MODULE 3

IP AS THE IOT NETWORKLAYER

Internet Protocol (IP), which has become the standard in many areas of IoT. With support from numerous standards and industry organizations, IP and its role as the network layer transport for IoT is a foundational element that has to be familiarized with.

3.1 The Business Case for IP

Data flowing from or to "things" is consumed, controlled, or monitored by data center servers either in the cloud or in locations that may be distributed or centralized. Dedicated applications are then run over virtualized or traditional operating systems or on network edge platforms (for example, fog computing). These lightweight applications communicate with thedata center servers. Therefore, thesystem solutions combining various physical and data link layers call for an architectural approach with a common layer(s) independent from the lower (connectivity) and/or upper (application) layers. This is how and why the Internet Protocol (IP) suite started playing a key architectural role in the early 1990s. IP was not only preferred in the IT markets but also for the OT environment.

3.1.1 The Key Advantages of Internet Protocol

One of themain differences between traditional information technology (IT) and operational technology (OT) is the lifetime of the underlying technologies and products. One way to guarantee multi-year lifetimes is to define a layered architecture such as the 30-year-old IP architecture. IP has largely demonstrated its ability to integrate small and large evolutions. At the same time, it is able to maintain its operations for large numbers of devices and users, such as the 3 billion Internet users. Before evaluating the pros and cons of IP adoption versus adaptation, this section provides a quick review of the key advantages of the IP suite for the Internet of Things:

- **Open and standards-based:** Operational technologies have often been delivered as turnkey features by vendors who may have optimized the communications through closed and proprietary networking solutions. The Internet of Things creates a new paradigm in which devices, applications, and users can leverage a large set of devices and functionalities while guaranteeing interchangeability and interoperability, security, and management. This calls for implementation, validation, and deployment of open, standards-based solutions. While manystandards development organizations (SDOs) are working on Internet of Things definitions, frameworks, applications, and technologies, none are questioning the role of the Internet Engineering Task Force (IETF) as the foundation for specifying and optimizing the network and transport layers. The IETF is an open standards body that focuses on the development of the Internet Protocol suite and related Internet technologies andprotocols.

- **Versatile:** A large spectrum of access technologies is available to offer connectivity of "things" in the last mile. Additional protocols and technologies are also used to transport IoTdata throughbackhaul links and in the data center. Even if physical and data link layers such as Ethernet, Wi-Fi, and cellular are widely adopted, the history of data communications demonstrates that no given wired or wireless

technology fits all deployment criteria. Furthermore, communication technologies evolve at a pace faster than the expected 10- to 20- year lifetime of OT solutions. So, the layered IP architecture is well equipped to cope with any type of physical and data link layers. This makes IP ideal as a long-term investment because various protocols at these layers can be used in a deployment now and over time, without requiring changes tothe whole solution architecture and data flow.

- **Ubiquitous:** All recent operating system releases, from general-purpose computers and servers to lightweight embedded systems (TinyOS, Contiki, and so on), have an integrated dual (IPv4 and IPv6) IP stack that gets enhanced over time. In addition, IoT application protocols in many industrial OT solutions have been updated in recent years to run over IP. While these updates have mostly consisted of IPv4 to this point, recent standardization efforts in several areas are adding IPv6.

- **Scalable:** As the common protocol of the Internet, IP has been massively deployed and tested for robust scalability. Millions of private and public IP infrastructure nodes have been operational for years, offering strong foundations for those not familiar with IP network management. Of course, adding huge numbers of "things" to private and public infrastructures may require optimizations and design rules specific to the newdevices.

- **Manageable and highly secure:** Communications infrastructure requires appropriate management and security capabilities for proper operations. One of the benefits that comes from 30 years of operational IP networks is the well-understood network management and security protocols, mechanisms, and toolsets that are widely available. Adopting IP network managementalso brings an operational business application to OT. Well-known network and security managementtools are easily leveraged with an IP network layer.

- **Stable and resilient:** IP has been around for 30 years, and it is clear that IP is a workable solution. IP has a large and well-established knowledge base and, more importantly, it has been used for years in critical infrastructures, such as financial and defense networks. In addition, IP has been deployed for critical services, such as voice and video, which have already transitioned from closed environments to open IP standards. Finally, its stability and resiliency benefit from the large ecosystem of IT professionals who can help design, deploy, and operate IP-based solutions.

- **Consumers' market adoption:** When developing IoT solutions and products targeting the consumer market, vendors knowthat consumers' access to applications and devices will occur predominantly over broadband and mobile wireless infrastructure. The main consumer devices range from smart phones to tablets and PCs. The common protocol that links IoT in the consumer space to these devices is IP.

- **The innovation factor:** The past two decades have largely established the adoption of IP as a factor for increased innovation. IP is the underlying protocol for applications ranging from file transfer and e-mail to the World Wide Web, e-commerce, social networking, mobility, and more. Even the recent computingevolution from PC to mobile and mainframes to cloud services are perfect demonstrations of the innovative ground enabled by IP.

3.1.2 Adoption or Adaptation of the Internet Protocol

How to implement IP in data center, cloud services, and operation centers hosting IoT applications mayseem

obvious, but the adoption of IP in the last mile is more complicated and often makes running IP end-to-end more difficult. The use of numerous network layer protocols in addition to IP is often a point of contention between computer networking experts. Typically, one of two models, adaptation or adoption, is proposed:

- ***Adaptation*** means application layered gateways (ALGs) must be implemented to ensure the translation between non-IP and IPlayers.
- ***Adoption*** involves replacing all non-IP layers with their IP layer counterparts, simplifying the deployment model and operations.

A similar transition is now occurring with IoT and its use of IP connectivity in the last mile. While IP is slowly becoming more prevalent, alternative protocol stacks are still often used. Let's look at a few examples in various industries tosee how IP adaptation and adoption are currentlyapplied to IoT last-mile connectivity.

In the industrial and manufacturing sector, there has been a move toward IP adoption. Solutions and product lifecycles in this space are spread over 10+ years, and many protocols have been developed for serial communications. While IP and Ethernet support were not specified in the initial versions, more recent specifications for these serial communications protocols integrate Ethernet and IPv4.

Supervisory control and data acquisition (SCADA) applications are typical examples of vertical market deployments that operate both the IP adaptation model and the adoption model. Found at the core of many modern industries, SCADA is an automation control system for remote monitoring and control of equipment. Implementations that make use of IP adaptation have SCADA devices attached through serial interfaces to a gateway tunneling or translating the traffic. With the IP adoption model, SCADA devices are attached via Ethernet to switches and routers forwarding their IPv4 traffic.

Another example is a ZigBee solution that runs a non-IP stack between devices and a ZigBee gateway that forwards traffic to an application server. A ZigBee gateway often acts as a translator between the ZigBee and IP protocol stacks. As highlighted by these examples, the IP adaptation versus adoption model still requires investigation for particular last-mile technologies used by IoT.

The following factors determine which model is best suited for last-mile connectivity:

- **Bidirectional versus unidirectional data flow:** While bidirectional communications are generally expected, some last-mile technologies offer optimization for unidirectional communication. For example, different classes of IoT devices, as defined in RFC 7228, may only infrequently need to report a few bytes of data to an application. These sorts of devices, particularly ones that communicate through LPWA technologies, include fire alarms sending alerts or daily test reports, electrical switches being pushed on or off, and water or gas meters sending weekly indexes. For these cases, it is not necessarily worth implementing a full IP stack. However, it requires the overall end-to-end architecture to solve potential drawbacks; for example, if there is only one-way communication to upload data to an application, then it is not possible to download new software or firmware to the devices. This makes integrating new features and bug and security fixes moredifficult.
- **Overhead for last-mile communications paths:** IP adoption implies a layered architecture with a per-packet overhead that varies depending on the IP version. IPv4 has 20 bytes of header at a minimum, and IPv6 has 40 bytes at the IP network layer. For the IP transport layer, UDP has 8 bytes of header overhead, while TCP has a minimum of 20 bytes. If the data to be forwarded by a device is

infrequent and only a few bytes, then it can potentially have more header overhead than device data—again, particularly in the case of LPWA technologies. Consequently, there is a need to decide whether the IP adoption model is necessary and, if it is, how it can be optimized. This same consideration applies to control plane traffic that is run over IP for low-bandwidth, last- mile links. Routing protocol and other verbose network services may either not be required or call for optimization.

- **Data flow model:** One benefit of the IP adoption model is the end-to-end nature of communications. Any node can easily exchange data with any other node in a network, although security, privacy, and other factors mayput controls and limits on the "end-to-end" concept. However, in many IoT solutions, a device's data flow is limited to one or two applications. In this case, theadaptation model can work because translation of traffic needs to occur only between the end device and one or two application servers. Depending on the network topology and the data flow needed, both IP adaptation and adoption models have roles to play in last- mile connectivity.
- **Network diversity:** One of the drawbacks of the adaptation model is a general dependency on single PHY and MAC layers. For example, ZigBee devices must only be deployed in ZigBee network islands. This same restriction holds for ITU G.9903 G3-PLC nodes. Therefore, a deployment must consider which applications have to run on the gateway connecting these islands and the rest of the world. Integration and coexistence of new physical and MAC layers or new applications impact how deployment and operations have to be planned. This is not a relevant consideration for the adoption model.

3.2 The Need for Optimization

The following sections take a detailed look at why optimization is necessary for IP. Both the nodes and the network itself can often be constrained in IoT solutions. Also, IP is transitioning from version 4 to version 6, which can add further confinements in the IoT space.

3.2.1 Constrained Nodes

Another limit is that this network protocol stack on an IoT node may be required to communicate through an unreliable path. Even if a full IP stack is available on the node, this causes problems such as limited or unpredictable throughput and low convergence when a topology change occurs.

Finally, power consumption is a key characteristic of constrained nodes. Many IoT devices are battery powered, with lifetime battery requirements varying from a few months to 10+ years. This drives theselection of networking technologies since high-speed ones, such as Ethernet, Wi-Fi, and cellular, are not (yet) capable of multi-year battery life. Current capabilities practically allow less than a year for these technologies on battery-powered nodes. Of course, power consumption is muchless of a concern on nodes that do not require batteries as an energy source.

The power consumption requirements on battery-powered nodes impact communication intervals. To help extend battery life, enable a "low-power" mode instead of one that is "always on." Another option is "always off," which means communications are enabled only when needed to senddata.

While it has been largely demonstrated that production IP stacks perform well in constrained nodes,

classification of these nodes helps when evaluating the IP adoption versus adaptation model selection. IoT constrained nodes can be classified as follows:

- **Devices that are very constrained in resources, may communicate infrequently to transmit a few bytes, and may have limited security and management capabilities:** This drives the need for the IP adaptation model, where nodes communicate through gateways and proxies.
- **Devices with enough power and capacities to implement a stripped-down IP stack or non- IP stack:** In this case, either an optimized IP stack and directly communicate with application servers (adoption model) or go for an IP or non-IP stack and communicate through gateways and proxies (adaptation model) can be implemented.
- **Devices that are similar to generic PCs in terms of computing and power resources but have constrained networking capacities, such as bandwidth:** These nodes usually implement a full IP stack (adoption model), but network design and application behaviors must cope with the bandwidth constraints.

3.2.2 Constrained Networks

Constrained networks have unique characteristics and requirements. In contrast with typical IP networks, where highly stable and fast links are available, constrained networks are limited by low- power, low-bandwidth links (wireless and wired). They operate between a few kbps and a few hundred kbps and may utilizea star, mesh, or combinednetwork topologies, ensuringproper operations. Witha constrained network,in addition to limited bandwidth, itis not unusualfor thepacket delivery rate (PDR) to oscillate between low and high percentages. Large bursts of unpredictable errors and even loss of connectivity at times may occur. These behaviors can be observed on both wireless and narrowband power-line communication links, where packet delivery variation may fluctuate greatly during the course of aday.

Unstable link layer environments create other challenges in terms of latency and control plane reactivity. One of the golden rules in a constrained network is to "underreact to failure." Due to the low bandwidth, a constrained network that overreacts can lead to a network collapse—which makes the existing problem worse. Control plane traffic must also be kept at a minimum; otherwise, it consumes the bandwidth that is needed by the data traffic. Finally, the power consumption in battery-powered nodes has to be considered. Any failure or verbose control plane protocol may reduce the lifetime of the batteries.

3.2.3 IP Versions

For 20+ years, the IETF has been working on transitioning the Internet from IP version 4 to IP version 6. The main driving force has been the lack of address space in IPv4 as the Internet has grown. IPv6 has a much larger range of addresses that should not be exhausted for the foreseeable future. Today, both versions of IP run over the Internet, but most traffic is still IPv4 based.

While it may seem natural to base all IoT deployments on IPv6, current infrastructures and their associated lifecycle of solutions, protocols, and products need to be taken into account. IPv4 is entrenched in these current infrastructures, and so support for it is required in most cases. Therefore, the Internet of Things

has to follow a similar path as the Internet itself and support both IPv4 and IPv6 versions concurrently. Techniques such as tunneling and translation need to be employed in IoT solutions to ensure interoperability between IPv4 and IPv6.

A variety of factors dictate whether IPv4, IPv6, or both can be used in an IoT solution. Most often these factors include a legacy protocol or technology that supports only IPv4. Newer technologies and protocols almost always support both IP versions.

