MODULE 5

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].

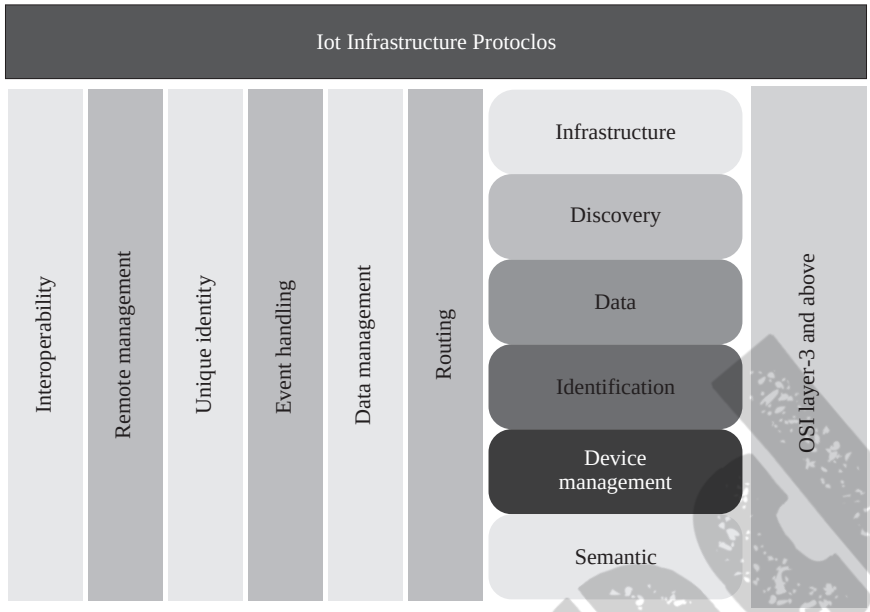


Figure 8.1 Various IoT communication protocol groups

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

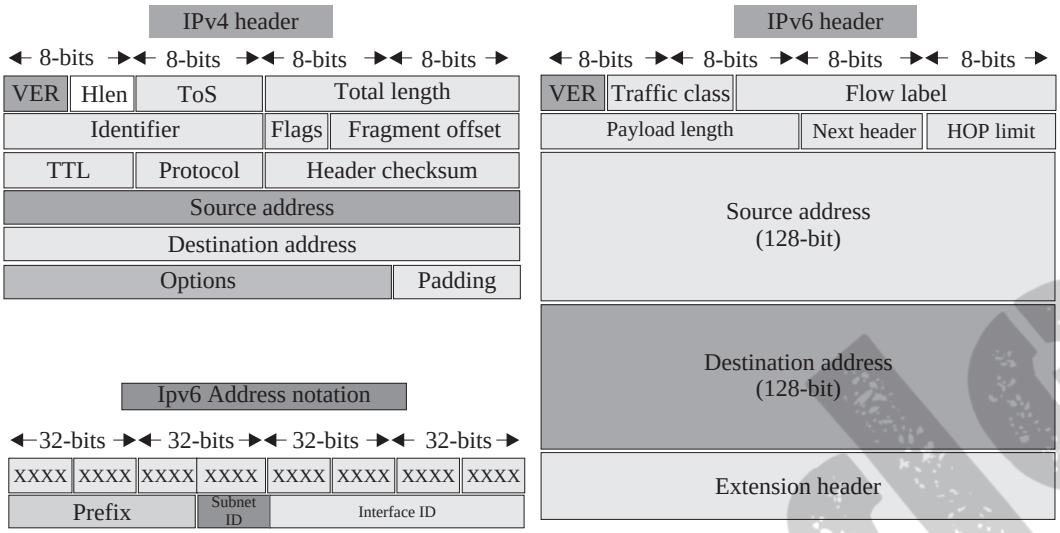


Figure 8.2 Differences between IPv4 and IPv6 packets and the IPv6 address notation

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.

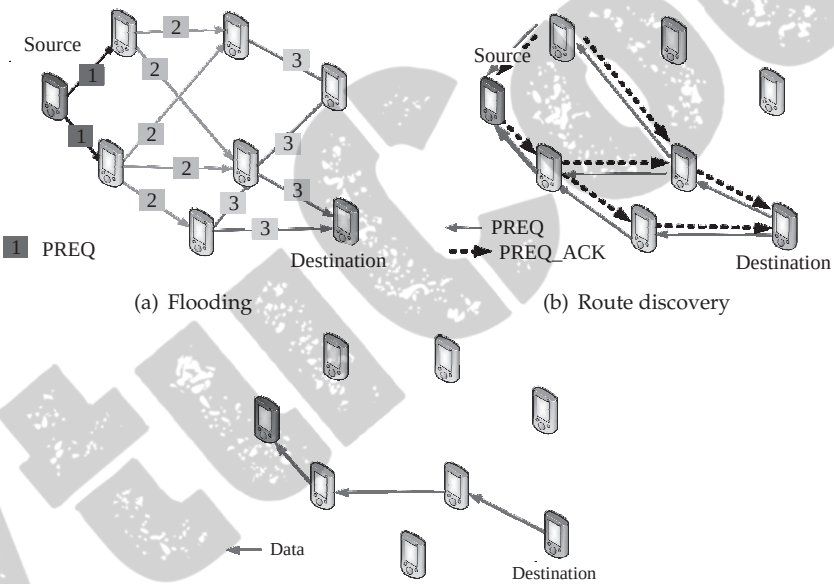


Figure 8.3 The LOADng routing mechanism

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.

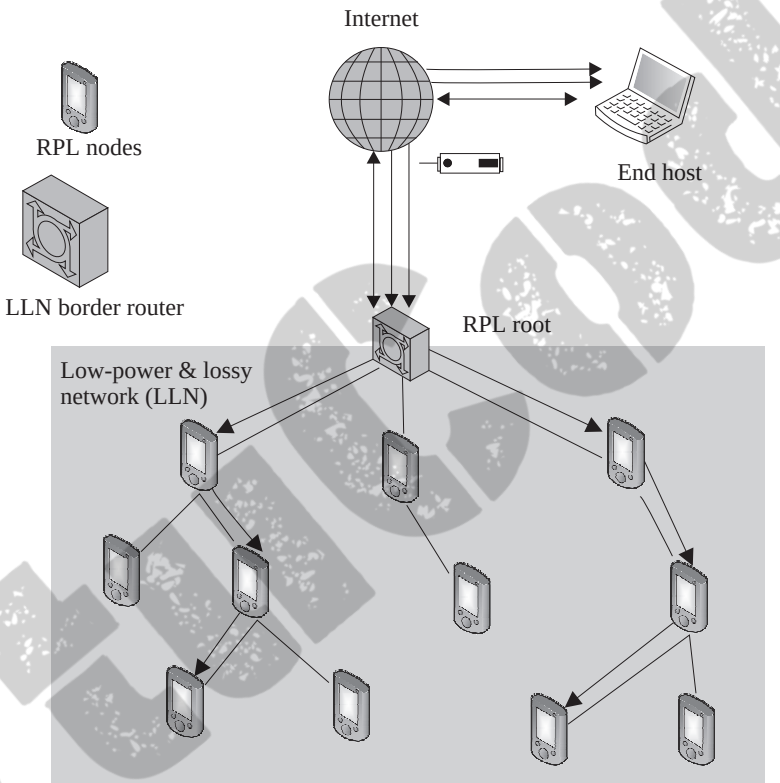


Figure 8.4 RPL information flow mechanism with different intra-mesh addressing and paths

