and tracks services, as well as automatically provisioning networks, servers, and storage. Utility, grid, and remote networks provide cloud platform software and network services. Amazon EC2, Microsoft Azure, Google App Engine, Xively, Nimbits, AWS IoT, CISCO IoT, IOx and Fog, IBM IoT Foundation, and TCS Connected Universe Network are examples of cloud systems.

Styles with anything as a service and cloud services

The cloud links computers, data, software, resources, individuals, and businesses. Cloud systems should be thought of as a delivery infrastructure a service that connects resources (computing functions, data stores, processing functions, networks, servers, and applications) and provides coordination between them.

A basic equation can be used to describe cloud computing:

Cloud computing is comprised of SaaS, PaaS, IaaS, and DaaS. **Software as a Service (SaaS)** is an abbreviation for Software as a Service. On-demand functionality is made available to an application or utility. SaaS is a service model in which software or utilities are deployed and hosted in the cloud and made accessible to service users across the Internet on demand. The cloud service provider is responsible for programme control, maintenance, updating to new versions, and hardware, platform, and resource specifications.

Platform as a Service (PaaS) is an abbreviation for Platform as a Service. The software is made open to an application developer on request. PaaS is a service model in which programmes and services are developed and executed using a network (for computation, data storage, and delivery services) that is made available to the application developer on demand through the Internet. The cloud service provider is responsible for the platform, network, resources, servicing,updating, and security as specified by the developers.

Infrastructure as a Service (IaaS) is an abbreviation forInfrastructure as a Service. The technology (data warehouses, servers, data centres, and

networks) is made available to an application customer or creator on request. The developer instals and manages the OS image, data store, and framework on the infrastructure. IaaS is a service model in which apps create or use technology that is made accessible on demand from the Internet on contract (pay as you go in a multi-tenancy model) by a developer or customer. The cloud service provider is responsible for IaaS operating systems, networks, and security.

Data as a Service (DaaS) is an abbreviation for Data as a Service. Data in a data centre is made available on demand to a customer or creator of an application. DaaS is a service model in which a data centre or data warehouse is made available to an organisation over the Internet on demand for rent (pay as you go in a multi-tenancy model). The data centre service provider is responsible for data centre management, 24x7 power, control, network, repair, scaling up, data replicating and mirror nodes and networks, and physical protection.

CHAPTER 6

IOT PHYSICAL SERVERS AND CLOUD

6.1 INTRODUCTION

The Internet of Things (IoT) refers to the internet- connected technologies we use to carry out the operations and services that help us live our lives. Cloud computing, which serves as a sort of front end, is another component package that will help IoT excel. Cloud infrastructure is a growing service that provides many benefits to IOT. It is built around the principle of empowering people to conduct standard computing operations using services provided completely over the internet. A worker will need to complete a large assignment that must be sent to a manager, however they may run into memory or space limitations on their computing system. Memory and resource limitations can be reduced if an application is distributed on the internet rather than locally. Since the data is handled centrally by a computer, the worker may complete their job using a cloud computing service. Another scenario is that you have an issue with your mobile computer and need to reformat or reinstall the operatingsystem. You will use Google Images to upload your photos toan online cloud service. After the reformat or reinstall, you can either transfer the images back to your laptop or open them on the internet wherever you want.

Concept

In fact, cloud storage and IoT are inextricably linked. The proliferation of IoT and the exponential evolution of related technology has resulted in a widespread interconnection of objects. This has resulted in massive volumes of data that must be collected, analysed, and obtained. Cloud computing as aparadigm for Internet of Things collection and analytics. Though IoT is thrilling in and of itself, the true breakthrough would come from integrating it with cloud computing. The integration of cloud storage and IoT would allow new surveillance services as well as efficient analysis of sensory data streams. Sensory data, for example, can be imported and

saved in the cloud before being used intelligently for smart sensing and actuation with other smart devices. Ultimately, the aim is to be able to turn data into information and use that insight to take efficient, cost-effective intervention. The cloud essentially acts as the brain for better decision-making and internet-based communications. As IoT enters the cloud, however, new problems emerge. There is an immediate need for novel network architectures that connect them seamlessly. During integration, critical considerations include quality of service (QoS) and quality of experience (QoE), as well as data protection, anonymity, and dependability. Integrating apps, storage systems, monitoring software, visualisation services, analytics tools, and client distribution are all part of the integrated architecture for practical mobile computing and interfacing. Cloud storage is a practical utility-based architecture that allows enterprises and consumers to access services on demand from any location and at any time. Cloud Computing Applications is shown in Figure 6.1

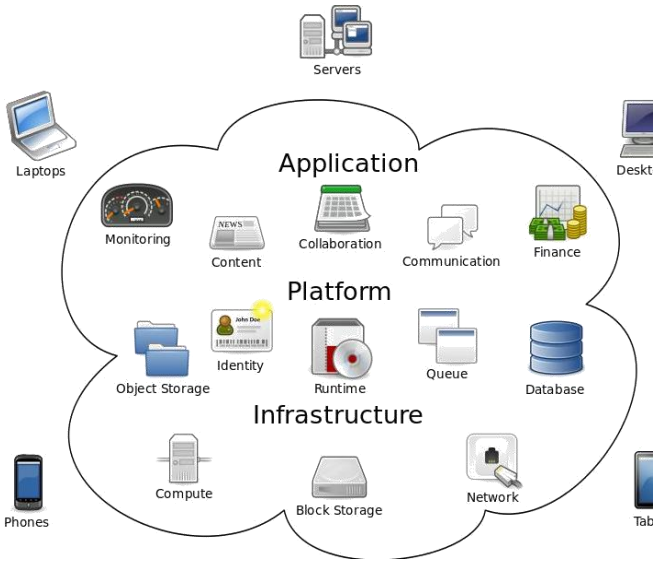

Figure 6.1: Cloud Computing Applications

6.2 CHARACTERISTICS

To begin, IoT cloud storage is an on-demand self-service, which means it is available anytime you need it. Cloud hosting is a web-based technology that can be used without the need for special help or authorization from others; however, you must have at least some level of internet connectivity. Second, IoT cloud storage requires large network connections, which means it provides a variety of networking options. Cloud storage capabilities can be accessed via a wide range of internet-connected devices, including smartphones, handheld devices, and notebook computers. Because of this degree of ease, users can access certain services in a multitude of ways, including from older devices. This, once again, stresses the importance of network access points. Third, cloud computing enables resource pooling, which means that information can be exchanged with others who know where and how (and have permission) to access the resource at any time and from any place. This facilitates greater cooperation or closer relationships with other users. From the standpoint of the Internet of Things, just as we can conveniently allocate an IP address to any "thing" on the globe, we can exchange the "address" of cloud-based secured and saved information with others and pool resources.

