ATME COLLEGE OF ENGINEERING

13th KM Stone, Bannur Road, Mysore - 560 028

A T M E
College of Engineering

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

(DATA SCIENCE)

(ACADEMIC YEAR 2024-25)

Notes

Course: Internet of Things (21CS735)

COURSE CODE: 21CS735

SEMESTER: VII-2021 CBCS Scheme

INSTITUTIONAL MISSION AND VISION

Objectives

- To provide quality education and groom top-notch professionals, entrepreneurs and leaders for different fields of engineering, technology and management.
- To open a Training-R & D-Design-Consultancy cell in each department, gradually introduce doctoral and postdoctoral programs, encourage basic & applied research in areas of social relevance, and develop the institute as a center of excellence.
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels.
- To cultivate strong community relationships and involve the students and the staff in local community service.
- To constantly enhance the value of the educational inputs with the participation of students, faculty, parents and industry.

Vision

- Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

Mission

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow's society.
- To strive to attain ever-higher benchmarks of educational excellence.

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING AND ENGINEERING

(DATA SCIENCE &ENGINEERING)

Vision of The Department

- To impart technical education in the field of data science of excellent quality with a high level of professional competence, social responsibility, and global awareness among the students

Mission

- To impart technical education that is up to date, relevant and makes students competitive and employable at global level
- To provide technical education with a high sense of discipline, social relevance in an intellectually, ethically and socially challenging environment for better tomorrow
- Educate to the global standards with a benchmark of excellence and to kindle the spirit of innovation.

Program Outcomes(PO)

- **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

- **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

- **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

- **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

- **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice
- **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

- **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

- **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

- **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

- **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

- PSO1: Develop relevant programming skills to become a successful data scientist
- PSO2: Apply data science concepts and algorithms to solve real world problems of the society
- PSO3: Apply data science techniques in the various domains like agriculture, education healthcare for better society

Program Educational Objectives (PEOs):

PEO1: Develop cutting-edge skills in data science and its related technologies, such as machine learning, predictive analytic, and data engineering.

PEO2: Design and develop data-driven solutions to real-world problems in a business, research, or social environment.

PEO3: Apply data engineering and data visualization techniques to discover, investigate, and interpret data.

PEO4: Demonstrate ethical and responsible data practices in problem solving

PEO5: Integrate fields within computer science, optimization, and statistics to develop better solutions

Chapter 4

Emergence of IoT

Learning Outcomes

After reading this chapter, the reader will be able to:

- Explain the chronology for the evolution of Internet of Things (IoT)
- Relate new concepts with concepts learned earlier to make a smooth transition to IoT
- List the reasons for a prevailing universal networked paradigm, which is IoT
- Compare and correlate IoT with its precursors such as WSN, M2M, and CPS
- List the various enablers of IoT
- Understand IoT networking components and various networking topologies
- Recognize the unique features of IoT which set it apart from other similar paradigms

4.1 Introduction

The modern-day advent of network-connected devices has given rise to the popular paradigm of the Internet of Things (IoT). Each second, the present-day Internet allows massively heterogeneous traffic through it. This network traffic consists of images, videos, music, speech, text, numbers, binary codes, machine status, banking messages, data from sensors and actuators, healthcare data, data from vehicles, home automation system status and control messages, military communications, and many more. This huge variety of data is generated from a massive number of connected devices, which may be directly connected to the Internet or connected through gateway devices. According to statistics from the Information Handling Services [7], the total number of connected devices globally is estimated to be around 25 billion. This figure is projected

to triple within a short span of 5 years by the year 2025. Figure 4.1 shows the global trend and projection for connected devices worldwide.

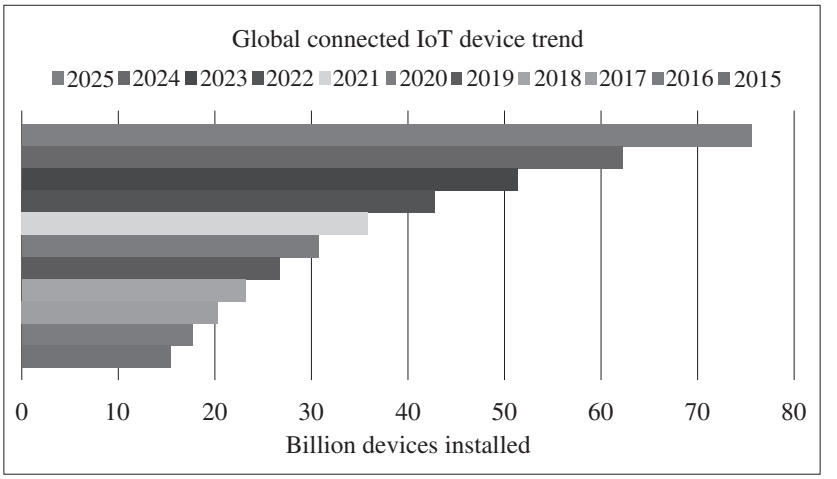


Figure 4.1 10-year global trend and projection of connected devices (statistics sourced from the Information Handling Services [7])

The traffic flowing through the Internet can be attributed to legacy systems as well as modern-day systems. The miniaturization of electronics and the cheap affordability of technology is resulting in a surge of connected devices, which in turn is leading to an explosion of traffic flowing through the Internet.

Points to ponder

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

—Gartner Research [5]

One of the best examples of this explosion is the evolution of smartphones. In the late 1990's, cellular technology was still expensive and which could be afforded only by a select few. Moreover, these particular devices had only the basic features of voice calling, text messaging, and sharing of low-quality multimedia. Within the next 10 years, cellular technology had become common and easily affordable. With time, the features of these devices evolved, and the dependence of various applications and services on these gadgets on packet-based Internet accesses started rapidly increasing. The present-day mobile phones (commonly referred to as smartphones) are more or less Internet-based. The range of applications on these gadgets such as messaging, video calling, e-mails, games, music streaming, video streaming, and others are solely dependent on network provider allocated Internet access or WiFi. Most of

the present-day consumers of smartphone technology tend to carry more than one of these units. In line with this trend, other connected devices have rapidly increased in numbers resulting in the number of devices exceeding the number of humans on Earth by multiple times. Now imagine that as all technologies and domains are moving toward smart management of systems, the number of sensor/actuator-based systems is rapidly increasing. With time, the need for location-independent access to monitored and controlled systems keep on rising. This rise in number leads to a further rise in the number of Internet-connected devices.

The original Internet intended for sending simple messages is now connected with all sorts of "Things". These things can be legacy devices, modern-day computers, sensors, actuators, household appliances, toys, clothes, shoes, vehicles, cameras, and anything which may benefit a product by increasing its scientific value, accuracy, or even its cosmetic value.

Internet of Things

"In the 2000s, we are heading into a new era of ubiquity, where the 'users' of the Internet will be counted in billions and where humans may become the minority as generators and receivers of traffic. Instead, most of the traffic will flow between devices and all kinds of "Things", thereby creating a much wider and more complex Internet of Things."

—ITU Internet Report 2005 [6]

IoT is an anytime, anywhere, and anything (as shown in Figure 4.2) network of Internet-connected physical devices or systems capable of sensing an environment and affecting the sensed environment intelligently. This is generally achieved using low-power and low-form-factor embedded processors on-board the "things" connected to the Internet. In other words, IoT may be considered to be made up of connecting devices, machines, and tools; these things are made up of sensors/actuators and processors, which connect to the Internet through wireless technologies. Another school of thought also considers wired Internet access to be inherent to the IoT paradigm. For the sake of harmony, in this book, we will consider any technology enabling access to the Internet—be it wired or wireless—to be an IoT enabling technology. However, most of the focus on the discussion of various IoT enablers will be restricted to wireless IoT systems due to the much more severe operating constraints and challenges faced by wireless devices as compared to wired systems. Typically, IoT systems can be characterized by the following features [2]:

- Associated architectures, which are also efficient and scalable.
- No ambiguity in naming and addressing.
- Massive number of constrained devices, sleeping nodes, mobile devices, and non-IP devices.
- Intermittent and often unstable connectivity.

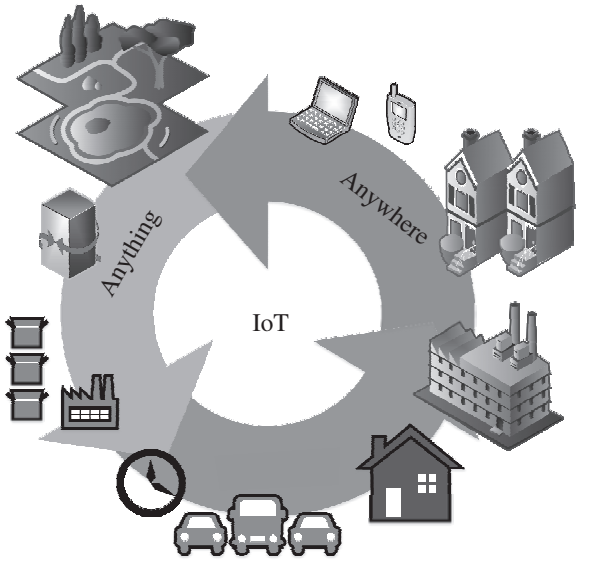


Figure 4.2 The three characteristic features—anytime, anywhere, and anything—highlight the robustness and dynamic nature of IoT

IoT is speculated to have achieved faster and higher technology acceptance as compared to electricity and telephony. These speculations are not ill placed as evident from the various statistics shown in Figures 4.3, 4.4, and 4.5.

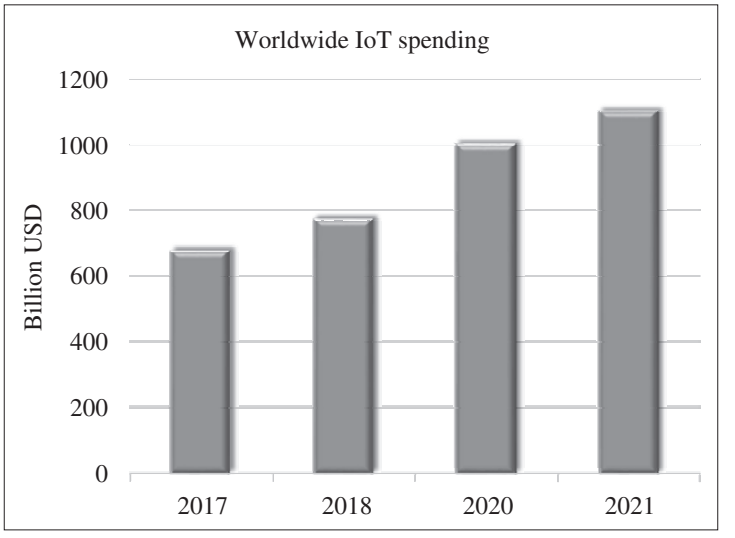


Figure 4.3 The global IoT spending across various organizations and industries and its subsequent projection until the year 2021 (sourced from International Data Corporation [1])

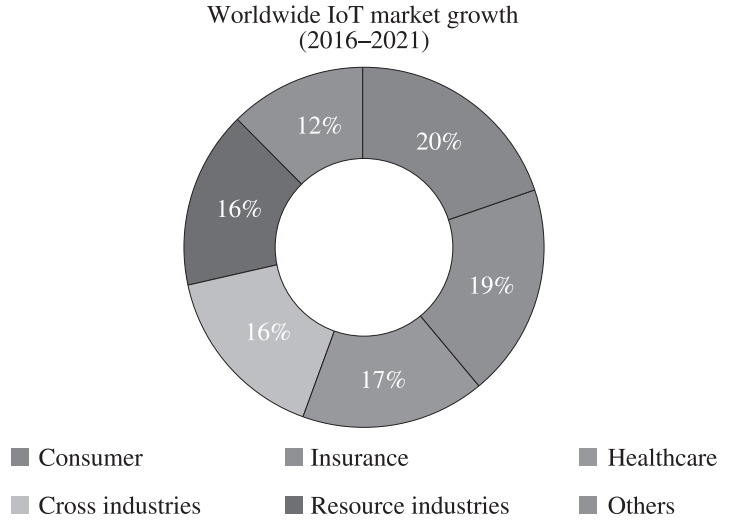


Figure 4.4 The compound annual growth rate (CAGR) of the IoT market (statistics sourced from [1])

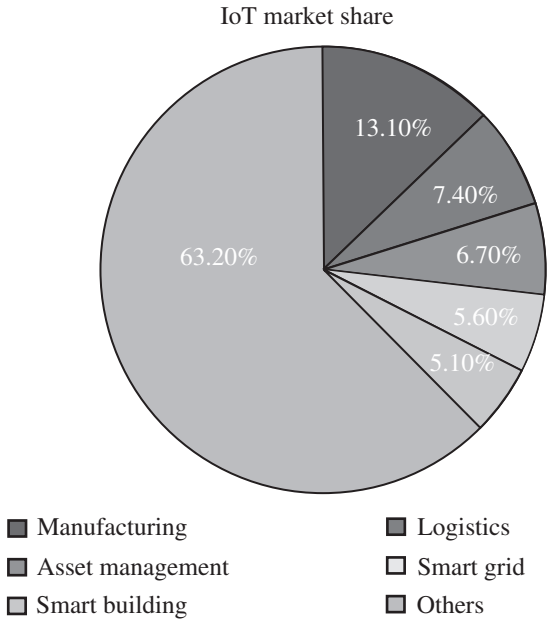


Figure 4.5 The IoT market share across various industries (statistics sourced from International Data Corporation [8])

According to an International Data Corporation (IDC) report, worldwide spending on IoT is reported to have crossed USD 700 billion. The projected spending on IoT-based technologies worldwide is estimated to be about USD 1.1 trillion [1]. Similarly,

the compounded annual growth rate of IoT between the years 2016 and 2021, as depicted in Figure 4.4, shows that the majority of the market share is captured by consumer goods, which is closely followed by insurance and healthcare industries. However, the combined industrial share of IoT growth (both cross and resource) is 32% of the collective market, which is again more than that of the consumer market. In continuation, Figure 4.5 shows the IoT market share of various sectors. The manufacturing, logistics, and asset management sectors were purported to be the largest receivers of IoT-linked investments in 2017 [8].

4.2 Evolution of IoT

The IoT, as we see it today, is a result of a series of technological paradigm shifts over a few decades. The technologies that laid the foundation of connected systems by achieving easy integration to daily lives, popular public acceptance, and massive benefits by using connected solutions can be considered as the founding solutions for the development of IoT. Figure 4.6 shows the sequence of technological advancements for shaping the IoT as it is today. These sequence of technical developments toward the emergence of IoT are described in brief:

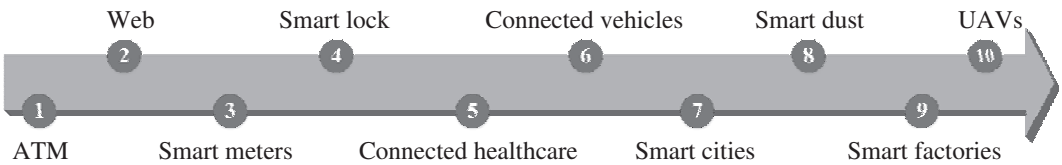


Figure 4.6 The sequence of technological developments leading to the shaping of the modern-day IoT

- ATM: ATMs or automated teller machines are cash distribution machines, which are linked to a user's bank account. ATMs dispense cash upon verification of the identity of a user and their account through a specially coded card. The central concept behind ATMs was the availability of financial transactions even when banks were closed beyond their regular work hours. These ATMs were ubiquitous money dispensers. The first ATM became operational and connected online for the first time in 1974.
- Web: World Wide Web is a global information sharing and communication platform. The Web became operational for the first time in 1991. Since then, it has been massively responsible for the many revolutions in the field of computing and communication.
- Smart Meters: The earliest smart meter was a power meter, which became operational in early 2000. These power meters were capable of communicating remotely with the power grid. They enabled remote monitoring of subscribers' power usage and eased the process of billing and power allocation from grids.

- Digital Locks: Digital locks can be considered as one of the earlier attempts at connected home-automation systems. Present-day digital locks are so robust that smartphones can be used to control them. Operations such as locking and unlocking doors, changing key codes, including new members in the access lists, can be easily performed, and that too remotely using smartphones.
- Connected Healthcare: Here, healthcare devices connect to hospitals, doctors, and relatives to alert them of medical emergencies and take preventive measures. The devices may be simple wearable appliances, monitoring just the heart rate and pulse of the wearer, as well as regular medical devices and monitors in hospitals. The connected nature of these systems makes the availability of medical records and test results much faster, cheaper, and convenient for both patients as well as hospital authorities.
- Connected Vehicles: Connected vehicles may communicate to the Internet or with other vehicles, or even with sensors and actuators contained within it. These vehicles self-diagnose themselves and alert owners about system failures.
- Smart Cities: This is a city-wide implementation of smart sensing, monitoring, and actuation systems. The city-wide infrastructure communicating amongst themselves enables unified and synchronized operations and information dissemination. Some of the facilities which may benefit are parking, transportation, and others.
- Smart Dust: These are microscopic computers. Smaller than a grain of sand each, they can be used in numerous beneficial ways, where regular computers cannot operate. For example, smart dust can be sprayed to measure chemicals in the soil or even to diagnose problems in the human body.
- Smart Factories: These factories can monitor plant processes, assembly lines, distribution lines, and manage factory floors all on their own. The reduction in mishaps due to human errors in judgment or unoptimized processes is drastically reduced.
- UAVs: UAVs or unmanned aerial vehicles have emerged as robust public-domain solutions tasked with applications ranging from agriculture, surveys, surveillance, deliveries, stock maintenance, asset management, and other tasks.

The present-day IoT spans across various domains and applications. The major highlight of this paradigm is its ability to function as a cross-domain technology enabler. Multiple domains can be supported and operated upon simultaneously over IoT-based platforms. Support for legacy technologies and standalone paradigms, along with modern developments, makes IoT quite robust and economical for commercial, industrial, as well as consumer applications. IoT is being used in vivid and diverse areas such as smart parking, smartphone detection, traffic congestion, smart lighting, waste management, smart roads, structural health, urban noise maps, river floods, water flow, silos stock calculation, water leakages, radiation levels, explosive and hazardous gases, perimeter access control, snow

level monitoring, liquid presence, forest fire detection, air pollution, smart grid, tank level, photovoltaic installations, NFC (near-field communications) payments, intelligent shopping applications, landslide and avalanche prevention, early detection of earthquakes, supply chain control, smart product management, and others.

Figure 4.7 shows the various technological interdependencies of IoT with other domains and networking paradigms such as M2M, CPS, the Internet of environment (IoE), the Internet of people (IoP), and Industry 4.0. Each of these networking paradigms is a massive domain on its own, but the omnipresent nature of IoT implies that these domains act as subsets of IoT. The paradigms are briefly discussed here:

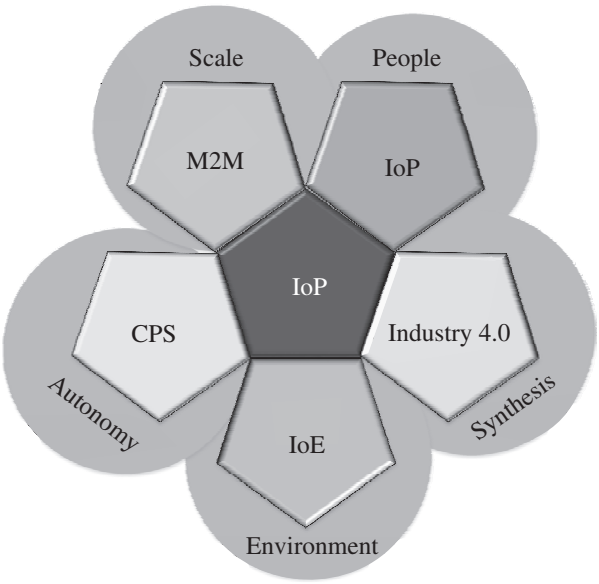


Figure 4.7 The interdependence and reach of IoT over various application domains and networking paradigms

(i) **M2M**: The M2M or the machine-to-machine paradigm signifies a system of connected machines and devices, which can talk amongst themselves without human intervention. The communication between the machines can be for updates on machine status (stocks, health, power status, and others), collaborative task completion, overall knowledge of the systems and the environment, and others.

(ii) **CPS**: The CPS or the cyber physical system paradigm insinuates a closed control loop—from sensing, processing, and finally to actuation—using a feedback mechanism. CPS helps in maintaining the state of an environment through the feedback control loop, which ensures that until the desired state is attained, the system keeps on actuating and sensing. Humans have a simple supervisory role in CPS-based systems; most of the ground-level operations are automated.

(iii) **IoE**: The IoE paradigm is mainly concerned with minimizing and even reversing the ill-effects of the permeation of Internet-based technologies on the environment [3]. The major focus areas of this paradigm include smart and sustainable farming, sustainable and energy-efficient habitats, enhancing the energy efficiency of systems and processes, and others. In brief, we can safely assume that any aspect of IoT that concerns and affects the environment, falls under the purview of IoE.

(iv) **Industry 4.0**: Industry 4.0 is commonly referred to as the fourth industrial revolution pertaining to digitization in the manufacturing industry. The previous revolutions chronologically dealt with mechanization, mass production, and the industrial revolution, respectively. This paradigm strongly puts forward the concept of smart factories, where machines talk to one another without much human involvement based on a framework of CPS and IoT. The digitization and connectedness in Industry 4.0 translate to better resource and workforce management, optimization of production time and resources, and better upkeep and lifetimes of industrial systems.

(v) **IoP**: IoP is a new technological movement on the Internet which aims to decentralize online social interactions, payments, transactions, and other tasks while maintaining confidentiality and privacy of its user's data. A famous site for IoP states that as the introduction of the Bitcoin has severely limited the power of banks and governments, the acceptance of IoP will limit the power of corporations, governments, and their spy agencies [4].

4.2.1 IoT versus M2M

M2M or the machine-to-machine paradigm refers to communications and interactions between various machines and devices. These interactions can be enabled through a cloud computing infrastructure, a server, or simply a local network hub. M2M collects data from machinery and sensors, while also enabling device management and device interaction. Telecommunication services providers introduced the term M2M, and technically emphasized on machine interactions via one or more communication networks (e.g., 3G, 4G, 5G, satellite, public networks). M2M is part of the IoT and is considered as one of its sub-domains, as shown in Figure 4.7. M2M standards occupy a core place in the IoT landscape. However, in terms of operational and functional scope, IoT is vaster than M2M and comprises a broader range of interactions such as the interactions between devices/things, things, and people, things and applications, and people with applications; M2M enables the amalgamation of workflows comprising such interactions within IoT. Internet connectivity is central to the IoT theme but is not necessarily focused on the use of telecom networks.

4.2.2 IoT versus CPS

Cyber physical systems (CPS) encompasses sensing, control, actuation, and feedback as a complete package. In other words, a digital twin is attached to a CPS-based system. As mentioned earlier, a digital twin is a virtual system–model relation, in which the system signifies a physical system or equipment or a piece of machinery, while the model represents the mathematical model or representation of the physical system's behavior or operation. Many a time, a digital twin is used parallel to a physical system, especially in CPS as it allows for the comparison of the physical system's output, performance, and health. Based on feedback from the digital twin, a physical system can be easily given corrective directions/commands to obtain desirable outputs. In contrast, the IoT paradigm does not compulsorily need feedback or a digital twin system. IoT is more focused on networking than controls. Some of the constituent sub-systems in an IoT environment (such as those formed by CPS-based instruments and networks) may include feedback and controls too. In this light, CPS may be considered as one of the sub-domains of IoT, as shown in Figure 4.7.

4.2.3 IoT versus WoT

From a developer's perspective, the Web of Things (WoT) paradigm enables access and control over IoT resources and applications. These resources and applications are generally built using technologies such as HTML 5.0, JavaScript, Ajax, PHP, and others. REST (representational state transfer) is one of the key enablers of WoT. The use of RESTful principles and RESTful APIs (application program interface) enables both developers and deployers to benefit from the recognition, acceptance, and maturity of existing web technologies without having to redesign and redeploy solutions from scratch. Still, designing and building the WoT paradigm has various adaptability and security challenges, especially when trying to build a globally uniform WoT. As IoT is focused on creating networks comprising objects, things, people, systems, and applications, which often do not consider the unification aspect and the limitations of the Internet, the need for WoT, which aims to integrate the various focus areas of IoT into the existing Web is really invaluable. Technically, WoT can be thought of as an application layer-based hat added over the network layer. However, the scope of IoT applications is much broader; IoT also which includes non-IP-based systems that are not accessible through the web.

4.3 Enabling IoT and the Complex Interdependence of Technologies

IoT is a paradigm built upon complex interdependencies of technologies (both legacy and modern), which occur at various planes of this paradigm. Regarding Figure 4.8, we can divide the IoT paradigm into four planes: services, local connectivity, global connectivity, and processing. If we consider a bottom-up view, the services offered fall

under the control and purview of service providers. The service plane is composed of two parts: 1) things or devices and 2) low-power connectivity.

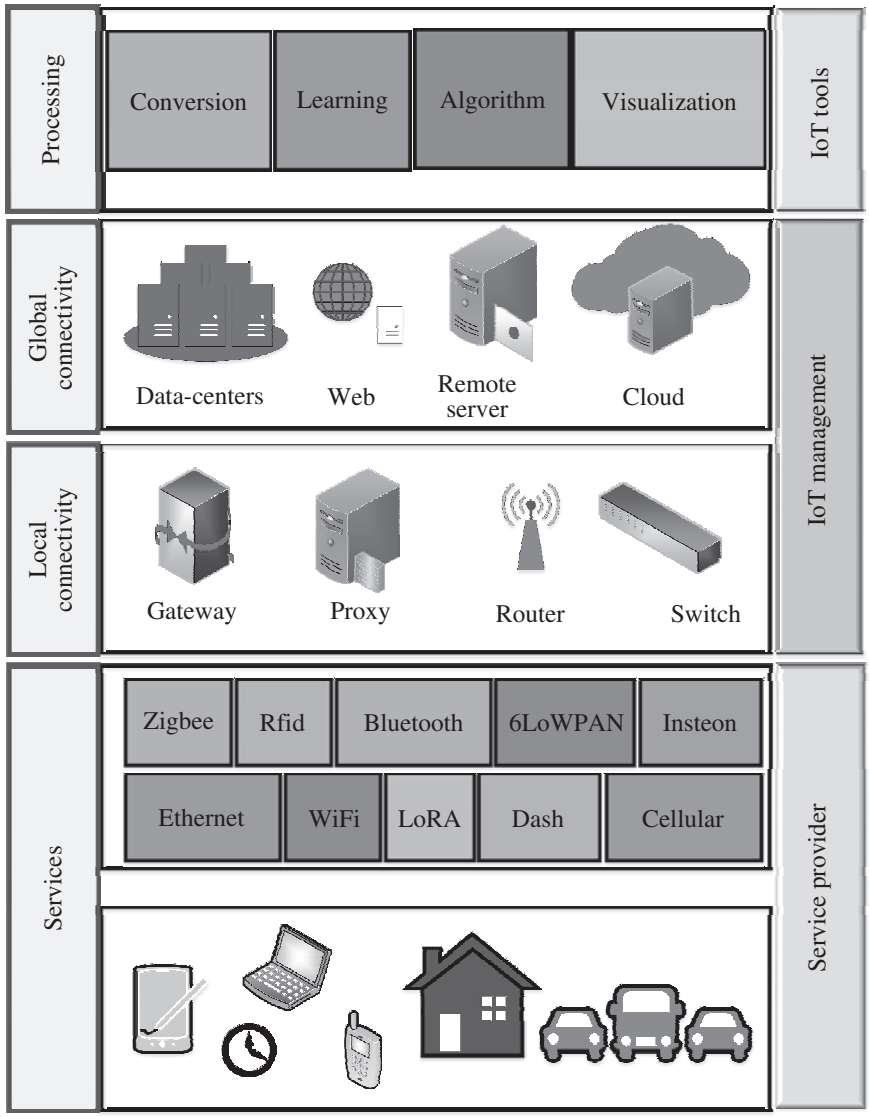


Figure 4.8 The IoT planes, various enablers of IoT, and the complex interdependencies among them

Typically, the services offered in this layer are a combination of things and low-power connectivity. For example, any IoT application requires the basic setup of sensing, followed by rudimentary processing (often), and a low-power, low-range network, which is mainly built upon the IEEE 802.15.4 protocol. The things may be wearables, computers, smartphones, household appliances, smart glasses, factory

machinery, vending machines, vehicles, UAVs, robots, and other such contraptions (which may even be just a sensor). The immediate low-power connectivity, which is responsible for connecting the things in local implementation, may be legacy protocols such as WiFi, Ethernet, or cellular. In contrast, modern-day technologies are mainly wireless and often programmable such as Zigbee, RFID, Bluetooth, 6LoWPAN, LoRA, DASH, Insteon, and others. The range of these connectivity technologies is severely restricted; they are responsible for the connectivity between the things of the IoT and the nearest hub or gateway to access the Internet.

The local connectivity is responsible for distributing Internet access to multiple local IoT deployments. This distribution may be on the basis of the physical placement of the things, on the basis of the application domains, or even on the basis of providers of services. Services such as address management, device management, security, sleep scheduling, and others fall within the scope of this plane. For example, in a smart home environment, the first floor and the ground floor may have local IoT implementations, which have various things connected to the network via low-power, low-range connectivity technologies. The traffic from these two floors merges into a single router or a gateway. The total traffic intended for the Internet from a smart home leaves through a single gateway or router, which may be assigned a single global IP address (for the whole house). This helps in the significant conservation of already limited global IP addresses. The local connectivity plane falls under the purview of IoT management as it directly deals with strategies to use/reuse addresses based on things and applications. The modern-day "edge computing" paradigm is deployed in conjunction with these first two planes: services and local connectivity.

In continuation, the penultimate plane of global connectivity plays a significant role in enabling IoT in the real sense by allowing for worldwide implementations and connectivity between things, users, controllers, and applications. This plane also falls under the purview of IoT management as it decides how and when to store data, when to process it, when to forward it, and in which form to forward it. The Web, data-centers, remote servers, Cloud, and others make up this plane. The paradigm of "fog computing" lies between the planes of local connectivity and global connectivity. It often serves to manage the load of global connectivity infrastructure by offloading the computation nearer to the source of the data itself, which reduces the traffic load on the global Internet.

The final plane of processing can be considered as a top-up of the basic IoT networking framework. The continuous rise in the usefulness and penetration of IoT in various application areas such as industries, transportation, healthcare, and others is the result of this plane. The members in this plane may be termed as IoT tools, simply because they wring-out useful and human-readable information from all the raw data that flows from various IoT devices and deployments. The various sub-domains of this plane include intelligence, conversion (data and format conversion, and data cleaning), learning (making sense of temporal and spatial data patterns), cognition (recognizing patterns and mapping it to already known patterns), algorithms (various control and monitoring algorithms), visualization (rendering

numbers and strings in the form of collective trends, graphs, charts, and projections), and analysis (estimating the usefulness of the generated information, making sense of the information with respect to the application and place of data generation, and estimating future trends based on past and present patterns of information obtained). Various computing paradigms such as "big data", "machine Learning", and others, fall within the scope of this domain.

4.4 IoT Networking Components

An IoT implementation is composed of several components, which may vary with their application domains. Various established works such as that by Savolainen et al. [2] generally outline five broad categories of IoT networking components. However, we outline the broad components that come into play during the establishment of any IoT network, into six types: 1) IoT node, 2) IoT router, 3) IoT LAN, 4) IoT WAN, 5) IoT gateway, and 6) IoT proxy. A typical IoT implementation from a networking perspective is shown in Figure 4.9. The individual components are briefly described here:

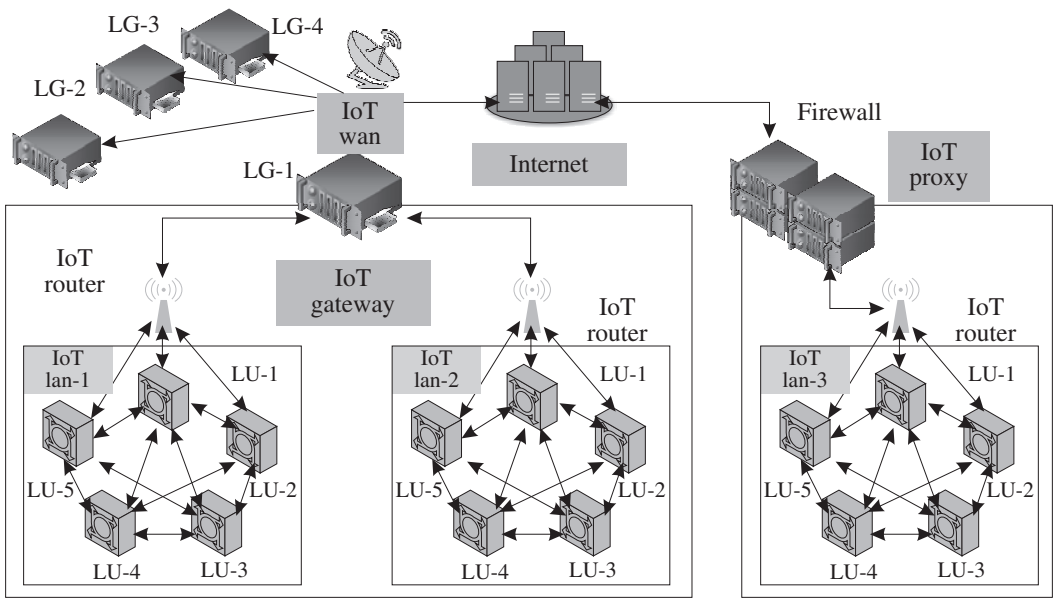


Figure 4.9 A typical IoT network ecosystem highlighting the various networking components—from IoT nodes to the Internet

(i) **IoT Node**: These are the networking devices within an IoT LAN. Each of these devices is typically made up of a sensor, a processor, and a radio, which communicates with the network infrastructure (either within the LAN or outside it). The nodes may be connected to other nodes inside a LAN directly or by

means of a common gateway for that LAN. Connections outside the LAN are through gateways and proxies.

(ii) **IoT Router**: An I oT router is a piece of networking equipment that is primarily tasked with the routing of packets between various entities in the IoT network; it keeps the traffic flowing correctly within the network. A router can be repurposed as a gateway by enhancing its functionalities.

(iii) **IoT LAN**: The local area network (LAN) enables local connectivity within the purview of a single gateway. Typically, they consist of short-range connectivity technologies. IoT LANs may or may not be connected to the Internet. Generally, they are localized within a building or an organization.

(iv) **IoT WAN**: The wide area network (WAN) connects various network segments such as LANs. They are typically organizationally and geographically wide, with their operational range lying between a few kilometers to hundreds of kilometers. IoT WANs connect to the Internet and enable Internet access to the segments they are connecting.

(v) **IoT Gateway**: An IoT gateway is simply a router connecting the IoT LAN to a WAN or the Internet. Gateways can implement several LANs and WANs. Their primary task is to forward packets between LANs and WANs, and the IP layer using only layer 3.

(vi) **IoT Proxy**: Proxies actively lie on the application layer and performs application layer functions between IoT nodes and other entities. Typically, application layer proxies are a means of providing security to the network entities under it ; it helps to extend the addressing range of its network.

In Figure 4.9, various IoT nodes within an IoT LAN are configured to to one another as well as talk to the IoT router whenever they are in the range of it. The devices have locally unique (LU-*x*) device identifiers. These identifiers are unique only within a LAN. There is a high chance that these identifiers may be repeated in a new LAN. Each IoT LAN has its own unique identifier, which is denoted by IoT LAN-*x* in Figure 4.9. A router acts as a connecting link between various LANs by forwarding messages from the LANs to the IoT gateway or the IoT proxy. As the proxy is an application layer device, it is additionally possible to include features such as firewalls, packet filters, and other security measures besides the regular routing operations. Various gateways connect to an IoT WAN, which links these devices to the Internet. There may be cases where the gateway or the proxy may directly connect to the Internet. This network may be wired or wireless; however, IoT deployments heavily rely on wireless solutions. This is mainly attributed to the large number of devices that are integrated into the network; wireless technology is the only feasible and neat-enough solution to avoid the hassles of laying wires and dealing with the restricted mobility rising out of wired connections.

4.5 Addressing Strategies in IoT

Table 4.1 lists the differences in features of IPv4 and IPv6. The most interesting point to note is that as compared to IPv4, which relies more on reliable delivery of packets between source and destination, an IPv6 packet is more address-oriented. Due to the increasing rate of devices being connected to the Internet, the early developers of IPv6 felt the need for accommodating addresses as more crucial than the need for reliable transmission of packets (which was the main feature of IPv4-based routing of packets).

Table 4.1 Feature-wise difference between IPv4 and IPv6 capabilities

Feature	IPv4	IPv6
Developed	IETF 1974	IETF 1998
Address length (bits)	32	128
No. of addresses	2^{32}	2^{128}
Notation	Dotted decimal	Hexadecimal
Dynamic allocation of addresses	DHCP	DHCPv6, SLAAC
IPSec	Optional	Compulsary
Header size	Variable	Fixed
Header checksum	Yes	No
Header options	Yes	No
Broadcast addresses	Yes	No
Multicast addresses	No	Yes
Feature	Focus on reliable transmission	Focus on addressing

In the context of IoT, we will consider and center our discussions on addressing schemes primarily focused on IPv6. The IPv4 and IPv6 header packet formats are shown in Chapter 1 of this book. In continuation, Figure 4.10 shows the address format of IPv6, which is 128 bits long.

The first three blocks are designated as the global prefix, which is globally unique. The next block is designated as the subnet prefix, which identifies the subnet of an interface/gateway through which LANs may be connected to the Internet. Finally, the last four blocks (64 bits) of hexadecimal addresses are collectively known as the interface identifier (IID). IIDs may be generated based on MAC (media access control) identifiers of devices/nodes or using pseudo-random number generator algorithms [2]. The IPv6 addresses can be divided into seven separate address types, which is generally based on how these addresses are used or where they are deployed.

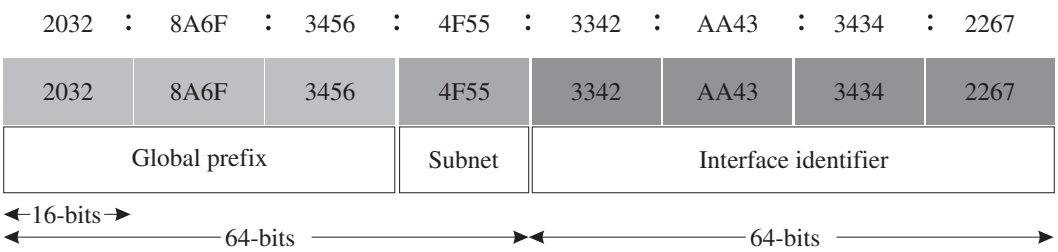


Figure 4.10 The IPv6 address format

(i) **Global Unicast (GUA)**: These addresses are assigned to single IoT entities/interfaces; they enable the entities to transmit traffic to and from the Internet. In regular IoT deployments, these addresses are assigned to gateways, proxies, or WANs.

(ii) **Multicast**: These addresses enable transmission of messages from a single networked entity to multiple destination entities simultaneously.

(iii) **Link Local (LL)**: The operational domain of these addresses are valid only within a network segment such as LAN. These addresses may be repeated in other network segments/LANs, but are unique within that single network segment.

(iv) **Unique Local (ULA)**: Similar to LL addresses, ULA cannot be routed over the Internet. These addresses may be repeated in other network segments/LANs, but are unique within that single network segment.

(v) **Loopback**: It is also known as the localhost address. Typically, these addresses are used by developers and network testers for diagnostics and system checks.

(vi) **Unspecified**: Here, all the bits in the IPv6 address are set to zero and the destination address is not specified.

(vii) **Solicited-node Multicast**: It is a multicast address based on the IPv6 address of an IoT node or entity.

Points to ponder

Multihoming in IoT networks: It is a network configuration in which a node/network connects to multiple networks simultaneously for improved reliability. Network proxies are used to manage multiple IP addresses and map them to LL addresses of IoT nodes in small deployments, where the allotment of address prefixes is not possible. Other approaches for multihoming include the use of gateways for assigning LL addresses to IoT nodes under the gateway's operational purview.

4.5.1 Address management classes

As discussed previously, the IoT deployment and network topology are largely dependent on where it is deployed. Unlike traditional IPv4 networked devices, the newer IoT devices largely depend on IPv6 for address allocation and management of addresses, which again is dictated by the application and the place of deployment of the IoT solution. Keeping these requirements in consideration, the addressing strategies in IoT may be broadly differentiated into seven classes, as shown in Figure 4.11. These classes are as follows:

(i) **Class 1**: The IoT nodes are not connected to any other interface or the Internet except with themselves. This class can be considered as an isolated class, where the communication between IoT nodes is restricted within a LAN only. The IoT nodes in this class are identified only by their link local (LL) addresses, as shown in Figure 4.11(a). These LL addresses may be repeated for other devices outside the purview of this network class. The communication among the nodes may be direct or through other nodes (as in a mesh configuration).

(ii) **Class 2**: The class 1 configuration is mainly utilized for enabling communication between two or more IoT LANs or WANs. The IoT nodes within the LANs cannot directly communicate to nodes in the other LANs using their LL addresses, but through their LAN gateways (which have a unique address assigned to them). Generally, ULA is used for addressing; however, in certain scenarios, GUA may also be used. Figure 4.11(b) shows a class 2 IoT network topology. L1–L5 are the LL addresses of the locally unique IoT nodes within the LAN; whereas U1 and U2 are the unique addresses of the two gateways extending communication to their LANs with the WAN. The WAN may or may not connect to the Internet.

(iii) **Class 3**: Figure 4.11(c) shows a class 3 IoT network configuration, where the IoT LAN is connected to an IoT proxy. The proxy performs a host of functions ranging from address allocation, address management to providing security to the network underneath it. In this class, the IoT proxy only uses ULA (denoted as L*x*-U*x* in the figure).

(iv) **Class 4**: In this class, the IoT proxy acts as a gateway between the LAN and the Internet, and provides GUA to the IoT nodes within the LAN. A globally unique prefix is allotted to this gateway, which it uses with the individual device identifiers to extend global Internet connectivity to the IoT nodes themselves. This configuration is shown in Figure 4.11(d). An important point to note in this class is that the gateway also enables local communication between the nodes without the need for the packets to be routed through the Internet. Additionally, the IoT nodes within the gateway can talk to one another directly without always involving the gateway. A proxy beyond the gateway enables global communication through the Internet.

(v) **Class 5**: This class is functionally similar to class 4. However, the main difference with class 4 is that this class follows a star topology with the gateway as the center of the star. All the communication from the IoT nodes under the gateway has to go through the gateway, as shown in Figure 4.11(e). A proxy beyond the gateway enables global communication through the Internet. The IoT nodes within a gateway's operational purview have the same GUA.

(vi) **Class 6**: The configuration of this class is again similar to class 5. However, the IoT nodes are all assigned unique global addresses (GUA), which enables a point-to-point communication network with an Internet gateway. A class 6 IoT network configuration is shown in Figure 4.11(f). Typically, this class is very selectively used for special purposes.

(vii) **Class 7**: The class 7 configuration is shown in Figure 4.11(g). Multiple gateways may be present; the configuration is such that the nodes should be reachable through any of the gateways. Typically, organizational IoT deployments follow this class of configuration. The concept of multihoming is important and inherent to this class.

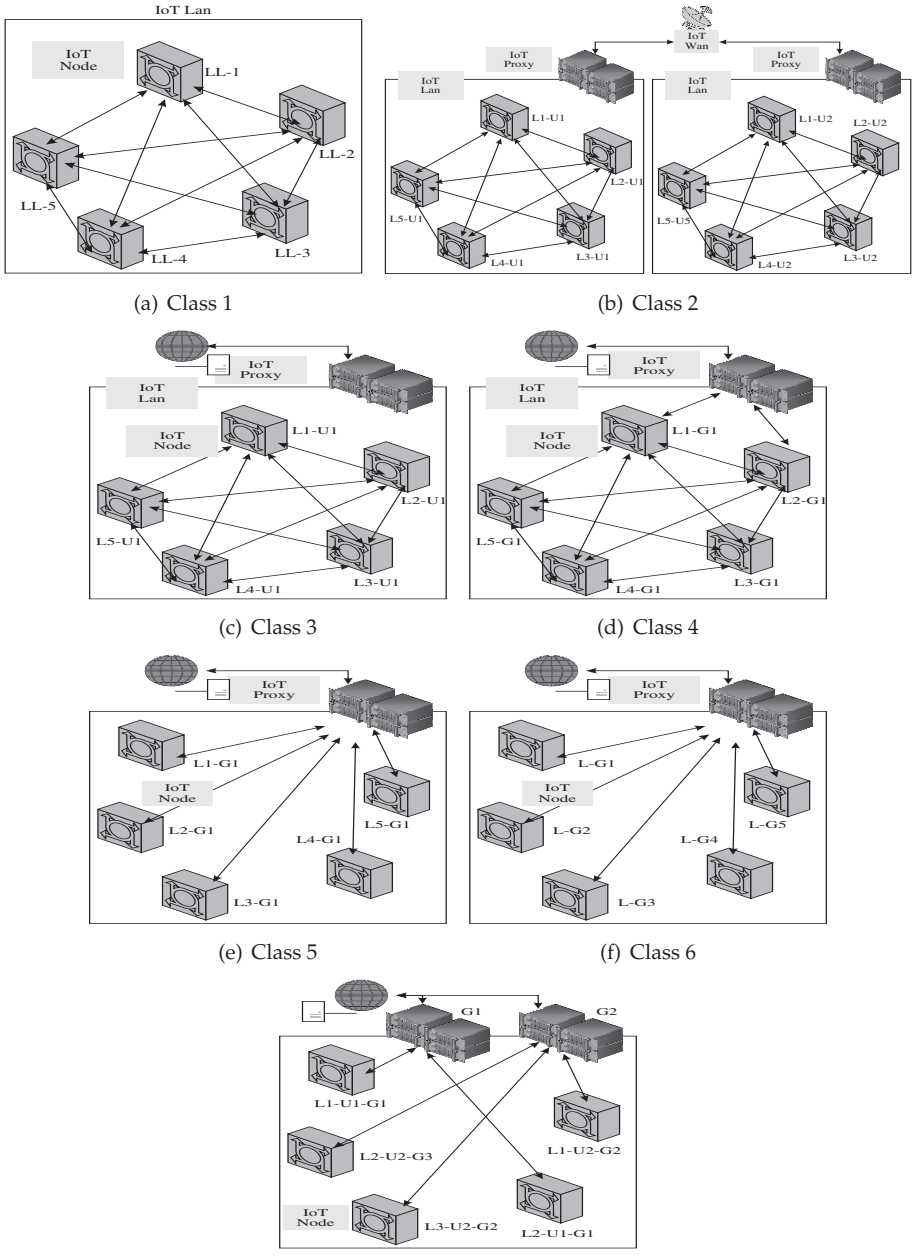
Points to ponder

Tunneling: It is a networking protocol in which data from private networks can be seamlessly streamed over a public network in the form of encapsulated packets. This is mainly used for ensuring connectivity and security of data generated from various technologies and protocols that may not be supported over the public communication channel. Some of the best examples of tunneling are virtual private networks (VPNs), secure shell (SSH), and others.