The following are some of the main factors applicable to IPv4 and IPv6 support in an IoT solution:

- **Application Protocol:** IoT devices implementing Ethernet or Wi-Fi interfaces can communicate over both IPv4 and IPv6, but the application protocol may dictate the choice of the IP version. For IoT devices with application protocols defined by the IETF, such as HTTP/HTTPS, CoAP, MQTT, and XMPP, both IP versions are supported. The selection of the IP version is only dependent on the implementation.
- **Cellular Provider and Technology:** IoT devices with cellular modems are dependent on the generation of the cellular technology as well as the data services offered by the provider. For the first three generations of data services—GPRS, Edge, and 3G—IPv4 is the base protocol version. Consequently, if IPv6 is used with these generations, it must be tunneled over IPv4. On 4G/LTE networks, data services can use IPv4 or IPv6 as a base protocol, depending on the provider.
- **Serial Communications:** Many legacy devices in certain industries, such as manufacturing and utilities, communicate through serial lines. Data is transferred using either proprietary or standards-based protocols, such as DNP3, Modbus, or IEC 60870-5-101. In the past, communicating this serial data over anysort of distance could be handled by an analog modem connection. However, as service provider support for analog line services has declined, the solution for communicating with these legacy devices has been to use local connections. To make this work, connect the serial port of the legacy device to a nearby serial port on a piece of communications equipment, typically a router. This local router then forwards the serial traffic over IP to the central server for processing. Encapsulation of serial protocols over IP leverages mechanisms such as raw socket TCP or UDP. While raw socket sessions can run over both IPv4 and IPv6, current implementations are mostly available for IPv4 only.
- **IPv6 Adaptation Layer:** IPv6-only adaptation layers for some physical and data link layers for recently standardized IoT protocols support only IPv6. While themost commonphysical and data link layers (Ethernet, Wi-Fi, and so on) stipulate adaptation layers for both versions, newer technologies, such as IEEE 802.15.4 (Wireless Personal Area Network), IEEE 1901.2, and ITU G.9903 (Narrowband Power Line Communications) only have an IPv6 adaptation layer specified. This means that any device implementing a technology that requires an IPv6 adaptation layer must communicate over an IPv6-only subnetwork. This is reinforced by the IETF routing protocol for LLNs, RPL, which is IPv6 only.

3.3 Optimizing IP for IoT

While the Internet Protocol is key for a successful Internet of Things, constrained nodes and constrained networks mandate optimization at various layers and on multiple protocols of the IP architecture. The following sections introduce some of these optimizations already available from the market or under development by the IETF. Figure 3.1 highlights the TCP/IP layers where optimization is applied.

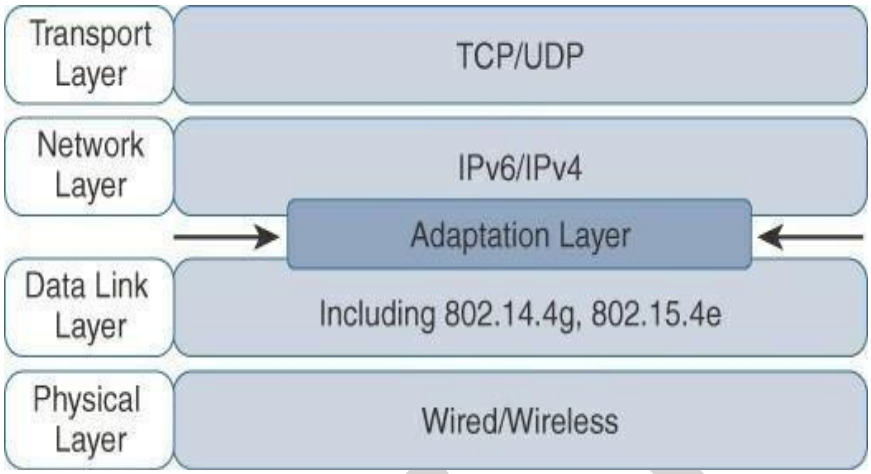


Figure 3.1 *Optimizing IP for IoT Using an Adaptation Layer*

3.3.1 From 6LoWPAN to 6Lo

In the IP architecture, the transport of IP packets over any given Layer 1 (PHY) and Layer 2 (MAC) protocol must be defined and documented. The model for packaging IP into lower-layer protocols is often referred to as an *adaptation layer*.

Unless the technology is proprietary, IP adaptation layers are typically defined by an IETF working group and released as a Request for Comments (RFC). An RFC is a publication from the IETF that officially documents Internet standards, specifications, protocols, procedures, and events. For example, RFC 864 describes how an IPv4 packet gets encapsulated over an Ethernet frame, and RFC 2464 describes how the same function is performed for an IPv6 packet.

IoT-related protocols follow a similar process. The main difference is that an adaptation layer designed for IoT may include some optimizations to deal with constrained nodes and networks. The main examples of adaptation layers optimized for constrained nodes or "things" are the ones under the 6LoWPANworking group and its successor, the 6Lo working group. The initial focus of the 6LoWPAN working group was to optimize the transmission of IPv6 packets over constrained networks such as IEEE 802.15.4. Figure 3.2 shows an example of an IoT protocol stack using the 6LoWPAN adaptation layer beside the well- known IP protocol stack for reference.

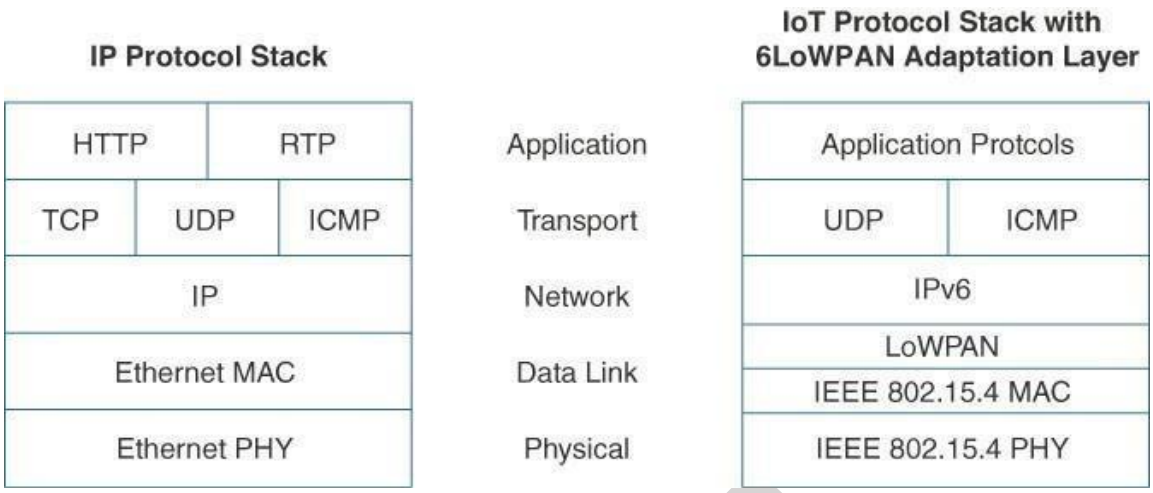


Figure 3.2 *Comparison of an IoT Protocol Stack Utilizing 6LoWPAN and an IP Protocol Stack*

The 6LoWPAN working group published several RFCs, but RFC 4994 is foundational because it defines frame headers for the capabilities of header compression, fragmentation, and mesh addressing. These headers can be stacked in the adaptation layer to keep these concepts separate while enforcing a structured method for expressing each capability. Depending on the implementation, all, none, or any combination of these capabilities and their corresponding headers can beenabled.
Figure 3.3 shows some examples of typical 6LoWPAN header stacks.

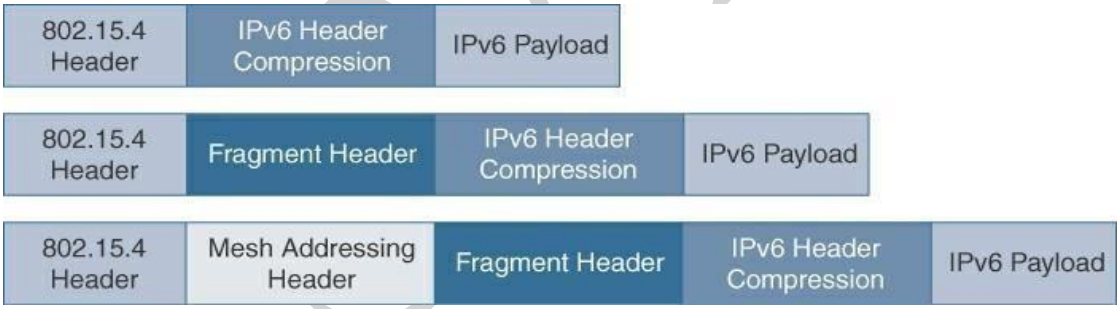


Figure 3.3 *6LoWPAN Header Stacks*

3.3.1.1 Header Compression

IPv6 header compression for 6LoWPANwas defined initially in RFC 4944 and subsequentlyupdated by RFC 6282. This capability shrinks the size of IPv6's 40-byte headers and User Datagram Protocol's (UDP's) 8-byte headers down as low as 6 bytes combined in some cases.

6LoWPAN header compression is stateless, and conceptually it is not too complicated. However, a number of factors affect the amount of compression, such as implementation of RFC 4944 versus RFC 6922, whether UDP is included, and various IPv6 addressing scenarios. It is beyond the scope of this book to cover every use case and how the header fields change for each. At a high level, 6LoWPAN works by taking advantage of shared information known by all nodes from their participation in the local network. In addition, it omits some standard header fields by assuming commonly used values. Figure 3.4 highlights an example that shows the amount of reduction that is possible with 6LoWPAN header compression.

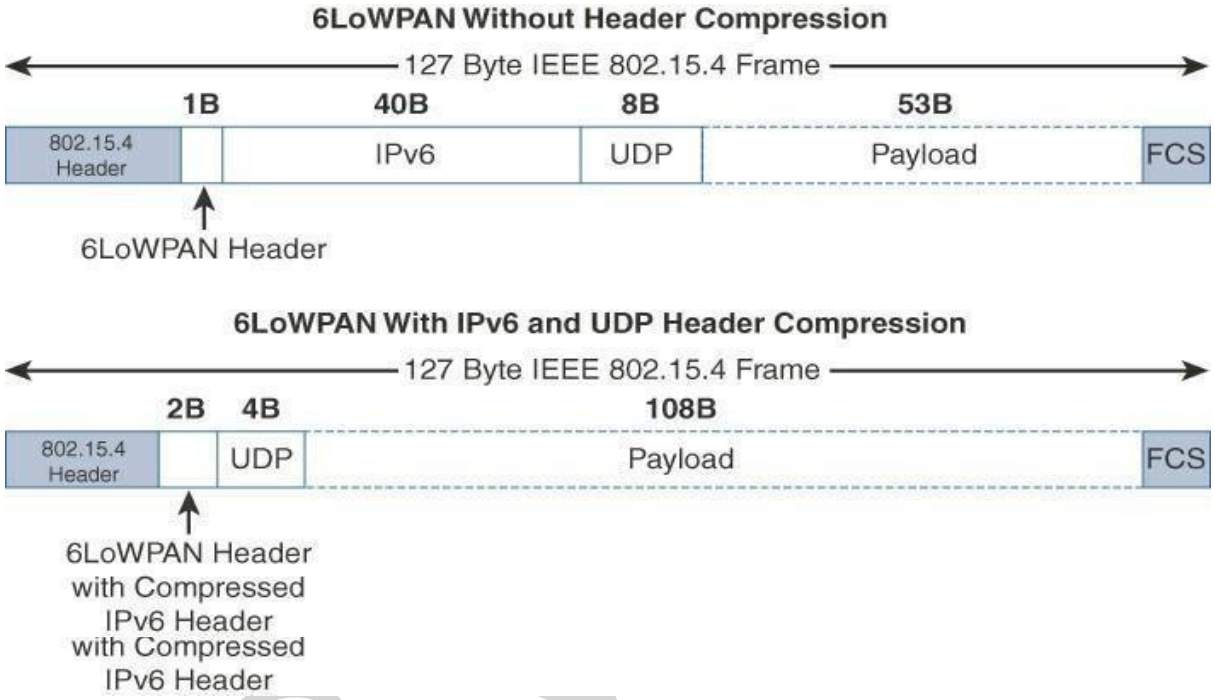


Figure 3.4 *6LoWPAN Header Compression*

At the top of Figure 3.4, a 6LoWPAN frame without any header compression enabled: The full 40-byte IPv6 header and 8-byte UDP header are visible. The 6LoWPAN header is only a single byte in this case. Notice that uncompressed IPv6 and UDP headers leave only 53 bytes of data payload out of the 127-byte maximum frame size in the case of IEEE 802.15.4.

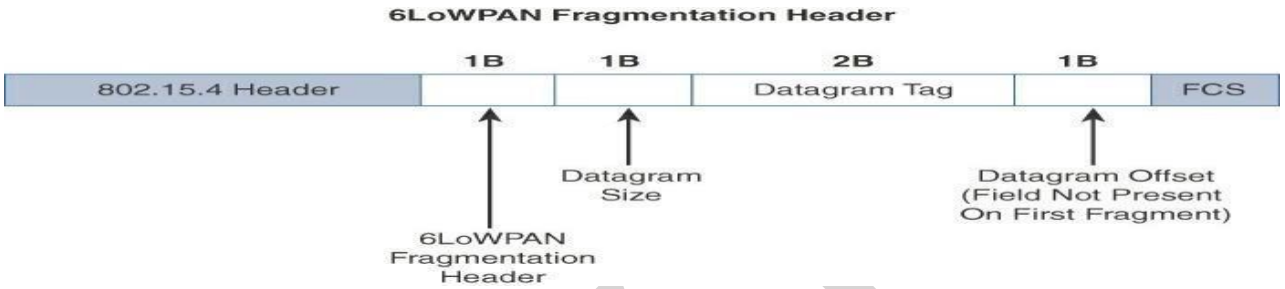
The bottom half of Figure 3.4 shows a frame where header compression has beenenabled for a best- case scenario. The 6LoWPAN header increases to 2 bytes to accommodate the compressed IPv6 header, and UDP has been reduced in half, to 4 bytes from 8. Most importantly, theheader compression has allowed thepayload to more than double, from 53 bytes to 108 bytes, which is obviously muchmore efficient. Note that the 2-byte header compression applies to intra-cell communications, while communications external to the cell may require some field of the header to not be compressed.

3.3.1.2 Fragmentation

The maximum transmission unit (MTU) for an IPv6 network must be at least 1280 bytes. The term *MTU*

defines the size of the largest protocol data unit that can be passed. For IEEE 802.15.4, 127 bytes is the MTU. Because of this is there is a problem that IPv6 with a much larger MTU, is carried inside the 802.15.4 frame with a much smaller one. To remedy this situation, large IPv6 packets must be fragmented across multiple 802.15.4 frames at Layer 2.

The fragment header utilized by 6LoWPAN is composed of three primary fields: Datagram Size, Datagram Tag, and Datagram Offset. The 1-byte Datagram Size field specifies the total size of theunfragmented payload. Datagram Tag identifies the set of fragments for a payload. Finally, the Datagram Offset field delineates how far into a payload a particular fragment occurs. Figure 3.5 provides an overview of a



6LoWPAN fragmentation header.