Fourth, cloud infrastructure has high elasticity, which allows customers to easily scale the technology to their needs. You can easily and efficiently change the device configuration, add or delete accounts, expand disc capacity, and so on. This feature would enhance IoT by offering elastic processing power, storage, and networking.

Finally, IoT cloud storage is a calculated operation, which means you get what you pay for. Storage, encoding, bandwidth, and active user accounts can all be easily measured by providers inside your cloud case. Because of the pay per use (PPU) model, the costs scale with your consumption. In IoT terms, it's analogous to the ever- expanding network of physical objects with an IP address for internet connectivity, and the communication that occurs between these objects and other internet-enabled devices and systems; similarly to the cloud service, the service rates for that IoT infrastructure can scale with usage.

6.3 DEPLOYMENT SERVICE MODELS

In cloud computing, data distribution is divided into three categories: software as a service (SaaS), application as a service (PaaS), and infrastructure as a service (IaaS) (IaaS).

Software as a service (SaaS) delivers cloud-based services to end users through a web interface or service-oriented architecture-based web service technologies. On the application layer, these services can be viewed as ASP (application service provider). Typically, a single organisation that uses the service can manage, administer, and provide maintenance to ensure that it will be used consistently for a long period of time.

PaaS (Platform as a Service) refers to the actual environment for creating and provisioning cloud software. This layer's primary users are developers who choose to buildand run a cloud framework for a specific reason. The framework (a series of essential core services) promoted and offered a proprietary language to simplify coordination, reporting, billing, and other issues such as startup, as well as to ensure an application's scalability and functionality. Disadvantages can include limitations in the programming languages supported, the programming model, the ability to access services, and long-term persistence.

Infrastructure as a service (IaaS) offers the hardware and software required by a company to create a tailored computing environment. Computing services, data storage resources, and the connectivity channel are interconnected with these critical IT resources to ensure the stability of cloud-based applications.This stack models may be referred to as the IoT medium, as they are used and expressed by users in various ways to maximise interoperability. Connecting vehicles, wearables, TVs, laptops, exercise devices, robotics, ATMs, and vending machines, as well as the vertical apps, security and technical services, and analytics systems that go with them, is part of this.

6.3.1 DEPLOYMENT MODELS

In cloud computing, there are four implementation models: private cloud, public cloud, shared cloud, and mixed cloud.

A private cloud has software that is only available to a single entity of different users, such as business Communities. It could be leased, maintained, and run by the company, a third party, or a hybrid of the two, and it could be on or off premises.

A digital cloud is made available to the general public for free access. Anyone with access to the internet can purchase services from the public cloud. (An example of a major public cloud provider is Amazon Web Services.) This model is appropriate for market conditions that necessitate the management of load spikes and the software used by the business, operations that would otherwise necessitate a larger investment in infrastructure. As a result, public cloud aims to reduce capital spending and operational IT costs. A community cloud is maintained and used by a single entity of organisations who have similar objectives, suchas specific security needs or a common purpose.

Finally, a hybrid cloud incorporates two or more separate private, cooperative, or public cloud infrastructures, allowing them to remain distinct entities while being linked through structured or proprietary technologies that allows data and device portability. Non-critical information is often outsourcedto the digital cloud, while business-critical services and data remain under the organization's jurisdiction.

6.4 CLOUD STORAGE API

A cloud computing facility An API is an application programme interface that links a locally-based application to a cloud-based storage server, allowing users to send data to it as well as view and interact with data stored in it. The cloud computing infrastructure, including tape or disk-based storage, is yet another target interface to the programme. An application programme interface (API) is a piece of code that enables two

software programmes to communicate with one another. The API specifies how a developer can write a programme that requests services from an operating system (OS) or other application. APIs are introduced by function callsmade up of verbs and nouns. The syntax needed is defined in the documentation of the application being named.

APIs are made up of two interconnected components. The first is a standard that specifies how information is shared between applications, using a request for processing and a return of the required data. The second is a programme interface written to the standard and made available for use in any form. The software that wishes to use the API's functions and functionality is said to name it, and the software that produces the API is said to publish it.

6.4.1 ESSENTIAL OF API's FOR INDUSTRY.

The internet, applications designed to share information over the internet, and cloud computing have all contributed toa growth in interest in APIs in general, and services in particular. Software that was once custom-developed for aparticular use is often often written by referencing APIs that have widely usable functionality, reducing development time and expense and lowering error danger. Over the last decade, APIs have increasingly increased product efficiency, and the growing number of network services exposed by APIs by cloud vendors is also supporting the development of cloud- specific applications, internet of things (IoT) initiatives, and apps to support mobile devices and consumers.

6.4.2 TYPES OF API's

APIs are classified into three types: local, web-like, and program-like.

1. The original type, from which the name was derived, is local APIs. They provide operating system or middleware services to application systems. Local APIs include Microsoft's.NET APIs, the TAPI (Telephony API) for speech applications, and database control APIs.

2. Web APIs are intended to display often used services such as HTML pages

and are accessible using a standard HTTP protocol. A web API can be accessed from any web URL. Since the publisher of REST interfaces does not save any data internally between requests, web APIs are also referred to as REST (representational state transfer) or RESTful. As a result, requests by several users will be mixed together, much like they will onthe internet.

3. Application APIs are built on remote procedure call (RPC) technology, which allows a remote programme component appear to the rest of the software to be local. APIs for service- oriented architecture (SOA), such as Microsoft's WS-series of APIs, are examples of application APIs.