4.5.2 Addressing during node mobility

Traditional networks, mainly computer networks, and even paradigms such as M2M and CPS seldom take into account the need for addressing strategies when the IoT nodes are mobile. However, in a realistic scenario, especially in modern-day IoT systems (which are low-power and have low form-factor), the need for addressing of mobile nodes is extremely crucial to avoid address clashes of addresses accommodating a large number of IoT nodes. One of the following three strategies may be to for ensure portability of addresses in the event of node mobility in IoT deployments [2] as shown in Figure 4.12:

(i) **Global Prefix Changes**: Figure 4.12(a) abstracts the addressing strategy using global prefix changes. A node from the left LAN moves to the LAN on the right. The node undergoing movement is highlighted in the figure. The nodes in the first LAN have the prefix **A**, which changes to **B** under the domain of the new gateway overseeing the operation of nodes in the new LAN. However, it may



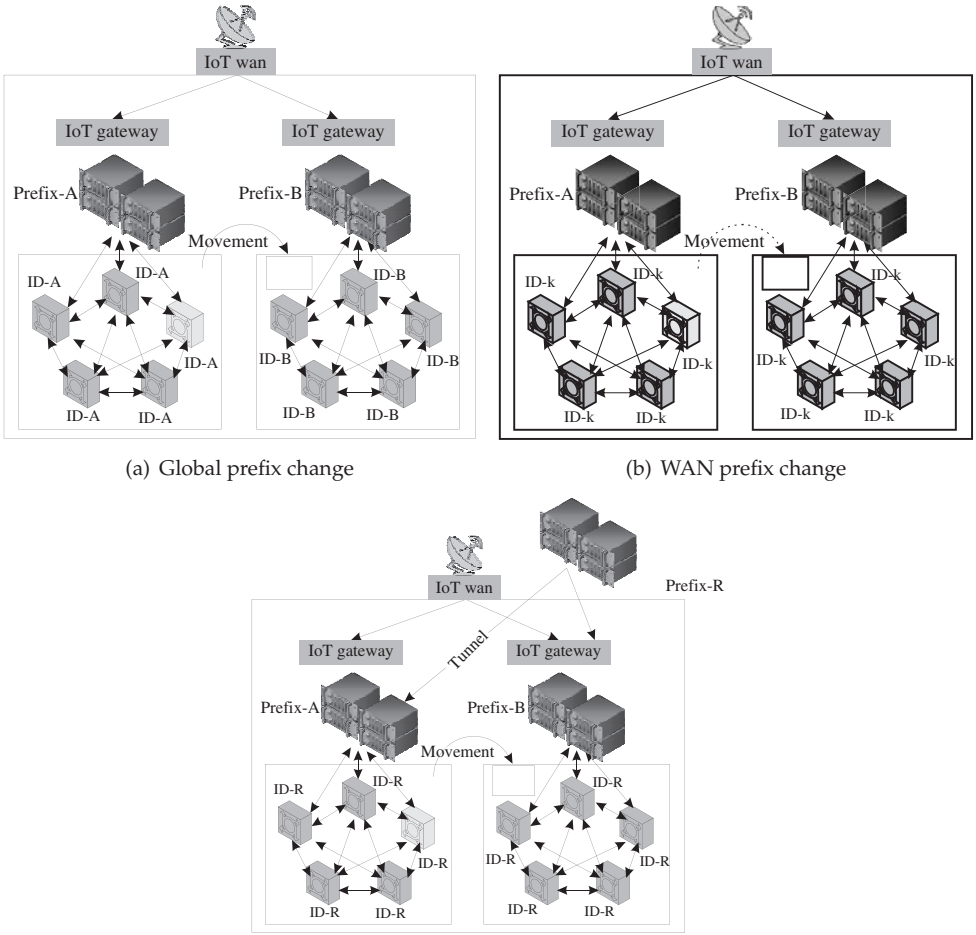
(a) Class 1 (b) Class 2

(c) Class 3 (d) Class 4

(e) Class 5 (f) Class 6

(g) Class 7

Figure 4.11 Various IoT topology configurations. LL/L denotes the link local addresses, LU denotes the locally unique link addresses (ULA), and LG denotes the globally unique link addresses (GUA)



(a) Global prefix change

(b) WAN prefix change

(c) Remote anchoring

Figure 4.12 Various scenarios during mobility of IoT nodes and their addressing strategies. ID-***prefix*** denotes the point to which the IoT node is attached to for address allocation

happen that due to movement, the device identifier may face clashes. Recall the structure of the IPv6 address (Figure 4.10). The device identifier, if allotted randomly, might face an address clash upon the node's arrival into the new LAN as there may already be a similar node identifier present in it. Typically, addresses are assigned using DHCPv6/ SLAAC; however, in this scenario, it is always prudent to have static node IP addresses to avoid a clash of addresses. This strategy is, in most cases, beneficial as the IoT nodes may be resource-constrained and have low-processing resources due to which it may not be able to handle protocols such as DHCPv6 or SLAAC.

(ii) **Prefix Changes within WANs**: Figure 4.12(b) abstracts the addressing strategy for prefix changes within WANs. In case the WAN changes its global prefix, the network entities underneath it must be resilient to change and function normally. The address allocation is hence delegated to entities such as gateways and proxies, which make use of ULAs to manage the network within the WAN.

(iii) **Remote Anchoring**: Figure 4.12(c) abstracts the addressing strategy using a remote anchoring point. This is applicable in certain cases which require that the IoT node's global addresses are maintained and not affected by its mobility or even the change in network prefixes. Although a bit expensive to implement, this strategy of having a remote anchoring point from which the IoT nodes obtain their global addresses through tunneling ensures that the nodes are resilient to changes and are quite stable. Even if the node's original network's (LAN) prefix changes from **A** to **B**, the node's global address remains immune to this change.

Check yourself

DHCP, DHCPv6, SLAAC, MIPv6, PMIPv6, DS-MIPv6

Summary

This chapter covered an overview of the IoT paradigm. Starting from the variations in global market trends and the rapidly expanding trend toward connected systems and devices, to the actual market capture of various IoT solutions in diverse domains, this chapter highlights the importance of IoT in the modern world. Subsequently, the emergence of IoT from its precursors, the IoT ecosystem, and thematic differences between IoT and similar technologies (M2M, CPS, WoT) are outlined. The complex technological interdependence between technologies and paradigms towards enabling IoT is described in the form of planes of functionalities. Keeping in tune with the networking theme of this book, the various networking entities in an IoT ecosystem are described, which is naturally followed by various IoT deployment topology classes and addressing schemes. This chapter concludes with a discussion on IoT node address management during node mobility.

Exercises

(i) What is IoT?

(ii) What is smart dust?

(iii) Differentiate between IoT and M2M.

(iv) Differentiate between IoT and WoT.

(v) What is Web of Things (WoT)?

(vi) What are the various IoT connectivity terminologies?

(vii) Differentiate between an IoT proxy and an IoT gateway.

(viii) What is gateway prefix allotment?

(ix) How are locally unique (LU) addresses different from globally unique (GU) addresses?

(x) How is mobility handled in IoT networks?

(xi) What is the function of a remote anchor point in IoT networks?

(xii) What is tunneling?

(xiii) What is multihoming in IoT networks?

References

[1] International Data Corporation. 2017. "IDC Forecasts Worldwide Spending on the Internet of Things to Reach USD 772 Billion in 2018." https://www.idc.com/getdoc.jsp?containerId=prUS43295217.

[2] Savolainen, T., J. Soininen, and B. Silverajan. 2013. "IPv6 Addressing Strategies for IoT." *IEEE Sensors Journal* 13(10): 3511–3519.

[3] Malek, M. 2017. "The Development of the Internet of Environment." https://www.future-processing.com/blog/the-development-of-the-internet-of-environment/.

[4] Brans, Cristiaan. 2018. "Internet Of People: Building A New Internet." https://iop.global/.

[5] Gartner Research. 2016. "Internet of Things Information Handling Services." https://www.gartner.com/it-glossary/internet-of-things/.

[6] International Telecommunication Union (ITU). 2005. *ITU Internet Reports 2005: The Internet of Things: Executive Summary*. https://www.itu.int/net/wsis/tunis/newsroom/stats/The-Internet-of-Things-2005.pdf.

[7] IHS. "IoT Platforms: Enabling the Internet of Things." https://ihsmarkit.com/industry/telecommunications.html.

[8] International Data Corporation. 2016. "IDC Says Worldwide Spending on the Internet of Things Forecast to Reach Nearly USD 1.4 Trillion in 2021." https://www.idc.com/getdoc.jsp?containerId=prUS42799917.

Chapter 5

IoT Sensing and Actuation

Learning Outcomes

After reading this chapter, the reader will be able to:

- List the salient features of transducers
- Differentiate between sensors and actuators
- Characterize sensors and distinguish between types of sensors
- List the multi-faceted considerations associated with sensing
- Characterize actuators and distinguish between types of actuators
- List the multi-faceted considerations associated with actuation

5.1 Introduction

A major chunk of IoT applications involves sensing in one form or the other. Almost all the applications in IoT—be it a consumer IoT, an industrial IoT, or just plain hobby-based deployments of IoT solutions—sensing forms the first step. Incidentally, actuation forms the final step in the whole operation of IoT application deployment in a majority of scenarios. The basic science of sensing and actuation is based on the process of transduction. Transduction is the process of energy conversion from one form to another. A transducer is a physical means of enabling transduction. Transducers take energy in any form (for which it is designed)—electrical, mechanical, chemical, light, sound, and others—and convert it into another, which may be electrical, mechanical, chemical, light, sound, and others. Sensors and actuators are deemed as transducers. For example, in a public announcement (PA) system, a microphone (input device) converts sound waves into electrical signals, which is amplified by an amplifier system (a process). Finally, a loudspeaker (output device) outputs this into audible sounds by converting the amplified electrical signals back

into sound waves. Table 5.1 outlines the basic terminological differences between transducers, sensors, and actuators.

Table 5.1 Basic outline of the differences between transducers, sensors, and actuators

Parameters	Transducers	Sensors	Actuators
Definition	Converts energy from one form to another.	Converts various forms of energy into electrical signals.	Converts electrical signals into various forms of energy, typically mechanical energy.
Domain	Can be used to represent a sensor as well as an actuator.	It is an input transducer.	It is an output transducer.
Function	Can work as a sensor or an actuator but not simultaneously.	Used for quantifying environmental stimuli into signals.	Used for converting signals into proportional mechanical or electrical outputs.
Examples	Any sensor or actuator	Humidity sensors, Temperature sensors, Anemometers (measures flow velocity), Manometers (measures fluid pressure), Accelerometers (measures the acceleration of a body), Gas sensors (measures concentration of specific gas or gases), and others	Motors (convert electrical energy to rotary motion), Force heads (which impose a force), Pumps (which convert rotary motion of shafts into either a pressure or a fluid velocity).

5.2 Sensors

Sensors are devices that can measure, or quantify, or respond to the ambient changes in their environment or within the intended zone of their deployment. They generate responses to external stimuli or physical phenomenon through characterization of the input functions (which are these external stimuli) and their conversion into typically electrical signals. For example, heat is converted to electrical signals in a temperature sensor, or atmospheric pressure is converted to electrical signals in a barometer. A

sensor is only sensitive to the measured property (e.g., a temperature sensor only senses the ambient temperature of a room). It is insensitive to any other property besides what it is designed to detect (e.g., a temperature sensor does not bother about light or pressure while sensing the temperature). Finally, a sensor does not influence the measured property (e.g., measuring the temperature does not reduce or increase the temperature). Figure 5.1 shows the simple outline of a sensing task. Here, a temperature sensor keeps on checking an environment for changes. In the event of a fire, the temperature of the environment goes up. The temperature sensor notices this change in the temperature of the room and promptly communicates this information to a remote monitor via the processor.

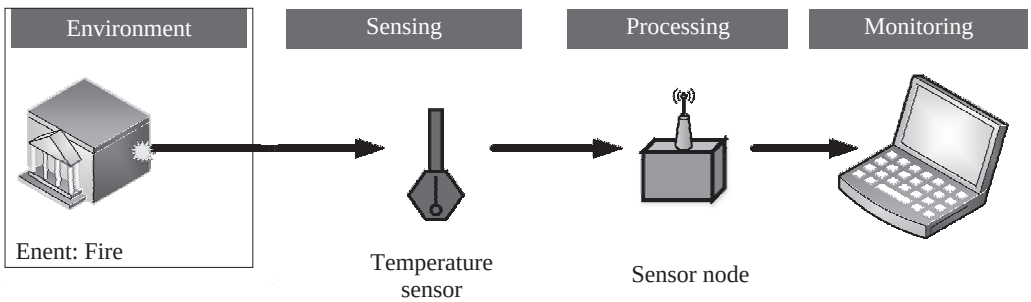


Figure 5.1 The outline of a simple sensing operation

The various sensors can be classified based on: 1) power requirements, 2) sensor output, and 3) property to be measured.

- **Power Requirements**: The way sensors operate decides the power requirements that must be provided for an IoT implementation. Some sensors need to be provided with separate power sources for them to function, whereas some sensors do not require any power sources. Depending on the requirements of power, sensors can be of two types.

  (i) Active: Active sensors do not require an external circuitry or mechanism to provide it with power. It directly responds to the external stimuli from its ambient environment and converts it into an output signal. For example, a photodiode converts light into electrical impulses.

  (ii) Passive: Passive sensors require an external mechanism to power them up. The sensed properties are modulated with the sensor's inherent characteristics to generate patterns in the output of the sensor. For example, a thermistor's resistance can be detected by applying voltage difference across it or passing a current through it.

- **Output**: The output of a sensor helps in deciding the additional components to be integrated with an IoT node or system. Typically, almost all modern-day processors are digital; digital sensors can be directly integrated to the processors.

However, the integration of analog sensors to these digital processors or IoT nodes requires additional interfacing mechanisms such as analog to digital converters (ADC), voltage level converters, and others. Sensors are broadly divided into two types, depending on the type of output generated from these sensors, as follows.

(i) Analog: Analog sensors generate an output signal or voltage, which is proportional (linearly or non-linearly) to the quantity being measured and is continuous in time and amplitude. Physical quantities such as temperature, speed, pressure, displacement, strain, and others are all continuous and categorized as analog quantities. For example, a thermometer or a thermocouple can be used for measuring the temperature of a liquid (e.g., in household water heaters). These sensors continuously respond to changes in the temperature of the liquid.

(ii) Digital: These sensors generate the output of discrete time digital representation (time, or amplitude, or both) of a quantity being measured, in the form of output signals or voltages. Typically, binary output signals in the form of a logic **1** or a logic **0** for **ON** or **OFF**, respectively are associated with digital sensors. The generated discrete (non-continuous) values may be output as a single "bit" (serial transmission), eight of which combine to produce a single "byte" output (parallel transmission) in digital sensors.

- **Measured Property**: The property of the environment being measured by the sensors can be crucial in deciding the number of sensors in an IoT implementation. Some properties to be measured do not show high spatial variations and can be quantified only based on temporal variations in the measured property, such as ambient temperature, atmospheric pressure, and others. Whereas some properties to be measured show high spatial as well as temporal variations such as sound, image, and others. Depending on the properties to be measured, sensors can be of two types.

(i) Scalar: Scalar sensors produce an output proportional to the magnitude of the quantity being measured. The output is in the form of a signal or voltage. Scalar physical quantities are those where only the magnitude of the signal is sufficient for describing or characterizing the phenomenon and information generation. Examples of such measurable physical quantities include color, pressure, temperature, strain, and others. A thermometer or thermocouple is an example of a scalar sensor that has the ability to detect changes in ambient or object temperatures (depending on the sensor's configuration). Factors such as changes in sensor orientation or direction do not affect these sensors (typically).

(ii) Vector: Vector sensors are affected by the magnitude as well as the direction and/or orientation of the property they are measuring. Physical quantities such as velocity and images that require additional information besides

their magnitude for completely categorizing a physical phenomenon are categorized as vector quantities. Measuring such quantities are undertaken using vector sensors. For example, an electronic gyroscope, which is commonly found in all modern aircraft, is used for detecting the changes in orientation of the gyroscope with respect to the Earth's orientation along all three axes.

Points to ponder

A sensor node is made up of a combination of sensor/sensors, a processor unit, a radio unit, and a power unit. The nodes are capable of sensing the environment they are set to measure and communicate the information to other sensor nodes or a remote server. Typically, a sensor node should have low-power requirements and be wireless. This enables them to be deployed in a vast range of scenarios and environments without the constant need for changing their power sources or managing wires. The wireless nature of sensor nodes would also allow them to be freely relocatable and deployed in large numbers without bothering about managing wires. The functional outline of a typical IoT sensor node is shown in Figure 5.2.

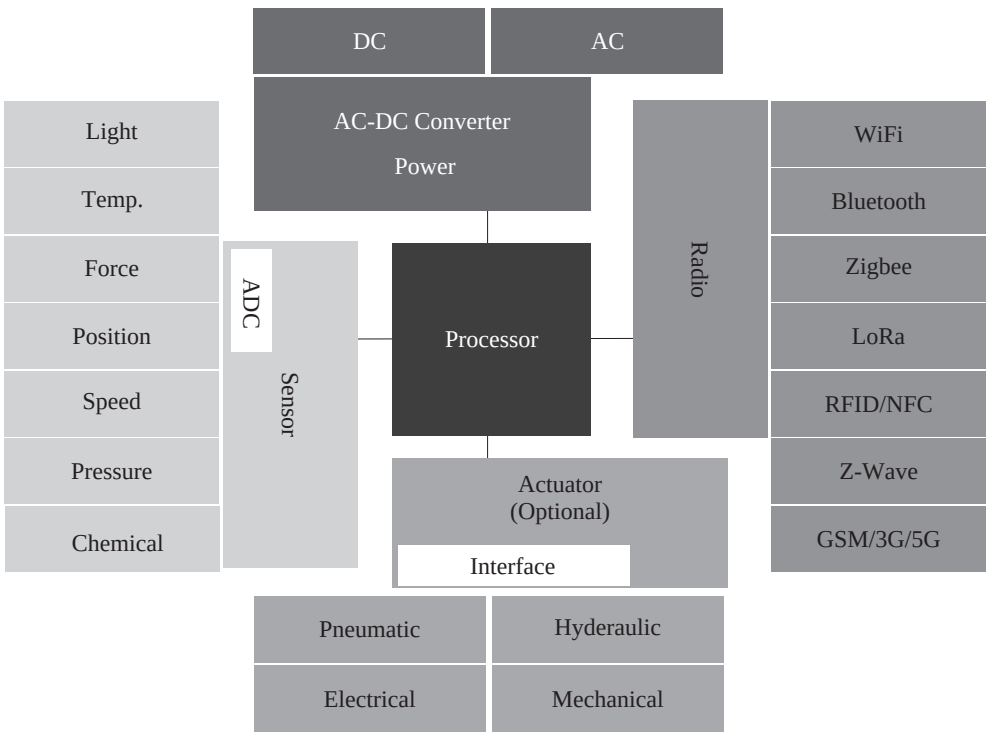
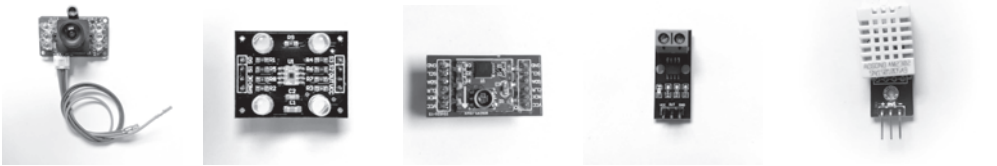


Figure 5.2 The functional blocks of a typical sensor node in IoT

Figure 5.3 shows some commercially available sensors used for sensing applications.



(a) Camera sensor (b) Color sensor (c) Compass and barometer (d) Current sensor (e) Digital temperature and humidity sensor

(f) Flame sensor (g) Gas sensor (h) Infrared sensor (i) Rainfall sensor (j) Ultrasonic distance measurement sensor

Figure 5.3 Some common commercially available sensors used for IoT-based sensing applications

5.3 Sensor Characteristics

All sensors can be defined by their ability to measure or capture a certain phenomenon and report them as output signals to various other systems. However, even within the same sensor type and class, sensors can be characterized by their ability to sense the phenomenon based on the following three fundamental properties.

- **Sensor Resolution**: The smallest change in the measurable quantity that a sensor can detect is referred to as the resolution of a sensor. For digital sensors, the smallest change in the digital output that the sensor is capable of quantifying is its sensor resolution. The more the resolution of a sensor, the more accurate is the precision. A sensor's accuracy does not depend upon its resolution. For example, a temperature sensor **A** can detect up to 0.5° C changes in temperature; whereas another sensor **B** can detect up to 0.25° C changes in temperature. Therefore, the resolution of sensor **B** is higher than the resolution of sensor **A**.
- **Sensor Accuracy**: The accuracy of a sensor is the ability of that sensor to measure the environment of a system as close to its true measure as possible. For example, a weight sensor detects the weight of a 100 kg mass as 99.98 kg. We can say that this sensor is 99.98% accurate, with an error rate of ±0.02%.
- **Sensor Precision**: The principle of repeatability governs the precision of a sensor. Only if, upon multiple repetitions, the sensor is found to have the same error

rate, can it be deemed as highly precise. For example, consider if the same weight sensor described earlier reports measurements of 98.28 kg, 100.34 kg, and 101.11 kg upon three repeat measurements for a mass of actual weight of 100 kg. Here, the sensor precision is not deemed high because of significant variations in the temporal measurements for the same object under the same conditions.

Points to ponder

The more the resolution of a sensor, the more accurate is the precision. A sensor's accuracy does not depend upon its resolution.

5.4 Sensorial Deviations

In this section, we will discuss the various sensorial deviations that are considered as errors in sensors. Most of the sensing in IoT is non-critical, where minor deviations in sensorial outputs seldom change the nature of the undertaken tasks. However, some critical applications of IoT, such as healthcare, industrial process monitoring, and others, do require sensors with high-quality measurement capabilities. As the quality of the measurement obtained from a sensor is dependent on a large number of factors, there are a few primary considerations that must be incorporated during the sensing of critical systems.

In the event of a sensor's output signal going beyond its designed maximum and minimum capacity for measurement, the sensor output is truncated to its maximum or minimum value, which is also the sensor's limits. The measurement range between a sensor's characterized minimum and maximum values is also referred to as the full-scale range of that sensor. Under real conditions, the sensitivity of a sensor may differ from the value specified for that sensor leading to *sensitivity error*. This deviation is mostly attributed to sensor fabrication errors and its calibration.

If the output of a sensor differs from the actual value to be measured by a constant, the sensor is said to have an *offset error* or *bias*. For example, while measuring an actual temperature of 0° C, a temperature sensor outputs 1.1° C every time. In this case, the sensor is said to have an offset error or bias of 1.1° C.

Similarly, some sensors have a non-linear behavior. If a sensor's transfer function (TF) deviates from a straight line transfer function, it is referred to as its non-linearity. The amount a sensor's actual output differs from the ideal TF behavior over the full range of the sensor quantifies its behavior. It is denoted as the percentage of the sensor's full range. Most sensors have linear behavior. If the output signal of a sensor changes slowly and independently of the measured property, this behavior of the sensor's output is termed as *drift*. Physical changes in the sensor or its material may result in long-term drift, which can span over months or years. Noise is a temporally varying random deviation of signals.

In contrast, if a sensor's output varies/deviates due to deviations in the sensor's previous input values, it is referred to as *hysteresis error*. The present output of the sensor depends on the past input values provided to the sensor. Typically, the phenomenon of hysteresis can be observed in analog sensors, magnetic sensors, and during heating of metal strips. One way to check for hysteresis error is to check how the sensor's output changes when we first increase, then decrease the input values to the sensor over its full range. It is generally denoted as a positive and negative percentage variation of the full-range of that sensor.

Focusing on digital sensors, if the digital output of a sensor is an approximation of the measured property, it induces *quantization error*. This error can be defined as the difference between the actual analog signal and its closest digital approximation during the sampling stage of the analog to digital conversion. Similarly, dynamic errors caused due to mishandling of sampling frequencies can give rise to *aliasing errors*. Aliasing leads to different signals of varying frequencies to be represented as a single signal in case the sampling frequency is not correctly chosen, resulting in the input signal becoming a multiple of the sampling rate.

Finally, the environment itself plays a crucial role in inducing sensorial deviations. Some sensors may be prone to external influences, which may not be directly linked to the property being measured by the sensor. This sensitivity of the sensor may lead to deviations in its output values. For example, as most sensors are semiconductor-based, they are influenced by the temperature of their environment.

5.5 Sensing Types

Sensing can be broadly divided into four different categories based on the nature of the environment being sensed and the physical sensors being used to do so (Figure 5.4): 1) scalar sensing, 2) multimedia sensing, 3) hybrid sensing, and 4) virtual sensing—[2].

5.5.1 Scalar sensing

Scalar sensing encompasses the sensing of features that can be quantified simply by measuring changes in the amplitude of the measured values with respect to time [3]. Quantities such as ambient temperature, current, atmospheric pressure, rainfall, light, humidity, flux, and others are considered as scalar values as they normally do not have a directional or spatial property assigned with them. Simply measuring the changes in their values with passing time provides enough information about these quantities. The sensors used for measuring these scalar quantities are referred to as scalar sensors, and the act is known as scalar sensing. Figures 5.3(b), 5.3(d), 5.3(e), 5.3(f), 5.3(g), 5.3(h), 5.3(i), and 5.3(j) show scalar sensors. A simple scalar temperature sensing of a fire detection event is shown in Figure 5.4(a).

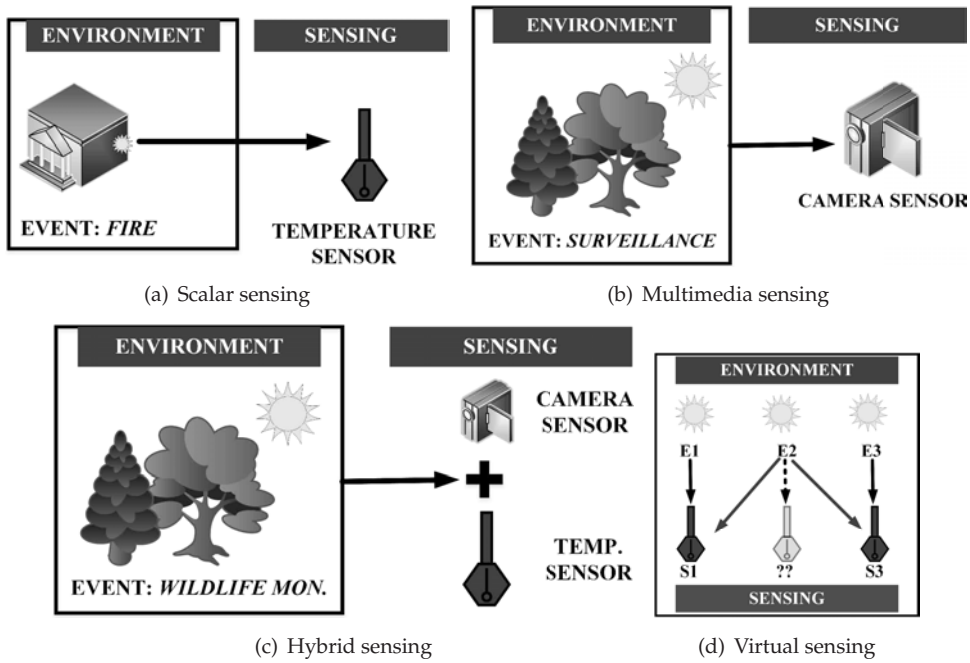


Figure 5.4 The different sensing types commonly encountered in IoT

5.5.2 Multimedia sensing

Multimedia sensing encompasses the sensing of features that have a spatial variance property associated with the property of temporal variance [4]. Unlike scalar sensors, multimedia sensors are used for capturing the changes in amplitude of a quantifiable property concerning space (spatial) as well as time (temporal). Quantities such as images, direction, flow, speed, acceleration, sound, force, mass, energy, and momentum have both directions as well as a magnitude. Additionally, these quantities follow the vector law of addition and hence are designated as vector quantities. They might have different values in different directions for the same working condition at the same time. The sensors used for measuring these quantities are known as vector sensors. Figures 5.3(a) and 5.3(c) are vector sensors. A simple camera-based multimedia sensing using surveillance as an example is shown in Figure 5.4(b).

5.5.3 Hybrid sensing

The act of using scalar as well as multimedia sensing at the same time is referred to as hybrid sensing. Many a time, there is a need to measure certain vector as well as scalar properties of an environment at the same time. Under these conditions, a range of various sensors are employed (from the collection of scalar as well as multimedia sensors) to measure the various properties of that environment at any instant of

time, and temporally map the collected information to generate new information. For example, in an agricultural field, it is required to measure the soil conditions at regular intervals of time to determine plant health. Sensors such as soil moisture and soil temperature are deployed underground to estimate the soil's water retention capacity and the moisture being held by the soil at any instant of time. However, this setup only determines whether the plant is getting enough water or not. There may be a host of other factors besides water availability, which may affect a plant's health. The additional inclusion of a camera sensor with the plant may be able to determine the actual condition of a plant by additionally determining the color of leaves. The aggregate information from soil moisture, soil temperature, and the camera sensor will be able to collectively determine a plant's health at any instant of time. Other common examples of hybrid sensing include smart parking systems, traffic management systems, and others. Figure 5.4(c) shows an example of hybrid sensing, where a camera and a temperature sensor are collectively used to detect and confirm forest fires during wildlife monitoring.

5.5.4 Virtual sensing

Many a time, there is a need for very dense and large-scale deployment of sensor nodes spread over a large area for monitoring of parameters. One such domain is agriculture [5]. Here, often, the parameters being measured, such as soil moisture, soil temperature, and water level, do not show significant spatial variations. Hence, if sensors are deployed in the fields of farmer **A**, it is highly likely that the measurements from his sensors will be able to provide almost concise measurements of his neighbor **B**'s fields; this is especially true of fields which are immediately surrounding **A**'s fields. Exploiting this property, if the data from **A**'s field is digitized using an IoT infrastructure and this system advises him regarding the appropriate watering, fertilizer, and pesticide regimen for his crops, this advisory can also be used by **B** for maintaining his crops. In short, **A** 's sensors are being used for actual measurement of parameters; whereas virtual data (which does not have actual physical sensors but uses extrapolation-based measurements) is being used for advising **B**. This is the virtual sensing paradigm. Figure 5.4(d) shows an example of virtual sensing. Two temperature sensors S1 and S3 monitor three nearby events E1, E2, and E3 (fires). The event E2 does not have a dedicated sensor for monitoring it; however, through the superposition of readings from sensors S1 and S3, the presence of fire in E2 is inferred.

5.6 Sensing Considerations

The choice of sensors in an IoT sensor node is critical and can either make or break the feasibility of an IoT deployment. The following major factors influence the choice of sensors in IoT-based sensing solutions: 1) sensing range, 2) accuracy and precision, 3) energy, and 4) device size. These factors are discussed as follows:

(i) **Sensing Range**: The sensing range of a sensor node defines the detection fidelity of that node. Typical approaches to optimize the sensing range in deployments include fixed k-coverage and dynamic k-coverage. A lifelong fixed k-coverage tends to usher in redundancy as it requires a large number of sensor nodes, the sensing range of some of which may also overlap. In contrast, dynamic k-coverage incorporates mobile sensor nodes post detection of an event, which, however, is a costly solution and may not be deployable in all operational areas and terrains [1].

Additionally, the sensing range of a sensor may also be used to signify the upper and lower bounds of a sensor's measurement range. For example, a proximity sensor has a typical sensing range of a couple of meters. In contrast, a camera has a sensing range varying between tens of meters to hundreds of meters. As the complexity of the sensor and its sensing range goes up, its cost significantly increases.

(ii) **Accuracy and Precision**: The accuracy and precision of measurements provided by a sensor are critical in deciding the operations of specific functional processes. Typically, off-the-shelf consumer sensors are low on requirements and often very cheap. However, their performance is limited to regular application domains. For example, a standard temperature sensor can be easily integrated with conventional components for hobby projects and day-to-day applications, but it is not suitable for industrial processes. Regular temperature sensors have a very low-temperature sensing range, as well as relatively low accuracy and precision. The use of these sensors in industrial applications, where a precision of up to 3–4 decimal places is required, cannot be facilitated by these sensors. Industrial sensors are typically very sophisticated, and as a result, very costly. However, these industrial sensors have very high accuracy and precision score, even under harsh operating conditions.

(iii) **Energy**: The energy consumed by a sensing solution is crucial to determine the lifetime of that solution and the estimated cost of its deployment. If the sensor or the sensor node is so energy inefficient that it requires replenishment of its energy sources quite frequently, the effort in maintaining the solution and its cost goes up; whereas its deployment feasibility goes down. Consider a scenario where sensor nodes are deployed on the top of glaciers. Once deployed, access to these nodes is not possible. If the energy requirements of the sensor nodes are too high, such a deployment will not last long, and the solution will be highly infeasible as charging or changing of the energy sources of these sensor nodes is not an option.

(iv) **Device Size**: Modern-day IoT applications have a wide penetration in all domains of life. Most of the applications of IoT require sensing solutions which are so small that they do not hinder any of the regular activities that were possible before the sensor node deployment was carried out. Larger the size of a sensor node, larger is the obstruction caused by it, higher is the cost and

energy requirements, and lesser is its demand for the bulk of the IoT applications. Consider a simple human activity detector. If the detection unit is too large to be carried or too bulky to cause hindrance to regular normal movements, the demand for this solution would be low. It is because of this that the onset of wearables took off so strongly. The wearable sensors are highly energy-efficient, small in size, and almost part of the wearer's regular wardrobe.

Check yourself

Principle of virtualization, MEMS

5.7 Actuators

An actuator can be considered as a machine or system's component that can affect the movement or control the said mechanism or the system. Control systems affect changes to the environment or property they are controlling through actuators. The system activates the actuator through a control signal, which may be digital or analog. It elicits a response from the actuator, which is in the form of some form of mechanical motion. The control system of an actuator can be a mechanical or electronic system, a software-based system (e.g., an autonomous car control system), a human, or any other input. Figure 5.5 shows the outline of a simple actuation system. A remote user sends commands to a processor. The processor instructs a motor controlled robotic arm to perform the commanded tasks accordingly. The processor is primarily responsible for converting the human commands into sequential machine-language command sequences, which enables the robot to move. The robotic arm finally moves the designated boxes, which was its assigned task.

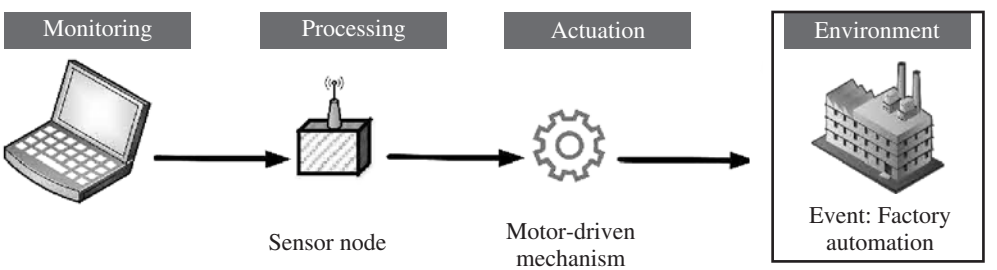


Figure 5.5 The outline of a simple actuation mechanism

5.8 Actuator Types

Broadly, actuators can be divided into seven classes: 1) Hydraulic, 2) pneumatic, 3) electrical, 4) thermal/magnetic, 5) mechanical, 6) soft, and 7) shape memory polymers. Figure 5.6 shows some of the commonly used actuators in IoT applications.

5.8.1 Hydraulic actuators

A hydraulic actuator works on the principle of compression and decompression of fluids. These actuators facilitate mechanical tasks such as lifting loads through the use of hydraulic power derived from fluids in cylinders or fluid motors. The mechanical motion applied to a hydraulic actuator is converted to either linear, rotary, or oscillatory motion. The almost incompressible property of liquids is used in hydraulic actuators for exerting significant force. These hydraulic actuators are also considered as stiff systems. The actuator's limited acceleration restricts its usage.

5.8.2 Pneumatic actuators

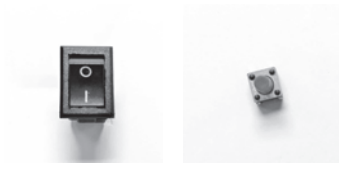
A pneumatic actuator works on the principle of compression and decompression of gases. These actuators use a vacuum or compressed air at high pressure and convert it into either linear or rotary motion. Pneumatic rack and pinion actuators are commonly used for valve controls of water pipes. Pneumatic actuators are considered as compliant systems. The actuators using pneumatic energy for their operation are typically characterized by the quick response to starting and stopping signals. Small pressure changes can be used for generating large forces through these actuators. Pneumatic brakes are an example of this type of actuator which is so responsive that they can convert small pressure changes applied by drives to generate the massive force required to stop or slow down a moving vehicle. Pneumatic actuators are responsible for converting pressure into force. The power source in the pneumatic actuator does not need to be stored in reserve for its operation.

5.8.3 Electric actuators

Typically, electric motors are used to power an electric actuator by generating mechanical torque. This generated torque is translated into the motion of a motor's shaft or for switching (as in relays). For example, actuating equipments such as solenoid valves control the flow of water in pipes in response to electrical signals. This class of actuators is considered one of the cheapest, cleanest and speedy actuator types available. Figures 5.6(a), 5.6(b), 5.6(c), 5.6(d), 5.6(e), 5.6(f), 5.6(i), and 5.6(j) show some of the commonly used electrical actuators.

(a) Brushless DC motor (b) Brushless DC motor (c) Stepper motor (d) Geared stepper motor (e) DC motor

(f) Relay array (g) Hydroelectric generator (h) Hydroelectric generator (i) Solenoid-based flow valve (j) Solenoid-based flow valve



(k) DPDT switch (l) Push button switch

Figure 5.6 Some common commercially available actuators used for IoT-based control applications

5.8.4 Thermal or magnetic actuators

The use of thermal or magnetic energy is used for powering this class of actuators. These actuators have a very high power density and are typically compact, lightweight, and economical. One classic example of thermal actuators is shape memory materials (SMMs) such as shape memory alloys (SMAs). These actuators do not require electricity for actuation. They are not affected by vibration and can work with liquid or gases. Magnetic shape memory alloys (MSMAs) are a type of magnetic actuators.

5.8.5 Mechanical actuators

In mechanical actuation, the rotary motion of the actuator is converted into linear motion to execute some movement. The use of gears, rails, pulleys, chains, and other devices are necessary for these actuators to operate. These actuators can be easily used in conjunction with pneumatic, hydraulic, or electrical actuators. They can also work in a standalone mode. The best example of a mechanical actuator is a rack and pinion mechanism. Figures 5.6(g), 5.6(h), 5.6(k), and 5.6(l) show some of the commonly available mechanical actuators. The hydroelectric generator shown in

Figures 5.6(g) and 5.6(h) convert the water-flow induced rotary motion of a turbine into electrical energy. Similarly, the mechanical switches shown in Figures 5.6 (k) and 5.6(l) uses the mechanical motion of the switch to switch on or off an electrical circuit.

5.8.6 Soft actuators

Soft actuators (e.g., polymer-based) consists of elastomeric polymers that are used as embedded fixtures in flexible materials such as cloth, paper, fiber, particles, and others [7]. The conversion of molecular level microscopic changes into tangible macroscopic deformations is the primary working principle of this class of actuators. These actuators have a high stake in modern-day robotics. They are designed to handle fragile objects such as agricultural fruit harvesting, or performing precise operations like manipulating the internal organs during robot-assisted surgeries.

5.8.7 Shape memory polymers

Shape memory polymers (SMP) are considered as smart materials that respond to some external stimulus by changing their shape, and then revert to their original shape once the affecting stimulus is removed [6]. Features such as high strain recovery, biocompatibility, low density, and biodegradability characterize these materials. SMP-based actuators function similar to our muscles. Modern-day SMPs have been designed to respond to a wide range of stimuli such as pH changes, heat differentials, light intensity, and frequency changes, magnetic changes, and others.

Photopolymer/light-activated polymers (LAP) are a particular type of SMP, which require light as a stimulus to operate. LAP-based actuators are characterized by their rapid response times. Using only the variation of light frequency or its intensity, LAPs can be controlled remotely without any physical contact. The development of LAPs whose shape can be changed by the application of a specific frequency of light have been reported. The polymer retains its shape after removal of the activating light. In order to change the polymer back to its original shape, a light stimulus of a different frequency has to be applied to the polymer.

5.9 Actuator Characteristics

The choice or selection of actuators is crucial in an IoT deployment, where a control mechanism is required after sensing and processing of the information obtained from the sensed environment. Actuators perform the physically heavier tasks in an IoT deployment; tasks which require moving or changing the orientation of physical objects, changing the state of objects, and other such activities. The correct choice of actuators is necessary for the long-term sustenance and continuity of operations, as well as for increasing the lifetime of the actuators themselves. A set of four characteristics can define all actuators:

- **Weight**: The physical weight of actuators limits its application scope. For example, the use of heavier actuators is generally preferred for industrial applications and applications requiring no mobility of the IoT deployment. In contrast, lightweight actuators typically find common usage in portable systems in vehicles, drones, and home IoT applications. It is to be noted that this is not always true. Heavier actuators also have selective usage in mobile systems, for example, landing gears and engine motors in aircraft.
- **Power Rating**: This helps in deciding the nature of the application with which an actuator can be associated. The power rating defines the minimum and maximum operating power an actuator can safely withstand without damage to itself. Generally, it is indicated as the power-to-weight ratio for actuators. For example, smaller servo motors used in hobby projects typically have a maximum rating of 5 VDC, 500 mA, which is suitable for an operations-driven battery-based power source. Exceeding this limit might be detrimental to the performance of the actuator and may cause burnout of the motor. In contrast to this, servo motors in larger applications have a rating of 460 VAC, 2.5 A, which requires standalone power supply systems for operations. It is to be noted that actuators with still higher ratings are available and vary according to application requirements.
- **Torque to Weight Ratio**: The ratio of torque to the weight of the moving part of an instrument/device is referred to as its torque/weight ratio. This indicates the sensitivity of the actuator. Higher is the weight of the moving part; lower will be its torque to weight ratio for a given power.
- **Stiffness and Compliance**: The resistance of a material against deformation is known as its stiffness, whereas compliance of a material is the opposite of stiffness. Stiffness can be directly related to the modulus of elasticity of that material. Stiff systems are considered more accurate than compliant systems as they have a faster response to the change in load applied to it. For example, hydraulic systems are considered as stiff and non-compliant, whereas pneumatic systems are considered as compliant.

Check yourself

Operation of PLC and SCADA, Working principle of electric motors, applications of pneumatic and hydraulic actuators, Differences between pneumatic, hydraulic, electrical, and mechanical actuators

Summary

This chapter covered the basics of sensing and actuation in order to help the readers grasp the intricacies of designing an IoT solution keeping in mind the need to select

the proper sensors and actuators. The first part of this chapter discusses sensors, sensing characteristics, considerations of various sensorial deviations, and the sensing types possible in a typical IoT-based implementation of a sensing solution. This part concludes with a discussion on the various considerations to be thought of while selecting sensors for architecting a viable IoT-based sensing solution. The second part of this chapter focuses on actuators and the broad classes of actuators available. This part concludes with a discussion on the various considerations to be thought of while selecting actuators for architecting a viable IoT-based control solution using actuators. After completing this chapter, the reader will be able to decide upon the most appropriate sensing and actuation solutions to use with their IoT-based applications.

Exercises

(i) Differentiate between sensors and actuators.

(ii) Differentiate between sensors and transducers.

(iii) How is sensor resolution different from its accuracy?

(iv) Differentiate between scalar and vector sensors.

(v) Differentiate between analog and digital sensors.

(vi) What is a an offset error?

(vii) What is a hysteresis error?

(viii) What is a quantization error?

(ix) What is aliasing error?

(x) Differentiate between hydraulic and pneumatic actuators with examples.

(xi) What are shape memory alloys (SMA)?

(xii) What are soft actuators?

(xiii) What are the main features of shape memory polymers?

(xiv) What are light activated polymers?

References

[1] Alam, Kh Mahmudul, Joarder Kamruzzaman, Gour Karmakar, and Manzur Murshed. 2014. "Dynamic Adjustment of Sensing Range for Event Coverage in Wireless Sensor Networks." *Journal of Network and Computer Applications* 46: 139–153. Elsevier.

[2] Popović, T., N. Latinović, A. Pešić, Z. Zečević, B. Krstajić, and S. Djukanović. 2017. "Architecting an IoT-enabled Platform for Precision Agriculture and Ecological Monitoring: A Case Study." *Computers and Electronics in Agriculture* 140: 255–265.

[3] Kelly, S. D. T., N. K. Suryadevara, and S. C. Mukhopadhyay. 2013. "Towards the Implementation of IoT for Environmental Condition Monitoring in Homes." *IEEE Sensors Journal* 13(10): 3846–3853.

[4] Rosário, D., Z. Zhao, A. Santos, T. Braun, and E. Cerqueira. 2014. "A Beaconless Opportunistic Routing based on a Cross-layer Approach for Efficient Video Dissemination in Mobile Multimedia IoT Applications." *Computer Communications* 45: 21–31.

[5] Ojha, T., S. Misra, N. S. Raghuwanshi, and H. Poddar. 2019. "DVSP: Dynamic Virtual Sensor Provisioning in Sensor-Cloud based Internet of Things." *IEEE Internet of Things Journal* 6 (3): 5265–5272.

[6] Lendlein, A., and O. E. Gould. 2019. "Reprogrammable Recovery and Actuation Behaviour of Shape-memory Polymers." *Nature Reviews Materials* 4(2): 116–133.

[7] Lessing, J. A., R. V. Martinez, A. S. Tayi, J. M. Ting, and G. M. Whitesides. 2019. "Flexible and Stretchable Electronic Strain-limited Layer for Soft Actuators." U. S. Patent Application 15/972, 412.

Chapter 6

IoT Processing Topologies and Types

Learning Outcomes

After reading this chapter, the reader will be able to:

- List common data types in IoT applications
- Understand the importance of processing
- Explain the various processing topologies in IoT
- Understand the importance of processing off-loading toward achieving scalability and cost-effectiveness of IoT solutions
- Determine the importance of choosing the right processing topologies and associated considerations while designing IoT applications
- Determine the requirements that are associated with IoT-based processing of sensed and communicated data.

