Unit 3

IP as the IoT Network Layer

In the previous chapter we have learnt about Creating IOT network and protocols used to communicate with smart objects to access and communicate with a network.

This Unit focuses on protocol stack and extend the conversation to network layer connectivity.

The Business Case for IP

Overview of IP/TCP

Suppose that you wanted to send a message to the authors of this book, but you didn't have the postal address, and you didn't have any way to look up our phone number (because in this example you don't have the Internet).

You remember that we're from the UK, and London is the biggest city in the UK. So you send a postcard to your cousin Bob, who lives there.

Your cousin sees that the postcard is for some crazy hardware and technology people. So he puts the postcard in an envelope and drops it off at the London Hackspace because the guys there probably know what to do with it.

At the Hackspace, Jonty picks up the envelope and sees that it's for some people in Liverpool. Like all good Londoners, Jonty never goes anywhere to the north of Watford, but he remembers that Manchester is in the north too. So he calls up the Manchester Digital Laboratory (MadLab), opens the envelope to read the contents, and says, "Hey, I've got this message for Adrian and Hakim in Liverpool. Can you pass it on?"

The guys at MadLab ask whether anyone knows who we are, and it turns out that Hwa Young does. So the next time she comes to Liverpool, she delivers the postcard to us.

IP

The preceding scenario describes how the Internet Protocol (IP) works. **data is sent from one machine to another in a packet, with a destination address and a source address in a standardised format** (a "protocol").

Just like the original sender of the message in the example, the sending machine doesn't always know the best route to the destination in advance.

Most of the time, the packets of data have to go through a number of intermediary machines, called **routers**, to reach their destination. The underlying networks aren't always the same: just as we used the phone, the postal service, and delivery by hand, so data packets can be sent over wired or wireless networks, through the phone system, or over satellite links.

In our example, a postcard was placed in an envelope before getting passed onwards. This happens with Internet packets, too. So, an IP packet is a block of data along with the same kind of information you would write on a physical envelope: the name and address of the server, and so on.

But if an IP packet ever gets transmitted across your local wired network via an Ethernet cable—the cable that connects your home broadband router or your office local area network (LAN) to a desktop PC—then the whole packet will get bundled up into another type of envelope, an Ethernet Frame, which adds additional information about how to complete the last few steps of its journey to your computer.

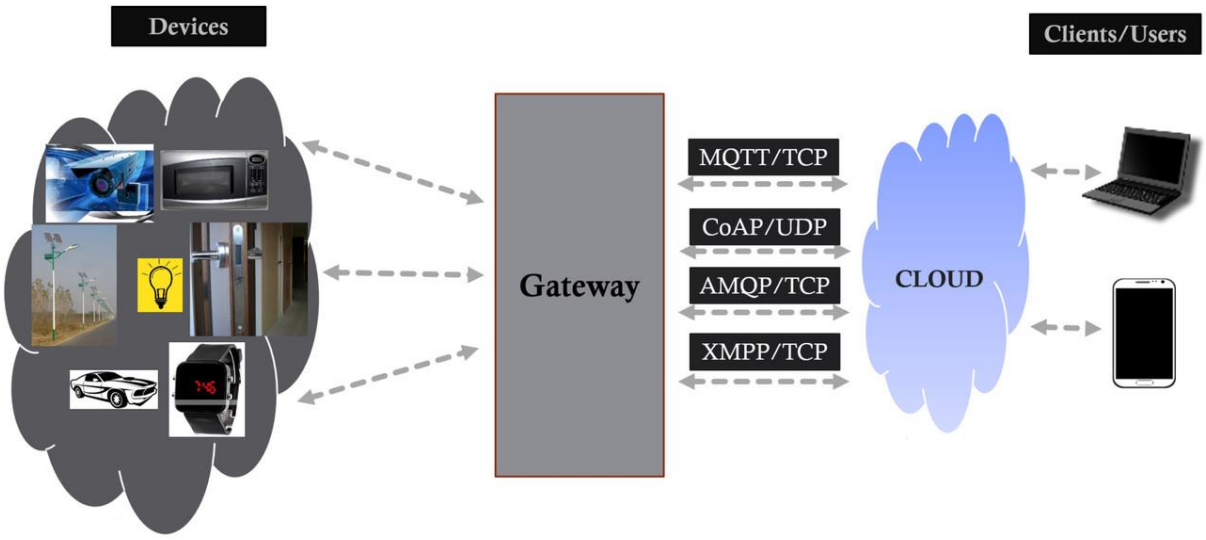
TCP

What if you wanted to send longer messages than fit on a postcard? Or wanted to make sure your messages got through?

That is basically how the **Transmission Control Protocol (TCP)** works. The simplest transport protocol on the Internet, TCP is built on top of the basic IP protocol and adds sequence numbers, acknowledgements, and retransmissions.

This means that a message sent with TCP can be arbitrarily long and give the sender some assurance that it actually arrived at the destination intact.

(Reference: Designing the Internet of Things by Adrian McEwen, Hakim cassimally Pageno:42)



The IoT devices are typically connected to the Internet via an IP (Internet Protocol) network.

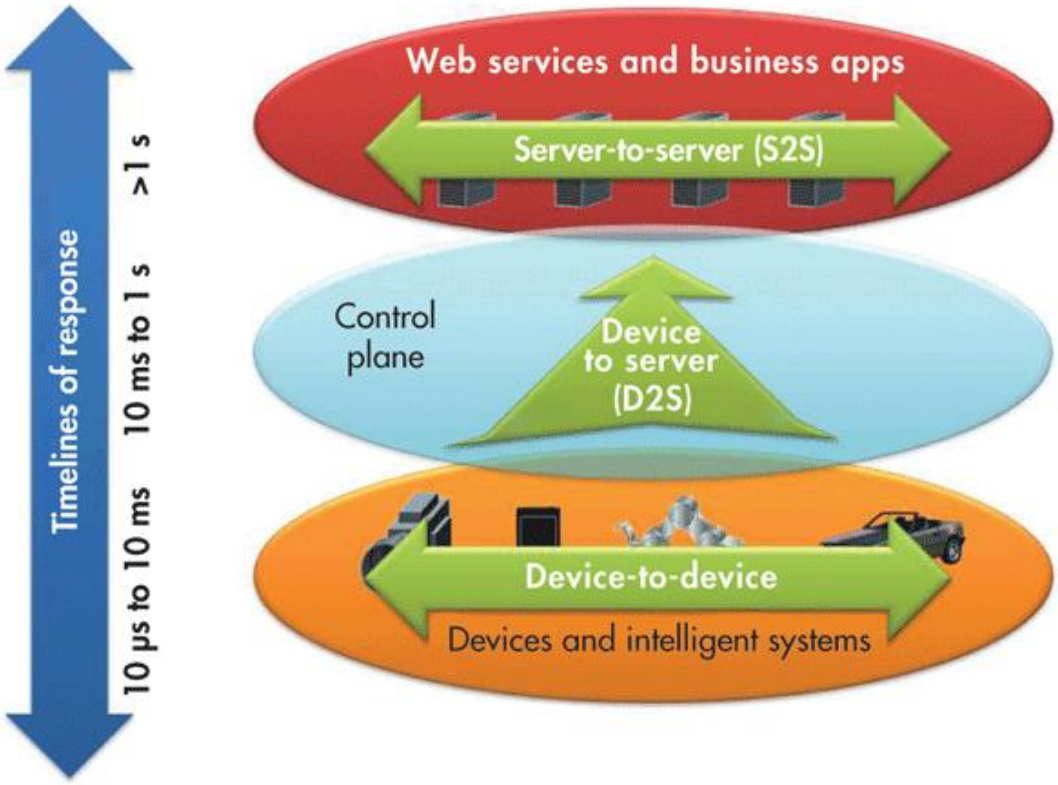
However, devices such as Bluetooth and RFID allow IoT devices to connect locally.