6.5 IOT - CLOUD CONVERGENCE

The scalability, performance, and pay-as-you-go design of cloud computing infrastructures will support the Internet of Things. Indeed, since IoT applications generate large amounts of data and have numerous computational components (e.g., data processing and analytics algorithms), integrating them with cloud computing infrastructures can provide opportunities for cost-effective on-demand scaling. Consider the following settings as notable examples:

1. A Small and Medium-Sized Enterprise (SME) is creating an Internet of Things (IoT) product for energy management.

2. Homes and buildings that are clever. By broadcasting the product's data (e.g., sensors and WSN),

3. Data into the cloud, it would be able to meet its expanding needs in a flexible and cost-effective manner.

If the SMEs acquires more clients and completes more software deployments, it is ableThe cloud-based implementation of IoT systems andapplications will support a smart city. Many IoT technologies, such as those for smart energy management, smart water management, smart transportation management, residenturban mobility, and others, are likely to be deployed by a city. These systems include a variety of sensors and instruments, as well as computational elements. Furthermore, they are

likely to generate massive amounts of data. The use of the cloud allows the city to store these data and software at a low cost. Furthermore, the cloud's elasticity will directly enable not only the extension of these programmes, but also the accelerated introduction of new ones without significant questions about the provisioning of the necessary cloud computing services.

A cloud storage platform that provides public cloud services will expand them to the IoT region by allowing third parties to access its resources in order to incorporate IoT data and/or computational modules that operate over IoT computers. The operator can provide IoT data connectivity and services on a pay-as-you-go basis by allowing third parties to access its infrastructure capabilities and paying them in a utility-based manner.

These illustrative examples demonstrate the value and need of integrating IoT and cloud computing technology. Despite these advantages, this integration has often been difficult, owing to the contrasting properties of IoT and cloud infrastructures.

In particular, IoT devices are location-specific, resource-constrained, costly and inherently inflexible (in terms of resource access and availability). Cloud computing capabilities, on the other hand, are usually location-independent and affordable, while still having rapid and flexibly elasticity. To address these incompatibilities, sensors and systems are virtualized prior to incorporating their data and services in the cloud, allowing them to be distributed across all cloud resources. Additionally, service and sensor discovery functionalities are being implemented on the cloud to allow the discovery of services and sensors that are located in various locations.

Based on these ideas, IoT/cloud fusion activities have been ongoing for over a decade, dating back to the early days of IoT and cloud computing. Early research activities (from 2005 to 2009) in the research commCHAPTERy centred on streaming sensor and WSN data in a cloud infrastructure. Since 2007, we've also seen the rise of public IoT clouds, as well as commercial efforts. The well-known Pachube.com

infrastructure was one of the first attempts (used extensively for radiation detection and production of radiation maps during earthquakes in Japan). Pachube.com has grown (as a result of many purchases and evolutions of this infrastructure)into Xively.com, which is now one of the most popular public IoT clouds. Nonetheless, there are plenty of other public IoT clouds, including ThingsWorx, ThingsSpeak, Sensor-Cloud, Realtime.io, among others. The list is by no means exhaustive. This public IoT clouds provide end-users who choose to instal IoT apps on the cloud with commercial pay-as-you-go access. Most of them provide developer-friendly tools for creating cloud apps, effectively serving as a PaaS for IoT in the cloud. IoT/internet infrastructures and associated services, including cloud computing infrastructures, can be grouped into the following models:

1 Infrastructure-as-a-Service (IaaS) (IaaS) IoT/Clouds: These services allow access to sensors and actuators in the cloud. The related business model entails IoT/Cloud providers acting as either data or sensor providers. IaaS services for IoT include resource access control as a condition for the provision of similar pay-as-you-go services.

2 IoT/Cloud Platform-as-a-Service (PaaS): This is the most common model for IoT/cloud services, since it is supported by all of the public IoT/cloud infrastructures mentioned above. As previously illustrated, most public IoT clouds provide a variety of tools and similar environments for application creation and implementation in the cloud. The key feature of PaaS IoT services is that they have access to data rather than hardware. When compared to IaaS, this is a direct differentiator.

3 SaaS (Software-as-a-Service) (SaaS) IoT/Cloud Computing: SaaS IoT services enable users to access full IoT- based software applications from the cloud, on-demand, and on a pay-as-you-go basis. When sensors and IoT devices are not available, SaaS IoT applications closely mimic traditional cloud-based SaaS applications. However, there are instances where the IoT dimension is strong and obvious, such as applications that include the collection of sensors and the combination of data from the selected sensors in integrated applications. Since they

have on-demand access to the servicesof several sensors, many of these systems are widely referred to as Sensing-as-a-Service. It is important to note that SaaS IoTapplications are usually designed on top of a PaaS architectureand support utility-driven business models based on IoT software and services.

These concepts and illustrations include a high-leveldescription of IoT and cloud integration, as well as why it is necessary and useful. More and more IoT apps are being integrated with the cloud to take advantage of its performance,business agility, and pay-as-you-go features. In the following tutorial chapters, we will show how to maximise the cloud's benefits for IoT by ensuring semantic interoperability of IoT data and services in the cloud, enabling advanced data analytics applications, as well as the integration of a wide range of vertical (silo) IoT applications that are now available in areas such as smart energy, smart transportation, and smart homes. We will also see how IoT/cloud convergence supports various fields and divisions of IoT, such as IoT-based wearablecomputing.

6.6 WAMP for IOT

Web Application Messaging Protocol (WAMP) is a Websocket sub-protocol that supports publish-subscribe and remote procedure call (RPC) messaging patterns. WAMP is shown in Figure 6.2

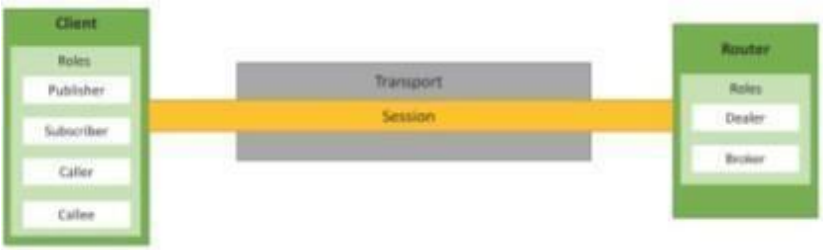

Figure 6.2: Web Application Messaging Protocol(Wamp)

- Transport: A transport channel binds two peers.

- Session: A session is a dialogue that takes place between two peers over a transport.
- lient: Clients are peers who can play one or more positions. Clients in the publish-subscribe model will play the following roles:
- Publisher: The publisher publishes events (including payload) to the broker's subject.
- Subscriber: A subscriber is someone who subscribes to a subject and receives events that include the payload.