6.1 Data Format

The Internet is a vast space where huge quantities and varieties of data are generated regularly and flow freely. As of January 2018, there are a reported 4.021 billion Internet users worldwide. The massive volume of data generated by this huge number of users is further enhanced by the multiple devices utilized by most users. In addition to these data-generating sources, non-human data generation sources such as sensor nodes and automated monitoring systems further add to the data load on the Internet. This huge data volume is composed of a variety of data such as e-mails, text documents (Word docs, PDFs, and others), social media posts, videos, audio files, and images, as shown in Figure 6.1. However, these data can be broadly grouped into two types

based on how they can be accessed and stored: 1) Structured data and 2) unstructured data.

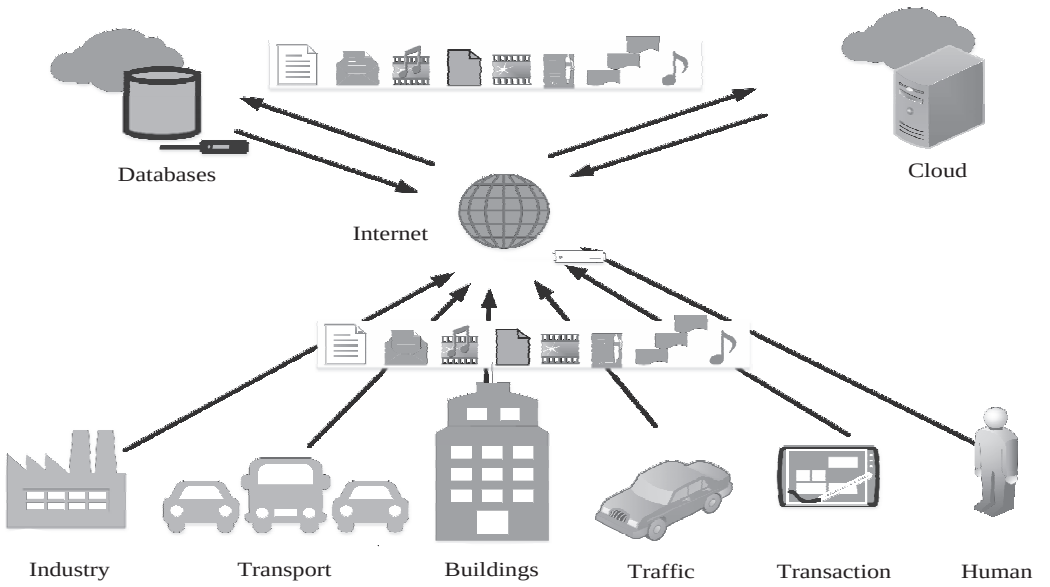


Figure 6.1 The various data generating and storage sources connected to the Internet and the plethora of data types contained within it

6.1.1 Structured data

These are typically text data that have a pre-defined structure [1]. Structured data are associated with relational database management systems (RDBMS). These are primarily created by using length-limited data fields such as phone numbers, social security numbers, and other such information. Even if the data is human or machine-generated, these data are easily searchable by querying algorithms as well as human-generated queries. Common usage of this type of data is associated with flight or train reservation systems, banking systems, inventory controls, and other similar systems. Established languages such as Structured Query Language (SQL) are used for accessing these data in RDBMS. However, in the context of IoT, structured data holds a minor share of the total generated data over the Internet.

6.1.2 Unstructured data

In simple words, all the data on the Internet, which is not structured, is categorized as unstructured. These data types have no pre-defined structure and can vary according to applications and data-generating sources. Some of the common examples of human-generated unstructured data include text, e-mails, videos, images, phone

recordings, chats, and others [2]. Some common examples of machine-generated unstructured data include sensor data from traffic, buildings, industries, satellite imagery, surveillance videos, and others. As already evident from its examples, this data type does not have fixed formats associated with it, which makes it very difficult for querying algorithms to perform a look-up. Querying languages such as NoSQL are generally used for this data type.

6.2 Importance of Processing in IoT

The vast amount and types of data flowing through the Internet necessitate the need for intelligent and resourceful processing techniques. This necessity has become even more crucial with the rapid advancements in IoT, which is laying enormous pressure on the existing network infrastructure globally. Given these urgencies, it is important to decide—*when to process and what to process*? Before deciding upon the processing to pursue, we first divide the data to be processed into three types based on the urgency of processing: 1) Very time critical, 2) time critical, and 3) normal. Data from sources such as flight control systems [3], healthcare, and other such sources, which need immediate decision support, are deemed as very critical. These data have a very low threshold of processing latency, typically in the range of a few milliseconds.

Data from sources that can tolerate normal processing latency are deemed as time-critical data. These data, generally associated with sources such as vehicles, traffic, machine systems, smart home systems, surveillance systems, and others, which can tolerate a latency of a few seconds fall in this category. Finally, the last category of data, normal data,can tolerate a processing latency of a few minutes to a few hours and are typically associated with less data-sensitive domains such as agriculture, environmental monitoring, and others.

Considering the requirements of data processing, the processing requirements of data from very time-critical sources are exceptionally high. Here, the need for processing the data in place or almost nearer to the source is crucial in achieving the deployment success of such domains. Similarly, considering the requirements of processing from category 2 data sources (time-critical), the processing requirements allow for the transmission of data to be processed to remote locations/processors such as clouds or through collaborative processing. Finally, the last category of data sources (normal) typically have no particular time requirements for processing urgently and are pursued leisurely as such.

Check yourself

Difference between microprocessors and microcontrollers, network bandwidth, network latency

6.3 Processing Topologies

The identification and intelligent selection of processing requirement of an IoT application are one of the crucial steps in deciding the architecture of the deployment. A properly designed IoT architecture would result in massive savings in network bandwidth and conserve significant amounts of overall energy in the architecture while providing the proper and allowable processing latencies for the solutions associated with the architecture. Regarding the importance of processing in IoT as outlined in Section 6.2, we can divide the various processing solutions into two large topologies: 1) On-site and 2) Off-site. The off-site processing topology can be further divided into the following: 1) Remote processing and 2) Collaborative processing.

6.3.1 On-site processing

As evident from the name, the on-site processing topology signifies that the data is processed at the source itself. This is crucial in applications that have a very low tolerance for latencies. These latencies may result from the processing hardware or the network (during transmission of the data for processing away from the processor). Applications such as those associated with healthcare and flight control systems (real-time systems) have a breakneck data generation rate. These additionally show rapid temporal changes that can be missed (leading to catastrophic damages) unless the processing infrastructure is fast and robust enough to handle such data. Figure 6.2 shows the on-site processing topology, where an event (here, fire) is detected utilizing a temperature sensor connected to a sensor node. The sensor node processes the information from the sensed event and generates an alert. The node additionally has the option of forwarding the data to a remote infrastructure for further analysis and storage.

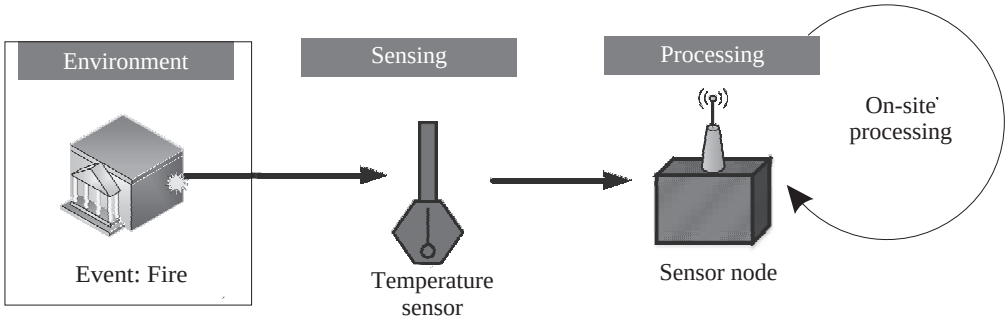


Figure 6.2 Event detection using an on-site processing topology

6.3.2 Off-site processing

The off-site processing paradigm, as opposed to the on-site processing paradigms, allows for latencies (due to processing or network latencies); it is significantly cheaper than on-site processing topologies. This difference in cost is mainly due to the low demands and requirements of processing at the source itself. Often, the sensor nodes are not required to process data on an urgent basis, so having a dedicated and expensive on-site processing infrastructure is not sustainable for large-scale deployments typical of IoT deployments. In the off-site processing topology, the sensor node is responsible for the collection and framing of data that is eventually to be transmitted to another location for processing. Unlike the on-site processing topology, the off-site topology has a few dedicated high-processing enabled devices, which can be borrowed by multiple simpler sensor nodes to accomplish their tasks. At the same time, this arrangement keeps the costs of large-scale deployments extremely manageable [5]. In the off-site topology, the data from these sensor nodes (data generating sources) is transmitted either to a remote location (which can either be a server or a cloud) or to multiple processing nodes. Multiple nodes can come together to share their processing power in order to collaboratively process the data (which is important in case a feasible communication pathway or connection to a remote location cannot be established by a single node).

Remote processing

This is one of the most common processing topologies prevalent in present-day IoT solutions. It encompasses sensing of data by various sensor nodes; the data is then forwarded to a remote server or a cloud-based infrastructure for further processing and analytics. The processing of data from hundreds and thousands of sensor nodes can be simultaneously offloaded to a single, powerful computing platform; this results in massive cost and energy savings by enabling the reuse and reallocation of the same processing resource while also enabling the deployment of smaller and simpler processing nodes at the site of deployment [4]. This setup also ensures massive scalability of solutions, without significantly affecting the cost of the deployment. Figure 6.3 shows the outline of one such paradigm, where the sensing of an event is performed locally, and the decision making is outsourced to a remote processor (here, cloud). However, this paradigm tends to use up a lot of network bandwidth and relies heavily on the presence of network connectivity between the sensor nodes and the remote processing infrastructure.

Collaborative processing

This processing topology typically finds use in scenarios with limited or no network connectivity, especially systems lacking a backbone network. Additionally, this topology can be quite economical for large-scale deployments spread over vast areas, where providing networked access to a remote infrastructure is not viable. In such scenarios, the simplest solution is to club together the processing power of nearby

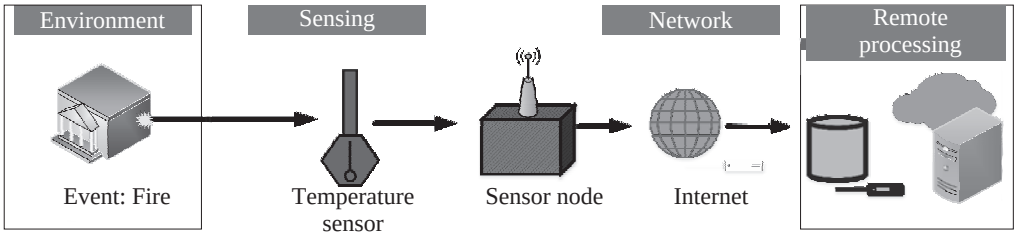


Figure 6.3 Event detection using an off-site remote processing topology

processing nodes and collaboratively process the data in the vicinity of the data source itself. This approach also reduces latencies due to the transfer of data over the network. Additionally, it conserves bandwidth of the network, especially ones connecting to the Internet. Figure 6.4 shows the collaborative processing topology for collaboratively processing data locally. This topology can be quite beneficial for applications such as agriculture, where an intense and temporally high frequency of data processing is not required as agricultural data is generally logged after significantly long intervals (in the range of hours). One important point to mention about this topology is the preference of mesh networks for easy implementation of this topology.

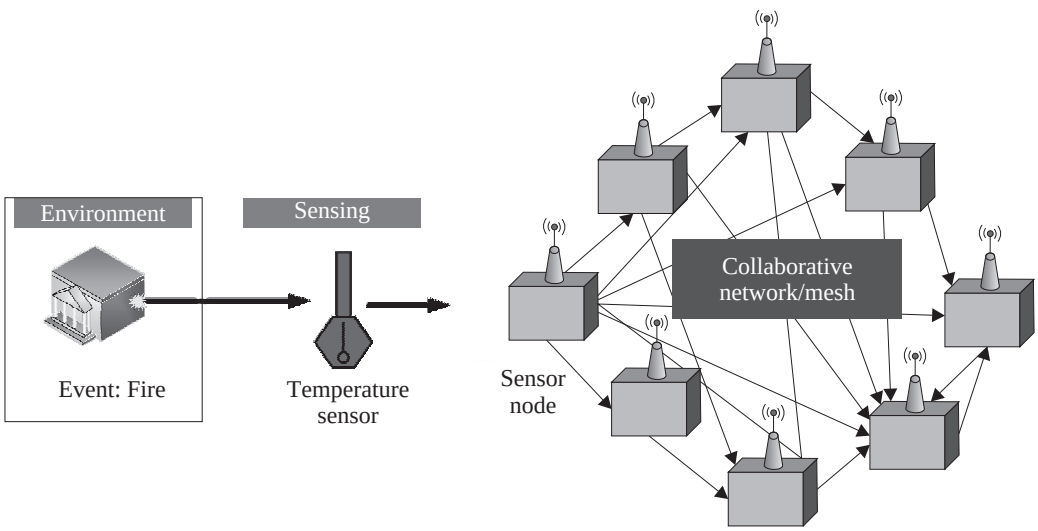


Figure 6.4 Event detection using a collaborative processing topology

6.4 IoT Device Design and Selection Considerations

The main consideration of minutely defining an IoT solution is the selection of the processor for developing the sensing solution (i.e., the sensor node). This selection is governed by many parameters that affect the usability, design, and affordability of the

designed IoT sensing and processing solution. In this chapter, we mainly focus on the deciding factors for selecting a processor for the design of a sensor node. The main factor governing the IoT device design and selection for various applications is the processor. However, the other important considerations are as follows.

- **Size**: This is one of the crucial factors for deciding the form factor and the energy consumption of a sensor node. It has been observed that larger the form factor, larger is the energy consumption of the hardware. Additionally, large form factors are not suitable for a significant bulk of IoT applications, which rely on minimal form factor solutions (e.g., wearables).
- **Energy**: The energy requirements of a processor is the most important deciding factor in designing IoT-based sensing solutions. Higher the energy requirements, higher is the energy source (battery) replacement frequency. This principle automatically lowers the long-term sustainability of sensing hardware, especially for IoT-based applications.
- **Cost**: The cost of a processor, besides the cost of sensors, is the driving force in deciding the density of deployment of sensor nodes for IoT-based solutions. Cheaper cost of the hardware enables a much higher density of hardware deployment by users of an IoT solution. For example, cheaper gas and fire detection solutions would enable users to include much more sensing hardware for a lesser cost.
- **Memory**: The memory requirements (both volatile and non-volatile memory) of IoT devices determines the capabilities the device can be armed with. Features such as local data processing, data storage, data filtering, data formatting, and a host of other features rely heavily on the memory capabilities of devices. However, devices with higher memory tend to be costlier for obvious reasons.
- **Processing power**: As covered in earlier sections, processing power is vital (comparable to memory) in deciding what type of sensors can be accommodated with the IoT device/node, and what processing features can integrate on-site with the IoT device. The processing power also decides the type of applications the device can be associated with. Typically, applications that handle video and image data require IoT devices with higher processing power as compared to applications requiring simple sensing of the environment.
- **I/O rating**: The input–output (I/O) rating of IoT device, primarily the processor, is the deciding factor in determining the circuit complexity, energy usage, and requirements for support of various sensing solutions and sensor types. Newer processors have a meager I/O voltage rating of 3.3 V, as compared to 5 V for the somewhat older processors. This translates to requiring additional voltage and logic conversion circuitry to interface legacy technologies and sensors with the newer processors. Despite low power consumption due to reduced I/O voltage levels, this additional voltage and circuitry not only affects the complexity of the circuits but also affects the costs.

- **Add-ons**: The support of various add-ons a processor or for that matter, an IoT device provides, such as analog to digital conversion (ADC) units, in-built clock circuits, connections to USB and ethernet, inbuilt wireless access capabilities, and others helps in defining the robustness and usability of a processor or IoT device in various application scenarios. Additionally, the provision for these add-ons also decides how fast a solution can be developed, especially the hardware part of the whole IoT application. As interfacing and integration of systems at the circuit level can be daunting to the uninitiated, the prior presence of these options with the processor makes the processor or device highly lucrative to the users/developers.

Check yourself

RISC versus CISC processors, volative versus non-volatile memory

6.5 Processing Offloading

The processing offloading paradigm is important for the development of densely deployable, energy-conserving, miniaturized, and cheap IoT-based solutions for sensing tasks. Building upon the basics of the off-site processing topology covered in the previous sections in this chapter, we delve a bit further into the various nuances of processing offloading in IoT.

Figure 6.5 shows the typical outline of an IoT deployment with the various layers of processing that are encountered spanning vastly different application domains—from as near as sensing the environment to as far as cloud-based infrastructure. Starting from the primary layer of sensing, we can have multiple sensing types tasked with detecting an environment (fire, surveillance, and others). The sensors enabling these sensing types are integrated with a processor using wired or wireless connections (mostly, wired). In the event that certain applications require immediate processing of the sensed data, an on-site processing topology is followed, similar to the one in Figure 6.2. However, for the majority of IoT applications, the bulk of the processing is carried out remotely in order to keep the on-site devices simple, small, and economical.

Typically, for off-site processing, data from the sensing layer can be forwarded to the fog or cloud or can be contained within the edge layer [6]. The edge layer makes use of devices within the local network to process data that which is similar to the collaborative processing topology shown in Figure 6.4. The devices within the local network, till the fog, generally communicate using short-range wireless connections. In case the data needs to be sent further up the chain to the cloud, long-range wireless connection enabling access to a backbone network is essential. Fog-based processing is still considered local because the fog nodes are typically localized within a geographic area and serve the IoT nodes within a much smaller coverage area as compared to the

cloud. Fog nodes, which are at the level of gateways, may or may not be accessed by the IoT devices through the Internet.

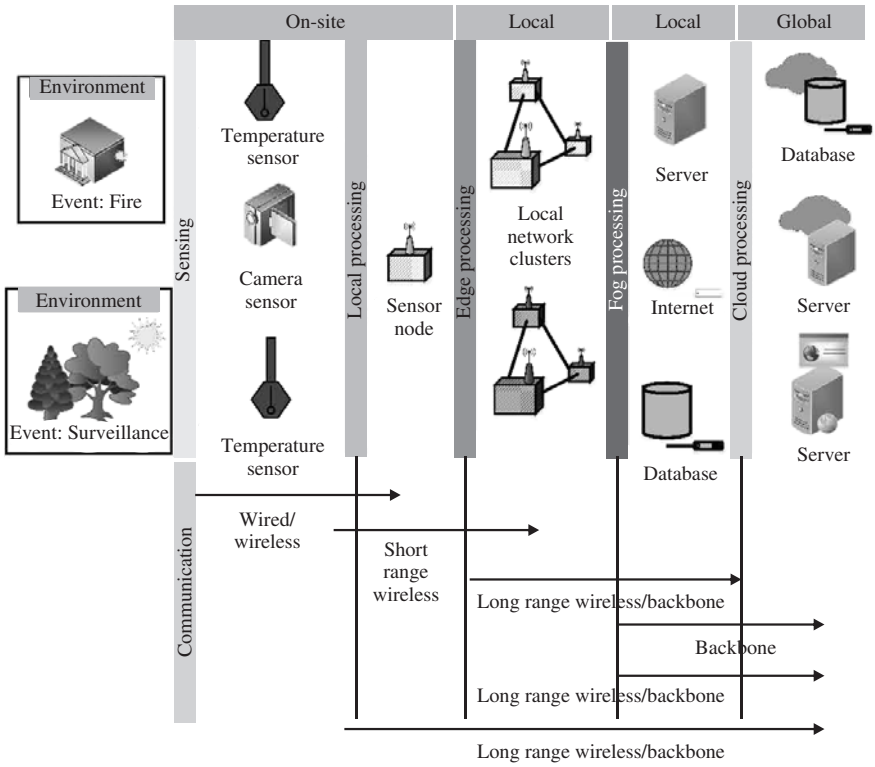


Figure 6.5 The various data generating and storage sources connected to the Internet and the plethora of data types contained within it

Finally, the approach of forwarding data to a cloud or a remote server, as shown in the topology in Figure 6.3, requires the devices to be connected to the Internet through long-range wireless/wired networks, which eventually connect to a backbone network. This approach is generally costly concerning network bandwidth, latency, as well as the complexity of the devices and the network infrastructure involved.

This section on data offloading is divided into three parts: 1) offload location (which outlines where all the processing can be offloaded in the IoT architecture), 2) offload decision making (how to choose where to offload the processing to and by how much), and finally 3) offloading considerations (deciding when to offload).

6.5.1 Offload location

The choice of offload location decides the applicability, cost, and sustainability of the IoT application and deployment. We distinguish the offload location into four types:

- **Edge**: Offloading processing to the edge implies that the data processing is facilitated to a location at or near the source of data generation itself. Offloading to the edge is done to achieve aggregation, manipulation, bandwidth reduction, and other data operations directly on an IoT device [7].
- **Fog**: Fog computing is a decentralized computing infrastructure that is utilized to conserve network bandwidth, reduce latencies, restrict the amount of data unnecessarily flowing through the Internet, and enable rapid mobility support for IoT devices. The data, computing, storage and applications are shifted to a place between the data source and the cloud resulting in significantly reduced latencies and network bandwidth usage [8].
- **Remote Server**: A simple remote server with good processing power may be used with IoT-based applications to offload the processing from resource-constrained IoT devices. Rapid scalability may be an issue with remote servers, and they may be costlier and hard to maintain in comparison to solutions such as the cloud [4].
- **Cloud**: Cloud computing is a configurable computer system, which can get access to configurable resources, platforms, and high-level services through a shared pool hosted remotely. A cloud is provisioned for processing offloading so that processing resources can be rapidly provisioned with minimal effort over the Internet, which can be accessed globally. Cloud enables massive scalability of solutions as they can enable resource enhancement allocated to a user or solution in an on-demand manner, without the user having to go through the pains of acquiring and configuring new and costly hardware [9].

6.5.2 Offload decision making

The choice of where to offload and how much to offload is one of the major deciding factors in the deployment of an offsite-processing topology-based IoT deployment architecture. The decision making is generally addressed considering data generation rate, network bandwidth, the criticality of applications, processing resource available at the offload site, and other factors. Some of these approaches are as follows.

- **Naive Approach**: This approach is typically a hard approach, without too much decision making. It can be considered as a rule-based approach in which the data from IoT devices are offloaded to the nearest location based on the achievement of certain offload criteria. Although easy to implement, this approach is never recommended, especially for dense deployments, or deployments where the data generation rate is high or the data being offloaded in complex to handle (multimedia or hybrid data types). Generally, statistical measures are consulted for generating the rules for offload decision making.
- **Bargaining based approach**: This approach, although a bit processing-intensive during the decision making stages, enables the alleviation of network traffic congestion, enhances service QoS (quality of service) parameters such as

bandwidth, latencies, and others. At times, while trying to maximize multiple parameters for the whole IoT implementation, in order to provide the most optimal solution or QoS, not all parameters can be treated with equal importance. Bargaining based solutions try to maximize the QoS by trying to reach a point where the qualities of certain parameters are reduced, while the others are enhanced. This measure is undertaken so that the achieved QoS is collaboratively better for the full implementation rather than a select few devices enjoying very high QoS. Game theory is a common example of the bargaining based approach. This approach does not need to depend on historical data for decision making purposes.

- **Learning based approach**: Unlike the bargaining based approaches, the learning based approaches generally rely on past behavior and trends of data flow through the IoT architecture. The optimization of QoS parameters is pursued by learning from historical trends and trying to optimize previous solutions further and enhance the collective behavior of the IoT implementation. The memory requirements and processing requirements are high during the decision making stages. The most common example of a learning based approach is machine learning.

6.5.3 Offloading considerations

There are a few offloading parameters which need to be considered while deciding upon the offloading type to choose. These considerations typically arise from the nature of the IoT application and the hardware being used to interact with the application. Some of these parameters are as follows.

- **Bandwidth**: The maximum amount of data that can be simultaneously transmitted over the network between two points is the bandwidth of that network. The bandwidth of a wired or wireless network is also considered to be its data-carrying capacity and often used to describe the data rate of that network.
- **Latency**: It is the time delay incurred between the start and completion of an operation. In the present context, latency can be due to the network (network latency) or the processor (processing latency). In either case, latency arises due to the physical limitations of the infrastructure, which is associated with an operation. The operation can be data transfer over a network or processing of a data at a processor.
- **Criticality**: It defines the importance of a task being pursued by an IoT application. The more critical a task is, the lesser latency is expected from the IoT solution. For example, detection of fires using an IoT solution has higher criticality than detection of agricultural field parameters. The former requires a response time in the tune of milliseconds, whereas the latter can be addressed within hours or even days.

- **Resources**: It signifies the actual capabilities of an offload location. These capabilities may be the processing power, the suite of analytical algorithms, and others. For example, it is futile and wasteful to allocate processing resources reserved for real-time multimedia processing (which are highly energy-intensive and can process and analyze huge volumes of data in a short duration) to scalar data (which can be addressed using nominal resources without wasting much energy).
- **Data volume**: The amount of data generated by a source or sources that can be simultaneously handled by the offload location is referred to as its data volume handling capacity. Typically, for large and dense IoT deployments, the offload location should be robust enough to address the processing issues related to massive data volumes.

Summary

This chapter started with an overview of the various data formats available on the Internet and to which various IoT solutions are exposed. The complexities in handling the numerous data formats available present a significant challenge to the design of IoT-based solutions. In order to address these challenges, the importance of processing in IoT is discussed. This discussion is followed by an introduction to various processing topologies, which can be chosen to address the challenges of IoT processing. These topologies are broadly made up of two categories: 1) On-site processing and 2) Off-site processing. The off-site processing is typically composed of approaches to offload data to locations which are not the same as the one from which the data was generated. A discussion on processing offloading follows these topologies. Various offload location types, means of deciding offload location and quantity are explained. Finally, the various parameters to be considered for offloading are discussed to enable the reader to grasp the nuances of processing in IoT.

Exercises

(i) What are the different data formats found in IoT network traffic streams?

(ii) Depending on the urgency of data processing, how are IoT data classified?

(iii) Highlight the pros and cons of on-site and off-site processing.

(iv) Differentiate between structured and unstructured data.

(v) How is collaborative processing different from remote processing?

(vi) What are the critical factors to be considered during the design of IoT devices?

(vii) What are the typical data offload locations available in the context of IoT?

(viii) What are the various decision making approaches chosen for offloading data in IoT?

(ix) What factors are to be considered while deciding on the data offload location?

References

[1] Misra, S., A. Mukherjee, and A. Roy. 2018. "Knowledge Discovery for Enabling Smart Internet of Things: A Survey." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8(6): 1276.

[2] Jiang, L., L. Da Xu, H. Cai, Z. Jiang, F. Bu, and B. Xu. 2014. "An IoT-oriented Data Storage Framework in Cloud Computing Platform." *IEEE Transactions on Industrial Informatics* 10(2): 1443–1451.

[3] Mukherjee, A., S. Misra, V. S. P. Chandra, and M. S. Obaidat. 2018. "Resource-Optimized Multi-Armed Bandit Based Offload Path Selection in Edge UAV Swarms." *IEEE Internet of Things Journal* 6(3): 4889–4896.

[4] Mukherjee, A., S. Misra, N. S. Raghuwanshi, and S. Mitra. 2018. "Blind Entity Identification for Agricultural IoT Deployments." *IEEE Internet of Things Journal* 6(2): 3156–3163.

[5] Mukherjee, A., N. Pathak, S. Misra, and S. Mitra. 2018. "Predictive Intra-Edge Packet-Source Mapping in Agricultural Internet of Things." In 2018 *IEEE Globecom Workshops (GC Wkshps)* (pp. 1–6). IEEE. doi: 10.1109/GLOCOMW.2018.8644296

[6] Cheng, N., F. Lyu, W. Quan, C. Zhou, H. He, W. Shi, and X. Shen. 2019. "Space/Aerial-Assisted Computing Offloading for IoT Applications: A Learning-Based Approach." *IEEE Journal on Selected Areas in Communications* 37(5): 1117–1129.

[7] Huang, L., X. Feng, C. Zhang, L. Qian, and Y. Wu. 2019. "Deep Reinforcement Learning-based Joint Task Offloading and Bandwidth Allocation for Multi-user Mobile Edge Computing." *Digital Communications and Networks* 5(1): 10–17.

[8] Adhikari, M., M. Mukherjee, and S. N. Srirama. 2019. "DPTO: A Deadline and Priority-aware Task Offloading in Fog Computing Framework Leveraging Multi-level Feedback Queueing." *IEEE Internet of Things Journal*. doi: 10.1109/JIOT.2019.2946426.

[9] Mahmoodi, S. E., K. Subbalakshmi, and R. N. Uma. 2019. "Classification of Mobile Cloud Offloading." In *Spectrum-Aware Mobile Computing* (pp. 7–11). Springer, Cham.

Chapter 7

IoT Connectivity Technologies

Learning Outcomes

After reading this chapter, the reader will be able to:

- List common connectivity protocols in IoT
- Identify the salient features and application scope of each connectivity protocol
- Understand the terminologies and technologies associated with IoT connectivity
- Determine the requirements associated with each of these connectivity protocols in real-world solutions
- Determine the most appropriate connectivity protocol for each segment of their IoT implementation

7.1 Introduction

This chapter outlines the main features of fifteen identified commonly used and upcoming IoT connectivity enablers. These connectivity technologies can be integrated with existing sensing, actuation, and processing solutions for extending connectivity to them. Some of these solutions necessarily require integration with a minimal form of processing infrastructure, such as Wi-Fi. In contrast, others, such as Zigbee, can work in a standalone mode altogether, without the need for external processing and hardware support. These solutions are outlined in the subsequent sections in this chapter.

7.2 IEEE 802.15.4

The IEEE 802.15.4 standard represents the most popular standard for low data rate wireless personal area networks (WPAN) [1]. This standard was developed to enable monitoring and control applications with lower data rate and extend the operational life for uses with low-power consumption. This standard uses only the first two layers—physical and data link—for operation along with two new layers above it: 1) logical link control (LLC) and 2) service-specific convergence sublayer (SSCS). The additional layers help in the communication of the lower layers with the upper layers. Figure 7.1 shows the IEEE 802.15.4 operational layers. The IEEE 802.15.4 standard was curated to operate in the ISM (industrial, scientific, and medical) band.

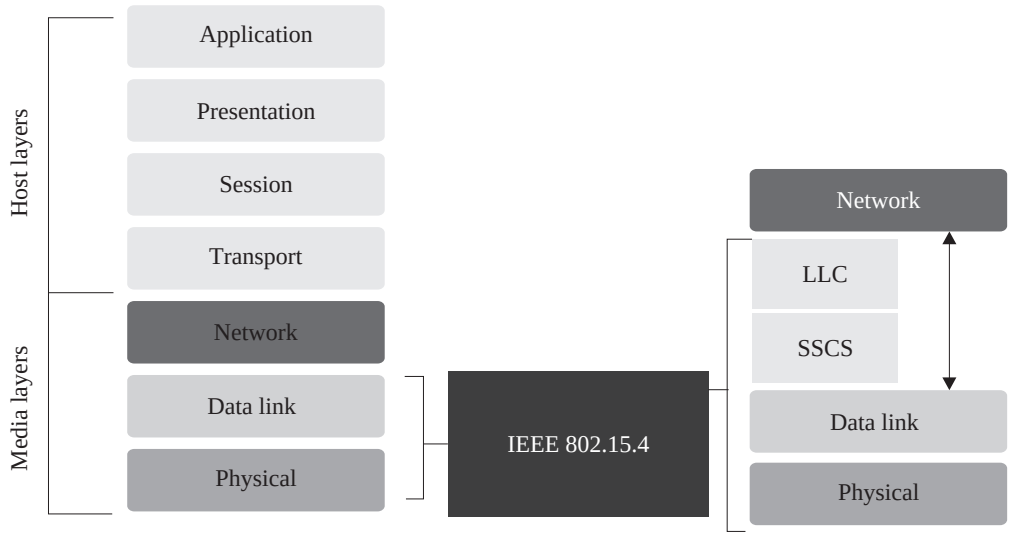


Figure 7.1 The operational part of IEEE 802.15.4's protocol stack in comparison to the OSI stack

The direct sequence spread spectrum (DSSS) modulation technique is used in IEEE 802.15.4 for communication purposes, enabling a wider bandwidth of operation with enhanced security by the modulating pseudo-random noise signal. This standard exhibits high tolerance to noise and interference and offers better measures for improving link reliability. Typically, the low-speed versions of the IEEE 802.15.4 standard use binary phase shift keying (BPSK), whereas the versions with high data rate implement offset quadrature phase shift keying (O-QPSK) for encoding the message to be communicated. Carrier sense multiple access with collision avoidance (CSMA-CA) is the channel access method used for maintaining the sequence of transmitted signals and preventing deadlocks due to multiple sources trying to access the same channel. Temporal multiplexing enables access to the same channel by multiple users or nodes at different times in a maximally interference-free manner.

The IEEE 802.15.4 standard [2] utilizes infrequently occurring and very short packet transmissions with a low duty cycle (typically, < 1%) to minimize the power consumption. The minimum power level defined is –3 dBm or 0.5 mW for the radios utilizing this standard. The transmission, for most cases, is line of sight (LOS), with the standard transmission range varying between 10 m to 75 m. The best-case transmission range achieved outdoors can be up to 1000 m.

This standard typically defines two networking topologies: 1) Star and 2) mesh. There are seven variants identified with in IEEE 802.15.4—A, B. C, D, E, F, and G. Variants A/B are the base versions, C is assigned for China, and D for Japan. Variants E, F, and G are assigned respectively for industrial applications, active RFID (radio frequency identification) uses, and smart utility systems.

The IEEE 802.15.4 standard supports two types of devices: 1) reduced function device (RFD) and 2) full function devices (FFD). FFDs can talk to all types of devices and support full protocol stacks. However, these devices are costly and energy-consuming due to increased requirements for support of full stacks. In contrast, RFDs can only talk to an FFD and have lower power consumption requirements due to minimal CPU/RAM requirements. Figure 7.2 shows the device types and network types supported by the IEEE 802.15.4 standard.

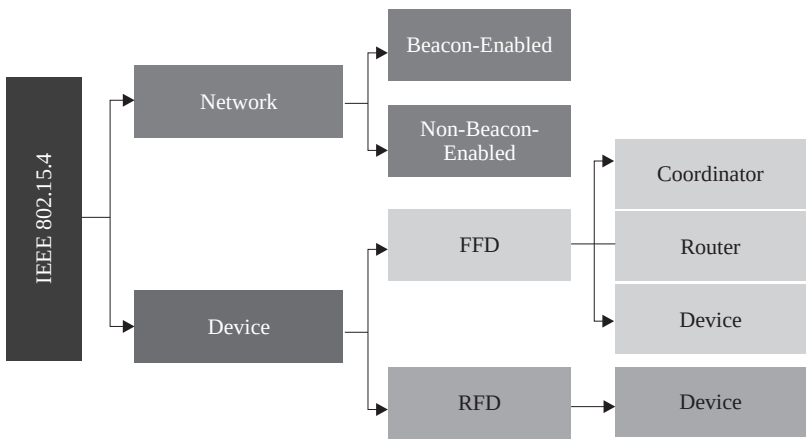


Figure 7.2 The various device and network types supported in the IEEE 802.15.4 standard

The IEEE 802.15.4 standard supports two network types: 1) Beacon-enabled networks and 2) non-beacon-enabled networks. The periodic transmission of beacon messages characterizes beacon-enabled networks. Here, the data frames sent via slotted CSMA/CA with a superframe structure managed by a personal area network (PAN) coordinator. These beacons are used for synchronization and association of other nodes with the coordinator. The scope of operation of this network type spans the whole network.

In contrast, for non-beacon-enabled networks, unslotted CSMA/CA (contention-based) is used for transmission of data frames, and beacons are used only for link

layer discovery. This network typically requires both source and destination IDs of the communicating nodes. As the IEEE 802.15.4 is primarily a mesh protocol, all protocol addressing must adhere to mesh configurations such that there is a decentralized communication amongst nodes.

Figure 7.3 shows the frame types associated with the IEEE 802.15.4 standard. Beacon frames are used for signaling and synchronization; data transmission is done through the data frames; and message reception is confirmed using the acknowledgment frames. MAC and command frames are used for association requests/responses, dissociation requests, data requests, beacon requests, coordinator realignment, and orphan notifications.

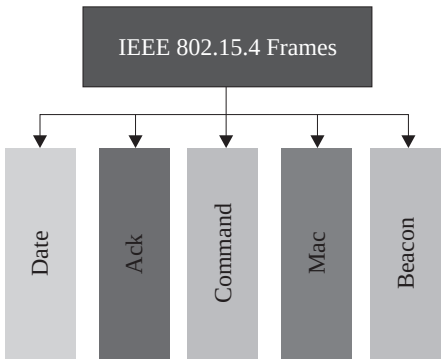


Figure 7.3 Various frame types supported in the IEEE 802.15.4 standard

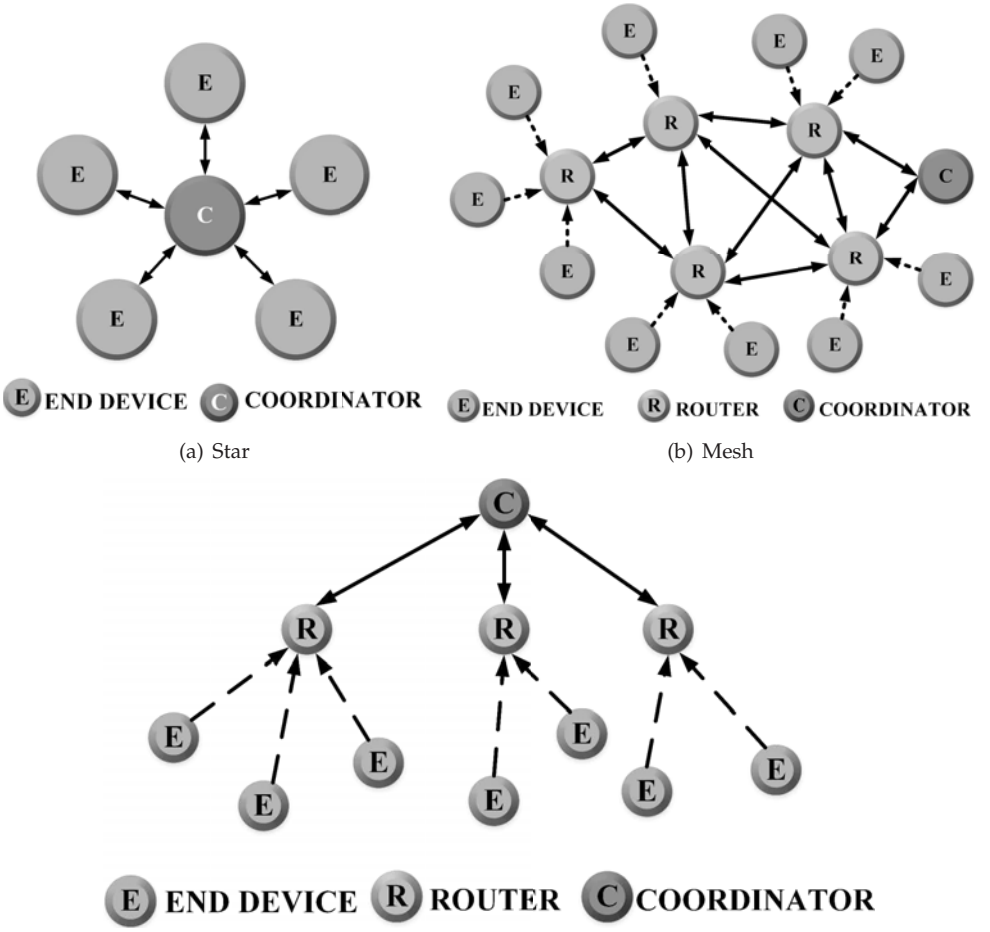
Check yourself

CSMA/CD versus CSMS/CA, transmission power, DSSS, BPSK, OQPSK

7.3 Zigbee

The Zigbee radio communication is designed for enabling wireless personal area networks (WPANs). It uses the IEEE 802.15.4 standard for defining its physical and medium access control (layers 1 and 2 of the OSI stack). Zigbee finds common usage in sensor and control networks [4]. It was designed for low-powered mesh networks at low cost, which can be broadly implemented for controlling and monitoring applications, typically in the range of 10–100 meters [3]. The PHY and MAC layers in this communication are designed to handle multiple low data rate operating devices. The frequencies of 2.4 GHz, 902–928 MHz or 868 MHz are commonly associated with Zigbee WPAN operations. The Zigbee commonly uses 250 kbps data rate which is optimal for both periodic and intermittent full-duplex data transmission between two Zigbee entities.

Zigbee supports various network configurations such as master-to-master communication or master-to-slave communication. Several network topologies are supported in Zigbee, namely the star (Figure 7.4(a)), mesh (Figure 7.4(b)), and cluster tree (Figure 7.4(c)). Any of the supported topologies may consist of a single or multiple coordinators. In star topology, a coordinator initiates and manages the other devices in the Zigbee network. The other devices which communicate with the coordinator are called end devices. As the star topology is easy to maintain and deploy, it finds widespread usage in applications where a single central controller manages multiple devices.



(a) Star

(b) Mesh

(c) Cluster tree

Figure 7.4 Various communication topologies in Zigbee

A network can be significantly extended in the Zigbee mesh and tree topologies by using multiple routers where the root of the topology is the coordinator. These configurations allow any Zigbee device or node to communicate with any other

adjacent node. In case of the failure of one or more nodes, the information is automatically forwarded to other devices through other functional devices. In a Zigbee cluster tree network, a coordinator is placed in the leaf node position of the cluster, which is, in turn, connected to a parent coordinator who initiates the entire network.

A typical Zigbee network structure can consist of three different device types, namely the Zigbee coordinator, router, and end device, as shown in Figure 7.4. Every Zigbee network has a minimum of one coordinator device type who acts as the root; it also functions as the network bridge. The coordinator performs data handling and storing operations. The Zigbee routers play the role of intermediate nodes that connect two or more Zigbee devices, which may be of the same or different types. Finally, the end devices have restricted functionality; communication is limited to the parent nodes. This reduced functionality enables them to have a lower power consumption requirement, allowing them to operate for an extended duration. There are provisions to operate Zigbee in different modes to save power and prolong the deployed network lifetime.

The PHY and MAC layers of the IEEE 802.15.4 standard are used to build the protocol for Zigbee architecture; the protocol is then accentuated by network and application layers designed especially for Zigbee. Figure 7.5 shows the Zigbee protocol stack. The various layer of the Zigbee stack are as follows.

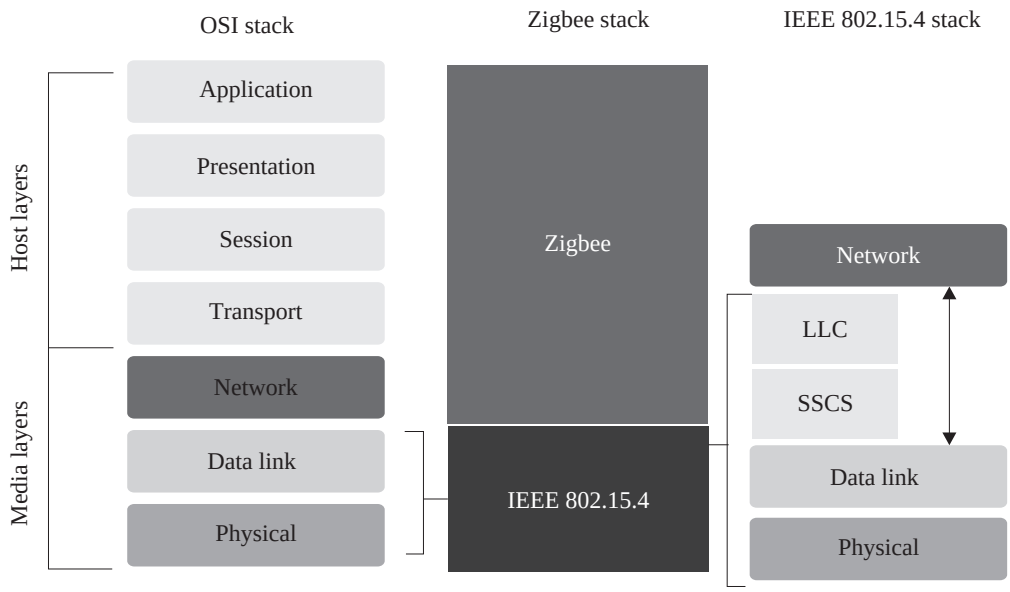


Figure 7.5 The Zigbee protocol stack in comparison to the OSI stack

- **Physical Layer**: This layer is tasked with transmitting and receiving signals, and performing modulation and demodulation operations on them, respectively. The

Zigbee physical layer consists of 3 bands made up of 27 channels: the 2.4 GHz band has 16 channels at 250 kbps the 868.3 MHz has one channel at 20 kbps; and the 902-928 MHz has ten channels at 40 kbps.

- **MAC Layer**: This layer ensures channel access and reliability of data transmission. CSMA-CA is used for channel access and intra-channel interference avoidance. This layer handles communication synchronization using beacon frames.
- **Network Layer**: This layer handles operations such as setting up the network, connecting and disconnecting the devices, configuring the devices, and routing.
- **Application Support Sub-Layer**: This layer handles the interfacing services, control services, bridge between network and other layers, and enables the necessary services to interface with the lower layers for Zigbee device object (ZDO) and Zigbee application objects (ZAO). This layer is primarily tasked with data management services and is responsible for service-based device matching.
- **Application Framework**: Two types of data services are provided by the application framework: provision of a key-value pair and generation of generic messages. A key-value pair is used for getting attributes within the application objects, whereas a generic message is a developer-defined structure.

Zigbee handles two-way data transfer using two operational modes: 1) Non-beacon mode and 2) beacon mode. As the coordinators and routers monitor the active state of the received data continuously in the non-beacon mode, it is more power-intensive. In this mode, there is no provision for the routers and coordinators to sleep. In contrast, a beacon mode allows the coordinators and routers to launch into a very low-power sleep state in the absence of data communication from end devices. The Zigbee coordinator is designed to periodically wake up and transmit beacons to the available routers in the network. These beacon networks are used when there is a need for lower duty cycles and more extended battery power consumption.

Check yourself

Ad-hoc networks, WASN, Zigbee ZDO, Zigbee APS

7.4 Thread

Thread is built upon the IEEE 802.15.4 radio standard; it is used for extremely low power consumption and low latency deployments [5]. Unlike Zigbee, Thread can extend direct Internet connectivity to the devices it is connected with. Thread removes the need for a mobile phone or a proprietary gateway to be in the range of devices for accessing the Internet. It is specially designed for IoT with the need for interoperability, security, power, and architecture addressed in a single radio platform.