In these cases, there's a difference in power, range, and memory used. Connection through IP networks are comparatively complex, requires increased memory and power from the IoT devices while the range is not a problem.

On the other hand, non-IP networks demand comparatively less power and memory but have a range limitation.

The key Advantages of Internet Protocol

- 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.



Device-to-device protocols is used to transfer data from one device to another
Device to server protocols is used to transfer data from devices to cloud

Server-to-server infrastructure has to share device data (S2S), possibly providing it back to devices, to analysis programs, or to people.

This section provides a quick review of the key advantages of the IP suite for the Internet of Thing.

Open and standards-based:

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 many standards 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 and protocols.

Versatile:

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 to the 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.
In fact, IP is the most pervasive protocol when you look at what is supported across the various IoT solutions and industry verticals.

Scalable:

Adding huge numbers of "things" to private and public infrastructures may require optimizations and design rules specific to the new devices. However, you should realize that this is not very different from the recent evolution of voice and video endpoints integrated over IP. IP has proven before that scalability is one of its strengths

Manageable and highly secure:

Adopting IP network management also brings an operational business application to OT. Well-known network and security management

tools are easily leveraged with an IP network layer. However, you should be aware that despite the secure nature of IP, real challenges exist in this area. Specifically, the industry is challenged in securing constrained nodes, handling legacy OT protocols, and scaling operations.

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 know that 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.

Summary:

- The adoption of IP provides a solid foundation for the Internet of Things by allowing secured and manageable bidirectional data communication capabilities between all devices in a network.
- IP is a standards-based protocol that is ubiquitous, scalable, versatile, and stable.
- Network services such as naming, time distribution, traffic prioritization, isolation, and so on are well-known and developed techniques that can be leveraged with IP.

- From cloud, centralized, or distributed architectures, IP data flow can be developed and implemented according to business requirements.

Adoption or Adaptation of the Internet Protocol

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 IP layers.

ALG or Application Layer Gateway is a software component that manages specific application protocols such as SIP (Session Initiation Protocol) and FTP (File Transfer Protocol). An ALG acts as an intermediary between the Internet and an application server that can understand the application protocol.

Adoption involves replacing all non-IP layers with their IP layer counterparts, simplifying the deployment model and operations.

Let's look at a few examples in various industries to see how IP adaptation and adoption are currently applied 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.
- 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.

You should consider the following factors when trying to 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.

Protocol such as 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 more difficult.

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, you can potentially have more header overhead than device data—again, particularly in the case of LPWA technologies.

Consequently, you 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 may put 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, the adaptation 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.

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.

Constrained Nodes

Small **devices** with limited CPU, memory, and power resources, so- called "**constrained devices**" (often used as sensors/actuators, smart objects, or smart **devices**) can form a network, becoming "**constrained** nodes" in that network.

Depending on its functions in a network, a “thing” architecture may or may not offer similar characteristics compared to a generic PC or server in an IT environment.

Limits:

- Network protocol stack on an IOT node may be required to communicate through an unreliable. 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.
- Power consumption is a key characteristic of constrained nodes.

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, you may implement 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).

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.

Constrained Networks

In the early years of the Internet, network bandwidth capacity was restrained due to technical limitations. Connections often depended on low-speed modems for transferring data. However, these low-speed connections demonstrated that IP could run over low-bandwidth 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 utilize a star, mesh, or combined network topologies, ensuring proper operations.
- With a constrained network, in addition to limited bandwidth, it is not unusual for the packet delivery rate (PDR) to oscillate between low and high percentages.

In summary, constrained nodes and networks pose major challenges for IoT connectivity in the last mile. This in turn has led various standards organizations to work on optimizing protocols for IoT.

IP Versions

IP v4

IP stands for Internet Protocol and v4 stands for Version Four (IPv4). IPv4 was the primary version brought into action for production within the ARPANET in 1983.IP version four addresses are 32-bit integers which will be expressed in hexadecimal notation.

Example- 192.0.2.126 could be an IPv4 address.

IP v6

IP v6 was developed by Internet Engineering Task Force (IETF) to deal with the problem of IP v4 exhaustion. IP v6 is 128-bits address having an address space of 2^128, which is way bigger than IPv4. In IPv6 we use Colon-Hexa representation. There are 8 groups and each group represents 2 Bytes.

(Reference Geeks for Geeks)

In contrast to IPv4**, the IPv6 system is based on 128-bit addresses and is able to facilitate close to 340 undecillion unique IP identifiers. This is a massive increase in capability that promises to supercharge the IoT revolution**.

- 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.

- 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.

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. A legacy device refers to a computing device or equipment that is outdated, obsolete or no longer in production.

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 any sort 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, you 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 the most common physical and data link layers (Ethernet, Wi-Fi, and so on) stipulate adaptation layers for both versions, newer technologies, such as IEEE 802.15.4

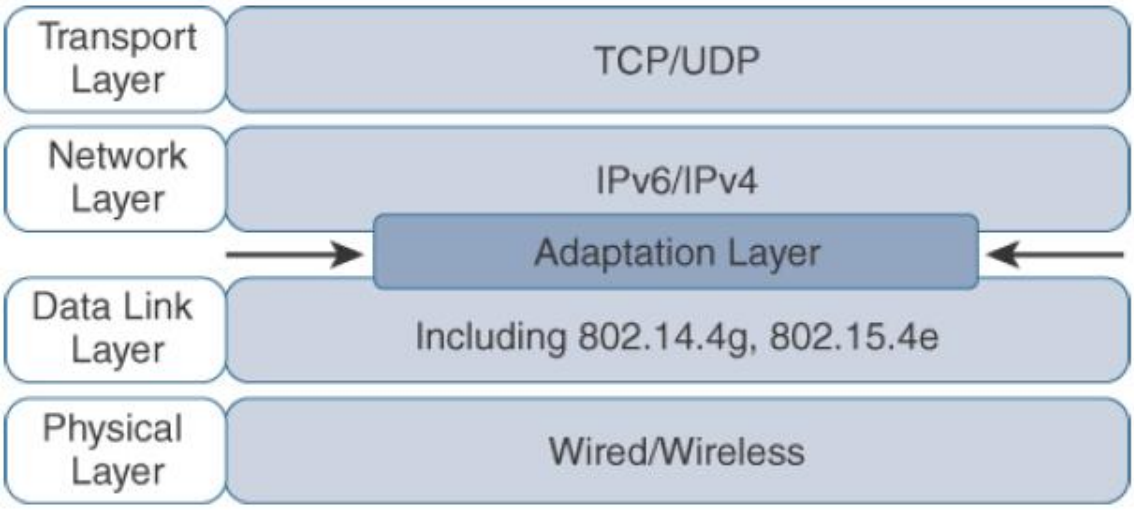
(Wireless Personal Area Network), IEEE 1901.2, and ITU G.9903 (Narrowband Power Line Communications) only have an IPv6 adaptation layer specified.

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.**

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. Below Figure highlights the TCP/IP layers where optimization is applied.



