CHAPTER 16

The 6LoWPAN Adaptation Layer

16.1 TERMINOLOGY

Before digging into the IP protocols developed for smart object networks, several terms that may be confusing need to be defined. According to [156] a LoWPAN is Low-power Wireless Personal Area Networks (LoWPANs) composed of devices conforming to the IEEE 802.15.4-2003 standard defined by the IEEE [129]. IEEE 802.15.4 devices are characterized by short range, low bit rate, low power, and low cost.

IEEE 80.15.4 networks have the following characteristics:

- Small packet size (the maximum transmission unit or MTU on IEEE 802.15.4 links is 127 bytes), which provides even less room for data when including other headers (as discussed in detail in Section 16.2).
- Support for both 16-bit short or IEEE 64-bit extended media access control (MAC) addresses.
- Low data rates; the IEEE 802.15.4 specification allows various data rates from 20 Kbits/s (868 MHz) to 250 Kbits/s (2.45 GHz).
- Support of star and mesh topologies.
- Constrained devices regarding power (e.g., battery-operated devices), memory, and CPU. Most of the time these devices are low cost.
- Large number of deployed devices in the network requiring scalable technologies.
- IEEE 802.15.4 networks are usually ad hoc networks since their location is usually not predetermined. Furthermore, some locations (e.g., mobile smart objects used for asset tracking, wearable sensors) may be moving devices.
- The nodes within a LoWPAN are interconnected by IEEE 802.15.4 links, which are usually unreliable, especially when compared to wired links such as Ethernet or fiber-optic links. This key aspect of such smart object networks has been discussed in Chapter 12.
- It is very common for nodes to be in sleep mode for long periods of time. Depending on the device, it can be in various sleep mode states that have a different impact on the energy consumption while in sleep mode and the speed at which the node can wake up (see Chapter 11 for more details).

An ad hoc network is a temporary type of Local Area Network (LAN). If you set up an ad hoc network permanently, it becomes a LAN

A LoWPAN is a Low-power and Lossy Network (LLN) where the links interconnecting the nodes are IEEE 802.15.4 links. When the Internet Engineering Task Force (IETF) 6LoWPAN Working Group was formed, it was decided to exclusively work on the required IPv6 protocol extensions for

LoWPAN (such as fragmentation and reassembly, header compression, neighbor discovery adaptation, etc.) where the nodes were exclusively interconnected by IEEE 802.15.4 links.

Then the Routing Over Low-power and Lossy network (ROLL) Working Group was formed to deal with routing issues in networks with similar characteristics at the IP layer thus alleviating the restriction of using IEEE 802.15.4 links, since by definition routing operates at the network layer. This led to the use of the more generic term Low-power and Lossy Network (LLN).

Note that the terms “nodes,” “routers” (when discussing a routing-related item), and even “devices” (since most smart objects performing sensing or actuating are usually routers) are used interchangeably.

16.2 THE 6LoWPAN ADAPTATION LAYER

Since IPv6 mandates supporting links with an MTU (Maximum Transmission Unit) of 1280 bytes, it was necessary for IEEE 802.15.4 links that have an MTU of 127 bytes to specify an adaptation layer below IP responsible for handling packet fragmentation and reassembly.

The MTU size of IEEE 802.15.4 links was purposely small to cope with limited buffering capabilities and to limit the packet error rate since the bit error rate (BER) is relatively high. Various header compression techniques have been added to the adaptation layer that are specified in [176] and [124]. The compression header techniques originally specified in [176] were improved in [124] in many ways: individual compression on the traffic class (TC) and flow label field, use of share contexts that is particularly useful when using non-link-local addresses, and optimizations for multicast addresses.

The IEEE 802.15.4 frame MTU is 127 bytes minus a set of protocol fields:

- Maximum MAC frame overhead: Frame control (2 bytes) + sequence number (1 byte) + addressing field (up to 20 bytes with the source and destination PAN ID and the source and destination 64-bit extended addresses) + FCS (2 bytes) = 25 bytes.
- MAC security header: 21 bytes (AES-CCM-128), 13 bytes (AES-CCM-64), and 9 bytes (AES-CCM-32).

In the worst case this only leaves 81 bytes (127 bytes − 25 − 21 = 81) for the data payload (IPv6 packets). After removing the size of the IPv6 header (40 bytes), there are 41 bytes left. Next, we must deduct the transport layer protocol header (8 bytes for UDP and 20 bytes for TCP), thus leading to a very short payload for the application layer.

This shows that an adaptation layer is needed to comply with the IPv6 requirement to support a minimum MTU size of 1280 bytes as well as to support compression techniques to reduce protocol overhead.

The 6LoWPAN adaptation layer provides three main services:

- Packet fragmentation and reassembly
- Header compression
- Link layer (layer 2) forwarding when multi-hop is used by the link layer

In most cases the use of efficient compression techniques allows most applications to send their data within a single IPv6 packet.

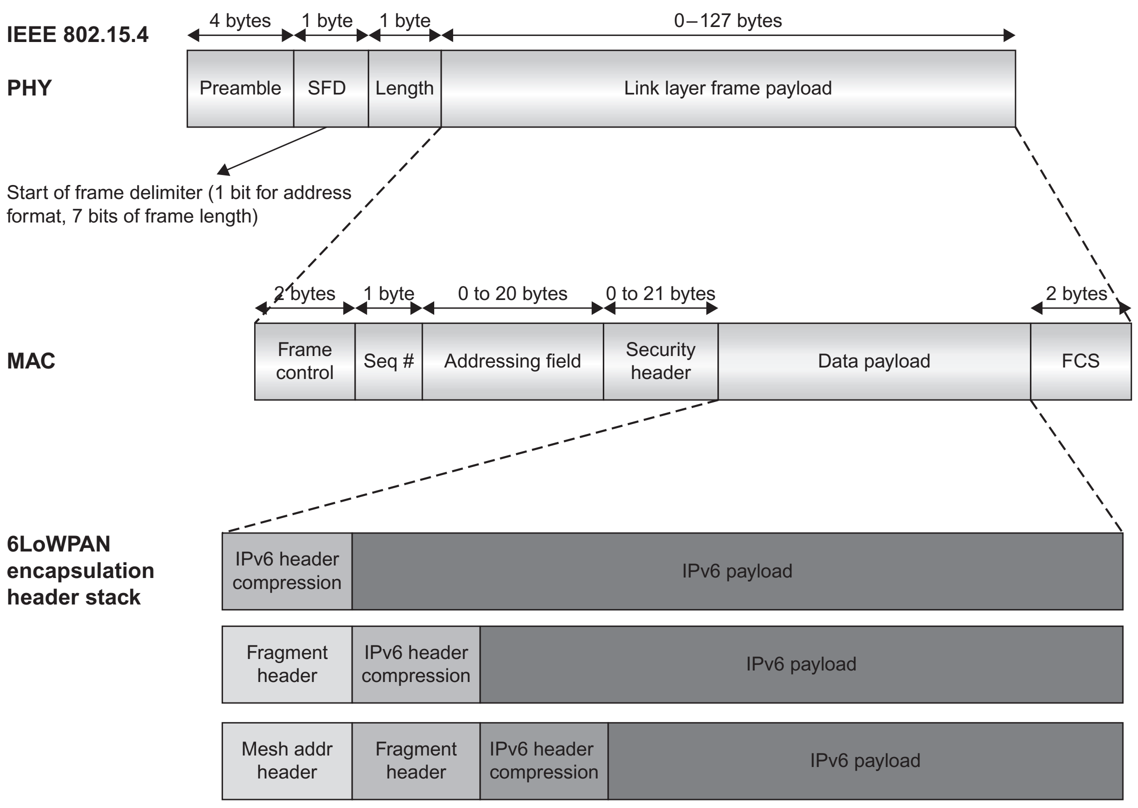


FIGURE 16.1

6LoWPAN encapsulation header stack.

As previously discussed in Chapter 12, IEEE 802.15.4 frames support the use of 16-bit short addresses (temporary addresses allocated by the personal area network or PAN coordinator or 64-bit long addresses (24 bits are used for the organizational unique identifier; OUI + 40 bits assigned by the chipset manufacturer).

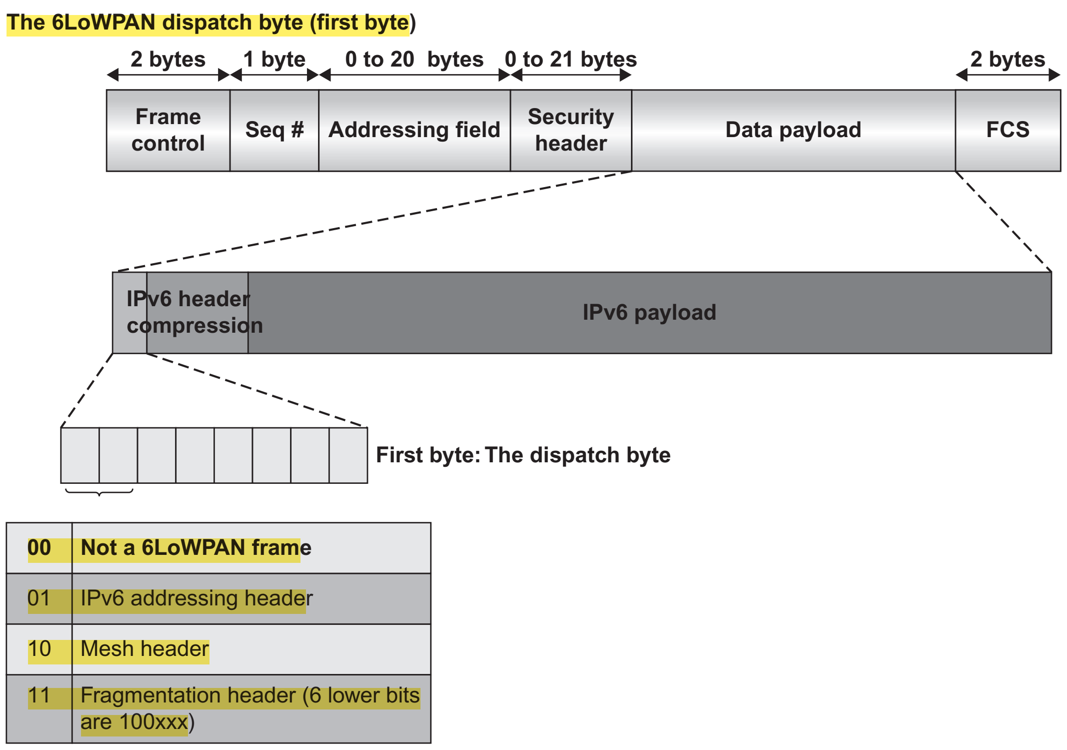
Similar to IPv6, the 6LoWPAN adaptation layer makes use of header stacking (headers are added only when needed).

The 6LoWPAN adaptation currently supports three headers: a mesh addressing header, the fragment header, and the IPv6 header compression header (they must appear in that order when present).

The 6LoWPAN adaptation layer defines what is called the "encapsulation header stack," which precedes each IPv6 datagram. The encapsulation header stack is shown in Figure 16.1.

As shown in Figure 16.2, the first byte of the encapsulation header identifies the next header. For example, if the first 2 bits are equal to 11, the next header is a fragmentation header.

If the first 8 bits are equal to 01000001, what follows is an IPv6 *uncompressed* packet. In contrast, a value of 01000010 indicates that what follows is a header related to a compressed header using HC1 compression (see Section 16.2.3 for details on 6LoWPAN header compression techniques).



00	Not a 6LoWPAN frame
01	IPv6 addressing header
10	Mesh header
11	Fragmentation header (6 lower bits are 100xxx)

FIGURE 16.2

Dispatch byte of the IPv6 header compression header.

16.2.1 The Mesh Addressing Header

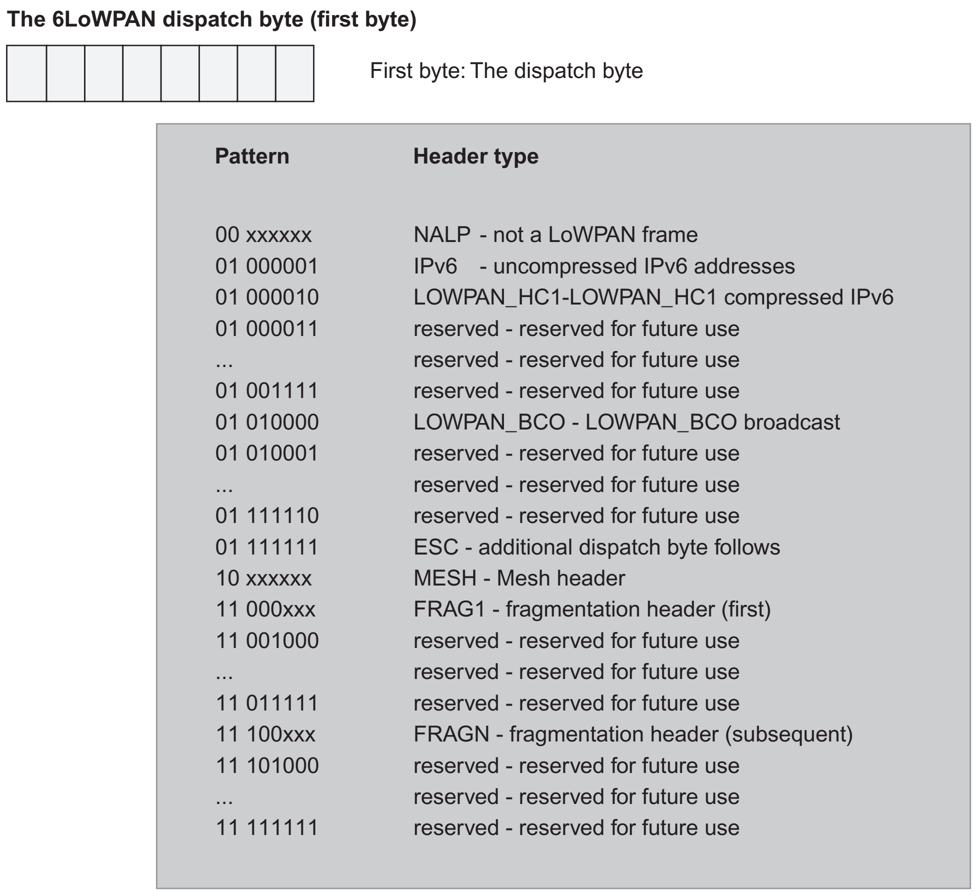
The mesh addressing header is used in conjunction with a mesh-under "routing" approach where nodes that are not in direct communication make use of multi-hop "routing" at the link layer using link layer addresses. According to IEEE 802.15.4, only full function devices (FFDs) perform mesh-under operation. Reduced function devices (RFDs) systematically send all of their traffic to FFDs.

The source and destination nodes are then referred to as the originator and final destination, respectively.

As shown in Figure 16.2, the first 2 bits of the dispatch byte identify the presence of a mesh-header and are equal to 10.

Figure 16.4 shows the various bits of mesh addressing type and header:

- Bit 2 (V, Very first bit):
  0: The originator address is an IEEE extended 64-bit address (EUI-64).
  1: The originator address is a short 16-bit address.


The 6LoWPAN dispatch byte (first byte)
First byte: The dispatch byte

Pattern	Header type
00 xxxxxx	NALP - not a LoWPAN frame
01 000001	IPv6 - uncompressed IPv6 addresses
01 000010	LOWPAN_HC1-LOWPAN_HC1 compressed IPv6
01 000011	reserved - reserved for future use
...	reserved - reserved for future use
01 001111	reserved - reserved for future use
01 010000	LOWPAN_BCO - LOWPAN_BCO broadcast
01 010001	reserved - reserved for future use
...	reserved - reserved for future use
01 111110	reserved - reserved for future use
01 111111	ESC - additional dispatch byte follows
10 xxxxxx	MESH - Mesh header
11 000xxx	FRAG1 - fragmentation header (first)
11 001000	reserved - reserved for future use
...	reserved - reserved for future use
11 011111	reserved - reserved for future use
11 100xxx	FRAGN - fragmentation header (subsequent)
11 101000	reserved - reserved for future use
...	reserved - reserved for future use
11 111111	reserved - reserved for future use

FIGURE 16.3

Value of the 6LoWPAN dispatch byte.

- Bit 3 (F, Final destination):
  0: The final address is an IEEE extended 64-bit address (EUI-64).
  1: The final address is a short 16-bit address.
- Bits 4 through 7 (HopLeft): The HopLeft field value is decremented by each node before sending the packet to its next hop. When the HopLeft field reaches the value of 0, the packet is simply discarded. When equal to 15, an additional byte (called the deep hops left) immediately follows when forwarding along a path with more than 14 hops is needed.
- The originator and final link layer address fields then follow (16 or 64 bits).