Figure 7.6 shows the comparison of the Thread stack against the standard ISO-OSI stack. Thread is built on open standards to achieve a low-power wireless mesh

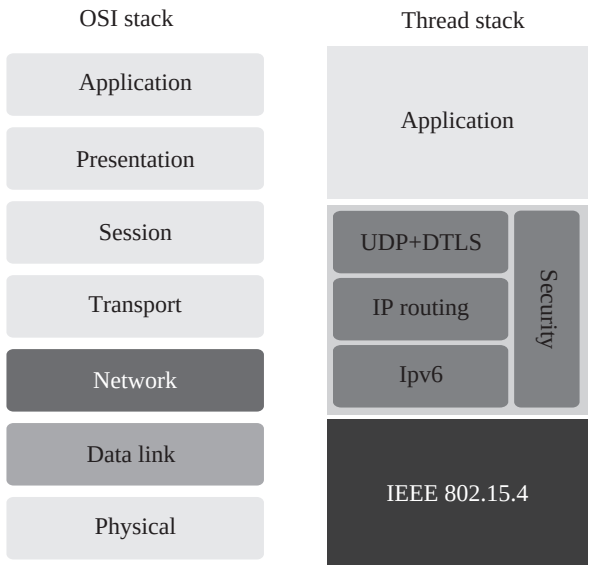


Figure 7.6 The functional protocol stack of Thread in comparison to the OSI stack

networking protocol with universal Internet Protocol (IP) support. The standard is easy to set up and simple to use; it can reliably connect thousands of devices to the Internet or a cloud with no single point of failure. It has the distinctive feature of self-healing and reconfiguration in the event of the addition or removal of a device. Figure 7.7 shows the Thread network architecture.

Thread enables IoT interoperability by utilizing a certification application that validates a device's conformance to the specification as well as its interoperability against multiple certified stacks. This feature ensures the resilience of connectivity, even with diverse networks, in turn enabling its users to have consistent operational experience.

Empowering low-power wireless devices with IP connectivity enables Thread to seamlessly accommodate itself with larger IP-based networks and be a robust option for most IoT applications such as smart homes/buildings, connected vehicles, and others. This feature of Thread devices removes the need for Internet-enabled proprietary gateways and cross-stack translators for connection between other technologies. The additional benefits of this feature include better resilience to single point of failures, highly economical deployments, less complex infrastructure, and enhanced IoT end-to-end device security on the Internet.

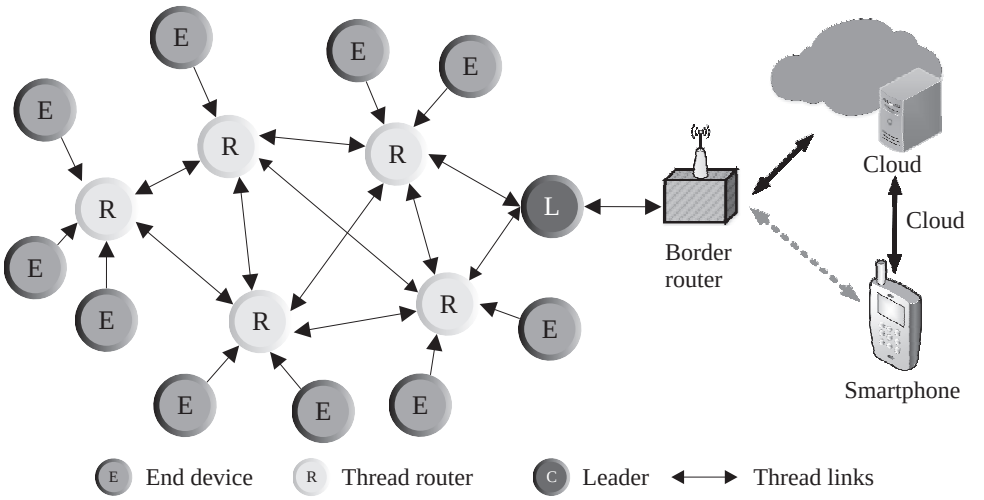


Figure 7.7 Outline of the Thread network architecture (from end devices to the cloud)

Thread devices can use common infrastructure similar to Wi-Fi networks and can connect directly to smartphones or computers if they are on the same IP network, without any additional setup for Thread.

Check yourself

IP-networks versus non-IP-networks, types of interoperability

7.5 ISA100.11A

The ISA100.11A is a very low power communication standard and has been developed and managed by ISA (International Society of Automation) [7]. Similar to the previous protocols, it uses the IEEE 802.15.4 standard as a base for building its protocol. The standard was mainly proposed for industrial plant automation systems. The ISA100.11A is characterized by an IoT compliant protocol stack, which can also be integrated with wired networks using Ethernet, support for open access protocols and device-level interoperability; it boasts of a 128-bit AES (Advanced Encryption Standard) encryption securing all communications. The security in ISA100.11A is in two layers: Transport layer and data link layer. ISA100.11A provides extensive support for IPv6 and UDP and uses TDMA (time-division multiple access)-based resource sharing with CSMA-CA. Both IPv6 and UDP as well as star topologies are supported by this standard. The utilization of IPv6 provides certain distinct benefits to ISA100.11A, such as increased address sizes, enhanced IPSec-based security measures, savings in network bandwidth by virtue of multicasting and auto address configuration.

An ISA100.11A wireless network utilizes the 2.4 GHz frequency band for communication, similar to Wi-Fi and Bluetooth. To avoid interference over wireless channels in the same band, it uses frequency hopping spread spectrum (FHSS) over a total of 16 channels. A definitive feature of this protocol is channel blacklisting, which blacklists the channels already in use by other protocols. This enables the protocol to perform even better by further achieving immunity from interference. Figure 7.8

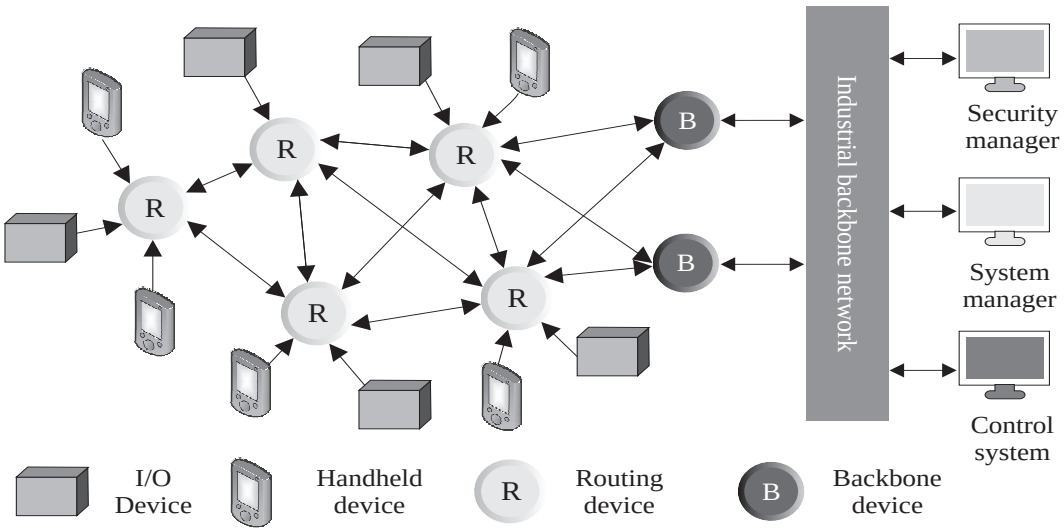


Figure 7.8 A typical ISA100.11A network architecture

shows the ISA100.11A network architecture. The ISA100.11A architecture consists of the following: 1) field devices and 2) backbone devices. Field devices may be non-routing I/O devices, handheld devices, routing devices, which may or may not be fixed or mobile. For industrial usage, the inclusion of portable and mobile devices is highly desirable as it allows floor supervisors and workers to keep checking various parts of the plant without the need for dedicated devices for each part. In contrast, backbone devices include backbone routers, gateways, the system manager, and the security manager, which are kept fixed and not portable. The ISA100.11A architecture provides support for mesh, star, and star–mesh topologies. The connected devices in ISA100.11A are collectively referred to as the downLink (DL) subnet. A wireless industrial sensor network (WISN) gateway connects the ISA100.11A network to the plant network.

The average ISA100.11A protocol stack consists of five different layers: 1) Application layer, 2) transport layer, 3) network layer, 4) data link layer, and 5) physical layer. Figure 7.9 compares the ISA100.11A stack with the standard ISO-OSI stack. A central system manager handles network routing by scheduling communication. The functionalities of the ISA100.11A protocol stack can be outlined as follows:

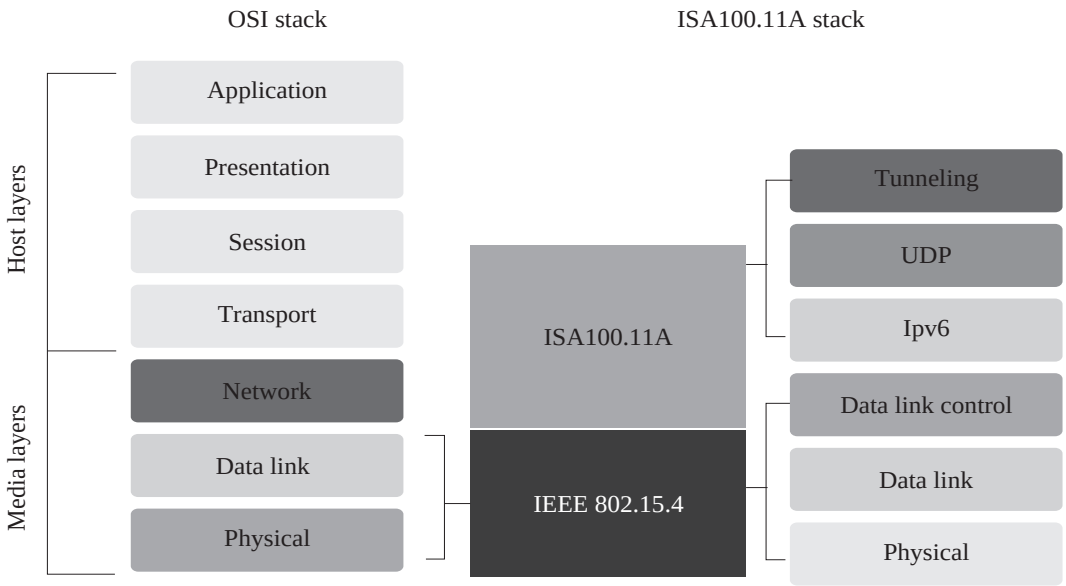


Figure 7.9 The ISA100.11A protocol stack in comparison to the OSI stack

- **Physical Layer**: It is built upon the IEEE 802.15.4-2006 standard. The physical layer communicates on the 2.4 GHz frequency band using a DSSS (direct sequence spread spectrum) modulation.
- **Data Link Layer**: It handles the creation, maintenance, and forwarding packet functionalities in addition to typical MAC functionalities. Additionally, it is responsible for operations dealing with the structure of the data packet, formation of the frame, detecting the error, and bus arbitration. A data link control (DLC) layer in ISA100.11A, which uses a graph-based routing, is responsible for specific distinctive functions such as adaptive channel hopping, detection and recovery of message loss, and clock synchronization.
- **Network Layer**: The ISA100.11A network layer is 6LoWPAN-compliant and uses IPv6 addressing for an end-to-end routing. Protocol conversion from IPv6 to 6LoWPAN and 6LoWPAN to IPv6 is executed at this layer by a router.
- **Transport Layer**: The ISA100.11A transport layer supports UDP-based connectionless services.
- **Application Layer**: The ISA100.11A stack only specifies system management application in this layer.

Check yourself

Wireless Industrial Sensor Networks (WISN), 6LoWPAN, field devices, routing and non-routing devices

7.6 WirelessHART

WirelessHART can be considered as the wireless evolution of the highway addressable remote transducer (HART) protocol [7]. It is a license-free protocol, which was developed for networking smart field devices in industrial environments. The lack of wires makes the adaptability of this protocol significantly advantageous over its predecessor, HART, in industrial settings. By virtue of its highly encrypted communication, wireless HART is very secure and has several advantages over traditional communication protocols. Similar to Zigbee, wirelessHART uses the IEEE 802.15.4 standard for its protocols designing.

Figure 7.10 shows the WirelessHART network architecture. WirelessHART can communicate with a central control system in any of the two ways: 1) Direct and 2) indirect. Direct communication is achieved when the devices transmit data directly to the gateway in a clear LOS (typically 250 m). Indirect communication is achieved between devices in a mesh and a gateway when messages jump from device to device until it reaches the gateway. WirelessHART communication is 99.999% reliable due to the maintenance of a tight schedule between message transmissions. All wirelessHART devices are back-compatible and allow for the integration of legacy devices as well as new ones.

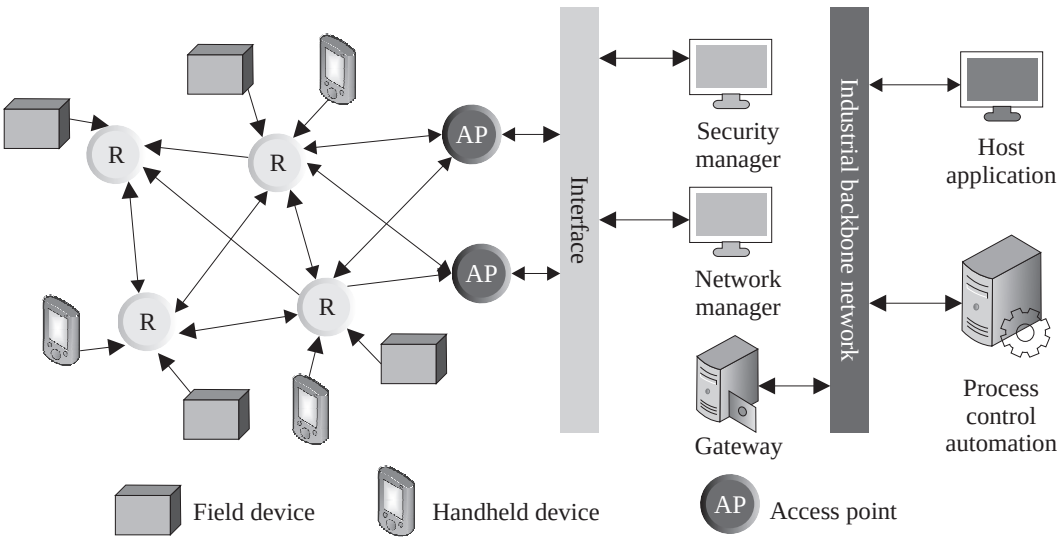


Figure 7.10 The WirelessHART network architecture

The HART encompasses the most number of field devices incorporated in any field network. WirelessHART makes device placements more accessible and cheaper, such as the top of a reaction tank, inside a pipe, or widely separated warehouses. The wired and unwired versions differ mainly in the network, data link, and physical layer. The wired HART lacks a network layer. HART ensures congestion control in the 2.4 Ghz ISM band by eliminating channel 26 because of its restricted usage in certain areas. The

use of interference-prone channels is avoided by using channel switching after every transmission. The transmissions are synchronized using 10 ms time-slots. During each time-slot, all available channels can be utilized by the various nodes in the network, allowing for the simultaneous propagation of 15 packets through the network, which also minimizes the risk of collisions between channels.

A network manager supervises each node in the network and guides them on when and where to send packets. This network manager allows for collision-free and timely delivery of packets between a source and the destination. It updates information regarding neighbors, signal strength, and information needing a delivery receipt. This network manager also decides which nodes transmit, which nodes listen, and the frequency to be utilized in each time-slot. It also handles code-based network security and prevents unauthorized nodes from joining the network.

Figure 7.11 shows the comparison of the wirelessHART protocol stack against the standard ISO-OSI stack. The various layers of the wirelessHART stack are outlined as follows:

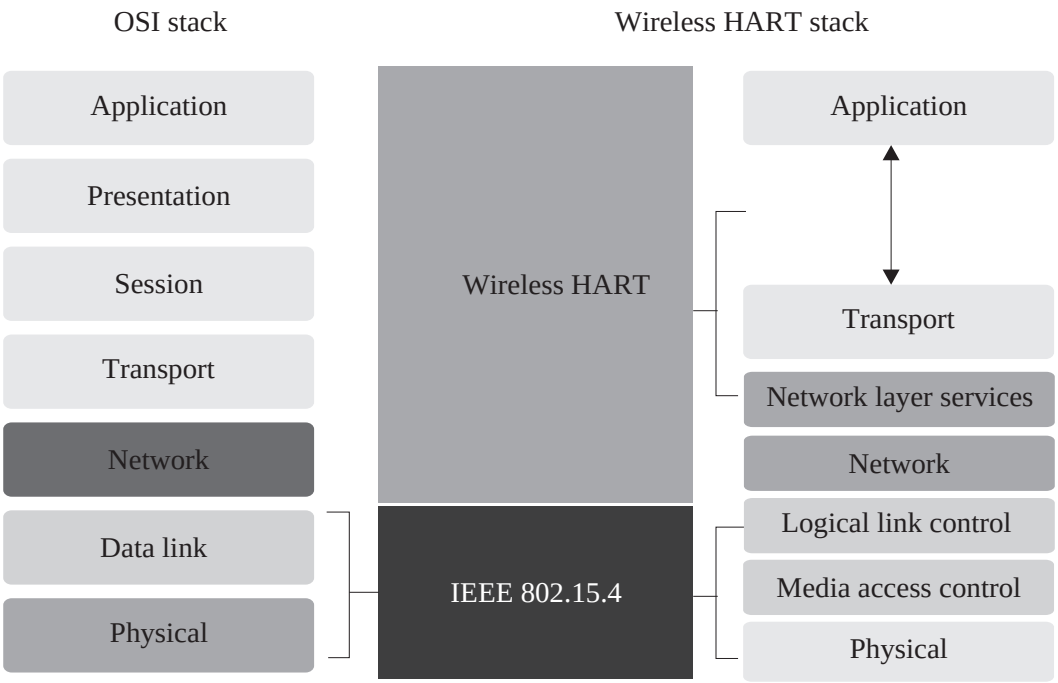


Figure 7.11 The WirelessHART protocol stack in comparison to the OSI stack

- **Physical Layer**: The IEEE 802.15.4 standard specification is used for designing the physical layer of this protocol. Its operation is limited to the use of the 2.4 GHz frequency band. The channel reliability is significantly increased by utilizing only 15 channels of the 2.4 GHz band.
- **Data Link Layer**: The data link layer avoids collisions by the use of TDMA. The communication is also made deterministic by the use of superframes.

WirelessHART superframes consist of 10 ms wide time-slots that are grouped together. The use of superframes ensures better controllability of the transmission timing, collision avoidance, and communication reliability. This layer incorporates channel hopping and channel blacklisting to increase reliability and security. A characteristic feature of the wirelessHART is channel blacklisting. This feature identifies channels consistently affected by interference and removes them from use.

- **Network and Transport Layers**: The network and the transport layer work in tandem to address issues of network traffic, security, session initiation/termination, and routing. WirelessHART is primarily a mesh-based network, where each node can accept data from other nodes in range and forward them to the next node. All the devices in its network have an updated network graph, which defines the routing paths to be taken. Functionally, the OSI stack's network, transport, and session layers constitute the WirelessHART's network layer.
- **Application Layer**: The application layer connects gateways and devices through various command and response messages. This layer enables back-compatibility with legacy HART devices as it does not differentiate between the wired and wireless versions of HART.

Check yourself

Graph-based routing, superframes, co-channel interference

7.7 RFID

RFID stands for radio frequency identification. This technology uses tags and readers for communication. RFID tags have data encoded onto them digitally [8]. The RFID readers can read the values encoded in these tags without physically touching them. RFIDs are functionally similar to barcodes as the data read from tags are stored in a database. However, RFID does not have to rely on line of sight operation, unlike barcodes.

The automatic identification and data capture (AIDC) technology can be considered as the precursor of RFID. Similar to AIDC techniques, RFID systems are capable of automatically categorizing objects. Categorization tasks such as identifying tags, reading data, and feeding the read data directly into computer systems through radio waves outline the operation of RFID systems. Typically, RFID systems are made up of three components: 1) RFID tag or smart label, 2) RFID reader, and 3) an antenna. Figure 7.12 shows the various RFID components.

In RFID, the tags consist of an integrated circuit and an antenna, enclosed in a protective casing to protect from wear and tear and environmental effects. These

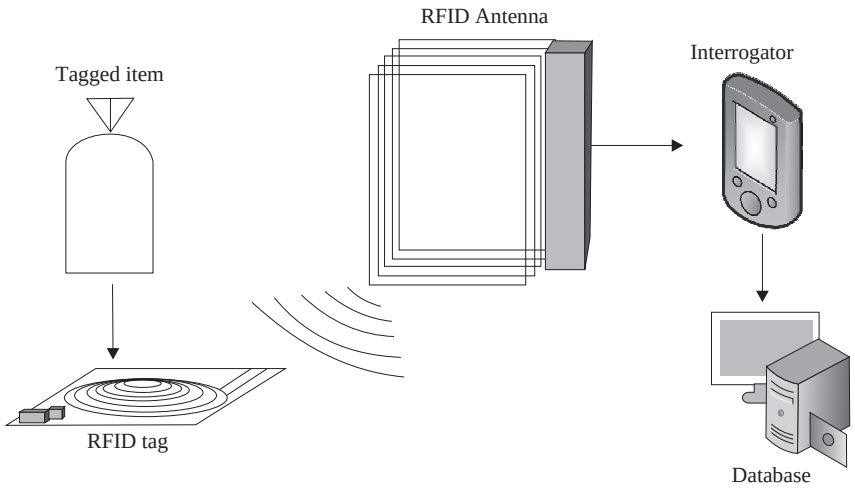


Figure 7.12 An outline of the RFID operation and communication

tags can be either active or passive. Passive tags find common usage in a variety of applications due to its low cost; however, it has to be powered using an RFID reader before data transmission. Active tags have their own power sources and do not need external activation by readers. Tags are used for transmitting the data to an RFID interrogator or an RFID reader. The radio waves are then converted to a more usable form of data by this reader. A host computer system accesses the collected data on the reader by a communication technology such as Wi-Fi or Ethernet. The data on the host system is finally updated onto a database. RFID applications span across domains such as inventory management, asset tracking, personnel tracking, and supply chain management.

Check yourself

How does RFID tackle various services, such as asset tracking and inventory management?

7.8 NFC

Near field communication (NFC) was jointly developed by Philips and Sony as a short-range wireless connectivity standard, enabling peer-to-peer (P2P) data exchange network. Communication between NFC devices is achieved by the principle of magnetic induction, whenever the devices are brought close to one another [9]. NFC can also be used with other wireless technologies such as Wi-Fi after establishing and configuring the P2P network. The communication between compatible devices requires a pair of transmitting and receiving devices. The typical NFC operating

frequency for data is 13.56 MHz, which supports data rates of 106, 212, or 424 kbps. NFC devices can be grouped into two types: 1) passive NFC and 2) active NFC. Figure 7.13 shows the various NFC types, components, and its usage.

A small electric current is emitted by the NFC reader, which creates a magnetic field that acts as a bridge in the physical space between two NFC devices. The generated EM (electromagnetic) field is converted back into electrical impulses through another coil on the client device. Data such as identifiers, messages, currency, status, and others can be transmitted using NFCs. NFC communication and pairing are speedy due to the use of inductive coupling and the absence of manual pairing.

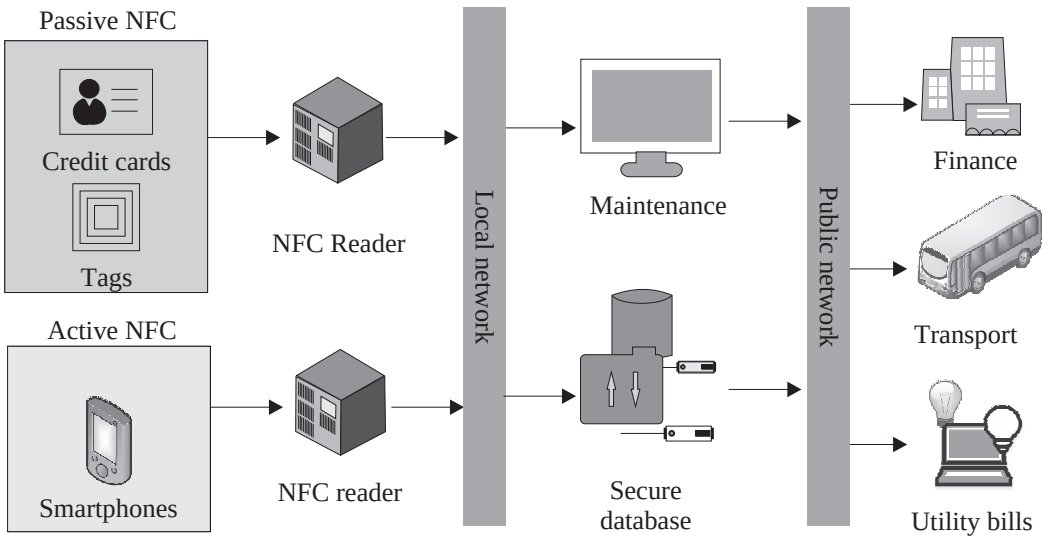


Figure 7.13 An outline of the NFC operation and communication

Passive NFC devices do not need a power source for communicating with the NFC reader. Tags and other small transmitters can act as passive NFC devices. However, passive devices cannot process information; they simply store information, which is read by an NFC reader. In contrast, active NFC devices can communicate with active as well as passive NFC devices. Active devices are capable of reading as well as writing data to other NFC terminals or devices. Some of the most commonly used NFC platforms are smartphones, public transport card readers, and commercial touch payment terminals.

NFC currently supports three information exchange modes: 1) peer-to-peer, 2) read/write, and 3) card emulation. The peer-to-peer mode is commonly used in NFC modes; it enables two NFC devices to exchange information. In the peer-to-peer mode of information exchange, the transmitting device goes active while the receiving device becomes passive. During the reverse transfer, both devices change roles. The read/write mode of information exchange allows only one-way data transmission. An active NFC device connects to a passive device to read information from it. Finally,

the card emulation mode enables an NFC device (generally, smartphones) to act as a contactless credit card and make payments using just a simple tap on an NFC reader.

Check yourself

Magnetic induction, inductive coupling, peer-to-peer data exchange

7.9 DASH7

The DASH7 protocol is based on an active RFID standard [10]. It operates in the 433 MHz frequency band and is being rapidly accepted in agriculture, vehicles, mobiles, and other consumer electronics-related applications. The messages in DASH7 are modulated using FSK (frequency shift keying) modulation before transmission over the 433 MHz frequency band. A very crucial aspect of DASH7 is its capability to use its 433.92 MHz operational band to enable communications with NFC devices. Recall, as the NFCs operate in the 13.56 MHz band, they can communicate with DASH7 radios by temporarily modifying/altering their antenna to access the higher-order harmonics of the DASH7 band (433.92/13.56 = 32 or 2^5). Figure 7.14 shows the DASH7 network architecture.

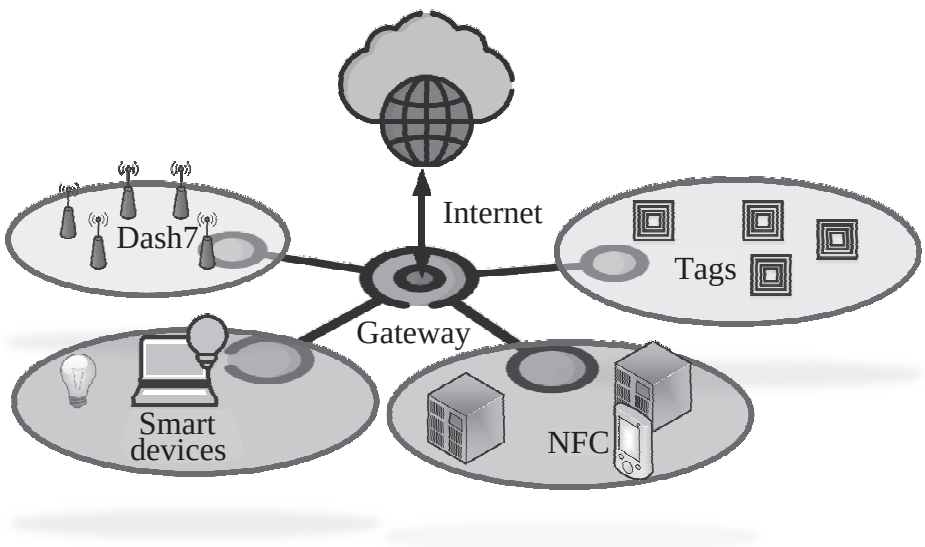


Figure 7.14 The DASH7 communication architecture

Compared to the IEEE 802.15.4 and its dependent technologies, the DASH7 protocol has a fully defined and complete OSI stack. This enables the DASH7 stack to be made adaptable to the physical layers of technologies such as Sigfox or LoRa. The

DASH7 stack includes support for cheap processing systems by virtue of its integrated file system. Figure 7.15 shows the protocol stack of DASH 7 in comparison to the ISO-OSI stack. DASH7 gateways can query devices in proximity to it without waiting for pre-defined time-slots to listen to end-device beacons.

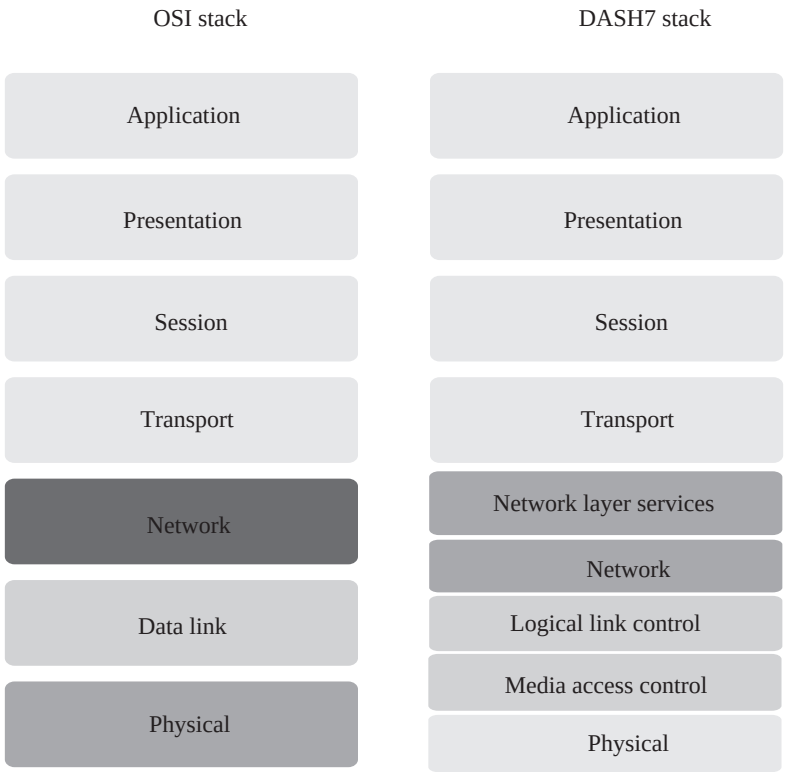


Figure 7.15 The DASH7 protocol stack in comparison to the OSI stack

DASH7 is capable of very dense deployments, has a low memory footprint, consumes minuscule power, and considered by many as a bridge between NFC and IoT communication systems. It can also be used to enable tag-to-tag communication without needing the tags to pass their information through a base station or a tag reader. This feature of DASH7 is quite synonymous with the multinode hopping mesh networks found in Zigbee and Z-wave. The reported range of DASH7 is between 1 to 10 km and a typical querying latency of 1 to 10 seconds.

Check yourself

File-system, node hopping mesh network, frequency harmonics

7.10 Z-Wave

Z-Wave is an economical and less complicated alternative to Zigbee. It was developed by Zensys, mainly for home automation solutions [11]. It boasts of a power consumption much lower than Wi-Fi, but with ranges greater than Bluetooth. This feature makes Z-Wave significantly useful for home IoT use by enabling inter-device communication between Z-wave integrated sensors, locks, home power distribution systems, appliances, and heating systems. Figure 7.16 shows the network architecture of the Z-Wave protocol.

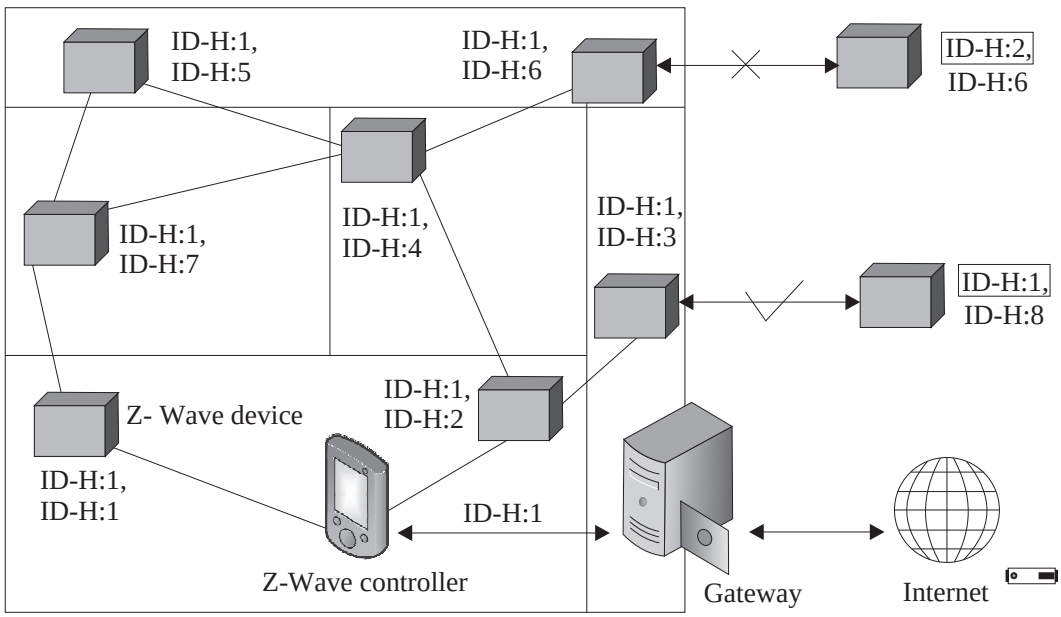


Figure 7.16 A typical Z-Wave deployment and communication architecture

Figure 7.17 shows the stack for this protocol. The Z-Wave operational frequency is in the range of 800–900 MHz, which makes it mostly immune to the interference effects of Wi-Fi and other radios utilizing the 2.4 GHz frequency band. Z-wave utilizes gaussian frequency shift keying (GFSK) modulation, where the baseband pulses are passed through a Gaussian filter before modulation. The filtering operation smoothens the pulses consisting of streams of –1 and 1 (known as pulse shaping), which limits the modulated spectrum's width. A Manchester channel encoding is applied for preparing the data for transmission over the channel.

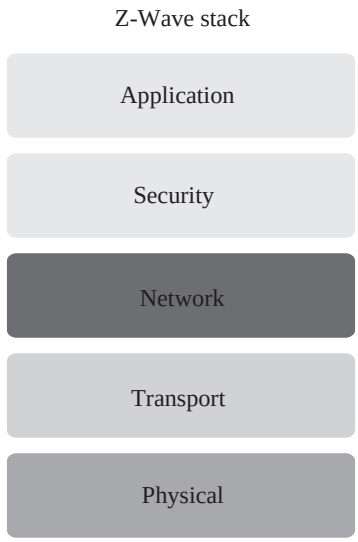


Figure 7.17 The Z-Wave protocol stack

Z-wave devices are mostly configured to connect to home-based routers and access points. These routers and access points are responsible for forwarding Z-wave messages to a central hub. Z-wave devices can also be configured to connect to the central hub directly if they are in range. Z-wave routing within the home follows a source-routed mesh network topology. When the Z-wave devices are not in range, messages are routed through different nodes to bypass obstructions created by household appliances or layouts. This process of avoiding radio dead-spots is done using a message called healing. Healing messages are a characteristic of Z-wave.

A central network controller device sets up and manages a Z-wave network (Figure 7.16), where each logical Z-wave network has one home (network) ID and multiple node IDs for the devices in it. Each network ID is 4 bytes long, whereas the node ID length is 1 byte. Z-Wave nodes with different home IDs cannot communicate with one another. The central hub is designed to be connected to the Internet, but their quantities are limited to one hub per home. Each home can have multiple devices, which can talk to the hub using Z-Wave. However, the devices themselves cannot connect to the Internet. The Z-wave can support 232 devices in a single home deployment (a single hub). This technology has been designed to be backward compatible. As Z-wave uses a source-routed static network, mobile devices are excluded from the network; only static devices are considered.

Check yourself

GFSK, Manchester encoding, pulse shaping principle

7.11 Weightless

Weightless is yet another emerging open standard for enabling networked communication in IoT; it is especially useful for low-power wide area networks [12]. It was designed for useful for low-power, low-throughput, and moderate to high latency applications supporting either or both public and private networks. The operating frequency of Weightless is restricted to sub-GHz bands, which are also exempted from the requirements of licensing such as 138 MHz, 433 MHz, 470 MHz, 780 MHz, 868 MHz, 915 MHz, and 923 MHz. Initially, three standards were released for Weightless: Weightless P, Weightless N, and Weightless W. Weightless P is the only currently accepted and used standard as it has features for bi-directional communication over both licensed as well as unlicensed ISM bands. Weightless N was designed as an LWPAN uplink-only technology, whereas Weightless W was designed to make use of the TV whitespace frequencies for communication.

As Weightless P was the most commonly adopted and accepted standard among the three Weightless standards, it came to be referred to merely as Weightless. Weightless provides a true bi-directional and reliable means of communication, where each message transaction is validated using an acknowledgment message. As it was designed initially for dense deployments of low-complexity IoT end devices, its payload size was limited to less than 48 bytes. Weightless networks can be optimized to attain ultra-low-power consumption status compared to cellular networks. However, this is at the cost of network latency and throughput with data rates in the range of 0.625 kbps to 100 kbps. Weightless has been identified with three architectural components: end devices, base stations, and base station network (Figure 7.18). The end devices (ED) form the leaf nodes in the Weightless network. These devices are typically low complexity and low cost. The duty cycle of EDs is also low, with a nominal transmiting power of 14 dBm (which can be increased up to 30 dBm).

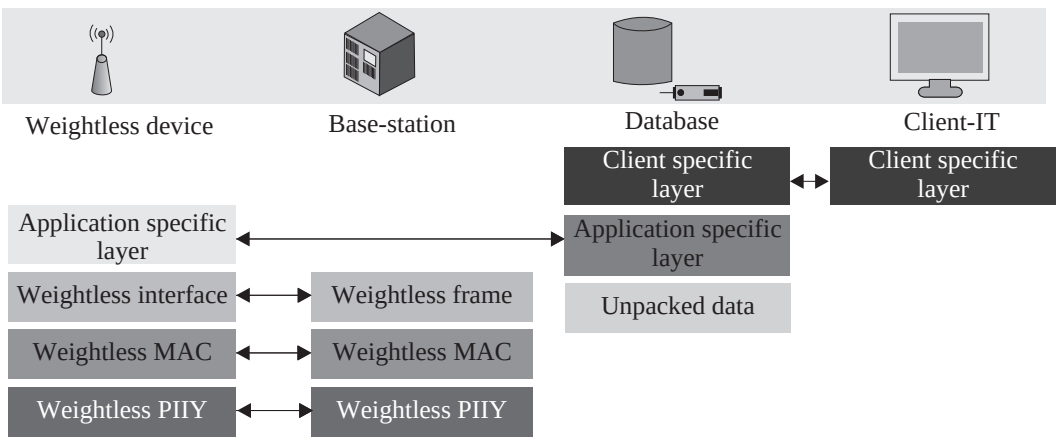


Figure 7.18 Typical components of the Weightless standard and its protocols

The base stations (BS) act as the central coordinating node in each cell. A star topology is deployed to connect the EDs to the BS. The transmit powers of a typical BS lie in the range of 27 dBm to 30 dBm. Finally, the base station network (BSN) is responsible for connecting all the BS of a single network. This enables the BSN to manage the allocation and scheduling of radio resources across the network. Additional tasks of the BSN include addressing authentication, roaming, and scheduling responsibilities.

Check yourself

Cellular architecture, base stations, whitespace bands

7.12 Sigfox

Sigfox is a low-power connectivity solution, which was developed for various businesses such as building automation and security, smart metering, agriculture, and others. It uses ultra-narrowband technology (192 kHz wide) for accessing and communicating through the radio spectrum [13]. The typical data rates achieved in Sigfox is in the range 100–600 bits per second. A binary phase shift keying (BPSK) is used for encoding the message transmission by changing the phase of the carrier waves, where each message is 100 Hz wide. Sigfox in Europe utilizes the 868 and 868.2 MHz spectrum, whereas it uses 902 and 928 MHz elsewhere. As the Sigfox receiver has to access only a very tiny part of the spectrum for receiving messages, the effects of noise are significantly reduced. It can even communicate in the presence of jamming signals, making this standard quite resilient.

Figure 7.19 shows the network architecture of Sigfox. Sigfox has an exciting message forwarding principle called random access, which ensures the high quality of services in this standard. Each Sigfox device emits a message at an arbitrary frequency;

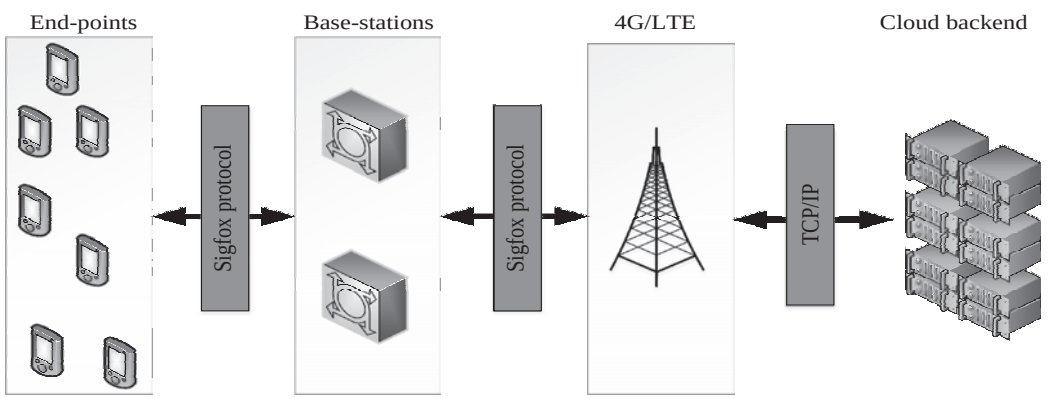


Figure 7.19 The Sigfox communication architecture

it simultaneously sends two replicas of the same message at different frequencies; it time using a principle known as time frequency diversity. Although the Sigfox devices are relatively less complicated, the base stations are very complicated as they monitor the whole 192 kHz spectrum looking for UNB (ultra narrow band) transmissions for demodulation. The base stations in Sigfox follow a cooperative reception principle. The messages in Sigfox are not attached to any base station, and any base station in the vicinity of the device can receive messages from it. This is called the principle of spatial diversity in Sigfox. The time and frequency diversity, along with the spatial diversity, ensures excellent quality of service for Sigfox.

Figure 7.20 shows the comparison of the Sigfox stack with the standard ISO-OSI stack. The Sigfox communication is bi-directional and asynchronous with a significant difference between the uplink and downlink speeds. As the devices are less complex than the base stations, the uplink budget (device to base station) is high compared to the downlink budget (base station to device). It is mainly due to this reason that the Sigfox was designed to have small message lengths ranging from 0 to 12 bytes. This 12-byte payload supports the simultaneous transfer of sensor data, the status of an event/alerts, GPS coordinates, and even application data. Sigfox boasts of excellent security features with support for authentication, integrity, and anti-replay on messages transmitted through the network. AES is supported by this standard. All

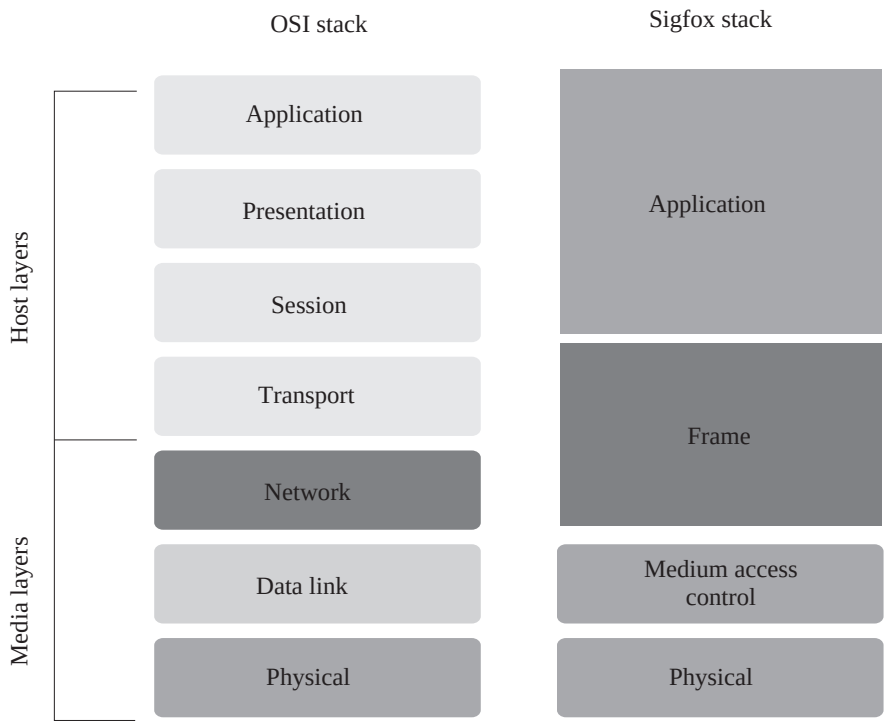


Figure 7.20 The Sigfox protocol stack in comparison to the OSI stack

these collective features of Sigfox enables it to be a low-power and resilient standard. However, due to the low data rates and asynchronous links, it is better utilized in applications requiring infrequent communication with small bursts of data. The Sigfox architecture and range supports wide and dense deployments depending on topologies and is better suited for indoor use; however, mobility is not an aspect associated with it.

Check yourself

AES, asynchronous versus synchronous communication

7.13 LoRa

LoRa or long range is a patented wireless technology for communication developed by Cycleo of Grenoble, France for cellular-type communications aimed at providing connectivity to M2M and IoT solutions [14]. It is a sub-GHz wireless technology that operationally uses the 169 MHz, 433 MHz, 868 MHz, and 915 MHz frequency bands for communication. LoRa uses bi-directional communication links symmetrically and a spread spectrum with a 125 kHz wideband for operating. Applications such as electric grid monitoring are typically suited for utilizing LoRa for communications. Typical communication of LoRa devices ranges from 15 to 20 km, with support for millions of devices. Figure 7.21 shows the LoRa network architecture.