Optimizing IP for IOT using an Adaptation Layer

From 6LoWPAN to 6Lo

Leveraging 6LoWPAN, the IETF 6Lo working group has targeted adaptation of IPv6 over a new generation of communication technologies for the IoT. These comprise Bluetooth LE, ITU-T G.9959, DECT ULE, MS/TP, NFC, IEEE 1901.2, and IEEE 802.11ah.

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. 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 6LoWPAN working 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.

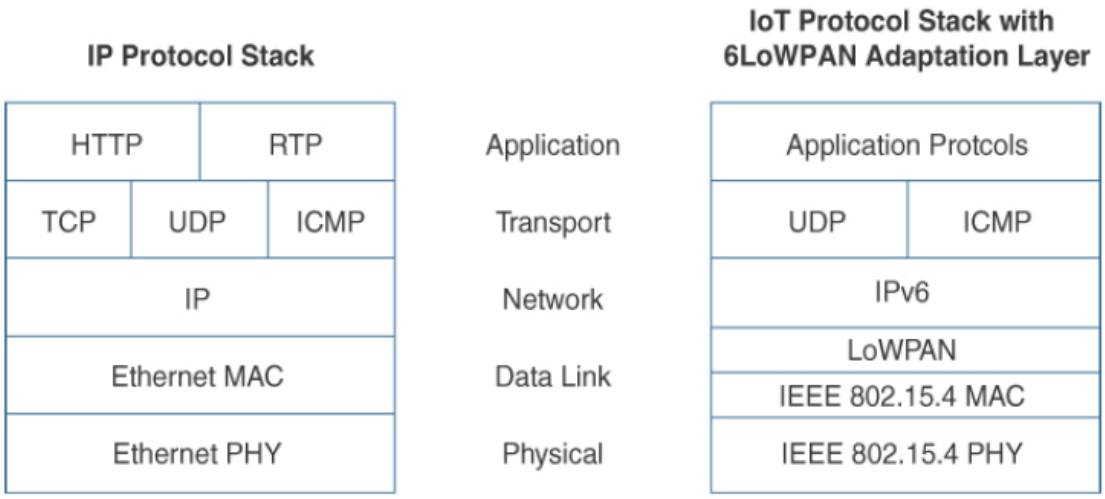


Figure shows an example of an IoT protocol stack using the 6LoWPAN adaptation layer beside the well-known IP protocol stack for reference.

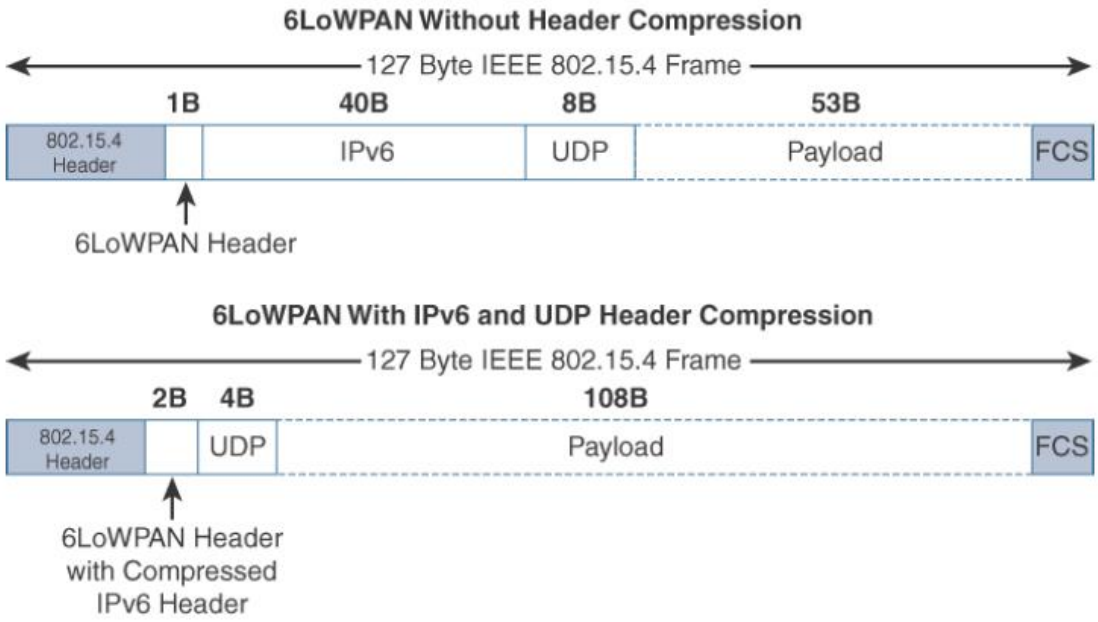
Header Compression

IPv6 header compression for 6LoWPAN was defined initially in RFC 4944 and subsequently updated 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.

Note that header compression for 6LoWPAN is only defined for an IPv6 header and not IPv4.
The 6LoWPAN protocol does not support IPv4, and, in fact, there is no standardized IPv4 adaptation layer for IEEE 802.15.4.

Features:

6LoWPAN header compression is stateless, and conceptually it is not too complicated.



At the top of Figure, you see 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 shows a frame where header compression has been enabled 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, the header compression has allowed the payload to more than double, from 53 bytes to 108 bytes, which is obviously

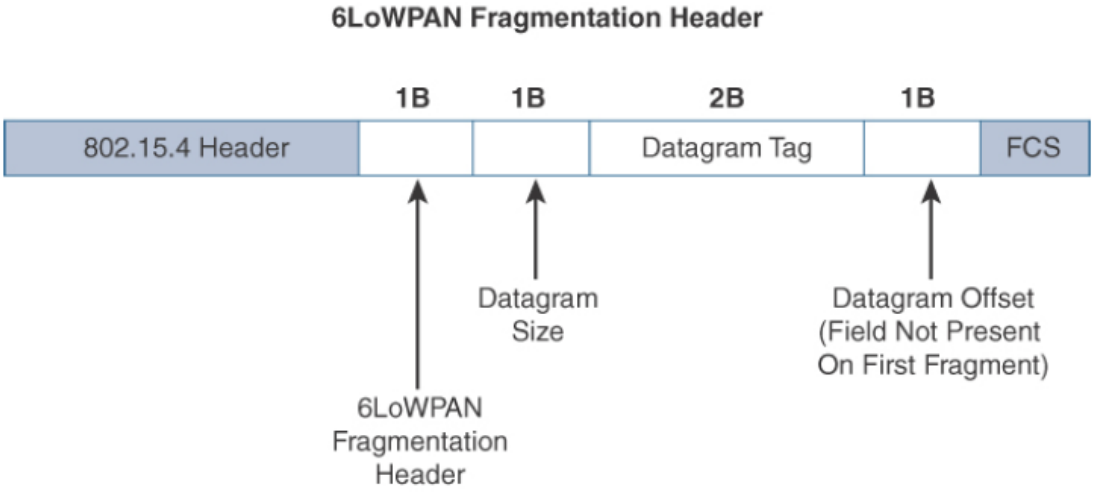
much more efficient. Note that the 2-byte header compression applies to intra-cell communications, while communications external to the cell may require some field of the header to not be compressed.

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. You can see that this is a problem because 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 the unfragmented 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.



6LoWPAN Fragmentation Header

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.