It is possible to use short 16-bit addresses for broadcast and 64-bit addresses as a source address since the V and F permit the use of different link layer address formats.

With mesh-under routing it is necessary to provide the originator and final destination as well as the hop-by-hop source and destination addresses.

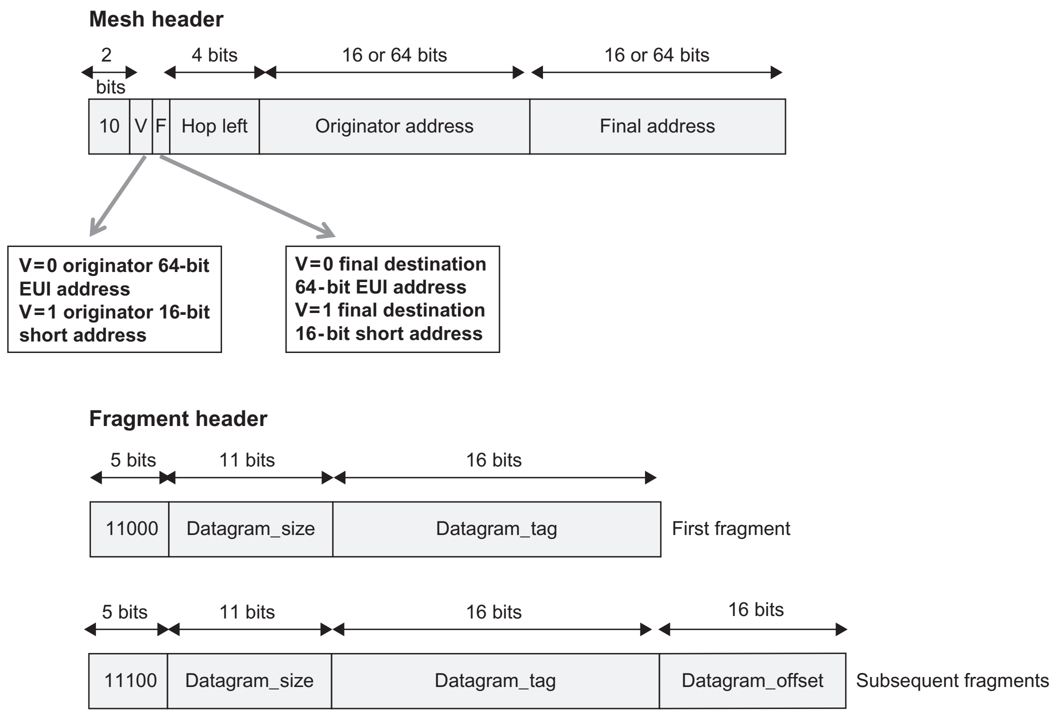


FIGURE 16.4

6LoWPAN mesh and fragmentation headers.

Thus the set of link layer addresses is as follows. When a node A sends a frame to a final destination C via the node B:

- The originator address of the mesh header is set to the link layer address of A.
- The final destination address of the mesh header is set to the link layer address of C.
- The source address of the IEEE 802.15.4 frame is the address of the node sending the frame (A).

The destination address of the IEEE 802.15.4 frame is the link layer address of the next-hop node as determined by the mesh-under routing protocol (B in this example). Upon receiving the frame, B performs the following process:

- The hop left field is decremented.
- If the hop left field is not equal to 0 (if equal to 0, the frame is discarded), then B determines that the next hop is C.
- The originator and final destination address of the mesh header are unchanged.
- The source address of the IEEE 802.15.4 frame is set to the link layer address of B.
- The destination address of the IEEE 802.15.4 frame is set to the link layer address of C.

This is similar to the mode of operation with IP routing over a link layer where the source and destination addresses of the IP packet are never changed, and the source and destination addresses present in the link layer frame correspond to the address of two adjacent nodes (connected by a common link layer).

As previously discussed, there is no mesh-under protocol defined. For further discussion about routing at multiple layers see Chapter 5.

16.2.2 Fragmentation

Fragmentation may be required at the 6LoWPAN adaptation layer when the IPv6 payload cannot be carried within a single IEEE 802.15.4 frame because it exceeds the MTU size. In this case, the link frame is broken into multiple link fragments using the fragment header shown in Figure 16.4. All fragment sizes are expressed in units of 8 bytes. The first fragment does not contain a datagram offset, which makes it slightly different from the subsequent fragment.

Description of the fragment fields (see Figure 16.4):

- datagram_size: This 11-bit field is used to indicate the size in 8-byte units of the original IPv6 packet (or IPv6 fragmentation also taking place at the IP layer). Link layer fragmentation supports a 1280-byte packet as mandated by the IPv6 specification [51]. The datagram_size may only be needed in the first link fragment and then elided in other link fragments. The drawback of this approach is that subsequent link fragments (other than the first link fragment) may arrive first, especially in the presence of multi-hop routing. In this case the receiver would not know how much memory should be allocated for the entire frame.
- datagram_tag: This field is used in conjunction with the IEEE 802.15.4 source address (or originator address if a mesh header is present), the IEEE 802.15.4 destination address (or the final destination address if a mesh header is present), and the datagram_size to uniquely identify the fragmented frame and must be identical for all link fragments. It is recommended to increment the datagram_tag for successive fragmented frames.
- datagram_offset: The 8-bit datagram_offset field is present in all link fragments except the first fragment and indicates the offset in 8-byte units from the beginning of the payload datagram.

[176] specifies the use of a reassembly timer that is started when receiving the first link fragment and upon the expiration of which, if not all link fragments have been received, all fragments must be discarded. The maximum value of the reassembly timer is 60 seconds.

16.2.3 6LoWPAN Header Compression

16.2.3.1 Header Compression Using LOWPAN_HC1 and LOWPAN_HC2

A plethora of IP compression techniques have been designed over the past decade (e.g., ROHC, see [18]). These techniques rely on stateful flow-based compression optimized for long-lived flows. The basis of this principle consists of suppressing common values within a long-lived flow, which

CHAPTER

RPL Routing in Smart Object Networks 17

17.1 INTRODUCTION

As already discussed in Chapter 14, the Internet Engineering Task Force (IETF) formed a new Working Group called ROLL (Routing Over Low-power and Lossy networks; http://www.ietf.org/dyn/wg/charter/roll-charter.html) in 2008 with the objective of specifying routing solutions for Low-power and Lossy Networks (LLNs). The first objectives of the Working Group were to produce a set of routing requirements (discussed in Section 17.2), determine whether or not existing IETF routing protocols would satisfy the requirements spelled out in the routing requirement documents, and establish a routing security framework and define new routing metrics for routing in LLNs. The Working Group quickly converged on the fact that none of the existing routing protocols would satisfy the fairly unique set of routing requirements for LLNs. Thus ROLL was re-chartered to design a new routing protocol called RPL (Routing Protocol for Low-power and Lossy Networks) explained in detail in this chapter. Note that the terminology used in ROLL specifications can be found in [248].

17.2 WHAT IS A LOW-POWER AND LOSSY NETWORK?

When not familiar with the environment of IP smart object networks interconnected by lossy links, one may wonder: How lossy is lossy? Ethernet and Optical links have remarkably low BERs. A lossy link is not just a link with higher BER uniformly distributed errors. Packet drops on lossy links are extremely frequent, and the links may become completely unusable for quite some time for a number of reasons such as interference. This observation has strong consequences on the protocol design. Indeed, knowing that link failures are frequent and usually transient also means that the routing protocol should not overreact to failures in an attempt to stabilize under unstable conditions. For example, if node A selected node B as its preferred next-hop, and as a result of temporary lack of connectivity between A and B, node A chooses an alternate next-hop C and immediately triggers a re-computation of the routing table. This would not only lead to routing instabilities but would generate a significant amount of control plane traffic impacting the entire network.

It is worth pointing out that by lossy link what immediately comes to mind are wireless links, but remember that Powerline communication (PLC) links are also lossy.

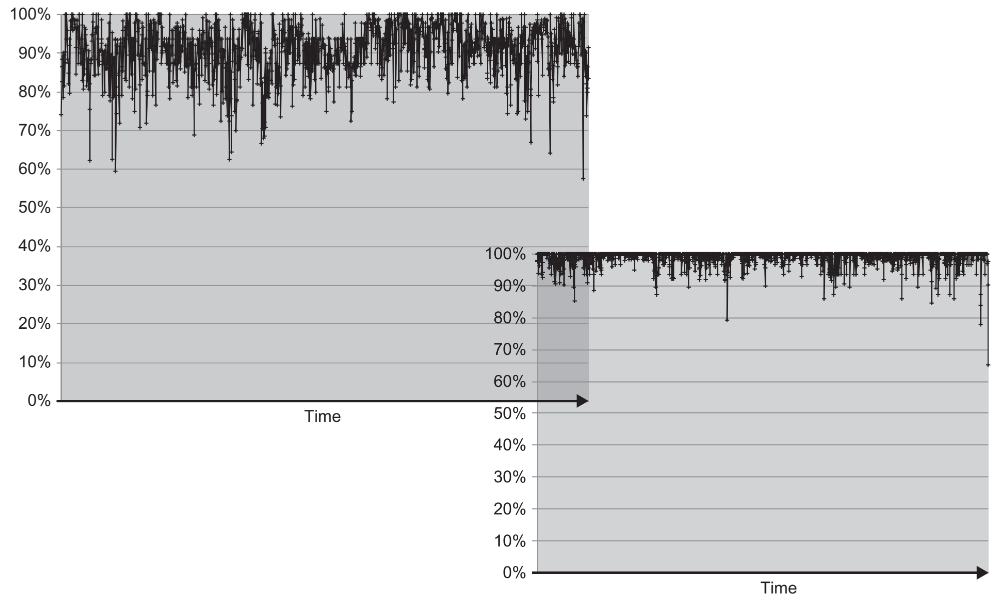


FIGURE 17.1

Packet Delivery Ratio for two IEEE 802.15.4 links.

Figure 17.1 shows the packet delivery ratio (PDR) for two low-power IEEE 802.15.4 links as a function of time (in seconds). The PDR significantly varies from 60 to 100%.

17.3 ROUTING REQUIREMENTS

When defining a new protocol, it is always tempting to start right away with the protocol specification, processing rules, packet encoding, etc. But without a clear understanding of the requirements, this unavoidably leads to further difficulties when trying to adapt the protocol as new requirements are added. To avoid such situations, IETF Working Groups usually produce requirement documents that follow the “informational” track (please refer to Chapter 14 for more details on standardization tracks). In the case of the ROLL Working Group one of the main challenges was to determine the scope of the work. In contrast with traditional IP networks (e.g., a core Service Provider network), LLNs can greatly vary from each other. A mobile Delay Tolerant Network (DTN) used to study wildlife does not have much in common with a dense “always on” network used for industrial automation. Thus the choice was made to limit the scope to four main applications: urban networks (including Smart Grid applications), building automation, industrial automation, and home automation. These applications are representative of other types of networks and there was an urgent need to design routing solutions for them. Thus it was believed that by addressing the routing requirements of these applications, a routing protocol for LLN would address the vast majority of routing requirements of smart object networks.

Requirement documents usually use normative language in IETF terms (see [23]): the MUST, SHOULD, and MAY in these documents indicate if a feature is mandated or simply desirable. MUST, SHOULD, and MAY are used in protocol specifications. For example, if a protocol document specifies that a feature MUST be supported then an implementation is not compliant with the RFC if it does not support the feature in question.

The ROLL Working Group has produced the following four routing requirements: [169], [24], [197], and [57]. These sections provide an overview of the major routing requirements spelled out in these documents (the MUST).

These routing requirements make no assumption on the link layer in use; they specify a list of routing requirements for networks made of LLNs.

- Unicast/anycast/multicast: Several requirement documents list the support of unicast, anycast, and multicast traffic as mandatory. The support of the multicast traffic is explicitly listed in the ROLL Working Group charter.
- Adaptive routing: Most requirements specify the need for adaptive routing where new paths are dynamically and automatically recomputed as conditions change in the network (e.g., link/node failure, mobility, etc.). Furthermore, the routing protocol must be able to compute routes optimized for different metrics (e.g., minimize latency, maximize reliability, etc.). [169] also specifies that the routing protocol must be able to find a path that satisfies specific constraints such as providing a path with a latency lower than a specified value.
- Constraint-based routing: All documents mention that the routing protocol has to support constraint-based routing to take into account various node characteristics used as constraints such as energy, CPU, and memory as well as link attributes ([197]) such as link latency.
- Traffic characteristics: There are a number of LLNs highly focused on data collection (e.g., telemetry) where most of the traffic is from leaf nodes such as sensors to a data collection sink. This type of traffic is also referred to as multipoint-to-point (MP2P) traffic. It is often necessary in these networks to also support point-to-multipoint (P2MP) traffic; for example, when the sink sends a request to all nodes in the network, acknowledgments in the context of reliable messaging are necessary or a central management tool performs a software update. Furthermore, as pointed out in [169] and [24], the routing protocol must support point-to-point (P2P) communication between devices in the network. The routing protocol must also support the computation of parallel paths (not necessarily disjoint) to absorb bursts of traffic more efficiently. In some cases ([197]) it was required to not just support Equal Cost Multiple Path (ECMP). Note that other routing protocols such as ISIS or OSPF only support ECMP (avoiding loops with non equal load balancing is somewhat challenging).
- Scalability: As discussed throughout the entire book, LLNs are composed of a very large number of nodes, thus scalability is very important. The routing protocol requirement documents indicate a number of nodes between 250 [24] to 1000 [169] and up to 10^4 in [57]. There are deployments that even require the support of millions of nodes (see Part III); in this particular case, the deployment of the routing protocol may follow specific rules (e.g., network partitioning).
- Configuration and management: As expected, there is a long list of requirements related to configuration. In most documents, it is clearly spelled out that the routing protocol must be able to autoconfigure with minimal or even 0-configuration. In other words, the end user must be able to place the node in its environment without intervening in the configuration and the routing protocol must

be able to join the routing domain and start functioning from a routing perspective (see [197] for a detailed example). [24] also specifies that the routing protocol must be able to isolate a misbehaving node to limit/eliminate its impact on other nodes. [169] mentions that an application should not require any reconfiguration even after replacement of the devices (in other words, a new IP address must not be reassigned to the node).

- Node attribute: [169] mentions that when there are sleeping nodes in the network (a frequent situation with battery-operated nodes), the routing protocol must discover the capability of a node to act as a proxy. A packet could be delivered to a proxy that could relay the packet to the destination once awakened.
- Performance: Indicating performance numbers in requirement documents is always a risky proposition. Performance may not only greatly vary between implementations but is subject to potential changes as new applications emerge. A protocol should never be designed with hard numbers in mind to preserve its future use. Thus performance numbers in requirement documents should not be seen as "hard" numbers or bounds but simple indications providing some order of magnitude. For example, [197] mentions that the routing protocol must find routes and report success or failure within several minutes. In [24], the routing protocol must provide mobility with a convergence time below 0.5 s and it must converge within 0.5 s if no nodes have moved and within 2 s if the destination has moved. But again, these numbers should be seen as indicative as opposed to hard performance targets or bounds.
- Security: As discussed in Chapter 8 and shown in Part III, security is very important in most LLNs. There are some LLNs (e.g., Smart Cities telemetry networks) where minimal security is required, but in most cases (e.g., Smart Grid, building automation, industrial automation, etc.) security is absolutely critical. Authentication is listed as an absolute must in all documents. Encryption is also an absolute must. Note that [169] mentions that "the routing protocol must gracefully handle routing temporal security updates (e.g., dynamic keys) to sleeping devices on their '*awake*' cycle to assure that sleeping devices can readily and efficiently access the network."

How should conflicting objectives be dealt with? It is always challenging to consider a set of requirements dictated by several applications that significantly differ from each other. The first naïve approach is to consider the union of all of the requirements. Unfortunately, such an approach is usually unrealistic or undesirable. The union of all requirements may not be possible considering the constrained nature of smart objects and the need to bound the complexity of the protocol. There are even cases where some of these requirements are contradictory. Even if all of these requirements were satisfied by a single routing protocol, the results may not be beneficial. Why would a routing protocol operating in a building have to support features needed for urban networks? It may be more advantageous to only support the required features to limit the resource (node and network) consumption in the network. The other approach adopted by RPL was to design a modular routing protocol where the core component of the application would be specified by the RPL specification with optional features activated only where and when needed. For example, RPL specifies how to build a destination oriented directed acyclic graph (DODAG), but the characteristics of the DODAG are specified by an objective function. For the time being, think of a DODAG as a logical routing topology over a physical network that is built by the routing protocol to meet specific criteria. How RPL builds DODAGs is further explored in detail in the rest of this chapter. It is even possible for a node to join multiple DODAGs (if the application requires different objectives that must be realized through the use of multiple DODAGs) and mark the traffic according to the DODAG characteristics in support of Quality of Service (QoS) awareness and constrained-based routing. Then applicability documents will be produced to provide guidance on how the core RPL protocol could be used, in conjunction with specific objective functions, and configured to meet specific requirements supporting the application and environment.