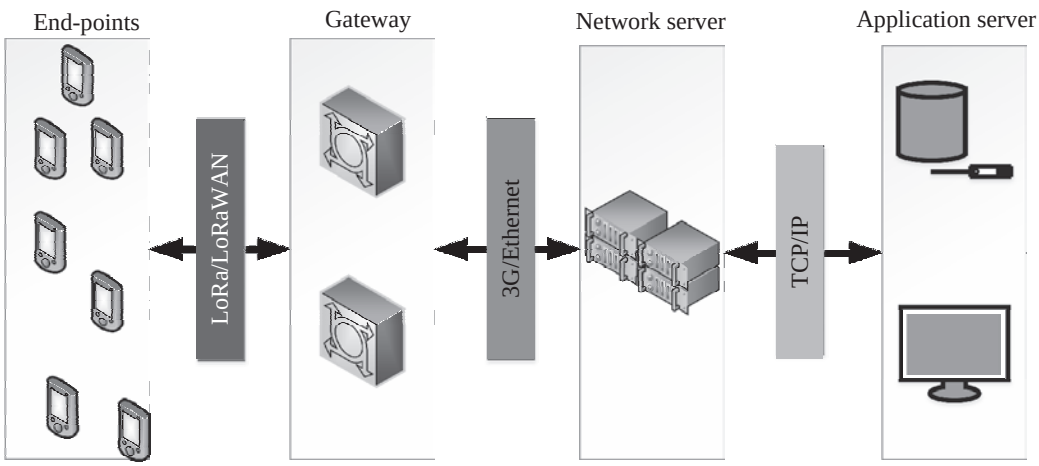


Figure 7.21 A typical LoRa deployment and communication architecture

It is a spread spectrum technology with a broader band (usually 125 kHz or more). LoRa achieves high receiver sensitivity by utilizing frequency-modulated chirp coding gain. LoRa devices provide excellent support for mobility, which makes them very

useful for applications such as asset tracking and asset management. In comparison with similar technologies such as NB-IoT, LoRa devices have significantly higher battery lives, but these devices have low data rates (27 to 50 kbps) and longer latency times. Figure 7.22 shows the LoRa protocol stack.

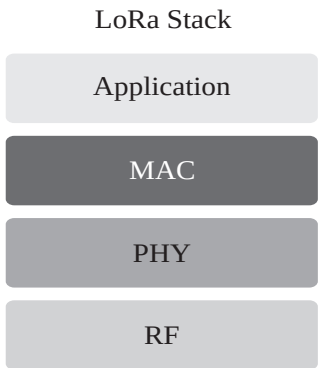


Figure 7.22 The LoRa protocol stack

LoRa devices make use of a network referred to as LoRaWAN, which enables the routing of messages between end nodes and the destination via a LoRaWAN gateway. Unlike Sigfox, LoRaWAN has a broader spectrum resulting in interference, which is solved using coding gains of the chirp signals. Additionally, unlike Sigfox, the LoRaWAN end nodes and the base stations are quite inexpensive. The LoRaWAN protocol is designed for WAN communications and is an architecture that makes use of LoRa, whereas LoRa is used as an enabling technology for a wide area network. Messages transmitted over LoRaWAN is received by all base stations in proximity to the device, which induces message redundancy in the network. However, this enhances the resilience of the network by ensuring more messages are successfully delivered between entities in the network.

A LoRa network follows the star topology and is made up of four crucial entities: end points/nodes, gateways, network server, and a remote computer (Figure 7.21). The end nodes deal with all the sensing and control solutions. The gateways forward messages from end nodes to a backhaul network. The LoRa network can comprise both or either of wired and wireless technologies. The gateways themselves are connected to the network server utilizing IP-based connections (either private or public). The LoRa network server is responsible for scheduling message acknowledgments, modifying data rates, and removing message redundancies. Finally, the remote computers have control over the end nodes and act as data sinks for data originating from these nodes.

The LoRa network security is achieved through various mechanisms such as unique network key, which ensures security on the network level, unique application key, which ensures an end-to-end security on the application level and device specific key.

Check yourself

Features of a chirp signal, coding gains, spread spectrum technology

7.14 NB-IoT

NB-IoT or narrowband IoT is an initiative by the Third Generation Partnership Project (3GPP) to develop a cellular standard, which can coexist with cellular systems (2G/3G/4G), be highly interoperable and that too using minimum power [15]. It is reported that a major portion of the NB-IoT applications can support a battery life of up to ten years. NB-IoT also boasts of significant improvements in reliability, spectrum efficiencies, and system capacities. NB-IoT uses orthogonal frequency division multiplexing (OFDM) modulation, which enhances the system capacity and increases spectrum efficiency (Figure 7.23). However, device complexities are quite high. NB-IoT also provides support for security features such as confidentiality, authentication, and integrity. Figure 7.24 shows the protocol stacks of the various components of NB-IoT.

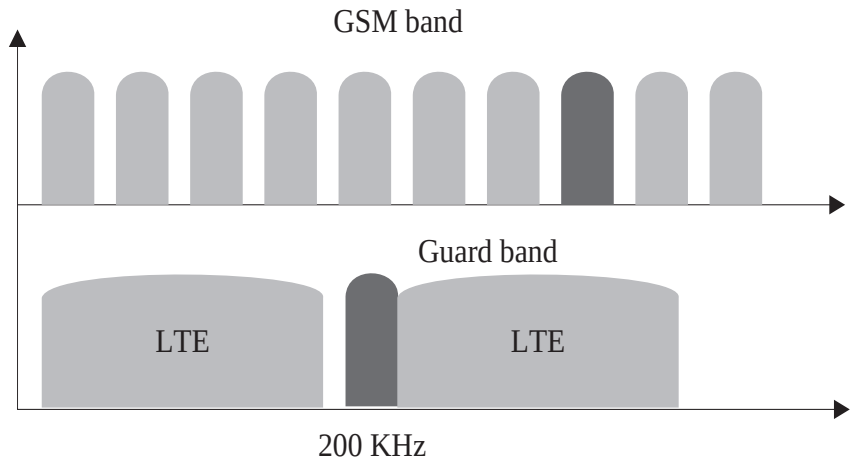


Figure 7.23 A location of NB-IoT band within the LTE spectrum

The coverage of NB-IoT supports deployments in indoor environments as well as in dense urban areas. When compared with technologies such as LoRa, NB-IoT ensures a higher quality of service as well as reduced latencies. Because of its design principles, the transfer of large messages is not efficient. NB-IoT is better suited for static deployments such as energy metering, fixed sensors, and others. Mobility support is not provided in this standard. NB-IoT communication can either make use of the available 200-kHz GSM (global system for mobile communications) bands or be allocated resource blocks on the guard bands by LTE base stations. This ensures

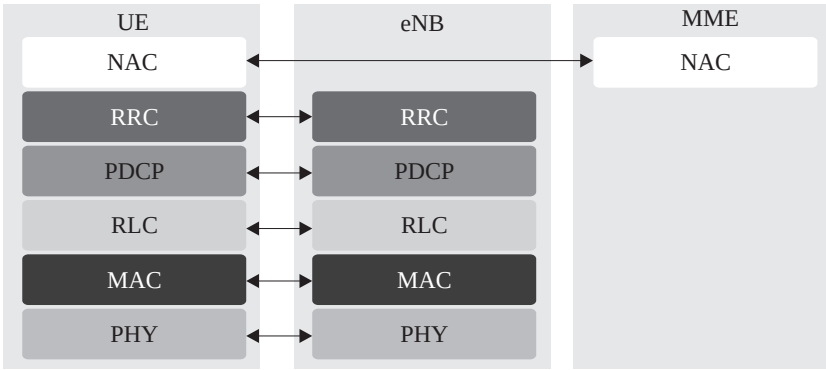


Figure 7.24 The NB-IoT protocol stack with respect to its entities

that the NB-IoT can achieve more extensive coverage while coexisting with cellular systems.

NB-IoT was developed for non-IP based applications requiring quite small volumes of daily data transactions, typically in the range of a few tens to a hundred bytes of data per device daily. Unlike technologies such as Sigfox and LoRa, the use of OFDM (orthogonal frequency division multiplexing's) faster modulation rates ensures higher data handling capacities for NB-IoT.

Check yourself

OFDM, LTE, guard band, GSM

7.15 Wi-Fi

Wi-Fi or WiFi is technically referred to by its standard, IEEE 802.11, and is a wireless technology for wireless local area networking of nodes and devices built upon similar standards (Figure 7.25). Wi-Fi utilizes the 2.4 GHz ultra high frequency (UHF) band or the 5.8 GHz super high frequency (SHF) ISM radio bands for communication [16]. For operation, these bands in Wi-Fi are subdivided into multiple channels. The communication over each of these channels is achieved by multiple devices simultaneously using time-sharing based TDMA multiplexing. It uses CSMA/CA for channel access.

Various versions of IEEE 802.11 have been popularly adapted, such as a/b/g/n. The IEEE 802.11a achieves a data rate of 54 Mbps and works on the 5 GHz band using OFDM for communication. IEEE 802.11b achieves a data rate of 11 Mbps and operates on the 2.4 GHz band. Similarly, IEEE 802.11g also works on the 2.4 GHz band but achieves higher data rates of 54 Mbps using OFDM. Finally, the newest version, IEEE 802.11n, can transmit data at a rate of 140 Mbps on the 5 GHz band.

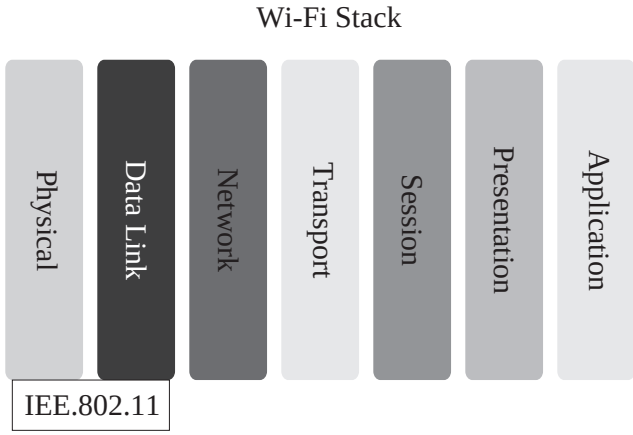


Figure 7.25 The IEEE 802.11 Wi-Fi stack

Wi-Fi devices can network using a technology referred to as wireless LAN (WLAN), as shown in Figure 7.26. A Wi-Fi enabled device has to connect to a wireless access point, which connects the device to the WLAN. WLAN is then responsible for forwarding the messages from the devices to and fro between the devices and the Internet.

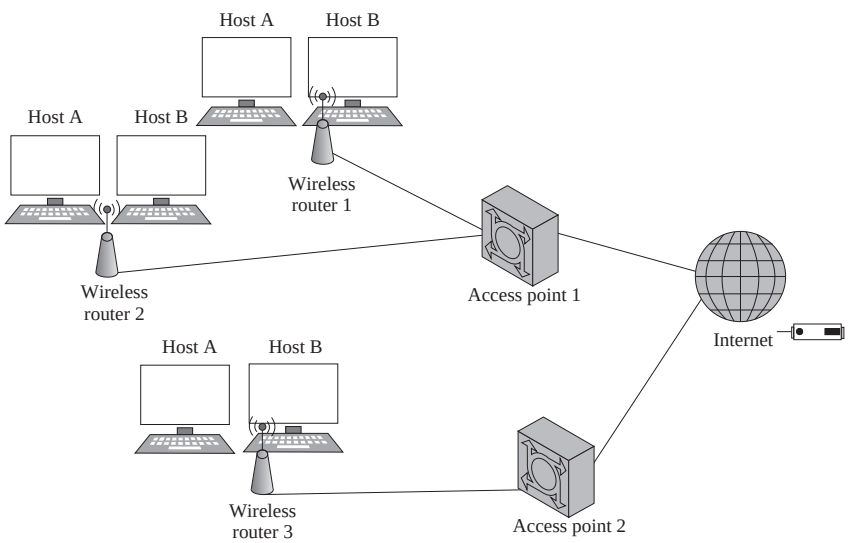


Figure 7.26 The Wi-Fi deployment architecture

Check yourself

TDMA

7.16 Bluetooth

Bluetooth is defined by the IEEE 802.15.1 standard and is a short-range wireless communication technology operating at low power to enable communication among two or more Bluetooth-enabled devices [17]. It was initially developed as a cable replacement technology for data communication between two or more mobile devices such as smartphones and laptops. This standard allows the transmission of data as well as voice-over short distances. Bluetooth functions on the 2.4 GHz ISM band and has a range of approximately 10 m. The transmission of data is done through frequency hopping spread spectrum (FHSS), which also reduces the interference caused by other devices functioning in the 2.4 GHz band. The data is divided into packets before transmitting them by Bluetooth. The packets are transmitted over the 79 designated channels, each 1MHz wide in the 2.4 GHz band. Adaptive frequency hopping (AFH) enables this standard to perform 800 hops per second over these channels. Initial versions of this standard followed Gaussian frequency shift keying (GFSK) modulation, which was known as the basic rate (BR) mode, and was capable of data rates of up to 1 Mbps. However, with the development of newer variants, modulation schemes such as $\pi/4$ DQPSK (differential quadrature phase shift keying) and 8-DPSK (differential phase shift keying) were adopted, which enabled data rates of 2 Mbps and 3 Mbps respectively.

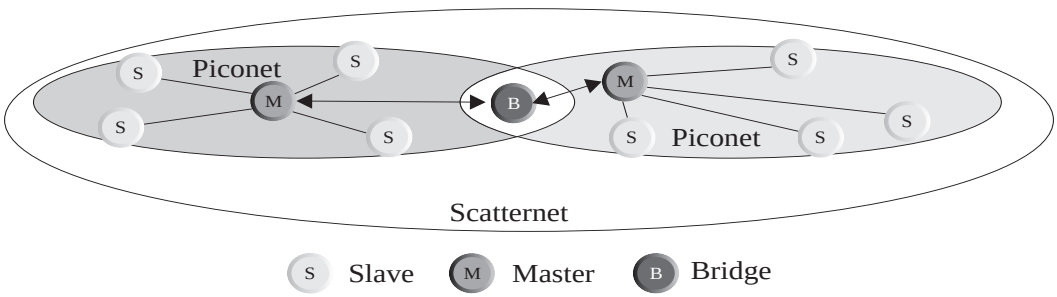


Figure 7.27 The Bluetooth device network architecture

Bluetooth follows a master–slave architecture (Figure 7.27). It enables a small network, which can accommodate seven slave devices simultaneously with a single master node. A slave node in one piconet cannot be part of another piconet at the same time, that is, it can have a single master node at a time. This network is known as a personal area network (PAN) or piconet. All the devices in a piconet share the master node's clock. Two piconets can be joined using a bridge. The whole network is also referred to as a scatternet.

Bluetooth Low Energy (BLE), the advanced variant of Bluetooth has 2 MHz wide bands, which can accommodate 40 channels. Its features include low energy consumption, low cost, multivendor interoperability, and an enhanced range of operations.

Bluetooth connections are encrypted and prevent eavesdropping of communications between devices. The inclusion of service-level security adds an additional layer of security by restricting the usage and device features and activities.

The Bluetooth standard consists of four parts: 1) core protocols, 2) cable replacement protocols, 3) telephony control protocols, and 4) adopted protocols. Figure 7.28 shows the Bluetooth protocol stack. Link Manager Protocol (LMP), Logical Link Control and Adaptation Protocol (L2CAP), Host Controller Interface (HCI), Radio Frequency Communications (RFCOMM), and Service Discovery Protocol (SDP) are some of the well-known protocols associated with Bluetooth. These protocols can be enumerated as follows:

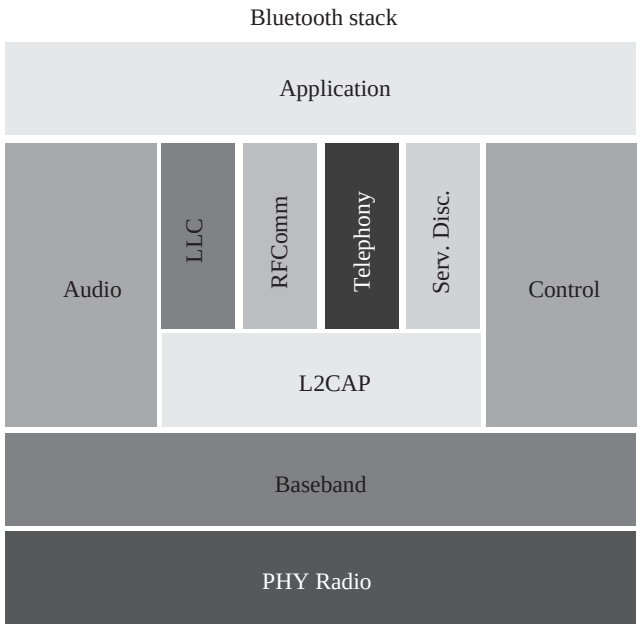


Figure 7.28 The Bluetooth protocol stack

(i) **Link Manager Protocol:** It manages the establishment, authentication, and links configuration. LMPs consist of some protocol data units (PDU), between which transmission occurs for availing services such as name requests, link address requests, connection establishment, connection authentication, mode negotiation, and data transfer.

(ii) **Host Controller Interface:** It enables access to hardware status and control registers and connects the controller with the link manager. The automatic discovery of Bluetooth devices in its proximity is one of the essential tasks of HCI.

(iii) **L2CAP:** It multiplexes logical connections between two devices. It is also tasked with data segmentation, flow control, and data integrity checks.

(iv) **Service Discovery Protocol:** It is tasked with the discovery of services provided by other Bluetooth devices.

(v) **Radio Frequency Communications:** It is a cable replacement protocol, which generates a virtual stream of serial data. This protocol supports many telephony related profiles as AT commands and Object Exchange Protocol (OBEX) over Bluetooth.

(vi) **Telephony Control Protocol – Binary (TCS BIN):** It is a bit-oriented protocol to control call signaling prior to initiation of voice or data communications between devices.

Check yourself

Bluetooth paging and inquiry, $\pi/4$ DQPSK, 8-DPSK, AFH

Summary

This chapter covered the various IoT connectivity technologies and their functional requirements and focuses on those technologies, which primarily rely on wireless media for communications. The standards covered in this chapter are a mix of the ones used for general consumer electronics, household devices, as well as speciality applications such as industries. The connectivity technologies covered here range from near-field ones to long-range ones. After going through this chapter, readers will be able to select various connectivity technologies, which will be suitable for an IoT application or architecture under consideration.

Exercises

(i) What is a piconet?

(ii) What is a scatternet? Explain the working of a scatternet with a brief description of its various members.

(iii) Describe the protocol stack of Bluetooth.

(iv) What is BLE?

(v) Differentiate between class 1, 2, and 3 Bluetooth devices.

(vi) What are the various modes of operation of Bluetooth?

(vii) Describe the L2CAP layer in Bluetooth.

(viii) Describe the RFCOMM layer in Bluetooth.

(ix) What is service discovery protocol (SDP) in Bluetooth?

(x) Describe the Bluetooth baseband.

(xi) How does Bluetooth avoid collisions between simultaneously transmitting nodes?

(xii) Explain the protocol stack of Zigbee.

(xiii) What is ZDO? How is it different from APS?

(xiv) Elaborate on the various network topologies of Zigbee.

(xv) What are the various Zigbee device types?

(xvi) Describe the Zigbee network layer.

(xvii) What is AODV? Explain with an example.

(xviii) How is Zigbee different from Bluetooth?

(xix) How is Zigbee different from 6LoWPAN?

(xx) Explain the protocol stack of IEEE 802.15.4

(xxi) How is LWPAN different from PANs?

(xxii) Explain the terms:

(a) DSSS

(b) BPSK

(c) QPSK

(d) O-QPSK

(xxiii) Differentiate between CSMA/CA and CSMA/CD.

(xxiv) Differentiate between star and mesh network topologies.

(xxv) What are the various IEEE 802.15.4 network types?

(xxvi) Differentiate between RFD and FFD.

(xxvii) Differentiate between a PAN coordinator, router, and a device in IEEE 802.15.4.

(xxviii) What are the various IEEE 802.15.4 frame types?

(xxix) What is beaconing?

(xxx) How are beacon-enabled networks different from non-Beacon enabled networks?

(xxxi) What is HART? How is it different from wirelessHART?

(xxxii) Describe the protocol stack of HART.

(xxxiii) Describe the HART physical layer.

(xxxiv) Describe the HART data link layer.

(xxxv) Describe the HART network and transport layers.

(xxxvi) What is TDMA? Describe with an example.

(xxxvii) What is channel blacklisting?

(xxxviii) What are superframes?

(xxxix) Describe the HART congestion control mechanism.

(xl) Describe the working of the wirelessHART network manager.

(xli) How is wirelessHART different from Zigbee?

(xlii) What is RFID? Explain its working.

(xliii) How is RFID different from QR codes?

(xliv) Differentiate between active and passive RFID.

(xlv) List some of the typical applications of RFID.

(xlvi) What is NFC? Describe its working.

(xlvii) How is NFC different from RFID?

(xlviii) What are the different types of NFC? Explain in detail.

(xlix) Describe the various modes of operation of NFC.

(l) List some of the popular applications of NFC.

(li) What is ISA 100.11a?

(lii) Describe the various transport services in ISA100.11a.

(liii) What are the various networks permitted in ISA100.11a?

(liv) What network topologies are allowed in ISA100.11a?

(lv) What are the various device types in ISA100.11a?

(lvi) List the salient features of ISA100.11a.

(lvii) What are the security features of ISA100.11a?

(lviii) Differentiate between an NRD and backbone device in ISA100.11a.

(lix) Differentiate between an RD and an NRD in ISA100.11a.

(lx) What are the typical usage classes in ISA100.11a?

(lxi) What is Z-Wave?

(lxii) Describe the working of a Z-Wave implementation.

(lxiii) Describe GFSK.

(lxiv) What is Manchester encoding?

(lxv) What is healing in the context of Z-Wave?

(lxvi) Differentiate between Z-Wave and Zigbee.

(lxvii) What are the different variants of Weightless? Enumerate the highlighting features of each.

(lxviii) How does Weightless provide true bi-directional communication?
(lxix) In Weightless, what topology is deployed to connect the EDs to the BS?
(lxx) What is the typical payload size restriction of Weightless?
(lxxi) What are the typical application domains of Sigfox?
(lxxii) What are the general data rates associated with Sigfox?
(lxxiii) Which encoding is used in Sigfox for transmitting messages?
(lxxiv) How does Sigfox communicate even in the presence of jamming signals?
(lxxv) What is the principle of spatial diversity in Sigfox?
(lxxvi) Why is the Sigfox uplink budget different from its downlink budget?
(lxxvii) What frequency bands are typically associated with LoRa?
(lxxvii) Differentiate between LoRa and NB-IoT.
(lxxix) How is the spread spectrum used for enhancing the efficiency of LoRa?
(lxxx) What is LoRaWAN? How is it different from LoRa?
(lxxxi) Differentiate between LoRaWAN and Sigfox.
(lxxxii) Describe the network topology of LoRa.
(lxxxiii) What are the modes of existence of NB-IoT?
(lxxxiv) How does NB-IoT make use of existing redundant GSM/CDMA bands?
(lxxxv) How does NB-IoT ensure high data handling capacities?
(lxxxvi) How does IEEE 802.11g achieve higher data rates?
(lxxxvii) What is the typical data transmission rate of IEEE 802.11n?
(lxxxviii) What is WLAN?
(lxxxix) Differentiate between WiFi and Bluetooth.
(xc) Differentiate between WiFI and Zigbee.

References

[1] Sikora, A. and V. F. Groza. 2005. "Coexistence of IEEE 802. 15.4 with Other Systems in the 2.4 GHz-ISM-Band." In 2005 *IEEE Instrumentation and Measurement Technology Conference Proceedings* 3: 1786–1791. IEEE.

[2] 802.15.4-2015 - IEEE Standard for Low-Rate Wireless Networks, IEEE. https://standards.ieee.org/standard/802_15_4-2015.html.

[3] Zigbee Alliance. https://zigbee.org/zigbee-for-developers/zigbee-3-0/.

[4] Farahani, S. 2011. *ZigBee Wireless Networks and Transceivers*. Newnes.

[5] Thread. https://www.threadgroup.org/.

[6] OpenThread: An Open Foundation for the Connected Home. https://openthread.io/.

[7] Petersen, S. and S. Carlsen. 2011. "WirelessHART vs. ISA100. 11a: The Format War Hits the Factory Floor." *IEEE Industrial Electronics Magazine* 5(4): 23–34.

[8] Garfinkel, S. and H. Holtzman. 2006. "Understanding RFID Technology." *RFID*, pp. 15–36.

[9] Coskun, V., B. Ozdenizci, and K. Ok. 2013. "A Survey on Near Field Communication (NFC) Technology." *Wireless Personal Communications* 71(3): 2259–2294.

[10] Weyn, M., G. Ergeerts, L. Wante, C. Vercauteren, and P. Hellinckx. 2013. "Survey of the DASH7 Alliance Protocol for 433 MHz Wireless Sensor Communication." *International Journal of Distributed Sensor Networks* 9(12): 870430.

[11] Badenhop, C. W., S. R. Graham, B. W. Ramsey, B. E. Mullins, and L. O. Mailloux. 2017. "The Z-Wave Routing Protocol and its Security Implications." *Computers & Security* 68: 112–129.

[12] Weightless specification. http://www.weightless.org/about/weightless-specification.

[13] Lauridsen, M., H. Nguyen, B. Vejlgaard, I. Z. Kovács, P. Mogensen, and M. Sorensen. 2017. "Coverage Comparison of GPRS, NB-IoT, LoRa, and SigFox in a 7800 km^2 Area." In *Proceedings of IEEE 85th Vehicular Technology Conference* (VTC Spring) (pp. 1–5). IEEE.

[14] Augustin, A., J. Yi, T. Clausen, and W. Townsley. 2016. "A Study of LoRa: Long Range & Low Power Networks for the Internet of Things." *Sensors* 16(9): 1466.

[15] Sinha, R. S., Y. Wei, and S. H. Hwang. 2017. "A Survey on LPWA Technology: LoRa and NB-IoT." *Ict Express* 3(1): 14–21.

[16] IEEE 802.11TM WIRELESS LOCAL AREA NETWORKS – The Working Group for WLAN Standards. http://www.ieee802.org/11/.

[17] Bluetooth specifications. https://www.bluetooth.com/specifications/.

Chapter 8

IoT Communication Technologies

Learning Outcomes

After reading this chapter, the reader will be able to:

- List common communication protocols in IoT
- Identify the salient features and application scope of each communication protocol
- Understand the terminologies and technologies in IoT communication
- Determine the requirements associated with each of these communication protocols in real-world solutions
- Determine the most appropriate communication protocol for their IoT implementation

8.1 Introduction

Having covered the various connectivity technologies for IoT in the previous chapter, this chapter specifically focuses on the various intangible technologies that enable communication between the IoT devices, networks, and remote infrastructures. We organize the various IoT communication protocols according to their usage into six groups: 1) Infrastructure protocols, 2) discovery protocols, 3) data protocols, 4) identification protocols, 5) device management protocols, and 6) semantic protocols. These protocols are designed to enable one or more of the functionalities and features associated with various IoT networks and implementations such as routing, data management, event handling, identification, remote management, and interoperability. Figure 8.1 outlines the distribution of these IoT communication protocol groups [3].

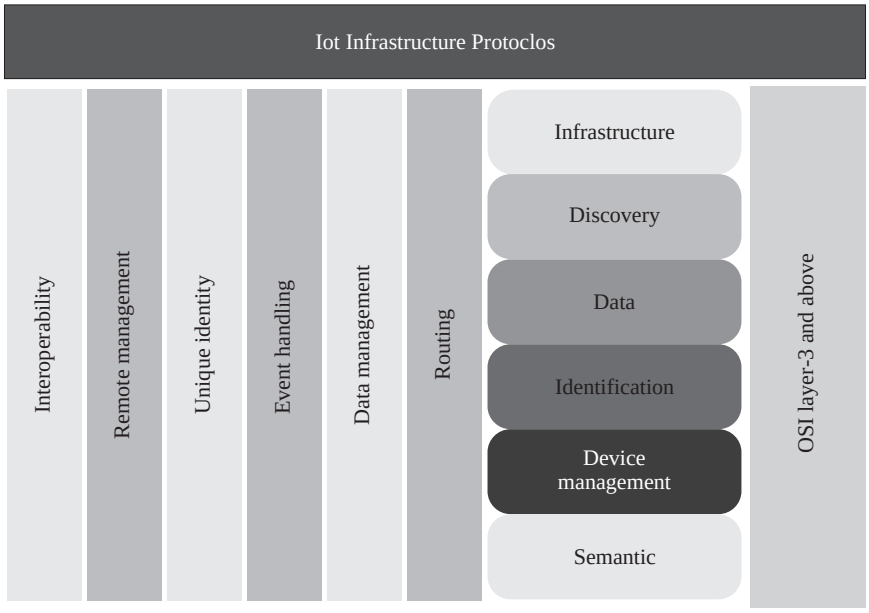


Figure 8.1 Various IoT communication protocol groups

Before delving into the various IoT communication protocols, we outline some of the essential terms associated with IoT networks that are indirectly responsible for the development of these communication protocols.

8.1.1 Constrained nodes

Constrained nodes is a term associated with those nodes where regular features of Internet-communicating devices are generally not available. These drawbacks are often attributed to the constraints of costs, size restrictions, weight restrictions, and available power for the functioning of these nodes. The resulting restrictions of memory and processing power often limit the usage of these nodes. For example, most of these nodes have a severely limited layer 2 capability and often lack full connectivity features and broadcasting capabilities. While architecting their use in networks and networked applications, these nodes require special design considerations. The issues of energy optimization and bandwidth utilization are dominant work areas associated with these nodes [1].

8.1.2 Constrained networks

Constrained networks [2], [1] are those networks in which some or all of the constituent nodes are limited in usage aspects due to the following constraints:

- limited processing power resulting in restrictions on achieving smaller duty cycles.

- low data rates and low throughput.
- asymmetric links and increased packet losses.
- restrictions on supported packet sizes due to increased packet losses.
- lack of advanced layer 3 functions such as multicasting and broadcasting.
- limited temporal device reachability from outside the network due to the inclusion of sleep states for power management in the devices.

8.1.3 Types of constrained devices

Constrained devices can be divided into three distinct classes according to the device's functionalities:

- **Class 0**: These devices are severely constrained regarding resources and capabilities. The barely feasible memory and processing available in these classes of devices do not allow for direct communication to the Internet. Even if the devices manage to communicate to the Internet directly, the mechanisms in place for ensuring the security of the device are not supported at all due to the device's reduced capabilities. Typically, this class of device communicates to the Internet through a gateway or a proxy.
- **Class 1**: These devices are constrained concerning available code space and processing power. They can primarily talk to the Internet, but cannot employ a regular full protocol stack such as HTTP (hyper text transfer protocol). Specially designed protocols stacks such as CoAP (common offer acceptance portal) can be used to enable Internet-based communication with other nodes. Compared to Class 0 devices, Class 1 devices have a comparatively increased power budget, which is attributed to the increased functionalities it supports over Class 0 devices. This class of devices does not need a gateway for accessing the Internet and can be armed with security features for ensuring safer communication over the Internet.
- **Class 2**: These devices are functionally similar to regular portable computers such as laptops and PDAs (personal digital assistants). They have the ability and capacity to support full protocol stacks of commonly used protocols such as HTTP, TLS, and others. However, as compared to the previous two classes of devices, these devices have a significantly higher power budget.

8.1.4 Low power and lossy networks

Low power and lossy networks (LLNs) typically comprise devices or nodes with limited power, small usable memory space, and limited available processing resources [4]. The network links between the devices in this network may be composed of low power Wi-Fi or may be based on the IEEE 802.15.4. The physical layers of the devices comprising LLNs are characterized by high variations in delivery rates,

significant packet losses, and other similar behavior, which makes it quite unreliable, and often compromises network stability. However, LLNs have found extensive use in application areas such as industrial automation and monitoring, building automation, smart healthcare, smart homes, logistics, environment monitoring, and energy management.

8.2 Infrastructure Protocols

The protocols covered in this section are hugely dependent on the network and the network infrastructure for its operation. This section covers eight popular IoT-based communication technologies: Internet Protocol Version 6 (IPv6), Lightweight On-demand Ad hoc Distance vector Routing Protocol–Next Generation (LOADng), Routing Protocol for Low-Power and Lossy Networks (RPL), IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN), Quick UDP Internet Connection (QUIC), micro IP (uIP), nanoIP, and Content-Centric Networking (CCN).

8.2.1 Internet protocol version 6 (IPv6)

The Internet Protocol Version 6 or IPv6, as it is commonly known, is a resultant of the developments on and beyond IPv4 due to fast depleting address ranges in IPv4. The IPv4 was not designed to handle the needs of the future Internet systems, making it cumbersome and wasteful to use for IoT-based applications. The needs of massive scalability and limited resources gave rise to IPv6, which was developed by the IETF (Internet Engineering Task Force); it is also termed as the Internet version 2 [5].

Similar to IPv4, IPv6 also works on the OSI layer 3 (network layer). However, in contrast to IPv4 (which is 32 bits long and offers around 4,294,967,296 addresses), IPv6 has a massive logical address range (which is 128 bits long). Additional features in IPv6 include auto-configuration features, end-to-end connectivity, inbuilt security measures (IPSec), provision for faster routing, support for mobility, and many others. These features not only make IPv6 practical for use in IoT but also makes it attractive for a majority of the present-day and upcoming IoT-based deployments. Interestingly, as IPv6 was designed entirely from scratch, it is not backward compatible; it cannot be made to support IPv4 applications directly. Figure 8.2 shows the differences between IPv4 and IPv6 packet structures.

Some of the important features of IPv6 are as follows:

(i) **Larger Addressing Range**: IPv6 has roughly four times more addressable bits than IPv4. This magnanimous range of addresses can accommodate the address requirements for any number of connected or massively networked devices in the world.

(ii) **Simplified Header Structure**: Unlike IPv4, the IPv6 header format is quite simple. Although much bigger than the IPv4 header, the IPv6 header's increased

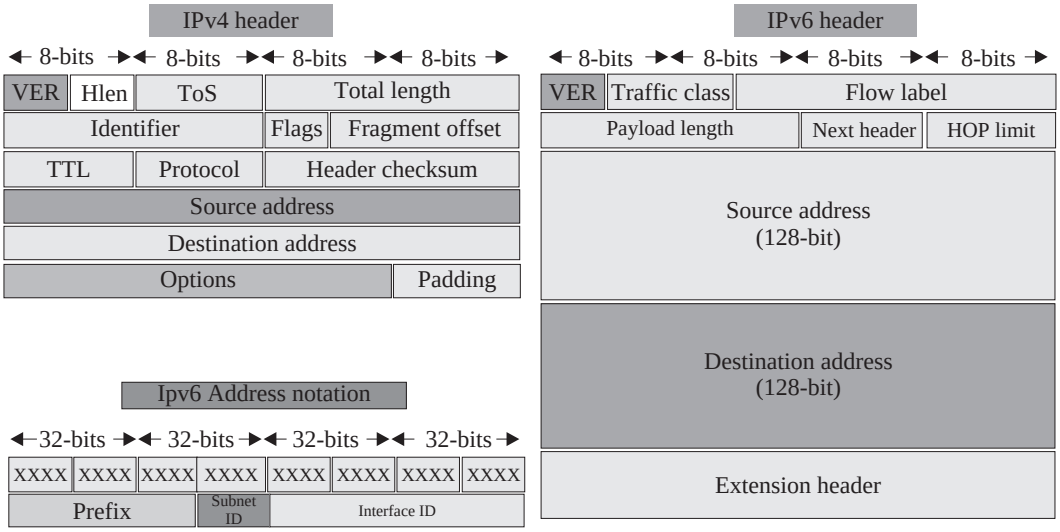


Figure 8.2 Differences between IPv4 and IPv6 packets and the IPv6 address notation

size is mainly attributed to the increased number of bits needed for addressing purposes.

(iii) **End-to-End Connectivity**: Unlike IPv4, the IPv6 paradigm allows for globally unique addresses on a significantly massive scale. This scheme of addressing enables packets from a source node using IPv6 to directly reach the destination node without the need for network address translations en route (as is the case with IPv4).

(iv) **Auto-configuration**: The configuration of addresses is automatically done in IPv6. It supports both stateless and stateful auto-configuration methods and can work even in the absence of DHCP (dynamic host configuration protocol) servers. This mechanism is not possible in IPv4 without DHCP servers.

(v) **Faster Packet Forwarding**: As IPv6 headers have all the seldom-used optional fields at the end of its packet, the routing decisions by a router are taken much faster, by checking only the first few fields of the header.

(vi) **Inbuilt Security**: IPv6 supports inbuilt security mechanisms (IPSec) that IPv4 does not directly support. IPv4 security measures were attained using separate mechanisms in conjunction with IPv4. The present-day version of IPv6 has security as an optional feature.

(vii) **Anycast Support**: Multiple networking interfacesare assigned the same IPv6 addresses globally; these addresses are known as anycast addresses. This mechanism enables routers to send packets to the nearest available destination during routing.

(viii) **Mobility Support**: IPv6 has one of the essential features that is crucial for IoT and the modern-day connected applications: mobility support. The mobility support of IPv6 allows for mobile nodes to retain their IP addresses and remain connected, even while changing geographic areas of operation.

(ix) **Enhanced Priority Support**: The priority support system in IPv6 is entirely simplified as compared to IPv4. The use of traffic classes and flow labels determine the most efficient routing paths of packets for the routers.

(x) **Extensibility of Headers**: The options part of an IPv6 header can be extended by adding more information to it; it is not limited in size. Some applications may require quite a large options field, which may be comparable to the size of the packet itself.

IPv6 Addressing

The IPv6 addressing scheme has a crucial component: the interface identifier (IID). IID is made up of the last 64 bits (out of the 128 bits) in the IPv6 address. IPv6 incorporates the MAC (media access control) address of the system for IID generation. As a device's MAC address is considered as its hardware footprint and is globally unique, the use of MAC makes IID unique too. The IID is auto-configured by a host using IEEE's extended unique identifier (EUI-64) format. Figure 8.2 illustrates the IPv6 addressing notation. IPv6 supports three types of unicasting: Global unicast address (GUA), link local address (LL), and unique local address (ULA).

The GUA is synonymous with IPv4's static addresses (public IP). It is globally identifiable and uniquely addressable. The global routing prefix is designated by the first (most significant) 48 bits. The first three bits of this routing prefix is always set to 001; these three bits are also the most significant bits of this prefix. In contrast, LLs are auto-configured IPv6 addresses, whose communication is limited to within a network segment only (under a gateway or a router). The first 16 bits of LL addresses are fixed and equals FE80 in hexadecimal. The subsequent 48 bits are set to 0. As these addresses are not routable, the LLs' scope is restricted to within the operational purview of a router or a gateway. Finally, ULAs are locally global and unique. They are meant for use within local networks only. Packets from ULAs are not routed to the Internet. The first half of an ULA is divided into four parts and the last half is considered as a whole. The four parts of the first part are the following: Prefix, local bit, global ID, and subnet ID, whereas the last half contains the IID. ULA's prefix is always assigned as FD in hexadecimal (1111 110 in binary). If the least significant bit in this prefix is assigned as 1, it signifies locally assigned addresses.

IPv6 Address Assignment

Any node in an IPv6 network is capable of auto-configuring its unique LL address. Upon assigning an IPv6 address to itself, the node becomes part of many multicast groups that are responsible for any communication within that segment of the network. The node then sends a neighbor solicitation message to all its IPv6 addresses.

If no reply is received in response to the neighbor solicitation message, the node assumes that there is no duplicate address in that segment, and its address is locally unique. This mechanism is known as duplicate address detection (DAD) in IPv6. Post DAD, the node configures the IPv6 address to all its interfaces and then sends out neighbor advertisements informing its neighbors about the address assignment of its interfaces. This step completes the IPv6 address assignment of a node.

IPv6 Communication

An IPv6 configured node starts by sending a router solicitation message to its network segment; this message is essentially a multicast packet. It helps the node in determining the presence of routers in its network segment or path. Upon receiving the solicitation message, a router responds to the node by advertising its presence on that link. Once discovered, the router is then set as that node's default gateway. In case the selected gateway is made unavailable due to any reason, a new default gateway is selected using the previous steps.

If a router upon receiving a solicitation message determines that it may not be the best option for serving as the node's gateway, the router sends a redirect message to the node informing it about the availability of a better router (which can act as a gateway) within its next hop.

IPv6 Mobility

A mobile IPv6 node located within its home link uses its home address for routing all communication to it. However, when the mobile IPv6 node goes beyond its home link, it has to first connect to a foreign link for enabling communication. A new IPv6 address is acquired from the foreign link, which is also known as the mobile node's care-of-address (CoA). The mobile node now binds its CoA to its home agent (a router/gateway to which the node was registered in its home segment). This binding between the CoA and the home agent is done by establishing a tunnel between them. Whenever the node's home agent receives a correspondence message, it is forwarded to the mobile node's CoA over the established tunnel. Upon receiving the message from a correspondent node, the mobile node may choose not to reply through its home agent; it can communicate directly to the correspondent node by setting its home address in the packet's source address field. This mechanism is known as route optimization.

Check yourself

IPv6 header structure, IPv6 extension header types, Neighbor discovery using IPv6, Mobility in IPv6

8.2.2 LOADng

LOADng stands for Lightweight On-demand Ad hoc Distance vector Routing Protocol – Next Generation. This protocol is inspired by the AODV (Ad hoc On-Demand Distance Vector) routing protocol, which is primarily a distance vector routing scheme [6]. Figure 8.3 illustrates the LOADng operation. Unlike AODV, LOADng was developed as a reactive protocol by taking into consideration the challenges of Mobile Ad hoc Networks (MANETs). The LOADng process starts with the initiation of the action of route discovery by a LOADng router through the generation of route requests (RREQs), as illustrated in Figure 8.3(a). The router forwards packets to its nearest connected neighbors, each of which again forwards the packets to their one-hop neighbors. This process is continued until the intended destination is reached. Upon receiving the RREQ packet, the destination sends back a route reply (RREP) packet toward the RREQ originating router (Figure 8.3(b)). In continuation, route error (RERR) messages are generated and sent to the origin router if a route is found to be down between the origin and the destination.

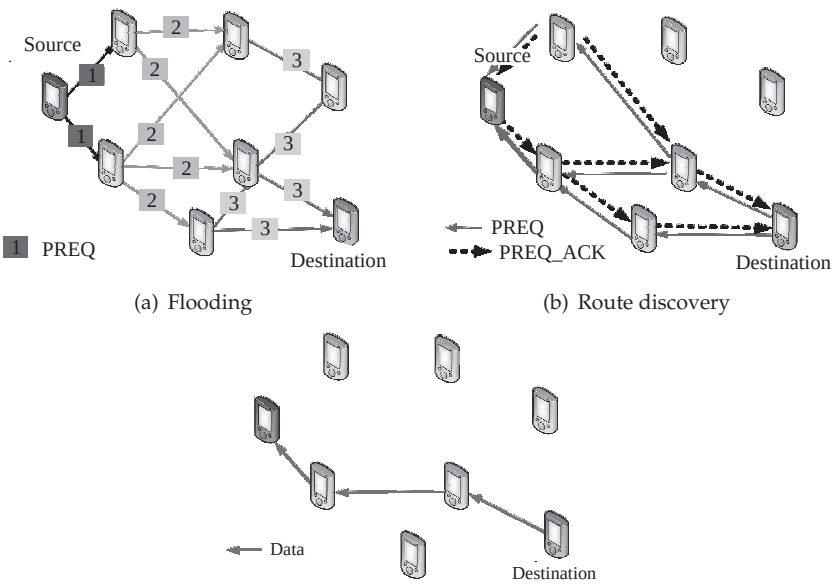


Figure 8.3 The LOADng routing mechanism

To summarize the operation of LOADng, a router performs the following tasks:

- Bi-directional network route discovery between a source and a destination.
- Route establishment and route maintenance between the source and the destination only when data is to be sent through the route.

- Generation of control and signaling traffic in the network only when data is to be transferred or a route to the destination is down.

Operational Principle

A LOADng router transmits an RREQ over all of its LOADng interfaces whenever a data packet from a local data source is received by it for transmission to a destination whose routing entry (a tuple) is not present with it. Figure 8.3(a) shows the flooding operation, where each LOADng's forward interfaces are numbered separately. Considering that it takes three hops to discover the destination from the source LOADng node, the individual forward interfaces are numbered from 1 to 3. The RREQ encodes the destination address received from the local source through the packet. The routing set managing the routing entries at each LOADng router updates or inserts an entry (with information of the originating address, and the immediate neighbor LOADng router) upon receiving an RREQ. This also works to enable a record of the reverse route between the source and destination (Figure 8.3(b)). The received RREQ initiates the checking of the destination address so that if the packets are intended for a local interface of a LOADng router, an RREP is sent back using the reverse route. In case the destination address is not local, it is forwarded to other LOADng interfaces in a hop-by-hop unicast manner through flooding.

When an RREP is received, it is recorded in the routing entry as the forward path toward the origin of the RREP along with the LOADng router that forwarded the message. The route metrics are additionally updated using RREQ and RREP messages. The LOADng determines the desired metric to be used (Figure 8.3(c)).

Check yourself

AODV routing, MANETs

8.2.3 RPL

RPL stands for routing protocol for low-power and lossy networks (LLN) and is designed for IPv6 routing. It follows a distance vector based routing mechanism [7]. The protocol aims to achieve a destination-oriented directed acyclic graph (DODAG). The network DODAG is formed based on an objective function and a set of network metrics. The DODAG built by RPL is a logical routing topology which is built over a physical network. The logical topology is built using specific criteria set by network administrators. The most optimum path (best path) is calculated from the objective function, a set of metrics, and constraints. The metrics in RPL may be expected transmission values (ETX), path latencies, and others. Similarly, the constraints in RPL include encryption of links, the presence of battery-operated nodes, and others. In general, the metrics are either minimized or maximized, whereas the constraints need to be minimized. The objective function dictates the rules for the formation of the

DODAG. Interestingly, in RPL, a single node in the mesh network may have multiple objective functions. The primary reason for this is attributed to the presence of different network traffic path quality requirements that need separate addressal within the same mesh network. Using RPL, a node within a network can simultaneously join more than one RPL instance (graphs). This enables RPL to support QoS-aware and constraint-based routing. An RPL node can also simultaneously take on multiple network roles: leaf node, router, and others. Figure 8.4 shows the RPL mechanism with different intra-mesh addressing arising due to different requirements of network and objective functions. The RPL border router, which is also the RPL root (in the illustrated figure), handles the intra-mesh addressing.