The following functions are available to clients in the RPC model: –

- Caller: The caller sends calls to remote procedures with call claims. – Callee: The callee performs the procedures specified by the caller and returns the results to the caller.
- Router: Routers are peers that handle call and event routing. The Router serves as a Broker in the publish-subscribe model: – Broker: A broker functions as a router, routing messages released to a topic to all subscribers who have subscribed to that topic.
- In the RPC model, the Router serves as a Broker: –
- Dealer: The Dealer serves as a modem, routing RPC calls from the Caller to the Callee and results from the Callee to the Caller.
  Application Code: This code is executed on the Clients (Publisher, Subscriber, Callee, or Caller).

6.7 AMAZON EC2 – PYTHON EXAMPLE

Boto is a Python package that includes Amazon Web Services interfaces (AWS). In this case, the EC2 service is first connected to by calling boto.ec2.connect to region. This feature receives the EC2 field, AWS access key, and AWS secret key. After connecting to EC2, the conn.run instances function is used to open a new instance. This feature receives the AMI-ID,

instance sort, EC2 key handle, and protection category.

Amazon AutoScaling – Python Example

1 AutoScaling Service: The first step is to connect to the AutoScaling service by calling boto.ec2.autoscale.connect toregionfunction.
2 Launch Configuration: After connecting to the AutoScaling facility, a new launch configuration is created by calling conn.create launch config. The launch setup provides instructions for launching new instances, such as the AMI-ID, instance sort, protection classes, and so on.
3 AutoScaling Group: After a launch configuration is created, it is assigned to a new AutoScaling group. Conn.createauto scaling group is used to build an AutoScaling group. The AutoScaling group's parameters, such as the maximum and minimum number of instances in the group, the start setup, compatibility zones, the optional load balancer to use with thegroup, and so on.

Amazon AutoScaling – Python Example

```
#Creating auto-scaling policies
scale_up_policy = ScalingPolicy(name='scale_up',

adjustment_type='ChangeInCapacity', as_name='My-Group',

scaling_adjustment=1,cooldown=180)

scale_down_policy =ScalingPolicy(name='scale_down',

adjustment_type='ChangeInCapacity',

as_name='My-Group', scaling_adjustment=-1,cooldown=180)

conn.create_scaling_policy(scale_up_policy)

conn.create_scaling_policy(scale_down_policy)
```

Policies for AutoScaling:

1. After forming an AutoScaling commCHAPTERy, the scalingup and

scaling down policies are established.
2. A scale-up policy with adjustment form ChangeInCapacity and scaling adjustment = 1 is described in this case.
3. A scale-down policy of adjustment form ChangeInCapacity and scaling adjustment = -1 is described in the same way.

CloudWatch Alarms

```
#Connecting to CloudWatch

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

aws_access_key_id=ACCESS_KEY,
aws_secret_access_key=SECRET_KEY)

alarm_dimensions = {"AutoScalingGroupName": 'My-Group'}#Creating a scale-up alarm
scale_up_alarm = MetricAlarm( name='scale_up_on_cpu', namespace='AWS/EC2', metric='CPUUtilization', statistic='Average', comparison='>', threshold='70',
period='60', evaluation_periods=2,

alarm_actions=[scale_up_policy.policy_arn],
dimensions=alarm_dimensions)
cloudwatch.create_alarm(scale_up_alarm)#Creating a scale-down alarm

scale_down_alarm =MetricAlarm( name='scale_down_on_cpu',namespace='AWS/EC2',
metric='CPUUtilization', statistic='Average', comparison='<',threshold='40',
period='60', evaluation_periods=2,

alarm_actions=[scale_down_policy.policy_arn],

dimensions=alarm_dimensions)
cloudwatch.create_alarm(scale_down_alarm)
```

1. After the scaling policies have been established, the next move is to build Amazon CloudWatch alarms that will activate these policies.
2. The CPUUtilization parameter with the Average statistic and a threshold greater than 70% for 60 seconds is used to describe the scale-up alert. This warning is consistent with the previously created scale-up strategy. This warning is activated when the total CPU consumption of the group's instances exceeds 70% for more than 60 seconds.
3. The scaled-down warning is described in the same way, with a threshold of less than 50%.

6.8 PYTHON FOR MAPREDUCE

#Inverted Index Mapper in Python

#!/usr/bin/env python import sys for line in sys.stdin:doc_id, content =

line.split(__) words = content.split() for word in words: print

_%s%s_ % (word, doc_id)

The sample software is an inverted index mapper. Themap function reads data from standard input (stdin) and divides the tab-limited data into document-ID and document contents. The map function returns key-value pairs, with the key being each term in the document and the value being the document-ID.

Python for MapReduce

#Inverted Index Reducer in Python

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

for line in sys.stdin: # remove leading and trailing whitespace line = line.strip() # parse the input we got from mapper.py word, doc_id = line.split(__) if current_word

== word: current_docids.append(doc_id) else: if current_word: print _%s%s_ %

(current_word, current_docids) current_docids = [] current_docids.append(doc_id) current_word = word

An inverted index reducer programme is seen in the illustration. The map phase's key-value pairs are shuffled to the reducers and grouped by key. The reducer reads key-value pairs grouped by the same key from the standard input (stdin)and generates a list of document-IDs that contain the title. Thereducer's output consists of key-value pairs, where the key is a specific word and the value is a sequence of document-IDs in which the word appears.

Python Packages of Interest

1. JSON: JavaScript Object Notation (JSON) is a data-interchange format that is simple to read and write. JSON is an alternative to XML that is simple for computers to parse and create. JSON is composed of two structures: a set of name-value pairs (similar toa Python dictionary) and structured lists of attributes (e.g.. a Python list).

2. XML (Extensible Markup Language): XML (Extensible Markup Language) is a data format for structured text interchange. The Python minidom library implements the Document Object Model interface minimally and has an API close to that of other languages.

3. HTTPLib2 and URLLib2: HTTPLib2 and URLLib2 are Python libraries that are used in network/internet programming.

4. SMTPLib: The Simple Mail Transfer Protocol (SMTP) is a protocol that manages email transmission and routing between mail servers. To submit emails, the Python smtplib module provides an SMTP client session object.

5. NumPy: NumPy is a Python package for science programming. NumPy supports massive multidimensional arrays and matrices.

6. Scikit-learn: Scikit-learn is an open-source Python machine learning library that implements various machine learning algorithms for grouping, clustering, regression, and dimension reduction problems.