17.4 ROUTING METRICS IN SMART OBJECT NETWORKS

Routing metrics are a critical component of the routing strategy and have been studied for decades. Most of the IP routing protocols used in today's networks such as OSPF [179] or IS-IS [131] use static link metrics. The network administrator is responsible for configuring the link metrics, which may reflect the link bandwidth, delay, or combine several metrics. Some Service Providers are combining up to three metrics (e.g., delays, bandwidth, cost) in the link metric. Then the routing protocol computes the shortest path taking into account these static link metrics.

Several attempts were made to use dynamic link metrics. For example, extensive studies were made in ARPANET-2 to dynamically compute the link metric based on the averaged queue length to reflect the level of congestion. These strategies were abandoned due to the difficulty in designing stable systems. One of the main challenges with dynamic metrics is to carefully control the rate at which new metrics are advertised. Frequent link metric refreshers provide a high level of accuracy but may also lead to routing oscillation. For example, when the link metric reflects the link utilization, increasing the metric discourages traffic from traversing the link and triggers the rerouting of traffic in other parts of the network. As the link utilization decreases, the link metric also decreases thus attracting more traffic. If not controlled carefully, such strategies unavoidably lead to traffic oscillation and thus to jitter, potential packet reordering, and so on. Extreme care must be taken to limit the control traffic overhead in LLN where bandwidth and energy are usually scarce resources. In addition to the potential traffic oscillation, routing updates too frequently create congestion in the network that would drain energy, which may be a real issue for battery-operated nodes.

Another characteristic of the current routing protocol's metrics is that they are only related to links, which makes perfect sense in the current Internet because most core routers are not traffic bottlenecks.

In contrast, routing in LLN does require more sophisticated routing metrics strategies.

Let's clarify the distinction between routing *metric* and *constraint*. A metric is a scalar used to determine the best path according to some objective function. For example, if the link metric is representative of the link propagation delay, the path cost represents the total propagation delay to the destination and the objective function may specify finding the shortest path based on the propagation delay. Some metrics may not be additive; for example, the objective function may be to find the path where the minimum link quality is maximized. A constraint is used to include or eliminate links or nodes that do not meet specific criteria (this is usually referred to as *constraint-based routing*). For example, the objective function may not select any path that traverses a node that is battery-operated or a link that does not provide link layer encryption. The objective function may combine link/node metrics and constraints such as "find the path with the minimum delay that does not traverse any non-encrypted link." An example is provided in Section 17.5.

The set of link and nodes metrics/constrained for RPL are defined in [250] and discussed in the next section. [250] allows routing objects to be defined as constraints or metrics with a great deal of flexibility. Let's consider the link quality level (LQL). The LQL is an integer between 0 and 3 that characterizes the link quality (poor, fair, good). The objective function (OF) may stipulate to prune links with a "poor" quality level (LQL is used as a constraint) or to find the path that provides the minimum number of links with poor quality (LQL is used as a metric). This applies to all routing objects that can be used as a metric or constraint.

17.4.1 Aggregated Versus Recorded Routing Metrics

The path cost is defined as the sum of the cost of all links along the path. This implicitly makes use of aggregated metrics. For example, if the metric reflects the link's throughput where the metric is inversely proportional to it, the best path is the path with the lowest cost (the path cost is the sum of all link metrics along the path). On the other hand, in some cases it might be useful to record each individual link metric as opposed to an aggregated value. In the reliability metric, one approach adds the link's LQL along the path (aggregated metric), but this comes with a loss of information in which case it might be useful to record the LQL of all links along the path. [250] supports both aggregated and recorded metrics.

17.4.2 Local Versus Global Metrics

A metric is said to be local when it is not propagated along the DODODAG. In other words, a node would indicate its local cost (in contrast with a global metric), but the cost will not be propagated any further.

17.4.3 The Routing Metrics/Constraints Common Header

[250] specifies a common header for all metrics and constraints with several flags used to indicate whether the routing object refers to a routing metric or a constraint, if the routing object is local versus global, if the global metric is aggregated versus recorded, if a constraint is optional or mandatory, and if a metric is additive or reports a maximum/minimum.

17.4.4 The Node State and Attributes Object

The node state and attribute (NSA) object is used to report various node state information and node attributes.

Nodes may act as traffic aggregators. Knowing that a node can aggregate traffic may influence the routing decision in an attempt to reduce the amount of traffic in the network. It is likely that a single flag will not suffice and additional information will have to be specified.

Nodes may have limited available resources. Extensive discussions took place in the ROLL Working Group to define which node parameters should be provided. One scheme would have been to report the available CPU processing power, available memory, etc. But this would become extremely bandwidth intensive and irrelevant considering how quickly such metrics vary. It was thus decided to simply make use of a 1-bit flag set when a node sustainably experiences some level of congestion. It is the responsibility of the node to determine, according to local policy, when the flag should be set potentially triggering traffic rerouting to avoid that node.

17.4.5 Node Energy Object

Energy is a critical metric in LLNs, especially in the presence of battery-operated nodes. The approach taken by [250] provided several levels of granularity to characterize the node energy: (1) the node power mode, (2) estimated remaining lifetime and potentially, and (3) potentially some detailed set of power-related metrics and attributes.

1. The node power mode: Three flags are used to indicate whether the node is main-powered, battery-powered, or if the node is powered by energy scavenging (solar panels, mechanical, etc.).
2. The approach to estimated remaining lifetime provides some indication of the power level for both battery-operated and scavenging nodes. With the battery-operated node, the unit is the current expected lifetime divided by the desired minimum lifetime. [250] provides two examples of how to compute this value.

   If the node can measure its average power consumption, then H can be calculated as the ratio of desired max power (initial energy E_0 divided by desired lifetime T) to actual power H = P_max/P_now. Alternatively, if the energy in the battery E_bat can be estimated, and the total elapsed lifetime, t, is available, then H can be calculated as the total stored energy remaining versus the target energy remaining: H = E_bat/[E_0 (T-t)/T].

   In the latter case (scavenger), the unit is a percentage (power provided by the scavenger divided by the power consumed by the application).
3. The detailed set of power-related metrics and attributes may potentially be used and is to be defined in the future.

17.4.6 Hop-count Object

The hop-count object simply reports the number of hops along the path.

17.4.7 Throughput Object

The throughput object is used to report the link throughput. When used as a metric, the throughput can be used as an additive metric or to report a maximum or a minimum.

17.4.8 Latency Object

The latency object is used to report the path latency. Similar to the throughput, latency can be used as a metric or a constraint. When used as a metric the latency object expresses the total latency (additive metric) and the maximum or minimum latency along the path. When used as a constraint, the latency can be used to exclude links that provide greater latency than predefined values.

17.4.9 Link Reliability Object

Routing protocols such as OSPF or IS-IS do not use reliability metrics simply because links used in the Internet such as SONET/SDH, Optical links, and Ethernet are extremely reliable with low error rates. They do fail and a plethora of fast recovery mechanisms have been defined, but the link quality usually expressed as BER for these types of links is not used for path selection. The situation is radically different in LLNs where links are lossy and not only can the BER be high, but the link states can vary quite significantly over time. Figure 17.1 illustrates the PDR for two links (indoor and outdoor) over time. This stresses the importance of considering the "lossyness" of a link when computing the best path to a destination. Very similar lossy characteristics can be shown in PLC links.

Many research papers have investigated a set of reliability metrics for lossy links such as low power links (e.g., [50], [85]). The most popular reliability metric thus far is the expected transmission (ETX) count metric, which characterizes the average number of packet transmissions required to successfully transmit a packet. The ETX is consequently tightly coupled to the throughput along a path. Several techniques have been proposed to compute ETX.

One method described in [50] sends regular probes in *each direction* to compute the delivery ratio for a specific link. ETX is defined as 1/(Df * Dr) where Df is the measured probability that a packet is received by the neighbor and Dr is the measured probability that the acknowledgment packet is successfully received. One way to compute Df and Dr is to send probes at regular time intervals, since both end points of the link know the frequency at which probes are sent. By reporting the number of received probes in the opposite direction, each node can easily compute both values. Other proposals have been made in [85] and [150].

It is important not to specify at the IETF the method for computing ETX values. The ETX is a link-specific quantity and the technique used to compute the ETX value should be independent of the link layer and not specified by the network layer that only carries it for routing protocol decisions. Some links may use link layer mechanisms, and in other cases probing techniques and the ETX value may be derived from one of these techniques or any combination.

The ETX for a path is computed as the sum of the ETX for each link along the path (e.g., RPL reports cumulative path ETX as discussed next).

17.4.10 Link Colors Attribute

There are circumstances where it may be useful to "color" a link to report a specific property. Such mechanisms have been defined in other protocols such as IS-IS, for example, to indicate that a link is protected with lower layer recovery mechanisms. A similar approach is adopted by RPL. The link color is encoded using a bit vector and the meaning of each color is left to the implementer. As described later in this section, RPL computes paths over a dynamically built DODAG. The DODAG root uses an OF required for each node along the path reporting path metrics to also report the set of colors of each link along the path. For example, suppose that the color blue is used to indicate the support of the link layer encryption. Upon receiving the path metric, if link colors are recorded, a node may decide to elect as a parent the parent reporting paths with encrypted links (blue links) or with the maximum number of blue links in the absence of a path exclusively made of blue links.

17.5 THE OBJECTIVE FUNCTION

The routing metric is insufficient for the routing protocol to compute the "best" path. The OF may be so simple that it could be implicit. For example, the OF of RIP [163] is to select the path with minimal hop count. OSPF or IS-IS would compute the paths that provide the minimum cost where the path cost is simply the sum of the static link cost along the path. In other cases such as MPLS TE the OF may be slightly more complex: "find the shortest path according to some metric such as the OSPF/IS-IS metric or the Traffic Engineering metric that satisfies some constraint such as the available reservable bandwidth or the type of recovery protection provided by the link." This is known as constraint-based routing. Still, the objective may be significantly more complicated. It is supported by the path computation element (PCE) architecture (http://www.ietf.org/dyn/wg/charter/pce-charter.html) to compute sophisticated MPLS Traffic Engineering Label Switch Paths (TE LSP). For example, the request might be to compute the shortest constraint path with multi-metric optimization (a Nondeterministic Polynomial (NP)-complete problem).

With LLNs there is strong interest in using several OFs because deployments greatly vary with different objectives and a single network may support traffic with very different requirements in terms of path quality. Consider the case of a mixed network with battery- and main-powered nodes, a variety of high and low bandwidth links, and two main applications (telemetry and critical alarms). This is a situation where it might be extremely useful for each node supporting both applications to be able to use two paths. These would include one "time sensitive" path for alarms where the objective is to have a short delay and a highly reliable path with no constraint on the type of nodes along the path to the destination, and another "not time sensitive path" for the telemetry traffic where it is beneficial to not traverse any battery-operated node to preserve energy and where the objective would be to minimize hops to avoid traffic congestion in the network. RPL addresses these requirements by building two DODAGs with each one having its own OF. The OF is used in conjunction with the routing metric to compute the path.

Consider Figure 17.2 which depicts an LLN. In this network, the link's LQLs are provided in addition to the latency and availability of link layer encryption. In addition, node 11 is battery-operated. The arrow shows the best computed path from the low-power and lossy network border router (LBR) to node 34 for two different OFs, OF1 and OF2, defined in the following:

OF1: "Use the LQL as a global recorded metric and favor paths with the minimum number of low and fair quality links, use the link color as a link constraint to avoid non-encrypted links." Note that two paths are available with an equivalent aggregated LQL metric: 34-35-24-13-1 and 34-33-23-22-12-1. But because the OF specifies using a recorded metric, the path 34-33-23-22-12-1 is chosen since it only has two links of "fair" quality.

OF2: "Find the best path in terms of latency (link latency is used as a global aggregated metric), while avoiding poor quality links and battery-operated nodes." Several paths have been pruned because they traverse battery-operated nodes (node 11) and traverse poor quality links (link 12-23). The best path (lowest latency) is 34-24-13-1.

17.6 RPL: THE NEW ROUTING PROTOCOL FOR SMART OBJECT NETWORKS

This section describes RPL (IPv6 Routing Protocol for Low-power and Lossy Networks), the newly specified IP routing protocol for smart object networks, in detail. RPL is still a work in progress and the IETF RFC should be used as the final reference. Various aspects may change or be added to the specification.

17.6.1 Protocol Overview

Similar to IETF specifications (see [214]), this section provides an overview of the RPL mode of operation.

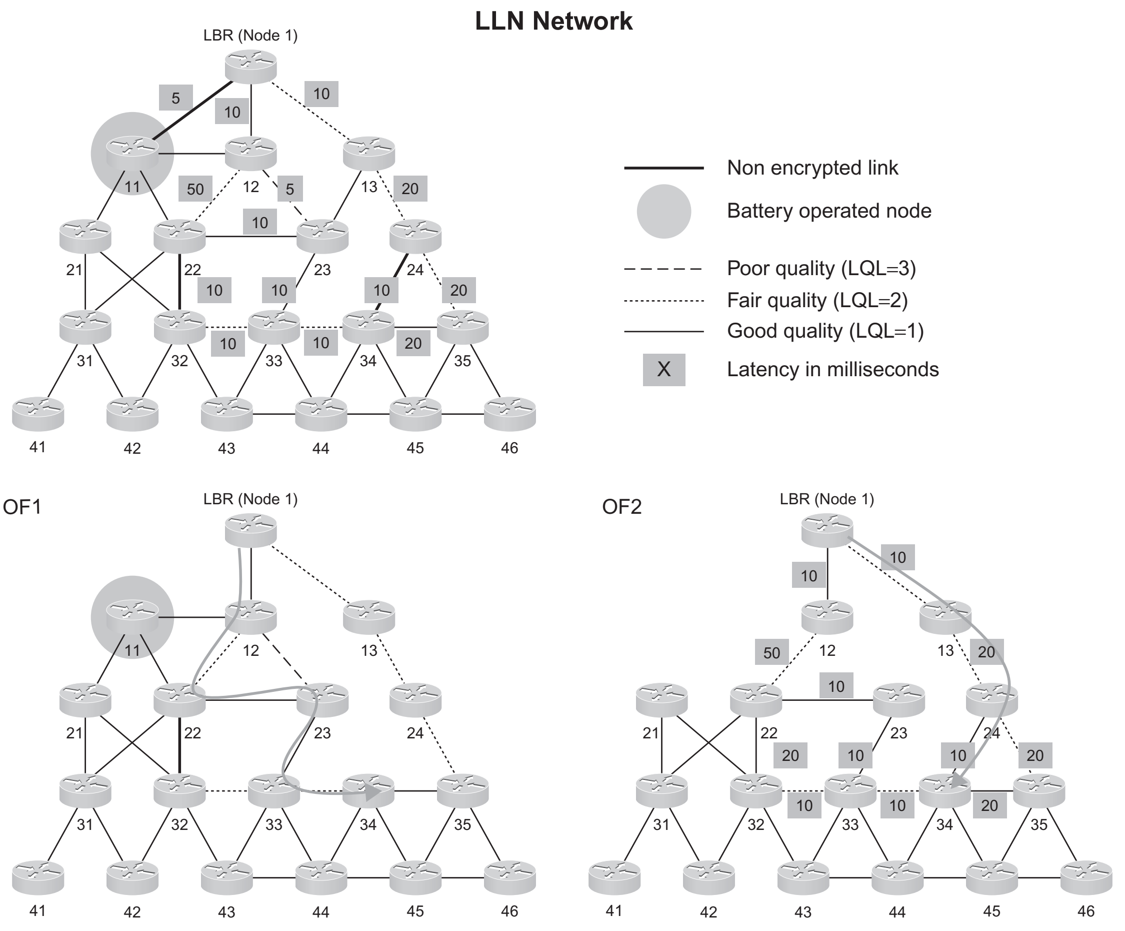


FIGURE 17.2

Examples with two different OFs.