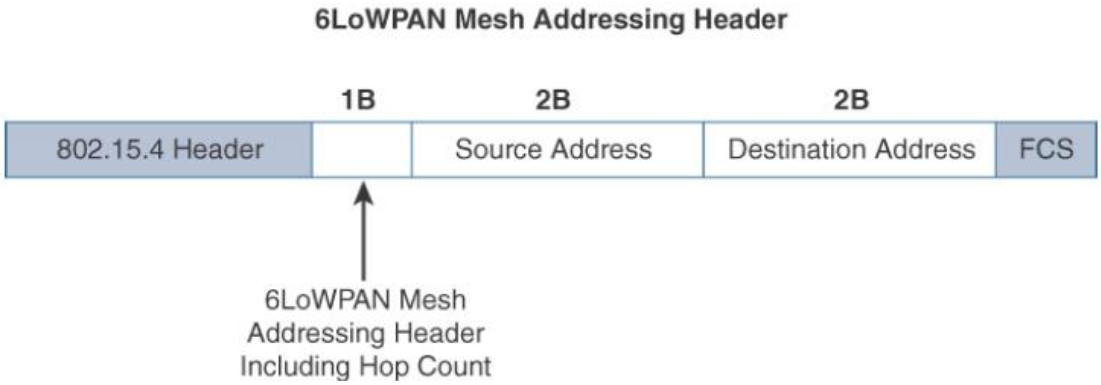
Also, in the first fragment, the Datagram Offset field is not present because it would simply be set to 0.
This results in the first fragmentation header for an IPv6 payload being only 4 bytes long.
The remainder of the fragments have a 5- byte header field so that the appropriate offset can be specified.

Mesh Addressing

The purpose of the 6LoWPAN mesh 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 5-6 details the 6LoWPAN mesh addressing header fields.



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.

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.

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**

In summary, the 6Lo working group is standardizing the 6LoWPAN adaptation layer that initially focused on the IEEE 802.15.4 Layer 2 protocol to others that are commonly found with constrained nodes. In fact, based on the work of the 6LoWPAN working group and now the 6Lo working group, the 6LoWPAN adaptation layer is becoming the de factor standard for connecting constrained nodes in IoT networks.

RPL

Routing Over Low Power and Lossy Networks (ROLL) is a working group (WG) of the IETF that handles Internet of Things (IoT) routing topics . ROLL started its work in February 2008 with the goal of developing a routing protocol suitable for low power and lossy networks (LLN). The working group was rechartered in 2016.

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.

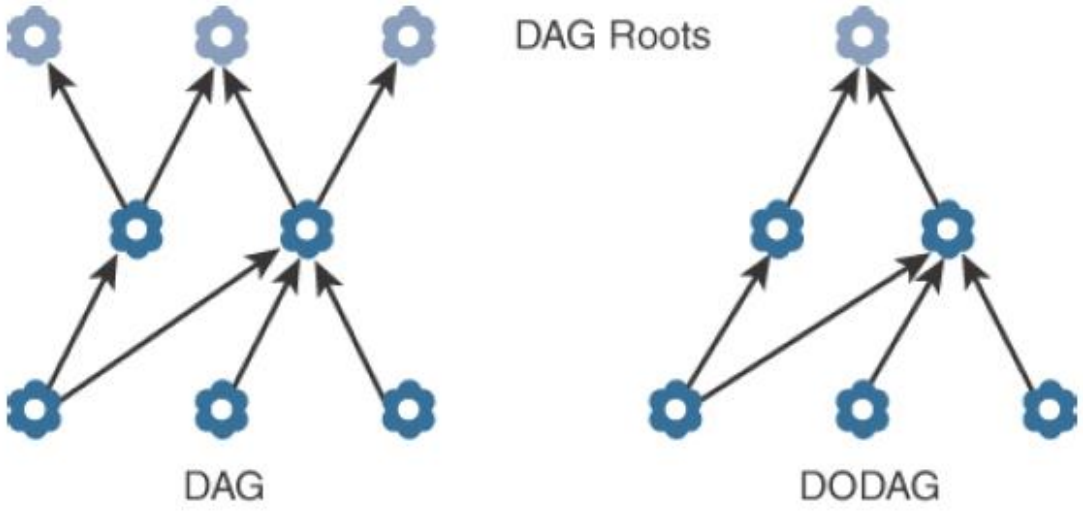
This new distance-vector routing protocol was named the IPv6 Routing Protocol for Low Power and Lossy Networks (RPL). The RPL specification was published as RFC 6550 by the RoLL working group.

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 from the MAClayer 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 final destination.

RPL is based on the concept 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.



DAG and DODAG Comparison

Figure compares DAG and DODAG

- DAG has multiple roots, whereas the DODAG has just one.

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.

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 may decrease 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. A border 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 a packet.

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 by high 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.

Example refer text book page number:266

In summary,

- RPL is a new routing protocol that enables an IPv6 standards based solution to be deployed on a large scale while being operated in a similar way to today's IP infrastructures.
- RPL was designed to meet the requirements of constrained nodes and networks, and this has led to it becoming one of the main network layer IPv6-based routing protocols in IoT sensor networks.

Authentication and Encryption on Constrained Nodes

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 the candidate solutions are validated, the ACE working group will focus its work on CoAP with the Datagram Transport Layer Security (DTLS) protocol.

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.**

Profiles and Compliances

This section introduces some of the main industry organizations working on profile definitions and certifications for IoT constrained nodes and networks. You can find various documents and promotions from these organizations in the IoT space, so it is worth being familiar with them and 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.

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.

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

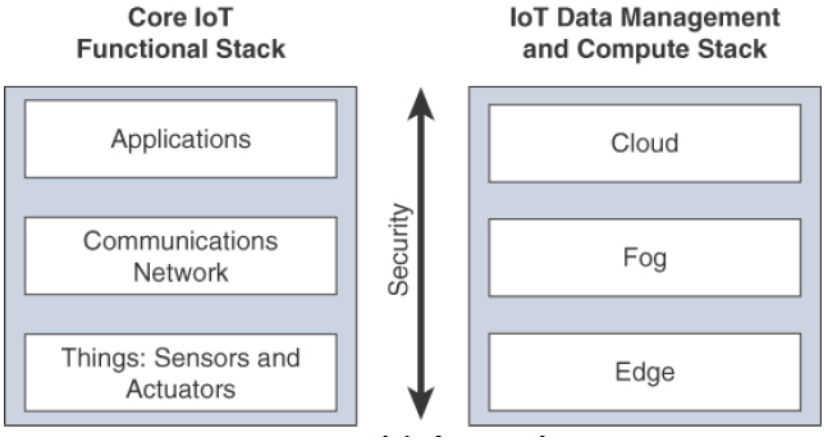
Initially, the IPv6 Forum ensured the promotion of IPv6 around the world. Once IPv6 implementations became widely available, the need for interoperability and certification led to the creation of the IPv6 Ready Logo program. The IPv6 Ready Logo program has established conformance and interoperability testing programs with the intent of increasing user confidence when implementing IPv6.

Summary

- The IP protocol suite has been deployed in private and public networks over the past three decades, interconnecting billions of IP devices and users.
- The architecture has proven to be highly flexible, and it has protected investments in many ways.
- The vast majority of the IP protocols and technologies, including addressing, address provisioning, QoS, transport, reliability, and so on, can be reused as is by IoT solutions.

Application Protocols for IoT

In the simplified architecture (unit1) we got to know top layer is application layer. IoT application protocols are dependent on the characteristics of the lower layers themselves. For example, application protocols that are sufficient for generic nodes and traditional networks often are not well suited for constrained nodes and networks.



