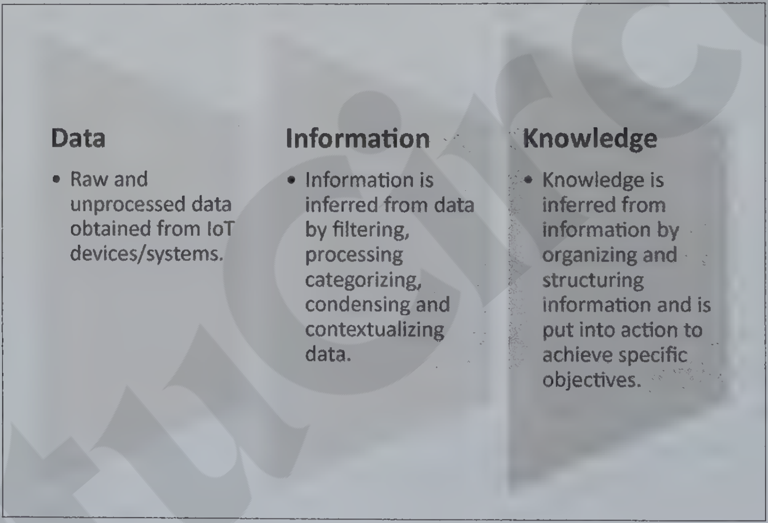
MODULE 1

This Chapter Covers

- Definition & Characteristics of IoT
- Physical Design of IoT
- Logical Design of IoT
- IoT Enabling Technologies
- IoT Levels & Deployment Templates

1.1 Introduction

Internet of Things (IoT) comprises things that have unique identities and are connected to the Internet. While many existing devices, such as networked computers or 4G-enabled mobile phones, already have some form of unique identities and are also connected to the Internet, the focus on IoT is in the configuration, control and networking via the Internet of devices or "things" that are traditionally not associated with the Internet. These include devices such as thermostats, utility meters, a bluetooth-connected headset, irrigation pumps and sensors, or control circuits for an electric car's engine. Internet of Things is a new revolution in the capabilities of the endpoints that are connected to the Internet, and is being driven by the advancements in capabilities (in combination with lower costs) in sensor networks, mobile devices, wireless communications, networking and cloud technologies. Experts forecast that by the year 2020 there will be a total of 50 billion devices/things connected to the Internet. Therefore, the major industry players are excited by the prospects of new markets for their products. The products include hardware and software components for IoT endpoints, hubs, or control centers of the IoT universe.



Data

- Raw and unprocessed data obtained from IoT devices/systems.

Information

- Information is inferred from data by filtering, processing categorizing, condensing and contextualizing data.

Knowledge

- Knowledge is inferred from information by organizing and structuring information and is put into action to achieve specific objectives.

Figure 1.1: Inferring information and knowledge from data

The scope of IoT is not limited to just connecting things (devices, appliances, machines) to the Internet. IoT allows these things to communicate and exchange data (control & information, that could include data associated with users) while executing meaningful applications towards a common user or machine goal. Data itself does not have a meaning until it is contextualized processed into useful information. Applications on IoT networks extract and create information from lower level data by filtering, processing, categorizing, condensing and contextualizing the data. This information obtained is then organized and structured to infer knowledge about the system and/or its users, its environment, and its operations and progress towards its objectives, allowing a smarter performance, as shown in Figure 1.1. For example, consider a series of raw sensor measurements ((72,45) ; (84, 56)) generated by a weather monitoring station, which by themselves do not have any meaning or context. To give meaning to the data, a context is added, which in this example can be that

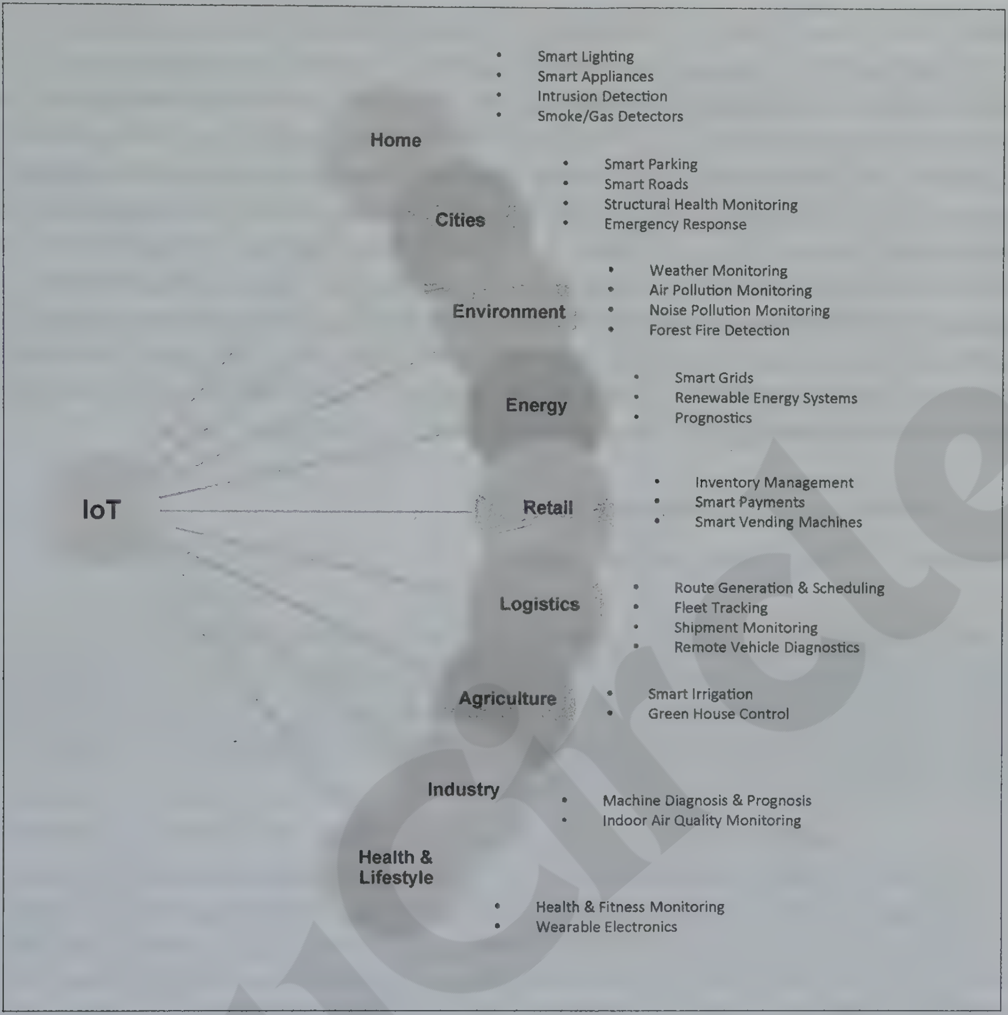


Figure 1.2: Applications of IoT

each tuple in data represents the temperature and humidity measured every minute. With this context added we know the meaning (or information) of the measured data tuples. Further information is obtained by categorizing, condensing or processing this data. For example, the average temperature and humidity readings for last five minutes is obtained by averaging the last five data tuples. The next step is to organize the information and understand the relationships between pieces of information to infer knowledge which can be put into action. For example, an alert is raised if the average temperature in last five minutes exceeds 120F, and this alert may be conditioned on the user's geographical position as well.

The applications of Internet of Things span a wide range of domains including (but not limited to) homes, cities, environment, energy systems, retail, logistics, industry, agriculture and health as listed in Figure 1.2. For homes, IoT has several applications such as smart lighting that adapt the lighting to suit the ambient conditions, smart appliances that can be remotely monitored and controlled, intrusion detection systems, smart smoke detectors, etc. For cities, IoT has applications such as smart parking systems that provide status updates on available slots, smart lighting that helps in saving energy, smart roads that provide information on driving conditions and structural health monitoring systems. For environment, IoT has applications such as weather monitoring, air and noise pollution, forest fire detection and river flood detection systems. For energy systems, IoT has applications such as including smart grids, grid integration of renewable energy sources and prognostic health management systems. For retail domain, IoT has applications such as inventory management, smart payments and smart vending machines. For agriculture domain, IoT has applications such as smart irrigation systems that help in saving water while enhancing productivity and green house control systems. Industrial applications of IoT include machine diagnosis and prognosis systems that help in predicting faults and determining the cause of faults and indoor air quality systems. For health and lifestyle, IoT has applications such as health and fitness monitoring systems and wearable electronics.

1.1.1 Definition & Characteristics of IoT

The Internet of Things (IoT) has been defined as [1]:

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

Let us examine this definition of IoT further to put some of the terms into perspective.

- **Dynamic & Self-Adapting:** IoT devices and systems may have the capability to dynamically adapt with the changing contexts and take actions based on their operating conditions, user's context, or sensed environment. For example, consider a surveillance system comprising of a number of surveillance cameras. The surveillance cameras can adapt their modes (to normal or infra-red night modes) based on whether it is day or night. Cameras could switch from lower resolution to higher resolution modes when any motion is detected and alert nearby cameras to do the same. In this example, the surveillance system is adapting itself based on the context and changing (e.g., dynamic) conditions.
- **Self-Configuring:** IoT devices may have self-configuring capability, allowing a large number of devices to work together to provide certain functionality (such as weather monitoring). These devices have the ability configure themselves (in association with the IoT infrastructure), setup the networking, and fetch latest software upgrades with minimal manual or user intervention.
- **Interoperable Communication Protocols:** IoT devices may support a number of interoperable communication protocols and can communicate with other devices and also with the infrastructure. We describe some of the commonly used communication protocols and models in later sections.
- **Unique Identity:** Each IoT device has a unique identity and a unique identifier (such as an IP address or a URI). IoT systems may have intelligent interfaces which adapt based on the context, allow communicating with users and the environmental contexts. IoT device interfaces allow users to query the devices, monitor their status, and

control them remotely, in association with the control, configuration and management infrastructure.