Considering the wide set of routing requirements spelled out in the application-specific documents and discussed in Section 17.3, RPL was designed to be highly modular. The main specification [256] covers the intersection of these requirements. The prime objective is to design a highly modular protocol where the core of the routing protocol would address the intersection of the application-specific routing requirements, and additional modules would be added as needed to address specific requirements.

RPL was designed for LLNs where constrained devices are interconnected by (wireless and wired) lossy links. Many of the routing protocol design decisions were strictly driven by the unique characteristics of these networks. When observing the link failure profiles of the link layers in the Internet or private IP networks (Ethernet, Optical links, etc.), error rates are relatively low and the link error profiles show uniform distribution. Thus routing protocols designed for such link profiles quickly react to link failure with no risk of oscillation since link flaps are rare events. When failures do occur, various dampening techniques are used. This drove the design principles of various "fast reroute" mechanisms. As soon as the link failure is detected (thanks to link layer notification or fast keepalive mechanisms such as Bidirectional Forwarding Detection; BFD [144] or link layer triggers), the traffic is immediately rerouted onto a backup path to minimize the traffic disruption. The situation in LLN is rather different. Figure 17.1 shows the packet delivery ratio (PDR) for two wireless links and the situation is extremely similar to PLC links. Such link failure profiles are not uncommon and demonstrate that it is imperative to handle link failure in a very different manner in LLN. First, a node should try to determine whether or not the link should be considered as down (not an easy decision in LLNs) and, consequently, inadequate for traffic forwarding. The same reasoning applies to determining whether or not a link should be considered as usable in the first place (known as "local confidence"). This means that a node should carefully observe a link and start using it or determine whether to stop using it (thus triggering a global path recomputation in the network).

The lossy nature of these links is not the only LLN characteristic that drove the design decisions of RPL. Because resources are scarce, the control traffic must be as tightly bounded as possible. In these networks the data traffic is usually limited and the control traffic should be reduced whenever possible to save bandwidth *and* energy. Using a fast probing mechanism as with many other routing protocols is just not an option, and ideally the control traffic should decrease as the routing topology stabilizes. Nodes are constrained in nature, which implies that the routing protocol should not require heavy state maintenance.

Bearing in mind the lossy nature of links in LLN helps understand the RPL design choices made during the specification design.

RPL is a distance vector protocol that builds a DODAG where paths are constructed from each node in the network to the DODAG root (typically a sink or an LBR). There are a number of reasons why it was decided to use a distance vector routing protocol as opposed to a link state protocol. The main reason was the constrained nature of the nodes in LLNs. Link state routing protocols are more powerful (the detailed topology is known by all nodes) but require a significant amount of resources such as memory (Link State Database; LSDB) and control traffic to synchronize the LSDBs. An example of DODAG is shown in Figure 17.2. Various procedures described in Section 17.6.2 govern how the DODAG is constructed and how nodes attach to each other according to an OF. In contrast with tree topologies, DODAGs offer redundant paths, which is a MUST requirement for LLNs. Thus if the topology permits, RPL may provision more than one path between a node and the DODAG root and even other nodes in the network.

Before digging into the protocol specification, a high-level overview of the protocol is in order. First, one or more nodes are configured as DODAG roots by the network administrator. A node discovery mechanism based on newly defined ICMPv6 messages is used by RPL to build the DODAG. RPL defines two new ICMPv6 messages called DODAG information object (DIO) messages and destination advertisement object (DAO) messages. DIO messages (simply referred to as DIO) are sent by nodes to advertise information about the DODAG, such as the DODAGID, the OF, DODAG rank (detailed in the next section), the DODAGSequenceNumber, along with other DODAG parameters such as a set of path metrics and constraints discussed in the previous section. When a node discovers multiple DODAG neighbors (that could become parents or sibling), it makes use of various rules to decide whether (and where) to join the DODAG. This allows the construction of the DODAG as nodes join. Once a node has joined a DODAG, it has a route toward the DODAG root (which may be a default route) in support of the MP2P traffic from the leaves to the DODAG root (in the up direction).

RPL uses “up” and “down” directions terminology. The up direction is from a leaf toward the DODAG root, whereas down refers to the opposite direction. The usual terminology of parents/children is used. RPL also introduces the “sibling”; two nodes are siblings if they have the same rank in the DODAG (note that they may or may not have a common parent). The parent of a node in the DODAG is the immediate successor within the DODAG in the up direction, whereas a DODAG sibling refers to a node at the same rank. Back to the example in Figure 17.2, 13 is a parent of 24, 22, and 23 are siblings, and 43 and 44 are children of 33. A DODAG is said to be grounded if it is connected to what RPL calls a “goal,” which can be a node connected to an external (non-LLN) private IP network or the public Internet. A non-grounded DODAG is called a floating DODAG.

RPL uses iterations controlled by the DODAG root to maintain the DODAG; the DODAGSequenceNumber is a counter incremented by the DODAG root to specify the iteration number of the DODAG.

A mechanism is now needed to provide routing information in the down direction (for the traffic from the route to the leaf) and for the P2P direction since the DODAG provides defaults routes to the DODAG root from each node in the network. For this mechanism, RPL has defined another ICMPv6 message called the DAO message. DAO messages (simply referred to as DAO) are used to advertise prefix reachability toward the leaves. DAOs carry prefix information along with a lifetime (to determine the freshness of the destination advertisement) and depth or path cost information to determine how far the destination is. Note that the path in this direction is dictated by the DODAG built by RPL in the other direction. In some cases DAOs may also record the set of visited nodes. This is particularly useful when the intermediate nodes cannot store any routing states, which is discussed later in Section 17.6.6. If a parent receives destination advertisements that can be aggregated from multiple children, local policy may be used to perform prefix aggregation in an attempt to reduce routing table and the size of DAO messages. Note that redundant DAO messages are aggregated along the DODAG. An OF may be specifically designed to maximize prefix aggregation.

What about P2P traffic? RPL supports P2P traffic. When node A sends a packet destined to node B, if B is not in direct reach, it forwards the packet to its DODAG parent. From there, if the destination is reachable from one of its children, the packet is forwarded in the down direction. In other words, the packet travels up to a common ancestor at which point it is forwarded in the down direction toward the destination. An interesting optimization periodically emits link-local multicast IPv6 DAOs. Thus if the destination is in direct range (one hop away), a node can send the packet directly to the destination without following the DODAG. The degree of optimality for P2P traffic is discussed in Section 17.6.10.

Sending DIO and DAO messages is governed by the use of trickle timers. The trickle timers use dynamic timers that govern the sending of RPL control messages in an attempt to reduce redundant messages as discussed in detail later in Section 17.6.10. When the DODAG is unstable (e.g., the DODAG is being rebuilt) RPL control messages are sent more frequently (the DODAG becomes inconsistent). On the other hand, as the DODAG stabilizes messages are sent less often to reduce the control plane overhead, which is very important in LLNs.

Once the DODAG is built and routing tables are populated, routing is fully operational. As links and nodes fail, paths are repaired using local and global repair mechanisms. Local repairs quickly find a backup path without an attempt to globally reoptimize the DODAG entirely, whereas global repairs rely on a reoptimization process driven by the DODAG root.

RPL also supports the concept of DODAG instances identified by an Instance ID called the RPLInstanceID. It might be useful to form different topologies according to various sets of constraints and OFs. An RPL node may join multiple DODAG instances; for example, one DODAG optimizes for high reliability and another DODAG optimizes for low latency. Data packets are then forwarded along the appropriate DODAG according to the application requirements.

17.6.2 Use of Multiple DODAG and the Concept of RPL Instance

As previously discussed, a DODAG is a set of vertices connected by directed edges with no directed cycles. As shown in Figure 17.2, RPL builds DODAGs forming a set of paths from each leaf to the DODAG root (typically an LBR). In contrast with tree topologies, DODAGs offer redundant paths, a MUST requirement for LLNs. Thus if the topology permits, there is always more than one path between a leaf and the DODAG root.

The notion of DODAG instance is quite straightforward and similar to the concept of multi-topology routing (MTR) supported by other routing protocols such as OSPF and IS-IS. The idea is to support the construction of multiple DODAGs over a given physical topology. Why more than one DODAG? This is done to steer traffic to different paths optimized according to the requirements. Consider the case of a physical network made of a series of links with different qualities (e.g., reliability, throughput, latency) and nodes with different attributes (e.g., battery-powered vs. main-powered). If the network carries traffic with different QoS requirements, it might be useful to build one DODAG optimized for low latency and another DODAG optimized to provide high reliability while avoiding battery-operated nodes. In this case, RPL can build two DODAGs according to two different OFs. *If a node carries both types of traffic it may then decide to join both DODAGs (DODAG instance).* When a delay-sensitive packet must be sent along the DODAG, it is flagged (in the packet header) with the appropriate DODAG instance and forwarded along the appropriate DODAG. This decision is made by the application.

Figure 17.3 shows how two DODAGs are built from a given physical topology. DODAG 1 (instance 1) is built to optimize the path reliability while avoiding battery-operated nodes, whereas DODAG 2 (instance 2) is optimizing the latency. Depending on the sequence event, RPL may not compute the exact same routing topology. Also note that only preferred parents are depicted on the picture along with siblings.

A destination-oriented DODAG (DODODAG) is a DODAG rooted at a single destination. Within an instance, the LLN routing topology can be partitioned among multiple DODAGs for a number of reasons such as providing a greater scalability. Figure 17.4 shows multiple DODAGs in a specific DODAG instance.

A node can only join a single DODODAG within a DODAG instance.

A DODAG is identified by its instance (RPLInstanceID). A DODAG is uniquely identified by the combination of the DODAG instances (RPLInstanceID) and the DODAGID (the identifier of the DODAG that must be unique within the scope of a DODAG instance in the LLN). A DODAG iteration is uniquely identified by the tuple {RPLInstanceID, DODAGID, DODAGSequenceNumber}.

17.6.3 RPL Messages

A good way to gain further insight into a protocol after a protocol overview is to look at the protocol packet formats. RPL specifies three messages (using the same ICMPv6 codepoint): the DODAG

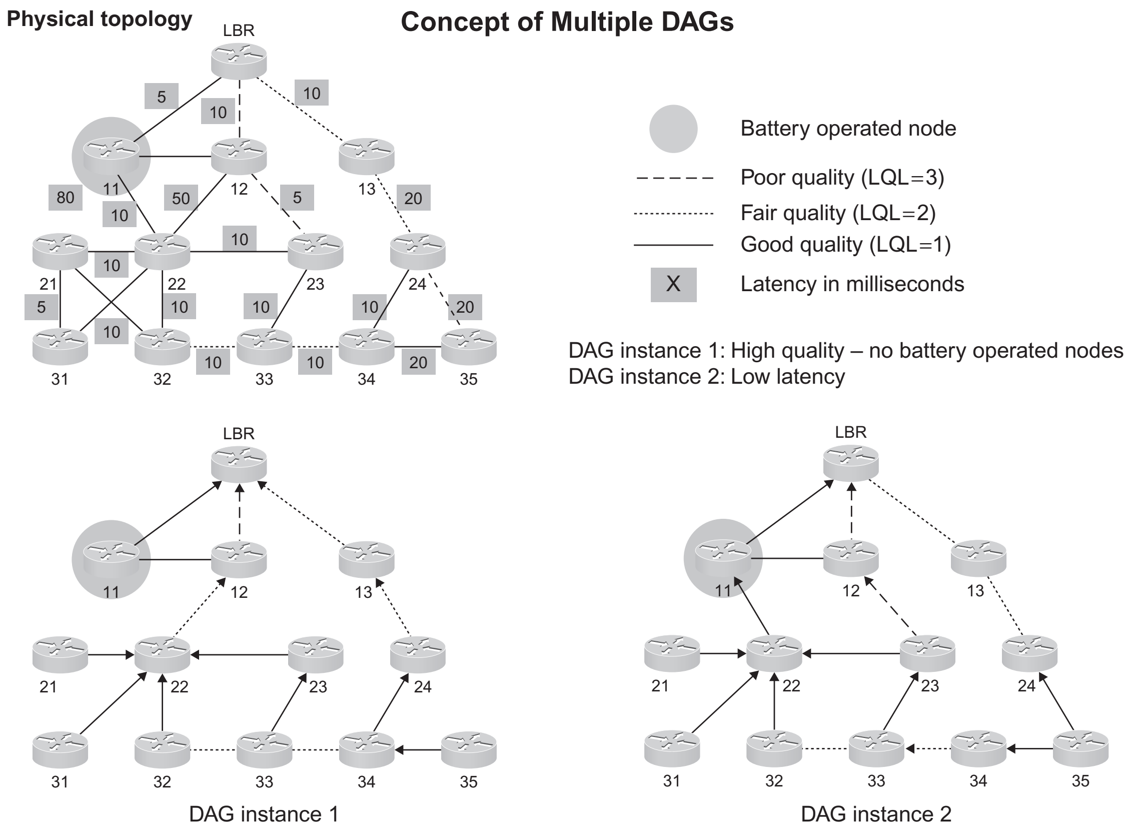


FIGURE 17.3

RPL DODODAG and instance.

Information Object (DIO), the DODAG Destination Advertisement Object (DAO), and the DODAG information solicitation message (DIS).

17.6.3.1 DIO Messages

DIO messages are sent by RPL nodes to advertise a DODAG and its characteristics, thus DIOs are used for DODAG discovery, formation, and maintenance. DIOs carry a set of mandatory information augmented with options.

The DIO base option is mandatory and may carry several suboptions. The following flags and fields are currently defined:

- Grounded (G): Indicates whether the DODAG is grounded, in other words, the DODAG root is a goal for the OF (e.g., the DODAG root is connected to a non-LLN IP network such as a private network or the public Internet).
- Destination Advertisement Trigger (T): The T bit is used to trigger a complete refresh of the routes in the down direction (downward routes).

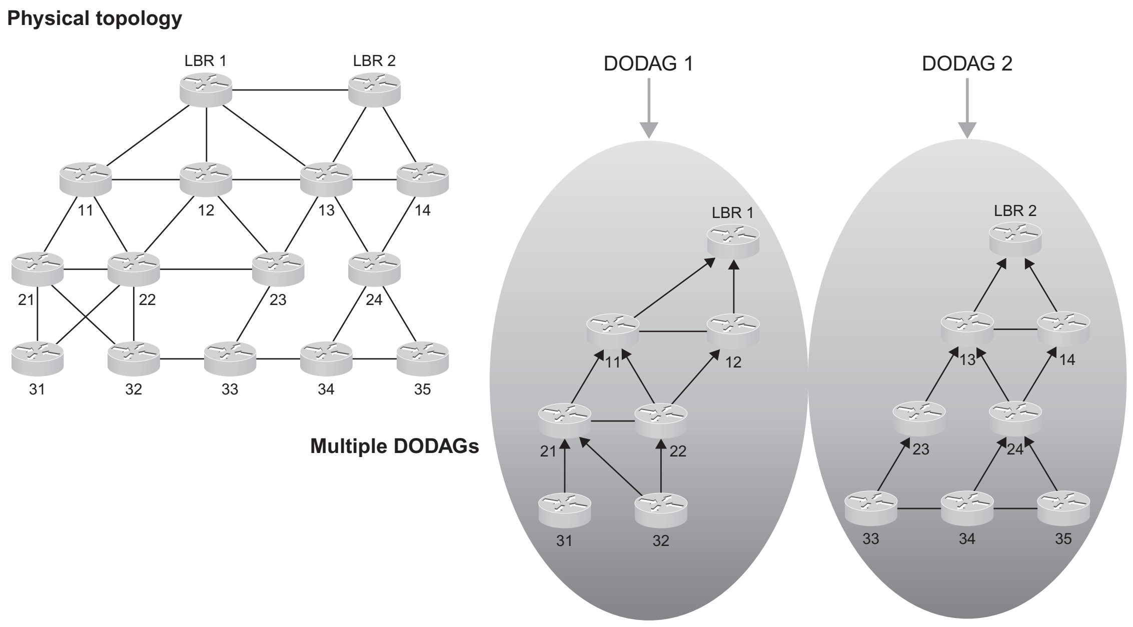


FIGURE 17.4

RPL multiple DODAGs within a DODAG instance.

- Destination Advertisement Stored (S): The S bit is used to indicate that a non-root ancestor is storing routing table entries learned from DAO messages.
- Destination advertisement supported (A flag): The A flag is set when the DODAG root supports the collection of prefix advertisements and enables the advertisement of prefixes in the DODAG.
- DODAGPreference (Prf): The Prf is a 3-bit field set by the DODAG root to report its preference. It can be used to engineer the network and make some DODAGs more attractive to join.

The DODAGSequenceNumber is the sequence number of the DODAG that characterizes the DODAG iteration and is exclusively controlled by the DODAG root.