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:

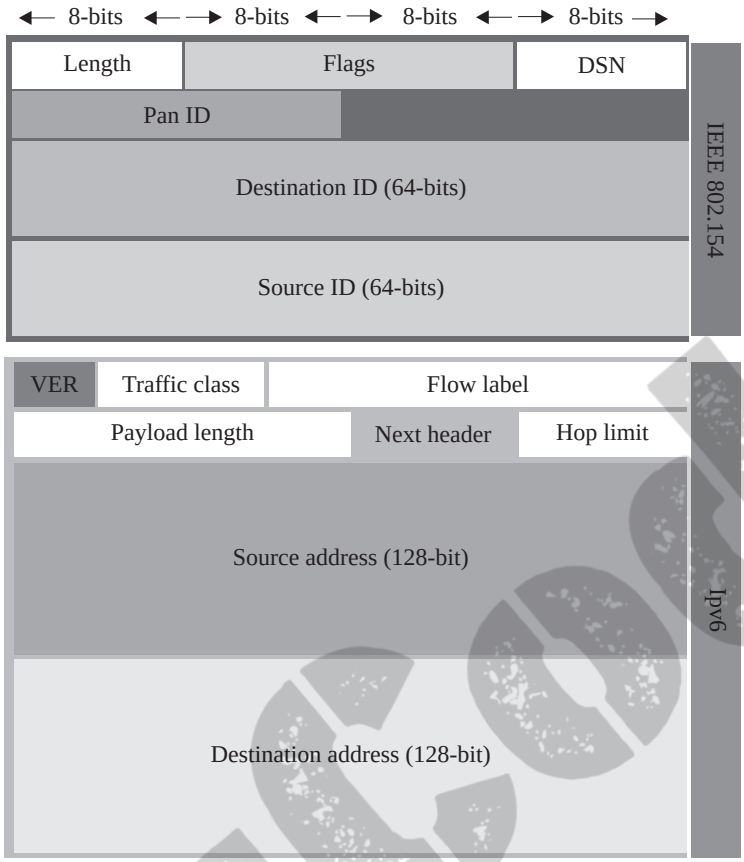


Figure 8.5 6LoWPAN packet structure

(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.

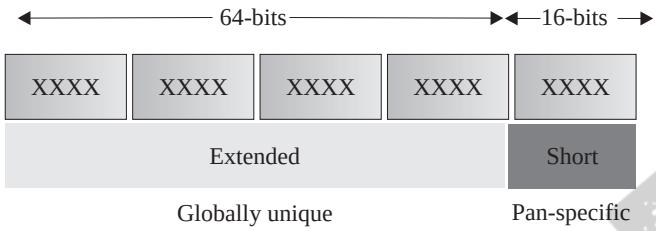


Figure 8.6 6LoWPAN address format

- **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

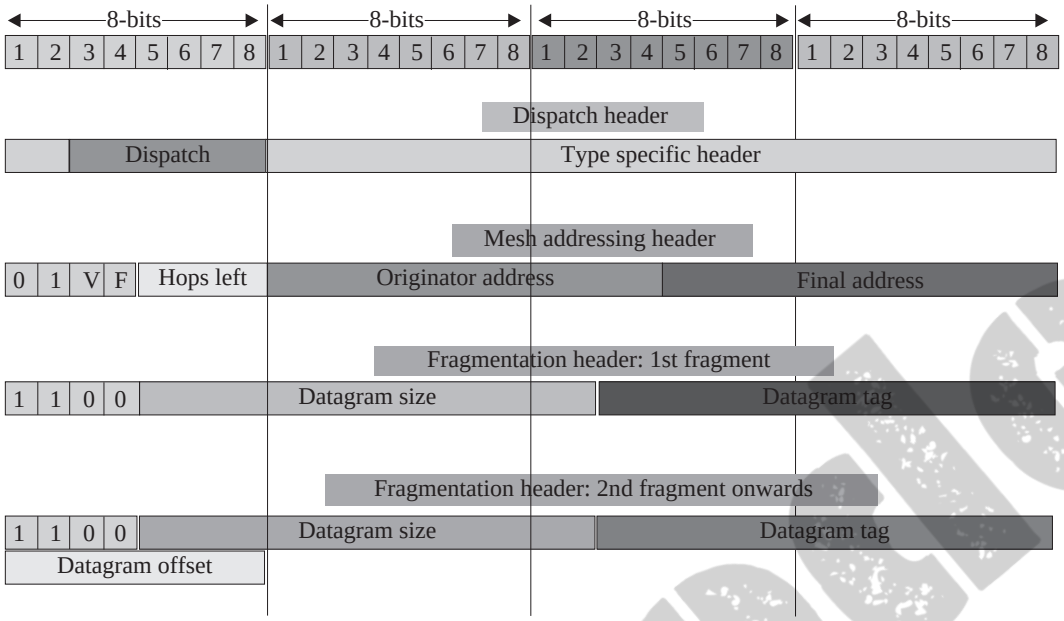


Figure 8.7 6LoWPAN header structures

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.

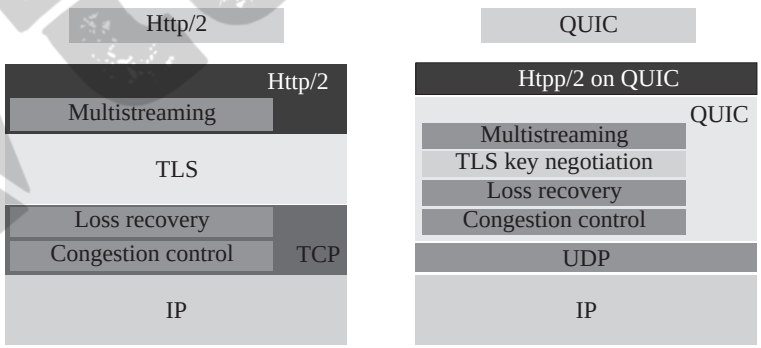


Figure 8.8 Differences between HTTP and QUIC protocols

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.