- **Integrated into Information Network:** IoT devices are usually integrated into the information network that allows them to communicate and exchange data with other devices and systems. IoT devices can be dynamically discovered in the network, by other devices and/or the network, and have the capability to describe themselves (and their characteristics) to other devices or user applications. For example, a weather monitoring node can describe its monitoring capabilities to another connected node so that they can communicate and exchange data. Integration into the information network helps in making IoT systems "smarter" due to the collective intelligence of the individual devices in collaboration with the infrastructure. Thus, the data from a large number of connected weather monitoring IoT nodes can be aggregated and analyzed to predict the weather.

1.2 Physical Design of IoT

1.2.1 Things in IoT

The "Things" in IoT usually refers to IoT devices which have unique identities and can perform remote sensing, actuating and monitoring capabilities. IoT devices can exchange data with other connected devices and applications (directly or indirectly), or collect data from other devices and process the data either locally or send the data to centralized servers or cloud-based application back-ends for processing the data, or perform some tasks locally and other tasks within the IoT infrastructure, based on temporal and space constraints (i.e., memory, processing capabilities, communication latencies and speeds, and deadlines).

Figure 1.3 shows a block diagram of a typical IoT device. An IoT device may consist of several interfaces for connections to other devices, both wired and wireless. These include (i) I/O interfaces for sensors, (ii) interfaces for Internet connectivity, (iii) memory and storage interfaces and (iv) audio/video interfaces. An IoT device can collect various types of data from the on-board or attached sensors, such as temperature, humidity, light intensity. The sensed data can be communicated either to other devices or cloud-based servers/storage. IoT devices can be connected to actuators that allow them to interact with other physical entities (including non-IoT devices and systems) in the vicinity of the device. For example, a relay switch connected to an IoT device can turn an appliance on/off based on the commands sent to the IoT device over the Internet.

IoT devices can also be of varied types, for instance, wearable sensors, smart watches, LED lights, automobiles and industrial machines. Almost all IoT devices generate data in some form or the other which when processed by data analytics systems leads to useful information to guide further actions locally or remotely. For instance, sensor data generated by a soil moisture monitoring device in a garden, when processed can help in determining the optimum watering schedules. Figure 1.4 shows different types of IoT devices.

1.2.2 IoT Protocols

Link Layer

Link layer protocols determine how the data is physically sent over the network's physical layer or medium (e.g., copper wire, coaxial cable, or a radio wave). The scope of the link

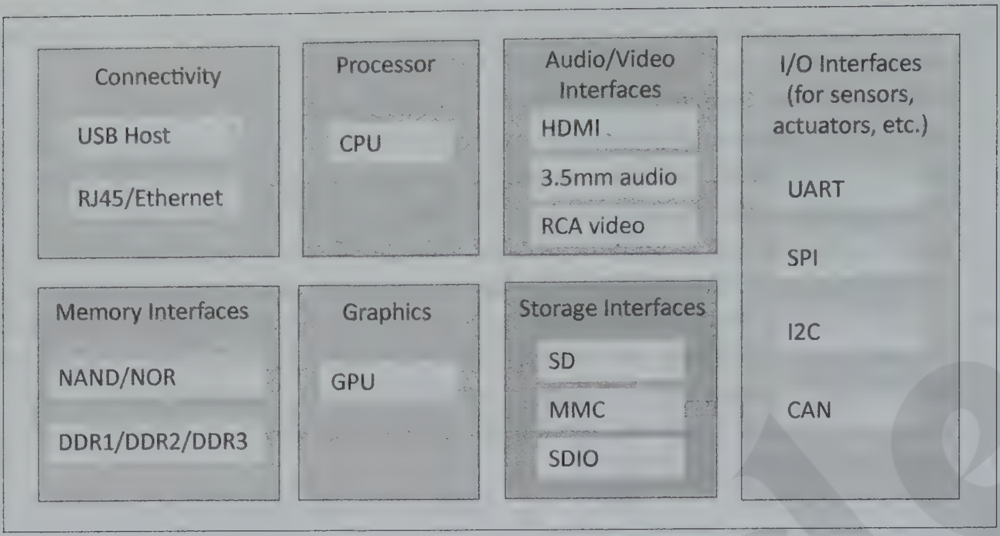


Figure 1.3: Generic block diagram of an IoT Device

layer is the local network connection to which host is attached. Hosts on the same link exchange data packets over the link layer using link layer protocols. Link layer determines how the packets are coded and signaled by the hardware device over the medium to which the host is attached (such as a coaxial cable). Let us now look at some link layer protocols which are relevant in the context of IoT.

- **802.3 - Ethernet :** IEEE 802.3 is a collection of wired Ethernet standards for the link layer. For example, 802.3 is the standard for 10BASE5 Ethernet that uses coaxial cable as a shared medium, 802.3.i is the standard for 10BASE-T Ethernet over copper twisted-pair connections, 802.3.j is the standard for 10BASE-F Ethernet over fiber optic connections, 802.3ae is the standard for 10 Gbit/s Ethernet over fiber, and so on. These standards provide data rates from 10 Mb/s to 40 Gb/s and higher. The shared medium in Ethernet can be a coaxial cable, twisted-pair wire or an optical fiber. The shared medium (i.e., broadcast medium) carries the communication for all the devices on the network, thus data sent by one device can received by all devices subject to propagation conditions and transceiver capabilities. The specifications of the 802.3 standards are available on the IEEE 802.3 working group website [2].
- **802.11 - WiFi :** IEEE 802.11 is a collection of wireless local area network (WLAN) communication standards, including extensive description of the link layer. For example, 802.11a operates in the 5 GHz band, 802.11b and 802.11g operate in the 2.4 GHz band, 802.11n operates in the 2.4/5 GHz bands, 802.11ac operates in the 5 GHz band and 802.11ad operates in the 60 GHz band. These standards provide data rates from 1 Mb/s to upto 6.75 Gb/s. The specifications of the 802.11 standards are available on the IEEE 802.11 working group website [3]
- **802.16 - WiMax :** IEEE 802.16 is a collection of wireless broadband standards, including extensive descriptions for the link layer (also called WiMax). WiMax standards provide data rates from 1.5 Mb/s to 1 Gb/s. The recent update (802.16m) provides data rates of 100 Mbit/s for mobile stations and 1 Gbit/s for fixed stations.

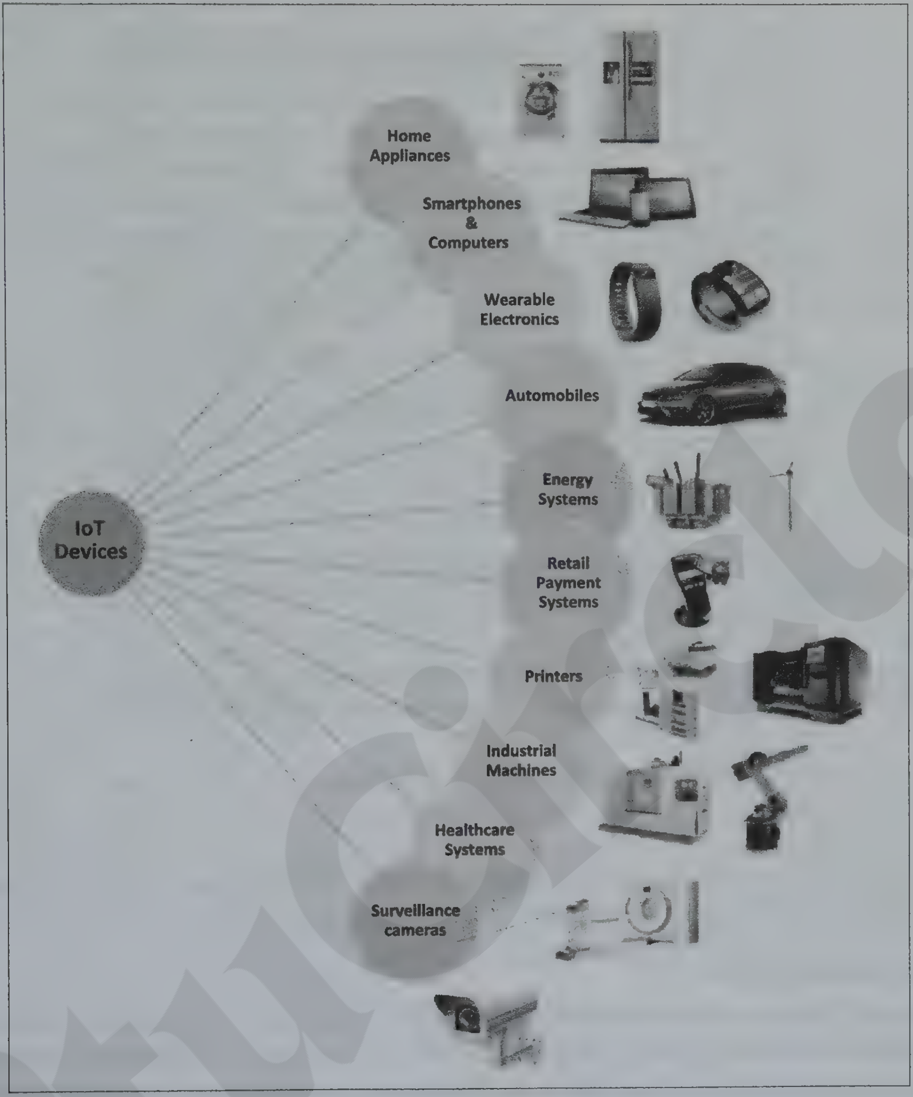


Figure 1.4: IoT Devices

The specifications of the 802.11 standards are readily available on the IEEE 802.16 working group website [4]

- **802.15.4 - LR-WPAN :** IEEE 802.15.4 is a collection of standards for low-rate wireless personal area networks (LR-WPANs). These standards form the basis of specifications for high level communication protocols such as ZigBee. LR-WPAN standards provide data rates from 40 Kb/s 250 Kb/s. These standards provide low-cost and low-speed communication for power constrained devices. The specifications of the 802.15.4 standards are available on the IEEE 802.15 working group website [5]
- **2G/ 3G/ 4G - Mobile Communication :** There are different generations of mobile communication standards including second generation (2G including GSM and CDMA), third generation (3G - including UMTS and CDMA2000) and fourth generation (4G - including LTE). IoT devices based on these standards can communicate over cellular