This chapter focuses on how higher-layer IoT protocols are transported. Specifically, this chapter includes the following sections:

- **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.
- **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.

The Transport Layer

This section reviews the selection 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:

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

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 the reception 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,

- 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.

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.

Example: most of the industrial application layer protocols, as discussed later in this chapter, 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.

While the use of TCP may not strain generic compute platforms and highdata-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.

Consider the following for TCP UDP and DLMS/COSEM(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 de facto standard in Europe. :

- Select TCP for cellular networks because these networks are typically more robust and can handle the overhead. For LLNs, where both the devices and network itself are usually constrained, UDP is a better choice and often mandatory.
- DLMS/COSEM can reduce the overhead associated with session establishment by offering a "long association" over LLNs. Long association means that sessions stay up once in place because the communications overhead necessary to keep a session established is much less than is involved in opening and closing many separate sessions over the same time period. Conversely, for cellular networks, a short association better controls the costs by tearing down the open associations after transmitting.
- 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.

IoT Application Transport Methods

Because of the diverse types of IoT application protocols, there are various means for transporting these protocols across a network. Sometimes you may be dealing with legacy utility and industrial IoT protocols that have certain requirements, while other times you might need to consider the transport requirements of more modern application layer protocols.

To make these decisions easier, it makes sense to categorize the common IoT application protocols and then focus on the transport methods available for each category.

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), covered later in this chapter, are two well-known examples of IoT application layer protocols.

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.