Figure 3.5 *6LoWPAN Fragmentation Header*

In Figure 3.5, the 6LoWPAN fragmentation header field itself uses a unique bit value to identify that the subsequent fields behind it are fragment fields as opposed to another capability, such as header compression. Also, in the first fragment, the Datagram Offset field is not present because it would simply be set to 0. This results in the first fragmentation header for an IPv6 payload being only 4 bytes long. The remainder of the fragments has a 5-byte header field so that the appropriate offset can be specified.

3.3.1.3 Mesh Addressing

The purpose of the 6LoWPANmesh addressing function is to forward packets over multiple hops. Three fields are defined for this header: Hop Limit, Source Address, and Destination Address. Analogous to the IPv6 hop limit field, the hop limit for mesh addressing also provides an upper limit on how many times the frame can be forwarded. Each hop decrements this value by 1 as it is forwarded. Once the value hits 0, it is dropped and no longer forwarded. The Source Address and Destination Address fields for mesh addressing are IEEE 802.15.4 addresses indicating the endpoints of an IP hop. Figure 3.6 details the 6LoWPAN mesh addressing header fields.

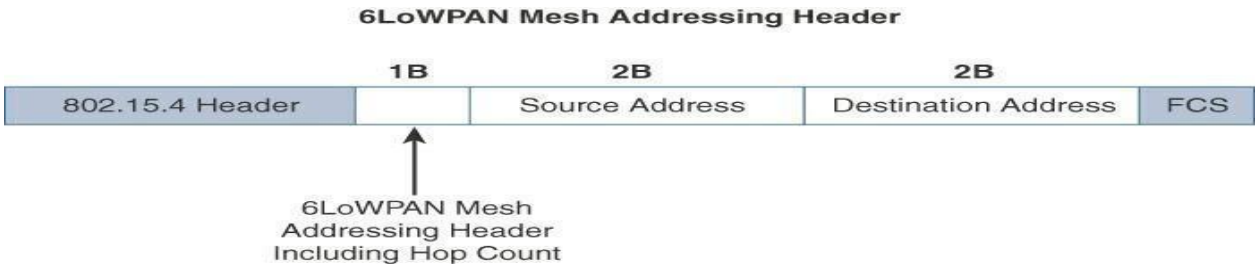


Figure 3.6 *6LoWPAN Mesh Addressing Header*

Mesh-Under Versus Mesh-Over Routing

For network technologies such as IEEE 802.15.4, IEEE 802.15.4g, and IEEE 1901.2a that support mesh topologies and operate at the physical and data link layers, two main options exist for establishing reachability and forwarding packets. With the first option, mesh-under, the routing of packets is handled at the 6LoWPAN adaptation layer. The other option, known as "**mesh-over**" or "route-over," utilizes IP routing for getting packets to their destination.

The term ***mesh-under*** is used because multiple link layer hops can be used to complete a single IP hop. Nodes have a Layer 2 forwarding table that they consult to route the packets to their final destination within the mesh. An edge gateway terminates the mesh-under domain. The edge gateway must also implement a mechanism to translate between the configured Layer 2 protocol and any IP routing mechanism implemented on other Layer 3 IP interfaces.

In mesh-over or route-over scenarios, IP Layer 33 routing is utilized for computing reachability and then getting packets forwarded to their destination, either inside or outside the mesh domain. Each full-functioning node acts as an IP router, so each link layer hop is an IP hop. When a LoWPAN has been implemented using different link layer technologies, a mesh-over routing setup is useful. While traditional IP routing protocols can be used, a specialized routing protocol for smart objects, such as RPL, is recommended.

3.3.2 6Lo Working Group

With the work of the 6LoWPAN working group completed, the 6Lo working group seeks to expand on this completed work with a focus on IPv6 connectivity over constrained-node networks. While the 6LoWPAN working group initially focused its optimizations on IEEE 802.15.4 LLNs, standardizing IPv6 over other link layer technologies is still needed.

Therefore, the charter of the 6Loworking group, now called the IPv6 over Networks of Resource- Constrained Nodes, is to facilitate the IPv6 connectivityover constrained-node networks. In particular, this working group is focused on the following:

- **IPv6-over-foo adaptation layer specifications using 6LoWPAN technologies (RFC4944, RFC6282, RFC6775) for link layer technologies:** For example, this includes:
    - IPv6 over Bluetooth Low Energy
    - Transmission of IPv6 packets over near-field communication IPv6 over 802.11ah

- Transmission of IPv6 packets over DECT Ultra Low Energy
- Transmission of IPv6 packets on WIA-PA (Wireless Networks for Industrial Automation– Process Automation)
- Transmission of IPv6 over Master Slave/Token Passing (MS/TP)

- **Information and data models such as MIB modules:** One example is RFC 7388, "Definition of Managed Objects for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)."
- **Optimizations that are applicable to more than one adaptation layer specification:** For example, this includes RFC 7400, "6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)."
- **Informational and maintenance publications needed for the IETF specifications in this area**

3.3.3 6TiSCH

Manyproprietary wireless technologies have been developed and deployed in various industry verticals over the years. However, the publication of the IEEE 802.15.4 physical and data link layer specifications, followed by IEEE 802.15.4e amendments, has opened thepath to standardized, deterministic communications over wireless networks. IEEE 802.15.4e, Time-Slotted Channel Hopping (TSCH), is an add-on to the Media Access Control (MAC) portion of the IEEE 802.15.4 standard, with direct inheritance from other standards, such as WirelessHART and ISA100.11a.

Devices implementing IEEE 802.15.4e TSCH communicate by following a Time Division Multiple Access (TDMA) schedule. An allocation of a unit of bandwidth or time slot is scheduled between neighbor nodes. This allows the programming of predictable transmissions and enables deterministic, industrial-type applications. In comparison, other 802.15.4 implementations do not allocate slices of bandwidth, so communication, especially during times of contention, maybe delayed or lost because it is always best effort.

To standardize IPv6 over the TSCH mode of IEEE 802.15.4e (known as 6TiSCH), the IETF formed the 6TiSCH working group. This working group works on the architecture, information model, and minimal 6TiSCH configuration, leveraging and enhancing work done by the 6LoWPAN working group, RoLL working group, and CoRE working group. The RoLL working group focuses on Layer 3 routing for constrained networks.

An important element specified by the6TiSCH working group is 6top, a sublayer that glues together the MAC layer and 6LoWPANadaptation layer. This sublayer provides commands to the upper network layers, such as RPL. In return, these commands enable functionalities including network layer routing decisions, configuration, and control procedures for 6TiSCH schedule management.

The IEEE 802.15.4e standard defines a time slot structure, but it does not mandate a scheduling algorithm for how the time slots are utilized. This is left to higher-level protocols like 6TiSCH. Scheduling is critical because it can affect throughput, latency, and power consumption. Figure 3.7 shows where 6top resides in relation to IEEE 802.15.4e, 6LoWPAN HC, and IPv6. HC (Header Compression)

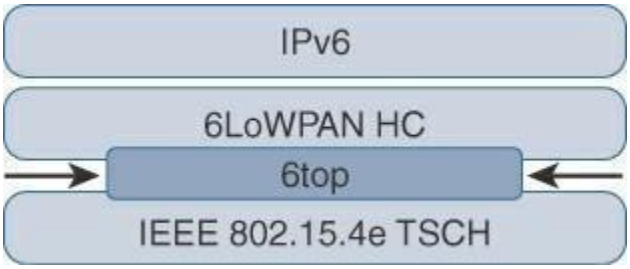


Figure 5-7 *Location of 6TiSCH's 6top Sublayer*

Schedules in 6TiSCH are broken down into cells. A cell is simply a single element in the TSCH schedule that can be allocated for unidirectional or bidirectional communications between specific nodes. Nodes only transmit when the schedule dictates that their cell is open for communication. The 6TiSCH architecture defines four schedule managementmechanisms:

- **Static scheduling:** All nodes in theconstrained network share a fixed schedule. Cells are shared, and nodes contend for slot access in a slotted aloha manner. Slotted aloha is a basic protocol for sending data using time slot boundaries when communicating over a shared medium. Static scheduling is a simple scheduling mechanism that can be used upon initial implementation or as a fallback in the case of network malfunction. The drawback with static scheduling is that nodes mayexpect a packet at any cell in the schedule. Therefore, energy is wasted idly listening across all cells.
- **Neighbor-to-neighbor scheduling:** A schedule is established that correlates with the observed number of transmissions between nodes. Cells in this schedule can be added or deleted as traffic requirements and bandwidth needs change.
- **Remote monitoring and scheduling management:** Time slots and other resource allocation are handled by a management entity that can be multiple hops away. The scheduling mechanism leverages 6top and even CoAP in some scenarios. This scheduling mechanism provides quite a bit of flexibility and control in allocating cells for communication betweennodes.
- **Hop-by-hop scheduling:** A node reserves a path to a destination node multiple hops away by requesting the allocation of cells in a schedule at each intermediate node hop in thepath. The protocol that is used by a node to trigger this scheduling mechanism is not defined at this point.

In addition to schedule management functions, the 6TiSCH architecture also defines three different forwarding models. Forwarding is the operation performed on each packet by a node that allows it to be delivered to a next hop or an upper-layer protocol. The forwarding decision is based on a preexisting state that was learned from a routing computation. There are three 6TiSCH forwarding models:

- **Track Forwarding (TF):** This is the simplest and fastest forwarding model. A "track" in this modelis a unidirectional path between a source and a destination. This track is constructed by pairing bundles of receive cells in a schedule with a bundle of receive cells set to transmit. So, a frame received within a particular cell or cell bundle is switched to another cell or cell bundle. This forwarding occurs regardless of the network layer protocol.

- **Fragment forwarding (FF):** This model takes advantage of 6LoWPAN fragmentation to build a Layer 2 forwarding table. IPv6 packets can get fragmented at the 6LoWPAN sublayer to handle the differences between IEEE 802.15.4 payload size and IPv6 MTU. Additional headers for RPL source route information can further contribute to the need for fragmentation. However, with FF, a mechanism is defined where the first fragment is routed based on the IPv6 header present. The 6LoWPANsublayer learns the next-hop selection of this first fragment, which is then applied to all subsequent fragments of that packet. Otherwise, IPv6 packets undergo hop-by-hop reassembly. This increases latency and can be power- and CPU-intensive for a constrained node.
- **IPv6 Forwarding(6F):** This model forwards traffic based on its IPv6 routing table. Flows of packets should be prioritized by traditional QoS (quality of service) and RED (random early detection) operations. QoS is a classification scheme for flows based on their priority, and RED is a common congestion avoidance mechanism.

For many IoT wireless networks, it is not necessary to be able to control the latency and throughput for sensor data. However, when some sort of determinism is needed, 6TiSCH provides an open, IPv6-based standard solution for ensuring predictable communications over wireless sensor networks. However, its adoption by the industry is still an ongoing effort.

3.3.4 RPL

The IETF chartered the RoLL (Routing over Low-Power and Lossy Networks) working group to evaluate all Layer 3 IP routing protocols and determine the needs and requirements for developing a routing solution for IP smart objects. After study of various use cases and a survey of existing protocols, the consensus was that a new routing protocol should be developed for use by IP smart objects, given the characteristics and requirements of constrained networks. This new distance-vector routing protocol was named the **IPv6 Routing Protocol for Low Power and LossyNetworks (RPL).**The RPL specification was published as RFC 6550 by the RoLL working group. In an RPL network, each node acts as a router and becomes part of a mesh network. Routing is performed at the IP layer. Each node examines every received IPv6 packet and determines the next- hop destination based on the information contained in the IPv6 header. No information fromthe MAC-layer header is needed to perform next-hop determination. To cope with the constraints of computing and memory that are common characteristics of constrained nodes, the protocol defines two modes:

- **Storing mode:** All nodes contain the full routing table of the RPL domain. Every node knows how to directly reach every other node.
- **Non-storing mode:** Only the border router(s) of the RPL domain contain(s) the full routing table. All other nodes in the domain only maintain their list of parents and use this as a list of default routes toward the border router. This abbreviated routing table saves memory space and CPU. When communicating in non-storing mode, a node always forwards its packets to the border router, which knows how to ultimately reach the finaldestination.

RPLis based on the concept of a directed acyclic graph (DAG). A DAG is a directed graph where no cycles exist. This means that from any vertex or point in the graph, it cannot follow an edge or a line back to this same point. All of the edges are arranged in paths oriented toward and terminating at one or more root nodes. Figure 3.8 shows a basic DAG.

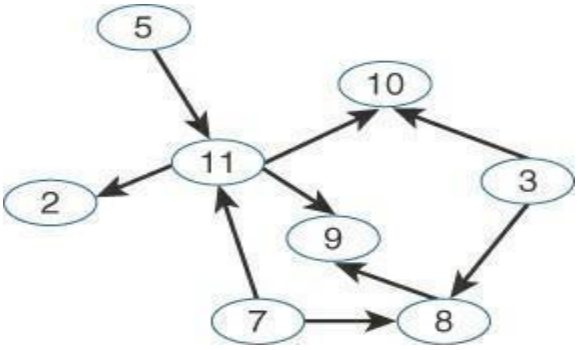


Figure 3.8 *Example of a Directed Acyclic Graph (DAG)*

A basic RPL process involves building a destination-oriented directed acyclic graph (DODAG). A DODAG is a DAG rooted to one destination. In RPL, this destination occurs at a border router known as the DODAG root. Figure 3.9 compares a DAG and a DODAG. From figure a DAG has multiple roots, whereas the DODAG has just one.