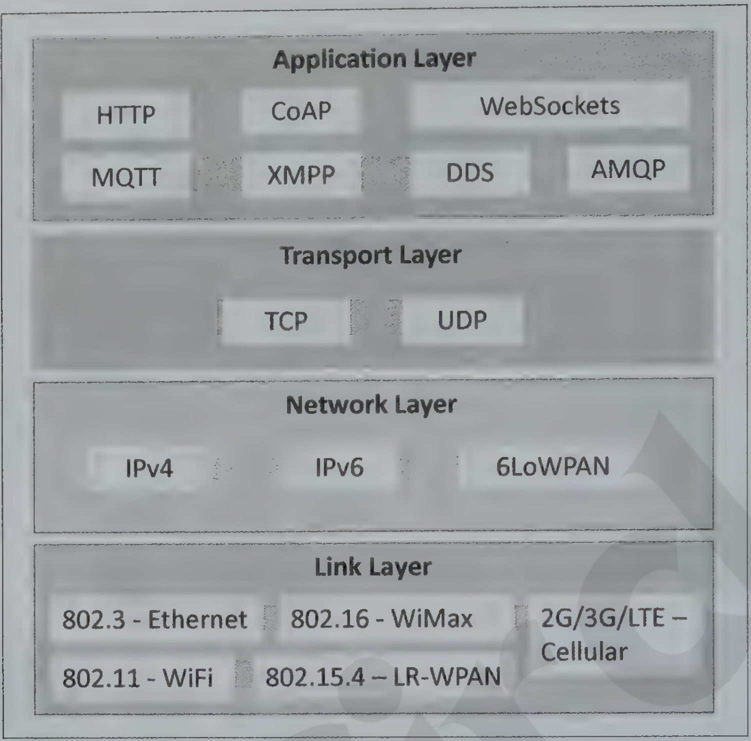


Figure 1.5: IoT Protocols

networks. Data rates for these standards range from 9.6 Kb/s (for 2G) to upto 100 Mb/s (for 4G) and are available from the 3GPP websites.

Network/Internet Layer

The network layers are responsible for sending of IP datagrams from the source network to the destination network. This layer performs the host addressing and packet routing. The datagrams contain the source and destination addresses which are used to route them from the source to destination across multiple networks. Host identification is done using hierarchical IP addressing schemes such as IPv4 or IPv6.

- **IPv4 :** Internet Protocol version 4 (IPv4) is the most deployed Internet protocol that is used to identify the devices on a network using a hierarchical addressing scheme. IPv4 uses a 32-bit address scheme that allows total of 2^{32} or 4,294,967,296 addresses. As more and more devices got connected to the Internet, these addresses got exhausted in the year 2011. IPv4 has been succeeded by IPv6. The IP protocols establish connections on packet networks, but do not guarantee delivery of packets. Guaranteed delivery and data integrity are handled by the upper layer protocols (such as TCP). IPv4 is formally described in RFC 791 [6].
- **IPv6 :** Internet Protocol version 6 (IPv6) is the newest version of Internet protocol and successor to IPv4. IPv6 uses 128-bit address scheme that allows total of 2^{128} or 3.4×10^{38} addresses. IPv4 is formally described in RFC 2460 [7].
- **6LoWPAN :** 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks) brings IP protocol to the low-power devices which have limited processing capability.

6LoWPAN operates in the 2.4 GHz frequency range and provides data transfer rates of 250 Kb/s. 6LoWPAN works with the 802.15.4 link layer protocol and defines compression mechanisms for IPv6 datagrams over IEEE 802.15.4-based networks [8].

Transport Layer

The transport layer protocols provide end-to-end message transfer capability independent of the underlying network. The message transfer capability can be set up on connections, either using handshakes (as in TCP) or without handshakes/acknowledgements (as in UDP). The transport layer provides functions such as error control, segmentation, flow control and congestion control.

- **TCP :** Transmission Control Protocol (TCP) is the most widely used transport layer protocol, that is used by web browsers (along with HTTP, HTTPS application layer protocols), email programs (SMTP application layer protocol) and file transfer (FTP). TCP is a connection oriented and stateful protocol. While IP protocol deals with sending packets, TCP ensures reliable transmission of packets in-order. TCP also provides error detection capability so that duplicate packets can be discarded and lost packets are retransmitted. The flow control capability of TCP ensures that rate at which the sender sends the data is not too high for the receiver to process. The congestion control capability of TCP helps in avoiding network congestion and congestion collapse which can lead to degradation of network performance. TCP is described in RFC 793 [9].
- **UDP :** Unlike TCP, which requires carrying out an initial setup procedure, UDP is a connectionless protocol. UDP is useful for time-sensitive applications that have very small data units to exchange and do not want the overhead of connection setup. UDP is a transaction oriented and stateless protocol. UDP does not provide guaranteed delivery, ordering of messages and duplicate elimination. Higher levels of protocols can ensure reliable delivery or ensuring connections created are reliable. UDP is described in RFC 768 [10].

Application Layer

Application layer protocols define how the applications interface with the lower layer protocols to send the data over the network. The application data, typically in files, is encoded by the application layer protocol and encapsulated in the transport layer protocol which provides connection or transaction oriented communication over the network. Port numbers are used for application addressing (for example port 80 for HTTP, port 22 for SSH, etc.). Application layer protocols enable process-to-process connections using ports.

- **HTTP :** Hypertext Transfer Protocol (HTTP) is the application layer protocol that forms the foundation of the World Wide Web (WWW). HTTP includes commands such as GET, PUT, POST, DELETE, HEAD, TRACE, OPTIONS, etc. The protocol follows a request-response model where a client sends requests to a server using the HTTP commands. HTTP is a stateless protocol and each HTTP request is independent of the other requests. An HTTP client can be a browser or an application running on the client (e.g., an application running on an IoT device, a mobile application or other software). HTTP protocol uses Universal Resource Identifiers (URIs) to identify HTTP resources. HTTP is described in RFC 2616 [11].
- **CoAP :** Constrained Application Protocol (CoAP) is an application layer protocol for

machine-to-machine (M2M) applications, meant for constrained environments with constrained devices and constrained networks. Like HTTP, CoAP is a web transfer protocol and uses a request-response model, however it runs on top of UDP instead of TCP. CoAP uses a client-server architecture where clients communicate with servers using connectionless datagrams. CoAP is designed to easily interface with HTTP. Like HTTP, CoAP supports methods such as GET, PUT, POST, and DELETE. CoAP draft specifications are available on IEFT Constrained environments (CoRE) Working Group website [12].

- **WebSocket :** WebSocket protocol allows full-duplex communication over a single socket connection for sending messages between client and server. WebSocket is based on TCP and allows streams of messages to be sent back and forth between the client and server while keeping the TCP connection open. The client can be a browser, a mobile application or an IoT device. WebSocket is described in RFC 6455 [13].
- **MQTT :** Message Queue Telemetry Transport (MQTT) is a light-weight messaging protocol based on the publish-subscribe model. MQTT uses a client-server architecture where the client (such as an IoT device) connects to the server (also called MQTT Broker) and publishes messages to topics on the server. The broker forwards the messages to the clients subscribed to topics. MQTT is well suited for constrained environments where the devices have limited processing and memory resources and the network bandwidth is low. MQTT specifications are available on IBM developerWorks [14].
- **XMPP :** Extensible Messaging and Presence Protocol (XMPP) is a protocol for real-time communication and streaming XML data between network entities. XMPP powers wide range of applications including messaging, presence, data syndication, gaming, multi-party chat and voice/video calls. XMPP allows sending small chunks of XML data from one network entity to another in near real-time. XMPP is a decentralized protocol and uses a client-server architecture. XMPP supports both client-to-server and server-to-server communication paths. In the context of IoT, XMPP allows real-time communication between IoT devices. XMPP is described in RFC 6120 [15].
- **DDS :** Data Distribution Service (DDS) is a data-centric middleware standard for device-to-device or machine-to-machine communication. DDS uses a publish-subscribe model where publishers (e.g. devices that generate data) create topics to which subscribers (e.g., devices that want to consume data) can subscribe. Publisher is an object responsible for data distribution and the subscriber is responsible for receiving published data. DDS provides quality-of-service (QoS) control and configurable reliability. DDS is described in Object Management Group (OMG) DDS specification [16].
- **AMQP :** Advanced Message Queuing Protocol (AMQP) is an open application layer protocol for business messaging. AMQP supports both point-to-point and publisher/subscriber models, routing and queuing. AMQP brokers receive messages from publishers (e.g., devices or applications that generate data) and route them over connections to consumers (applications that process data). Publishers publish the messages to exchanges which then distribute message copies to queues. Messages are either delivered by the broker to the consumers which have subscribed to the queues or the consumers can pull the messages from the queues. AMQP specification is available

on the AMQP working group website [17].

1.3 Logical Design of IoT

Logical design of an IoT system refers to an abstract representation of the entities and processes without going into the low-level specifics of the implementation. In this section we describe the functional blocks of an IoT system and the communication APIs that are used for the examples in this book. The steps in logical design are described in additional detail in Chapter-5.

1.3.1 IoT Functional Blocks

An IoT system comprises of a number of functional blocks that provide the system the capabilities for identification, sensing, actuation, communication, and management as shown in Figure 1.6. These functional blocks are described as follows:

- **Device :** An IoT system comprises of devices that provide sensing, actuation, monitoring and control functions. You learned about IoT devices in section 1.2.
- **Communication :** The communication block handles the communication for the IoT system. You learned about various protocols used for communication by IoT systems in section 1.2.
- **Services :** An IoT system uses various types of IoT services such as services for device monitoring, device control services, data publishing services and services for device discovery.
- **Management :** Management functional block provides various functions to govern the IoT system.
- **Security** Security functional block secures the IoT system and by providing functions such as authentication, authorization, message and content integrity, and data security.
- **Application :** IoT applications provide an interface that the users can use to control and monitor various aspects of the IoT system. Applications also allow users to view the system status and view or analyze the processed data.