6.9 PYTHON WEB APPLICATION FRAMEWORK - DJANGO

Django is an open-source software development platform for creating Python-based web applications. In general, a web application architecture is a set of solutions, bundles, and bestpractises that enable the development of web applications andinteractive websites. Django is built on the Model-Template- View architecture, which separates the data model from the business rules and the user interface. Django offers a single API for connecting to a database backend. As a result, Django- based web apps will interact with a variety of databases without needing any code changes. Django is ideally designedfor cloud applications due to its simplicity in web application architecture combined with the strong features of the Python language and the Python ecosystem. Django is made up of an object-relational mapper, a network templating scheme, and a URL dispatcher built on regular expressions.

Django Architecture

1. Model: The model defines certain stored data and performsrelations with the database. Data in a web application can be contained in a relational database, a non-relational database, an XML format, and so on. A Django model is a Python class that defines the variables and methods for a specific data type.

2. Sample: A template is essentially an example of a Django web application. With a few extra placeholders, this is an HTML page. Django's modelling language can be used to generate a variety of text files (XML, email, CSS, Javascript, CSV, etc.)

3. Vision: The view is what connects the image to the prototype. The view is where the code that creates the web pages is written. The view decides what data is to be viewed, retrieves the datafrom the archive, and sends the data to the prototype.

6.10 CASE STUDIES

6.10.1 CASE STUDY 1: HOME AUTOMATION

An Internet of Things (IoT) software-based solution to home automation. Measurement of home conditions, monitoring of home equipment, and control of home entry via RFID cards, for example, and windows via servo locks, are common use-cases. However, the primary goal of this paper is to improve home protection by IoT. Monitoring and maintaining servo door locks, door alarms, surveillance cameras, surveillance vehicles, and smoke detectors, to ensure and optimize home safety and protection. A user can access the following features through a mobile application:

1. May activate or deactivate LED lights and track their status.
2. May use servo motors to lock and unlock doors and track whether the doors are locked or unlocked
3. May detect when the doors are closed or opened using infrared sensors
4. Is contacted by email if the door is left open for an extended period of time.
5. Is aware about who entered through the door when the camera records the face picture and emails it to him/her.
6. If the _re detector senses smoke, the user is alerted by email.
7. Has the ability to operate the surveillance car from everywhere in order to watch his or her residence.

Due to the reasons listed in the preceding paragraphs, I chose to work on the field of Home Automation via IoT as part of this research, especially in maintaining and ensuring protection and safety within the home. Home Automation is shown in Figure 6.3

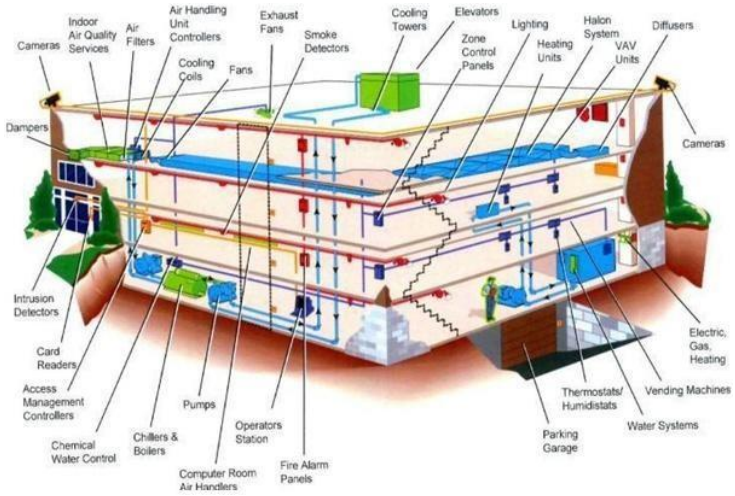

Figure 6.3: Home Automation

The Internet of Things (IoT) seeks to create a network of devices equipped with sensors that can store, analyse, interact, and share data over the internet. This results in more competitive industry, construction, energy efficiency, resource management, accurate health care, wiser business choices based on analysed data, safer transportation by smart vehicles that can connect with one another, smart home automation, and a plethora of other applications.

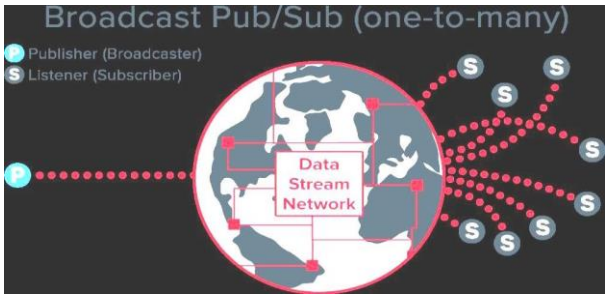

Figure 6.4: Publish/Subscribe Model

To control the various electrical devices attached to the system developed for the home automation project outlined in this article, a control CHAPTER, such as a computer, is required. The Raspberry Pi is a credit-card-sized device that can be plugged into a display that uses a regular keyboard and mouse to teach people of all ages how to program

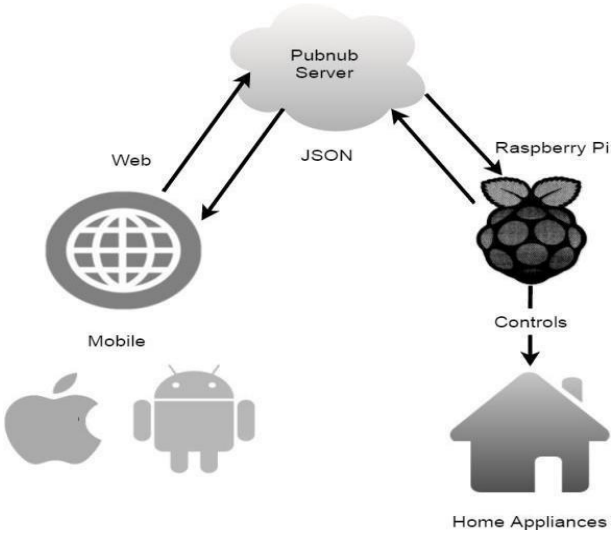

Figure 6.5: System Architecture Used In This HomeAutomation Project

Here is an overview of the steps involved in building the publish/subscribe model, as well as the system architecture used in this Home Automation project, to simplify it: Various cameras.