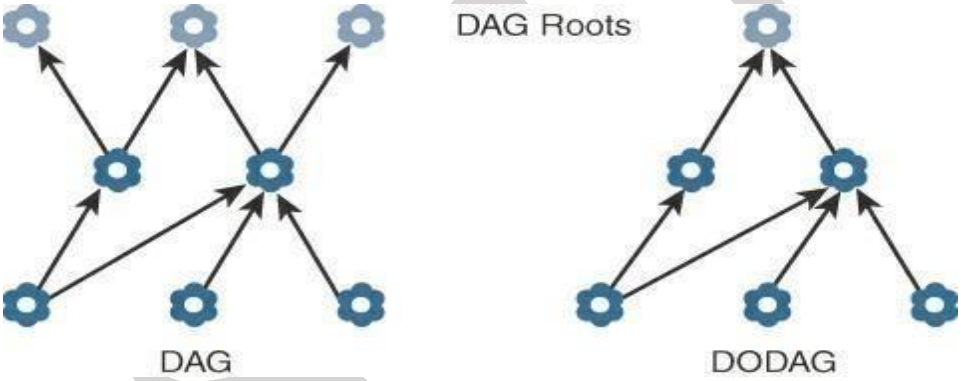


Figure 5-9 *DAG and DODAG Comparison*

- In a DODAG, each node maintains up to three parents that provide a path to the root. Typically, one of these parents is thepreferred parent, which means it is thepreferred next hop for upward routes toward the root.
- The routing graph created by theset of DODAG parents across all nodes defines the full set of upward routes. RPL protocol implementation should ensure that routes are loop free by disallowing nodes from selected DODAG parents that are positioned further away from the border router.
- Upward routes in RPL are discovered and configured using DAG Information Object (DIO) messages. Nodes listen to DIOs to handle changes in the topology that can affect routing. The information in DIO messages determines parents and the best path to the DODAG root.
- Nodes establish downward routes by advertising their parent set toward the DODAG root using a Destination Advertisement Object (DAO) message. DAO messages allow nodes to inform their parents

of their presence and reachability to descendants.

- In the case of the non-storing mode of RPL, nodes sending DAO messages report their parent sets directly to the DODAG root (border router), and only the root stores the routing information. The root uses the information to then determine source routes needed for delivering IPv6 datagrams to individual nodes downstream in the mesh.
- For storing mode, each node keeps track of the routing information that is advertised in the DAO messages. While this is more power- and CPU-intensive for each node, the benefit is that packets can take shorter paths between destinations in the mesh. The nodes can make their own routing decisions; in non-storing mode, on the other hand, all packets must go up to the root to get a route for moving downstream.
- RPL messages, such as DIO and DAO, run on top of IPv6. These messages exchange and advertise downstream and upstream routing information between a border router and the nodes under it. As illustrated in Figure 3.10, DAO and DIO messages move both up and down the DODAG, depending on the exact message type.

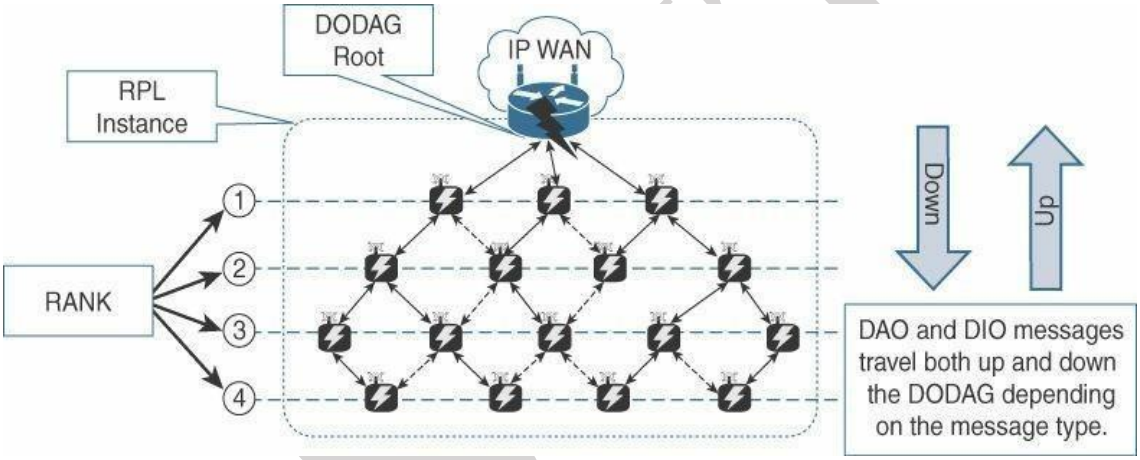


Figure 3.10 *RPL Overview*

Objective Function (OF)

An objective function(OF) defines how metrics are used to select routes and establish a node's rank. Standards such as RFC 6552 and 6719 have been published to document OFs specific to certain use cases and node types.

For example, nodes implementing an OF based on RFC 6719's Minimum Expected Number of Transmissions (METX) advertise the METX among their parents in DIO messages. Whenever a node establishes its rank, it simply sets the rank to the current minimum METX among its parents.

Rank

The rank is a rough approximation of how "close" a node is to the root and helps avoid routing loops and the count-to-infinity problem. Nodes can only increase their rank when receiving a DIO message with a larger version number. However, nodes maydecrease their rank whenever they have established lower-cost routes. While the rank and routing metrics are closely related, the rank differs from routing metrics in that it is used

as a constraint to prevent routing loops.

RPL Headers

Specific network layer headers are defined for datagrams being forwarded within an RPL domain. One of the headers is standardized in RFC 6553, "The Routing Protocol for Low-Power and Lossy Networks (RPL) Option for Carrying RPL Information in Data-Plane Datagrams," and the other is discussed in RFC 6554, "An IPv6 Routing Header for Source Routes with the Routing Protocol for Low-Power and Lossy Networks (RPL)."

RFC 6553 defines a new IPv6 option, known as the RPL option. The RPL option is carried in the IPv6 Hop-by-Hop header. The purpose of this header is to leverage data-plane packets for loop detection in a RPL instance. As discussed earlier, DODAGs only have single paths and should be loop free.

RFC 6554 specifies the Source Routing Header (SRH) for use between RPL routers. Aborder router or DODAG root inserts the SRH when specifying a source route to deliver datagrams to nodes downstream in the mesh network.

Metrics

RPL defines a large and flexible set of new metrics and constraints for routing in RFC 6551. Developed to support powered and battery-powered nodes, RPL offers a far more complete set than any other routing protocol. Some of the RPL routing metrics and constraints defined in RFC 6551 include the following:

- **Expected Transmission Count(ETX):** Assigns a discrete value to the number of transmissions a node expects to make to deliver apacket.
- **Hop Count:** Tracks the number of nodes traversed in a path. Typically, a path with a lower hop count is chosen over a path with a higher hop count.
- **Latency:** Varies depending on power conservation. Paths with a lower latency are preferred.
- **Link Quality Level:** Measures the reliability of a link by taking into account packet error rates caused by factors such as signal attenuation and interference.
- **Link Color:** Allows manual influence of routing by administratively setting values to make a link more or less desirable. These values can be either statically or dynamically adjusted for specific traffic types.
- **Node State and Attribute:** Identifies nodes that function as traffic aggregators and nodes that are being impacted byhigh workloads. High workloads could be indicative of nodes that have incurred high CPU or low memory states. Naturally, nodes that are aggregators are preferred over nodes experiencing high workloads.
- **Node Energy:** Avoids nodes with low power, so a battery-powered node that is running out of energy can be avoided and the life of that node and the network can be prolonged.
- **Throughput:** Provides the amount of throughput for a node link. Often, nodes conserving power use lower throughput. This metric allows the prioritization of paths with higher throughput.

In addition to the metrics and constraints listed in RFC 6551, others can also be implemented. For example, let's look at a scenario in which two constraints are used as a filter for pruning links that do not satisfy the specified conditions. One of the constraints is ETX. ETX, which is described in RFC 6551. The other constraint, Relative Signal Strength Indicator (RSSI), specifies the power present in a received radio signal. Signals with low strength are generally less reliable and more susceptible to interference, resulting in packet loss.

In this scenario, a DODAG root and nodes form an IEEE 802.15.4 mesh. When a node finds a potential parent, it enters the neighbor into its routing table. However, it does not yet use the new neighbor for routing. Instead, the node mustfirst establish that thelink quality to its neighbor is sufficient for forwarding datagrams. The node determines whether the link quality to a potential parent is sufficient by looking at its programmed constraints. In this example, the configured constraints are ETX and RSSI. If the RSSI in both directions exceeds a threshold and the ETX falls below a threshold, then the node confirms that the link quality to the potential parent is sufficient.

Once a node has determined that the link quality to a potential parent is sufficient, it adds the appropriate default route entry to its forwarding table. Maintaining RSSI and ETX for neighboring nodes is done at the link layer and stored in the link layer neighbor table. The results from all link layer unicast traffic are fed into the RSSI and ETX computation for neighboring devices. If the link quality is not sufficient, then the link is not addedto the forwarding table and is therefore not used for routing packets.Authentication and Encryption on Constrained Nodes

IoT security is a complex topic that often spawns discussions and debates across the industry. So it is worth mentioning here the IETF workinggroups that are focused on their security: ACE and DICE.

ACE

Much like the RoLL working group, the Authentication and **Authorization for Constrained Environments (ACE)** working group is tasked with evaluating the applicability of existing authentication and authorization protocols and documenting their suitability for certain constrained- environment use cases. Once thecandidate solutions are validated, the ACE working group will focus its work on CoAP with the Datagram Transport Layer Security (DTLS) protocol. The ACE working group mayinvestigate other security protocols later, with a particular focus on adapting whatever solution is chosen to HTTP and TLS.

The ACE working group expects to produce a standardized solution for authentication and authorization that enables authorized access (Get, Put, Post, Delete) to resources identified by a URI and hosted on a resource server in constrained environments. An unconstrained authorization server performs mediation of the access. Aligned with the initial focus, access to resources at a resource server by a client device occurs using CoAP and is protected byDTLS.

DICE

New generations of constrained nodes implementing an IP stack over constrained access networks are expected to run an optimized IP protocol stack. For example, when implementing UDP at the transport layer, the IETF Constrained Application Protocol (CoAP) should be used at the application layer.

In constrained environments secured by DTLS, CoAP can be used to control resources on a device.

The **DTLS in Constrained Environments (DICE)** working group focuses on implementing the DTLS

transport layer security protocol in these environments. The first task of the DICE working group is to define an optimized DTLS profile for constrained nodes. In addition, the DICE working group is considering the applicability of the DTLS record layer to secure multicast messages and investigating how the DTLS handshake in constrained environments can get optimized.

3.4 Profiles and Compliances

Leveraging the Internet Protocol suite for smart objects involves a collection of protocols and options that must work incoordination with lower and upper layers.
Therefore, profile definitions, certifications, and promotion by alliances can help implementers develop solutions that guarantee interoperability and/or interchangeability of devices.

This section introduces some of the main industry organizations working on profile definitions and certifications for IoT constrained nodes and networks. Therearevarious documents and promotions from these organizations in the IoT space, so it is worth being familiar with themand their goals.

- **Internet Protocol for Smart Objects (IPSO) Alliance**

  Established in 2008, the Internet Protocol for Smart Objects (IPSO) Alliance has had its objective evolve over years. The alliance initially focused on promoting IP as the premier solution for smart objects communications. Today, it is more focused on how to use IP,with the IPSO Alliance organizing interoperability tests between alliance members to validate that IP for smart objects can work together and properly implement industry standards. The IPSO Alliance does not define technologies, as that is the role of the IETF and other standard organizations, but it documents the use of IP-based technologies for various IoT use cases and participates in educating the industry. As the IPSO Alliance declares in its value and mission statement, it wants to ensure that "engineers and product builders will have access to the necessary tools for 'how to build the IoTRIGHT.'"

- **Wi-SUN Alliance**

  The Wi-SUN Alliance is an example of efforts from the industry to define a communication profile that applies to specific physical and data link layer protocols. Currently, Wi-SUN's main focus is on the IEEE 802.15.4g protocol and its support for multiservice and secure IPv6 communications with applications running over the UDP transport layer. The utilities industry is the main area of focus for the Wi-SUN Alliance. The Wi-SUN field area network (FAN) profile enables smart utility networks to provide resilient, secure, and cost-effective connectivity with extremely good coverage in a range of topographic environments, from dense urban neighborhoods to ruralareas.

- **Thread**

  A group of companies involved with smart object solutions for consumers created the Thread Group. This group has defined an IPv6-based wireless profile that provides the best way to connect more than 250 devices into a low-power, wireless mesh network. The wireless technology used by Thread is IEEE 802.15.4, which is different from Wi-SUN's IEEE 802.15.4g

- **IPv6 Ready Logo**

Initially, the IPv6 Forum ensured the promotion of IPv6 around the world. Once IPv6 implementations became widely available, the need for interoperability and certification led to the creation of the IPv6 Ready Logo program. The IPv6 Ready Logo program has established conformance and interoperability testing programs with the intent of increasing user confidence when implementing IPv6. The IPv6 Core and specific IPv6 components, such as DHCP, IPsec, and customer edge router certifications, are in place. These certifications have industry-wide recognition, and many products are already certified. An IPv6 certification effort specific to IoT is currently under definition for the program.

3.5 Application Protocols for IoT

Application protocols that are sufficient for generic nodes and traditional networks often are not well suited for constrained nodes and networks. So here the focus is on how higher-layer IoT protocols are transported with following sections:

i. The Transport Layer: IP-based networks use either TCP or UDP. However, the constrained nature of IoT networks requires a closer look at the use of these traditional transport mechanisms.

ii. IoT Application Transport Methods: This section explores the various types of IoT application data and the ways this data can be carried across a network.

As in traditional networks, TCP or UDP are utilized in most cases when transporting IoT application data. With the lower-layer IoT protocols, there are typically multiple options and solutions presented for transporting IoT application data. This is because IoT is still developing and maturing and has to account for the transport of not only new application protocols and technologies but legacy ones as well.

3.6 The Transport Layer

This section reviews theselection of a protocol for the transport layer as supported by the TCP/IP architecture in the context of IoT networks. With the TCP/IP protocol, two main protocols are specified for the transport layer:

i. Transmission Control Protocol (TCP): This connection-oriented protocol requires a session to get established between the source and destination before exchanging data. It can be viewed equivalent to a traditional telephone conversation, in which two phones must be connected and the communication link established before the parties can talk.

i. User Datagram Protocol (UDP): With this connectionless protocol, data can be quickly sent between source and destination—but with no guarantee of delivery. This is analogous to the traditional mail delivery system, in which a letter is mailed to a destination. Confirmation of thereception of this letter does not happen until another letter is sent in response.

With the predominance of human interactions over the Internet, TCP is the main protocol used at the transport layer. This is largely due to its inherent characteristics, such as its ability to transport large volumes of data into smaller sets of packets. In addition, it ensures reassembly in a correct sequence, flow control and

window adjustment, and retransmission of lost packets. These benefits occur with the cost of overhead per packet and per session, potentially impacting overall packet per second performances and latency.