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 shows the raw data for temperature and relative humidity and how it can be decoded by the application.

```
Temperature data payload over the network: Tx = 0x090c
Temperature conversion required by the application
T = Tx/32 - 50  to  T = 0x090c/32 - 50  to  T = 2316/32 - 50 =
22.4°
RH data payload over the network: RHx = 0x062e
RH conversion required by the application:
100RH = RHx/16-24  to 100RH = 0x062e/16-24 = 74.9  to 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.

(example refer text book page number:277)

SCADA

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 IP over serial links, before being adapted to Ethernet and IPv4.

A Little Background on SCADA

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 you find SCADA mainly concentrated in **the utilities and manufacturing/industrial verticals.** Within these specific industries, SCADA commonly uses 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.

These protocols go back decades and are **serial based.** So, transporting them over current IoT and traditional networks requires that certain accommodations be made from both protocol and implementation perspectives.

Adapting SCADA for IP

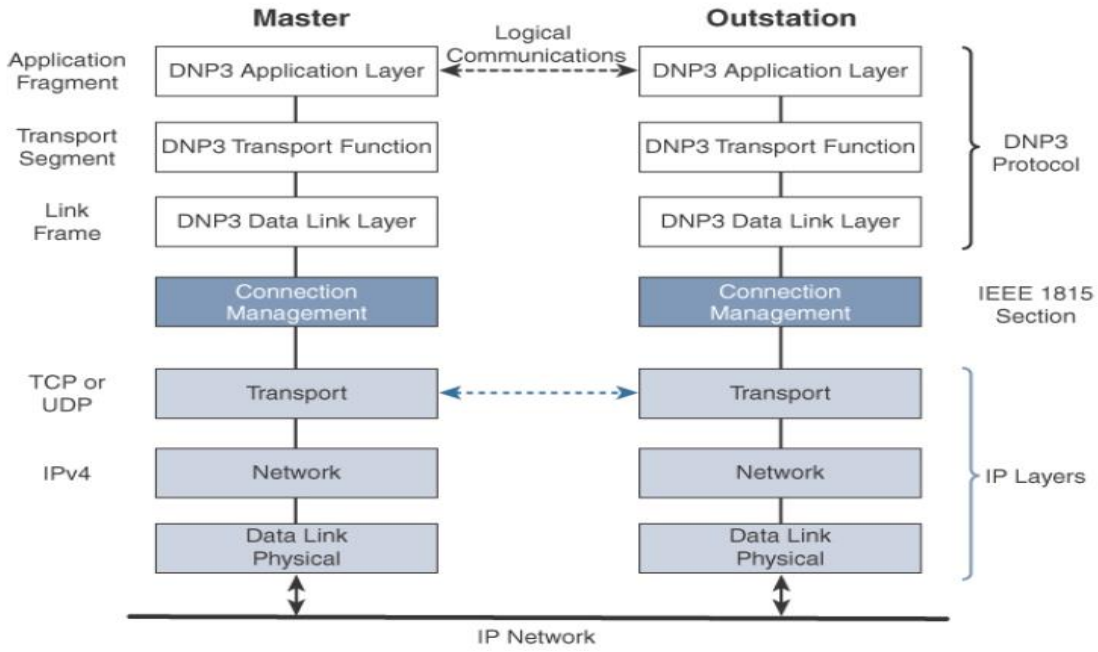
In the 1990s, the rapid adoption of Ethernet networks in the industrial world drove the evolution of SCADA application layer protocols. For example, the IEC adopted the Open System Interconnection (OSI) layer model to define its protocol framework. Other protocol user groups also slightly modified their 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

corporate WAN 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 the evolution 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.

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.



Protocol Stack for Transporting Serial DNP3 SCADA over IP

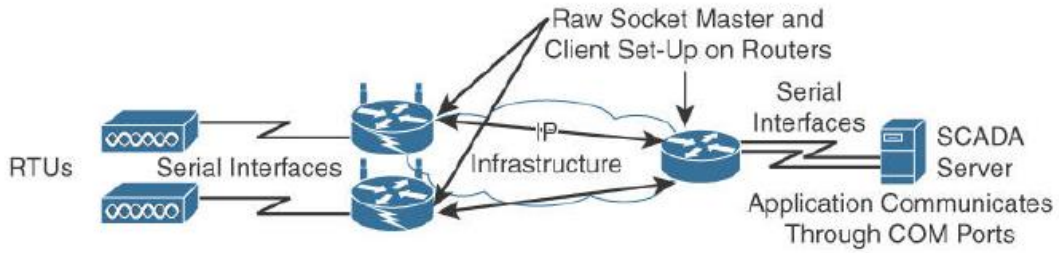
From the above figure

- 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.

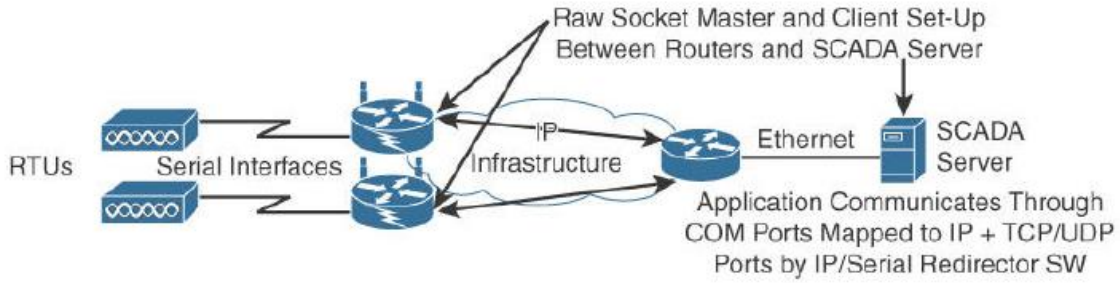
- Master stations may parse multiple DNP3 data link layer frames from a single UDP datagram, while DNP3 data link layer frames cannot span multiple UDP datagrams.
- Single or multiple connections to the master may get established while a TCP keepalive timer monitors the status of the connection. Keepalive 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.

Tunneling Legacy SCADA over IP Networks.

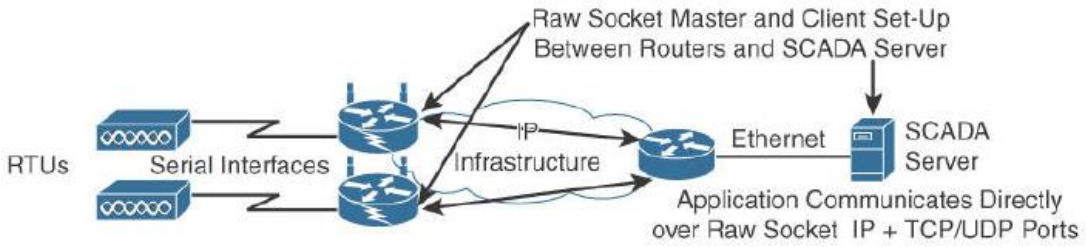
- Deployments of legacy industrial protocols, such as DNP3 and other SCADA protocols, in modern IP networks call for flexibility when integrating several generations of devices or operations that are tied to various releases and versions of application servers.
- Native support for IP can vary and may require 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.



Scenario A: Raw Socket between Routers – no change on SCADA server



Scenario B: Raw Socket between Router and SCADA Server – no SCADA application change on server but IP/Serial Redirector software and Ethernet interface to be added



Scenario C: Raw Socket between Router and SCADA Server – SCADA application knows how to directly communicate over a Raw Socket and Ethernet interface