The RPLInstanceID is used to identify the DODAG instance and is provisioned at the DODAG root.

The Destination Advertisement Trigger Sequence Number (DTSN) is an 8-bit integer set by the node sending the DIO. The DTSN is used by the procedure to maintain the downward routes as discussed in Section 17.6.6.

The DODAGID is a 128-bit integer set by the DODAG root and that uniquely identifies the DODAG.

The DODAG Rank is the rank of the node sending the DIO message.

The rank determines the relative position of a node in the DODAG and is used primarily for loop avoidance. The rank is computed according to the OF and is potentially subject to local node policy. The rank (although potentially derived from routing metrics) is not a metric. For example, a node that first joins a DODAG may not select the node with the lowest rank as a parent (closer to the DODAG root) should there be an alternate node with a deeper rank advertising a path with a lower cost. Once the rank has been computed, the node cannot join a new parent with deeper rank for loop avoidance except under specific circumstances discussed in Section 17.6.7.

When two nodes have the same rank, the nodes are said to be siblings (they are located at a similar level of optimality in the DODAG). It is highly desirable to make the rank a coarse value to favor the use of siblings. A sibling is a node that has the same rank and is used to increase connectivity. If the OF chooses to use the path ETX as the rank, more than likely the nodes will all have a different rank and thus the probability of finding a sibling will be very low. A rounded ETX (a coarse-grained value derived from the ETX) helps to increase the probability of finding siblings. A node may forward a packet to one of its siblings, if the link to its most preferred parent is not viable, at the risk of forming a loop (loop detection mechanisms can then be used to detect such a loop).

17.6.3.1.1 Use of the Rank for DODAG Parent Selection

If Rank (A) $<$ Rank (B), then node A is located in a more optimal location than node B and it is safe for B to select A as a DODAG parent with no risk of forming loops.

On the other hand, if Rank(A) $>$ Rank(B), it is not safe for B to select A as a parent (unless B joins the DODAG for the first time) since A may be in B's sub-DODAG. Selecting A as a parent would potentially form a routing loop. This may be allowed in a limited manner according to the max_depth rule explained in Section 17.6.7 to allow for local repair.

Note that the rank is a monotonic scalar. The rank of a node is always higher than the rank of any of its parents.

The rank is a 16-bit value used for the number of purposes described in detail in this chapter. At the time of writing, [256] suggests to consider the rank as a fixed point number, where the position of the decimal point is determined by value advertised by the DODAG root called the MinHopRankIncrease. The MinHopRankIncrease represents the minimum amount that a rank can increase on each hop and is used to detect siblings. The integer portion of the rank is called floor (Rank/MinHopRankIncrease).

A node A has a rank less than the rank of a node B if floor (Rank(A)/MinHopRankIncrease) is less than floor (Rank(B)/MinHopRankIncrease).

A node A has a rank greater than the rank of a node B if floor (Rank(A)/MinHopRankIncrease) is greater than floor (Rank(B)/MinHopRankIncrease).

Two nodes A and B are siblings if: floor (Rank(A)/MinHopRankIncrease) == floor (Rank(B)/MinHopRankIncrease). In other words, A and B are siblings if the integer portion of their rank is equal.

This can be better illustrated with an example. If MinHopRankIncrease is equal to, say, $2^5 = 32$ and the rank is equal to 953, then the integer portion of the rank is equal to int(953/32) $=$ 29. All the nodes with a rank between 928 and 959 will have the same integer part for their rank, so they will be siblings.

Note that this may still change but this would not affect how the notion of rank is used in [256].

The DODAGID is a 128-bit integer that uniquely identifies the DODAG and is set by the DODAG root. If the DODAG root uses an IPv6 address, the same IPv6 address must not be used by any other uncoordinated DODAG root within the LLN for the same DODAG instance.

Several suboptions are defined for DIOs. One of the most important is the DODAG metric container suboption used to report the path metrics described in the previous section.

A second important suboption is the destination prefix suboption used for prefix advertisement in the down direction (thus to provision state to route a packet in the up direction) for prefixes other than the default route. This may be useful to advertise prefixes other than the default route. The prefix is accompanied by a preference field compliant with [59] and a prefix lifetime.

The third important suboption is the DODAG configuration suboption used to advertise several DODAG configuration parameters such as trickle timers. Sending of RPL messages is governed by trickle timers and a detailed description of the trickle algorithm can be found later in Section 17.6.10. To ensure consistency across the DODAG, the trickle timer's configuration is advertised by the DODAG root. Since these timers are unlikely to change in the DODAG, a node may decide not to include the DODAG timer suboption in every DIO, except if the DIO is sent in reply to a DIS. The three parameters advertised in the DODAG timer configuration suboption include DIOIntervalDoubling, DIOIntervalMin, and DIORedundancyConstant. These are discussed in detail in Section 17.6.9. Other DODAG parameters such as the DAGMaxRankIncrease used by the local repair mechanism (specified in Section 17.6.7) and the MinHopRankIncrease are also advertised. Other parameters are likely to be added in further revisions of RPL to support additional features.

17.6.3.2 DAO Messages

DAO messages are used to propagate destination information along the DODAG in the up direction to populate the routing tables of ancestor nodes in support of P2MP and P2P traffic. The DAO message includes the following information:

- DAO sequence: A counter incremented by the node owning the advertised prefix each time a new DAO message is sent.
- RPLInstanceID: The topology instance ID as learned from the DIO.
- DAO rank: Corresponds to the rank of the node that owns the prefix.
- DAO lifetime: It is expressed in seconds and corresponds to the prefix lifetime.
- Route tag: 8-bit integer that can be used to tag "critical" routes. The priority could be used to indicate whether the route should be stored by the nodes with a lower rank (closer to the DODAG root), which could be useful if nodes have limited memory capacities and must be selective about which destination information to cache. Note that the size of that field has been changed several times and is subject to further changes.
- Destination Prefix: The Prefix Length field contains the number of valid leading bits in the prefix.
- Reverse Route Stack: The RRS is discussed in detail in Section 17.6.6, and contains a number of RRCount (another field of the DAO message) IPv6 addresses used in LLNs with nodes that cannot store routing tables.

17.6.3.3 DIS Messages

DIS messages are similar to the IPv6 router solicitation (RS) message, and used to discover DODAGs in the neighborhood and solicit DIOs from RPL nodes in the neighborhood. A DIS has no additional message body.

17.6.4 RPL DODAG Building Process

In this section, the DODAG building mode of operation for RPL is discussed. The DODAG formation is governed by several rules: the RPL rules used for loop avoidance (based on the DODAG ranks), the advertised OF, the advertised path metrics, and the policies of the configured nodes. A node may be part of several DODAG instances, and within a DODAG instance there may be several DODAGs rooted by different nodes.

DIO messages are sent upon the expiration of the trickle timer (see Section 17.6.10 for more details). The basic idea is to send DIOs more frequently when a DODAG inconsistency is detected (e.g., when the node receives a modified DIO with new DODAG parameters such as a new OF, new DODAGSequenceNumber, or the parent advertises a new DODAG Rank, etc.), a loop is detected (e.g., the node receives a packet from a child that is intended to move down along the same child according to its routing table), or the node joins a DODAG with a new DODAGID or has moved within a DODAG. When a DODAG inconsistency is detected the node resets its trickle timer to cause the advertisement of DIO messages more often. As the DODAG stabilizes and no inconsistency is detected, DIO messages are sent less frequently to limit the control traffic.

When a node starts its initialization process it may decide to remain silent until it hears a DIO advertising an existing DODAG. Alternatively, the node may issue a DIS message to probe the neighborhood and receive DIO messages from its neighbors more quickly. Another option is to start its own floating DODAG and to begin multicasting DIO messages for its own floating DODAG (note that this may be desired if it is required to establish and maintain inner connectivity between a set of nodes in the absence of a goal/grounded DODAG). Unicast DIOs are sent in reply to unicast DIS messages and also include a complete set of DODAG configuration options.

The G-bit is only set if the DODAG root is a goal. If the advertising node is the DODAG root, the rank is equal to the RPL variable called the ROOT_RANK (equal to 1).

Upon receiving a DIO message, a node must first determine whether or not the DIO message should be processed. If the DIO message is malformed, it is silently discarded. If not, the node must then determine whether the DIO was sent by a candidate neighbor. The notion of a candidate neighbor is tightly coupled with the notion of local confidence, and that important notion is implementation specific and used to determine if a node is eligible for parent selection. For example, when a node first hears about a neighbor it may choose to wait for a period of time to make sure that the connecting link is sufficiently reliable.

Then the node determines whether the DIO is related to a DODAG it is already a member of.

If the rank of the node advertising the DIO is less than the node's rank plus some RPL configurable value called the DAGMaxRankIncrease, then the DIO is processed. This rule is called the max_depth rule and is explained in detail in Section 17.6.7.

If the DIO message is sent by a node with a lesser rank and the DIO message advertises a (different) DODAG that provides a better path according to the OF, then the DIO message must be processed.

The DIO must also be processed if it is originated by a DODAG parent for a different DODAG than the node belongs to since the DODAG parent may have jumped to another DODAG.

A collision may occur if two nodes simultaneously send DIOs to each other and decide to join each other. This is why DIO messages received during the risk window are simply not processed. Because of the random effect of the trickle timers, it is expected that the next DIO messages are not likely to collide again.

For the DODAG root operation on the DODAGSequenceNumber the DODAGSequenceNumber is only incremented by the DODAG root. It may be incremented upon the expiration of a configurable timer, upon a manual command on the DODAG root, or upon the reception of a signal from downstream (yet to be determined by the RPL specification). A node may safely attach to a parent regardless of the advertised rank if the parent in the next DODAG iteration (the DODAGSequenceNumber is higher than the node's current one) since that parent cannot possibly belong to the sub-DODAG of that node. This is further discussed in Section 17.6.8.

17.6.4.1 A Step-by-step Example

The DODAG building process is illustrated by Figure 17.5, which shows the physical network topology and how the DODAG is built. The link metric is the ETX and the OF finds the path minimizing the path ETX where the path ETX is defined as the sum of the ETX for all traversed links. The OF specifies an additional constraint of avoiding battery-operated nodes and the rank is based on the hop count. Note that the OF could have been different; for example, it could have been computed as a function of the ETX (e.g., Rank = int(ETX*10)/10).

Step 1: The DODAG root starts sending link-local multicast DIO messages. This is one possible event sequence. One of the nodes could also decide to send a DIS message, in which case the DODAG root (LBR) would immediately send the DIO in reply to the DIS message.

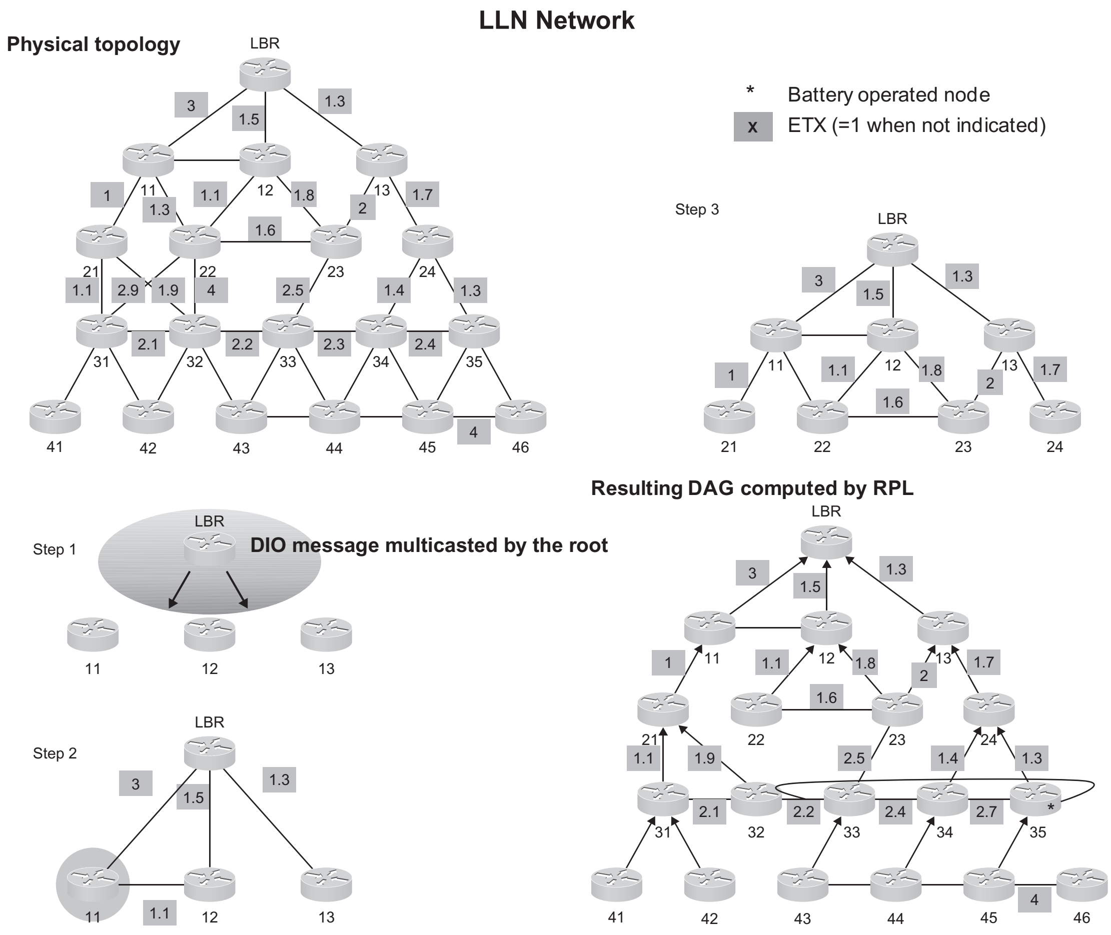

LLN Network
Physical topology
LBR
* Battery operated node
x ETX (=1 when not indicated)
Step 1
DIO message multicasted by the root
Step 2
Step 3
Resulting DAG computed by RPL

FIGURE 17.5

Example of DODAG formation.

Step 2: Nodes 11, 12, and 13 receive the LBR DIO. Upon processing the DIO (which comes from a lower ranked node thus a lower rank value), nodes 11, 12, and 13 select LBR as their DODAG parent (note that nodes 12 and 13 may have waited for some period of time to build enough local confidence). At this point, nodes 11, 12, and 13 compute their new rank based on the hop count and the path ETX value is computed. Node 11 also selects node 12 as a sibling and vice versa (same rank).
Step 3: Shows the resulting DODAG after another round of iteration. Note that link 22-11 has been pruned from the DODAG since node 12 is a better parent considering the OF (minimize the path ETX). Node 23 has selected two parents offering equal cost paths (ETX = 3.3).
Step 4: Shows the final DODAG. Node 46 has not selected node 35 as a best parent since the OF specifies the constraint of not traversing a battery-operated node. Local policy may be used to indicate whether constraints also apply to siblings (in this example, node 34 did select node 35 as a sibling). A potential sibling loop 33-34-35-33 has formed (discussed in Section 17.6.7).

The shape of the resulting DODAG depends on the event sequence ordering.

17.6.5 Movements of a Node Within and Between DODAGs

There are a few fundamental rules that govern movements within a DODAG:

1. A node is free to jump to any position in any other DODODAG that has not been previously visited at any time. For example, a node may decide to select a new node as a parent that belongs to a new DODAG regardless of the rank. The new DODAG may be the same DODAG (same DODAGID, same RPLInstanceID) but with a higher DAGSequenceNumber or it may be a different DODAG (different DODAGID and/or different RPLInstanceID). It is recommended to jump to another DODAG only when all queued packets have been transmitted along the previous DODAG. Jumping back to a previous DODAG is similar to moving inside a DODAG. This is why a node should remember its DODAG identified by the RPLInstanceID, DODAGID, and DODAGSequenceNumber along with its rank within that DODAG. Jumping (moving) back should then honor the rules of the previous position so as not to potentially create a loop (max_depth rule).
2. A node may advertise a lower rank at any time when it has jumped to another DODAG.
3. Within a DODODAG iteration a node must not advertise a rank deeper than L+DAGMaxRankIncrease where L is the lowest rank. The DAGMaxRankIncrease is an RPL variable advertised by the DODAG root, and a value of 0 has the effect of disabling this rule. There is one exception to this rule; the poison-and-wait rule where the node advertises an infinite rank that is described right after. The reasons for this rule are further discussed in Section 17.6.7.

When a node prepares to move to a new DODAG iteration it may decide to defer the movement to see if it could join another node with a better path (even if the rank is higher) cost according to the OF.