In contrast, UDP is most often used in the context of network services, such as Domain Name System (DNS), Network Time Protocol (NTP), Simple Network Management Protocol (SNMP), and Dynamic Host Control Protocol (DHCP), or for real-time data traffic, including voice and video over IP. In these cases, performance and latency are more important than packet retransmissions because re-sending a lost voice or video packet does not add value. When the reception of packets mustbe guaranteed error free, the application layer protocol takes care of that function.

When considering the choice of a transport layer by a given IoT application layer protocol, it is recommended to evaluate the impact of this choice on both the lower and upper layers of the stack. For example, most of the industrial application layer protocols, are implemented over TCP, while their specifications may offer support for both transport models. The reason for this is that often these industrial application layer protocols are older and were deployed when data link layers were often unreliable and called for error protection.

While the use of TCP may not strain generic compute platforms and high-data-rate networks, it can be challenging and is often overkill on constrained IoT devices and networks. This is particularly true when an IoT device needs to send only a few bytes of data per transaction. When using TCP, each packet needs to add a minimum of 20 bytes of TCP overhead, while UDP adds only 8 bytes. TCP also requires the establishment and potential maintenance of an open logical channel.

This may explain why a new IoT application protocol, such as Constrained Application Protocol (CoAP), almost always uses UDPand why implementations of industrial application layer protocols maycall for the optimization and adoption of the UDP transport layer if run over LLNs. For example, the Device Language Message Specification/Companion Specification for Energy Metering (DLMS/COSEM) application layer protocol, a popular protocol for reading smart meters in the utilities space, is the standard in Europe. Adjustments or optimizations to this protocol should be made depending on the IoT transport protocols that are present in the lower layers.

When transferring large amounts of DLMS/COSEM data, cellular links are preferred to optimize each open association. Smaller amounts of data can be handled efficiently over LLNs. Because packet loss ratios are generally higher on LLNs than on cellular networks, keeping the data transmission amounts small over LLNs limits the retransmission of large numbers of bytes.Multicast requirements are also impacted by the protocol selected for the transport layer. With multicast, a single message can be sent to multiple IoT devices. This is useful in the IoT context for upgrading the firmware of many IoT devices at once. Also, keep in mind that multicast utilizes UDP exclusively.

3.7 IoT Application Transport Methods

The following categories of IoT application protocols and their transport methods are explored in the following sections:

- **Application layer protocol not present:** In this case, the data payload is directly transported on top of the lower layers. No application layer protocol is used.
- **Supervisory control and data acquisition (SCADA):** SCADA is one of the most common industrial protocols in the world, but it was developed long before the days of IP, and it has been adapted for IP networks.
- **Generic web-based protocols:** Generic protocols, such as Ethernet, Wi-Fi, and 4G/LTE, are found on many consumer- and enterprise-class IoT devices that communicate over non- constrained networks.
- **IoT application layer protocols:** IoT application layer protocols are devised to run on constrained nodes with a small compute footprint and are well adapted to the network bandwidth constraints on cellular or satellite links or constrained 6LoWPAN networks. Message Queuing Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP), are two examples of IoT application layer protocols.

3.7.1 Application Layer Protocol Not Present

IETF RFC 7228 devices defined as class 0 send or receive only a few bytes of data. For myriad reasons, such as processing capability, power constraints, and cost, these devices do not implement a fully structured network protocol stack, such as IP, TCP, or UDP, or even an application layer protocol. Class 0 devices are usually simple smart objects that are severely constrained. Implementing a robust protocol stack is usually not useful and sometimes not even possible with the limited available resources.

For example, consider low-cost temperature and relative humidity(RH) sensors sending data over an LPWA LoRaWAN infrastructure. Temperature is represented as 2 bytes and RH as another 2 bytes of data. Therefore, this small data payload is directly transported on top of the LoRaWAN MAC layer, without the use of TCP/IP. Example 3-1 shows the raw data for temperature and relative humidity and how it can be decoded by the application.

Example 3-1 *Decoding Temperature and Relative Humidity Sensor Data*

Temperature data payload over the network: Tx = 0x090c Temperature conversion required by the application

T = Tx/32 - 50

T = 0x090c/32 - 50

T = 2316/32 - 50 = **22.4°**

RH data payload over the network: RHx = 0x062e RH conversion required by the application:

100RH = RHx/16-24

100RH = 0x062e/16-24 = 74.9

RH **= 74.9%**

While many constrained devices, such as sensors and actuators, have adopted deployments that have no application layer, this transportation method has not been standardized. This lack of standardization makes it difficult for generic implementations of this transport method to be successful from an interoperability perspective.

Imagine expanding Example 3-1 to different kinds of temperature sensors from different manufacturers. These sensors will report temperature data in varying formats. A temperature value will always be present in the data transmitted by each sensor, but decoding this data will be vendor specific. If same scenario is scaled across hundreds or thousands of sensors, the problem of allowing various applications to receive and interpret temperature values delivered in different formats becomes increasingly complex. The solution to this problem is to use an IoT data broker, as detailed in Figure 3.11. An IoT data broker is a piece of middleware that standardizes sensor output into a common format that can then be retrieved by authorized applications.

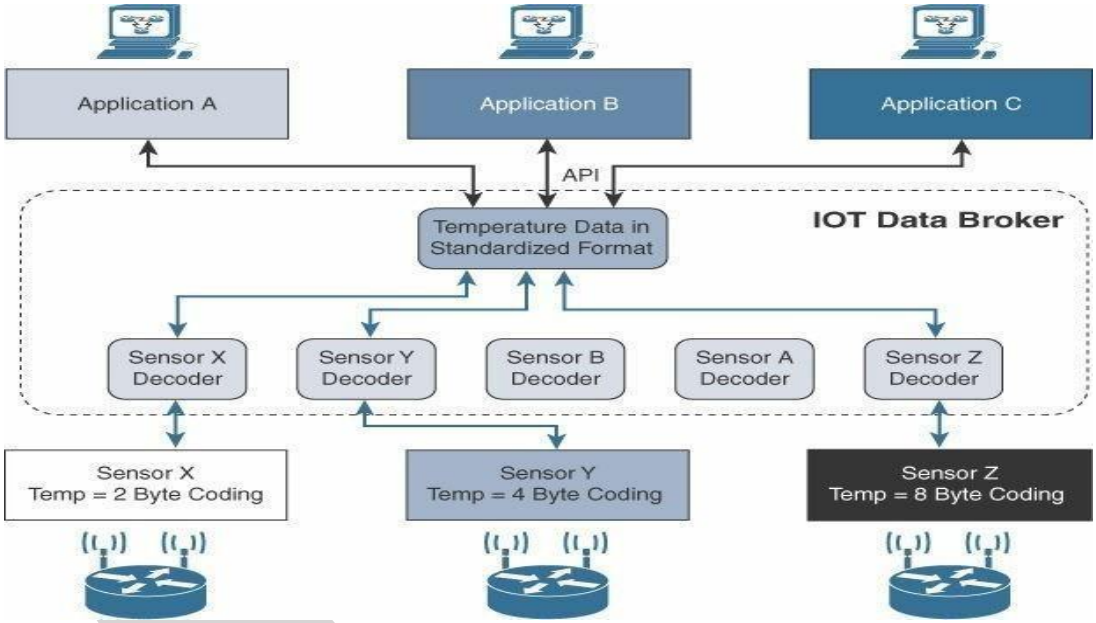


Figure 3.11 *IoT Data Broker*

In Figure 3.11, Sensors X, Y, and Z are all temperature sensors, but their output is encoded differently. The IoT data broker understands the different formats in which the temperature is encoded and is therefore able to decode this data into a common, standardized format. Applications A, B, and C in Figure 3.11 can access this temperature data without having to deal with decoding multiple temperature dataformats.

IoT data brokers are also utilized from a commercial perspective to distribute and sell IoT data to third parties. Companies can provide access to their data broker from another company's application for a fee. This makes an IoT data broker a possible revenue stream, depending on the value of the data it contains.

3.7.2 SCADA

In the world of networking technologies and protocols, IoT is relatively new. Combined with the fact that IP is the de facto standard for computer networking in general, older protocols that connected sensors and

actuators have evolved and adapted themselves to utilize IP.

A prime example of this evolution is **supervisory control and data acquisition (SCADA)**. Designed decades ago, SCADA is an automation control system that was initially implemented without IPover serial links, before being adapted to Ethernet and IPv4.

A Little Background on SCADA

For many years, vertical industries have developed communication protocols that fit their specific requirements. Many of them were defined and implemented when the most common networking technologies were serial link-based, such as RS-232 and RS-485. This led to SCADA networking protocols, which were well structured compared to the protocols described in the previous section, running directly over serial physical and data link layers.

At a high level, SCADA systems collect sensor data and telemetry from remote devices, while also providing the ability to control them. Used in today's networks, SCADA systems allow global, real- time, data-driven decisions to be made about how to improve business processes.

SCADA networks can be found across various industries, but SCADA is found mainly concentrated in theutilities and manufacturing/industrial verticals. Within these specific industries, SCADA commonlyuses certain protocols for communications between devices and applications. For example, Modbus and its variants are industrial protocols used to monitor and program remote devices via a master/slave relationship. Modbus is also found in building management, transportation, and energy applications. The DNP3 and International Electrotechnical Commission (IEC) 60870-5-101 protocols are found mainly in the utilities industry, along with DLMS/COSEM and ANSI C12 for advanced meter reading (AMR).

3.7.2.1 Adapting SCADA for IP

In the 1990s, the rapid adoption of Ethernet networks in the industrial world drove the evolution of SCADA application layer protocols. For example, the IEC adopted the Open SystemInterconnection (OSI) layer model to define its protocol framework. Other protocol user groups also slightly modified their protocols to run over an IP infrastructure. Benefits of this move to Ethernet and IP include the ability to leverage existing equipment and standards while integrating seamlessly the SCADA subnetworks to the corporateWAN infrastructures.

To further facilitate the support of legacy industrial protocols over IP networks, protocol specifications were updated and published, documenting the use of IP for each protocol. This included assigning TCP/UDP port numbers to the protocols, such as the following:

- DNP3 (adopted by IEEE 1815-2012) specifies the use of TCP or UDP on port 20000 for transporting DNP3 messages over IP.
- The Modbus messaging service utilizes TCP port 502.
- IEC 60870-5-104 is theevolution of IEC 60870-5-101 serial for running over Ethernet and IPv4 using port 2404.
- DLMS User Association specified a communication profile based on TCP/IP in the DLMS/COSEM Green Book (Edition 5 or higher), or in the IEC 62056-53 and IEC 62056-47 standards, allowing data exchange via IP and port 4059.

These legacy serial protocols have adapted and evolved to utilize IP and TCP/UDP as both networking

and transport mechanisms. This has allowed utilities and other companies to continue leveraging their investment in equipment and infrastructure, supporting these legacy protocols with modern IP networks. Let's dig deeper into how these legacy serial protocols have evolved to use IP by looking specifically at DNP3 as a representative use case.Like many of the other SCADA protocols, DNP3 is based on a master/slave relationship. The term *master* in this case refers to what is typically a powerful computer located in the control center of a utility, and a *slave* is a remote device with computing resources found in a location such as a substation. DNP3 refers to slaves specifically as *outstations*.

Outstations monitor and collect data from devices that indicate their state, such as whether a circuit breaker is on or off, and take measurements, including voltage, current, temperature, and so on. This data is then transmitted to the master when it is requested, or events and alarms can be sent in an asynchronous manner. The master also issues control commands, such as to start a motor or reset a circuit breaker, and logs the incoming data.

The IEEE 1815-2012 specification describes how the DNP3 protocol implementation must be adaptedto run either over TCP (recommended) or UDP. This specification defines connection management between the DNP3 protocol and the IP layers, as shown in Figure 3.12. Connection management links the DNP3 layers with the IP layers in addition to the configuration parameters and methods necessary for implementing the network connection. The IP layers appear transparent to the DNP3 layers as each piece of the protocol stack in one station logically communicates with therespective part in theother. This meansthat the DNP3 endpoints or devices are not aware of the underlying IP transport that is occurring.

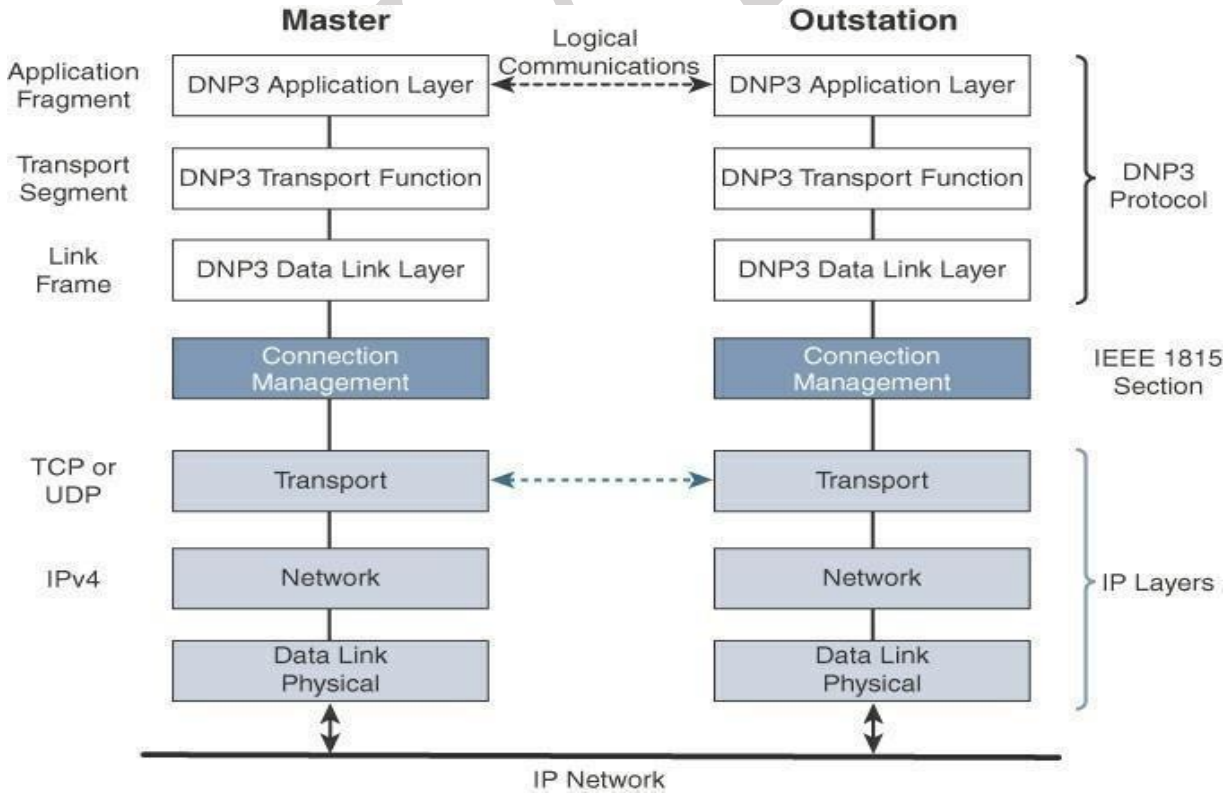


Figure 3.12 *Protocol Stack for Transporting Serial DNP3 SCADA over IP*

In Figure 3.12, the master side initiates connections by performing a TCP active open. The outstation listens for a connection request by performing a TCP passive open. *Dual endpoint* is defined as a process that can both listen for connection requests and perform an active open on the channel if required.