In all the scenarios in Figure 6-3, 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 RTUs in each above Figure 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.

- In Scenario A in Figure 6-3, 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.

- In Scenario C in Figure 6-3, 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.

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.

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.

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 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.

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.

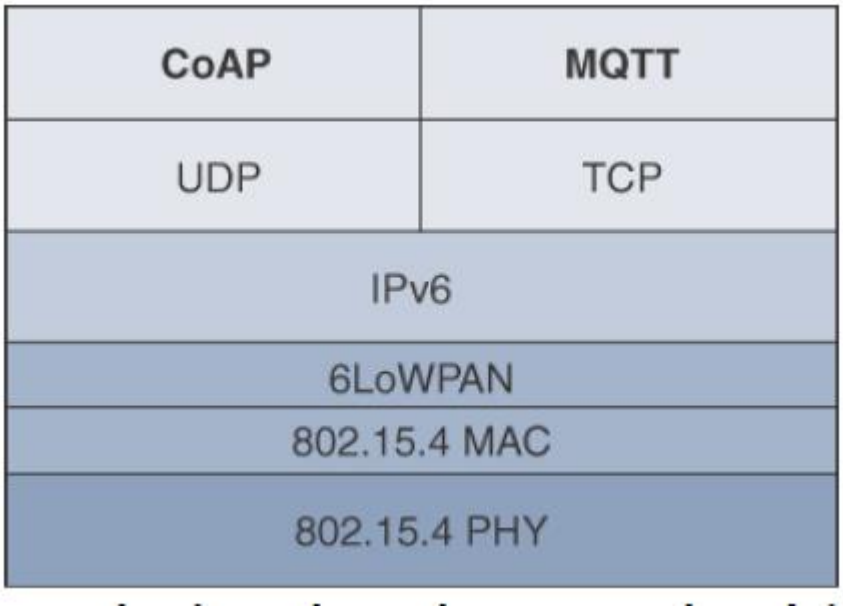
Interactions between real-time communication tools powering collaborative applications, such as voice and video, instant messaging, chat rooms, and IoT devices, are also emerging. This is driving the need for simpler communication systems between people and IoT devices.

In summary, the Internet of Things greatly benefits from the existing web based protocols. These protocols, including HTTP/HTTPS and XMPP, ease the integration of IoT devices in the Internet world through well-known and scalable programming techniques.

IoT Application Layer Protocols

When considering constrained networks and/or a large-scale deployment of constrained nodes, **verbose web-based and data model protocols, as discussed in the previous section, may be 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.**



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

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.

- Like HTTP, CoAP is a document transfer protocol. Unlike HTTP, CoAP is designed for the needs of constrained devices.

- CoAP packets are much smaller than HTTP TCP flows. Bitfields and mappings from strings to integers are used extensively to save space. Packets are simple to generate and can be parsed in place without consuming extra RAM in constrained devices.

- CoAP runs over UDP, not TCP. Clients and servers communicate through connectionless datagrams. Retries and reordering are implemented in the application stack.

- Removing the need for TCP may allow full IP networking in small microcontrollers. CoAP allows UDP broadcast and multicast to be used for addressing.

- CoAP follows a client/server model. Clients make requests to servers, servers send back responses. Clients may GET, PUT, POST and DELETE resources.

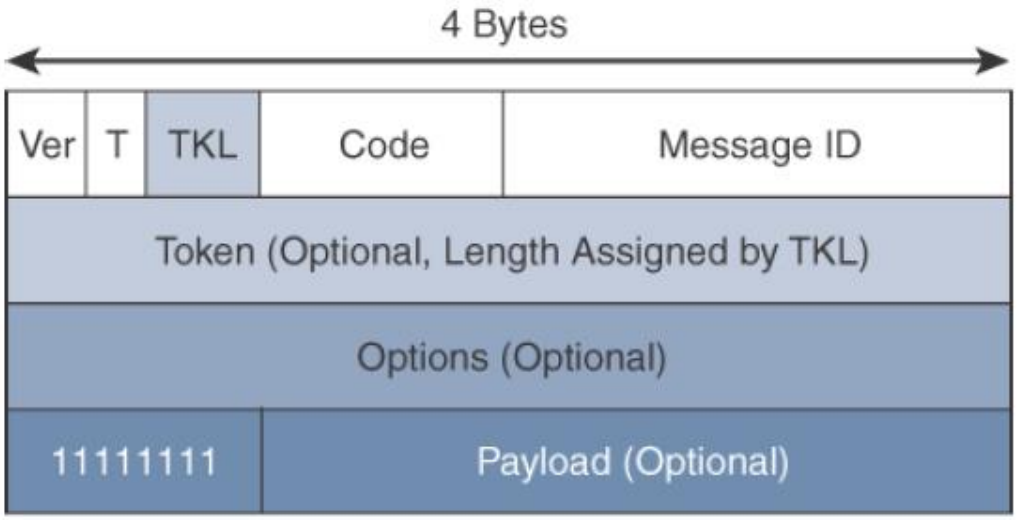
- CoAP is designed to interoperate with HTTP and the RESTful web at large through simple proxies.

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).

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 6-7 details the CoAP message format, which delivers low overhead while decreasing parsing complexity.



CoAP Message Format

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.

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.

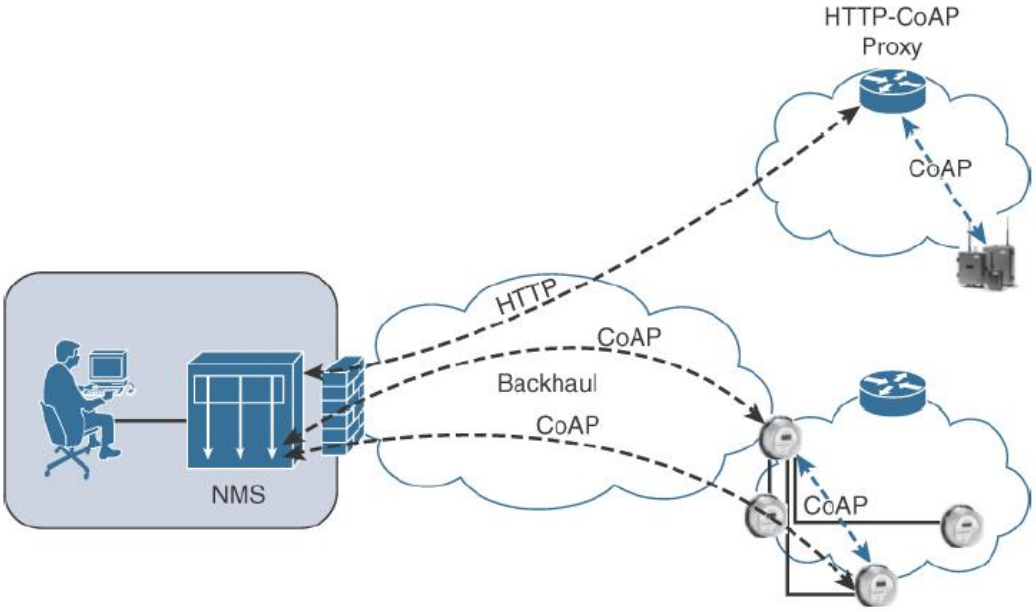
CoAP Message Fields

- 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, the maximum 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.

- 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.



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 the server identifies this resource. The server responds with a response code that may include a resource representation. The CoAP request/response semantics include the methods GET, POST, PUT, and DELETE.