It is perfectly safe for a node to move up in the DODAG and select new parents with a lower rank than its current parents' rank. In this case, the node must abandon all prior parents and siblings that have now become deeper than the node in the DODAG and potentially select new ones.

If a node wants to move down in its DODODAG causing the rank to increase, it may use the poison-and-wait rule discussed in Section 17.6.7.

What if a node receives a DIO message specifying an OF that it does not support or recognize? The two options are either not to join the DODAG or to join as a leaf. Such a node may not join as a router since the node would then be incapable of propagating an appropriate metric, which may lead to a DODAG using an inconsistent metric. Thus when a node joins as a leaf node, it can receive and process DIO messages and send DAO messages. But it should not send DIO messages and thus cannot act as an RPL router.

17.6.6 Populating the Routing Tables Along the DODAG Using DAO Messages

As the DODAG is being built, the next task is populating the routing tables along the DODAG in support of the down traffic (toward the leaves). DAO messages are used to propagate prefix reachability along the DODAG.

DAO operation is still being discussed within the IETF ROLL Working Group. More than likely several changes will take place and the mechanisms described in this section reflect the DAO mode of operation at the time of writing: the reference should be the final RFC for RPL.

A sequence number is included to detect the freshness of the information and outdated or duplicate messages are simply discarded. The sequence number is incremented by the node that owns the prefix. A node sends unicast DAO to its preferred parent only (note that this is the option taken by RPL at the time of writing; further revisions of RPL may suggest sending the DAO messages to a set of parents, which would require extensions to the DAO message propagation rules). Allowing for sending DAO messages to more than one parent would enable load balancing in the down direction of the DODAG.

The DAO message contains the rank of the node owning the advertised prefix. That rank may be used by a node who received multiple DAO from different children for the same destination prefix as a selection criteria to select the next-hop that provides the more optimal route, although the rank may not reflect the actual path cost to the advertising node. RPL also supports the inclusion of the DAG Metric Container in DAO messages to provide the path cost.

Note that RPL supports the ability to prune a route by sending a prefix with a lifetime set to 0. This is also called a "no-DAO" message.

17.6.6.1 Use of the Reverse Route Stack in DAO Message

Some nodes in the network may have significant constraints regarding memory and may be incapable of storing routing entries for downward routes. Although not an issue in support of the MP2P traffic, such nodes cannot store routing states upon receiving DAO messages from their children and, consequently, the P2MP traffic or P2P traffic cannot be routed to the destination leaf. Thus RPL has specified extensions to accommodate this type of node (also called non-storing nodes) in LLNs. The mechanism records paths traversing memory-less nodes when forwarding the DAO. Let's consider Figure 17.6 where nodes 22 and 32 cannot store any routing updates. P1 and P2 are two IPv6 prefixes owned by nodes 42 and 43, respectively, and advertised to node 32 by means of DAO. Upon receiving the unicast DAO message, node 32 appends the IPv6 prefix of node 42 to the reverse route stack of the received DAO. Upon receiving the DAO from node 32, node 22 (which is also memory-less) performs a similar operation and appends the IPv6 address of node 32. Each time, the RRCount counter is incremented. Once the DAO message reaches a node capable of storing routing states (node 12), the node detects that the DAO has traversed a region with nodes incapable of storing routing states by observing the presence of the reverse route stack in the DAO. Then node 12 simply extracts the set of

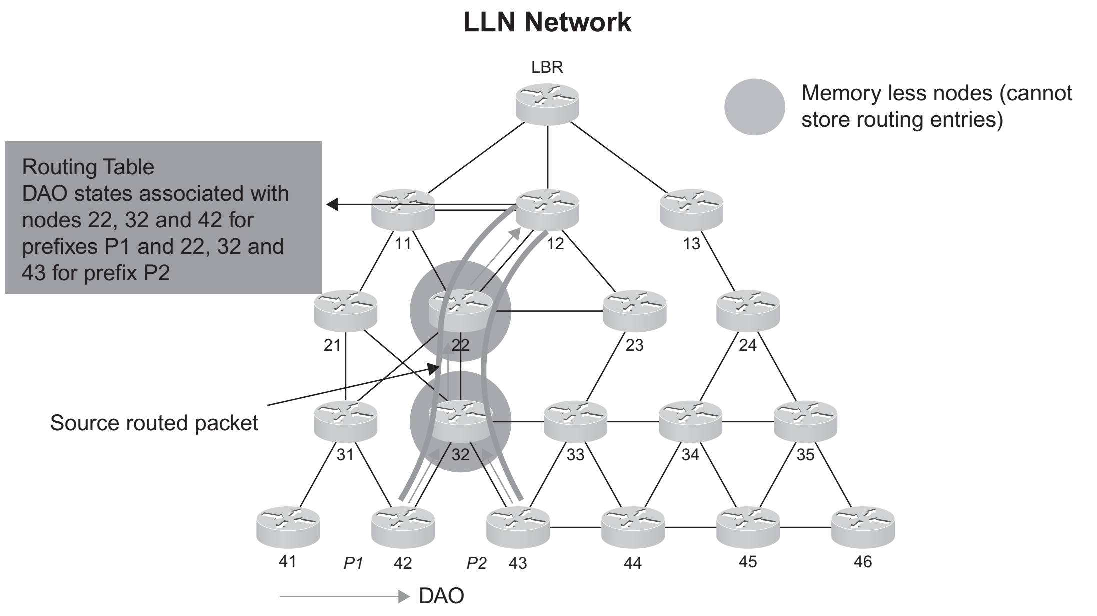

LLN Network
LBR
Memory less nodes (cannot store routing entries)
Routing Table
DAO states associated with nodes 22, 32 and 42 for prefixes P1 and 22, 32 and 43 for prefix P2
11
12
13
21
22
23
24
Source routed packet
31
32
33
34
35
41
P1
42
P2
43
44
45
46
DAO

FIGURE 17.6

Use of reverse route stack in DAO message.

hops associated with the advertised prefix, stores them locally in its routing table, and then clears the reverse stack header and the RRCount counter. Upon receiving a packet destined to, say, prefix P1, node 12 consults in the routing table and makes use of source routing to send the packet to node 42. This allows reaching the final destination with intermediate nodes incapable of storing states. This mechanism can be generalized to a network exclusively made of memory-less nodes thus leading to a situation where all node-to-node communication would transit via the DODAG root.

Thus DAO message can be used to propagate reachability information and also to record routes for regions comprising non-storing nodes. These two mechanisms could also be decoupled.

The source routing mechanisms used by RPL have not yet been defined. They could be based on IPv6 source routing, which would require a new extended header (potentially with compressed IPv6 addresses) or labels. Furthermore, the mechanism described here is subject to change and RPL may evolve to not allow for the mix of storing and non storing nodes in the same network in an attempt to simplify the specification.

17.6.6.2 Routing Table Maintenance

If a node loses routing adjacency with a child for which it has an associated prefix, it should clean up the corresponding routing entry and report the lost route to it parents by sending a no-DAO message for the corresponding entry.

Prefixes may be in three different states: (1) connected (prefix locally owned by the node), (2) reachable (prefix with a non-0 lifetime received from a child), and (3) unreachable (prefix that has timed out for which a no-DAO message will be sent to the parent the node had previously advertised that prefix to).

Two timers have been specified for the processing of DAO messages:

- DelayDAO timer: This timer is armed each time there is a trigger to send a new DAO message (e.g., reception of a DIO message that requests to receive new DAO messages). At the time of writing, the DelayDAO timer is set to a random value between [DEF_DAO-LATENCY/ Rank(Node)] and [DEF_DAO_LATENCY/Rank (parent's node)] for nodes deeper in the DODAG to advertise their prefixes first. By attempting to order the sequencing of DAO, the chances to aggregate prefixes along the DODAG in an attempt to reduce the number of DAO messages and routing table size increase.
- RemoveTimer: This timer is used to remove stale prefixes that are no longer advertised by nodes in the sub-DODAG. There is a mechanism that allows a node to request DAO to be sent to refresh the states. In the absence of replies after n requests, the timer is started and upon its expiration routes are removed in the absence of DAO advertising these routes. The node then also informs its own parent with a no-DAO.

One event that triggers the sending of a DAO message (or more precisely arming the DelayDAO timer) is the reception of a new DIO message from a parent.

All routes learned through DAO messages are removed if the corresponding interface or the routing adjacency for these prefixes is determined as down.

DAOs are sent as unicast messages to DODAG parents, but they can also be sent to the link-local scope all-nodes multicast address (FF02::1). In the case of multicast messages, the node only advertises its own local prefixes, and these prefixes can also be advertised by a node to its DODAG parent using a unicast DAO. A node is not allowed to advertise prefixes learned from one of its children using multicast DAO. The main purpose of multicast DAO is to help with the "one-hop" P2P traffic between two nodes that can communicate directly with each other even when the link does not belong to the DODAG.

As illustrated in Figure 17.7, a multicast DAO is received by the node from node 23 advertising prefix P1. Thus if a packet received or originated by node 32 is destined to prefix P1, it is sent directly to node 23 without having to follow the DODAG. In the absence of multicast DAO, such a packet would first be sent to the parent of node 32 (node 22), which would relay the packet to its parent (node 12). At this point, node 12 would have P1 in its routing table due to the DAO message received from its child, node 23. Thus the path would have been 32-22-12-23.

In its current form there is exactly one prefix per DAO message. But as prefixes travel along the DODAG, a node can factor out some of their common attributes. For example, prefixes advertised at the same rank could be packed in the same DAO message with a unique rank without needing to repeat the same rank for each prefix. The same reasoning applies to many other prefix attributes. Thus by packing prefixes into the same message and factoring out their common attributes, the control traffic overhead is reduced and wasting bandwidth is avoided. More than likely DAO packing will be added to the RPL specification.

17.6.7 Loop Avoidance and Loop Detection Mechanisms in RPL

Routing loops are always undesirable and one of the objectives of routing protocols is to avoid the formation of loops whenever possible.

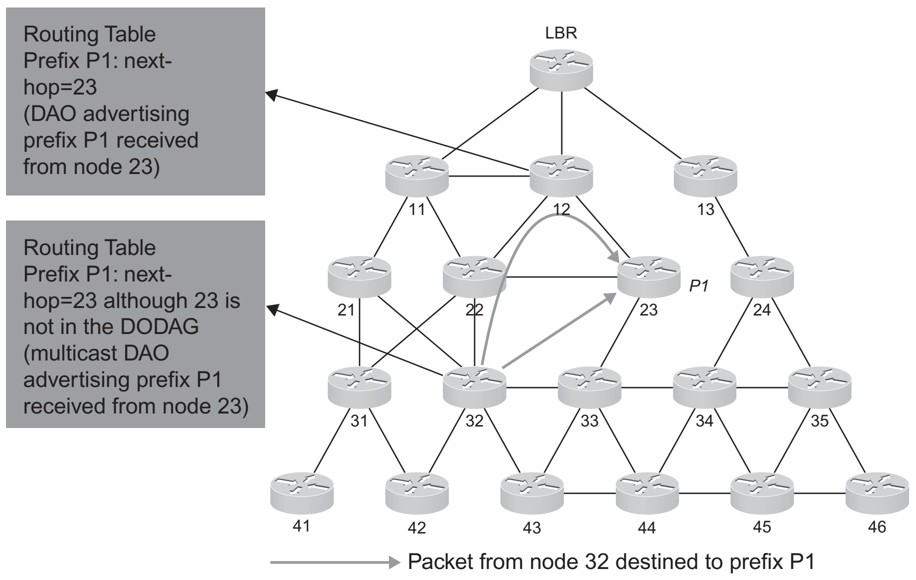

Routing Table
Prefix P1: next-hop=23
(DAO advertising prefix P1 received from node 23)
Routing Table
Prefix P1: next-hop=23 although 23 is not in the DODAG
(multicast DAO advertising prefix P1 received from node 23)
LBR
11
12
13
21
22
23
P1
24
31
32
33
34
35
41
42
43
44
45
46
Packet from node 32 destined to prefix P1

FIGURE 17.7

P2P routing in a DODAG with multicast DAO.

In high-speed networks, the packet TTL is decremented at each hop so a looping packet is quickly destroyed even if the loop has a short duration. Even with link state routing protocols such as OSPF and IS-IS, temporary loops (often called micro-loop due to their limited lifetime) may form during network topology changes due to the temporary lack of synchronization of the node's LSDB. At high data rates, even a short duration loop can lead to packet drops and link congestion. Various mechanisms have been proposed to avoid such loops.

In LLNs, the situation is somewhat different. First the traffic rate is generally very low, thus a temporary loop may have a very limited impact. Second, it is extremely important not to overreact in the presence of instability. In contrast with "traditional" IP networks where fast reaction (reconvergence) is very important, it is crucial not to react too quickly in LLNs. Thus loops may exist; they must be avoided whenever possible and detected when they occur. RPL does not fundamentally guarantee the absence of temporary loops, which would imply expensive mechanisms for the control plane and may not be appropriate to lossy and unstable environments. RPL instead tries to avoid loops by using a loop detection mechanism via data path validation.

17.6.7.1 Loop Avoidance

One of the RPL's rules, the max_depth rule, states that a node is not allowed to select as a parent a node with a rank higher than the node's rank+DAGMaxRankIncrease. Let's explain why this rule exists by considering the network depicted in Figure 17.5.

The first reason is simply to reduce the risk of a node attaching to another node that belongs to its own sub-DODAG, thus leading to a loop that may require counting to infinity. For example, in Figure 17.5 if node 24 loses all of its parents and decides to select node 46 as a parent, since the path to the root from node 46 is via node 24, a loop would form and node 24 has no way to learn that node 46 actually belongs to its own sub-DODAG. As explained in Section 17.6.8, the max_depth rule does not prevent loops from occurring, but it limits the loop sizes and allows the detection of such a loop without having to count to infinity.

Another RPL rule requires that the rank to increase the set of feasible parents should not be increased to avoid a "greediness" effect. Consider again Figure 17.5. Suppose that nodes 22 and 23 are both at rank 3, share a common parent (node 12), and there is a viable link between them (nodes 22 and 23 are siblings). Suppose that both nodes 22 and 23 try to increase their set of feasible successors to have alternate routes in case of link failure with their preferred DODAG parent (e.g., by detaching and moving down in a controlled manner). Suppose that node 22 first decides to select nodes 23 and 12 as DODAG parents (the new rank is now 4, the highest rank of both parents). Suppose now that node 23 does not follow the RPL rule and processes the DIO from node 22 (which now has a deeper rank than node 23). Node 23 may then decide to select both nodes 12 and 22 as DODAG parents, thus increasing its rank to 5. Then node 22 may reiterate the process until counting to infinity and restarting the process.

This explains two fundamental loop avoidance rules of RPL (except in specific conditions such as attempts to perform a local repair as explained next): (1) a first node is not allowed to select as a DODAG parent a neighboring node that is deeper in the DODAG than the first node's self rank+DAGMaxRankIncrease and (2) a node is not allowed to be greedy and attempt to move deeper in the DODAG to increase the selection of DODAG parents (possibly creating loops and instability). Indeed, suppose that node 23 is now allowed to, and for some reason (temporary better metric) decides to, select node 43 as a DODAG parent. This leads to a loop …

Still, even with the loop avoidance mechanisms stated earlier, loops may take place in a number of circumstances within a DODAG. DODAG loops can take place when a DIO message is lost (examples are given in Section 17.6.8), but these are not the only type of RPL loops. DAO loops may occur when a node fails to inform its parents that a destination is no longer reachable. If the DAO message is lost, the parent may keep the route to that destination in its routing table. If the child wants to send a packet to that destination, the parent would send it back to the child thus leading to a loop. One proposal is to use acknowledgments for DAO messages, which would dramatically reduce the risk of DAO loops. Another possible type of loop is a sibling loop. Consider again Figure 17.5. In case of multiple failures of links toward the root (e.g., links 35-24, 34-24, and 33-23), a packet sent by node 35 to the LBR may very well loop (35-34-33-35) since siblings are by definition at the same rank. If one link fails (e.g., 35-24) and node 35 reroutes a packet destined to the root to node 34, the packet will then be forwarded to the root by node 34 with no loop, but in a multi-failure scenario like the one described above a sibling loop may form. In most cases routing protocols may experience similar issues during multiple failures and do not even try to solve the problem.

How about loops between RPL DODAG instances? When a host sends a packet for a destination it also selects an RPL DODAG instance according to the path objectives. RPL states that once a packet is forwarded along an RPL instance (specified by the RPLInstanceID in its header), it should not be rerouted along another DODAG instance even if the corresponding DODAG is "broken," which is precisely to avoid such loops. RPL might be extended at some point to allow defaulting to a "wide" connectivity DODAG with minimal constraints to increase the chance of at least one valid path to the root, in which case, it will be necessary to specify a rule to avoid loops between DODAG instances.