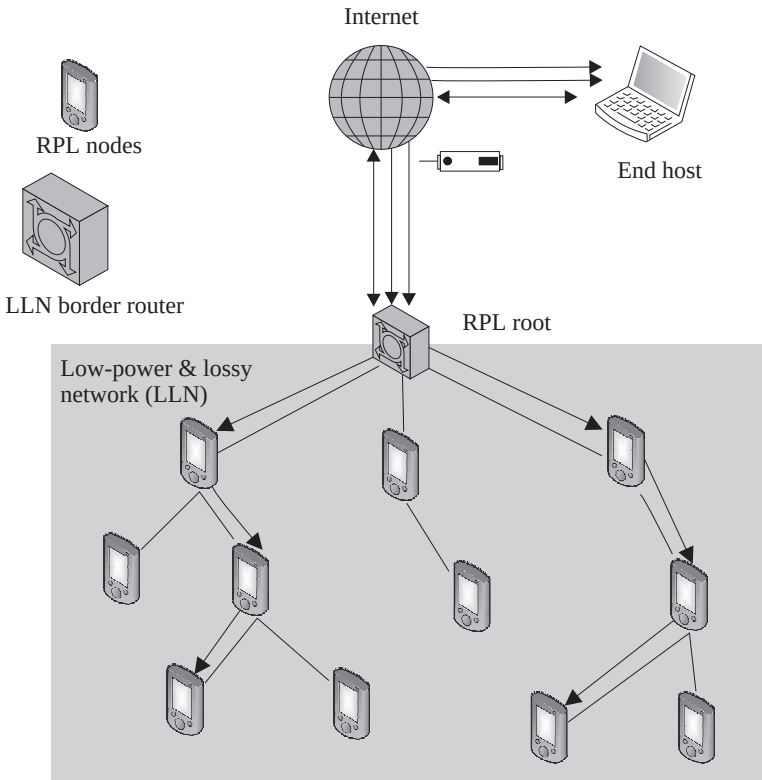


Figure 8.4 RPL information flow mechanism with different intra-mesh addressing and paths

RPL Instances

There are two instances associated with RPL: global and local. Global RPL instances are characterized by coordinated behavior and the possibility of the presence of more than one DODAG; they have a long lifetime. In contrast, local RPL instances are characterized by single DODAGs. The local RPL DODAG's root is associated directly with the DODAG-ID. The RPL instance ID is collaboratively and unilaterally allocated; it is divided between global and local RPL instances. Even the RPL control and data

messages are tagged with their corresponding RPL instances using RPL instance IDs to avoid any ambiguity in operations.

Check yourself

Directed acyclic graphs (DAG), destination oriented directed acyclic graph (DODAG), vector based routing

8.2.4 6LoWPAN

6LoWPAN allows low power and constrained devices/nodes to connect to the Internet. 6LoWPAN stands for IPv6 over low power wireless personal area networks. As the name suggests, it enables IPv6 support for WPANs, which are limited concerning power, communication range, memory, and throughput [8]. 6LoWPAN is designed to be operational and straightforward over low-cost systems, and extend IPv6 networking capabilities to IEEE 802.15.4-based networks. Popular uses of this protocol are associated with smart grids, M2M applications, and IoT. 6LoWPAN allows constrained IEEE 802.15.4 devices to accommodate 128-bit long IPv6 addresses. This is achieved through header compression, which allows the protocol to compress and retro-fit IPv6 packets to the IEEE 802.15.4 packet format.

6LoWPAN networks can consist of both limited capability (concerning throughput, processing, memory, range) devices—called reduced function devices (RFD)—and devices with significantly better capabilities, called full function devices (FFD). The RFDs are so constrained that for accessing IP-based networks, they have to forward their data to FFDs in their personal area network (PAN). The FFDs yet again forward the forwarded data from the RFD to a 6LoWPAN gateway in a multi-hop manner. The gateway connects the packet to the IPv6 domain in the communication network. From here on, the packet is forwarded to the destination IP-enabled node/device using regular IPv6-based networking.

6LoWPAN Stack

The 6LoWPAN stack rests on top of the IEEE 802.15.4 PHY and MAC layers, which are generally associated with low rate wireless personal area networks (LR-WPAN). The choice of IEEE 802.15.4 for the base layer makes 6LoWPAN suitable for low power LR-WPANs. The network layer in 6LoWPAN enabled devices (layer 3) serves as an adaptation layer for extending IPv6 capabilities to IEEE 802.15.4 based devices. Figure 8.5 shows the 6LoWPAN packet structure.

- **PHY and MAC layers**: The PHY layer consists of 27 wireless channels, each having their separate frequency band and varying data rates. The MAC layer defines the means and methods of accessing the defined channels and use them for communication. The 6LoWPAN MAC layer is characterized by the following:

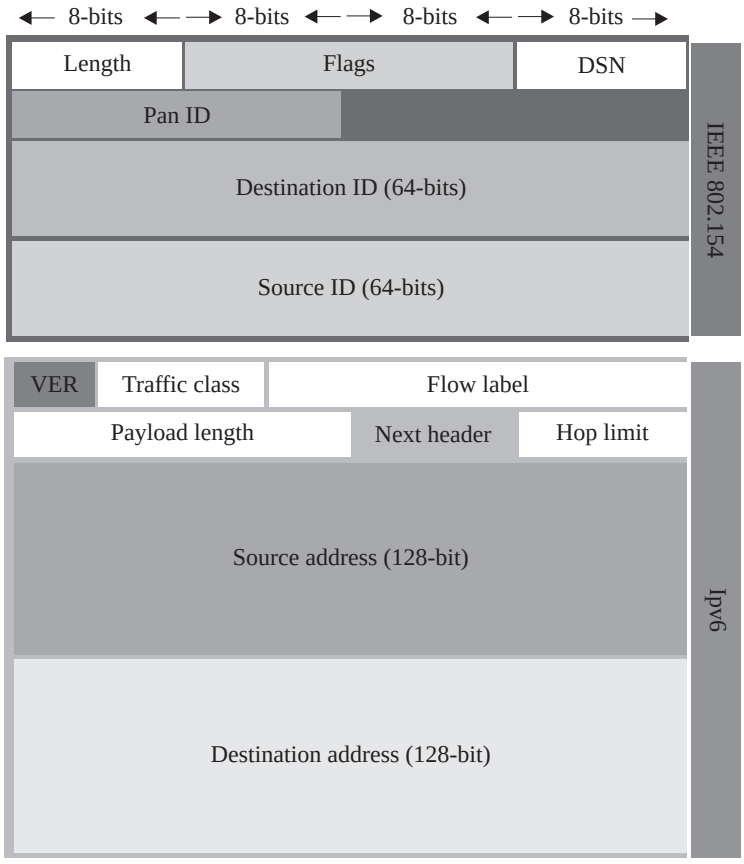


Figure 8.5 6LoWPAN packet structure

(i) Beaconing tasks for device identification. These tasks include both beacon generation and beacon synchronization.
(ii) Channel access control is provided by CSMA/CA.
(iii) PAN membership control functions. Membership functions include association and dissociation tasks.

- **Adaptation layer**: As mentioned previously, 6LoWPAN accommodates and retro-fits the IPv6 packet to the IEEE 802.15.4 packet format. The challenge presented to 6LoWPAN is evident from the fact that IPv6 requires a minimum of 1280 octets for transmission. In contrast, IEEE 802.15.4 can support a maximum of only 1016 octets (127 bytes): 25 octets for frame overheads and 102 octets for payload. Additional inclusion of options in the IEEE 802.15.4 frame, such as security in the headers, leaves only 81 octets for IPv6 packets to use, which is insufficient. Even out of these available 81 octets, the IPv6 header reserves 40 octets for itself, 8 octets for UDP (user datagram protocol), and 20 octets for TCP (transmission control protocol), which are added in the upper layers. This leaves

only 13 octets available at the disposal of the upper layers and the data itself. The 6LoWPAN adaptation layer between the MAC and the network layers takes care of these issues through the use of header compression, packet forwarding, and packet fragmentation.

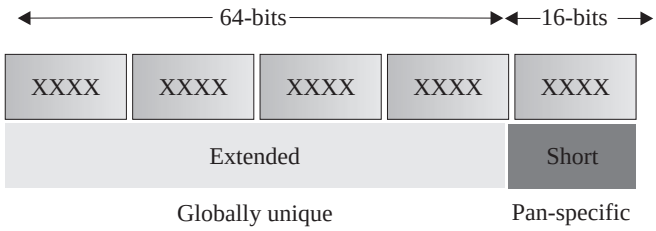


Figure 8.6 6LoWPAN address format

- **Address Format**: The 6LoWPAN address format is made up of two parts: 1) the short (16-bit) address and 2) the extended (64-bit) address. The short address is PAN specific and is used for identifying devices within a PAN only, which makes its operational scope highly restricted and valid within a local network only. In contrast, the globally unique extended address is valid globally and can be used to identify devices, even outside the local network uniquely. Figure 8.6 illustrates the 6LoWPAN address format.

Encapsulation Header Formats

The encapsulation headers, as the name suggests, defines methods and means by which 6LoWPAN encapsulates the IPv6 payloads within IEEE 802.15.4 frames. Figure 8.7 outlines the various header types associated with 6LoWPAN. 6LoWPAN has three encapsulation header types associated with it: dispatch, mesh addressing, and fragmentation. This system is similar to the IPv6 extension headers. The headers are identified by a *header type* field placed in front of the headers. The dispatch header type is used to initiate communication between a node and a destination node. The mesh addressing header is used for multi-hop forwarding by providing support for layer two forwarding of messages. Finally, the fragmentation header is used to fit large payloads to the IEEE 802.15.4 frame size.

Check yourself

LR-WPAN, WPAN, Beaconing

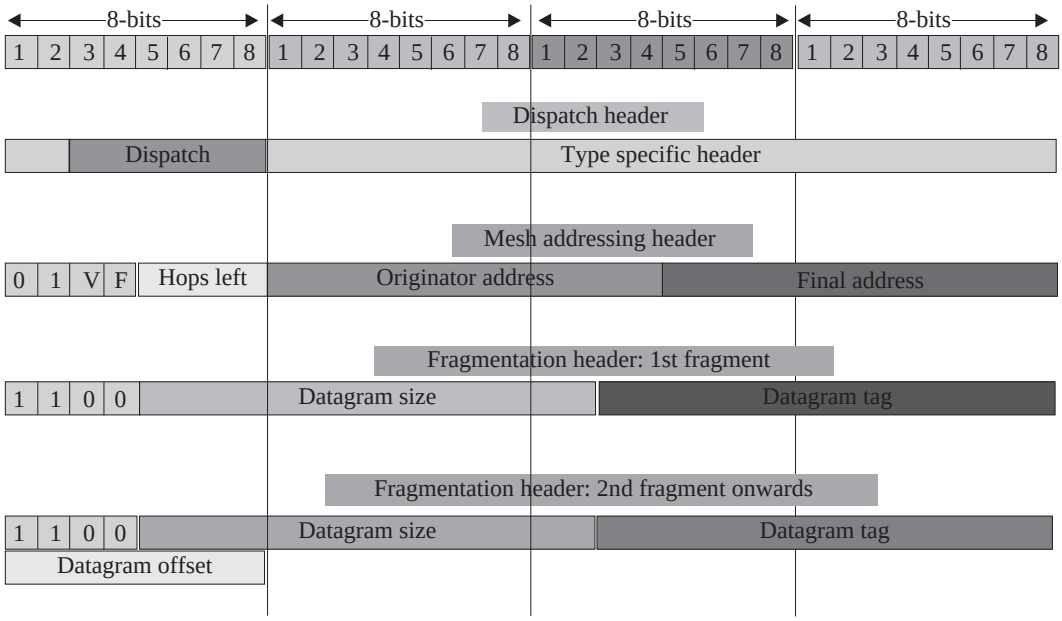


Figure 8.7 6LoWPAN header structures

8.2.5 QUIC

Quick UDP internet connections (QUIC) was developed (and still undergoing developments) to work as a low-latency and independent TCP connection [9]. The aim behind the development of this protocol is to achieve a highly reduced latency (almost zero round-trip-time) communication scheme with stream and multiplexing support like the SPDY protocol developed by Google. Figure 8.8 illustrates the differences between the positions of the various functionalities in QUIC and regular HTTP protocols.

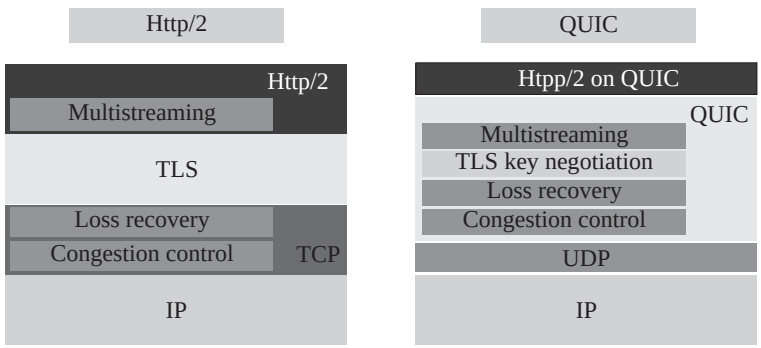


Figure 8.8 Differences between HTTP and QUIC protocols

The connection latency in QUIC is reduced by reducing the number of round trips incurred during connection establishment in TCP, such as those for handshaking, data requests, and encryption exchanges. This is achieved by including session negotiation information in the initial packet itself. The QUIC servers further enhance this compression by publishing a static configuration record corresponding to the connections. Clients synchronize connection information through cookies received from QUIC servers.

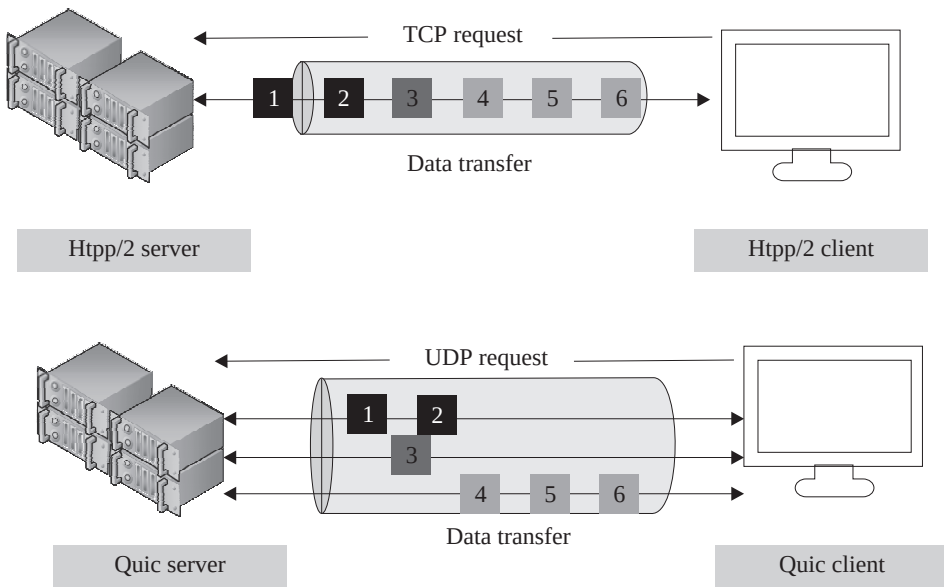


Figure 8.9 Differences between stream of packets over HTTP and QUIC protocols

QUIC uses advanced techniques such as packet pacing and proactive speculative retransmission to avoid congestion. Proactive speculative retransmission sends copies of most essential packets, which contain initial negotiation for encryption and error correction. The additional speedup is achieved using compression of data such as headers, which are generally redundant and repetitive. This feature enables QUIC connections to make multiple secured requests within a single congestion window, which would not have been possible using TCP. Figure 8.9 shows the difference in regular streaming of packets over HTTP and the improved performance of HTTP-over-QUIC during packet streaming. The use of UDP and multiple transmission paths significantly speeds up the performance of streaming over QUIC as compared to regular HTTP-based packet streaming.

Check yourself

QUIC use cases, SPDY protocol, TCP congestion control mechanism

8.2.6 Micro internet protocol (uIP)

The micro-IP (uIP) protocol is developed to extend the TCP/IP protocol stack capabilities to 8-bit and 16-bit microcontrollers [10]. uIP is an open-source protocol developed by the Swedish Institute of Computer Science (SICS). The low code space and memory requirements of uIP make it significantly useful for networking low-cost and low-power embedded systems. uIP now features a full IPv6 stack, which was developed jointly by Atmel, Cisco, and SICS. Figure 8.10 shows the micro-IP protocol stack.

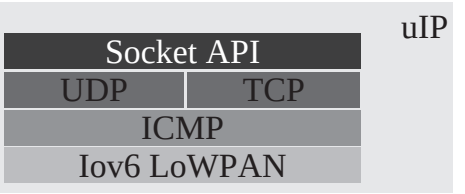


Figure 8.10 The uIP protocol

The main highlighting features of uIP, which makes it stand out from other IP-based protocols are as follows:

- The software interface of uIP does not require any operating system for working, making it quite easy to integrate with small computers.
- When called in a timed loop within the embedded system, it also manages all the network behavior and connection retries.
- The hardware driver for the uIP is responsible for packet builds, packet sending; it may also be used for response reception for the packets sent.
- uIP uses a minimal packet buffer (packet buffer = 1) in contrast to normal IP protocol stacks. This makes uIP suitable for low-power operations.
- The packet buffer is used in a half-duplex manner so that the same buffer can be repurposed for use in transmission and reception.
- Unlike regular TCP/IP protocols, uIP does not store data in buffers in case there is a need for retransmission. In the event of retransmission of packets, the previous data has to be reproduced and is recalled from the application code itself.
- Unlike conventional IP-based protocols, where a task is dedicated for each connection to a distant networked device/node, uIP stores connections in an array, and serves each connection sequentially through subroutine calls to the application for sending data.

Check yourself

uIP buffer format, uIP use-cases

8.2.7 Nano internet protocol (nanoIP)

The nano Internet protocol or NanoIP was designed to work with embedded devices, specifically sensor devices, by enabling Internetworking amongst these devices [11]. The concept of nanoIP enables wireless networking among low-power sensor devices, which is address-based, without incurring the overheads associated with the TCP/IP protocol stacks and mechanisms. Figure 8.11 shows the nano-IP TCP and UDP protocol stacks. The nanoIP is made up of two two transport mechanisms: nanoUDP and nanoTCP. These two transport mechanisms are analogous to the conventional UDP (unreliable transport protocol) and TCP (reliable transport protocol), respectively. NanoTCP even supports packet retransmissions and flow control, just like regular TCP. Instead of logical addressing, nanoIP uses hardware (MAC) addresses of devices for networking. The supported port range is 256 each for source and destination nodes, which is the allowable limit for an 8-bit port representation. With the advent of the nanoIP, several associated protocols have also come up, such as nHTTP and nPing.

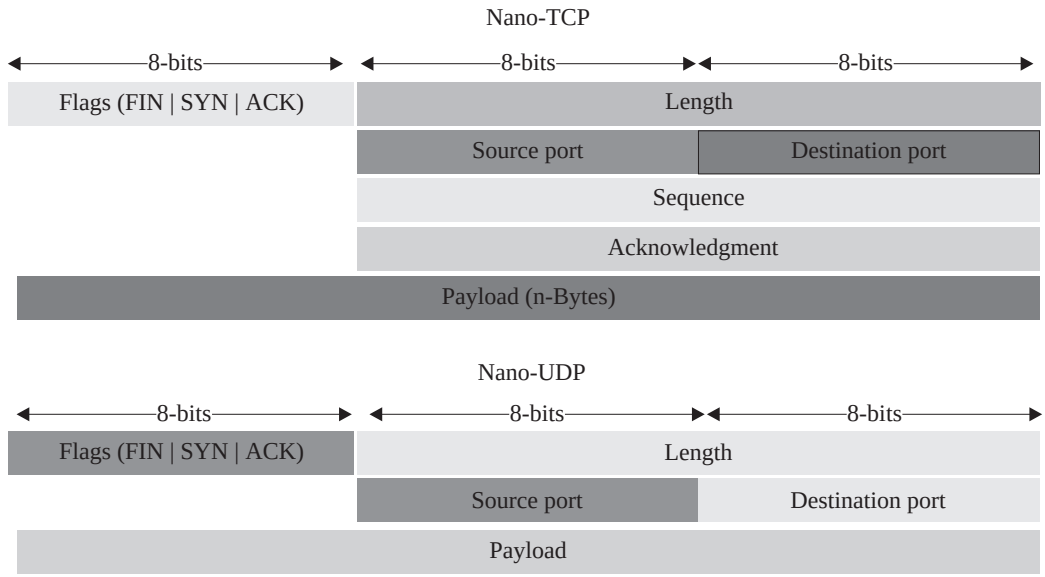


Figure 8.11 The nano-IP TCP and UDP protocols

Check yourself

nHTTP working, nPing working

8.2.8 Content-centric networking (CCN)

The content-centric networking (CCN) paradigm [12] is more commonly known as information-centric networking (ICN). Other names associated with this paradigm are named data networking (NDN) and publish–subscribe networking (PSIRP). CCN enables communication by defining and adhering to the concept of uniquely named data. This networking paradigm, unlike conventional networking approaches, is independent of location, application, and storage requirements. CCN is anchorless, which enables mobility and focuses on in-network caching for operations. These measures extend the features of data and communication efficiency, enhances scalability, and robustness, even in constrained and challenged networks. Figure 8.12 shows the operation of a typical CCN paradigm. Users can access cached content from multiple content generating sources by accessing data from trusted content servers, which also enable security of the data (not the communication channel).

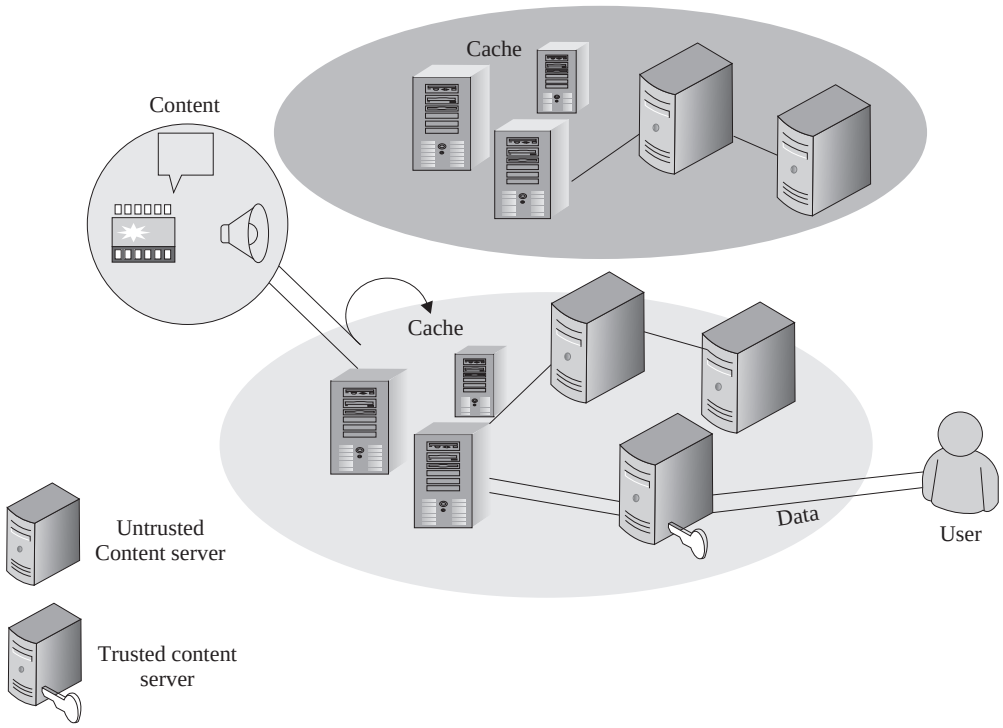


Figure 8.12 Content centric networking operation and its scope

In CCN, a forwarder checks a named request through hierarchical prefix matching (typically, longest prefix match) with a forwarding information base (FIB). A binary comparison is performed for prefix matching. The CCN request is a hierarchical sequence of network path segments. The FIB matching is then used to forward the named request to the appropriate network or network segment, which can respond to the issued request. The forwarder has to additionally determine the reverse path from the responder to the requester. All these operations are carried out without specifically binding a request to a network end point. The FIB at each CCN router stores information in a table, which is updated by a routing mechanism. Although the path segments and names are theoretically unbounded, they are restricted by the routing protocol being used for practical reasons.

Check yourself

Examples of publish–subscribe networking (PSIRP) and named data networking (NDN)

Points to ponder

A sensor node is made up of a combination of sensor/sensors, a processor unit, a radio unit, and a power unit.

8.3 Discovery Protocols

The protocols and paradigms covered in this section are largely focused on the discovery of services and logical addresses. We cover three interesting discovery protocols in this section: 1) Physical web, 2) mDNS, and 3) universal plug and play (UPnP).

8.3.1 Physical web

The physical web was designed to provide its users with the ability to interact with physical objects and locations seamlessly. The information to the users can be in the form of regular web pages or even dynamic web applications [13].

Some examples in the context of the physical web include user-friendly buses, which can alert its users about various route-related information, smart home appliances that can teach new users how to use them, self-diagnostic robots and machinery in industries, smart pet tags which can provide information about the pet's owner and its home location, and many more. Figure 8.13 shows the outline of a physical web model. The main takeaway of this concept is the seamless integration of several standalone smart systems through the web to provide information on demand to its users.

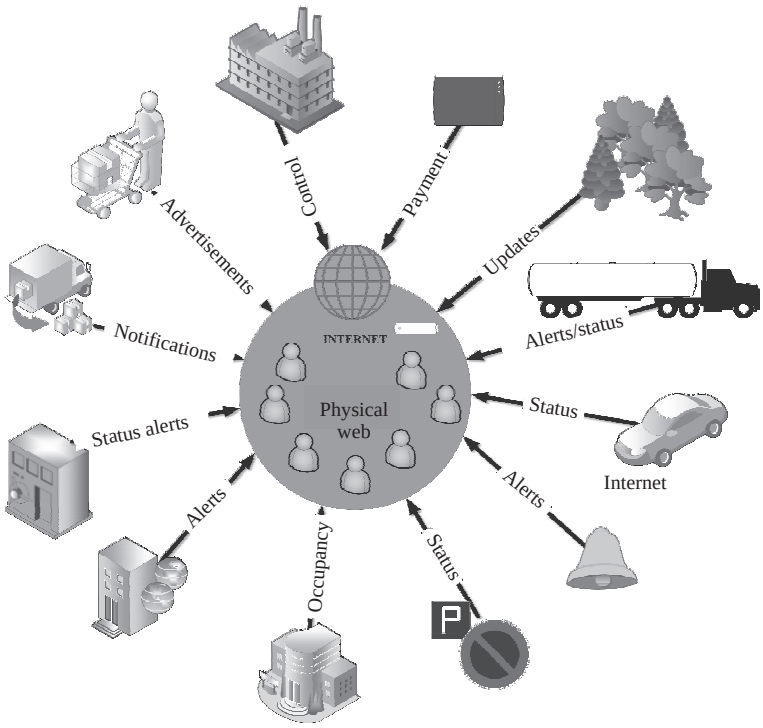


Figure 8.13 The physical web model

The physical web broadcasts a list of URLs within a short radius around it so that anyone in range can see the available URLs and interact with them. This paradigm is primarily built upon Bluetooth low energy (BLE), which is used to broadcast the content as beacons. The primary requirement of any supporting beacons to function in the physical web and broadcast URLs is their ability to support the Eddystone protocol specification. BLE was primarily chosen for the physical web due to its ubiquity, efficiency, and extended battery life of several years.

URLs are one of the core principles of the web and can be either flexible or decentralized. These URLs allow any application to have a presence on the web and enables the digital presence of an object or thing. As of now, physical web deployments have been undertaken in public spaces, and any device with a physical web scanner can detect these URLs. The use of URLs extends the benefits of the web security model to the physical web. Features such as secured login, secure communication over HTTPS (HTTP over secure socket layer), domain obfuscation, and others can be easily integrated with the physical web.

Check yourself

Eddystone protocol specification

8.3.2 Multicast DNS (mDNS)

The multicast domain name system or mDNS is explicitly designed for small networks and is analogous to regular DNS (domain name system), which is tasked with the resolution of IP addresses [14]. Interestingly, this system is free from any local name server from an operational point of view. However, it can work with regular DNS systems as it is a zero-configuration service. It uses multicast UDP for resolving host names. An mDNS client initiates a multicast query on the IP network, which asks a remote host to identify itself. The mDNS cache in the associated network subnet is updated with the multicast response received from the target. A node can give up its claim to a domain name by setting the time-to-live (TTL) field to zero in its response packet to an mDNS query. Some popular implementations of mDNS include the Apple Bonjour service and the networked printer discovery service in Windows 10 operating system from Microsoft. The main drawback of mDNS is its host name resolution reach to a top-level domain only.

Check yourself

DNS, DNS query, DNS response

8.3.3 Universal plug and play (UPnP)

Universal plug and play or UPnP encompasses a set of networking protocols aimed at service discovery and the establishment of functional network-based data sharing and communication services [15]. In brief, it is mainly used for enabling dynamic connections of devices to computing platforms. This paradigm is termed plug and play as the devices attaching to a computer network can configure themselves and update their hosts about their working configurations over a network. The UPnP is backed by a forum of many consumer electronics vendors and industries and is managed by the Open Connectivity Foundation. As UPnP is primarily designed for non-enterprise devices, its scope includes the discovery and intercommunication between networked devices such as mobiles, printers, access points, gateways, televisions, and other regular commercial systems enabled with IP capabilities. Figure 8.14 outlines the underlying UPnP stack and the relative location of the various functionalities in the stack.

The present-day UPnP has been designed to run on IP enabled networks, and makes use of the networking services of HTTP, XML, and SOAP for data transfer, device descriptions, and event generation and monitoring. UPnP enables UDP-based HTTP device search requests and advertisements using multicasting. The responses to device requests are necessarily unicast. UPnP advertisements use UDP port 1900 for multicasting. The unnecessary overheads and traffic generated by UPnP systems and their multicast behavior make them unsuitable for enterprise systems. The main

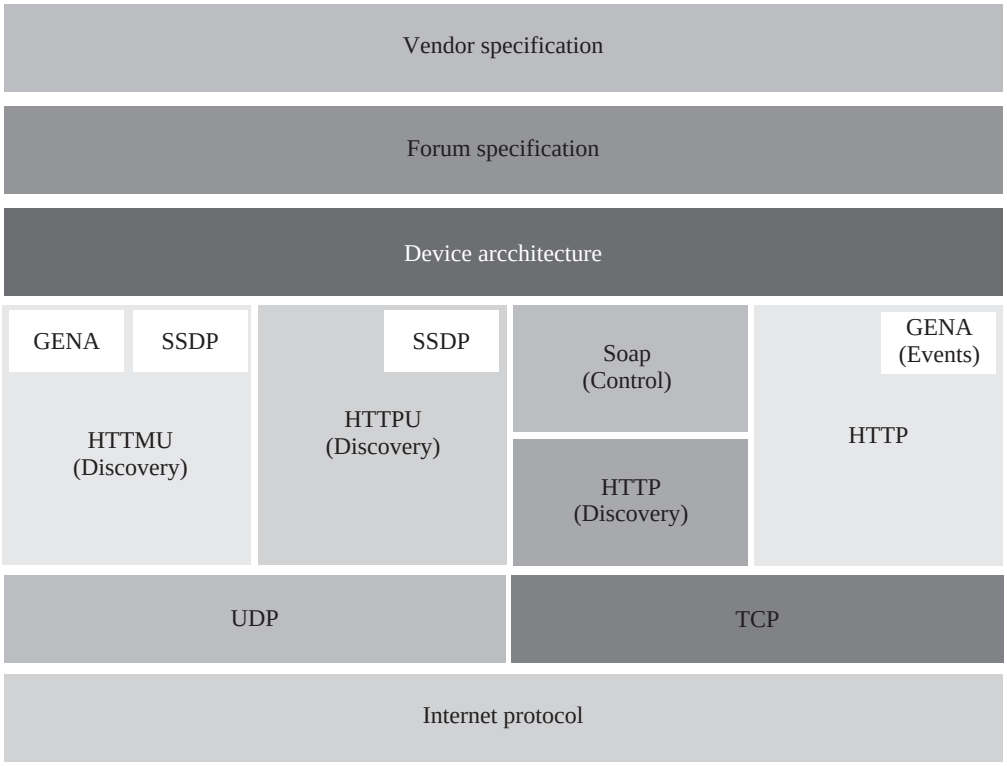


Figure 8.14 An outline of the basic UPnP stack

reason for this is because, on a large scale, the cost of this solution would be infeasible from an operational point of view.

Check yourself

Multicasting, Unicasting

8.4 Data Protocols

The protocols covered in this section are directly related to access, storage, and distribution of data through the IoT network. The data may be transferred between clients and servers as well as between brokers and subscribers in the IoT ecosystem. This section is further divided into seven parts: 1) MQTT, 2) MQTT-SN, 3) CoAP, 4) AMQP, 5) XMPP, 6) REST, and 7) websockets.

8.4.1 MQTT

Message queue telemetry transport or MQTT is a simple, lightweight publish–subscribe protocol, designed mainly for messaging in constrained devices and networks [16]. It provides a one-to-many distribution of messages and is payload-content agnostic. MQTT works reliably and flawlessly over high latency and limited bandwidth of unreliable networks without the need for significant device resources and device power. Figure 8.15 shows the working of MQTT. The MQTT paradigm consists of numerous clients connecting to a server; this server is referred to as a *broker*. The clients can have the roles of information publishers (sending messages to the broker) or information subscribers (retrieving messages from the broker). This allows MQTT to be largely decoupled from the applications being used with MQTT.

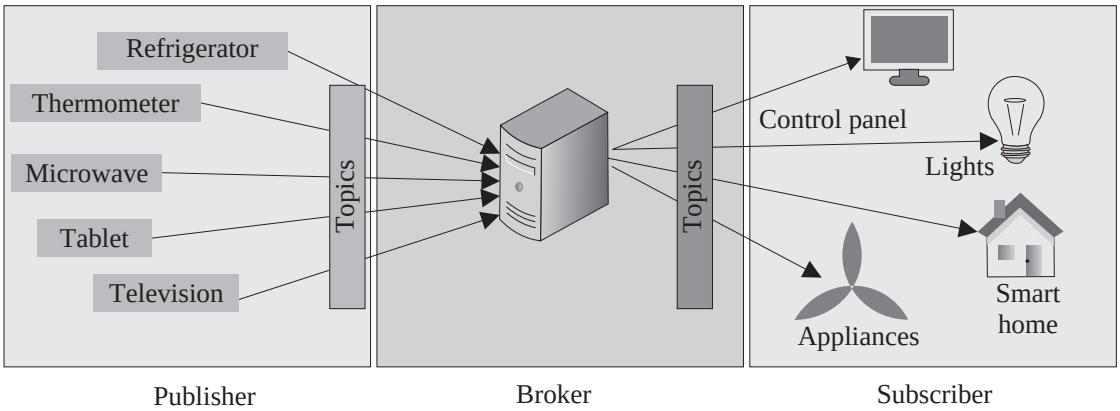


Figure 8.15 MQTT operation and its stakeholders

Operational Principle

MQTT is built upon the principles of hierarchical topics and works on TCP for communication over the network. Brokers receive new messages in the form of topics from publishers. A publisher first sends a control message along with the data message. Once updated in the broker, the broker distributes this topic's content to all the subscribers of that topic for which the new message has arrived. This paradigm enables publishers and subscribers to be free from any considerations of the address and ports of multiple destinations/subscribers or network considerations of the subscribers, and vice versa. In the absence of any subscribers of a topic, a broker normally discards messages received for that topic unless specified by the publisher otherwise. This feature removes data redundancies and ensures that maximally updated information is provided to the subscribers. It also reduces the requirements of storage at the broker. The publishers can set up default messages for subscribers in agreement with the broker if the publisher's connection is abruptly broken with the broker. This arrangement is referred to as the *last will and testament* feature of MQTT.

Multiple brokers can communicate in order to connect to a subscriber's topic if it is not present directly with the subscriber's primary broker.

MQTT's control message sizes can range between 2 bytes to 256 megabytes of data, with a fixed header size of 2 bytes. This enables the MQTT to reduce network traffic significantly. The connection credentials in MQTT are unencrypted and often sent as plain text. The responsibility of protecting the connection lies with the underlying TCP layer. The MQTT protocol provides support for 14 different message types, which range from connect/disconnect operations to acknowledgments of data. The following are the standard MQTT message types:

(i) CONNECT: Publisher/subscriber request to connect to the broker.

(ii) CONNACK: Acknowledgment after successful connection between publisher/subscriber and broker.

(iii) PUBLISH: Message published by a publisher to a broker or a broker to a subscriber.

(iv) PUBACK: Acknowledgment of the successful publishing operation.

(v) PUBREC: Assured delivery component message upon successfully receiving publish.

(vi) PUBREL: Assured delivery component message upon successfully receiving publish release signal.

(vii) PUBCOMP: Assured delivery component message upon successfully receiving publish completion.

(viii) SUBSCRIBE: Subscription request to a broker from a subscriber.

(ix) SUBACK: Acknowledgment of successful subscribe operation.

(x) UNSUBSCRIBE: Request for unsubscribing from a topic.

(xi) UNSUBACK: Acknowledgment of successful unsubscribe operation.

(xii) PINGREQ: Ping request message.

(xiii) PINGRESP: Ping response message.

(xiv) DISCONNECT: Message for publisher/subscriber disconnecting from the broker.

MQTT Message Delivery QoS

MQTT's features and content delivery mechanisms are primarily designed for message transmission over constrained networks and through constrained devices. However, MQTT supports three QoS features:

- At most once: This is a best-effort delivery service and is largely dependent on the best delivery efforts of the TCP/IP network on which the MQTT is supported. It may result in message duplication or loss.

- At least once: This delivery service guarantees assured delivery of messages. However, message redundancy through duplication is a possibility.
- Exactly once: This delivery service guarantees assured message delivery. Additionally, this service also prevents message duplication.

Check yourself

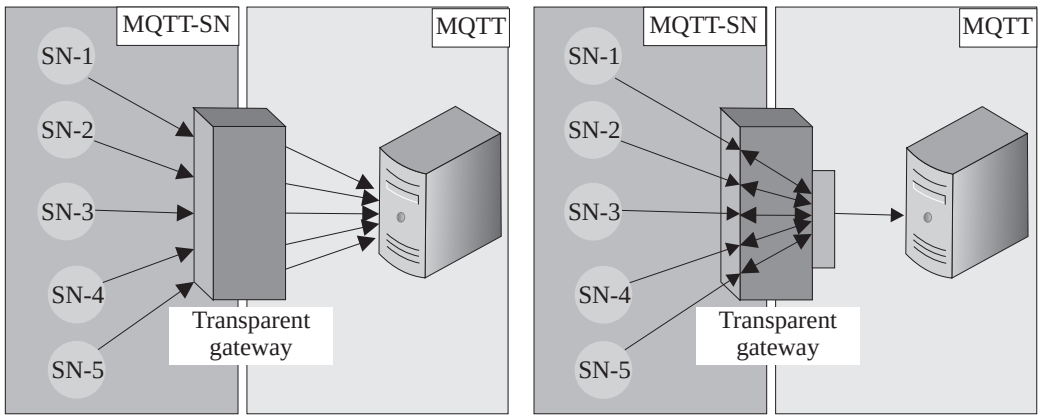
MQTT clients and servers available online, implementing MQTT

8.4.2 MQTT-SN

The primary MQTT protocols heavily inspire MQTT for sensor networks or MQTT-SN; however, the MQTT-SN is robust enough to handle the requirements and challenges of wireless communications networks in sensor networks [17]. Typical features of MQTT-SN include low bandwidth usage, ability to operate under high link failure conditions; it is suitable for low-power, low-cost constrained nodes and networks. The major differences between the original MQTT and MQTT-SN include the following:

- The CONNECT message types are broken into three messages in which two are optional and are tasked with the communication of the testament message and testament topic to the broker.
- The topic name in the PUBLISH messages are replaced by topic identifiers, which are only 2 bytes long. This reduces the traffic generated from the protocol and enables the protocol to operate over bandwidth-constrained networks.
- A separate mechanism is present for topic name registration with the broker in MQTT-SN. After a topic identifier is generated for the topic name, the identifier is informed to the publisher/subscribers. This mechanism also supports the reverse pathway.
- In special cases in MQTT-SN, pre-defined topic identifiers are present that need no registration mechanism. The mapping of topic names and identifiers are known in advance to the broker as well as the publishers/subscribers.
- The presence of a special discovery process is used to link the publisher/subscriber to the operational broker's network address in the absence of a preconfigured broker address.
- The subscriptions to a topic, Will topic, and Will message are persistent in MQTT-SN. The publishers/subscribers can modify their Will messages during a session.
- Sleeping publishers/subscribers are supported by a keep-alive procedure, which is offline, and which helps buffer the messages intended for them in the broker until they wake up. This feature of MQTT-SN is not present in regular MQTT.

Figure 8.16 shows the two gateway types in MQTT-SN: 1) the transparent gateway and 2) the aggregating gateway. The MQTT-SN converts/translates MQTT and MQTT-SN traffic by acting as a bridge between these two network types. The transparent gateway (Figure 8.16(a)) creates as many connections to the MQTT broker as there are MQTT-SN nodes within its operational purview; whereas the aggregating gateway (Figure 8.16(b)) creates a single connection to the MQTT broker, irrespective of the number of MQTT-SN nodes under it.



(a) Transparent gateway (b) Aggregating gateway

Figure 8.16 The MQTT-SN types

8.4.3 CoAP

The constrained application protocol, or CoAP as it is more popularly known, is designed for use as a web transfer protocol in constrained devices and networks, which are typically low power and lossy [18]. The constrained devices typically have minimal RAM and an 8-bit processor at most. CoAP can efficiently work on such devices, even when these devices are connected to highly lossy networks with high packet loss, high error rates, and bandwidth in the range of kilobits.

CoAP follows a request–response paradigm for communication over these lossy networks. Additional highlights of this protocol include support for service discovery, resource discovery, URIs (uniform resource identifier), Internet media handling support, easy HTTP integration, and multicasting support, that too while maintaining low overheads. Typically, CoAP implementations can act as both clients and servers (not simultaneously). A CoAP client's request signifies a request for action from an identified resource on a server, which is similar to HTTP. The response sent by the server in the form of a response code can contain resource representations as well. However, CoAP interchanges are asynchronous and datagram-oriented over UDP. Figure 8.17 shows the placement of CoAP in a protocol stack. Packet traffic

collisions are handled by a logical message layer incorporating the exponential back-off mechanism for providing reliability. The reliability feature of CoAP is optional. The two seemingly distinct layers of messaging (which handle the UDP and asynchronous messaging) and request-response (which handles the connection establishment) are part of the CoAP header.

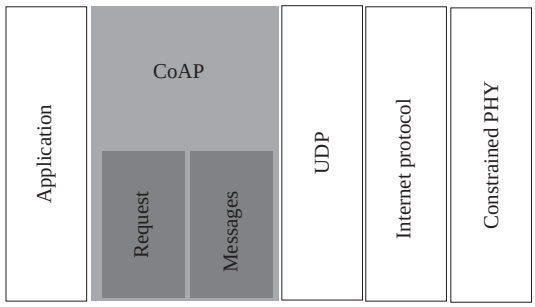


Figure 8.17 Position of the CoAP protocol in a stack

CoAP Features

The CoAP is characterized by the following main features:

(i) It has suitable web protocol for integrating IoT and M2M services in constrained environments with the Internet.

(ii) CoAP enables UDP binding and provides reliability concerning unicast as well as multicast requests.

(iii) Message exchanges between end points in the network or between nodes is asynchronous.

(iv) The limited packet header incurs significantly lower overheads. This also results in less complexity and processing requirements for parsing of packets.

(v) CoAP has provisions for URI and other content-type identifier support. CoAP additionally provides DTLS (datagram transport layer security) binding.

(vi) It has a straightforward proxy mechanism and caching capabilities, which is responsible for overcoming the effects of the lossy network without putting extra constraints on the low-power devices. The caching is based on the concept of the maximum age of packets.

(vii) The protocol provides a stateless mapping with HTTP. The server or receiving node does not retain information about the source of the message; rather, it is expected that the message packet carries that information with it. This enables CoAP's easy and uniform integration with HTTP.

CoAP Messaging

CoAP defines four messaging types: 1) Confirmable (CON), 2) non-confirmable (NON), 3) acknowledgment (ACK), and 4) reset. The method codes and the response codes are included in the messages being carried. These codes determine whether the message is a request message or a response message. Requests are typically carried in confirmable and non-confirmable message types. However, responses are carried in both of these message types as well as with the acknowledgment message. The transmission of responses with acknowledgment messages is known as piggybacking and is quite synonymous with CoAP.

Operational Principle

CoAP is built upon the exchange of messages between two or more UDP end points. Options and payload follow the compact 4-byte binary header in CoAP. This arrangement is typical of request and response messages of CoAP. A 2-byte message ID is used with each message to detect duplicates.

Whenever a message is marked as a CON message, it signifies that the message is reliable. In the event of delivery failure of a CON message, subsequent retries are attempted with exponential back-off until the receiving end point receives an ACK with the same message ID (Figure 8.18). In case the recipient does not have the resources to process the CON message, a RESET message is sent to the originator of the CON message instead of an ACK message.

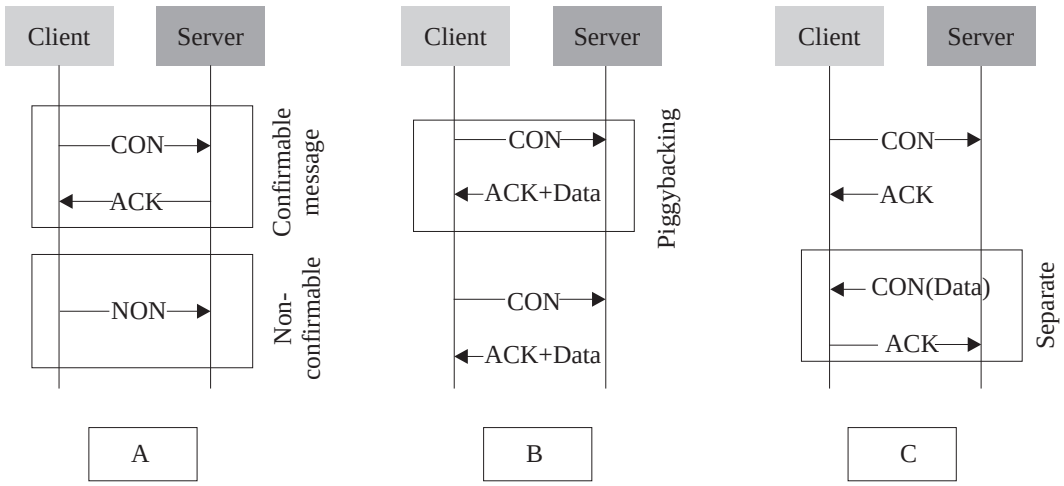


Figure 8.18 Various CoAP response–response models. (A): CON and NON messages, (B): Piggyback messages, and (C): Separate messages

Specific messages, which do not require reliable message transmission (such as rapid temporal readings of the environment from a sensor node), are sent as NON messages. NON messages do not receive an acknowledgment (Figure 8.18). However, the message ID associated with it prevents duplication. NON messages elucidate a

NON or CON response from a server, based on the settings and semantics of the application. If the receiver of the NON cannot process the message, a RESET message is sent to the originator of the NON message.