6.10.2 CASE STUDY 2: SMART CITY TAXONOMY

This section provides an IoT-related smart city taxonomy that categorises the literature based on current connectivity standards, major service providers, network modes, standardisation efforts, offered facilities, and critical criteria.

Protocols of Communication

To transport data between sensors and backend servers, IoT-based smart city realisation heavily relies on a plethora of short and long-distance networking protocols. Short-range wireless systems that are widely used include Zig-Bee, Bluetooth, Wi-Fi, Wireless Metropolitan Area Network (WiMAX), and IEEE 802.11p, which are mainly used in smart metering, e-healthcare, and vehicular connectivity. A variety of technologies, including GSM and GPRS, Long-Term Evolution (LTE), and LTE-

Advanced, are widely used in ITS applications such as vehicle-to-infrastructure (V2I), mobile e- healthcare, smart grid, and infotainment systems. Furthermore, LTE-M is regarded as a step forward in the evolution of cellular IoT. (C-IoT). 3GPP intends to increase coverage, battery lifespan, and system complexity in Release 13 [7]. Aside from well-known existing protocols, the LoRa Partnership standardises the LoRaWAN protocol to support smart city applications, with the primary goal of ensuring interoperability among multiple operators. Furthermore, SIGFOX is an ultra narrow band radio technology with a complete star-based infrastructure that provides a highly flexible global network for realising smart city applications while consuming very little electricity. A comparison of the main communication protocols2.

Providers of Services

According to Pike Research, the smart city industry will be worth hundreds of billions of dollars by 2020, with an average growth rate of nearly 16 billion. IoT has been identified as a possible source of business growth for service providers. As a result, well-known global service providers have already begun to investigate this innovative cutting-edge networking paradigm. Telefonica, SK Telecom, Nokia, Ericsson, Vodafone, NTT Docomo, Orange, Telenor group, and AT&T are major service companies that provide a variety of offerings and networks for smart city applications such as ITS and logistics, smart metering, home automation, and e-healthcare.

Network Types

To achieve a truly autonomous environment, IoT-based smart city applications depend on a variety of network topologies. Capillary IoT networks provide services over a limited distance. Wireless local area networks (WLANs), business area networks (BANs), and wireless personal area networks are some examples (WPANs). Indoor e-healthcare systems, home automation, and street lighting are among the implementation fields. ITS, mobile e-healthcare, and waste management, on the other hand, rely on large area networks (WANs), metropolitan area networks (MANs), and mobile communication networks. The networks

described above have distinct characteristics in terms of traffic, scale, coverage, latency requirements, and ability.

6.10.3 CASE STUDY 3: SMART ENVIRONMENT

Rapid advances in communication technology, as well as the exponential development of the Internet of Things (IoT), have allowed the real universe to seamlessly intertwine with actuators, sensors, and other computational elements while retaining continuous network connectivity. A smart ecosystem is formed by the continuous connection of the real world with computational components. A smart community seeks to assist and improve the skills of its inhabitants in carrying out their duties, such as negotiating new spaces and lifting large items for the elderly, to name a few. Several attempts have been made by researchers to use IoT to improve our lives and to explore the effect of IoT-based smart environments on human life. This paper examines the most recent research efforts to allow IoT-based smart environments. By developing a taxonomy focused on connectivity enablers, network modes, architectures, local area wireless requirements, aims, and features, we categorize and characterise the literature.Furthermore, the paper emphasises the extraordinary possibilities created by IoT-based smart ecosystems and their effect on human life. Any published case studies from various businesses are also discussed. Finally, we will go through opentesting issues for allowing IoT-based smart ecosystems. IoT based smart environment is shown in Figure 6.6

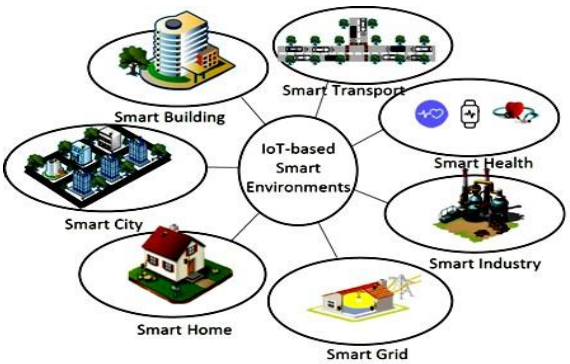

Figure 6.6: IoT Based Smart\Environments

Information technology's rapid advancement and miniaturization have allowed the incorporation of tiny sensors and processors into everyday items. This progress is further aided by significant advancements in fields such as portable appliances and electronics, ubiquitous computing, wireless sensor networking, wireless mobile communications, machine

Learning-based decision making, IPv6 support, human- computer interfaces, and agent capabilities, which are all contributing to the realisation of the dream of a smart world. A smart ecosystem is a connected small world in which sensor- enabled connected technologies collaborate to make dwellers' lives more comfortable. The word smart refers to the ability to acquire and apply information on one's own, and the term world refers to one's surroundings. As a result, a smart ecosystem is capable of acquiring information and applying it to respond to the needs of its occupants in order to improve their understanding of that environment.

The functional capabilities of smart objects are improved further by linking them to other objects through various wireless technologies. In this sense, IPv6 is critical due to many features such as improved authentication protocols, scalability in the case of billions of connected devices, and the removal of NAT barriers1. Kevin Ashton coined the term "Internet of Things" to describe the idea of linking smart devices to the Internet (IoT).

IoT is now gaining traction in a variety of industries, including hospitals, transportation, and manufacturing. Several research projects have been undertaken in order to combine IoT with smart worlds. The combination of IoT with a smart environment broadens the capabilities of smart devices by allowing users to view the environment from distant locations. Depending on the device specifications, IoT can be compatible with various smart environments. The study on IoT-based smart environments can be broadly divided into the following categories:

a) smart cities;
b) smart homes;

c) smartgrid;
d) smart buildings;
e) smart transportation;
f) smart health;
g) smart business exemplifies IoT-based smartworlds
IoT based smart Environment is shown in Figure 6.7