17.6.7.2 RPL Loop Detection Mechanism

In the previous section we showed that routing loops are hardly avoidable, thus loop detection mechanisms must be available. The loop detection mechanism piggybacks routing control data in data packets by setting flags in the packet header (this is sometimes referred to as data path validation).

RPL: IPv6 Routing Protocol for Low Power and Lossy Networks

Tsvetko Tsvetkov
Betreuer: Alexander Klein
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS 2011
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: tsvetko.tsvetkov@gmail.com

ABSTRACT

Today, Low Power and Lossy Networks (LLNs) represent one of the most interesting research areas. They include Wireless Personal Area Networks (WPANs), low-power Power Line Communication (PLC) networks and Wireless Sensor Networks (WSNs). Such networks are often optimized to save energy, support traffic patterns different from the standard unicast communication, run routing protocols over link layers with restricted frame-sizes and many others [14].
This paper presents the IPv6 Routing Protocol for Low power and Lossy Networks (RPL) [19], which has been designed to overcome routing issues in LLNs. It implements measures to reduce energy consumption such as dynamic sending rate of control messages and addressing topology inconsistencies only when data packets have to be sent. The protocol makes use of IPv6 and supports not only traffic in the upward direction, but also traffic flowing from a gateway node to all other network participants.
This paper focuses on the employment of RPL in WSNs and gives a brief overview of the protocol's performance in two different testbeds.

Keywords

RPL, Sensor Network, Low-Power Network, Lossy Link, Routing, Data Collection, Data Dissemination

1. INTRODUCTION

Over the last years WSNs have become a very important and challenging research field. Such networks consist of spatially distributed autonomous devices which usually operate untethered and additionally have limited power resources. This limits all aspects of their construction, architecture and communication capabilities. Several studies such as [2] and [11] reveal the impact of wireless lossy links on the overall reliability, power efficiency and maximum achievable throughput. There are cases where a network can only achieve approximately the half of the throughput of the corresponding lossless network. Moreover, lossy links effect the power consumption due to packet retransmissions and broadcasting. Zhao and Govindan [20] have estimated the impact of such links and concluded that 50% to 80% of the communication energy is wasted in overcoming packet collisions and environmental effects in indoor and outdoor scenarios.
Such LLNs are additionally characterized by connections that are not restricted to two endpoints. Many scenarios may include Point-to-Multipoint (P2MP) or Multipoint-to-Point (MP2P) traffic patterns. Such networks are also known for their asymmetric link properties. The communication is realized by using a separate uplink and downlink. Because each unidirectional link provides only one way traffic, the bandwidths in the two directions may differ substantially, possibly by many orders of magnitude.
In order to meet these requirements and challenges, the Internet Engineering Task Force (IETF) ROLL Working Group designed a new routing protocol, called RPL [18]. The highest goal of RPL is to provide efficient routing paths for P2MP and MP2P traffic patters in LLNs. The protocol successfully supports the latest version of the Internet Protocol which results from the research made by different organizations.
The IP for Smart Objects (IPSO) Alliance has made a great effort to promote the use of IP for small devices [4]. It is the leading organization for defining the *Internet of Things* and supports the use of the layered IP architecture for small computers. The cooperation with the IETF organization further accelerates the adoption of IPv6 on LLNs. IETF has specified the IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) standard [12] which supports the idea of applying IPv6 even to the smallest machines. In this way, devices with limited hardware resources are able to participate in the Internet of Things. This standard also enables the use of standard web services without application gateways.
The rest of this paper is organized as follows. Section 2 gives an overview of RPL's basic features and describes the terminology of the protocol. Section 3 discusses topics such as topology construction and structure of the used control message. An introduction to RPL's loop avoidance and detection mechanisms is presented in Section 4. Section 5 gives information about the different routing metrics. Section 6 describes how the support of P2MP traffic is realized and Section 7 gives an overview of the protocol's performance. Finally, the paper is concluded in Section 8.

2. RPL DESIGN OVERVIEW

RPL is a distance vector routing protocol for LLNs that makes use of IPv6. Network devices running the protocol are connected in such a way that no cycles are present. For this purpose a Destination Oriented Directed Acyclic Graph (DODAG), which is routed at a *single* destination, is built. The RPL specification calls this specific node a DODAG root. The graph is constructed by the use of an Objective

Function (OF) which defines how the *routing metric* is computed. In other words, the OF specifies how routing constraints and other functions are taken into account during topology construction.
In some cases a network has to be optimized for different application scenarios and deployments. For example, a DODAG may be constructed in a way where the Expected Number of Transmissions (ETX) or where the current amount of battery power of a node is considered. For this reason, RPL allows building a logical routing topology over an existing physical infrastructure. It specifies the so-called RPL Instance which defines an OF for a set of one or more DODAGs.
The protocol tries to avoid routing loops by computing a node's position relative to other nodes with respect to the DODAG root. This position is called a *Rank* and increases if nodes move away from the root and decreases when nodes move in the other direction, respectively. The Rank may be equal to a simple hop-count distance, may be calculated as a function of the routing metric or it may be calculated with respect to other constraints.
The RPL specification defines four types of control messages for topology maintenance and information exchange. The first one is called DODAG Information Object (DIO) and is the main source of routing control information. It may store information like the current Rank of a node, the current RPL Instance, the IPv6 address of the root, etc. The second one is called a Destination Advertisement Object (DAO). It enables the support of down traffic and is used to propagate destination information upwards along the DODAG. The third one is named DODAG Information Solicitation (DIS) and makes it possible for a node to require DIO messages from a reachable neighbor. The fourth type is a DAO-ACK and is sent by a DAO recipient in response to a DAO message. The RPL specification defines all four types of control messages as ICMPv6 information messages with a requested type of 155. This new type has been officially confirmed by IANA [6]. Note that the last two are not further described in this paper.
Another important fact about the protocol's design is the maintenance of the topology. Since most of devices in a LLN are typically battery powered, it is crucial to limit the amount of sent control messages over the network. Many routing protocols broadcast control packets at a fixed time interval which causes energy to be wasted when the network is in a stable condition. Thus, RPL adapts the sending rate of DIO messages by extending the Trickle algorithm [10]. In a network with stable links the control messages will be rare whereas an environment in which the topology changes frequently will cause RPL to send control information more often.

3. UPWARD ROUTING

Upward routing is a standard procedure which enables network devices to send data (e.g. temperature measurements) to a common data sink, also called sometimes a gateway or root node. In a typical WSN scenario, nodes periodically generate data packets (e.g. each minute) which have to find their way through the network. In this section, the RPL topology construction process is discussed and the structure of a DIO message is presented.

3.1 DIO Message Structure

As previously mentioned, a DIO message is the main source of information which is needed during topology construction. Figure 1 represents the structure of the message.

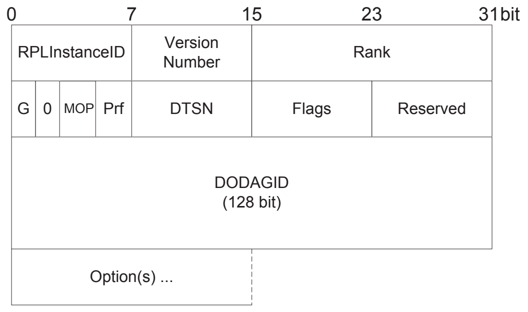


Figure 1: DIO Message Structure

A DIO first allows a node to discover the RPL Instance by storing the corresponding one in the first data field. The second and the third field include the DODAG Version and the Rank of the sender of the message. Section 3.2 describes the purpose of the RPL Instance, version number and the Rank update process. The next byte includes the 'G' flag which defines whether a DODAG is grounded. Grounded means that it can satisfy an application-defined goal. If it is not set, the DODAG is said to be floating. This may happen when a DODAG is disconnected from the rest of the network and supports only connectivity to its nodes. The MOP field (size of 3 bits) is set by the DODAG root and defines the used mode of operation for downward routing. Section 6 gives more information about it. The Prf field (size of 3 bits) defines how preferable the root node is compared to other root nodes. Such a node is identified by the DODAGID field. The last used field is DTSN and it is needed for saving a sequence number. Such a number is maintained by the node issuing the DIO message and guarantees the freshness of the message.
A DIO message may be extended by the use of options. In this paper, only the DODAG Configuration option is discussed, since it plays a crucial role for parameter exchange. Figure 2 outlines its structure.

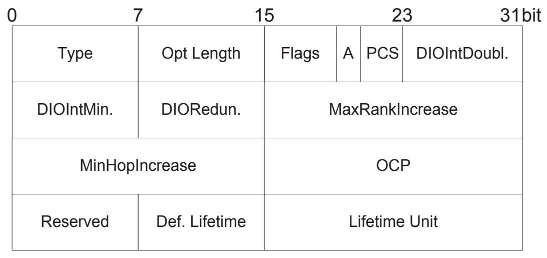


Figure 2: DODAG Configuration Option

The first two bytes always include the type (`0x04`) and the option's length (14 bytes). The next byte includes the 'A' flag, the Flags field and the PCS field. They will be not further discussed since they are not important for this paper. The next two bytes define the maximum timer value τ_{max} and the minimum timer value τ_{min} needed for the trickle timer setup. More information can be found in Section 3.2. The DIORedun. field defines a constant k used for suppressing DIO messages. If a node receives more than k DIOs, it may suppress them. The MaxRankIncrease field defines an upper limit for the Rank and is further discussed in Section 4. The MinHopIncrease field stores the minimum increase of the Rank between a node and any of its parent nodes. It creates a trade-off between the maximum number of hops and the hop cost precision. The DODAG Configuration Option concludes with the OCP field (OF identification), the Default Lifetime for all routes and the Lifetime Unit. The latter one defines in seconds the length of a time unit.

3.2 Constructing Topologies

In general, there are three[1] types of nodes in a RPL network. The first type are root nodes which are commonly referred in literature as gateway nodes that provide connectivity to another network. The second type are routers. Such nodes may advertise topology information to their neighbors. The third type are leafs that do not send any DIO messages and have only the ability to join an existing DODAG.

The construction of the topology starts at a root node that begins to send DIO messages. Each node that receives the message runs an algorithm to choose an appropriate parent. The choice is based on the used metric and constraints defined by the OF. Afterwards each of them computes its own Rank and in case a node is a router, it updates the Rank in the DIO message and sends it to all neighboring peers. Those nodes repeat the same steps and the process terminates when a DIO message hits a leaf or when no more nodes are left in range. A possible Rank computation is shown in Equation 1. In this example, *floor(v)* evaluates v to the greatest integer less than or equal to v.

$$DAGRank(rank) = floor\left(\frac{rank}{MinHopIncrease}\right) \quad (1)$$

However, in most sensor node deployments several data collection points (root nodes) are needed. Thus, three values have to be considered in order to uniquely identify a DODAG: (1) RPL Instance ID for identification of an independent set of DODAGs, optimized for a given scenario; (2) DODAG ID which is a routable IPv6 address belonging to the root; (3) DODAG version number which is incremented each time a DODAG reconstruction is needed. The RPL specification defines the combination of those three values as a DODAG Version. An example is shown in Figure 3. Each of the three constructed topologies may be positioned in different rooms where the construction of a single DODAG is impossible. Due to the fact that the three graphs belong to the same instance, their construction is realized in a similar way (e.g. by considering ETX values).

The RPL specification distinguishes between three logical sets when building upward routes. First, the candidate neighbor set which includes all reachable nodes. They may belong to different DODAG Versions. For example, in Figure 3 node 10 may store node 11 in its candidate neighbor set. Second, the parent set which is a subset of the candidate neighbor set. It includes only nodes that belong to the same DODAG Version. When a node stores a neighbor into the parent set, it becomes attached to the given DODAG.

[1]Virtual roots are not considered in this paper.

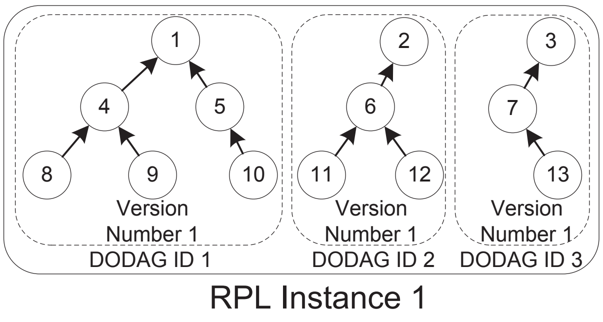


Figure 3: RPL Topology Partition

The last one contains one[2] element: the most preferred next hop taken from the parent set. Note that a node may belong to more than one RPL Instance. In this case, it must join a DODAG for each RPL Instance.

As previously mentioned, RPL dynamically adapts the sending rate of its control DIO messages. To achieve this, two values need to be used: one for defining the minimum sending time interval τ_{min} and another for defining the maximum sending interval τ_{max}. Whenever the sending timer expires, RPL doubles it up to the maximum value τ_{max}. Whenever RPL detects an event which indicates that the topology needs active maintenance, it resets the timer to τ_{min}. Such events are a found inconsistency when forwarding a data packet, joining of a new node, leaving the current DODAG and triggering topology repair.

4. ROUTING LOOPS

The formation of routing loops is a common problem in all kinds of networks. Due to topology changes caused by failure or mobility, a node may pick a new route to a given destination. If the new route includes a network participant which is a descendant, loops may occur. This leads to network congestion, packet drops, energy waste and delays. However, a quick and reliable detection of such topology inconsistencies is not a sufficient solution for LLNs. For example, even in a complete static sensor node deployment a malfunctioning antenna of a node may cause frequent changes of the node's distance to the root. Child nodes may be picked as next hops by their parents and a topology repair mechanism may be triggered. This leads to further energy consumption and waste of bandwidth. Therefore, a routing protocol for LLNs has to define a loop avoidance strategy considered during topology construction.

4.1 Avoidance Mechanisms

So far, it was only said that a Rank represents a node's position in the graph and that router nodes forward DIO control messages for topology maintenance. However, such messages are sent in a multicast manner to the neighboring nodes. If each node in range accepts such messages and takes the sender of the DIO into account for computation of the parent set, it may happen that child nodes are selected as best next hops. Consider the example shown in Figure 4. The topology is constructed by taking ETX into account. Further, the Rank value equals the ETX metric.

[2]RPL also allows multiple equally preferred parents.

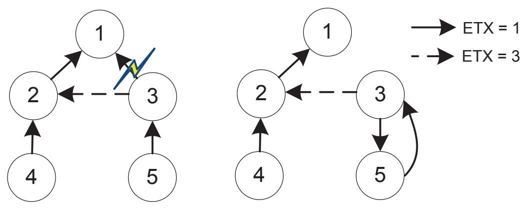


Figure 4: Loop Creation

If node 3 processes a DIO control message from node 5 it will consider it as a valid possible parent. If the link between the root node and node 3 fails, node 3 will simply pick its child node as next hop and a loop will occur, since the ETX cost to the root is 3. Compared to the other possible way to the root (through node 2) the ETX cost is 4.
Therefore, a RPL node does not process DIO messages from nodes deeper (higher Rank) than itself because such nodes may belong to its sub-DODAG. Even if node 4 is reachable for node 3, it should not be considered as next hop. Instead, node 3 has to declare an invalid state, poison its routes and join the DODAG Version again.
RPL also limits the movement of a node within a DODAG Version. Since moving up in a DODAG does not present the risk of creating a loop but moving down might, the RPL specification suggests that a node must never advertise within a DODAG Version a Rank higher than $Rank_{Lowest} + Rank_{MaxInc}$. $Rank_{Lowest}$ is the lowest Rank the node has advertised within a DODAG Version and $Rank_{MaxInc}$ is a predefined constant received via a DIO. Figure 5 illustrates a simple topology where node 1 is the DODAG root.

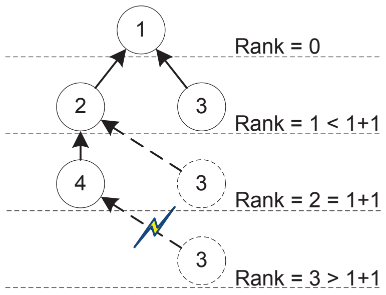


Figure 5: Movement Limitation within a DODAG Version

Let node 2 and 3 have a Rank of 1 and node 4 a Rank of 2. In addition, let $Rank_{MaxInc} = 1$. If node 3 moves away and increases its Rank to 2, it is allowed to choose node 2 as next hop. However, if node 3 moves even further away and increases its Rank to 3, it must not pick node 4 as a next hop. The RPL document does not specify what node 3 should do in this situation. A possible solution is to join another DODAG or advertise a DODAG rebuild request.