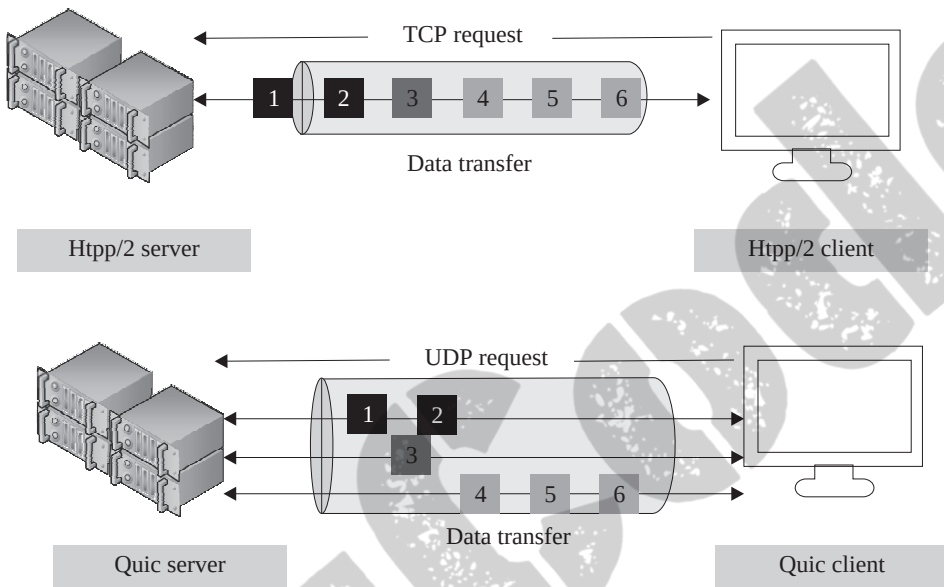


Figure 8.9 Differences between stream of packets over HTTP and QUIC protocols

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.

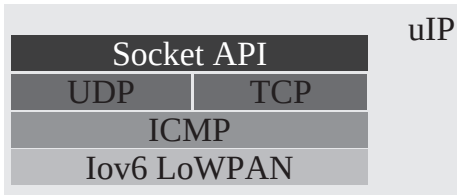


Figure 8.10 The uIP protocol

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.

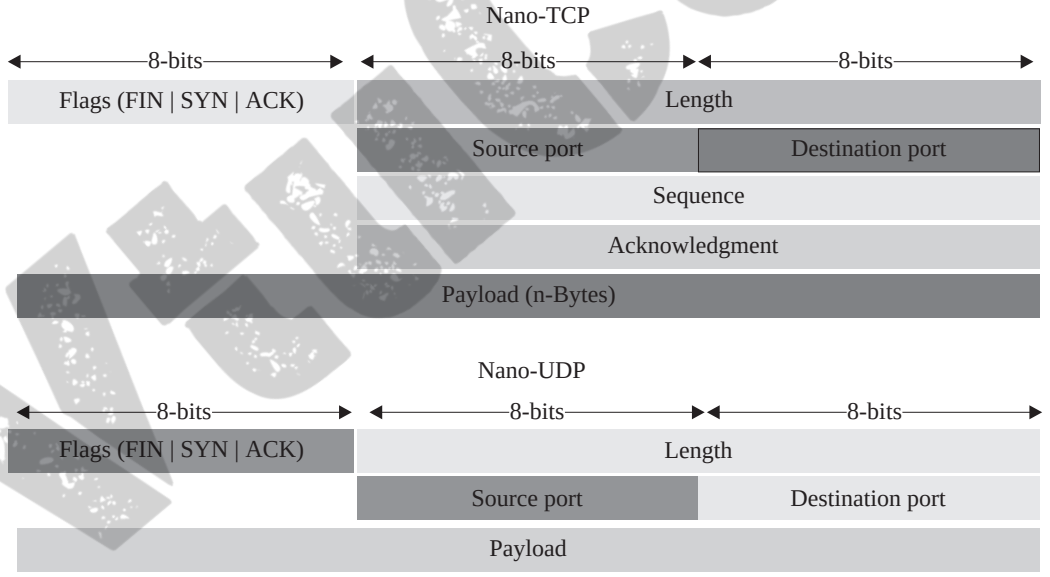


Figure 8.11 The nano-IP TCP and UDP protocols

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).

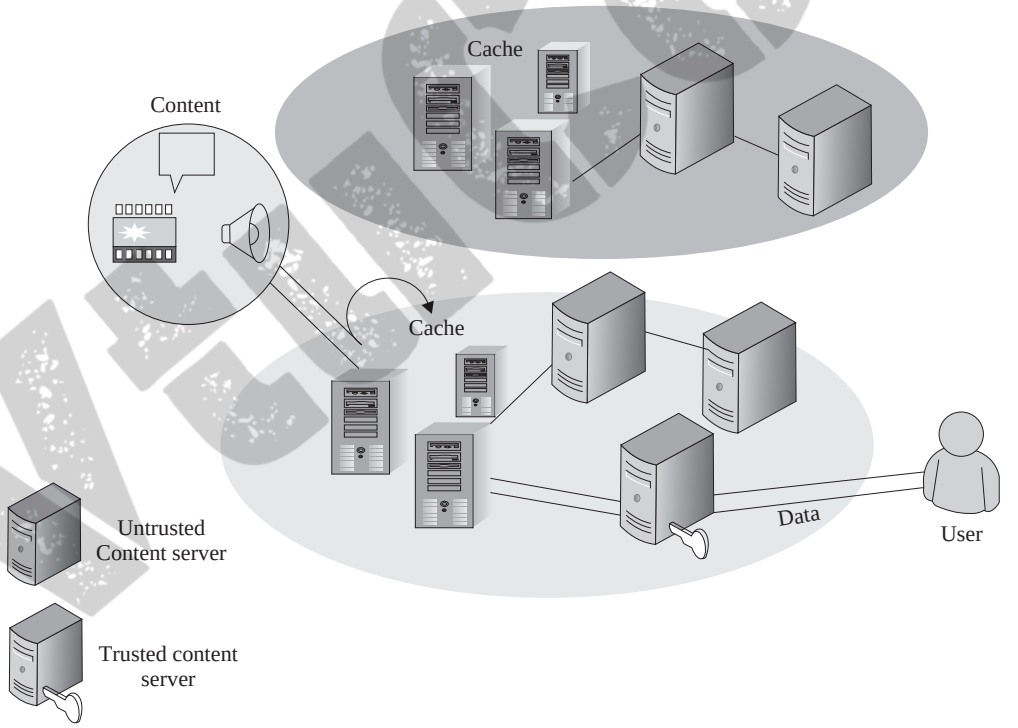


Figure 8.12 Content centric networking operation and its scope

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.