Master stations may parse multiple DNP3 data link layer frames from a single UDP datagram, while DNP3 data link layer frames cannot span multiple UDPdatagrams. Single or multiple connections to the master may get established while a TCP keepalive timer monitors the status of the connection. Keepalive messages are implemented as DNP3 data link layer status requests. If a response is not received to a keepalive message, the connection is deemed broken, and the appropriate action is taken.

3.7.2.2 Tunneling Legacy SCADA over IP Networks

Deployments of legacy industrial protocols, such as DNP3 and other SCADA protocols, in modern IP networks call for flexibility when integrating several generations of devices or operations that are tied to various releases and versions of application servers. Native support for IP can vary and mayrequire different solutions. Ideally, end-to-end native IP support is preferred, using a solution like IEEE 1815-2012 in the case of DNP3. Otherwise, transport of the original serial protocol over IP can be achieved either by tunneling using raw sockets over TCP or UDP or by installing an intermediate device that performs protocol translation between the serial protocol version and its IP implementation.

A raw socket connection simply denotes that the serial data is being packaged directly into a TCP or UDP transport. A socket in this instance is a standard application programming interface (API) composed of an IP address and a TCP or UDPport that is used to access network devices over an IP network. More modern industrial application servers may support this capability, while older versions typically require another device or piece of software to handle the transition from pure serial data to serial over IP using a raw socket. Figure 3.13 details raw socket scenarios for a legacy SCADA server trying to communicatewith remote serial devices.

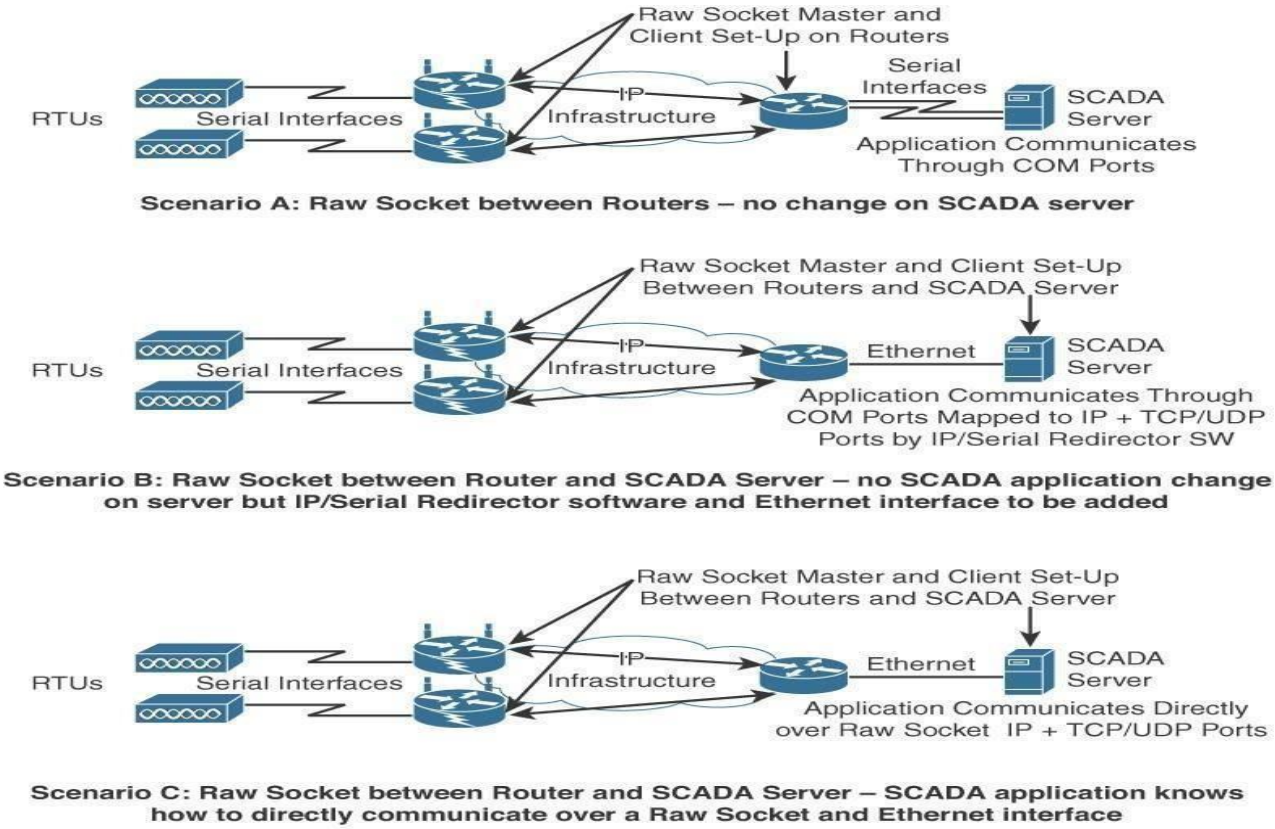


Figure 3.13 *Raw Socket TCP or UDP Scenarios for Legacy Industrial Serial Protocols*

In all the scenarios in Figure 3.13, notice that routers connect via serial interfaces to the remote terminal units (RTUs), which are often associated with SCADA networks. An RTU is a multipurpose device used to monitor and control various systems, applications, and devices managing automation. From the master/slave perspective, the RTUs are the slaves. Opposite the RTUsin each Figure 3.13 scenario is a SCADA server, or master, that varies its connection type. In reality, other legacy industrial application servers could be shown here as well.

Scenario A in Figure 3.13, both the SCADA server and the RTUs have a direct serial connection to their respective routers. The routers terminate the serial connections at both ends of the link and use raw socket encapsulation to transport the serial payload over the IP network.

Scenario B has a small change on the SCADA server side. A piece of software is installed on the SCADA server that maps the serial COM ports to IP ports. This software is commonly referred to as an IP/serial redirector. The IP/serial redirector in essence terminates the serial connection of the SCADA server and converts it to a TCP/IP port using a raw socket connection.

Scenario C in Figure 3.13, the SCADA server supports native raw socket capability. Unlike in Scenarios A and B, where a router or IP/serial redirector software has to map the SCADA server's serial ports to IP ports, in Scenario C the SCADA server has full IP support for raw socket connections.

3.7.2.3 SCADA Protocol Translation

As mentioned earlier, an alternative to a raw socket connection for transporting legacy serial data across an IP network is protocol translation. With protocol translation, the legacy serial protocol is translated to a corresponding IP version. For example, Figure 3.14 shows two serially connected DNP3 RTUs and two master applications supporting DNP3 over IP that control and pull data from the RTUs. The IoT gateway in this figure performs a protocol translation function that enables communication between the RTUs and servers, despite the fact that a serial connection is present on one side and an IP connection is used on the other.

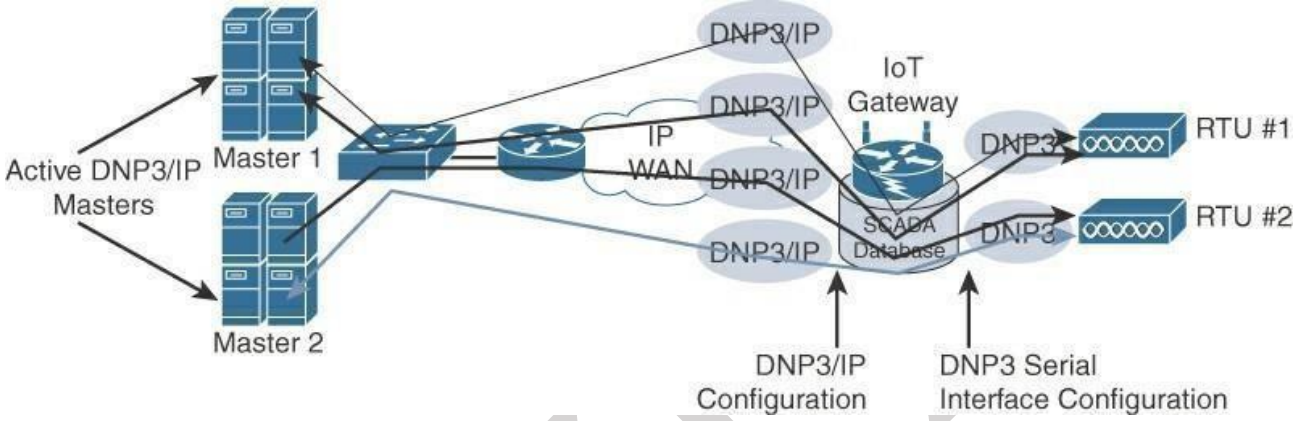


Figure 3.14 *DNP3 Protocol Translation*

By running protocol translation, the IoT gateway connected to the RTUs in Figure 3.14 is implementing a computing function close to the edge of the network. Adding computing functions close to the edge helps scale distributed intelligence in IoT networks. This can be accomplished by offering computing resources on IoT gateways or routers, as shown in this protocol translation example. Alternatively, this can also be performed directly on a node connecting multiple sensors. In either case, this is referred to as fog computing.

3.7.2.4 SCADA Transport over LLNs with MAP-T

Due to the constrained nature of LLNs, the implementation of industrial protocols should at a minimum be done over UDP. This in turn requires that both the application servers and devices support and implement UDP. While the long-term evolution of SCADA and other legacy industrial protocols is to natively support IPv6, it must be highlighted that most, if not all, of the industrial devices supporting IP today support IPv4 only. When deployed over LLN subnetworks that are IPv6 only, a transition mechanism, such as MAP-T (Mapping of Address and Port using Translation, RFC 7599), needs to be implemented. This allows the deployment to take advantage of native IPv6 transport transparently to the application and devices.

Figure 3.15 depicts a scenario in which a legacy endpoint is connected across an LLN running 6LoWPAN to

an IP-capable SCADA server. The legacy endpoint could be running various industrial and SCADA protocols, including DNP3/IP, Modbus/TCP, or IEC 60870-5-104. In this scenario, the legacy devices and the SCADA server support only IPv4 (typical in the industry today). However, IPv6 (with 6LoWPAN and RPL) is being used for connectivity to the endpoint. 6LoWPANis a standardized protocol designed for constrained networks, but it only supports IPv6. In this situation, the end devices, the endpoints, and the SCADA server support only IPv4, but the network in the middle supports onlyIPv6.

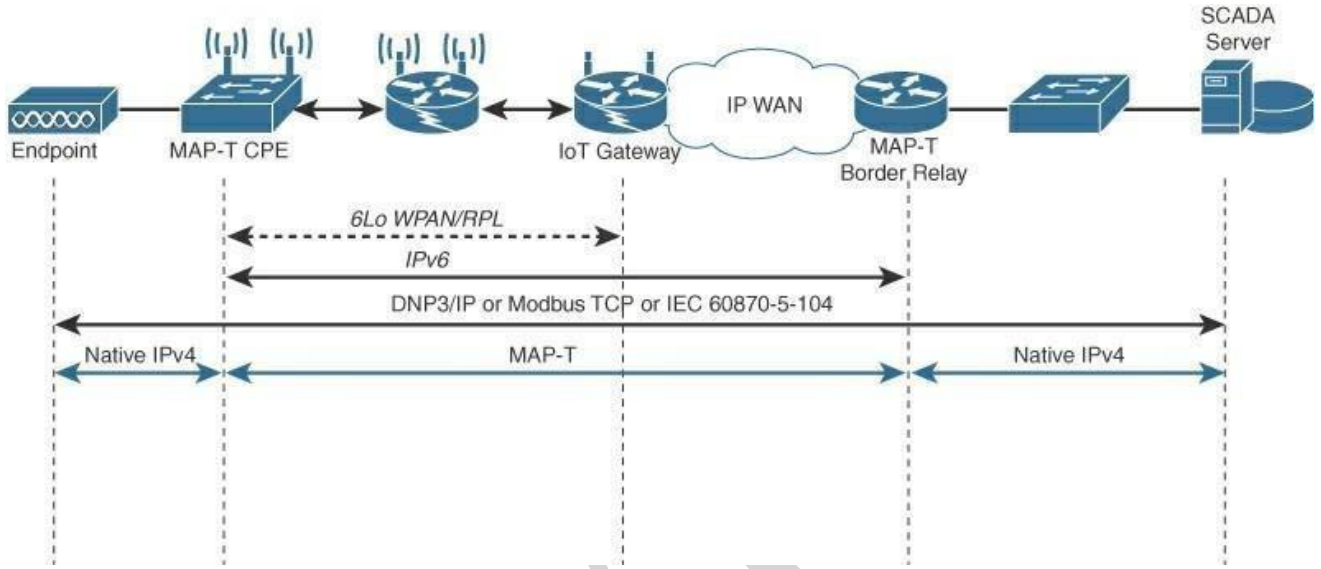


Figure 3.15 *DNP3 Protocol over 6LoWPAN Networks with MAP-T*

The solution to this problem is to use the protocol known as MAP-T, MAP-T makes the appropriate mappings between IPv4 and the IPv6 protocols. This allows legacy IPv4 traffic to be forwarded across IPv6 networks. In other words, older devices and protocols can continue running IPv4 even though the network is requiring IPv6.

In Figure 3.15 the IPv4 endpoint on the left side is connected to a Customer Premise Equipment (CPE) device. The MAP-T CPE device has an IPv6 connection to the RPL mesh. On the right side, a SCADA server with native IPv4 support connects to a MAP-T border gateway. The MAP-T CPE device and MAP-T border gateway are thus responsible for the MAP-T conversion from IPv4 to IPv6.