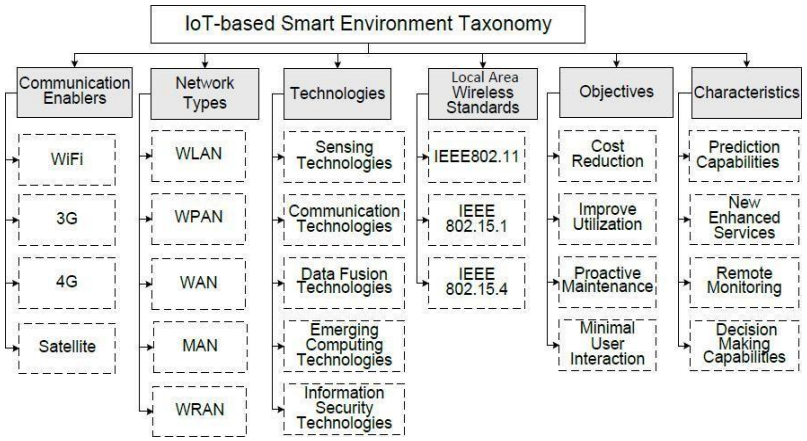

Figure 6.7: IoT Based Smart EnvironmentTaxonomy

The IoT-based smart world taxonomy. The taxonomy created is based on the following parameters: networking enablers, network modes, technology, wireless requirements, goals, and characteristics.

Communication Facilitators

Wireless technologies used to connect around the Internetare referred to as communication enablers. WiFi, 3G, 4G, and satellite are the most popular cellular Internet technology. WiFi is more commonly seen in smart homes, smart towns, smart transportation, smart industries, and smart building environments, while 3G and 4G are most commonly seen in smart communities and smart grid environments. Satellites are used in smart mobility, smart buildings, and smart grid applications. The table below provides a comparison of networking systems used in IoT-based smart environments.

Network Types

IoT-based smart environments rely on various forms of networks to execute shared activities that make residents' lives more secure. Wireless local area networks (WLANs), wireless personal area networks (WPANs), wide area networks (WANs), metropolitan area networks (MANs), and wireless regional area networks are the primary networks (WRANs). These networks vary in terms of complexity, data transmission, and assisted reachability.

Technological advances

IoT-based smart ecosystems combine different innovations to provide a comfortable and appropriate ecosystem. Sensing, networking, data fusion, emerging computing, and information security are examples of these developments. Sensing technologies are widely used to collect data from different locations and send it to a central location through communication technologies. Emerging computational systems, such as cloud computing and fog computing, which are distributed in a single location, make use of data integration technologies to integrate data from disparate sources. Smart environments also make use of information management tools to protect data confidentiality and consumer privacy.

Wireless Local Area Standards

IEEE 802.11, IEEE 802.15.1, and IEEE 802.15.4 are the most

widely used local area wireless protocols in IoT-based smart environments. This standard technology are used within the smart world to move data obtained from various sensors. IEEE

802.11 is a wireless technology that is found in smart houses, smart buildings, and smart cities. IEEE 802.15.1 and IEEE

802.15.4 have less coverage than IEEE 802.11 and are mostly found in sensors and other smart environment objects.

REFERENCES

- Villasenor, J. (2013). Embedded Systems Management and Design. IEEE Press.
- W. Dargie & C. Poellabauer (2010). Fundamentals of Wireless Sensor Networks. Wiley.
- ZigBee Alliance. ZigBee Specifications and Technical Documents.
- International Telecommunication Union (ITU). IoT Global Standards & Reports. Geneva.
- Jamil, T., & Kahn, J. (2018). IoT Protocols and Standards. IEEE Communications Magazine.
- McEwen, A., & Cassimally, H. (2014). Designing the Internet of Things. Wiley.
- Nadalin, A. (2012). Cloud Computing Concepts, Security & Architecture. IBM Technical Report.
- Pea, R. (2016). Python Programming: An Introduction to Computer Science. Franklin, Beedle & Associates.
- Rajkumar Buyya, et al. (2011). Cloud Computing: Principles and Paradigms. Wiley.
- Stallings, W. (2017). Computer Networking with Internet Protocols. Pearson Education.
- Tanenbaum, A. S., & Wetherall, D. (2010). Computer Networks. Prentice Hall.
- Tiwari, R. (2020). Fundamentals of Internet of Things. BPB Publications.
- Vermesan, O., & Friess, P. (2014). Internet of Things: From Research and Innovation to Market Deployment. River Publishers.

APPENDIX

Appendix B: List of Abbreviations

Abbreviation	Full Form
IoT	Internet of Things
M2M	Machine-to-Machine Communication
ICT	Information and Communication Technology
RFID	Radio Frequency Identification
NFC	Near Field Communication
SDN	Software Defined Networking
NFV	Network Function Virtualization
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
MQTT	Message Queuing Telemetry Transport
CoAP	Constrained Application Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
LPWAN	Low-Power Wide Area Network
BLE	Bluetooth Low Energy

CPU	Central Processing Unit
RAM	Random Access Memory
ROM	Read Only Memory
GUI	Graphical User Interface
WSN	Wireless Sensor Network
WLAN	Wireless Local Area Network
PAN	Personal Area Network
LAN	Local Area Network
WAN	Wide Area Network

Appendix B: Python Quick Reference

Basic Python Syntax

```
# Print statement
print("Hello IoT")
# Variable assignment
x = 10
y = 20
# Conditional
if x < y:
print("x is less than y")
# Loop
for i in range (5):
print(i)
```

Common Data Types

- int
- float
- str
- list
- tuple
- dict
- set

Appendix C: IoT Protocols Summary

Protocol	Layer	Purpose
Ethernet (802.3)	Link Layer	Wired LAN communication
Wi-Fi (802.11)	Link Layer	Wireless LAN
ZigBee	Link/MAC	Low-power wireless mesh
LoRa WAN	MAC	Long-range, low-power IoT
MQTT	Application	Pub/Sub messaging
CoAP	Application	Lightweight REST-like protocol
WebSocket	Application	Full-duplex communication

Appendix D: Sample IoT System Architecture

Stages:

- Sensors/Actuators

- IoT Gateways
- Edge Processing
- Cloud Analytics
- Applications (Mobile/Web Dashboard)

About the Author

Dr. P. Brundavani is a distinguished academician and researcher in the field of Electronics and Communication Engineering, with significant contributions to VLSI System Design and Embedded Technologies. She obtained her B.Tech in Electronics and Communication Engineering from G. Pulla Reddy Engineering College, Kurnool, affiliated with Sri Krishna Devaraya University, in 2004. She completed her M.Tech in VLSI System Design from Annamacharya Institute of Technology and Sciences (AITS), Rajampet, under Jawaharlal Nehru Technological University Anantapur (JNTUA), in 2010. She earned her Doctorate (Ph.D., 2024) from JNTUA, Ananthapuramu, for her research work titled: "Optimization of Power and Area Efficient EEG Signal Acquisition System."