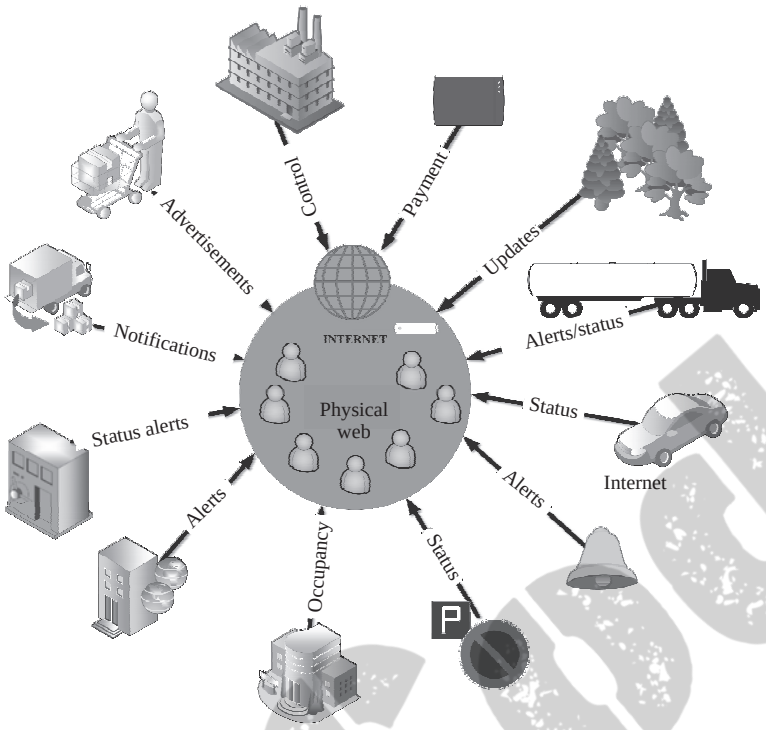


Figure 8.13 The physical web model

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

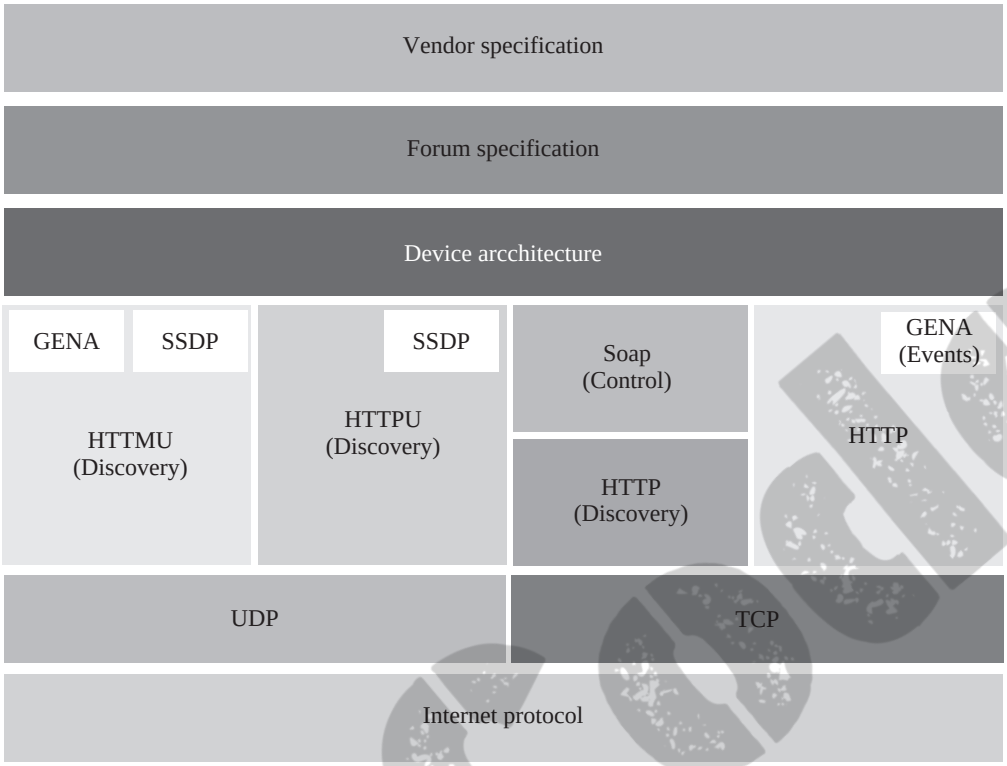


Figure 8.14 An outline of the basic UPnP stack

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.

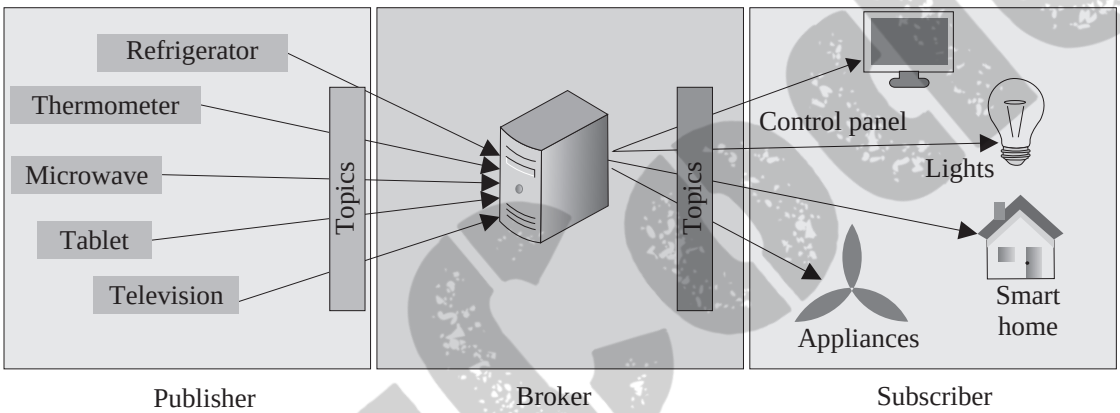


Figure 8.15 MQTT operation and its stakeholders

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

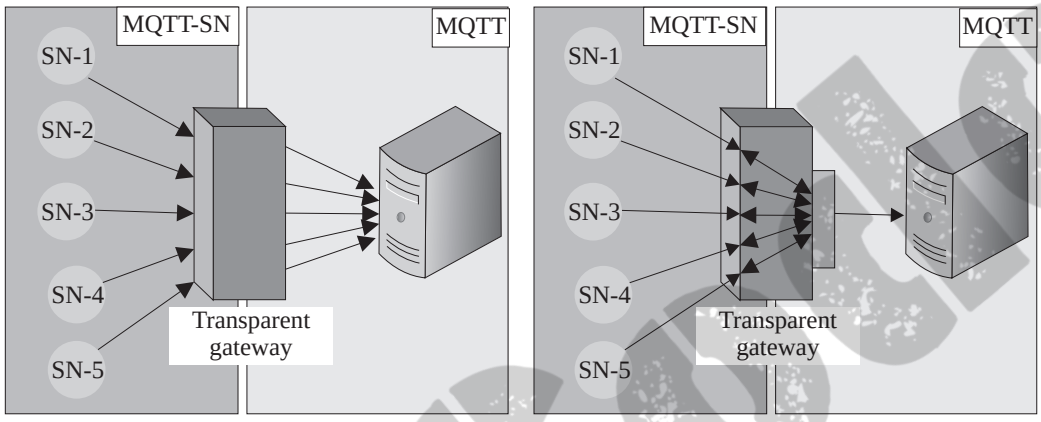
MQTT clients and servers available online, implementing MQTT

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.

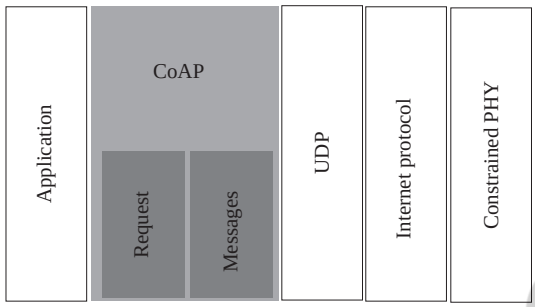


Figure 8.17 Position of the CoAP protocol in a stack

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.