Legacy implementations of SCADA and other industrial protocols are still widely deployed across many industries. While legacy SCADA has evolved from older serial connections to support IP, still it can be expected to see mixed deployments for many years. To address this challenge, OT networks require mechanisms such as raw sockets and protocol translation to transport legacy versions over modern IP networks. Even when the legacy devices have IPv4 capability, the constrained portions ofthe network often require IPv6, not IPv4. In these cases, a MAP-T solution can be put in place to enable IPv4 data to be carried across an IPv6 network.

3.7.3 Generic Web-Based Protocols

Over the years, web-based protocols have become common in consumer and enterprise applications and services. Therefore, it makes sense to try to leverage these protocols when developing IoT applications, services, and devices in order to ease the integration of data and devices from prototyping to production.The level of familiarity with generic web-based protocols is high. Therefore, programmers with basic web programming skills can work on IoT applications, and this may lead to innovative ways to deliver and handle real-time IoT data. For example, an IoT device generating an event can have the result of launching a video capture, while at the same time a notification is sent to a collaboration tool, such as a Cisco Spark room. This notification allows technicians and engineers to immediately start working on this alert. In addition to a generally high level of familiarity with web-based protocols, scaling methods for web environments are also well understood—and this is crucial when developing consumer applications for potentially large numbers of IoT devices.

Once again, the definition of constrained nodes and networks must be analyzed to select the most appropriate protocol. On non- constrained networks, such as Ethernet, Wi-Fi, or 3G/4G cellular, where bandwidth is not perceived as a potential issue, data payloads based on a verbose data model representation, including XML or JavaScript Object Notation (JSON), can be transported over HTTP/HTTPS or WebSocket. This allows implementers to develop their IoT applications in contexts similar to web applications.

The HTTP/HTTPS client/server model serves as the foundation for the World Wide Web. Recent evolutions of embedded web server software with advanced features are now implemented with very little memory (in the range of tens of kilobytes in some cases). This enables the use of embedded web services software on some constrained devices.

When considering web services implementation on an IoT device, the choice between supporting the client or server side of the connection must be carefully weighed. IoT devices that only push data to an application (for example, an Ethernet- or Wi-Fi-based weather station reporting data to a weather map application or a Wi-Fi–enabled body weight scale that sends data to a health application) may need to implement web services on the client side. The HTTP client side only initiates connections and does not accept incoming ones.

On the other hand, some IoT devices, such as a video surveillance camera, may have web services implemented on the server side. However, because these devices often have limited resources, the number of incoming connections must be kept low. In addition, advanced development in data modeling should be considered as a way to shift the workload from devices to clients, including web browsers on PCs, mobile phones, tablets, and cloud applications.

Interactions between real-time communication tools powering collaborative applications, such as voice and video, instant messaging, chat rooms, and IoT devices, are also emerging. This is driving the need for simpler communication systems between people and IoT devices. One protocol that addresses this need is Extensible Messaging and Presence Protocol (XMPP).

3.7.4 IoT Application Layer Protocols

Whenconsidering constrained networks and/or a large-scale deployment of constrained nodes, verbose web-based and data model protocols, as discussed in the previous section, maybe too heavy for IoT applications. To address this problem, the IoT industry is working on new lightweight protocols that are better suited to large numbers of constrained nodes and networks. Two of the most popular protocols are CoAP and MQTT. Figure 3.16 highlights their position in a common IoT protocol stack.

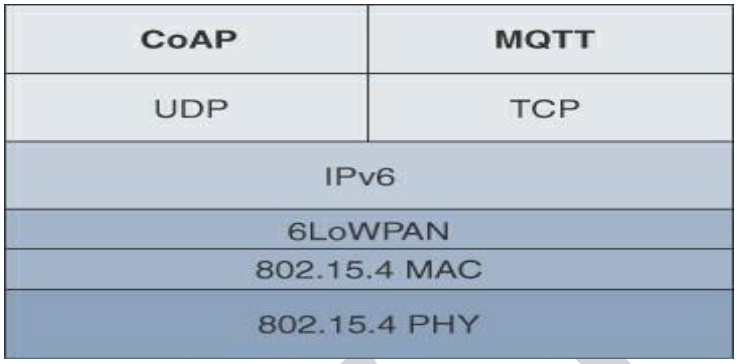


Figure 3.16 *Example of a High-Level IoT Protocol Stack for CoAP and MQTT*

In Figure 3.16, CoAP and MQTT are naturally at the top of this sample IoT stack, based on an IEEE 802.15.4 mesh network. While there are a few exceptions, like CoAP deployed over UDP and MQTT running over TCP. The following sections take a deeper look at CoAP and MQTT.

3.7.4.1 CoAP

Constrained Application Protocol (CoAP) resulted from the IETF Constrained RESTful Environments (CoRE) working group's efforts to develop a generic framework for resource-oriented applications targeting constrained nodes and networks.

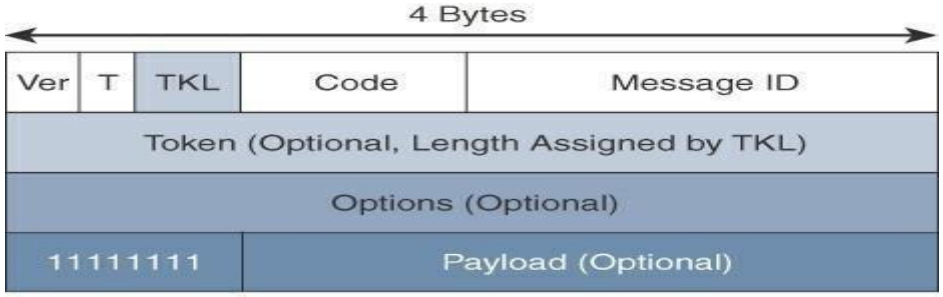
The CoAP framework defines simple and flexible ways to manipulate sensors and actuators for data or device management. The IETF CoRE working group has published multiple standards-track specifications for CoAP, including the following:

- **RFC 6690:** Constrained RESTful Environments (CoRE) Link Format
- **RFC 7252:** The Constrained Application Protocol (CoAP)
- **RFC 7641:** Observing Resources in the Constrained Application Protocol (CoAP)
- **RFC 7959:** Block-Wise Transfers in the Constrained Application Protocol (CoAP)
- **RFC 8075:** Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)

The CoAP messaging model is primarily designed to facilitate the exchange of messages over UDP between endpoints, including the secure transport protocol Datagram Transport Layer Security (DTLS). The IETF CoRE working group is studying alternate transport mechanisms, including TCP, secure TLS, and WebSocket. CoAP over Short Message Service (SMS) as defined in Open Mobile Alliance for Lightweight Machine-to-

Machine (LWM2M) for IoT device management is also being considered.

RFC 7252 provides more details on securing CoAP with DTLS. It specifies how a CoAP endpoint is provisioned with keys and a filtering list. Four security modes are defined: NoSec, PreSharedKey, RawPublicKey, and Certificate. The NoSec and RawPublicKey implementations are mandatory. From a formatting perspective, a CoAP message is composed of a short fixed-length Header field (4 bytes), a variable-



length but mandatory Token field (0–8 bytes), Options fields if necessary, and the Payload field. Figure 3.17 details the CoAP message format, which delivers low overhead while decreasing parsing complexity.

Figure 3.17 *CoAP Message Format*

From Figure 3.17, the CoAP message format is relatively simple and flexible. It allows CoAP to deliver low overhead, which is critical for constrained networks, while also being easy to parse and process for constrained devices. Table 6-1 provides an overview of the various fields of a CoAP message.

Table 6-1 *CoAP Message Fields*

CoAP Message Field	Description
Ver (Version)	Identifies the CoAP version.
T (Type)	Defines one of the following four message types: Confirmable (CON), Non-confirmable (NON), Acknowledgement (ACK), or Reset (RST). CON and ACK are highlighted in more detail in Figure 6-9.
TKL (Token Length)	Specifies the size (0–8 Bytes) of the Token field.
Code	Indicates the request method for a request message and a response code for a response message. For example, in Figure 6-9, GET is the request method, and 2.05 is the response code. For a complete list of values for this field, refer to RFC 7252.
Message ID	Detects message duplication and used to match ACK and RST message types to Con and NON message types.
Token	With a length specified by TKL, correlates requests and responses.
Options	Specifies option number, length, and option value. Capabilities provided by the Options field include specifying the target resource of a request and proxy functions.
Payload	Carries the CoAP application data. This field is optional, but when it is present, a single byte of all 1s (0xFF) precedes the payload. The purpose of this byte is to delineate the end of the Options field and the beginning of Payload.

CoAP can run over IPv4 or IPv6. However, it is recommended that the message fit within a single IP packet and UDP payload to avoid fragmentation. For IPv6, with the default MTU size being 1280 bytes and allowing for no fragmentation across nodes, themaximum CoAP message size could be up to 1152 bytes, including 1024 bytes for the payload. In the case of IPv4, as IP fragmentation may exist across the network, implementations should limit themselves to more conservative values and set the IPv4 Don't Fragment (DF)bit.

While most sensor and actuator traffic utilizes small-packet payloads, some use cases, such as firmware upgrades, require the capability to send larger payloads. CoAP doesn't rely on IP fragmentation but defines (in RFC 7959) a pair of Block options for transferring multiple blocks of information from a resource representation in multiple request/response pairs.

As illustrated in Figure 3.18, CoAP communications across an IoT infrastructure can take various paths. Connections can be between devices located on thesame or different constrained networks or between devices and generic Internet or cloud servers, all operating over IP. Proxy mechanisms are also defined, and RFC 7252 details a basic HTTP mapping for CoAP. As both HTTP and CoAP are IP-based protocols, the proxy function can be located practically anywhere in the network, not necessarily at the border between constrained and non-constrained networks.

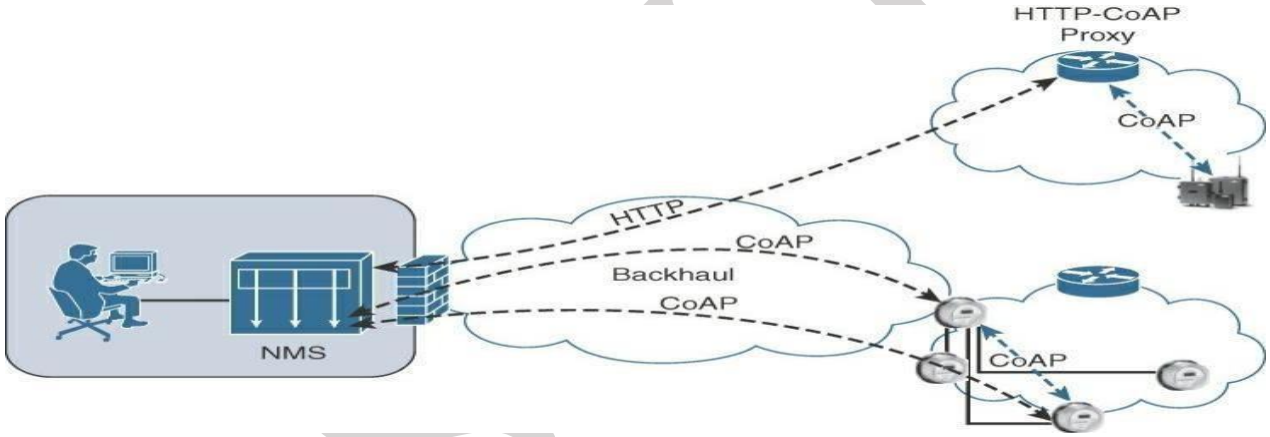


Figure 3.18 *CoAP Communications in IoT Infrastructures*

Just like HTTP, CoAP is based on the REST architecture, but with a "thing" acting as both the client and the server. Through the exchange of asynchronous messages, a client requests an action via a method code on a server resource. A uniform resource identifier (URI) localized on theserver identifies this resource. The server responds with a response code that mayinclude a resource representation. The CoAP request/response semantics include the methods GET, POST, PUT, and DELETE.

CoAP defines four types of messages: confirmable, non-confirmable, acknowledgement, and reset. Method codes and response codes included in some of these messages makethem carry requests or responses. CoAP code, method and response codes, option numbers, and content format have been assigned by IANA as Constrained RESTful Environments (CoRE) parameters.

While running over UDP, CoAP offers a reliable transmission of messages when a CoAP header is marked as "confirmable." In addition, CoAP supports basic congestion control with a default time-out, simple stop and wait retransmission with exponential back-off mechanism, and detection of duplicate messages through a

message ID. If a request or response is tagged as confirmable, the recipient must explicitly either acknowledge or reject the message, using the same message ID, as shown in Figure 3.19. If a recipient can't process a non-confirmable message, a reset message is sent.

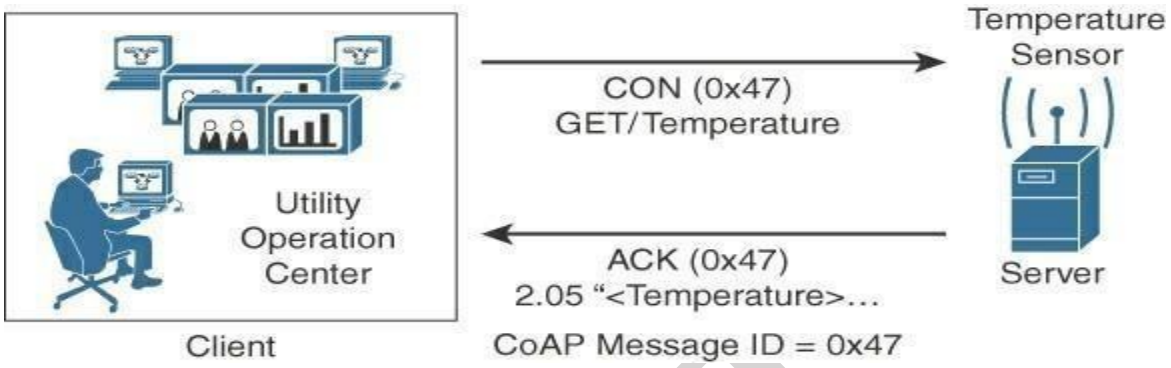


Figure 3.19 *CoAP Reliable Transmission Example*

Figure 3.19 shows a utility operations center on the left, acting as the CoAP client, with the CoAP server being a temperature sensor on the right of the figure. The communication between the client and server uses a CoAP message ID of 0x47. The CoAP Message ID ensures reliability and is used to detect duplicate messages. The client in Figure 3.19 sends a GET message to get the temperature from the sensor. Notice that the 0x47 message ID is present for this GET message and that the message is also marked with CON. A CON, or confirmable, marking in a CoAP message means the message will be retransmitted until the recipient sends an acknowledgement (or ACK) with the same message ID.