Dr. Brundavani began her academic career as a Lecturer and progressed to the role of Associate Professor, demonstrating commendable dedication to teaching, research, and academic leadership. She has also served as the Dean of Research and Development (R&D), where she played a pivotal role in strengthening institutional research capacity, fostering innovation, and promoting collaborative scholarly activities.

She has contributed as a reviewer for international conferences, particularly those focused on sustainable and socially impactful technologies. Her areas of review and engagement include Semiconductor Technologies, VLSI Design, Biomedical Engineering, and the Internet of Things (IoT).

Her primary research interests include Digital Systems Design, VLSI System Design, Biomedical Signal Processing, and Embedded Systems. She has published over 45 research papers in reputed national and international journals and conferences, authored two technical books, and holds two patents, reflecting her strong commitment to innovation and scholarly excellence.

Beyond research, Dr. Brundavani has actively contributed to academia by organizing symposiums, technical festivals, workshops, and Faculty Development Programs (FDPs). She is a dedicated mentor nurturing student innovation and project-based learning.

Dr. Brundavani is an active member of professional bodies such as the Institution of Engineers (India) and the Institute of Electrical and Electronics Engineers (IEEE). She is currently associated with Ramireddy Subbarami Reddy Engineering College, Kavali, SPSR Nellore, Andhra Pradesh, where she continues her contributions to academic, research, and institutional development. She also serves as the Vice-President of the Institution's Innovation Council (IIC).

About the Author**

Dr. Shaik Rahamtula is an accomplished academician and researcher in the field of Electronics and Communication Engineering. He obtained his B. Tech in Electronics and Communication Engineering from Khader Memorial College of Engineering and Technology (KMCET), affiliated to Jawaharlal Nehru Technological University, Hyderabad, in 2009. He subsequently earned his M. Tech in Digital Electronics and Communication Engineering from QIS College of Engineering and Technology, affiliated to Jawaharlal Nehru Technological University, Kakinada, in 2012. He was awarded his Doctor of Philosophy (Ph.D.) in Electronics and Communication Engineering from VELS Institute of Science, Technology & Advanced Studies (VISTAS), Chennai.

Dr. Rahamtula currently serves as an Associate Professor and Additional Controller of Examinations at Ramireddy Subbarami Reddy Engineering College, Kadanuthala, Andhra Pradesh. With a strong academic background and significant administrative responsibilities, he has contributed extensively to teaching, curriculum development, and institutional quality processes.

His research expertise spans several advanced domains, including Face Recognition, Content-Based Image Retrieval, Image Compression, Image Watermarking, Pattern Recognition, Information Retrieval, Data Mining, and Wavelet Neural Networks. His scholarly contributions extend to the field of biomedical image analysis, and he is presently engaged in the research project titled: “Automatic Segmentation of Optic Disc and Cup Region from Medical Images for Glaucoma Detection.”

Dr. Rahamtula continues to contribute to the academic community through research publications, scholarly guidance, and active involvement in emerging technological advancements.

About the Author***

Dr. T. Jaya is an accomplished academician and researcher in the field of Electronics and Communication Engineering with extensive experience in teaching, research guidance, and scholarly contributions. She received her B.E. degree in Electronics and Communication Engineering from Anna University, CEG Campus, Chennai. She completed her M.Tech degree in Computer Science and Engineering from Sathyabama University, Chennai, and subsequently earned her Ph.D. in Electronics and Communication Engineering from the Vels Institute of Science, Technology & Advanced Studies (VISTAS), Chennai.

Dr. Jaya is currently serving as a Professor in the Department of Electronics and Communication Engineering at VISTAS, Chennai. With over 13 years of teaching experience, she has demonstrated exemplary commitment to academic excellence, mentorship, and research advancement.

Her research expertise spans Wireless Networks, Quantum Communication, and Underwater Communication, with notable contributions in emerging and high-impact technological domains. She has published more than 55 research papers in reputed Scopus-indexed and Web of Science journals, reflecting her strong research orientation and scholarly depth.

Throughout her academic career, Dr. Jaya has supervised over 30 undergraduate and postgraduate scholars, and is currently guiding 6 research scholars. She has successfully guided and produced 10 Ph.D. scholars, establishing herself as a respected research supervisor in her field.

Dr. Jaya continues to contribute significantly to the academic and research ecosystem through her teaching, publications, mentorship, and active engagement in advanced technological research areas.

About the Author****

Dr. D. Vishnu Vardhan is a distinguished academician and administrator with over two decades of experience in teaching, research, and institutional leadership. He received his B.Tech in Electronics and Communication Engineering from R.G.M. College of Engineering and Technology, Nandyal, in 2000, and his M.Tech in Computers and Communications Engineering from JNTU College of Engineering, Kakinada, in 2005. He was awarded his Doctorate (Ph.D.) from Jawaharlal Nehru Technological University Anantapur (JNTUA) in 2015.

Dr. Vishnu Vardhan began his academic career as a Lecturer at JNTUA College of Engineering, Ananthapuramu, and steadily progressed through various academic and administrative roles, ultimately rising to the position of Professor. Throughout his service, he has held several key institutional responsibilities, including Placement Officer, Officer In-Charge of Academic Section, Officer In-Charge of Physical Education, TEQIP-III Coordinator, Deputy Controller of Examinations, NSS Programme Officer, and Head of the Department.

He is currently serving as the Principal of JNTUA College of Engineering, Ananthapuramu (Pulivendula Campus), where he continues to provide academic leadership, oversee institutional development, and mentor faculty and students.

Dr. Vishnu Vardhan's primary research interests include Embedded Systems, Image Processing, and VLSI Design. He has published more than 75 papers in reputed national and international journals and conferences. He has also played an active role in academic events, having organized numerous Conferences, Workshops, Symposia, Technical Festivals, Faculty Development Programs (FDPs), and Short-Term Training Programs (STTPs).

He contributes actively to the academic fraternity as a Member of Board of Studies and Academic Council Member for several autonomous engineering institutions across Andhra Pradesh. He is also an active volunteer and member of professional bodies including ISTE, IE, and IAENG.