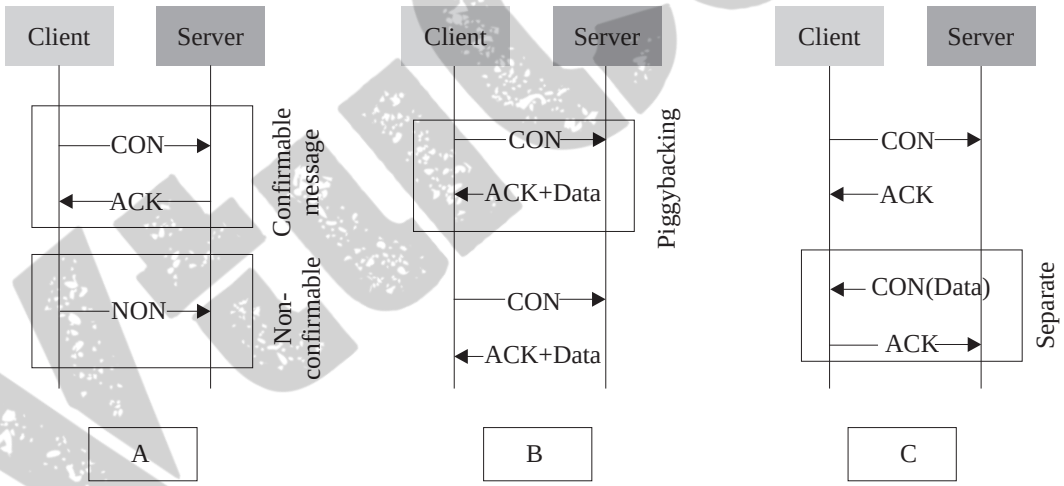


Figure 8.18 Various CoAP response–response models. (A): CON and NON messages, (B): Piggyback messages, and (C): Separate messages

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

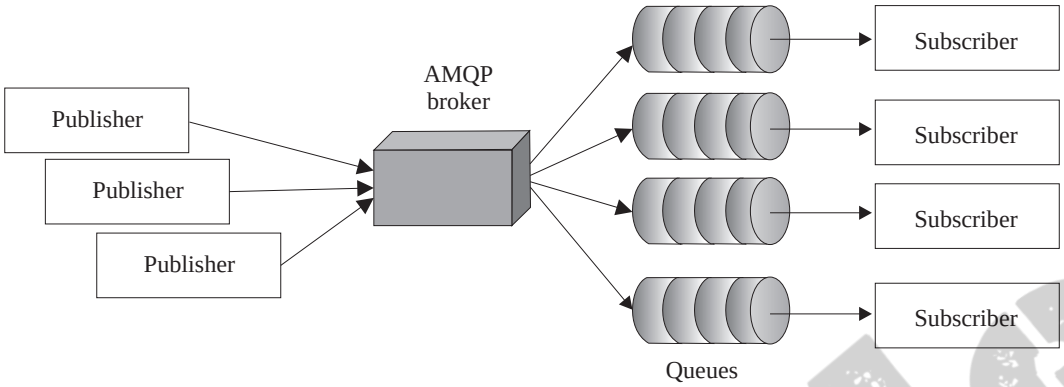


Figure 8.19 AMQP components and their relationships

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.

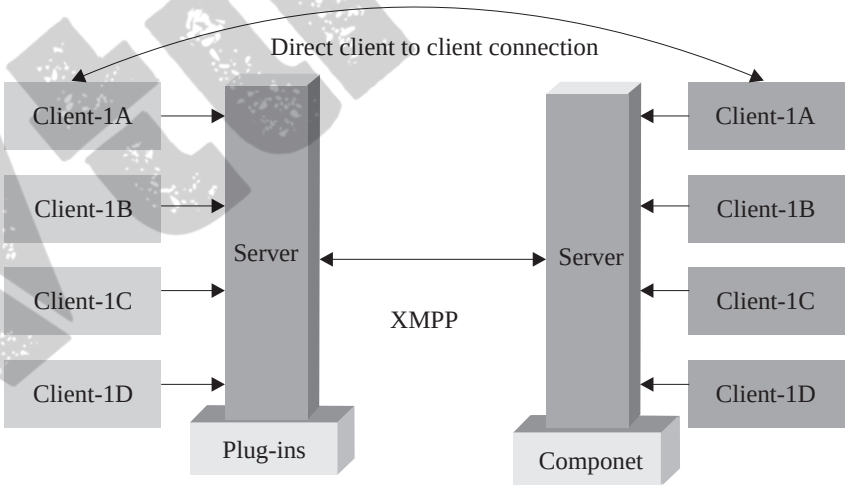


Figure 8.20 XMPP components

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.

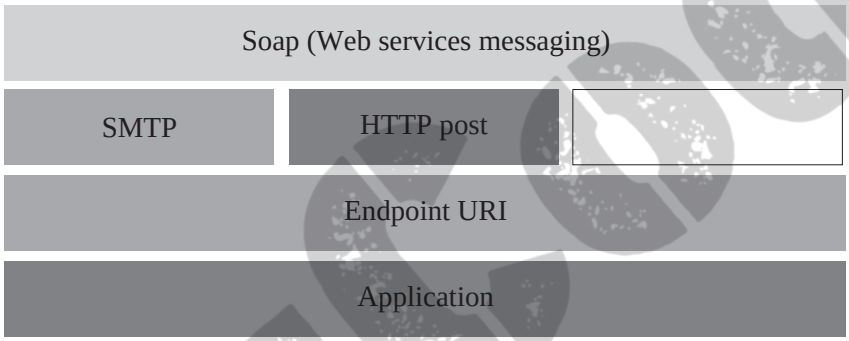


Figure 8.21 A representation of the position of the SOAP API in a stack

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

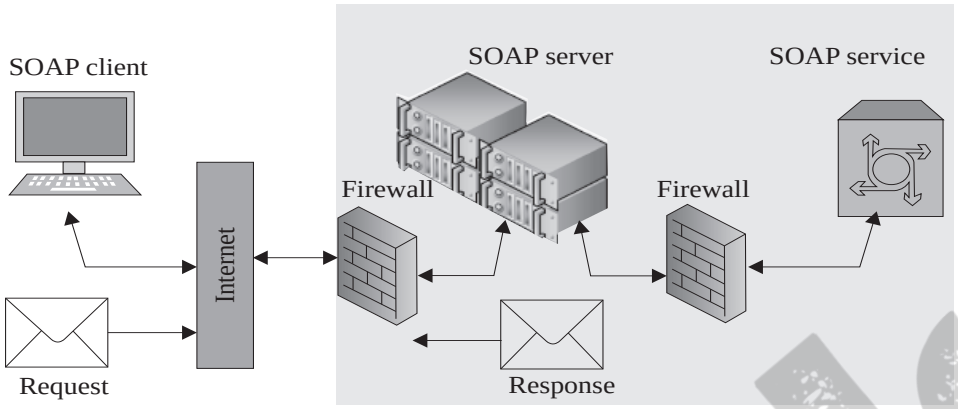


Figure 8.22 Working of SOAP

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.