In Figure 3.19, the temperature sensor does reply with an ACK message referencing the correct message ID of 0x47. In addition, this ACK message piggybacks a successful response to the GET request itself. This is indicated by the 2.05 response code followed by the requested data. CoAP supports data requests sent to a group of devices by leveraging the use of IP Multicast. Implementing IP Multicast with CoAP requires the use of all-CoAP-node multicast addresses. For IPv4 this address is 224.0.1.187, and for IPv6 it is FF0X::FD. These multicast addresses are joined by CoAP nodes offering services to other endpoints while listening on the default CoAP port, 5683. Therefore, endpoints can find available CoAP services through multicast service discovery. A typical use case for multicasting is deploying a firmware upgrade for a group of IoT devices, such as smart meters.

With often no affordable manual configuration on the IoT endpoints, a CoAP server offering services and resources needs to be discovered by the CoAP clients. Services from a CoAP server can either be discovered by learning a URI in a namespace or through the "All CoAP nodes" multicast address. When utilizing the URI scheme for discovering services, the default port 5683 is used for non-secured CoAP, or **coap**, while port 5684 is utilized for DTLS-secured CoAP, or **coaps**. The CoAP server must be in listening state on these ports, unless a different port number is associated with the URI in a namespace. Much as with accessing web server resources, CoAP specifications provide a description of the relationships between

resources in RFC 6690, "Constrained RESTful Environments (CoRE) Link Format."

To improve the response time and reduce bandwidth consumption, CoAP supports caching capabilities based on the response code. To use a cache entry, a CoAP endpoint must validate the presented request and stored response matches, including all options (unless marked as NoCacheKey). This confirms that the stored response is fresh or valid. A wide range of CoAP implementations are available. Some are published with open source licenses, and others are part of vendor solutions

3.7.4.2 Message Queuing Telemetry Transport (MQTT)

At the end of the 1990s, engineers from IBM and Arcom (acquired in 2006 byEurotech) were looking for a reliable, lightweight, and cost-effective protocol to monitor and control a large number of sensors and their data from a central server location, as typically used by the oil and gas industries.
Their research resulted in the development and implementation of the Message Queuing Telemetry Transport (MQTT) protocol that is now standardized by the Organization for the Advancement of Structured Information Standards (OASIS).

Considering the harsh environments in the oil and gas industries, an extremely simple protocol with only a few options was designed, with considerations for constrained nodes, unreliable WAN backhaul communications, and bandwidth constraints with variable latencies. These were some of the rationales for the selection of a client/server and publish/subscribe framework based on the TCP/IP architecture, as shown in Figure 6-10.

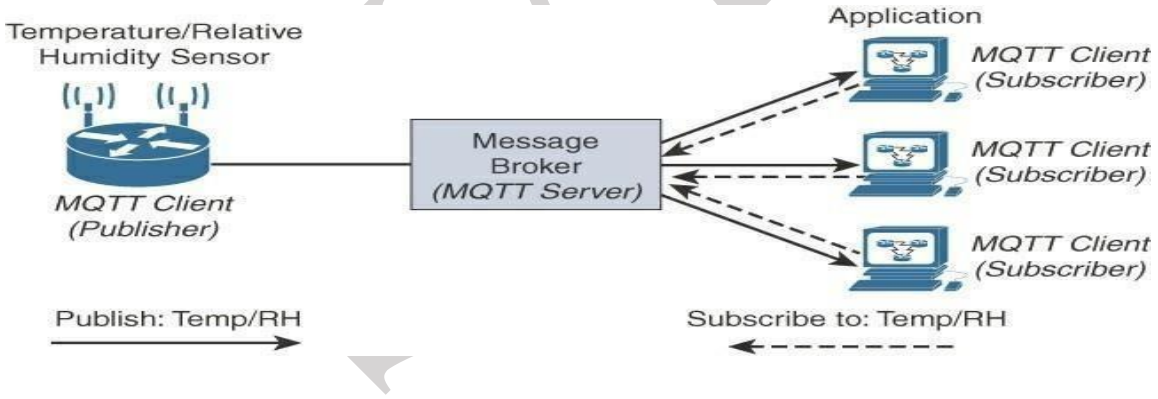


Figure 3.20 *MQTT Publish/Subscribe Framework*

An MQTT client can act as a publisher to send data (or resource information) to an MQTT server acting as an MQTT message broker. In the example illustrated in Figure 3.20, the MQTT client on the left side is a temperature (Temp) and relative humidity (RH) sensor that publishes its Temp/RH data. The MQTT server (or message broker) accepts the network connection along with application messages, such as Temp/RH data, from the publishers. It also handles the subscription and unsubscription process and pushes the application data to MQTT clients acting as subscribers.

The application on the right side of Figure 3.20 is an MQTT client that is a subscriber to the Temp/RH data being generated by the publisher or sensor on the left. This model, where subscribers express a desire to

receive information from publishers, is well known. A great example is the collaboration and social networking application Twitter.

With MQTT, clients can subscribe to all data (using a wildcard character) or specific data from the information tree of a publisher. In addition, the presence of a message broker in MQTT decouples the data transmission between clients acting as publishers and subscribers. In fact, publishers and subscribers do not even know (or need to know) about each other. A benefit of having this decoupling is that the MQTT message broker ensures that information can be buffered and cached in case of network failures. This also means that publishers and subscribers do not have to be online at the sametime.

MQTT control packets run over a TCP transport using port 1883. TCP ensures an ordered, lossless stream of bytes between the MQTT client and the MQTT server. Optionally, MQTT can be secured using TLS on port 8883, and WebSocket (defined in RFC 6455) can also be used.

MQTT is a lightweight protocol because each control packet consists of a 2-byte fixed header with optional variable header fields and optional payload. Control packet can contain a payload up to 256 MB.

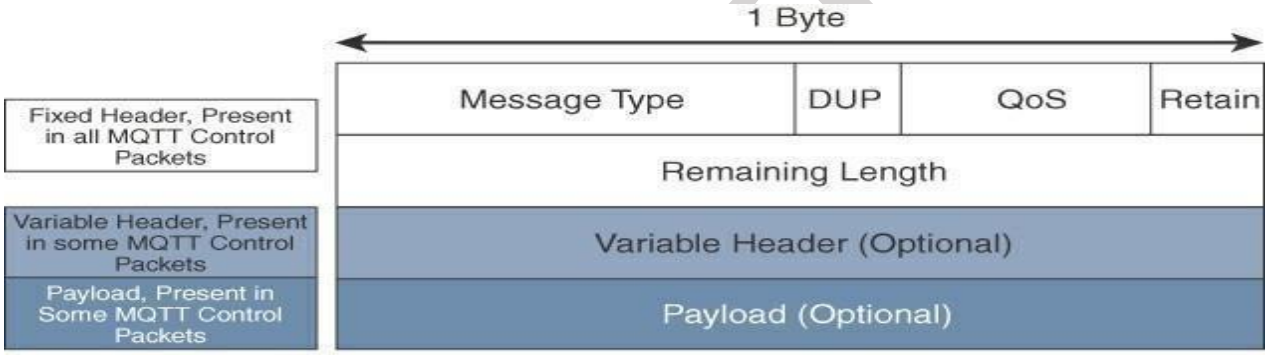


Figure 3.21 *MQTT Message Format*

Compared to the CoAP message format in Figure 3.17, MQTT contains a smaller header of 2 bytes compared to 4 bytes for CoAP. The first MQTT field in the header is Message Type, which identifies the kind of MQTT packet within a message. Fourteen different types of control packets are specified in MQTT version 3.1.1. Each of them has a unique value that is coded into the Message Type field. Note that values 0 and 15 are reserved. MQTT message types are summarized in Table3.2.

The next field in the MQTT header is DUP (Duplication Flag). This flag, when set, allows the client to notate that the packet has been sent previously, but an acknowledgement was not received. The QoS header field allows for the selection of three different QoS levels. The next field is the Retain flag. Only found in a PUBLISH message, the Retain flag notifies the server to hold onto the message data. This allows new subscribers to instantly receive the last known value without having to wait for the next update from the publisher. The last mandatory field in the MQTT message header is Remaining Length. This field specifies the number of bytes in the MQTT packet following thisfield.

Table 6-2 *MQTT Message Types*

Message Type	Value	Flow	Description
CONNECT	1	Client to server	Request to connect
CONNACK	2	Server to client	Connect acknowledgement
PUBLISH	3	Client to server Server to client	Publish message
PUBACK	4	Client to server Server to client	Publish acknowledgement
PUBREC	5	Client to server Server to client	Publish received
PUBREL	6	Client to server Server to client	Publish release
PUBCOMP	7	Client to server Server to client	Publish complete
SUBSCRIBE	8	Client to server	Subscribe request
SUBACK	9	Server to client	Subscribe acknowledgement
UNSUBSCRIBE	10	Client to server	Unsubscribe request
UNSUBACK	11	Server to client	Unsubscribe acknowledgement
PINGREQ	12	Client to server	Ping request
PINGRESP	13	Server to client	Ping response
DISCONNECT	14	Client to server	Client disconnecting

The MQTT protocol offers three levels of quality of service (QoS). QoS for MQTT is implemented when exchanging application messages with publishers or subscribers, and it is different fromthe IP QoS that most people are familiar with. The delivery protocol is symmetric. This means the client and server can each take the role of either sender or receiver. The delivery protocol is concerned solely with the delivery of an application message from a single sender to a single receiver. These are the three levels of MQTTQoS:

- **QoS 0:** This is a best-effort and unacknowledged data service referred to as “at most once” delivery. The publisher sends its message one time to a server, which transmits it once to the subscribers. No response is sent by the receiver, and no retry is performed by the sender. The message arrives at the receiver either once or not at all.
- **QoS 1:** This QoS level ensures that the message delivery between the publisher and server and then between the server and subscribers occurs at least once. In PUBLISH and PUBACK packets, a packet identifier is included in thevariable header. If the message is not acknowledged by a PUBACKpacket, it is sent again. This level guarantees “at least once” delivery.
- **QoS 2:** This is the highest QoS level, used when neither loss nor duplication of messages is acceptable. There is an increased overhead associated with this QoS level because each packet contains an optional variable header with a packet identifier. Confirming the receipt of a PUBLISH message requires a two-step acknowledgement process. The first step is done through the PUBLISH/PUBREC packet pair, and the second is achieved with the PUBREL/PUBCOMP packet pair. This level provides a “guaranteed service” known as “exactly once” delivery, with no consideration for the number of retries as long as the message is delivered once.

As mentioned earlier, the QoS process is symmetric in regard to the roles of sender and receiver, but two separate transactions exist. One transaction occurs between the publishing client and the MQTT server, and the other transaction happens between the MQTT server and the subscribing client. Figure 3.22 provides an overview of the MQTT QoS flows for the three different levels.

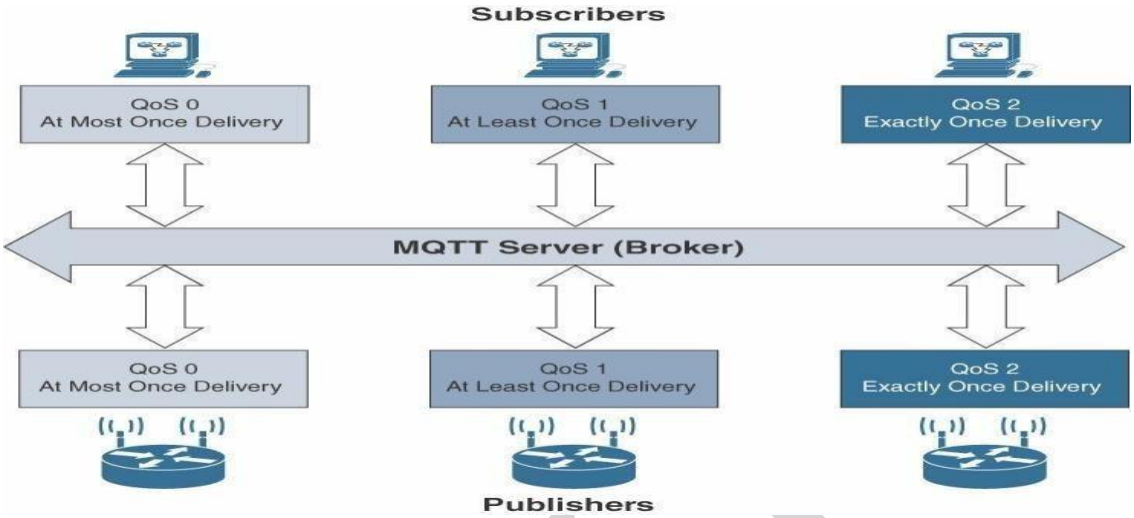


Figure 3.22 *MQTT QoS Flows*

As with CoAP, a wide range of MQTT implementations are now available. They are either published as open source licenses or integrated into vendors' solutions, such as Facebook Messenger.

Both CoAP and MQTT have been discussed in detail, there arises questions like "Which protocol is better for a given use case?" and "Which one should I used in my IoT network?" Unfortunately, the answer is not always clear, and both MQTT and CoAP have their place. Table 3-3 provides an overview of the differences between MQTT and CoAP, along with their strengths and weaknesses from an IoT perspective.

Table 3-3 *Comparison Between CoAP and MQTT*

Factor	CoAP	MQTT
Main transport protocol	UDP	TCP
Typical messaging	Request/response	Publish/subscribe
Effectiveness in LLNs	Excellent	Low/fair (Implementations pairing UDP with MQTT are better for LLNs.)
Security	DTLS	SSL/TLS
Communication model	One-to-one	many-to-many
Strengths	Lightweight and fast, with low overhead, and suitable for constrained networks; uses a RESTful model that is easy to code to; easy to parse and process for constrained devices; support for multicasting; asynchronous and synchronous messages	TCP and multiple QoS options provide robust communications; simple management and scalability using a broker architecture
Weaknesses	Not as reliable as TCP-based MQTT, so the application must ensure reliability.	Higher overhead for constrained devices and networks; TCP connections can drain low-power devices; no multicasting support