```
coap-URI = "coap:" "//" host [":" port] path-abempty ["?" query]
coaps-URI = "coaps:" "//" host [":" port] path-abempty  ["?"
query]
```

CoAP defines four types of messages: confirmable, non-confirmable, acknowledgement, and reset. Method codes and response codes included in some of these messages make them 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.

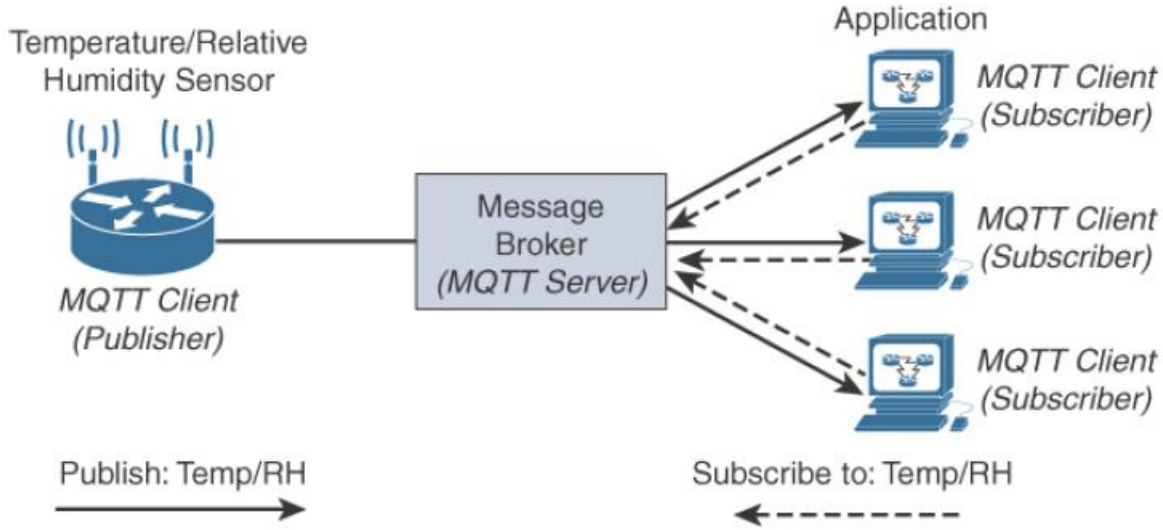
Message Queuing Telemetry Transport (MQTT)

At the end of the 1990s, engineers from IBM and Arcom (acquired in 2006 by Eurotech) 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).

MQTT (MQ Telemetry Transport) is a lightweight messaging protocol that provides resource-constrained network clients with a simple way to distribute telemetry information. The protocol, which uses a publish/subscribe communication pattern, is used for machine-to-machine (M2M) communication and plays an important role in the internet of things (IoT).

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.



MQTT Publish/Subscribe Framework

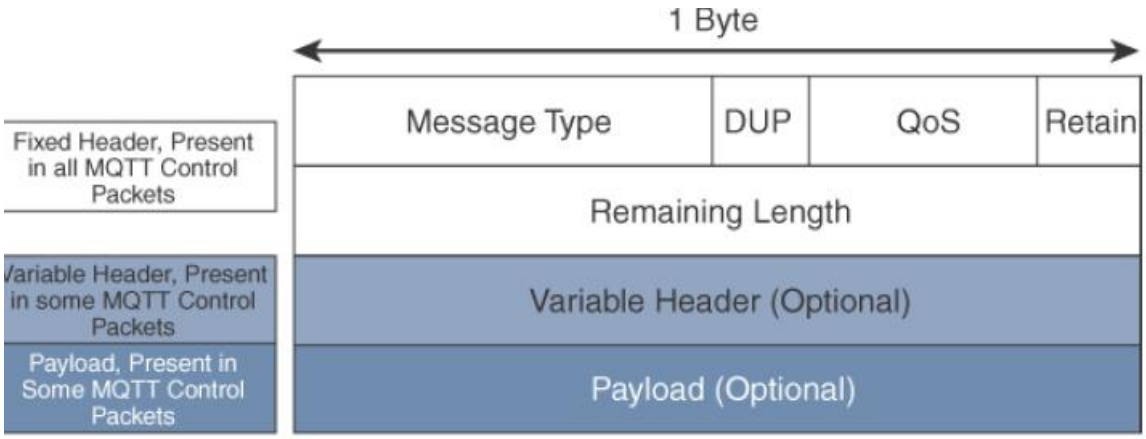
- **An MQTT client can act as a publisher to send data** (or resource information) to an MQTT server acting as an MQTT message broker. In the example illustrated in Figure, the MQTT client on the left side is a temperature (Temp) and relative humidity (RH) sensor that publishes its Temp/RH data.

- **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 6-10 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.

- 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.
- You should note that a control packet can contain a payload up to 256 MB.



MQTT Message Format

Compared to the CoAP message format in Figure 6-7, you can see that 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 Table.

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

MQTT Message Types

- **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.

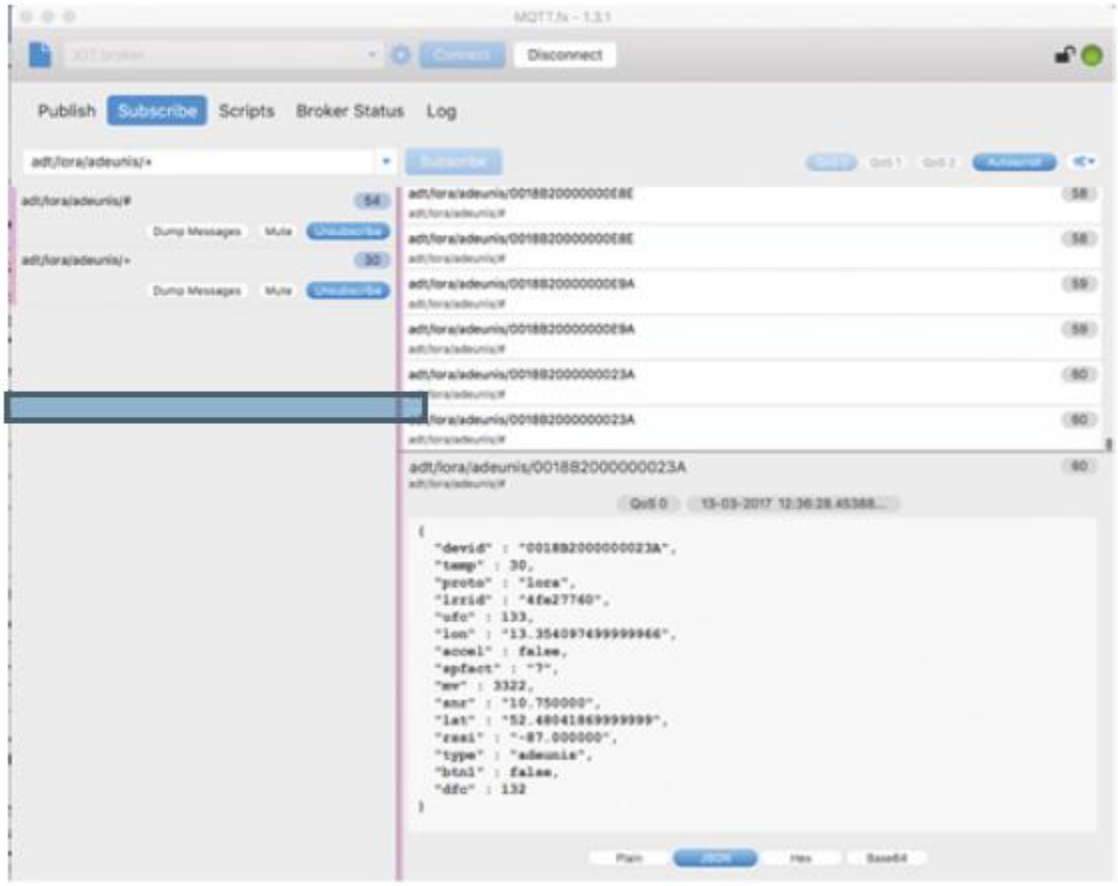
- The last mandatory field in **the MQTT message header is Remaining Length.**

- This field specifies the number of bytes in the MQTT packet following this field.

Working of MQTT protocol

- MQTT sessions between each client and server consist of four phases**: session establishment, authentication, data exchange, and session termination.**
- Each client connecting to a server has a unique client ID, which allows the identification of the MQTT session between both parties.
- When the server is delivering an application message to more than one client, each client is treated independently.

- Subscriptions to resources **generate SUBSCRIBE/SUBACK control** packets, while unsubscription is performed through the exchange of UNSUBSCRIBE/UNSUBACK control packets. Graceful termination of a connection is done through DISCONNECT control packet, which also offers the capability for a client to reconnect by re-sending its client ID to resume the operations.
- A message broker uses a topic string or topic name to filter messages for its subscribers.
- When subscribing to a resource, the subscriber indicates the one or more topic levels that are used to structure the topic name.
- The forward slash (/) in an MQTT topic name is used to separate each level within the topic tree and provide a hierarchical structure to the topic names. Figure illustrates these concepts with adt/lora.adeunis being a topic level and adt/lora/adeunis/0018B2000000023A being an example of a topic name.



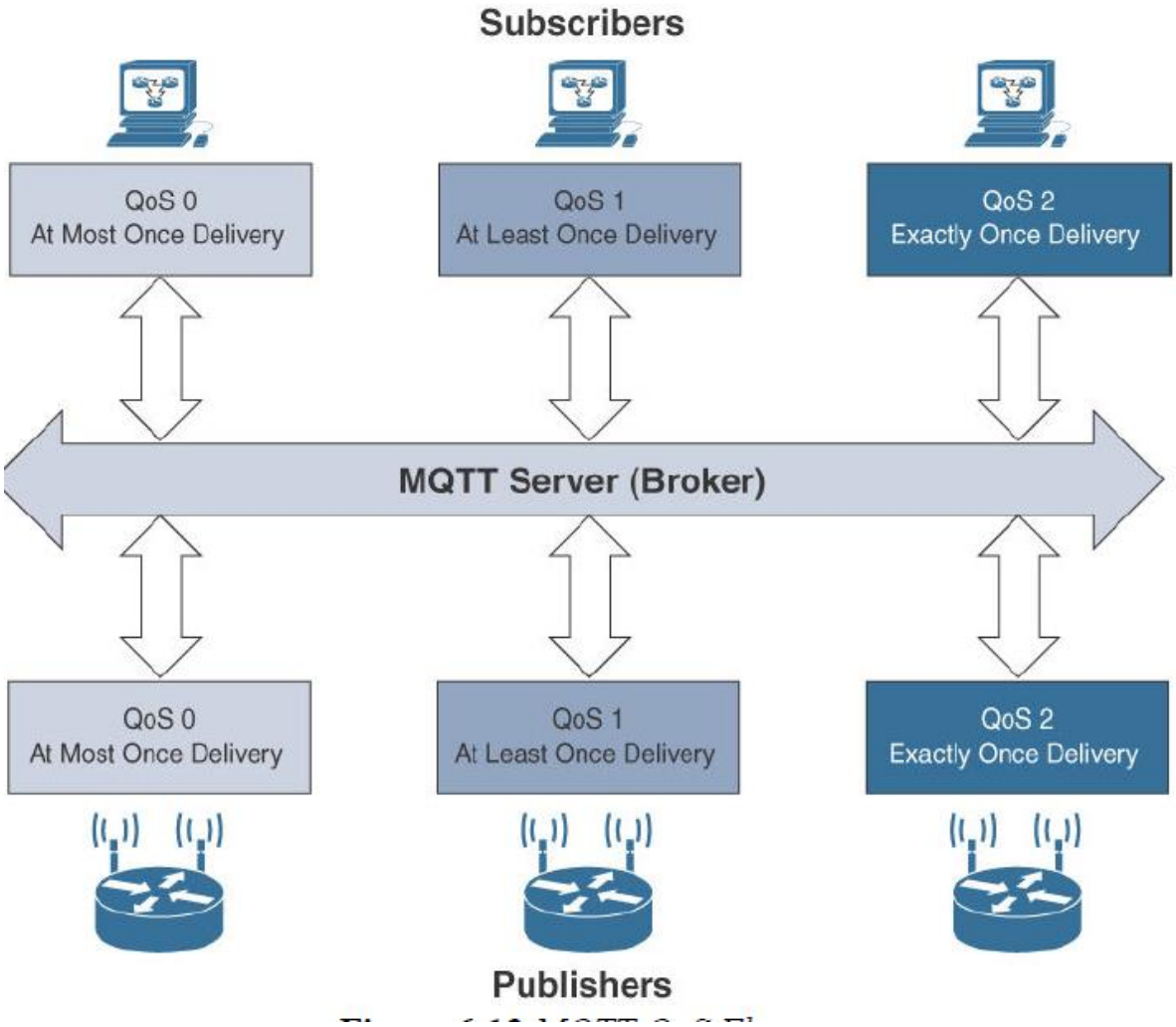
MQTT Subscription Example

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 from the 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 MQTT QoS:
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 the variable header. If the message is not acknowledged by a PUBACK packet, 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.



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.

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

Comparison Between CoAP and MQTT