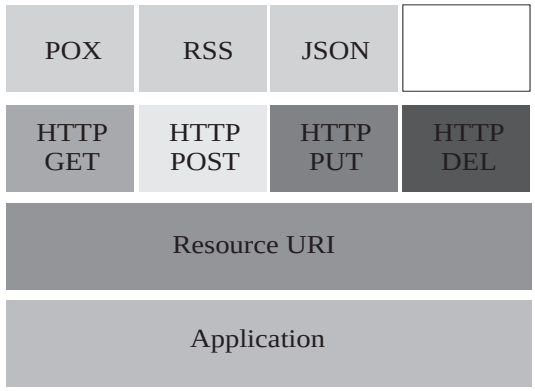


Figure 8.23 A representation of the REST style and its components

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

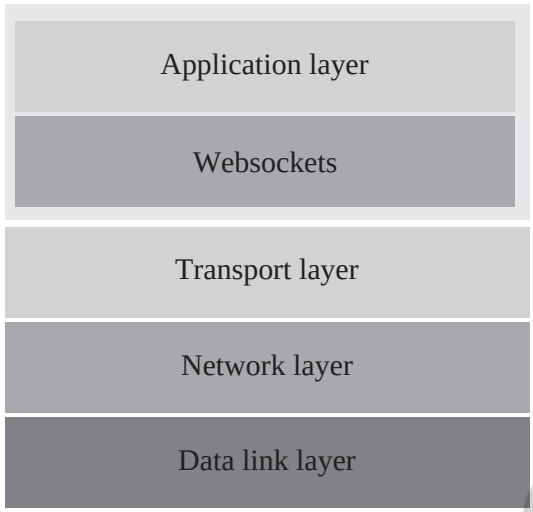


Figure 8.24 A representation of the position of websockets in a stack

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*.

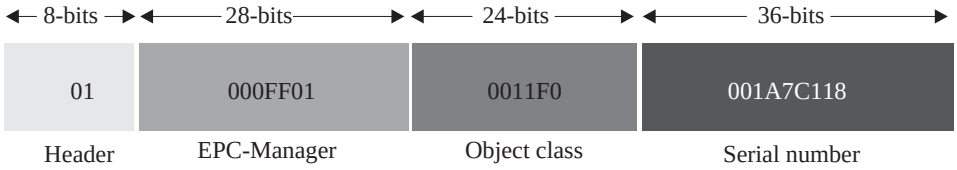


Figure 8.25 The EPC representation

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.

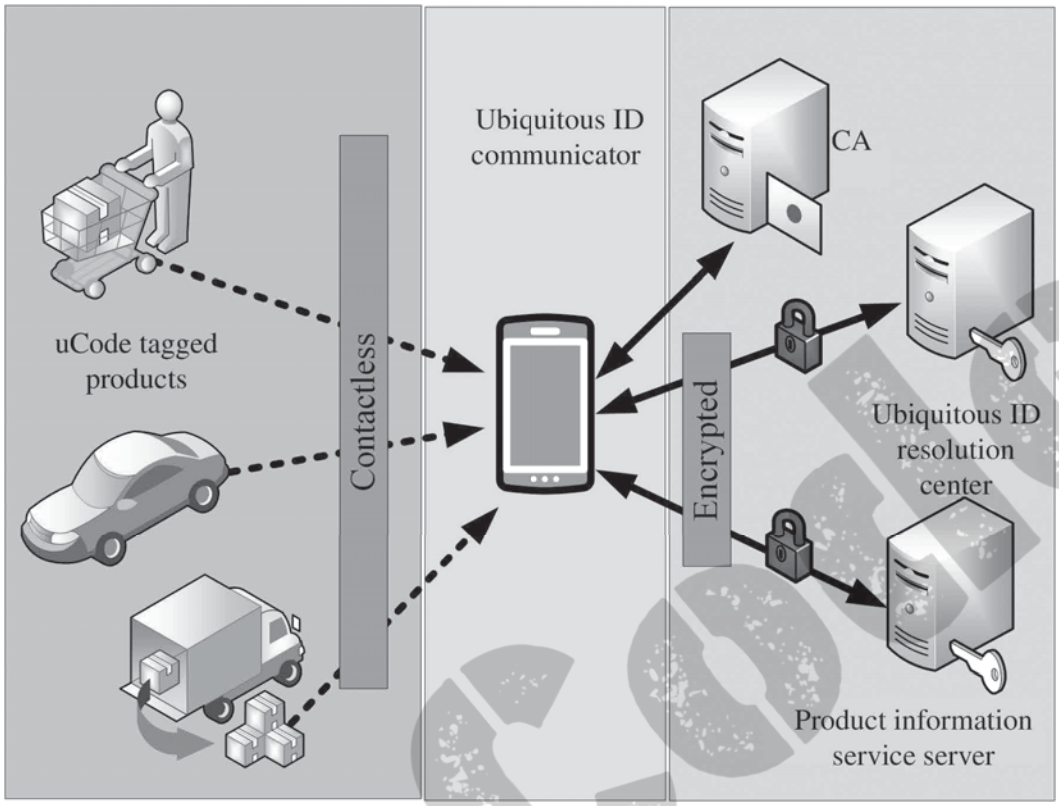


Figure 8.26 The operation of an uCode tag system

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.

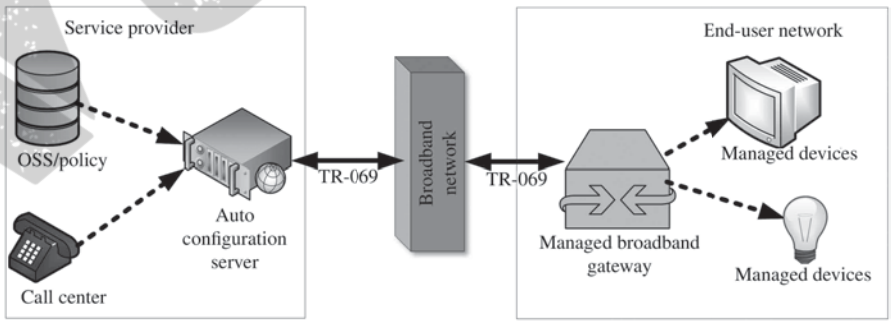


Figure 8.28 The various components of TR-069 and their inter-relations

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.