If a server fails to respond immediately to a request received by it in a CON message, an empty ACK response is sent to the requester to stop request retransmissions. Whenever the response is ready, a new CON message is used to respond to the previous request by the client. Here, the client then has to respond to the server using an ACK message. This scheme is known as a *separate response* (Figure 8.18).

The multicast support of CoAP over UDP results in multicast CoAP requests. The request and response semantics of CoAP is carried in the form of method and response codes in the CoAP messages itself. The options field of CoAP carries information about the requests and responses such as URI and MIME (multipurpose Internet mail extensions). The concept of tokens is used to match requests with their corresponding responses. The need for a token mechanism arose due to the asynchronous nature of the CoAP messaging. Similar to HTTP, CoAP uses GET, PUT, POST, and DELETE methods.

Check yourself

CoAP header fields, CoAP packet size

8.4.4 AMQP

AMQP or the advanced message queuing protocol is an open standard middleware at the application layer developed for message-oriented operations [19]. It tries to bring about the concept of interoperability between clients and the server by enabling cross-vendor implementations. Figure 8.19 shows the various components of AMQP and their relationships. An AMQP broker is tasked with maintaining message queues between various subscribers and publishers. The protocol is armed with features of message orientation, queuing, reliability, security, and routing. Both request–response and publish–subscribe methods are supported. AMQP is considered as a wire-level protocol. Here, the data format description is released on the network as a stream of bytes. This description allows AMQP to connect to anyone who can interpret and create messages in the same format. It also results in a level of interoperability where anyone with compliant or supporting means can make use of this protocol without any need for a specific programming language.

AMQP Features

AMQP is built for the underlying TCP and is designed to support a variety of messaging applications efficiently. It provides a wide variety of features such as flow-controlled communication, message-oriented communication, message delivery

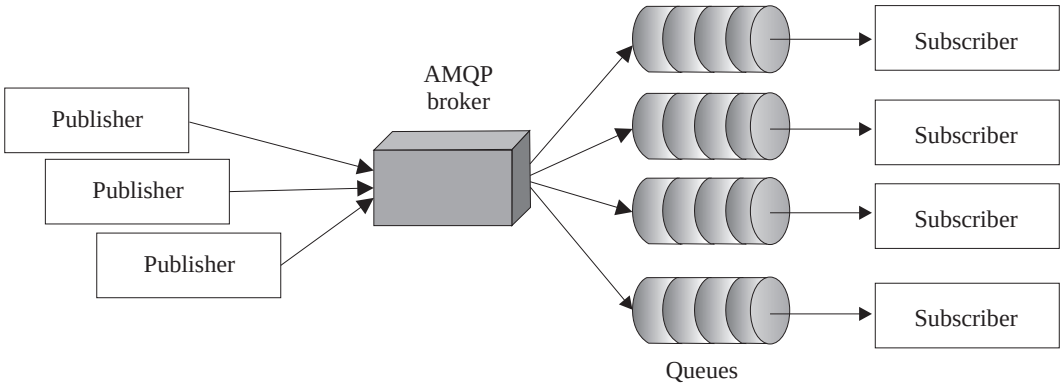


Figure 8.19 AMQP components and their relationships

guarantees (at most once, at least once, and exactly once), authentication support, and an optional SSL or TLS based encryption support. The AMQP is specified across four layers: 1) type system, 2) process to process asynchronous and symmetric message transfer protocol, 3) extensible message format, and 4) set of extensible messaging capabilities. In continuation, the primary unit of data in AMQP is referred to as a frame. These frames are responsible for the initiation of connections, termination of connections, and control of messages between two peers using AMQP. There are nine frame types in AMQP:

(i) Open: responsible for opening the connection between peers.

(ii) Begin: responsible for setup and control of messaging sessions between peers.

(iii) Attach: responsible for link attachment.

(iv) Transfer: responsible for message transfer over the link.

(v) Flow: responsible for updating the flow control state.

(vi) Disposition: responsible for updating of transfer state.

(vii) Detach: responsible for detachment of link between two peers.

(viii) End: responsible for truncation of a session.

(ix) Close: responsible for closing/ending a connection.

Operational Principle

The workings of AMQP revolve around the link protocol. A new link is initiated between peers that need to exchange messages by sending an ATTACH frame. A DETACH frame terminates the link between peers. Once a link is established, unidirectional messages are sent using the TRANSFER frame. Flow control is maintained by using a credit-based flow-control scheme, which protects a process from being overloaded by voluminous messages. Every message transfer state has to be mutually settled by both the sender and the receiver of the message. This

settlement scheme ensures reliability measures for messaging in AMQP. Any change in state and settlement of transfer is notified using a DISPOSITION frame. This allows for the implementation of various reliability guarantees. A session can accommodate multiple links in both directions. Unlike the link, a session is bidirectional and sequential. Upon initiation with a BEGIN frame, a session enables a conversation between peers. The session is terminated using an END frame. Multiple logically independent sessions can be multiplexed between peers over a connection. The OPEN frame initiates a connection and the connection is terminated by using a CLOSE frame.

8.4.5 XMPP

The extensible messaging and presence protocol, or XMPP, which was initially named as Jabber, is designed for message-oriented middlewares based on the extensible markup language (XML) [20]. XMPP was developed for instant messaging, maintenance of contacts, and information about network presence. Structured and extensible data between two networked nodes/devices can be exchanged in near real-time using this protocol. XMPP has found use in VOIP (voice-over Internet protocol) presence signaling, video and file transfers, smart grid, social networks, publish–subscribe systems, IoT applications, and others. The protocol, being open-source, has enabled a spurt of developments in various freeware as well as commercial messaging software. As XMPP follows a client–server architecture, peers in a network cannot talk directly to one another through XMPP. All communication between peers has to be routed through an XMPP server. The XMPP model is considered to be decentralized as anyone can host an XMPP server to which various clients can subscribe. Figure 8.20 shows the basic communication between the various XMPP stakeholders.

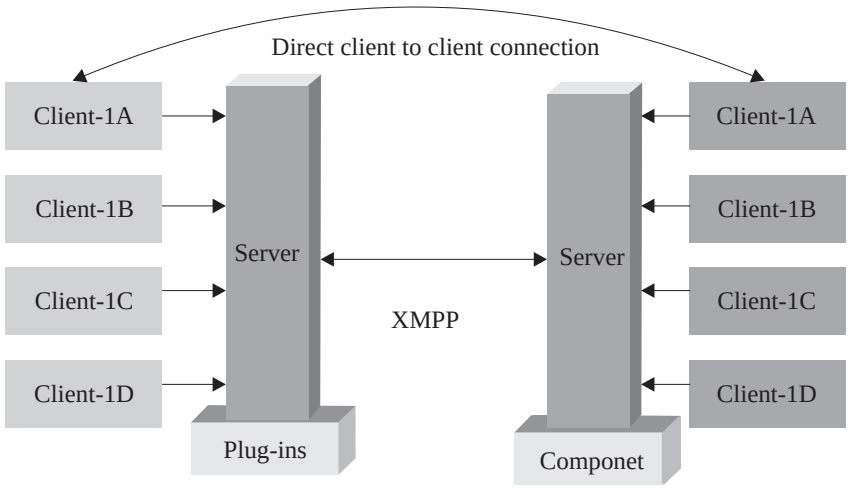


Figure 8.20 XMPP components

Operational Principle

A unique XMPP address, which is also referred to as a Jabber ID (JID), is assigned to every user on the network. The JID, similar to an email address, has a username and a domain name (`user@domain.com`). The domain name is mostly the IP address of the server hosting the XMPP service. XMPP allows its users to login from multiple devices by means of specifying resources. The resource is used to identify a user's clients/devices (home, mobile, work, laptop, and others), which is generally included in the JID by appending the JID with the resource name separated by a slash. A typical JID looks like this: `user@domain.com/resource`. Resources are prioritized using numerical IDs. Any message arriving at the default JID (without resource name) is forwarded to the resource with the highest priority (largest numerical ID value). Often JIDs without usernames are used for specific control messages and system messages, which are meant for the server. The use of JID in this mode—without an explicit IP address—allows XMPP to be used as an overlay network on top of multiple underlay networks.

XMPP Technologies

XMPP is an extensible, flexible, and diverse protocol; it has resulted in the development of a significant number of technologies based on it. Some key XMPP technologies include the following:

- **Core**: It deals with information about the core XMPP technologies for XML streaming over a network. The core includes the base XML layer for streaming, provides TLS-based encryption, imparts simple authentication and security layer (SASL) based authentication, informs about the availability of a network, provides UTF-8 support, and contact lists, which are presence enabled.
- **Jingle**: This provides session initiation protocol (SIP)-compatible multimedia signaling for voice, video, file transfer, and other applications. Various media transfer protocols such as TCP, UDP, RTP, or even in-band XMPP is supported. The Jingle session initiation signal is sent over XMPP, and the media transfer takes place in a peer-to-peer manner or over media relays.
- **Multi-user chat**: MUC is a flexible, multiparty communication exchange extension for XMPP. Here multiple users can exchange information in a chat room or channel. Support for strong chat room controls is also provided, which enables the banning of users and updation of chat room moderators.
- **Pub–sub**: This provides publish–subscribe functionality to XMPP by proving alerts and notifications for data syndication, vibrant presence, and more such features. Pub–sub enables XMPP clients to create topics at a pub–sub service and publish/subscribe to them.
- **BOSH**: It stands for bidirectional streams over synchronous HTTP. This is an HTTP binding for XMPP (and other) traffic. BOSH incurs lower latencies and lesser network bandwidth usage by doing away with HTTP polling. It is mainly used for the XMPP traffic exchange between clients and servers.

Check yourself

XMPP chat server, comparison between XMPP and AMQP

8.4.6 SOAP

SOAP or simple object access protocol is used for exchanging structured information in web services by making use of XML information set formatting over the application layer protocol (HTTP, SMTP) based transmission and negotiation of messages, as shown in Figure 8.21 [21]. This allows SOAP to communicate with two or more systems with different operating systems using XML, making it language and platform independent. The use of SOAP facilitates the messaging layer of the web services protocol stack.

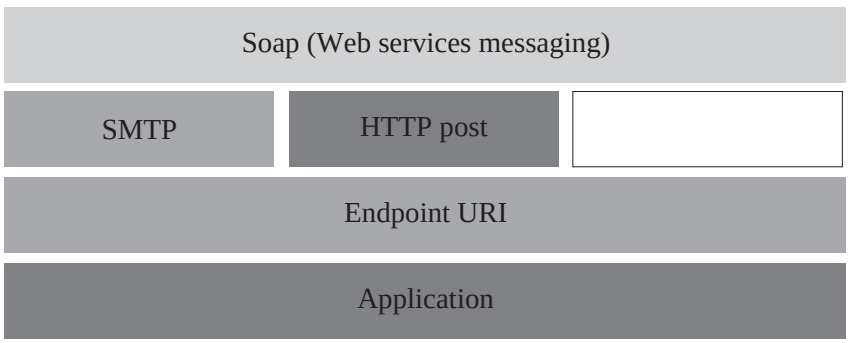


Figure 8.21 A representation of the position of the SOAP API in a stack

A SOAP application can send a request with the requisite search parameters to a server with web services enabled. The target server responds in a SOAP response format with the results of the search. The response from the server can be directly integrated with applications at the requester's end, as it is already in a structured and parsable format. Figure 8.22 illustrates the basic working of SOAP.

SOAP is made up of three broad components: 1) Envelope (which defines the structure of the message and its processing instructions), 2) encoding rules (which handles various datatypes arising out of the numerous applications), and 3) convention (which is responsible for web procedure calls and their responses). This messaging protocol extends the features of neutrality (can operate over any application layer protocol), independence (independent of programming models), and extensibility (features such as security and web service addressing can be extended) to its services. The use of SOAP with HTTP-based request–response exchanges does not require the modification of the communication and processing infrastructures. It can easily pass through network/system firewalls and proxies (similar to tunneling), as illustrated in Figure 8.22. However, the use of XML affects the

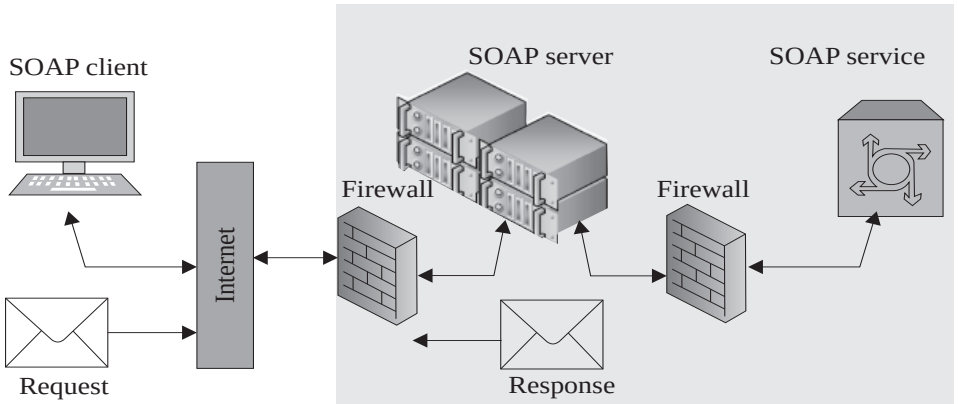


Figure 8.22 Working of SOAP

parsing speed and hence, the performance of this protocol. Additionally, the verbose nature of SOAP is not recommended for use everywhere. The specifications of the SOAP architecture are defined across several layers, such as message format layer, message exchange patterns (MEP) layer, transport protocol binding layer, message processing model layer, and protocol extensibility layer.

Check yourself

Limitations of SOAP, Protocols derived from SOAP

8.4.7 REST

Representational state transfer or REST encompasses a set of constraints for the creation of web services, mainly using a software architectural style [22]. The web services adhering to REST styles are referred to as RESTful services; these services enable interoperability between various Internet-connected devices. RESTful systems are stateless: the web services on the server do not retain client states. The use of stateless protocols and standards makes RESTful systems quite fast, reliable, and scalable. The reuse of components can be easily managed without hindering the regular operations of the system as a whole. Requesting systems can manipulate textual web resource representations by making use of this stateless behavior of REST. RESTful web services, in response to requests made to a resource's URI, mainly responds with either an HTML, XML, or JSON (JavaScript Object Notation) formatted payload. As RESTful services use HTTP for transfer over the network, the following four methods are commonly used: 1) GET (read-only access to a resource), 2) POST (for creating a new resource), 3) DELETE (used for removing a resource), and 4) PUT (used for updating an existing resource or creating a new one). Figure 8.23 represents the REST style and its components.

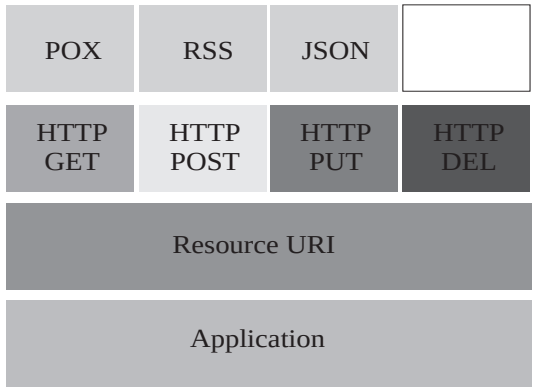


Figure 8.23 A representation of the REST style and its components

REST offers several advantages over regular web-based services. Enhanced network efficiency through the use of REST is ensured by an increase in the performance of interaction between components. Its use also enables a uniform and simple interface, easy live operational modification capabilities, reliability against component and data failures, portability of components, robust scalability, and support for a large number of components.

In REST, requests are used for identifying individual resources. As the resources can be represented in a variety of formats such as HTML, XML, JSON, and others, RESTful services can identify the individual resources from their representations, which allows them to modify, update, or delete these resources. The REST messages contain sufficient information in them to direct a parser on how to interpret the messages. REST client's can dynamically discover all web resources and actions associated with an initial URI. This enhances the dynamicity of applications using REST by avoiding the need to hard-code all clients with the information of the proper structure or dynamics of the web application.

RESTful systems are guided by six general constraints, which define and restrict the process of client–server interactions and requests–responses. These guidelines increase system performance, scalability, reliability, modifiability, portability, and visibility. All RESTful systems have to adhere to these six guidelines strictly:

(i) **Statelessness**: The statelessness of the client–server communication prevents the storage of any contextual information of the client on the server. Each client request has to be self-sufficient in informing its responders about its services and session state. This is done by including the possible links for new state transitions within the representation of each application state. Generally, upon detecting pending requests, the server infers that the client is in a state of transition.

(ii) **Uniform Interface**: Each part or component of a RESTful system must evolve independently as a result of the decoupling of architectures and its simplification.

(iii) **Cacheability**: The responses have to be implicitly, or in some cases, explicitly clear on whether they have to be cached or not. This helps the clients in retaining the most updated data in response to requests. Caching also reduces the number of client–server interactions, thereby improving the performance and scalability of the system as a whole.

(iv) **Client–server Architecture**: The user–interface interactions should be separate from data storage ones. This would result in enhanced portability of user interfaces across multiple platforms. The separation also allows for the independent evolution of components, which would result in scalability over the Internet across various organizational domains.

(v) **Layered System**: The client in RESTful services is oblivious to the nature of the server to which it is connected: an end point server or an intermediary server. The use of intermediaries also helps in improving the balancing of load and enhancing security measures and system scalability.

(vi) **Code on Demand**: This is an optional parameter. Here, the functionality of clients can be extended for a short period by the server. For example, the transfer of executable codes from compiled components.

Check yourself

Difference between REST and SOAP, evolution of REST

8.4.8 WebSocket

Websocket is an IETF (Internet Engineering Task Force)-standardized full-duplex communication protocol. Websockets (WS), an OSI layer seven protocol, enables reliable and full-duplex communication channels over a single TCP connection [23]. Figure 8.24 shows the position of a websocket layer in a stack. The WS relies on the OSI layer 4 TCP protocol for communication. Despite being different from the HTTP protocol, WS is compatible with HTTP and can work over HTTP ports 80 and 443, enabling support for network mechanisms such as the HTTP proxy, which is usually present during organizational Internet accesses through firewalls.

WS enables client–server interactions over the Web. Web servers and clients such as browsers can transfer real-time data between them without incurring many overheads. Upon establishment of a connection, servers can send content to clients without the clients requesting them first. Messages are exchanged over the established connection, which is kept open, in a standardized format. Support for WS is present in almost all modern-day browsers; however, the server must also include WS support for the communication to happen.

The full-duplex communication provided by WS is absent in protocols such as HTTP. Additionally, the use of TCP (which supports byte stream transfers) is also

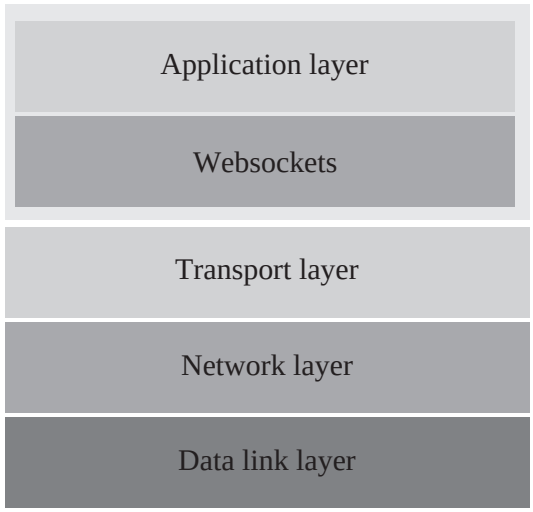


Figure 8.24 A representation of the position of websockets in a stack

enhanced by enabling it to provide message stream transfers using WS. Before the emergence of WS, comet channels were used for attaining full-duplex communication over port 80. However, comet systems were very complicated and incurred significant overheads, which made their utility limited for constrained application scenarios mainly associated with IoT.

Websocket (WS) and websocket secure (WSS) have been specified as uniform resource identifier (URI) schemes in the WS specification, which are meant for unencrypted and encrypted connections, respectively. The WS handshake process and the frames can be quickly inspected using browser development tools.

Operational Principle

A client initiates the WS connection process by sending a WS handshake request. In response, a WS server responds with a WS handshake response. As the servers have to incorporate both HTTP and WS connections on the same port, the handshaking is initiated by an HTTP request/response mechanism. Upon establishment of a connection between the client and server, the WS communication separates as a bi-directional protocol that is non-conformant with the HTTP protocol. The WS client sends an *update header* and a *sec-websocket-Key header*, which contains base64 encoded random bytes. The server responds to the client's request using a hash of the key included in the *Sec-WebSocket-Accept header*. This allows the WS to overcome a caching proxy's efforts to resend previous WS communication. A fixed string, 258EAFA5-E914-47DA-95CA-C5AB0DC85B11, is appended to the undecoded value from the *Sec-WebSocket-Key header* by a hashing function using the SHA (secure hash algorithm)-1, which is finally encoded using base64 encoding. Once the WS full-duplex connection is established, minimally framed data (small header and a payload), which may be data or text, can be exchanged. The WS transmissions or messages can be further split

into multiple data frames whenever the full message length is not available during message transfer. This feature is occasionally exploited to include/multiplex several simultaneous streams, using simple extensions to the WS protocol. This multiplexing avoids the monopoly of a single large payload over the WS port.

Check yourself

Difference between regular client–server sockets and websockets

8.5 Identification Protocols

The surge of IoT devices and Things which are connected over the Internet, makes it significantly hard to identify each device securely. The number of connected things is rising exponentially; with this rise the need to design and develop protocols that can provide unique and distinguishable identifiers to so many Things becomes overwhelming. However, unified global efforts have come up with certain solutions to address the challenges regarding identification of Things, which keep on updating from time to time. Some of the commonly encountered ones are EPC, uCode, and URIs. This section outlines the various nuances associated with each of these methods.

8.5.1 EPC

EPC or the electronic product code identification system was designed to act as a universal identifier and provide unique identities accommodating all physical objects in the world [24]. The open standard and free EPCglobal Tag Data Standard defines the EPC structure. The official representation of EPC is an URI (uniform resource identifier) and referred to as the *pure identity URI*. Figure 8.25 illustrates the standard EPC representation. This representation is used for referring to physical objects in communicating information and business systems and application software. The standard also defines representations for EPC identifiers: tag encoding URI formats and formats for binary EPC identifier storage. In systems such as passive RFIDs that generally have low memory, the EPC binary identifier storage format plays a crucial role. The standard also provides EPC encoding and decoding rules to use URI and binary formats interchangeably seamlessly. Being a very flexible framework, external support for various coding schemes such as those used with barcodes is also possible with EPC. The EPC standard currently supports EPC identifiers, general identifiers, and seven types of identification keys from the GS1 Application Identifier system. As the EPC is not designed to be restricted for use only with RFID data carriers, the data carrier technology-agnostic behavior of EPC is further enhanced by the *pure identity URI*.

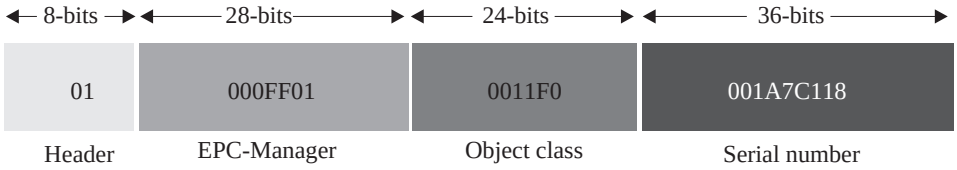


Figure 8.25 The EPC representation

8.5.2 uCode

Another identification number system, the uCode is designed to uniquely identify real-world things and objects whose information is digitally associated with the uCode system [25]. The uID center in Japan provides support for the uCode system. The uCode system can be used with any application, business processes, and technology (RFID, barcodes, matrix codes). uCode is application and technology independent and uses 128 bit codes for uniquely tagging/naming physical objects. The uCode provides 3.4×10^{38} unique codes for individually tagging objects. These features make uCode a crucial enabling technology for IoT. Figure 8.26 represents the working of uCode tags and its various stakeholders.

The uCode tags are generally grouped into five categories: 1) print tags, 2) acoustic tags, 3) active RF tags, 4) active infrared tags, and 5) passive RFID tags. In contrast to other identification systems, the uCode system has the following distinct features:

(i) It does not display product types, albeit it identifies individual objects. Existing codes identify products by individual vendors, making the possibility of identifier tag reuse a possibility, which is avoided in the uCode system.

(ii) In addition to physical objects, the uCode can be associated with places, concepts, and contents, enabling this system to identify such items universally.

(iii) Being application and business agnostic, the uCode system can be used across industries and organizations. The system provides a unique identification number, which does not carry any meaning or information about the tagged object/item. This enables the same system to be used seamlessly across organizations, industries, and product types.

(iv) uTRON, a ubiquitous security framework, which is incorporated with the ubiquitous ID architecture of the uCode system, makes it entirely secure and enables information privacy protection.

(v) The tag agnostic nature of the uCode system makes it possible for various systems such as RFIDs, and barcodes to store uCode information. This makes uCode highly ubiquitous and pervasive.

(vi) The uCode represents pure numbers and is devoid of any meaning or information related to the tagged item/object. This makes the reassignment of uCode tags quite robust and straightforward.

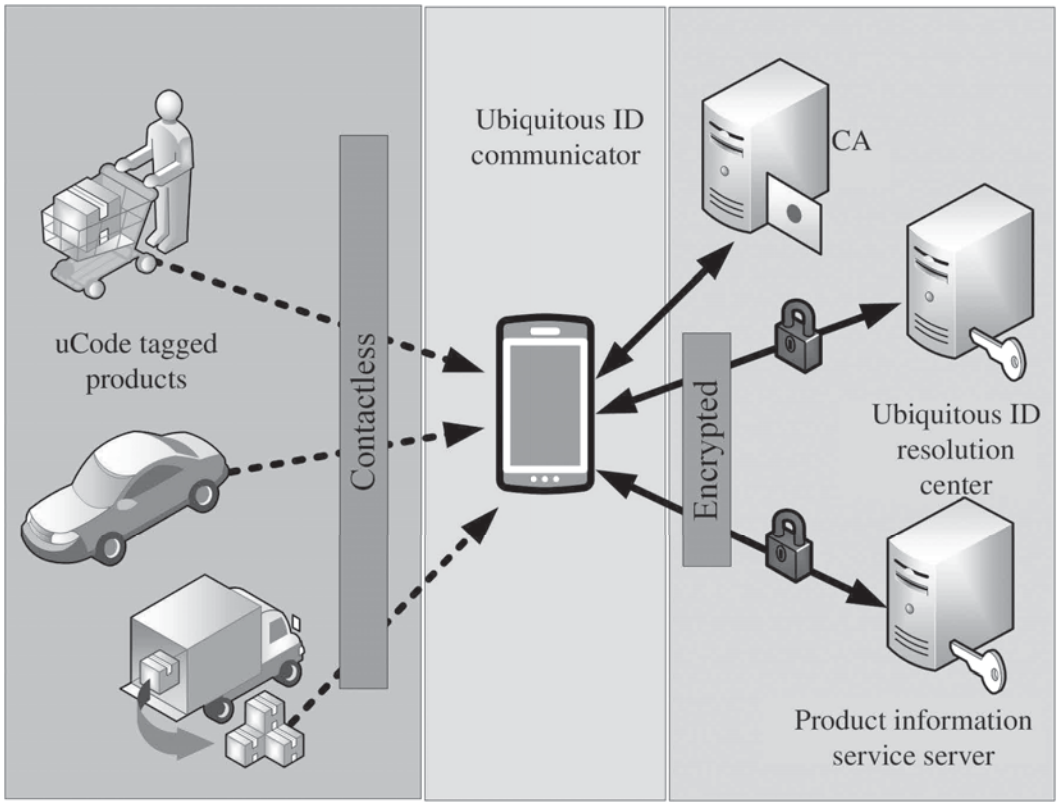


Figure 8.26 The operation of an uCode tag system

The ubiquitous ID architecture of the uCode system is made up of five distinct components: 1) uCode, 2) uCode tags, 3) ubiquitous communicators, 4) uCode resolution server, and 5) uCode information server. The operational process of reading a uCode is as follows:

(i) uCode tags are read using mobile phone cameras to identify the ucode.

(ii) An inquiry about the uCode is sent to the uCode resolution server from the mobile phone over the Internet.

(iii) The uCode resolution server returns information about the uCode to the mobile phone. The returned information contains the source of the read uCode information.

(iv) The ubiquitous communicator then acquires the contents and service information from the information providing source of the read uCode.

Just like the Internet DNS resolution mechanism, the uCode resolution system is hierarchically constructed. The three-tiered uCode resolution hierarchy has the root server at the top level. The uID center in Japan maintains the root server. The next level, the top level domain (TLD), is situated below the root server. As of now various

TLD servers are located around the globe in Japan, Finland, and a few other countries. Finally, the second level domain (SLD) is at the bottom of the hierarchy, below the TLD. The TLD and SLD servers are not restricted and can be added to the existing system.

Check yourself

Differences between EPC and uCode, Limitations of EPC, Limitations of uCode

8.5.3 URIs

One of the most common identifiers in use is the uniform resource identifier (URI). [26] The URI is used to identify individual resources only by using character strings distinctly. As with other protocols, the uniformity of this protocol is ensured by an agreed-upon set of syntax rules. These rules also allow for extensibility through the incorporation of separate hierarchical naming schemes such as "http://". URIs enable interaction with network-based resource representations through specific protocols, especially over the WWW. Some terms commonly derived from URIs are URLs and URNs. URLs or uniform resource locators are very commonly encountered during resource search over the Web or a network. URLs are generally referred to as web addresses and specify the location as well as the access mechanism for a remote resource. For example, "http://www.abc.xz/home/index" denotes the location of the resource at "/home/index", which is hosted at the domain "www.abc.xz", and can be accessed using HTTP. A less encountered form of URIs is the uniform resource name (URN), which identifies resources in particular namespaces only. URNs were initially designed to complement URLs. However, unlike URLs, URNs only identify resources and do not provide the location or method to access the identified resource. Figure 8.27 shows the typical URI format.

Figure 8.27 The representation of an URI link

Check yourself

Difference between URI, URL, and URN, Advantages of URI over URL and URN, Limitations of URI

8.6 Device Management

The need for device management protocols is vital given the rising number of applications of IoT in various application areas spread across the globe. In most of the cases, it is not possible to manage these devices or change their settings manually. Toward this goal, much work is being pursued in the domain of remote device management. We outline two of the most well-known device management protocols in this section.

8.6.1 TR-069

Owing to the rising need for remote management of customer premises equipment (CPE), the Broadband Forum defined the technical specifications for the application layer protocol for CPE over IP networks; these specifications are referred to as Technical Report 069 or TR-069 [27]. The TR-069 mainly focuses on the auto-configuration of Internet-connected devices using auto configuration servers (ACS). Within the premises of this report, the CPE WAN management protocol (CWMP) outlines the various support functions for CPE, which encompasses software and firmware management, status and performance report management, diagnostics, and auto-configuration. CWMP, a primarily SOAP/HTTP-based bi-directional protocol, which is also text-based, provides communication and management support between CPE and servers within a single framework. Devices connecting over the Internet such as routers, gateways, and end devices such as set-top boxes and VoIP devices fall under its purview. Figure 8.28 shows the main components of TR-069 and their relations between each other.

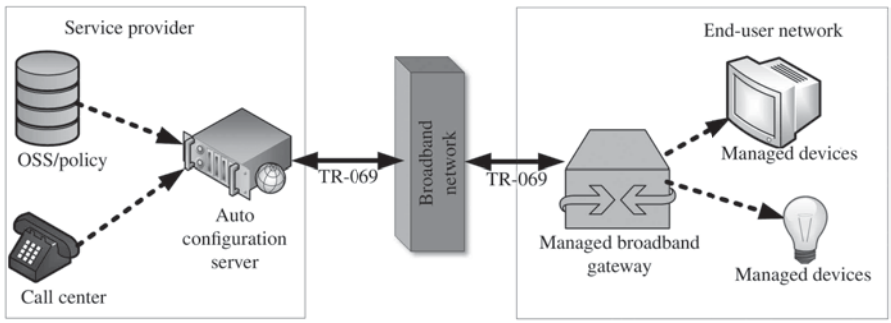


Figure 8.28 The various components of TR-069 and their inter-relations

The various functionalities of this protocol are as follows:

(i) The commands between the CPE and the ACS during provisioning sessions are either HTTP or HTTPS based, where the ACS is the server and the CPE are clients.

(ii) The provisioning session is responsible for the communications and operations between the CPE and ACS.

(iii) Session initiation is performed by the CPE through an "inform" message, to which the ACS indicates its readiness using an "inform response".

(iv) In the subsequent stage, the CPE transmits orders to the ACS, which is invoked using a "transfer complete" message. An empty HTTP-request completes the transmission from the CPE to the ACS.

(v) In response to the empty HTTP request, the HTTP response from the ACS to the device contains a CWMP (CPE WAN management protocol) request. An empty HTTP-response from the ACS indicates the completion of pending orders.

(vi) Information security during transmission (login, password, and others) is handled using HTTPS and ACS certificate verification. Authentication of CPE is done based on a shared secret key between the CPE and ACS.

(vii) A time limit of 30 seconds is imposed on the start of the provisioning session after receiving device confirmation.

Points to ponder

The use of TR-069 for remote management of home networked devices and terminals is endorsed by various forums such as Home Gateway Initiative (HGI), Digital Video Broadcasting (DVB), and WiMAX Forum.

Check yourself

Security risks of CWMP, Data model of CWMP, multi-instance object handling

8.6.2 OMA-DM

The open mobile alliance (OMA) device management (DM) protocol is specified by the OMA working group and the data synchronization (DS) working group for remote device management of mobile devices, including mobile phones and tablets [28]. The management functions include provisioning, device configuration, software upgrades, fault management, and others. On the device end, any or all of these features may be implemented. The OMA-DM specification is designed for constrained devices with limited bandwidth, memory, storage, and processing. Data exchanges

take place through SyncML, which is a subset of XML. OMA-DM supports both wired as well as wireless data transport (USB, RS-232, GSM, CDMA, Bluetooth, and others) over transport layer protocols such as WAP, HTTP, or OBEX.

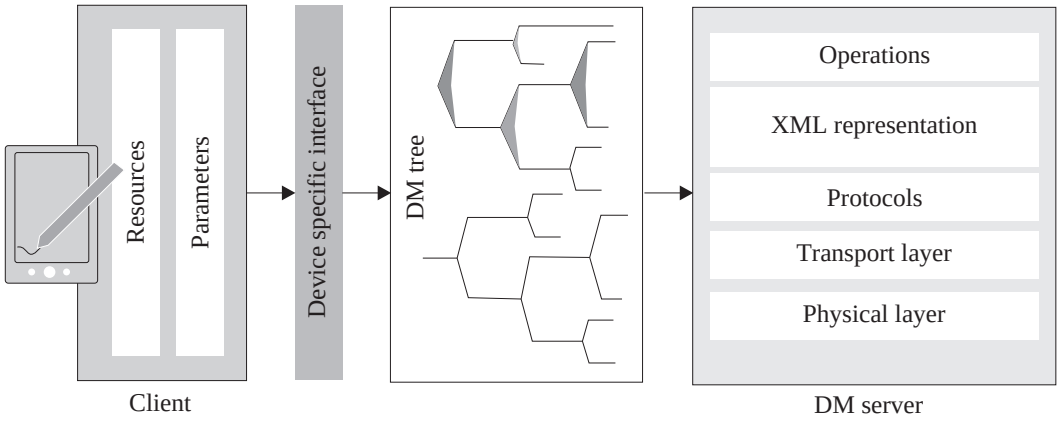


Figure 8.29 Communication between an OMA-DM client and a server

The OMA-DM follows a request–response communication model. The OMA-DM server asynchronously initiates the communication with the end device/client, which is generally in the form of a notification or alert message through WAP push or SMS. The client is meant to execute the command received from the server and reply with a message. More significant messages are generally broken down into chunks before transmission to the client. In terms of information security, authentication methods are built-in in this protocol, which prevents a client and a server from communicating until proper validation. Figure 8.29 shows the communication between a client and a server in OMA-DM.

Check yourself

Security mechanisms in OMA-DM commands

8.7 Semantic Protocols

The semantic protocols for IoT, which is a rapidly upcoming domain, focus on the meaning and logic behind data connectivity and formats. Examples include JSON-LD and the Web Thing model. Primarily designed to be cross-operable and modular, these protocols enhance the robustness and utility of IoT by incorporating the reach of the Web. As an example, the integration of semantic protocols such as JSON-LD with the Web Things model gives rise to the Semantic Web. The chapter on interoperability in this book discusses the challenges and developments in this domain.

8.7.1 JSON-LD

JavaScript object notation for linked data or JSON-LD is a lightweight protocol, which is designed for JSON-based encoding of linked data by seamlessly converting older JSON-based representations of data. The representations of the data are highly human-understandable and highly suitable for RESTful environments and unstructured data over the Web [29]. JSON-LD has an additional resource description framework (RDF) over and above the typical JSON model and is built to be contextual. This feature allows for the interoperability of JSON data over the Web. The contextual linking of the object properties of a JSON document follows a fixed ontology in JSON-LD through strategies such as tagging with a language by or forcibly assigning values to pre-defined groups/bins. Context embeddings in JSON-LD documents can be either direct or through the use of separate file references using HTTP link headers. Linked data allows for the existence of a network of machine-readable and standardized data over the Web, which can be parsed by starting at a singular piece of data and subsequently traversing the embedded links within it; this may lead to different locations across the Web.

A sample JSON-LD schema

```
<script type="application/ld+json">
{
  "@context": "https :// schema.org",
  "@type": "BlogPosting",
  "mainEntityOfPage": {
    "@type": "WebPage",
    "@id": "www.xyz.com"
  },
  "headline": "Hello Readers",
  "description": "This is a test",
  "image": {
    "@type": "ImageObject",
    "url": "www.img1234.com",
    "width": 696,
    "height": 14
  },
  "author": {
    "@type": "Person",
    "name": "abc"
  },
  "publisher": {
    "@type": "Organization",
    "name": "CUP",
     },
  "datePublished": ""
}
</script>
```

Check yourself

Types of linked data, RDF examples

8.7.2 Web thing model

The Web of Things (WoT) is another interoperability-driven initiative for achieving seamless Web-based uniformity for IoT devices. The main driving factor behind this initiative is to develop a unifying application layer-based framework for IoT which can provide URLs for the connected devices over the Web. This initiative aims to transform the traditionally predominant "Web of pages" to "Web of Things". As the current Web-based technologies and IoT-based integrations over the Web are vastly vendor-specific and use proprietary data formats, the cross-utilization of such technologies is seldom flawless. These drawbacks of the present technologies led to the need for a common syntactical vocabulary and API which will be able to induce ad hoc interoperability for IoT. The paradigms, such as "machine-to-machine communication", promote technological overhaul (most often complete technology replacement), without incorporating the existing technologies. In contrast, the WoT paradigm aims to integrate the existing Web with the various applications and systems already in place to fully utilize the infrastructural and technological leverage already present.

The following are the major sub-components of the WoT paradigm:

(i) **Integration Patterns:** Dictates how the Things in IoT connect to the Web. It is mainly composed of three schemes: Direct connectivity, gateway based connectivity, and cloud-based connectivity.

(ii) **Web Things (WT) Requirements:** Provides guidelines and recommendations for handling various constraints and protocol implementations to enhance the seamless interaction between the WoT entities. A typical web-server is referred to as a Web Thing; it should also confirm to these recommendations.

(i) **Web Thing Model:** Data exchange over the WoT ensues once a Web Thing is compliant. Additionally, in order to achieve context-awareness, this specification outlines RESTful web protocol, which has a defined set of payload syntax, data models, and resources. A fully compliant model is referred to as the Extended Web of Things model.

Summary

This chapter provided an outline of various communication technologies that are deemed as core technologies for developing IoT-based solutions. We initially explain the requirements and classification of IoT devices and communication types. We

divide the various communication protocols under six heads based on their usability and functionalities: 1) Infrastructure, 2) discovery, 3) data, 4) identification, 5) device management, and 6) semantic. After this chapter, readers will be able to distinguish between various requirements and constraints associated with these protocols and select the best one amongst them according to their application's requirements.

Exercises

(i) What are the salient features of 6LoWPAN?
(ii) What is a WPAN?
(iii) Describe the addressing types in 6LoWPAN.
(iv) Describe the LOADng routing.
(v) Describe the RPL routing.
(vi) What are the different header types in 6LoWPAN?
(vii) What constitutes a low power lossy network (LLN)?
(viii) What is AMQP? Describe in detail.
(ix) What are the various message guarantees provided by AMQP? Explain each in detail.
(x) List some of the salient features of AMQP.
(xi) What are the frame types in AMQP?
(xii) Differentiate between OPEN, BEGIN and ATTACH frame types in AMQP.
(xiii) Differentiate between DETACH, END, and CLOSE frame types in AMQP.
(xiv) Differentiate between TRANSFER and FLOW frame types in AMQP.
(xv) What are BINDINGS in the context of AMQP?
(xvi) What are the various types of AMQP exchanges? Describe each.
(xvii) What are the popular applications of AMQP?
(xviii) Explain the working of MQTT
(xix) How is MQTT different from HTTP?
(xx) What are the various MQTT methods?
(xxi) What is SMQTT? How is it different from MQTT?
(xxii) List the salient features of MQTT.
(xxiii) List the salient features of XMPP.
(xxiv) Describe the XMPP protocol.
(xxv) Differentiate between structured and unstructured data.
(xxvi) What is XML?

(xxvii) What is BOSH? Explain in detail.
(xxviii) What is CORE? Explain in detail.
(xxix) What is Jingle? Explain in detail.
(xxx) What is Pub–Sub? Explain in detail.
(xxxi) List the significant limitations of XMPP.
(xxxii) List some of the popular uses of XMPP.
(xxxiii) What is CoAP?
(xxxiv) Describe the working of CoAP.
(xxxv) Explain the various messaging modes in CoAP.
(xxxvi) List the salient features of the CoAP protocol.
(xxxvii) What is REST?
(xxxviii) What are RESTful services?
(xxxix) Describe the LOADng protocol.
(xl) What is a DODAG?
(xli) Explain the mechanism of formation of a DODAG in RPL.
(xlii) Explain the working of RPL protocol.
(xliii) Illustrate the salient features of RPL.
(xliv) How is the global instance different from local instances in RPL?
(xlv) What is QUIC? How is the connection latency reduced in QUIC?
(xlvi) What is the purpose of publishing static configuration records in QUIC?
(xlvii) Highlight the various features of uIP.
(xlviii) What led to the development of nanoIP?
(xlix) How is the CCN paradigm different from traditional networking approaches?
(l) How is the Physical Web able to interact with physical objects and locations? What are its advantages?
(li) How is mDNS different from DNS?
(lii) What are some of the commonly used discovery protocols in IoT?
(liii) What features separate MQTT-SN from MQTT?
(liv) What are the main functional differences between transparent and aggregate gateways in MQTT-SN?
(lv) Differentiate between SOAP and REST.
(lvi) How does SOAP enable communication between two syntactically different devices/machines?
(lvii) What are the functional components of SOAP?

(lviii) What are the advantages of using REST over regular web-based services?

(lix) What are the various methods used in REST for transferring data over the network?

(lx) What is statelessness in the context of REST?

(lxi) How are websockets different from simple HTTP?

(lxii) Describe the working of websockets?

(lxiii) What is the functional mechanism for EPC in IoT?

(lxiv) What is uCode and how is it different from EPC?

(lxv) What are the various categories associated with uCode tags?

(lxvi) Describe the uCode resolution system.

(lxvii) What are URIs? How is it used for identifying individual resources?

(lxviii) How is auto-configuration over Internet-connected devices achieved using the auto configuration server?

(lxix) What are the various components of TR-069?

(lxx) What is OMA-DM?

(lxxi) How is OMA-DM functionally different from TR-069?

(lxxii) Differentiate between JSON-LD and XML.

(lxxiii) What is the Web Thing model? Illustrate its strengths and weaknesses.

References

[1] Bormann, C., M. Ersue and A. Keranen. 2014. "Terminology for Constrained Node Networks." https://tools.ietf.org/html/rfc7228.

[2] Annamalaisamy, Vijay. 2019. "Introduction to IoT Constrained Node Networks." https://www.hcltech.com/blogs/introduction-iot-constrained-node-networks.

[3] Postscapes. 2019. "IoT Standards and Protocols." https://www.postscapes.com/internet-of-things-protocols/.

[4] Vasseur, J. P., Cisco Systems, Internet Engineering Task Force (IETF). 2014. *RFC-7102* ISSN: 2070-1721. https://tools.ietf.org/html/rfc7102.

[5] Deering, S., R. Hinden. 1998. "Internet Protocol, Version 6 (IPv6) Specification, IETF." https://tools.ietf.org/html/rfc2460.

[6] Sobral, J., J. Rodrigues, R. Rabelo, K. Saleem, and V. Furtado. 2019. "LOADng-IoT: An Enhanced Routing Protocol for Internet of Things Applications over Low Power Networks." *Sensors* 19(1): 150.

[7] Winter, T. 2012. "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks, IETF." https://tools.ietf.org/html/rfc6550.

[8] Kushalnagar, N., G. Montenegro, and C. Schumacher. 2005. "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals, IETF." https://datatracker.ietf.org/doc/rfc4919/.

[9] The Chromium Projects. 2015. "QUIC, a Multiplexed Stream Transport over UDP." Available online: https://www.chromium.org/quic.

[10] Dunkels, A. 2002. "uIP-A Free Small TCP/IP Stack." *The uIP* 1.

[11] Shelby, Z., J. Riihijärvi, O. Raivio, and O. Mähönen. 2003. "NanoIP: The Zen of Embedded Networking." *IEEE International Conference on Communications*.

[12] Franck, F., S. A. S. Alcatel Lucent. 2016. "Content-centric Networking." U. S. Patent 9,338,150.

[13] The Physical Web, Available online: https://google.github.io/physical-web/.

[14] Cheshire, S. and M. Krochmal. 2013. "Multicast dns." *RFC* 6762, February.