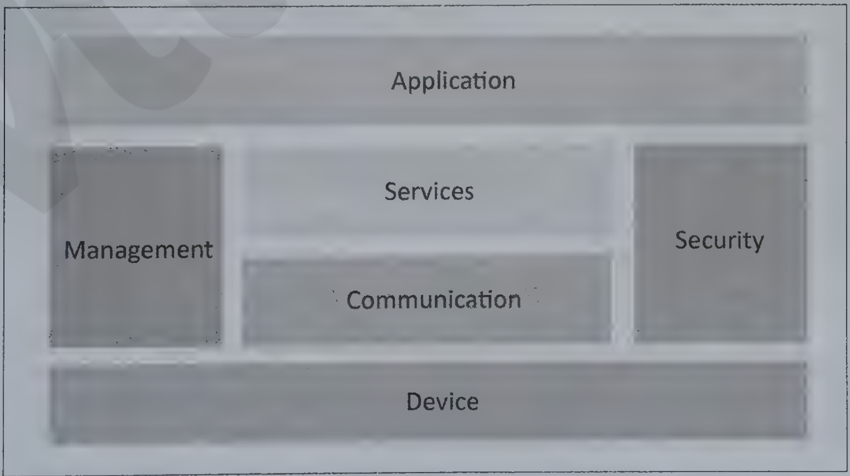


Figure 1.6: Functional Blocks of IoT

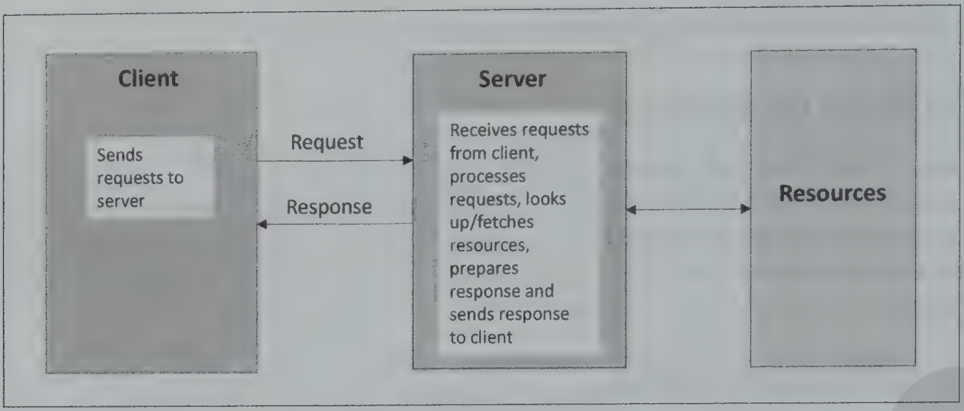

Client
Sends requests to server
Request
Response
Server
Receives requests from client, processes requests, looks up/fetches resources, prepares response and sends response to client
Resources

Figure 1.7: Request-Response communication model

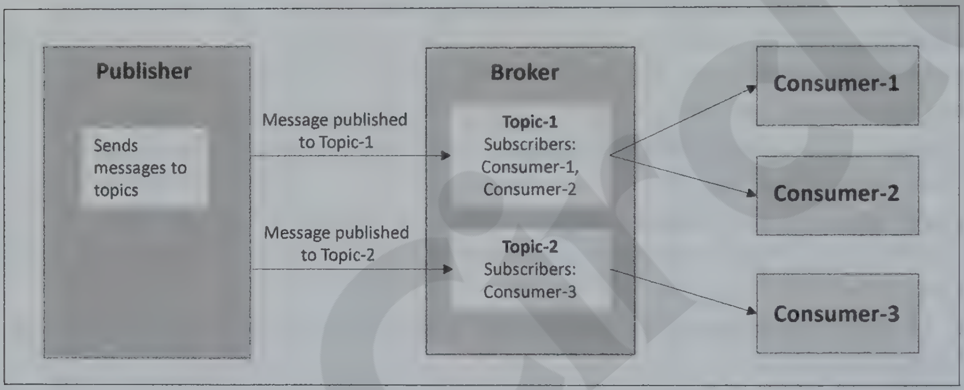

Publisher
Sends messages to topics
Message published to Topic-1
Message published to Topic-2
Broker
Topic-1 Subscribers: Consumer-1, Consumer-2
Topic-2 Subscribers: Consumer-3
Consumer-1
Consumer-2
Consumer-3

Figure 1.8: Publish-Subscribe communication model

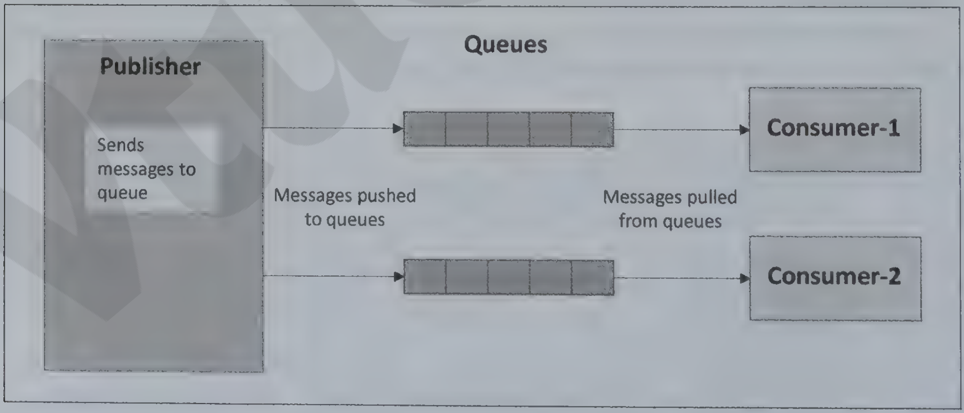


Figure 1.9: Push-Pull communication model

1.3.2 IoT Communication Models

- **Request-Response :** Request-Response is a communication model in which the client sends requests to the server and the server responds to the requests. When the server receives a request, it decides how to respond, fetches the data, retrieves

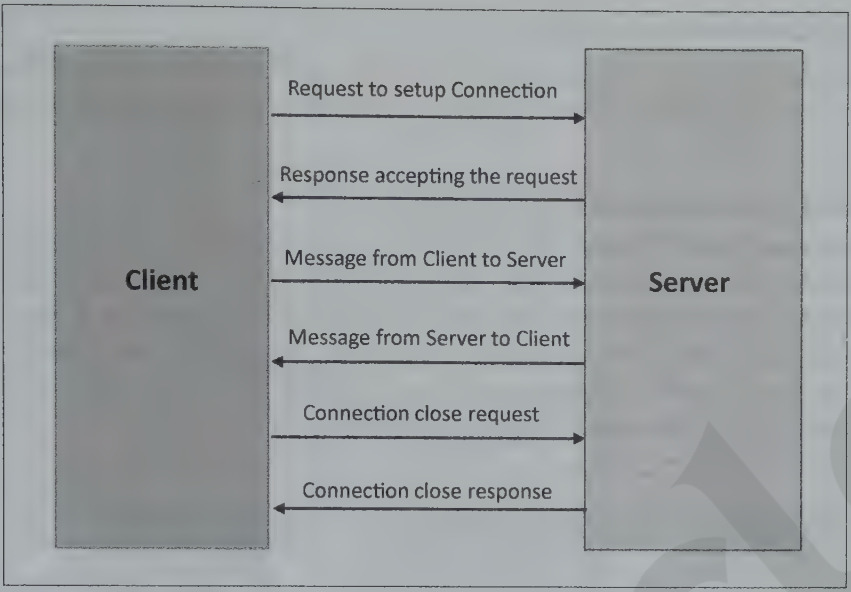

Client
Server
Request to setup Connection
Response accepting the request
Message from Client to Server
Message from Server to Client
Connection close request
Connection close response

Figure 1.10: Exclusive Pair communication model

resource representations, prepares the response, and then sends the response to the client. Request-Response model is a stateless communication model and each request-response pair is independent of others. Figure 1.7 shows the client-server interactions in the request-response model.

- **Publish-Subscribe :** Publish-Subscribe is a communication model that involves publishers, brokers and consumers. Publishers are the source of data. Publishers send the data to the topics which are managed by the broker. Publishers are not aware of the consumers. Consumers subscribe to the topics which are managed by the broker. When the broker receives data for a topic from the publisher, it sends the data to all the subscribed consumers. Figure 1.8 shows the publisher-broker-consumer interactions in the publish-subscribe model.
- **Push-Pull :** Push-Pull is a communication model in which the data producers push the data to queues and the consumers pull the data from the queues. Producers do not need to be aware of the consumers. Queues help in decoupling the messaging between the producers and consumers. Queues also act as a buffer which helps in situations when there is a mismatch between the rate at which the producers push data and the rate rate at which the consumers pull data. Figure 1.9 shows the publisher-queue-consumer interactions in the push-pull model.
- **Exclusive Pair :** Exclusive Pair is a bi-directional, fully duplex communication model that uses a persistent connection between the client and server. Once the connection is setup it remains open until the client sends a request to close the connection. Client and server can send messages to each other after connection setup. Exclusive pair is a stateful communication model and the server is aware of all the open connections. Figure 1.10 shows the client-server interactions in the exclusive pair model.

1.3.3 IoT Communication APIs

In the previous section you learned about various communication models. In this section you will learn about two specific communication APIs which are used in the examples in this book.