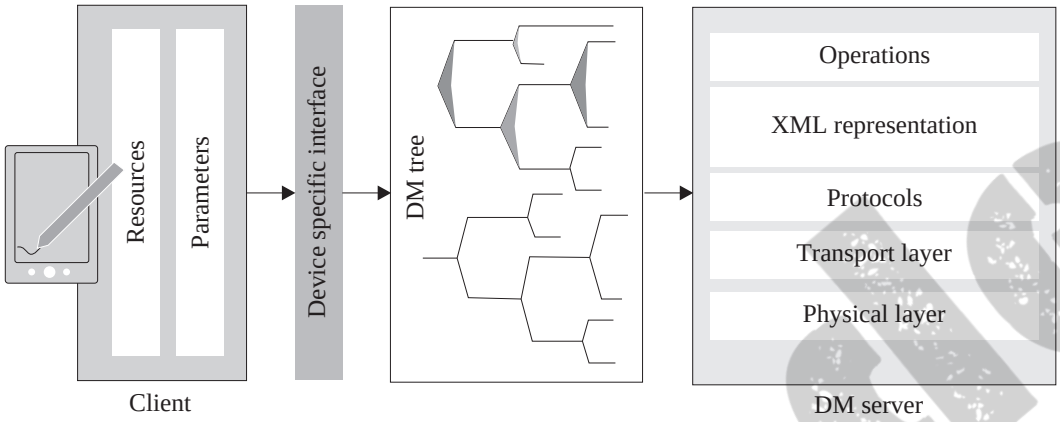


Figure 8.29 Communication between an OMA-DM client and a server

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.

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:

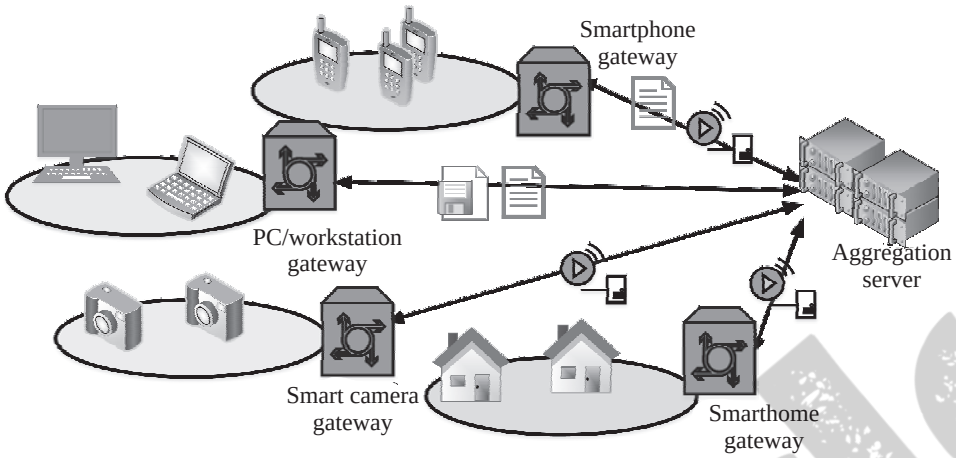


Figure 9.1 An illustration of the various facets of interoperability in IoT

(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

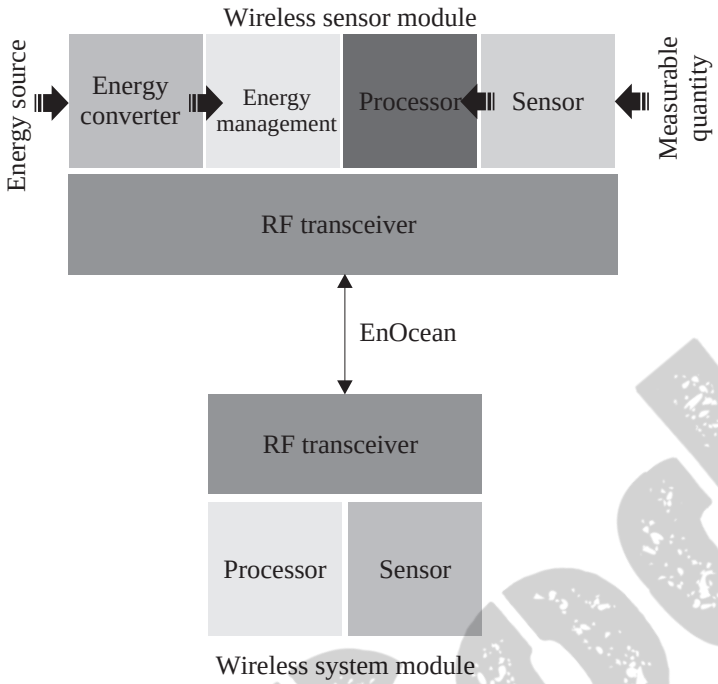


Figure 9.2 A representation of the major constituents of EnOcean devices

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].

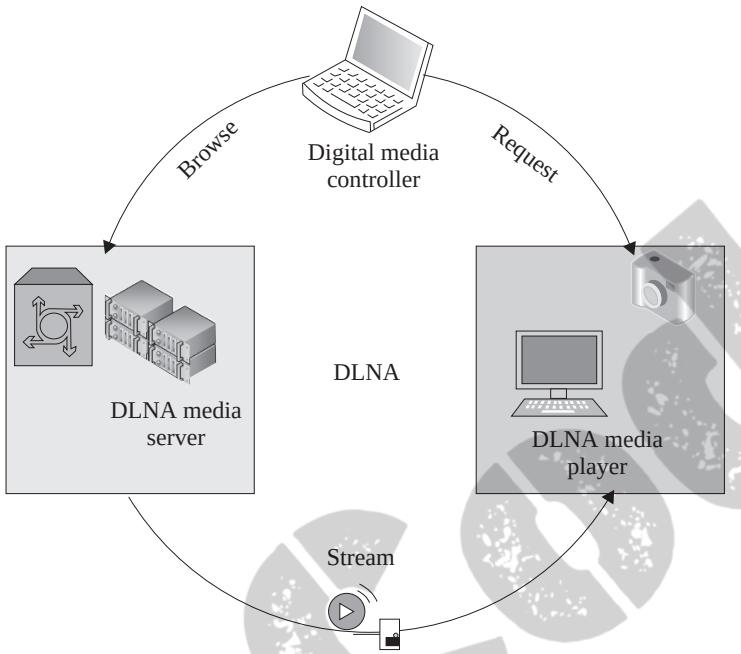


Figure 9.3 A representation of the various roles in a DLNA-based media streaming application

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.