[15] Jeronimo, M. and J. Weast. 2003. *UPnP Design by Example* (Vol. 158). Intel Press.

[16] Banks, A. and R. Gupta. 2014. "MQTT Version 3.1. 1." *OASIS Standard* 29: 89.

[17] Stanford-Clark, A. and H. L. Truong. 2013. "MQTT for Sensor Networks (MQTT-SN) Protocol Specification." International Business Machines (IBM) Corporation version 1: 2.

[18] Shelby, Z., K. Hartke, and C. Bormann. 2014. "The Constrained Application Protocol (CoAP)." *IETF, RFC* 7252.

[19] Vinoski, S. 2006. "Advanced Message Queuing Protocol." *IEEE Internet Computing* (6): 87–89.

[20] Saint-Andre, P., K. Smith, R. Tronçon, and R. Troncon. 2009. *XMPP: The Definitive Guide*. "O'Reilly Media, Inc."

[21] Box, D., D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. 2000. "Simple Object Access Protocol (SOAP) 1.1."

[22] Battle, R. and E. Benson. 2008. "Bridging the Semantic Web and Web 2.0 with Representational State Transfer (REST)." *Web Semantics: Science, Services and Agents on the World Wide Web* 6(1): 61–69.

[23] Fette, I. 2011. "The Websocket Protocol."

[24] Song, B. and C. J. Mitchell. 2008. "RFID Authentication Protocol for Low-cost Tags." In *Proceedings of the First ACM Conference on Wireless Network Security*. ACM. 140–147.

[25] Ishikawa, C. 2012. "A URN Namespace for uCode."

[26] Berners-Lee, T., R. Fielding, and L. Masinter. 1998. "Uniform Resource Identifiers (URI): Generic Syntax."

[27] Broadband Forum, TR-069: CPE WAN Management Protocol. 2018. https://www.broadband-forum.org/download/TR-069_Amendment-6.pdf.

[28] Open Mobile Alliance, OMA Device Management Protocol. 2016. http://www.openmobilealliance.org/release/DM/V1_3-20160524-A/OMA-TS-DM_Protocol-V1_3-20160524-A.pdf.

[29] JSON-LD Working Group. 2018. "JSON for Linking Data." https://www.w3.org/2018/json-ld-wg/.

[30] Guinard Dominique. 2017. "The Web Thing Model", WEB OF THINGS INTEREST GROUP. https://www.w3.org/blog/wotig/2017/01/13/web-thing-model-member-submission/.

Chapter 9

IoT Interoperability

Learning Outcomes

After reading this chapter, the reader will be able to:

- Understand the importance of interoperability in IoT
- List various interoperability types
- Identify the salient features and application scope of each interoperability type
- Understand the challenges associated with interoperability in IoT
- Comprehend the importance of real-world use of interoperability frameworks in IoT

9.1 Introduction

The introduction of billions of connected devices under the IoT environment, which may extend to trillions soon, has contributed massively to the evolution of interoperability. As more and more manufacturers and developers are venturing into IoT, the need for uniform and standard solutions is felt now more than ever before [1]. Figure 9.1 shows the various facets of interoperability in IoT. Interoperability is considered as the interface between systems or products—hardware, software, or middleware—designed in such a manner that the connecting devices can communicate, exchange data, or services with one another seamlessly irrespective of the make, model, manufacturer, and platform.

The urgency in the requirement for interoperability and interoperable solutions in IoT arose mainly due to the following reasons:

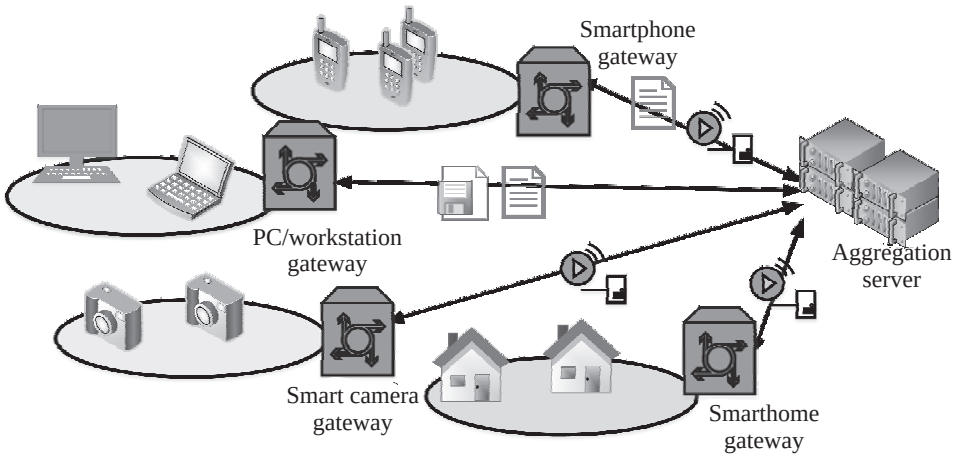


Figure 9.1 An illustration of the various facets of interoperability in IoT

(i) **Large-scale Cooperation:** There is a need for cooperation and coordination among the huge number of IoT devices, systems, standards, and platforms; this is a long-standing problem. Proprietary solutions are seldom reusable and economical in the long run, which is yet another reason for the demand for interoperability.

(ii) **Global Heterogeneity:** The network of devices within and outside the purview of gateways and their subnets are quite large considering the spread of IoT and the applications it is being adapted to daily. Device heterogeneity spans the globe when connected through the Internet. A common syntax, platform, or standard is required for unifying these heterogeneous devices.

(iii) **Unknown IoT Device Configuration:** Device heterogeneity is often accompanied by further heterogeneity in device configurations. Especially considering the global-scale network of devices, the vast combinations of device configurations such as data rate, frequencies, protocols, language, syntax, and others, which are often unknown beforehand, further raise the requirement of interoperable solutions.

(iv) **Semantic Conflicts:** The variations in processing logic and the way data is handled by the numerous sensors and devices making up a typical IoT implementation, makes it impossible for rapid and robust deployment. Additionally, the variations in the end applications and their supported platform configurations further add to the challenges.

The heterogeneity in IoT devices may arise due to several reasons. Some of the common ones are as follows:

- **Communication Potocols:** ZigBee(IEEE 802.15.4), Bluetooth (IEEE 802.15.1), GPRS, 6LowPAN, Wi-Fi (IEEE 802.11), Ethernet (IEEE 802.3), and Higher Layer LAN Protocols (IEEE 802.1)

- **Programming Languages:** JavaScript, JAVA, C, C++, Visual Basic, PHP, and Python
- **Hardware Platforms:** Crossbow, National Instruments, and others
- **Operating Systems:** TinyOS, SOS, Mantis OS, RETOS, NOOBS, Windows 10 IoT Core, and mostly vendor-specific OS
- **Databases:** DB2, MySQL, Oracle, PostgreSQL, SQLite, SQL Server, and Sybase
- **Data Representations:** Comma separated values (CSV), text, rich text format (RTF), open document format (ODF), strings, characters, floating-point values, integer values, and others
- **Control Models:** Event-driven, publish–subscribe, client–server, and others

9.1.1 Taxonomy of interoperability

The significant range of interoperable solutions that has been developed for IoT can be broadly categorized into the following groups:

(i) **Device:** The existence of a vast plethora of devices and device types in an IoT ecosystem necessitates device interoperability. Devices can be loosely categorized as low-end, mid-end, and high-end devices based on their processing power, energy, and communication requirements. Low-end devices are supposed to be deployed in bulk, with little or no chance of getting their energy supplies replenished, depending on the application scenario. These devices rely on low-power communication schemes and radios, typically accompanied by low-data rates. The interface of such devices with high-end devices (e.g., smartphones, tablets) requires device-level interoperability [2].

(ii) **Platform:** The variations in the platform may be due to variations in operating systems (Contiki, RIOT, TinyOS, OpenWSN), data structures, programming languages (Python, Java, Android, C++), or/and application development environment. For example, the Android platform is quite different from the iOS one, and devices running these are not compatible with one another [3].

(iii) **Semantic:** Semantic conflicts arise during IoT operations, mainly due to the presence of various data models (XML, CSV, JSON), information models (°C, °F, K, or different representations of the same physical quantity), and ontologies [4]. There is a need for semantic interoperability, especially in a WoT environment, which can enable various agents, applications, and services to share data or knowledge in a meaningful manner.

(iv) **Syntactic:** Syntactic interoperability is a necessity due to the presence of conflicts between data formats, interfaces, and schemas. The variation in the syntactical grammar between a sender and a receiver of information results in massive stability issues, redundancies, and unnecessary data handling efforts [5]. For

example, a packet from a device has a format as *Header-Identifier-SensorA-SensorB-Footer*, whereas another device from a different manufacturer, but deployed for the same application has the data format as *Header-Identifier-SensorB-SensorA-Footer*. This change in position of sensor A and sensor B in the two packets creates syntactic errors, although they contain the same information.

(v) **Network:** The large range of connectivity solutions, both wired and wireless, at the disposal of developers and manufacturers of IoT devices and components, further necessitates network interoperability. Starting from the networks and sub-networks on the ground, to the uplink connectivity solutions, there is a need for uniformity or means of integrating to devices enable seamless and interoperable operations.

9.2 Standards

Toward enabling IoT interoperability, various technologies have been standardized and are recognized globally for incorporating consistent interoperability efforts worldwide across various industries, domains, and technologies. We list seven of the popular ones in this chapter.

9.2.1 EnOcean

EnOcean is a wireless technology designed for building automation systems, primarily based on the principle of energy harvesting [6]. Due to the robustness and popularity of EnOcean, it is being used in domains such as industries, transportation, logistics, and homes. As of 2012, EnOcean was adopted as a wireless standard under ISO/IEC 14543-3-10, providing detailed coverage of the physical, data link, and networking layers. EnOceanbased devices are batteryless. They use ultra-low power consuming electronics along with micro energy converters to enable wireless communication among themselves; the devices include networking components such as wireless sensors, switches, controllers, and gateways. The energy harvesting modules in EnOcean use micro-level variations and differences in electric, electromagnetic, solar, or other forms of energy to transform the energy into usable energy through highly efficient energy converters. The wireless signals from the batteryless EnOcean sensors and switches, which are designed to be maintenance-free, can operate up to 30 meters in buildings and homes and up to 300 meters in the open. EnOcean wireless sensor modules wirelessly transmit their data to EnOcean system modules, as shown in Figure 9.2.

EnOcean is typically characterized by low data rates (of about 125 kbit/s) for wireless packets that are 14 bytes long. This reduces the energy consumption of the EnOcean devices. Additional features such as the transmission of RF (radio frequency) energy only during transmission of 1s in the binary encoded message further reduce

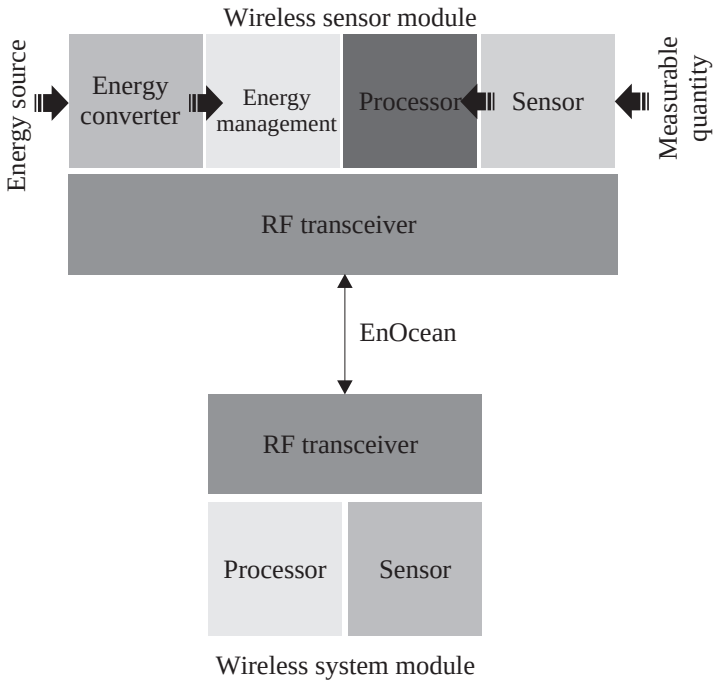


Figure 9.2 A representation of the major constituents of EnOcean devices

the energy consumption of these devices. Frequencies of 902 MHz, 928.35 MHz, 868.3 MHz, and 315 MHz are employed for transmission of messages in this technology.

Check yourself

EnOcean ultra-low power management, self-powered IoT

9.2.2 DLNA

The Digital Living Network Alliance (DLNA), previously known as the Digital Home Working Group (DHWG), was proposed by a consortium of consumer electronics companies in 2003 to incorporate interoperability guidelines for digital media sharing among multimedia devices such as smartphones, smart TVs, tablets, multimedia servers, and storage servers. Primarily designed for home networking, this standard relies majorly on WLAN for communicating with other devices in its domain and can easily incorporate cable, satellite, and telecom service providers to ensure data transfer link protection at either end. The inclusion of a digital rights management layer allows for multimedia data sharing among users while avoiding piracy of data. The consumers in DLNA, which may consist of a variety of devices such as TVs, phones, tablets, media players, PCs, and others, can view subscribable content without any

additional add-ons or devices through VidiPath. Figure 9.3 shows the steps involved in a typical DLNA-based multimedia streaming application. As of 2019, DLNA has over a billion devices following its guidelines globally [7].

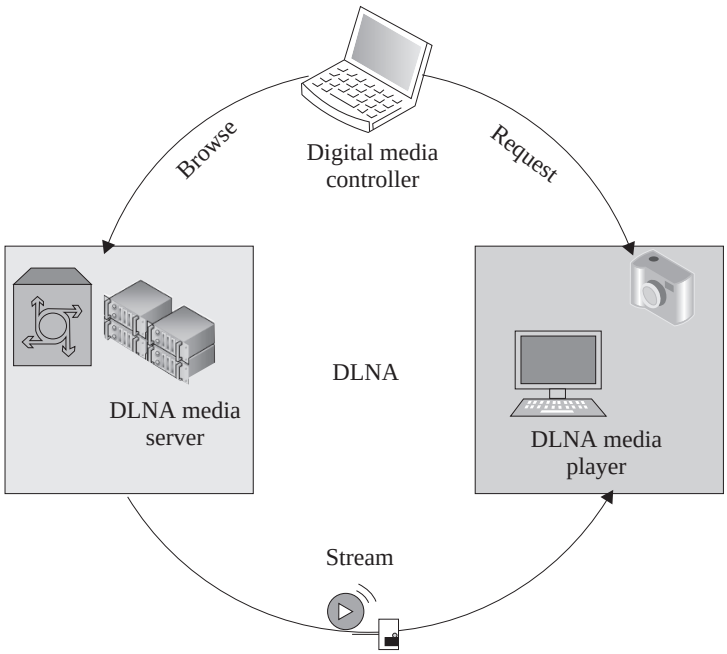


Figure 9.3 A representation of the various roles in a DLNA-based media streaming application

DLNA outlines the following key technological components, which enable interoperability guidelines for manufacturers [7].

(i) Network and Connectivity
(ii) Device and Service Discovery and Control
(iii) Media Format and Transport Model
(iv) Media Management, Distribution, and Control
(v) Digital Rights Management and Content Protection
(vi) Manageability

Check yourself

DLNA Home Network and Infrastructure devices and components, DLNA mobile infrastructure

9.2.3 Konnex

Konnex or KNX is a royalty-free open Home Automation Network (HAN) based wired standard for domestic building and home applications. It relies on wired communication for achieving automation [8]. Wired configurations such as a star, tree, or line topologies can be achieved by using a variety of physical communication technologies involving twisted pair, power line, RF (KNX-RF), or IP-based (KNX-net/IP) ones. KNX evolved from three previous standards: 1) BatiBUS, 2) European Home Systems Protocol (EHS), and 3) European Installation Bus (EIB or Instabus). It has a broad scope of applications in building automation, which involve tasks such as controlling lighting, doors, windows, high-voltage AC (HVAC) systems, security systems, audio/video systems, and energy management. Figure 9.4 represents a typical Konnex-based building network. The KNX facilitates automation through distributed applications and their interaction using standard data types, objects, logical devices, and channels, which form an interworking model. The technology is robust enough to be supported by a wide range of hardware platforms, starting from a simple microcontroller to a sophisticated computer. The requirements of building automation often dictate the hardware requirements.

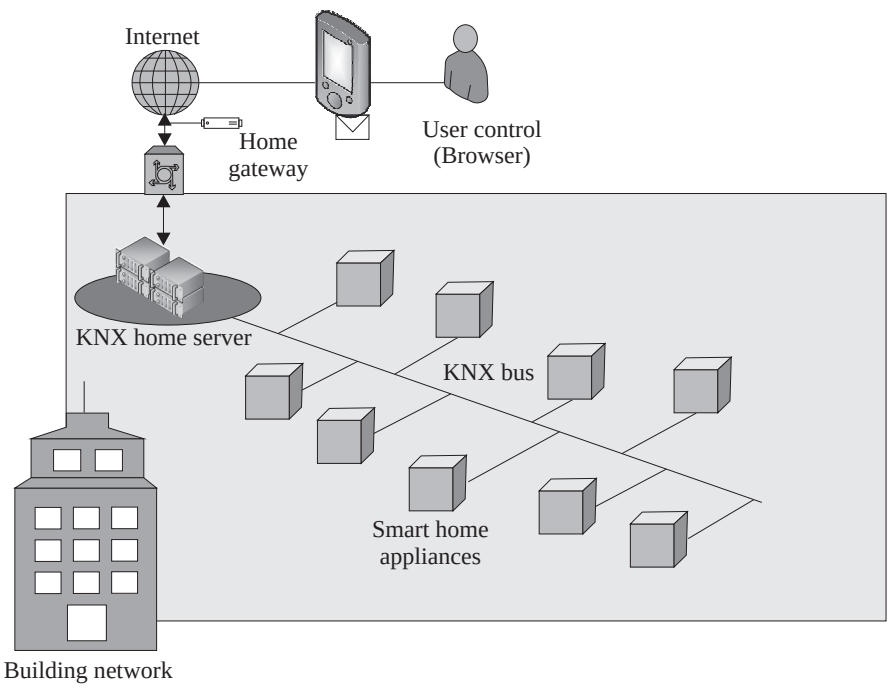


Figure 9.4 A representation of the Konnex network

The KNX architecture consists of sensors (temperature, current, light), actuators (motors, switches, solenoids, valves), controllers (implementable logic), and other

system devices and components (couplers). Typically, the KNX uses a twisted pair bus for communication, which is channeled through the building/home alongside the electrical wiring. Using a 16-bit address bus, KNX can accommodate 57375 devices. A KNXnet/IP installation allows the integration of KNX sub-networks via IP. A system interface component is used for loading application software, system topology, and operating software onto the devices, after which the devices can be accessed over LAN or phone networks. This feature also allows for the centralized as well as distributed control of systems remotely. KNX has three different configuration modes according to device categories.

(i) Automatic mode (A mode): Typically used for auto-configurable devices, and is generally installed by the end users.

(ii) Easy mode (E mode): Devices require initial training for installation, where the configuration is done as per the user's requirements; the device behavior is pre-programmed using E mode.

(iii) System mode (S mode): Some devices generally require specialists to install; the system mode is used for this. The devices do not have a default behavior but can be used for deploying complex building automation systems.

Points to ponder

KNX is an approved standard under International standards (ISO/IEC 14543-3), European standards (CENELEC EN 50090 and CEN EN 13321-1), US standards (ANSI/ASHRAE 135), and China Guobiao (GB/T 20965)

Check yourself

KNX architecture, KNX addressing, KNX use cases

9.2.4 UPnP

The Universal Plug and Play (UPnP) was designed primarily for home networks as a set of protocols for networking devices such as PCs, printers, mobile devices, gateways, and wireless access points. UPnP can discover the presence of other UPnP devices on the network, as well as establish networks amongst them for communication and data sharing [9]. Whenever they are connected to a network, UPnP devices can establish working configurations with other devices. As of 2016, UPnP is managed by the Open Connectivity Forum (OCF). The underlying assumption of UPnP is the presence of an IP network over which it uses HTTP to share events, data, actions, and service/device descriptions through a device-to-device networking arrangement. Device search and advertisements are multicast through HTTP over UDP (HTTPMU) over port 1900. The responses are returned in

a unicast manner through HTTP over UDP (HTTPU). UPnP is based on established protocols and architectures such as TCP/IP protocol suite, HTTP, XML, and SOAP. UPnP is a distributed and open standard. Devices controlled by UPnP are handled by UPnP control points (CPs). The networked UPnP devices are designed to dynamically join networks, obtain IP addresses, advertise its presence and capabilities, and detect the presence and capabilities of other neighboring and networked devices through a process known as zero configuration networking.

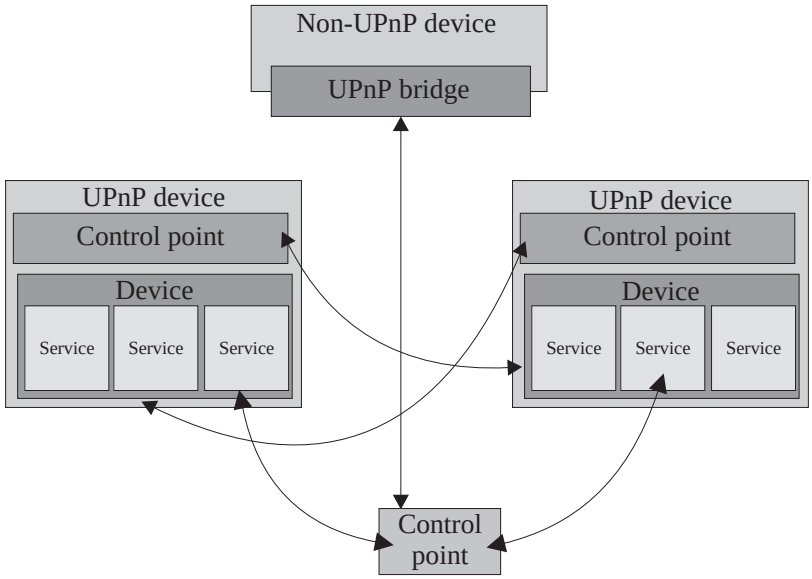


Figure 9.5 A representation of the UPnP operation

UPnP devices are typically characterized by a control point and service(s). The service(s) need to communicate with the control point for further instructions/execution. Figure 9.5 shows a typical UPnP operation. A central control point in a room can be used to control various UPnP services across a home. Non-UPnP devices can be easily integrated with the UPnP services through a bridge.

UPnP supports a range of IP supporting media such as Ethernet, IR, Bluetooth, Wi-Fi, FireWire, and others, without the need for individual device drivers. UPnP, being an OS and language independent protocol, typically uses web browsers for the user interface. Each UPnP device implements a DHCP (dynamic host configuration protocol) client and searches for a DHCP server during its first initiation in the network. These devices can also use a feature known as AutoIP to assign itself an IP address, in case a DHCP server is not available. The UPnP device then discovers the network through the simple service discovery protocol (SSDP), which advertises the device through the CPs (coordination protocols) on the network. The CP then retrieves the device's information through a location URL sent by the device. The device information is in the form of an XML schema using SOAP; it additionally contains

a list of services: commands, actions, and actionable variables and parameters. To the control URL in the description, CPs use control messages to send actions to a device's service. Finally, if a device has a URL for presentation, the CP retrieves the contents, allowing a user to control or view the device and device status.

Check yourself

UPnP device discovery, UPnP protocol, Event notification

9.2.5 LonWorks

LonWorks or local operating network, as it was initially named, is a protocol developed by the Echelon Corp [10]. It was primarily developed for addressing the needs of networked control applications within buildings over physical communication media such as twisted pair, fiber optic cables, powerlines, and RF. The twisted pair uses differential Manchester encoding and has a data rate of 78 kbit/s, whereas the powerline is much slower and can have either 5.4 kbit/s or 3.6 kbit/s depending on the frequency of the power line. This protocol was standardized by ANSI (American National Standards Institute) as early as 1999 when it was known as LonTalk and was used for control networking. This protocol has been used in a variety of deployment areas such as the pneumatic braking system of trains, semiconductor equipment manufacturing, petrol station controls, and as a building automation standard. LonWorks extends backward compatibility support to its legacy installations through an IP-based tunneling standard (ISO/IEC 14908-4). Regular IP-based services can be readily used with LonWorks platforms or installations for UI or control level applications. Figure 9.6 illustrates a typical LonWorks network.

Initially, a LonTalk protocol node could only be installed using a custom-designed IC with an 8-bit processor; this IC was referred to as the "neuron chip". The neuron chip is a system on a chip and is essentially the soul of the LonWorks-based devices. There are two types of neuron chips based on the memory capabilities and packaging: 1) the 3120 and 2) the 3150. Presently, a significant number of LonWorks-based devices use the neuron chip, which is also accessible by general processors by porting to an IP-based or 32-bit chip. A neuron chip has three CPUs, one each for MAC processing, network processing, and application processing. The MAC processor is tasked with CRCs (cyclic redundancy checks), transmitting and receiving messages over the physical media, and confirming message destinations. The network processor deals with addressing, routing, acknowledgments, and other network layer tasks. Finally, the application processor is used for deploying custom applications which typically support 8-bit operations; it can also be used as a communication co-processor for high-end processors. The decoupling of processors based on tasks enables the robust and speedy performance of the neuron chips. Each neuron chip has three memory types available with it: 1) ROM, 2) RAM, and 3) EEPROM. The LonTalk,

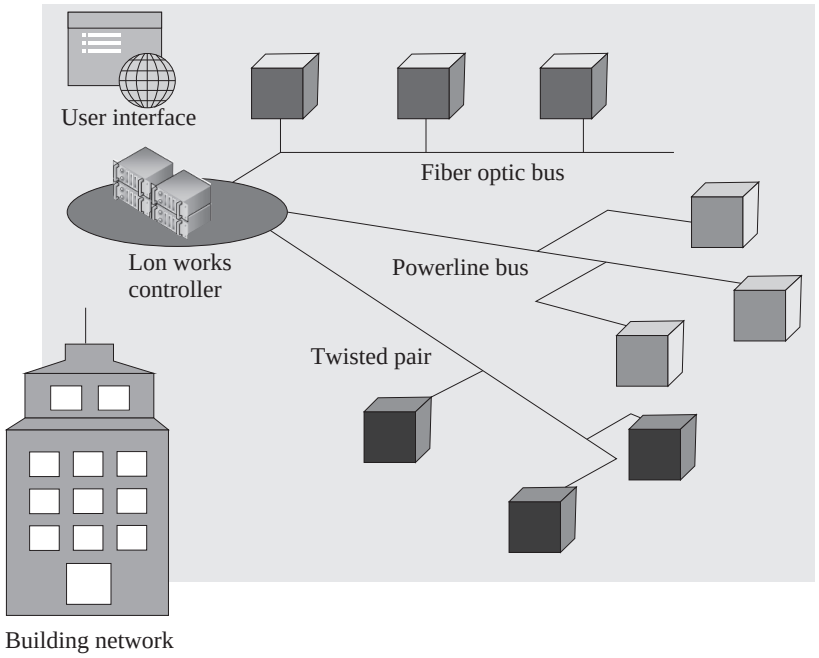


Figure 9.6 A representation of the LonWorks network

along with the OS and I/O libraries are typically programmed in the ROM during manufacturing.

Check yourself

LonWorks addressing, Memory handling by neurons, LonWorks network access

9.2.6 Insteon

Insteon was developed as a home automation technology by Smartlabs in 2005 and marketed under its subsidiary Insteon. Insteon enables interoperability and automation among household devices such as lights, switches, thermostats, motion and gas sensors, and others through RF or powerline communication [11]. Insteon-connected devices act as peers and can independently perform network-based functions such as message transmission and reception by using a dual mesh network topology. These devices operating over the powerline have a frequency of 131.65 kHz; the devices use binary phase shift keying (BPSK), with a minimum receive signal level of 10 mV. In contrast, Insteon devices using RF operates over a frequency of 915 MHz; these devices use frequency shift keying for communication and Manchester codes for encoding data with a data rate of 4.56 kbit/s over ranges of approximately 120 m without obstructions. Figure 9.7 shows a typical home-based Insteon network.

Insteon networks can have 16 million+ unique IDs and can support 65,000+ devices. Each of these devices has a built-in engine, which has an 80 byte RAM and a 3 kbyte ROM. Application-specific requirements of Insteon devices such as lights and switches require 256 bytes of RAM and EEPROM, and 7 kbytes of flash memory. Insteon devices have an average data rate of 180 bit/s, using which a standard message of 10 bytes or extended message of 24 bytes is transmitted. Each Insteon message can accommodate up to 14 bytes of user data and contain a two-bit field meant for counting hops. Message originating nodes initialize this field value to 3, which is decremented by the number of times a node repeats the message during its transmission. Upon receiving a message, each device performs error detection and correction. Retransmission of erroneous messages in this manner enhances the reliability of Insteon technology. All devices transmit the same message simultaneously using PSK to ensure synchronicity with the powerline frequency. This ensures message integrity and strengthens the signal over the powerline.

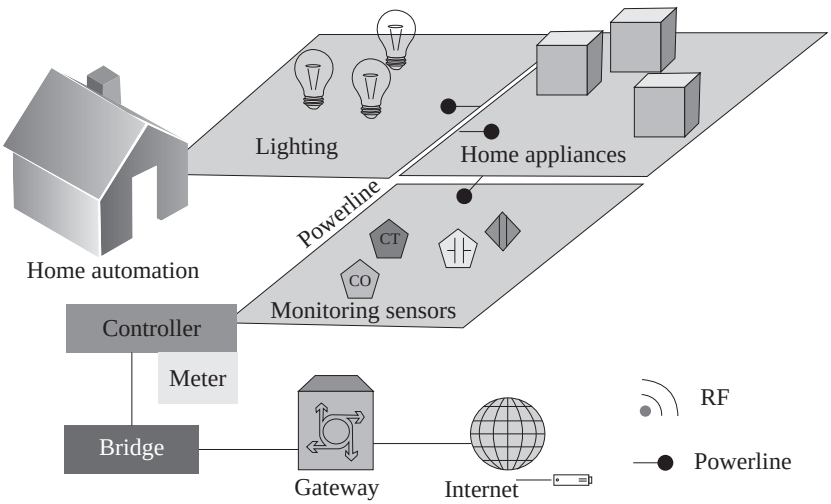


Figure 9.7 A representation of an Insteon network

The dual mesh/ dual band network topology of Insteon is named so mainly because, during operations over the RF band, interferences are mitigated by transmitting data over the powerline, and vice versa. As this is a peer-to-peer network, it can operate without the need for central controllers. Central controllers can be integrated with this technology to extend control operations over smartphones and tablets. As a security measure to avoid hijacking a neighbor's Insteon devices, Insteon requires users to have physical ownership of the devices they want to connect to their network and the respective device IDs (which is unique and similar to a MAC ID). The inbuilt firmware on the devices prevents Insteon devices from forming connections and identifying themselves to other devices until a button is physically pressed on them during their installation.

Points to ponder

Legacy Insteon chipsets are interoperable with X10 powerline messaging, but with reduced functionalities. Present-day initiatives have incorporated compatibility for certain functionalities of Insteon with Amazon Echo, Microsoft Cortana, Apple Watch, and the Google-owned NEST.

Check yourself

Insteon installation, Insteon functionalities with other platforms, Insteon use cases

9.2.7 X-10

The X-10 protocol was developed by Pico Electronics (Scotland) in 1975 as a means of achieving communication and automation among household devices over powerlines. It was one of the first home automation technologies, and yet it remains one of the most widely used even in the present day [12]. Data and controls are encoded as brief RF bursts for signaling and control over the powerlines. Household electrical wiring is used for sending data between X-10 devices by encoding it over a 120 kHz carrier frequency, which is transmitted during zero crossings of 50–60 Hz consumer AC signals as RF bursts, one bit per crossing. The data is made up of an address and a command between the controller and the device. X-10 signals are restricted within the power supply of a house/network using inductive filters, which act as attenuators. Coupling capacitors or active repeaters for X-10 are used to facilitate signal transmission over multiphase systems. An X-10 system can have 256 possible addresses, which is made up of 4-bit house codes (numbered from alphabets A to P), 4-bit unit codes, and finally, a 4-bit command. More than one house code can be simultaneously called within a single house. X-10 devices may be either one-way or two-way. One-way devices are typically very cheap and can only receive commands, whereas two-way devices are more expensive and can send as well as receive commands. These two-way devices are generally used as controllers. Figure 9.8 represents a typical X-10 setup and controller, which allows a user to connect to and control a variety of appliances and devices at home.

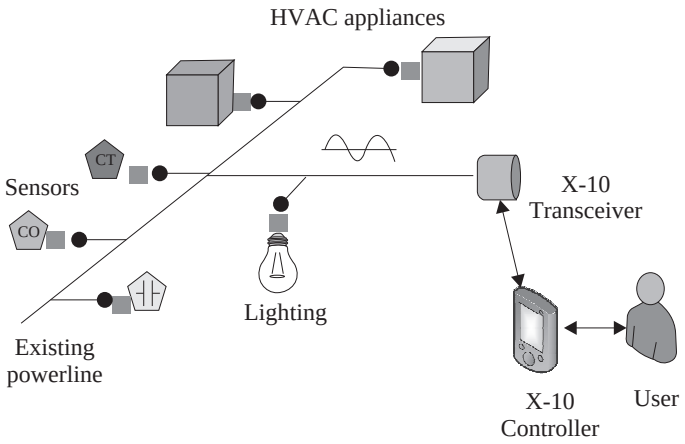


Figure 9.8 A representation of the X-10 network

A bit value of 1 is represented by a 1 ms burst of 120 kHz for a typical 60 Hz AC powerline at the zero crossing. The absence of a pulse follows this bit value. A 0 bit is represented by the absence of a 120 kHz burst at the zero crossings, followed by a pulse. The data rate for the X-10 protocol is typically around 20 bit/s, which includes retransmission time and control signals. Due to the meager data rates, X-10 commands are kept simple and have limited functionalities such as on/off. Each data frame in X-10 is transmitted twice, which although incurs redundancy also allows for reliable data transmission over noisy channels. A new command over the powerline is separated by at least six clear zero crossings from the previous command. An RF protocol is also defined under X-10 to accommodate wireless remotes and switches. This protocol operates over a 310 MHz channel in the US and a 433.92 MHz channel in Europe. The wireless data packets from X-10 devices communicate to a radio receiver, which acts as a bridge between the wireless devices and the powerline-based X-10 devices.

Points to ponder

A typical X-10 command may look like: "select code A5", which is followed by the command for that device such as "turn on/off". This command signals an X-10 device with address A5 to turn on/off.

Check yourself

X-10 applications, X-10 use cases, X-10 addressing

9.3 Frameworks

Similar to the standards, there has been a rise in universal interoperability frameworks. These frameworks span across platforms, devices, technologies, and application areas. We discuss five of the most popular interoperability frameworks in this chapter.

9.3.1 universAAL

UniversAAL is an open-source software framework designed for enabling runtime support in distributed service oriented environments comprising mainly of the system of systems [13]. This framework extends semantic interoperability by sharing compatible models/ontologies with service consumers such as mobile devices, embedded systems, and others. Managers, along with middleware, collectively form the universAAL platform. These managers are considered low-level applications and provide functional APIs (application programming interfaces) to final applications utilizing universAAL. Hardware such as sensors and actuators connect to the universAAL platform through exporters, which are specific for different technologies such as Zigbee, Konnex, and others.

The universAAL middleware is tasked with core coordination among the nodes within a peer-to-peer connectivity layout, referred to as the uSpace. The sharing of various universAAL communication semantics such as the shared ontological model, context, service interactions, and user interactions is performed in this uSpace, which creates a logical environment for enabling communications irrespective of the underlying device, technology, or network. The services or set of services run by a universAAL application is human/user-centric. A coordinator node is responsible for creating each uSpace, and subsequently keeping track of its status, and adding/deleting new nodes to it.

A container is responsible for supporting the middleware and the code and building rules under different environments such as Java environments, Android environments, and other embedded systems. As of now, universAAL supports only Bundles in OSGi (for embedded systems) and APKs in Android. The peering part handles various instances of middleware communication and interconnections. A UPnP-like connector is tasked with the discovery of universAAL nodes and multi-technology bridging.

The most crucial aspect of the middleware is the communication, which provides the logic for semantic information flow between the peers. This flow is enabled through purpose-specific buses to which various applications connect irrespective of the device, container, or peering technology. Buses have been defined for purposes such as context, service and user interactions, internal strategy handling, semantics, peer matchmaking, and others. The ontology model, encryption, and message parsing through message serialization are defined in a representation model. A uSpace

gateway handles communication across different uSpaces by handling message exchanges and authentication between them.

Check yourself

Composition of universAAL ontologies, context sharing in universAAL, service handling in universAAL, user interaction in universAAL

9.3.2 AllJoyn

The AllJoyn is an open-source software initiative proposed by Qualcomm in 2011 that allows devices within this framework to communicate with other devices near its vicinity [14]. The flexible AllJoyn framework encourages proximal connectivity and even has the option of including cloud connectivity to it. It was subsequently signed over to the Linux Foundation under the aegis of the AllSeen Alliance, which was formed primarily to promote IoT interoperability. Major global consumer electronics corporations such as LG, Sony, Panasonic, Haier, Cisco, HTC, Microsoft, and many others are part of the AllSeen Alliance. In 2016, AllJoyn merged with IoTvity and joined the Open Connectivity Forum (OCF), which allowed various open-source projects to include it within their framework. The AllJoyn and IoTvity technologies are currently interoperable and backward compatible with one another.

The open-source AllJoyn software framework enables interoperability amongst connected devices and applications, resulting in the creation of dynamic proximal networks using a D-Bus message bus. The software framework and the core components of the system seamlessly discover, communicate, and collaborate irrespective of platform, product, brand, or connection types, although within the limitations of the collaborating brands only (which is quite large). As of now, communication is only through Wi-Fi, but it includes devices concerning smart homes, smart TVs, smart audio, gateways, and even automotive devices.

The AllJoyn framework follows a client–server model. The clients are often referred to as "consumer" and the server as the "producer". For example, in a smart home environment, a proximity sensor senses the presence of humans in the house and switches on appliances based on the occupancy of the house. If the house is empty, the appliances are turned off. Here, the proximity sensor is the consumer, and the appliance (maybe, a light) is a producer. In this framework, each producer is characterized by an introspection file, which is an XML schema of the producer's capabilities and functionalities. The requests for each producer are based on its introspection file. The framework's capabilities can be extended by incorporating other protocols with it through bridging. Complex functionalities such as simultaneous audio streams to multiple devices can also be executed using this framework.

Some of the core services provided by the AllJoyn framework include onboarding services (attaching a new device to the framework's Wi-Fi network), configuration service (configuring device attributes such as languages, passwords, and names), notification service (text/view-URL based audio and image notifications), control panel (remote app-based control of all connected devices), and common device model service (unified monitoring of IoT devices irrespective of vendors or manufacturers).

Check yourself

Device XML schema, AllJoyn source code, AllJoyn products and services

9.3.3 IoTivity

Similar to the AllJoyn, IoTvity is an open-source project which is sponsored by the OSF (Open Science Framework) and hosted by the Linux Foundation [15]. This framework was developed to unify billions of IoT devices, be it wired or wireless, across the Internet, to achieve a robust and interoperable architecture for smart and thin devices. IoTvity is interoperable and backward compatible with AllJoyn. This framework can connect across profiles ranging from consumer, health, enterprise, industrial, and even automotive.

The IoTvity framework uses CoAP at the application layer and is not bothered with the physical layer requirements of devices. However, the network layer of the connecting devices must communicate using IP. The connectivity technologies of IoTvity connecting devices can consist of Wi-Fi, Ethernet, Bluetooth Low Energy, Thread, Z-Wave, Zigbee, or other legacy standards.

The IoTvity architecture supports the following core functionalities: Discovery (finding devices in one's vicinity and offering services to them), data transmission (standardized message transmission between devices), device management, and data management.

Under the purview of the resource-bounded context in IoTvity's OCF (Open Connectivity Foundation) Native Cloud 2.0 framework, which aims to utilize and enhance the benefits of IoT for companies fully, a resource hosting server has to be accessible through the OCF's native cloud. A resource is an object, which consists of a type, associated data, resource relationships, and operational methods. A server can only publish discoverable resources (which can be found by other connected clients), once it is successfully connected, authenticated, and authorized. Clients can discover resources, either based on the resource type or server identifiers.

Check yourself

IoTvity services and functionalities, IoTvity source code, IoTvity use cases

9.3.4 Brillo and Weave

Google introduced its IoT framework in 2015 as Project Brillo. It is primarily designed as an operating system for IoT devices; it can be considered as a skinny version of Android, having a minimal footprint [16]. Brillo is currently Wi-Fi and BLE (Bluetooth low energy) enabled, with ongoing efforts for the addition of further low-power solutions such as Thread. As the framework is Android-based, it extends scalability in terms of rapid acceptance and portability. Brillo extends interoperability amongst devices and platforms from various vendors and manufacturers.

The underlying communication layer of Brillo is known as Weave. Weave provides a common language for devices such as phones to talk to the cloud. The Weave is the communications layer by which Things can talk to one another. It provides a common language so that devices can talk to one another, with the cloud and the phone. The Brillo framework extends interoperability and uniformity over a diverse range of applications such as smart farming devices, smart homes, smart parking systems, and others. Weave devices communicate over TCP or UDP, using either IPv4 or IPv6. Interestingly, Weave is an information schema for devices that defines device types, functionalities, and modes of communication.

The Weave stack comprises four core modules: Security manager, exchange manager, message layer, and fabric state. Weave provides some core functionalities: Bulk data exchange (file transfers), common (system status and error reports), data management, echo (network connectivity testing), security, service directory, and others. Secondary protocols built on top of the core protocols of Weave include alarm, device control, service provisioning, network provisioning, heartbeat, and others.

Check yourself

Brillo and Weave use cases

9.3.5 HomeKit

The HomeKit software framework is designed by Apple to work with its iOS mobile operating system for achieving a centralized device integrating and control framework [17]. It enables device configuration, communication, and control of smart home appliances. Home automation is achieved by incorporating room designs, items, and their actions within the HomeKit service. Users can interact with the framework using speech-based voice commands through Apple's voice assistant, Siri, or through external apps. Smart home devices such as thermostats, lights, locks, cameras, plugs, and others, spread over a house can be controlled by a single HomeKit interface through smartphones. HomeKit-enabled device manufacturers need to have an MFi program, and all devices were initially required to have an encryption coprocessor. Later, the processor-based encryption was changed to a software-based one.

Non-HomeKit devices can have the benefits of HomeKit through the use of HomeKit gateways and hubs.

HomeKit devices within a smart home securely connect to a hub either through Wi-Fi or Bluetooth. However, as the range of Bluetooth is severely limited, the full potential of the HomeKit may not be adequately exploited. This framework allows for individual as well as grouped control of connected devices based on scenarios. Features such as preconfigured devices settings can be collectively commanded using voice commands to Siri.

Points to ponder

The MFi program is Apple's licensing program for hardware/software/firmware developers. It stands for "Made For iPhone/iPad/iMAC".

Check yourself

HomeKit interfacing, HomeKit controls, HomeKit use case

Summary

This chapter introduces the concept of interoperability in the context of IoT architectures, frameworks, and application domains. We initially outline the taxonomy of interoperability to give the readers a perspective of the challenges and the present-day solutions or attempts to solve these challenges. We outline the various standardization efforts to address interoperability issues in different domains. Further, at the end of this chapter, we also provide a brief description of the different interoperability-enabling frameworks that are under development by various corporations across the globe.

Exercises

(i) Differentiate between semantic and syntactic interoperability.

(ii) What are the various types of interoperability encountered in IoT environments?

(iii) What is meant by the heterogeneity of IoT devices in the context of interoperability?

(iv) How is device interoperability different from platform interoperability?

(v) Describe the following standards:

  (a) EnOcean

  (b) DLNA

(c) Konnex
(d) LonWorks
(e) UPnP
(f) X-10
(g) Insteon

(vi) How does EnOcean use energy harvesting for its operations?
(vii) What is LonTalk?
(viii) What is a neuron chip in the context of LonWorks?
(ix) How is X-10 different from DLNA?
(x) How is the UniversAAL framework different from the Alljoyn framework?
(xi) How is Brillo different from Weave?

References

[1] Al-Fuqaha, A., M. Guizani, M., Mohammadi, M., Aledhari, and Ayyash. 2015. "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications." *IEEE Communications Surveys and Tutorials* 17(4): 2347–2376.

[2] Aloi, G., G. Caliciuri, G. Fortino, R. Gravina, P. Pace, W. Russo, and C. Savaglio. 2017. "Enabling IoT Interoperability through Opportunistic Smartphone-based Mobile Gateways." *Journal of Network and Computer Applications* 81: 74–84.

[3] Bröring, A., S. Schmid, C. K. Schindhelm, A. Khelil, S. Käbisch, D. Kramer, D. Le Phuoc, J. Mitic, D. Anicic, and E. Teniente. 2017. "Enabling IoT Ecosystems through Platform Interoperability." *IEEE Software* 34(1): 54–61.

[4] Kiljander, J., A. D'elia, F. Morandi, P. Hyttinen, J. Takalo-Mattila, A. Ylisaukko-Oja, J. P. Soininen, and T. S. Cinotti. 2014. "Semantic Interoperability Architecture for Pervasive Computing and Internet of Things." *IEEE Access* 2: 856–873.

[5] Bandyopadhyay, S., M. Sengupta, S. Maiti, and S. Dutta. 2011. "Role of Middleware for Internet of Things: A Study." *International Journal of Computer Science and Engineering Survey* 2(3): 94–105.

[6] The EnOcean Alliance. https://www.enocean-alliance.org/.

[7] The Digital Living Networking Alliance. https://www.dlna.org/.

[8] Konnex. https://www.konnex.group/en/.

[9] Cheng, D. Y., Philips North America LLC. 2002. "UPnP Enabling Device for Heterogeneous Networks of Slave Devices." U. S. Patent Application 09/742,278.

[10] Echelon, "Introduction to the LonWorks Platform." https://www.echelon.com/assets/blt893a8b319e8ec8c7/078-0183-01B_Intro_to_LonWorks_Rev_2.pdf.

[11] Insteon: The Technology. https://www.insteon.com/technology.

[12] X10 Basics. https://www.x10.com/x10-basics.html.

[13] UniversAAL IoT. https://www.universaal.info/.

[14] AllJoyn Open Source Project. https://openconnectivity.org/developer/reference-implementation/alljoyn/.

[15] IoTivity. https://iotivity.org/.

[16] Brillo/Weave Part 1: High Level Introduction. https://events.static.linuxfound.org/sites/events/files/slides/Brillo%20and%20Weave%20-%20Introduction$_v3_1$.pdf.

[17] HomeKit. https://developer.apple.com/homekit/.