REST-based Communication APIs

Representational State Transfer (REST) [88] is a set of architectural principles by which you can design web services and web APIs that focus on a system's resources and how resource states are addressed and transferred. REST APIs follow the request-response communication model described in previous section. The REST architectural constraints apply to the components, connectors, and data elements, within a distributed hypermedia system. The REST architectural constraints are as follows:

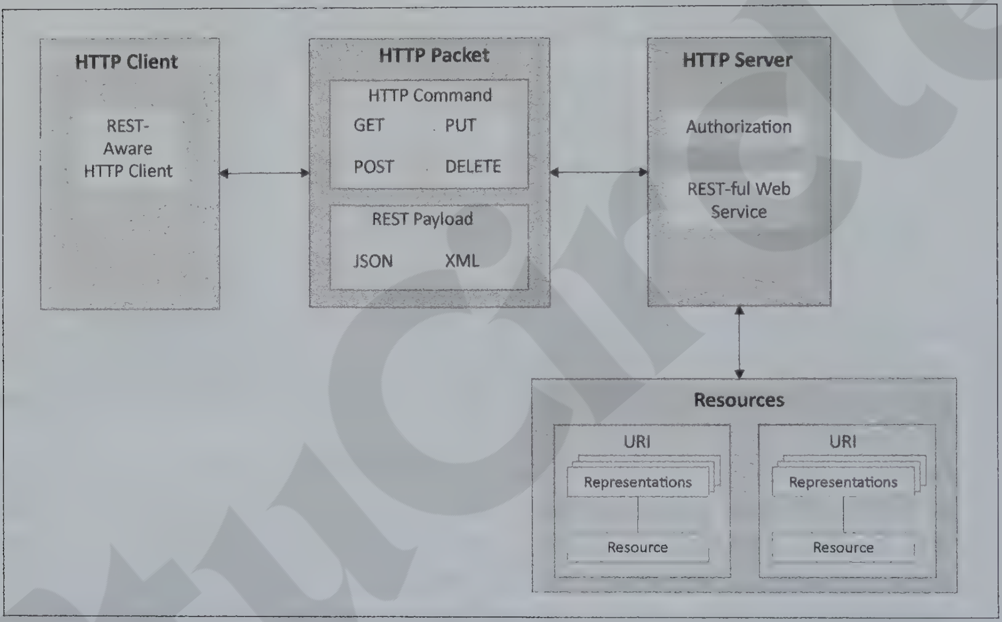


Figure 1.11: Communication with REST APIs

- **Client-Server**: The principle behind the client-server constraint is the separation of concerns. For example, clients should not be concerned with the storage of data which is a concern of the server. Similarly, the server should not be concerned about the user interface, which is a concern of the client. Separation allows client and server to be independently developed and updated.
- **Stateless**: Each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server. The session state is kept entirely on the client.
- **Cache-able**: Cache constraint requires that the data within a response to a request be implicitly or explicitly labeled as cache-able or non-cache-able. If a response is cache-able, then a client cache is given the right to reuse that response data for later, equivalent requests. Caching can partially or completely eliminate some interactions and improve efficiency and scalability.

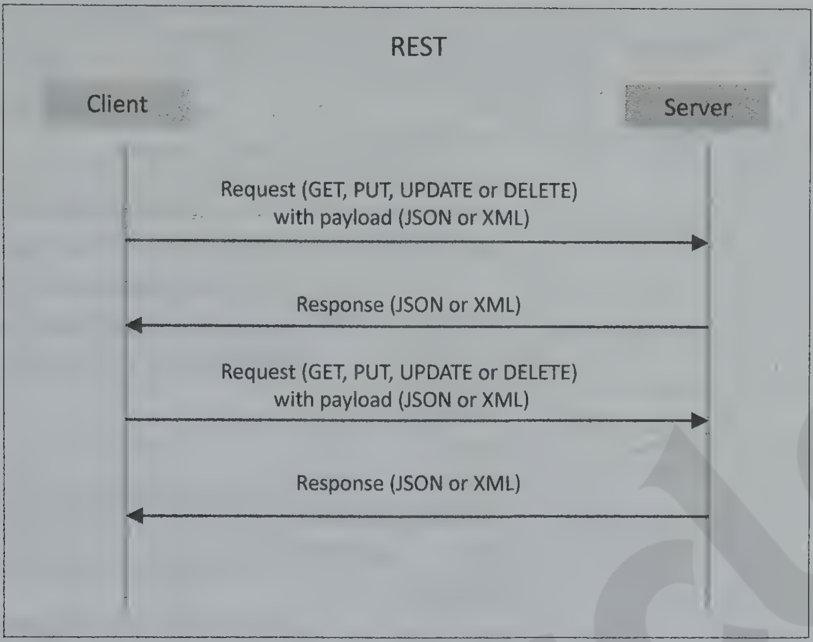

REST
Client
Server
Request (GET, PUT, UPDATE or DELETE) with payload (JSON or XML)
Response (JSON or XML)
Request (GET, PUT, UPDATE or DELETE) with payload (JSON or XML)
Response (JSON or XML)

Figure 1.12: Request-response model used by REST

- **Layered System**: Layered system constraint, constrains the behavior of components such that each component cannot see beyond the immediate layer with which they are interacting. For example, a client cannot tell whether it is connected directly to the end server, or to an intermediary along the way. System scalability can be improved by allowing intermediaries to respond to requests instead of the end server, without the client having to do anything different.
- **Uniform Interface**: Uniform Interface constraint requires that the method of communication between a client and a server must be uniform. Resources are identified in the requests (by URIs in web based systems) and are themselves separate from the representations of the resources that are returned to the client. When a client holds a representation of a resource it has all the information required to update or delete the resource (provided the client has required permissions). Each message includes enough information to describe how to process the message.
- **Code on demand**: Servers can provide executable code or scripts for clients to execute in their context. This constraint is the only one that is optional.

A RESTful web service is a "web API" implemented using HTTP and REST principles. Figure 1.11 shows the communication between client and server using REST APIs. Figure 1.12 shows the interactions in the request-response model used by REST. RESTful web service is a collection of resources which are represented by URIs. RESTful web API has a base URI (e.g. http://example.com/api/tasks/). The clients send requests to these URIs using the methods defined by the HTTP protocol (e.g., GET, PUT, POST, or DELETE), as shown in Table 1.1. A RESTful web service can support various Internet media types (JSON being the most popular media type for RESTful web services). IP for Smart Objects Alliance (IPSO Alliance) has published an Application Framework that defines a RESTful design for use in

HTTP Method	Resource Type	Action	Example
GET	Collection URI	List all the resources in a collection	http://example.com/api/tasks/ (list all tasks)
GET	Element URI	Get information about a resource	http://example.com/api/tasks/1/ (get information on task-1)
POST	Collection URI	Create a new resource	http://example.com/api/tasks/ (create a new task from data provided in the request)
POST	Element URI	Generally not used	
PUT	Collection URI	Replace the entire collection with another collection	http://example.com/api/tasks/ (replace entire collection with data provided in the request)
PUT	Element URI	Update a resource	http://example.com/api/tasks/1/ (update task-1 with data provided in the request)
DELETE	Collection URI	Delete the entire collection	http://example.com/api/tasks/ (delete all tasks)
DELETE	Element URI	Delete a resource	http://example.com/api/tasks/1/ (delete task-1)

Table 1.1: HTTP request methods and actions

IP smart object systems [18].

WebSocket-based Communication APIs

WebSocket APIs allow bi-directional, full duplex communication between clients and servers. WebSocket APIs follow the exclusive pair communication model described in previous section and as shown in Figure 1.13. Unlike request-response APIs such as REST, the WebSocket APIs allow full duplex communication and do not require a new connection to be setup for each message to be sent. WebSocket communication begins with a connection setup request sent by the client to the server. This request (called a WebSocket handshake) is sent over HTTP and the server interprets it as an upgrade request. If the server supports WebSocket protocol, the server responds to the WebSocket handshake response. After the connection is setup, the client and server can send data/messages to each other in full-duplex mode. WebSocket APIs reduce the network traffic and latency as there is no overhead for connection setup and termination requests for each message. WebSocket is suitable for IoT applications that have low latency or high throughput requirements.

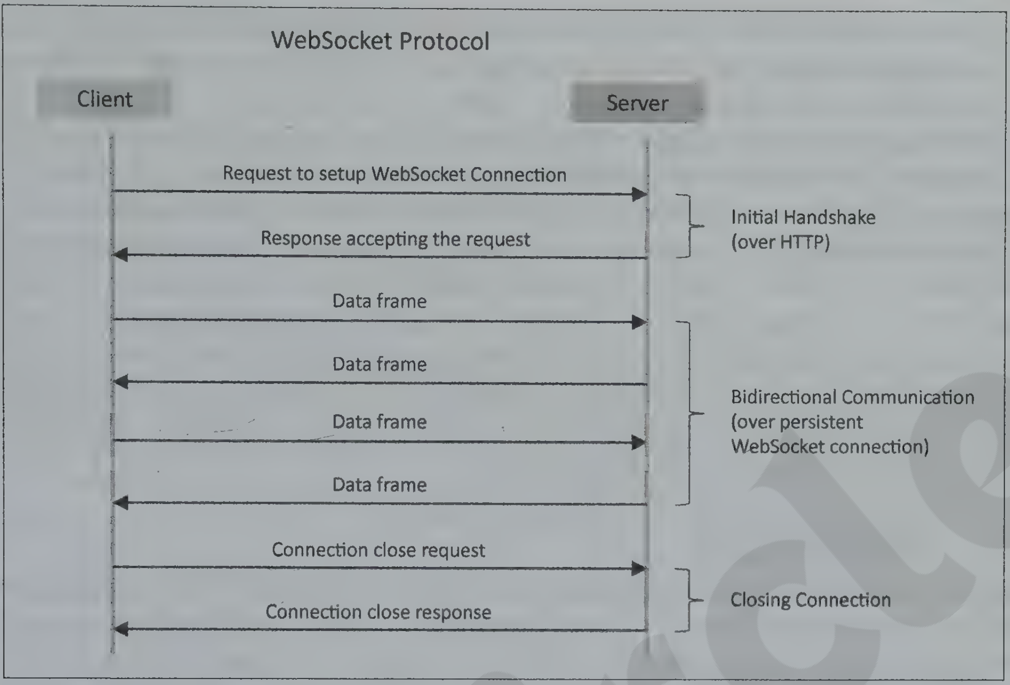

WebSocket Protocol
Client
Server
Request to setup WebSocket Connection
Response accepting the request
Initial Handshake (over HTTP)
Data frame
Data frame
Data frame
Data frame
Bidirectional Communication (over persistent WebSocket connection)
Connection close request
Connection close response
Closing Connection

Figure 1.13: Exclusive pair model used by WebSocket APIs

1.4 IoT Enabling Technologies

IoT is enabled by several technologies including wireless sensor networks, cloud computing, big data analytics, embedded systems, security protocols and architectures, communication protocols, web services, mobile Internet, and semantic search engines. This section provides an overview of some of these technologies which play a key-role in IoT.