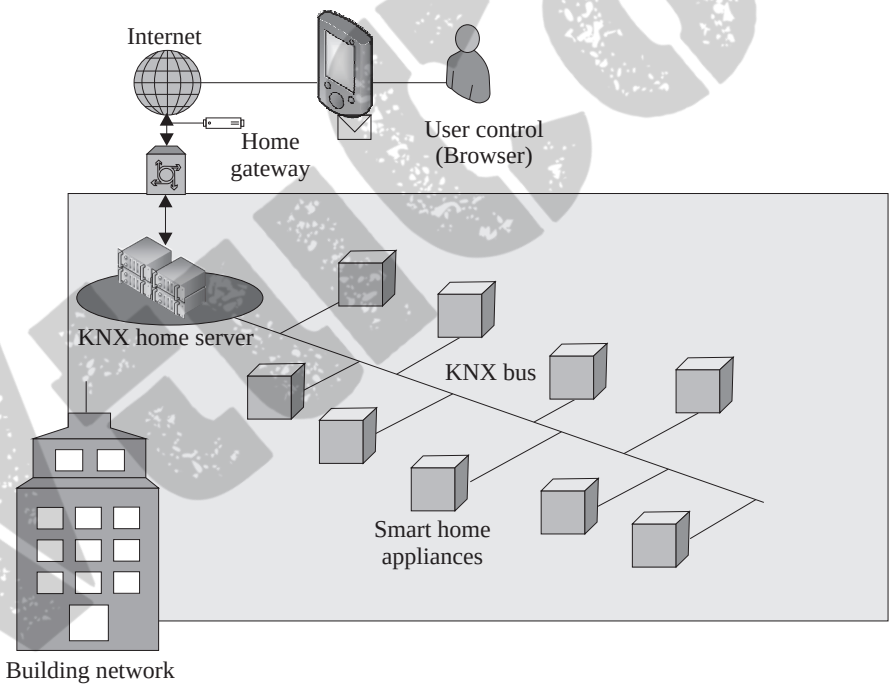


Figure 9.4 A representation of the Konnex network

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.

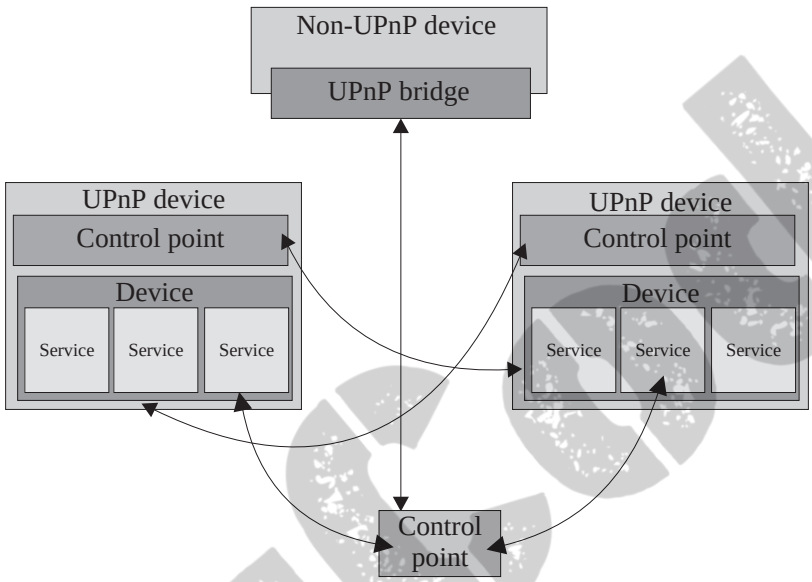


Figure 9.5 A representation of the UPnP operation

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,

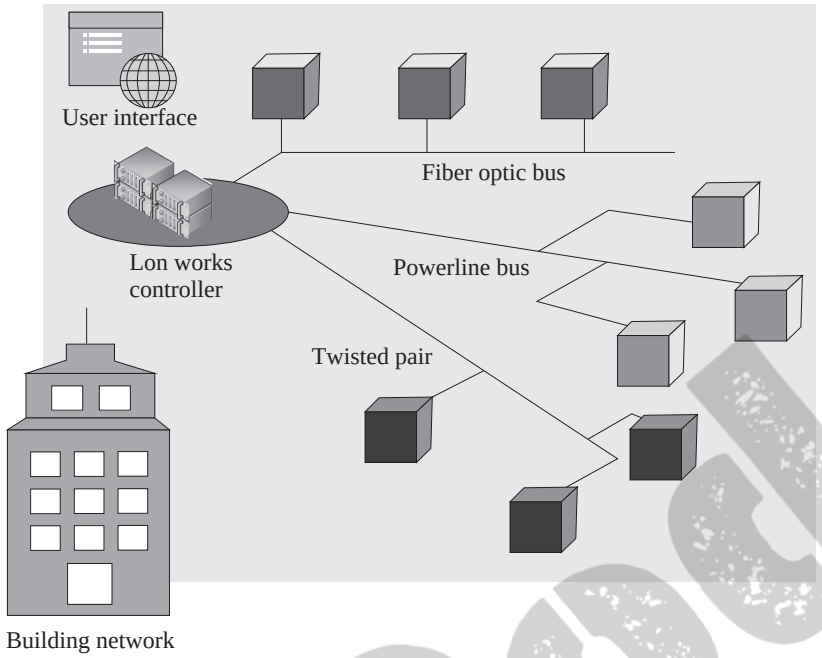


Figure 9.6 A representation of the LonWorks network

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.

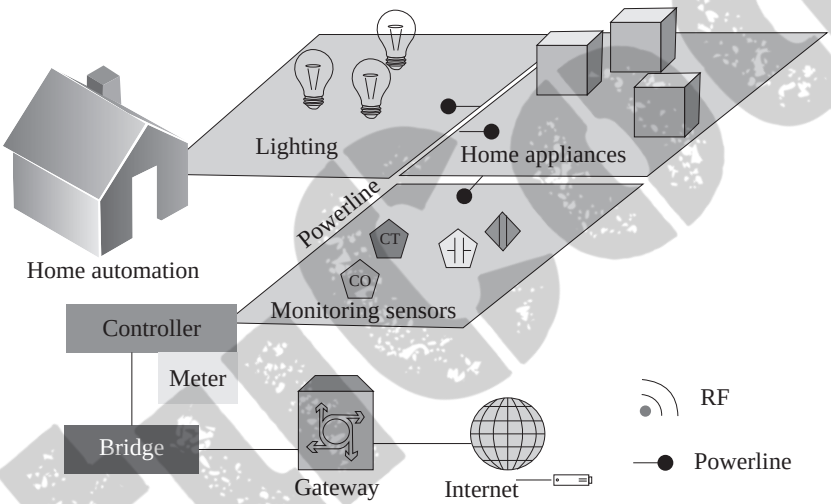


Figure 9.7 A representation of an Insteon network

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.

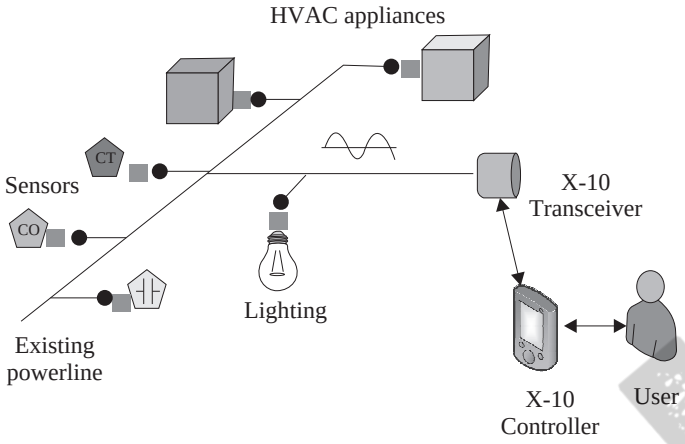


Figure 9.8 A representation of the X-10 network

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.