1.4.1 Wireless Sensor Networks

A Wireless Sensor Network (WSN) comprises of distributed devices with sensors which are used to monitor the environmental and physical conditions. A WSN consist of a number of end-nodes and routers and a coordinator. End nodes have several sensors attached to them. End nodes can also act as routers. Routers are responsible for routing the data packets from end-nodes to the coordinator. The coordinator collects the data from all the nodes. Coordinator also acts as a gateway that connects the WSN to the Internet. Some examples of WSNs used in IoT systems are described as follows:

- Weather monitoring systems use WSNs in which the nodes collect temperature, humidity and other data, which is aggregated and analyzed.
- Indoor air quality monitoring systems use WSNs to collect data on the indoor air quality and concentration of various gases.
- Soil moisture monitoring systems use WSNs to monitor soil moisture at various locations.
- Surveillance systems use WSNs for collecting surveillance data (such as motion detection data)
- Smart grids use WSNs for monitoring the grid at various points.

- Structural health monitoring systems use WSNs to monitor the health of structures (buildings, bridges) by collecting vibration data from sensor nodes deployed at various points in the structure.

WSNs are enabled by wireless communication protocols such as IEEE 802.15.4. ZigBee is one of the most popular wireless technologies used by WSNs. ZigBee specifications are based on IEEE 802.15.4. ZigBee operates at 2.4 GHz frequency and offers data rates upto 250 KB/s and range from 10 to 100 meters depending on the power output and environmental conditions. The power of WSNs lies in their ability to deploy large number of low-cost and low-power sensing nodes for continuous monitoring of environmental and physical conditions. WSNs are self-organizing networks. Since WSNs have large number of nodes, manual configuration for each node is not possible. The self-organizing capability of WSN makes the network robust. In the event of failure of some nodes or addition of new nodes to the network, the network can reconfigure itself.

1.4.2 Cloud Computing

Cloud computing is a transformative computing paradigm that involves delivering applications and services over the Internet. Cloud computing involves provisioning of computing, networking and storage resources on demand and providing these resources as metered services to the users, in a "pay as you go" model. Cloud computing resources can be provisioned on-demand by the users, without requiring interactions with the cloud service provider. The process of provisioning resources is automated. Cloud computing resources can be accessed over the network using standard access mechanisms that provide platform-independent access through the use of heterogeneous client platforms such as workstations, laptops, tablets and smart-phones. The computing and storage resources provided by cloud service providers are pooled to serve multiple users using multi-tenancy. Multi-tenant aspects of the cloud allow multiple users to be served by the same physical hardware. Users are assigned virtual resources that run on top of the physical resources.

Cloud computing services are offered to users in different forms (see the authors' companion book on Cloud Computing, for instance):

- **Infrastructure-as-a-Service (IaaS) :** IaaS provides the users the ability to provision computing and storage resources. These resources are provided to the users as virtual machine instances and virtual storage. Users can start, stop, configure and manage the virtual machine instances and virtual storage. Users can deploy operating systems and applications of their choice on the virtual resources provisioned in the cloud. The cloud service provider manages the underlying infrastructure. Virtual resources provisioned by the users are billed based on a pay-per-use paradigm.
- **Platform-as-a-Service (PaaS) :** PaaS provides the users the ability to develop and deploy application in the cloud using the development tools, application programming interfaces (APIs), software libraries and services provided by the cloud service provider. The cloud service provider manages the underlying cloud infrastructure including servers, network, operating systems and storage. The users, themselves, are responsible for developing, deploying, configuring and managing applications on the cloud infrastructure.
- **Software-as-a-Service (SaaS) :** SaaS provides the users a complete software application or the user interface to the application itself. The cloud service provider manages

the underlying cloud infrastructure including servers, network, operating systems, storage and application software, and the user is unaware of the underlying architecture of the cloud. Applications are provided to the user through a thin client interface (e.g., a browser). SaaS applications are platform independent and can be accessed from various client devices such as workstations, laptop, tablets and smart-phones, running different operating systems. Since the cloud service provider manages both the application and data, the users are able to access the applications from anywhere.

1.4.3 Big Data Analytics

Big data is defined as collections of data sets whose volume, velocity (in terms of its temporal variation), or variety, is so large that it is difficult to store, manage, process and analyze the data using traditional databases and data processing tools. Big data analytics involves several steps starting from data cleansing, data munging (or wrangling), data processing and visualization. Some examples of big data generated by IoT systems are described as follows:

- Sensor data generated by IoT systems such as weather monitoring stations.
- Machine sensor data collected from sensors embedded in industrial and energy systems for monitoring their health and detecting failures.
- Health and fitness data generated by IoT devices such as wearable fitness bands.
- Data generated by IoT systems for location and tracking of vehicles.
- Data generated by retail inventory monitoring systems.

The underlying characteristics of big data include:

- **Volume:** Though there is no fixed threshold for the volume of data to be considered as big data, however, typically, the term big data is used for massive scale data that is difficult to store, manage and process using traditional databases and data processing architectures. The volumes of data generated by modern IT, industrial, and health-care systems, for example, is growing exponentially driven by the lowering costs of data storage and processing architectures and the need to extract valuable insights from the data to improve business processes, efficiency and service to consumers.
- **Velocity:** Velocity is another important characteristic of big data and the primary reason for exponential growth of data. Velocity of data refers to how fast the data is generated and how frequently it varies. Modern IT, industrial and other systems are generating data at increasingly higher speeds.
- **Variety:** Variety refers to the forms of the data. Big data comes in different forms such as structured or unstructured data, including text data, image, audio, video and sensor data.

1.4.4 Communication Protocols

Communication protocols form the backbone of IoT systems and enable network connectivity and coupling to applications. Communication protocols allow devices to exchange data over the network. In section 1.2.2 you learned about various link, network, transport and application layer protocols. These protocols define the data exchange formats, data encoding, addressing schemes for devices and routing of packets from source to destination. Other functions of the protocols include sequence control (that helps in ordering packets determining lost packets), flow control (that helps in controlling the rate at which the sender is sending the data so that the receiver or the network is not overwhelmed) and retransmission of lost packets.

1.4.5 Embedded Systems

An Embedded System is a computer system that has computer hardware and software embedded to perform specific tasks. In contrast to general purpose computers or personal computers (PCs) which can perform various types of tasks, embedded systems are designed to perform a specific set of tasks. Key components of an embedded system include, microprocessor or microcontroller, memory (RAM, ROM, cache), networking units (Ethernet, WiFi adapters), input/output units (display, keyboard, etc.) and storage (such as flash memory). Some embedded systems have specialized processors such as digital signal processors (DSPs), graphics processors and application specific processors. Embedded systems run embedded operating systems such as real-time operating systems (RTOS). Embedded systems range from low-cost miniaturized devices such as digital watches to devices such as digital cameras, point of sale terminals, vending machines, appliances (such as washing machines), etc. In the next chapter we describe how such devices form an integral part of IoT systems.

1.5 IoT Levels & Deployment Templates

In this section we define various levels of IoT systems with increasing completely. An IoT system comprises of the following components:

- **Device:** An IoT device allows identification, remote sensing, actuating and remote monitoring capabilities. You learned about various examples of IoT devices in section 1.2.1.
- **Resource:** Resources are software components on the IoT device for accessing, processing, and storing sensor information, or controlling actuators connected to the device. Resources also include the software components that enable network access for the device.
- **Controller Service:** Controller service is a native service that runs on the device and interacts with the web services. Controller service sends data from the device to the web service and receives commands from the application (via web services) for controlling the device.
- **Database:** Database can be either local or in the cloud and stores the data generated by the IoT device.
- **Web Service:** Web services serve as a link between the IoT device, application, database and analysis components. Web service can be either implemented using HTTP and REST principles (REST service) or using WebSocket protocol (WebSocket service). A comparison of REST and WebSocket is provided below:
  - **Stateless/Stateful:** REST services are stateless in nature. Each request contains all the information needed to process it. Requests are independent of each other. WebSocket on the other hand is stateful in nature where the server maintains the state and is aware of all the open connections.
  - **Uni-directional/Bi-directional:** REST services operate over HTTP and are uni-directional. Request is always sent by a client and the server responds to the

requests. On the other hand, WebSocket is a bi-directional protocol and allows both client and server to send messages to each other.
- **Request-Response/Full Duplex:** REST services follow a request-response communication model where the client sends requests and the server responds to the requests. WebSocket on the other hand allow full-duplex communication between the client and server, i.e., both client and server can send messages to each other independently.
- **TCP Connections:** For REST services, each HTTP request involves setting up a new TCP connection. WebSocket on the other hand involves a single TCP connection over which the client and server communicate in a full-duplex mode.
- **Header Overhead:** REST services operate over HTTP, and each request is independent of others. Thus each request carries HTTP headers which is an overhead. Due the overhead of HTTP headers, REST is not suitable for real-time applications. WebSocket on the other hand does not involve overhead of headers. After the initial handshake (that happens over HTTP), the client and server exchange messages with minimal frame information. Thus WebSocket is suitable for real-time applications.
- **Scalability:** Scalability is easier in the case of REST services as requests are independent and no state information needs to be maintained by the server. Thus both horizontal (scaling-out) and vertical scaling (scaling-up) solutions are possible for REST services. For WebSockets, horizontal scaling can be cumbersome due to the stateful nature of the communication. Since the server maintains the state of a connection, vertical scaling is easier for WebSockets than horizontal scaling.

- **Analysis Component:** The Analysis Component is responsible for analyzing the IoT data and generate results in a form which are easy for the user to understand. Analysis of IoT data can be performed either locally or in the cloud. Analyzed results are stored in the local or cloud databases.
- **Application:** IoT applications provide an interface that the users can use to control and monitor various aspects of the IoT system. Applications also allow users to view the system status and view the processed data.

1.5.1 IoT Level-1

A level-1 IoT system has a single node/device that performs sensing and/or actuation, stores data, performs analysis and hosts the application as shown in Figure 1.14. Level-1 IoT systems are suitable for modeling low-cost and low-complexity solutions where the data involved is not big and the analysis requirements are not computationally intensive.

Let us now consider an example of a level-1 IoT system for home automation. The system consists of a single node that allows controlling the lights and appliances in a home remotely. The device used in this system interfaces with the lights and appliances using electronic relay switches. The status information of each light or appliance is maintained in a local database. REST services deployed locally allow retrieving and updating the state of each light or appliance in the status database. The controller service continuously monitors the state of each light or appliance (by retrieving state from the database) and triggers the relay switches accordingly. The application which is deployed locally has a user interface for controlling the lights or appliances. Since the device is connected to the Internet, the application can be accessed remotely as well.

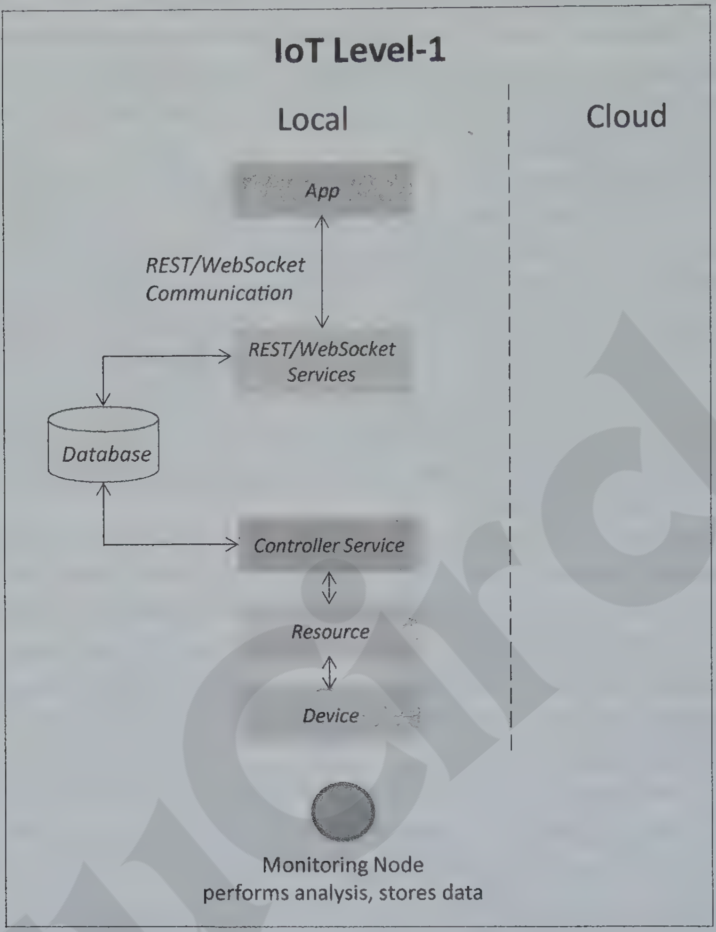


Figure 1.14: IoT Level-1

1.5.2 IoT Level-2

A level-2 IoT system has a single node that performs sensing and/or actuation and local analysis as shown in Figure 1.15. Data is stored in the cloud and application is usually cloud-based. Level-2 IoT systems are suitable for solutions where the data involved is big, however, the primary analysis requirement is not computationally intensive and can be done locally itself.

Let us consider an example of a level-2 IoT system for smart irrigation. The system consists of a single node that monitors the soil moisture level and controls the irrigation system. The device used in this system collects soil moisture data from sensors. The controller service continuously monitors the moisture levels. If the moisture level drops below a threshold, the irrigation system is turned on. For controlling the irrigation system actuators such as solenoid valves can be used. The controller also sends the moisture data to the computing cloud. A cloud-based REST web service is used for storing and retrieving moisture data which is stored in the cloud database. A cloud-based application is used for visualizing the moisture levels over a period of time, which can help in making decisions about irrigation schedules.

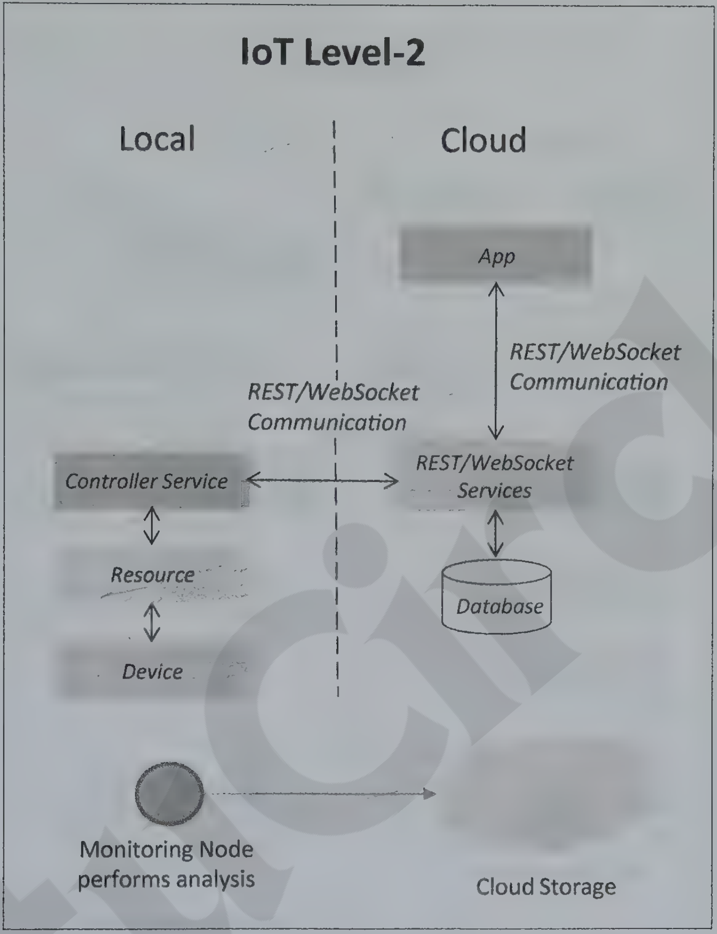


Figure 1.15: IoT Level-2

1.5.3 IoT Level-3

A level-3 IoT system has a single node. Data is stored and analyzed in the cloud and application is cloud-based as shown in Figure 1.16. Level-3 IoT systems are suitable for solutions where the data involved is big and the analysis requirements are computationally intensive.

Let us consider an example of a level-2 IoT system for tracking package handling. The system consists of a single node (for a package) that monitors the vibration levels for a package being shipped. The device in this system uses accelerometer and gyroscope sensors for monitoring vibration levels. The controller service sends the sensor data to the cloud in real-time using a WebSocket service. The data is stored in the cloud and also visualized using a cloud-based application. The analysis components in the cloud can trigger alerts if the vibration levels become greater than a threshold. The benefit of using WebSocket service instead of REST service in this example is that, the sensor data can be sent in real time to the cloud. Moreover, cloud based applications can subscribe to the sensor data feeds for viewing the real-time data.

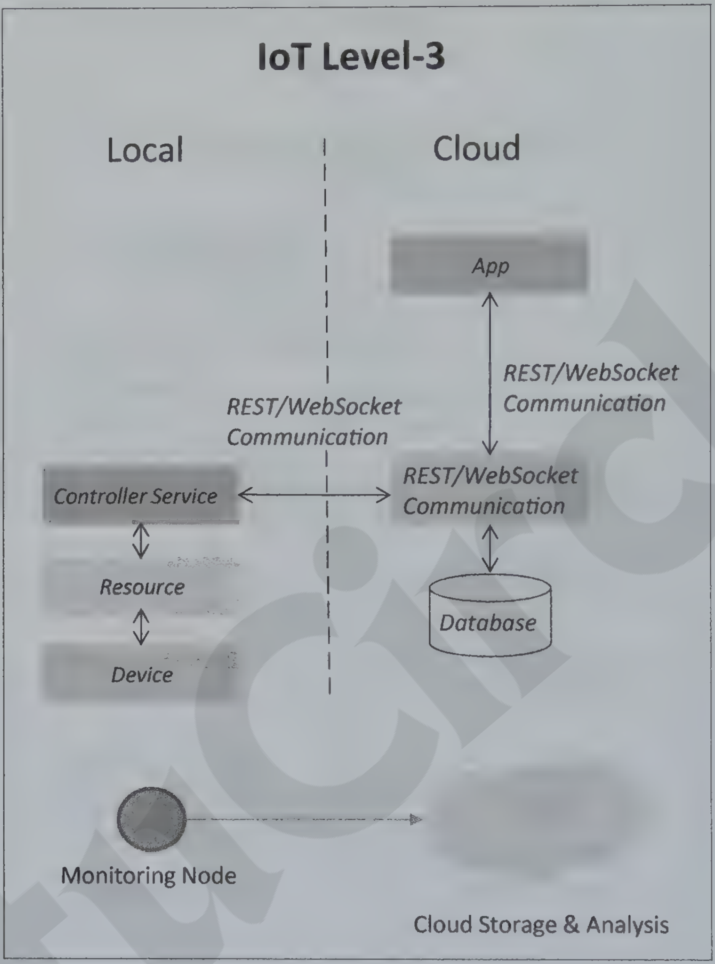


Figure 1.16: IoT Level-3

1.5.4 IoT Level-4

A level-4 IoT system has multiple nodes that perform local analysis. Data is stored in the cloud and application is cloud-based as shown in Figure 1.17. Level-4 contains local and cloud-based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices. Observer nodes can process information and use it for various applications, however, observer nodes do not perform any control functions. Level-4 IoT systems are suitable for solutions where multiple nodes are required, the data involved is big and the analysis requirements are computationally intensive.

Let us consider an example of a level-4 IoT system for noise monitoring. The system consists of multiple nodes placed in different locations for monitoring noise levels in an area. The nodes in this example are equipped with sound sensors. Nodes are independent of each other. Each node runs its own controller service that sends the data to the cloud. The data is stored in a cloud database. The analysis of data collected from a number of nodes is done in the cloud. A cloud-based application is used for visualizing the aggregated data.

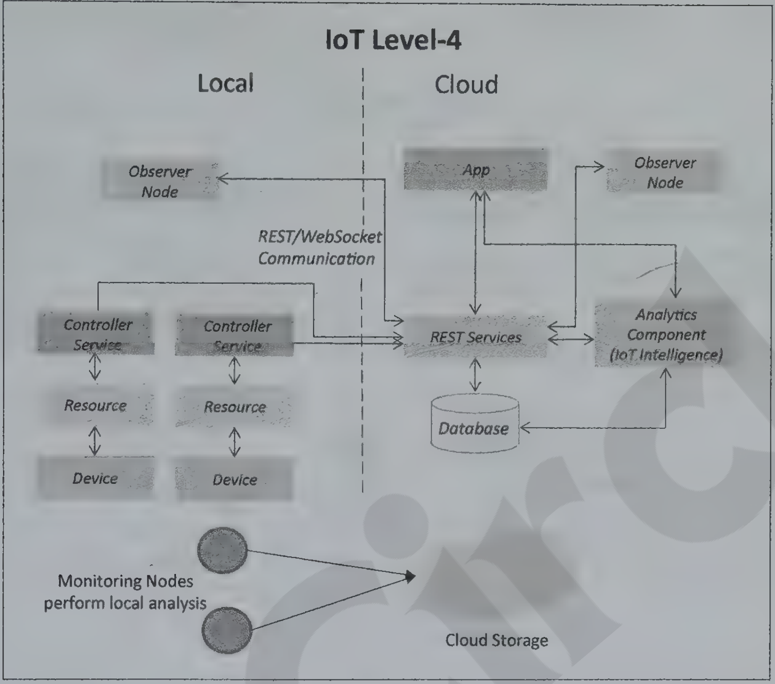


Figure 1.17: IoT Level-4

1.5.5 IoT Level-5

A level-5 IoT system has multiple end nodes and one coordinator node as shown in Figure 1.18. The end nodes that perform sensing and/or actuation. Coordinator node collects data from the end nodes and sends to the cloud. Data is stored and analyzed in the cloud and application is cloud-based. Level-5 IoT systems are suitable for solutions based on wireless sensor networks, in which the data involved is big and the analysis requirements are computationally intensive.

Let us consider an example of a level-5 IoT system for forest fire detection. The system consists of multiple nodes placed in different locations for monitoring temperature, humidity and carbon dioxide (CO_2) levels in a forest. The end nodes in this example are equipped with various sensors (such as temperature, humidity and CO_2). The coordinator node collects the data from the end nodes and acts as a gateway that provides Internet connectivity to the IoT system. The controller service on the coordinator device sends the collected data to the cloud. The data is stored in a cloud database. The analysis of data is done in the computing cloud to aggregate the data and make predictions. A cloud-based application is used for visualizing the data.

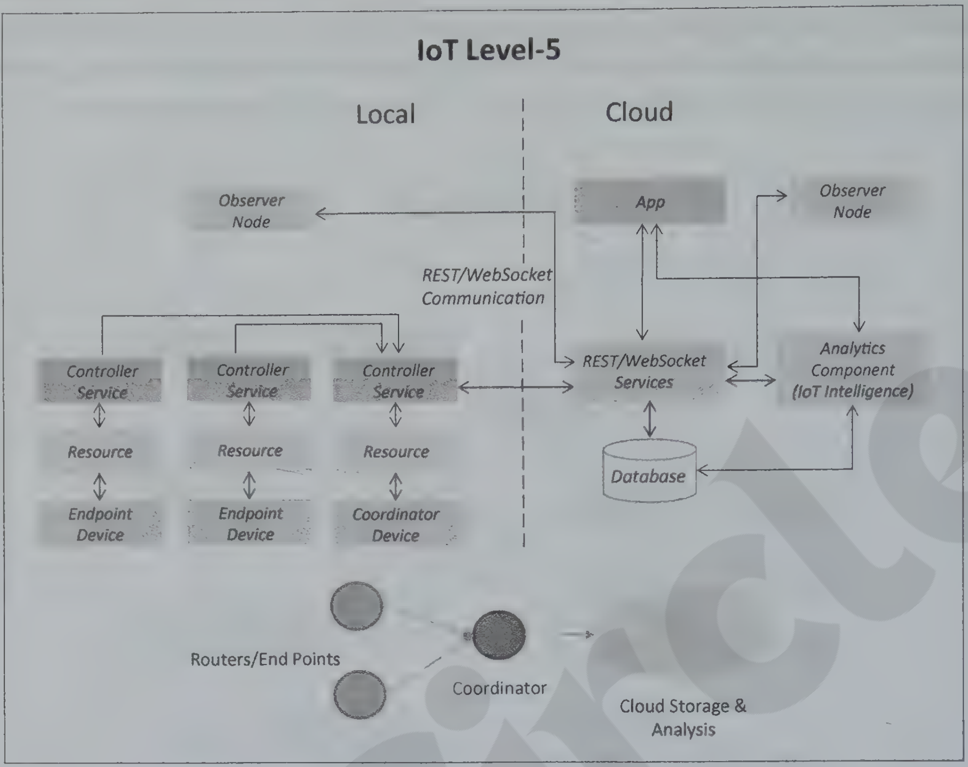


Figure 1.18: IoT Level-5

1.5.6 IoT Level-6

A level-6 IoT system has multiple independent end nodes that perform sensing and/or actuation and send data to the cloud. Data is stored in the cloud and application is cloud-based as shown in Figure 1.19. The analytics component analyzes the data and stores the results in the cloud database. The results are visualized with the cloud-based application. The centralized controller is aware of the status of all the end nodes and sends control commands to the nodes.

Let us consider an example of a level-6 IoT system for weather monitoring. The system consists of multiple nodes placed in different locations for monitoring temperature, humidity and pressure in an area. The end nodes are equipped with various sensors (such as temperature, pressure and humidity). The end nodes send the data to the cloud in real-time using a WebSocket service. The data is stored in a cloud database. The analysis of data is done in the cloud to aggregate the data and make predictions. A cloud-based application is used for visualizing the data.

Summary

Internet of Things (IoT) refers to physical and virtual objects that have unique identities and are connected to the Internet. This allows the development of intelligent applications that make energy, logistics, industrial control, retail, agriculture and many other domains of human endeavour "smarter". IoT allows different types of devices, appliances, users and

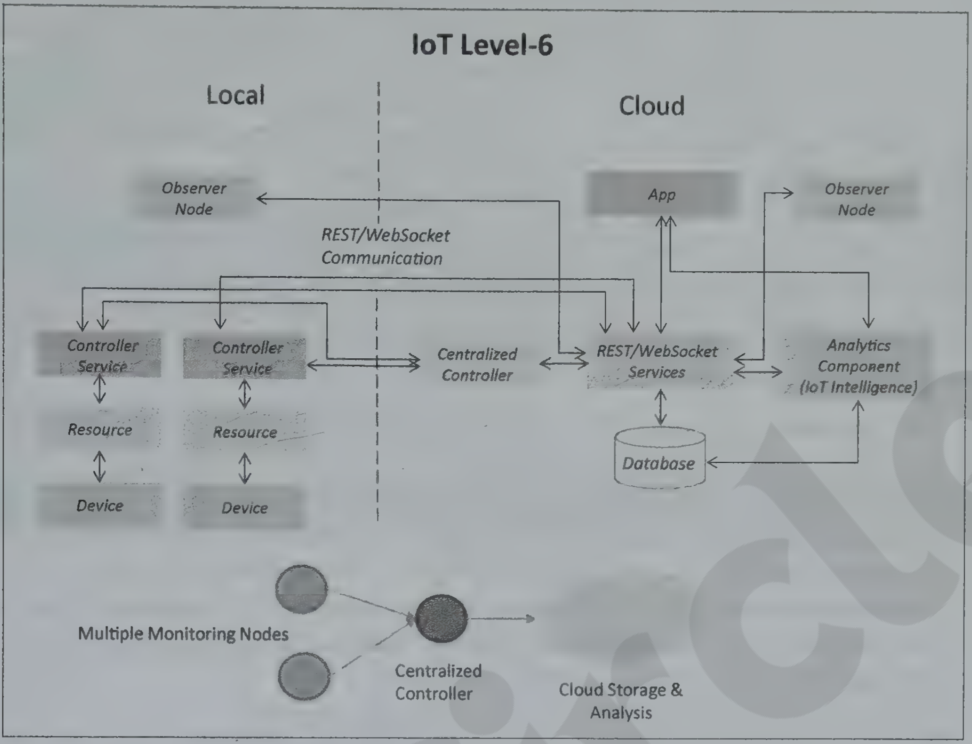


Figure 1.19: IoT Level-6

machines to communicate and exchange data. The applications of Internet of Things (IoT) span a wide range of domains including (but not limited to) homes, cities, environment, energy systems, retail, logistics, industry, agriculture and health. Things in IoT refers to IoT devices which have unique identities and allow remote sensing, actuating and remote monitoring capabilities. Almost all IoT devices generate data in some form or the other which when processed by data analytics systems leads to useful information to guide further actions. You learned about IoT protocols for link, network, transport and application layers. Link layer protocols determine how the data is physically sent over the network. The network/internet layers is responsible for sending of IP datagrams from the source network to the destination network. The transport layer protocols provides end-to-end message transfer capability independent of the underlying network. Application layer protocols define how the applications interface with the lower layer protocols to send the data over the network. You learned about functional blocks of an IoT system including device communication, services, management, security and application blocks. You learned about IoT communication models such as request-response, publish-subscribe, push-pull and exclusive pair. You learned about REST-based and WebSocket-based communication APIs. REST is a set of architectural principles by which you can design web services and web APIs that focus on a system's resources and how resource states are addressed and transferred. A RESTful web service is a web API implemented using HTTP and REST principles. WebSocket APIs allow bi-directional, full duplex communication between clients and servers. You learned about enabling technologies of IoT such as wireless sensor networks, cloud computing, big data analytics, communication protocols and embedded systems. Finally, you learned about IoT levels. A level-1 IoT system has a single node/device that performs sensing and/or actuation, stores data, performs analysis and hosts the application. A level-2 IoT system has a single node that performs sensing and/or actuation and local analysis. A level-3 IoT system has a single node. Data is stored and analyzed in the cloud and application is cloud-based. A level-4 IoT system has multiple nodes that perform local analysis. Data is stored in the cloud and application is cloud-based. A level-5 IoT system has multiple end nodes and one coordinator node. A level-6 IoT system has multiple independent end nodes that perform sensing and/or actuation and send data to the cloud.

Review Questions

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