4.2 Detection Mechanisms

Usually, in LLNs nodes rarely generate data traffic which makes keeping the topology consistent all the time wasteful in terms of energy consumption. For example, Koen Langendoen et al. introduced in their paper [9] a large-scale experiment in which several sensor nodes were disseminated over a potato field programmed to send data every 60 seconds. Even for this relatively large time interval they have experienced loss of battery power after three weeks of operation. If the topology is kept consistent all the time, it may happen that nodes experience lack of energy after a shorter period of time. Thus, changes in connectivity need not to be addressed until data packets are present.
RPL loop detection uses additional information that is transported in the data packets. It places a RPL Packet Information in the IPv6 option field which is updated and examined on each hop. There are five control fields within the RPL Packet Information. The first one indicates whether the packet is sent in a upward or downward direction. The second one reports if a Rank mismatch has been detected. It is used to indicate whenever the Rank of the sender, stored in the packet, is lower than the Rank of the receiver. The RPL specification suggests that packets should not be immediately dropped if such a inconsistency is detected. Instead, the packet should be forwarded. However, if an inconsistency is detected on the packet for the second time, it must be dropped and the trickle timer must be reset. In this way, route repair is triggered. The third one is the forwarding error field and is used by a child node to report that it does not have a valid route to the destination of the packet. The last two are the Rank of the sender and the RPL Instance ID.

5. RPL METRICS

Many of today's routing protocols use link metrics that do not take a node's current status into account. The status includes typical resources such as CPU usage, available memory and left energy. This may be crucial for LLNs where network devices are usually battery powered and have limited hardware resources. For example, if a chain topology occurs in a sensor network deployment, the last node before the root will usually experience a higher traffic load and forwarding overhead than the others. If in Figure 6 all nodes frequently generate data packets and send them to the root, node 2 may fail very quickly due to the lack of energy. Even if the link between node 3 and 7 is not an optimal solution, it may be reasonable to send data packets through node 7 since it may offer a more stable node condition. Otherwise, if node 2 fails it may take some time until node 3 picks node 7 as next hop and packet drop may occur.

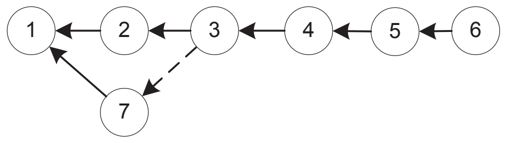


Figure 6: Chain Topology

Therefore, the ROLL Working Group defined several metric categories in [7] that may be considered when selecting the next hop. Some of the metrics defined in the document are carried in messages that are optional from the point of view of a RPL implementation. They have to be additionally specified and are not further described in this paper. In the following two possible metric computations from the RPL metrics document will be shortly discussed.

5.1 Node Energy Consumption

This method suggests that a node should consider the energy level of its neighbors before picking them as possible parents. For this purpose, two units of information are used: (1) the type of the node which indicates how it is supplied with power and (2) the Energy Estimation (EE). The RPL metric specification defines three possible states for the first information field: powered, on batteries and scavenger. If a network device is powered it means that it may be the root node connected to a PC or it may be some sort of special data collector (e.g. cluster-heads in hierarchical routing). Such nodes may report a maximum EE value and, in general, are preferable during parent selection. If a node is on batteries, it has to compute its EE value by using Equation 2. The $Power_{now}$ value is the remaining energy and $Power_{max}$ is the power estimation reported at boot up.

$$EE = \frac{Power_{now}}{Power_{max}} \cdot 100 \tag{2}$$

However, if a node derives energy from external sources [13] it may report EE as a quantity value that is computed by dividing the amount of power the node has acquired by the power consumed. This may be a rough estimation of how much load a node experiences for a given period of time.

5.2 ETX

This metric is an approximation of the expected number of transmissions until a data packet reaches the gateway node. A node that is one hop away from the root, with perfect signal strength and very little interference, may have an ETX of 1. Another node with a less reliable connection to a root may have a higher ETX.
ETX is a bidirectional single-hop link quality computation between two neighbor nodes [1]. For the computation a metric called Packet Reception Rate (PRR) is used. PRR is calculated at the receiver node for each window ρ of received packets, as follows:

$$PRR(\rho) = \frac{Number\ of\ received\ packets}{Number\ of\ sent\ packets} \tag{3}$$

In literature the value computed in Equation 3 is also defined as *in-quality*, which is the quality from node A to node B measured by node B by counting the successfully received packets from A among all transmitted. In this paper it will be called PRR_{down}. For the actual ETX estimation the *out-quality* is further needed. This is the in-quality estimated by node A and is defined as PRR_{up} at node B. In this way node B can calculate ETX as shown in Equation 4:

$$ETX = \frac{1}{PRR_{down} \cdot PRR_{up}} \tag{4}$$

6. DOWNWARD ROUTING

The support of downward routing is another important key feature of the protocol. By supporting P2MP traffic it is possible for a network administrator to control nodes that are even not in range. This is very useful for performance evaluation purposes where usually several hundred nodes are spread over a large area. If such traffic is not supported, even the slightest changes, such as a timer value, may require to find the node, disconnect it from the network and upload a new code image. Moreover, if the idea of the Internet of Things is considered, P2MP becomes a must for LLN routing protocols [17].
The RPL specification defines two modes of operation for supporting P2MP. First, the non-storing mode which makes use of source routing. In this mode each node has to propagate its parent list up to the root. After receiving such topology information, the root computes the path to the destinations. Second, the storing mode which is fully stateful. Here, each non-root and non-leaf network participant has to maintain a routing table for possible destinations. Note that any given RPL Instance is either storing or non-storing.

6.1 DAO Message Structure

As mentioned in Section 2, DAO messages are used by RPL nodes to propagate routing information in order to enable P2MP traffic. Figure 7 represents the structure of a DAO message.

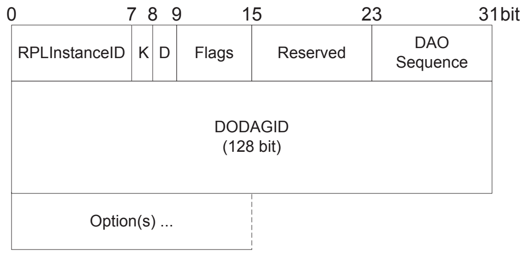


Figure 7: DAO Message Structure

Similar to the DIO message, the DAO message includes an RPL Instance ID. This is the same one that the node has learned from a received DIO. The next field is the Flags field where only the first two bits are used. The first one is the 'K' flag which indicates whether the sender of the DAO expects to receive a DAO-ACK in response. The second one is the 'D' flag which indicates if the DODAGID field is present. Due to the fact that the DODAGID field represents the IPv6 address of the root it may be omitted if there is only one root node present. The DAO Sequence field is a sequence number that is incremented for each outgoing DAO message by the sender. The sequence number ensures the freshness of a DAO message and is echoed back by the parent when DAO-ACKs are used.
The message can be further extended by the use of options. In this paper, only two will be discussed: the Target option and the Transit Information option. The first one is used to indicate a target IPv6 address, prefix or multicast group. In terms of routing, it represents reachability information. RPL defines the structure of it, as shown in Figure 8.

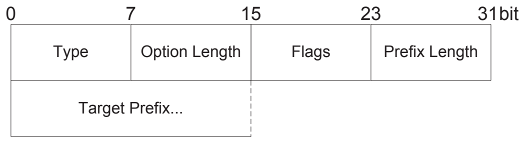


Figure 8: DAO Target Option

First of all, it stores the type of the option (`0x05`). Afterwards, the length of the option and the length of the prefix are included. In the latter case, the number of valid leading bits of the routing prefix is meant. The last field is the target prefix and it may identify, for example, a single node or a whole group of nodes that can be reached via a common prefix. The Flags field is experimental and is not used.
The second type is the Transit Information option. It is used to indicate attributes for a path to one or more destinations. Destinations are indicated by one or more Target options that precede the Transit Information option(s). Figure 9 outlines its structure.

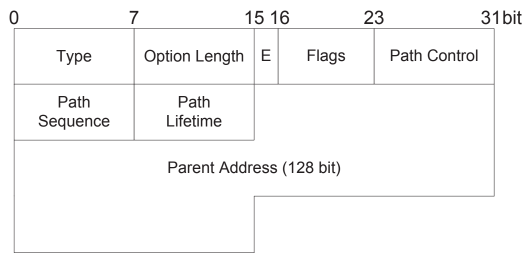


Figure 9: DAO Transit Information Option

The first field represents the type of the option and is always set to `0x06`. The second field is the Option Length field which indicates if the Parent Address field is present. Note that in case of storing mode the IPv6 address of the parent may be omitted. The next field is the Flags field where only the first bit is used. The 'E' flag indicates whether the parent router redistributes external targets into the RPL network. Such targets may have been learned by an external protocol. However, they do not play a crucial role for this paper. The next three fields are needed for reachability control. First, the Path Control field is used to limit the number of parents to which a DAO message may be sent. Second, the Path Sequence field indicates if a Target option with updated information has been issued. Third, the Path Lifetime defines how long a prefix for a destination should be kept valid. The time is measured in Lifetime Units and is implementation specific.

6.2 Non-Storing Mode

In the non-storing mode each node generates a DAO message and sends it to the DODAG root. The time generation interval in which DAO messages are sent depends on the implementation. However, the RPL specification suggests that the needed delay between two DAO sending operations may be inversely proportional to the Rank. In this way, if a node is far away from the root it will generate DAOs more often than a node that is closely positioned to the gateway. Furthermore, each node has to extend the DAO message by using the aforementioned Transit Information option. In the Parent Address field the IPv6 address of a parent node is stored. It should be kept in mind that a typical non-storing node may use multiple Transit Information options in order to report its complete parent set to the root node. The resulting DAO message is sent directly to the DODAG root along the default route created during parent selection. Figure 10 illustrates this process.

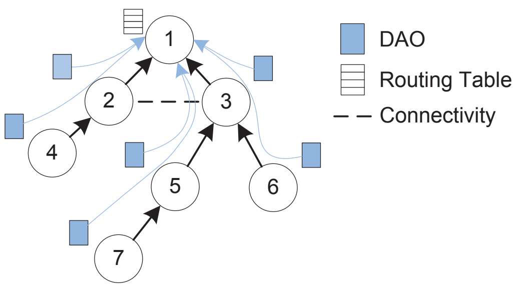


Figure 10: RPL Non-Storing Mode

Usually, intermediate nodes inspect DAO messages and compare the stored path sequence number with the last seen. In this way, a node can distinguish between stale and up-to-date routing information.
After collecting the needed information, the root pieces the downward route together. If it needs to send a data packet to a given destination the IPv6 Source Routing header is used. Thus, network nodes can easily forward a data packet until it reaches the given destination or the IPv6 Hop Limit reaches 0.

6.3 Storing Mode

Similar to the non-storing mode, the storing mode also requires the generation of DAO messages. The configuration of the timer triggering such messages may be implemented in the same way as it was mentioned above. However, a DAO is no longer propagated to the DODAG root. Instead, it is sent as unicast to all parent nodes which maintain additional downward routing tables. Figure 11 gives an overview of this process.

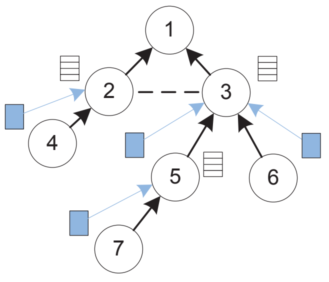


Figure 11: RPL Storing Mode

When a node sends a DAO message it has to keep the Parent address field in the Transit Information option empty since a node's responsibility is not to advertise its parent set, but to announce prefixes that are reachable through it. If the device is a router it has to use the Target option in order to advertise a prefix. In case it has multiple prefixes to advertise, it must extent the DAO by multiple Target options.
If a data packet is sent from the DODAG root node, it must be sent only to all one-hop neighbors. Afterwards, through table look-up the packet is routed downwards until it reaches its destination or the Hop Limit in the IPv6 header reaches 0. Such networks may also be hierarchically organized and route aggregation may be performed. Moreover, a sub-DODAG may run another RPL Instance which makes it possible to

combine storing and non-storing mode. In this way, nodes with constraints regarding memory may be grouped together and operate only in non-storing mode.

7. PERFORMANCE EVALUATION

Gnawali et al. [8] introduced a RPL implementation, called TinyRPL. In their work they use the Berkeley Low-power IP (BLIP) stack in TinyOS 2.x [15] which interacts with their protocol implementation. More precisely, the control plane of TinyRPL communicates with the BLIP stack which offers a forwarding plane implementation. It should be kept in mind that BLIP also offers an implementation in TinyOS of a number of IP-based protocols such as TCP and UDP. In this way, during one test run several transport protocol configurations can be evaluated and compared.
TinyRPL is further compared with the Collection Tree Protocol (CTP) [5], the de-facto routing protocol standard for TinyOS. For this purpose, a 51-node TelosB [3] testbed scenario is build where only one node is acting as a root. Overall, they use two network configurations: one that has a data packet generation interval of 5 seconds and another that has a data packet time interval of 10 seconds. For each of them the protocols are tested against each other and an estimation of the packet reception ratio and the average number of control packets is made. Each testbed run lasts 24 hours. Since CTP uses ETX for topology maintenance, the OF of TinyRPL is set to use the same method for metric computation. In order to make a fair comparison the downstream routing options are disabled since the standard CTP configuration does not define a mechanism similar to the one realized with DAO messages.
In all test runs both protocols managed to achieve a reception rate of approximately 99 %. The amount of control traffic reported also stays at the same level since both protocols make use of the trickle timer and are evaluated in static scenarios.

8. CONCLUSION

LLNs and, in particular, WSNs are rapidly emerging as a new type of distributed systems, with applications in different areas such as target tracking, building environment, traffic management, etc. However, to achieve a reliable communication, to guarantee a high delivery ratio and to be at same time energy efficient requires special mechanisms realized at the network layer.
As a result, RPL was specified and developed in order to overcome these requirements. The protocol is an end-to-end IP-based solution which does not require translation gateways in order to address nodes within the network from the outside world. Moreover, the use of IPv6 allows deploying RESTful web services for sensor networks [16]. In this way, a client (e.g. PC connected to the root) may initiate requests by using HTTP to nodes within the network which will return the appropriate responses. It is even easier to use such a feature, since RPL defines in its specification the support of downward traffic. Because of P2MP a root node can easily propagate such requests to the nodes.
It should also be mentioned that RPL may run on nodes that have limited energy and memory capabilities. The protocol dynamically adapts the sending rate of routing control messages which will be frequently generated only if the network is in a unstable condition. In addition, the protocol allows the use of source routing when P2MP is needed which reduces the memory overhead on intermediate nodes.
RPL also allows optimization of the network for different application scenarios and deployments. For example, it may take into account the link quality between nodes or their current amount of energy which makes it an efficient solution for WSN deployments.

9. REFERENCES

[1] N. Baccour, A. Koubaa, M. B. Jamaa, H. Youssef, M. Zuniga, and M. Alves. A comparative simulation study of link quality estimators in wireless sensor networks. Technical Report TR-09-03-30, open-ZB, 2009.

[2] A. Cerpa, J. Wong, L. Kuang, M. Potkonjak, and D. Estrin. Statistical model of lossy links in wireless sensor networks. In *Information Processing in Sensor Networks, IPSN 2005*, pages 81–88, 2005.

[3] Crossbow Technology. TelosB datasheet. http://www.willow.co.uk/TelosB_Datasheet.pdf, 2010.

[4] A. Dunkels and J. P. Vasseur. Why IP. IPSO Alliance White Paper #1, 2008.

[5] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection Tree Protocol. Technical Report SING-09-01, Stanford University, 2009.

[6] Internet Assigned Numbers Authority (IANA). Internet Control Message Protocol version 6 (ICMPv6) Parameters. http://www.iana.org/assignments/, 2011.

[7] M. Kim, J. P. Vasseur, K. Pister, N. Dejean, and D. Barthel. Routing Metrics used for Path Calculation in Low Power and Lossy Networks. Internet draft, http://datatracker.ietf.org/wg/roll/, 2011.

[8] J. Ko, S. Dawson-Haggerty, O. Gnawali, D. Culler, and A. Terzis. Evaluating the performance of RPL and 6LoWPAN in TinyOS. In *Proceedings of the Workshop on Extending the Internet to Low power and Lossy Networks (IP+SN)*, 2011.

[9] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *The International Workshop on Parallel and Distributed Real-Time Systems*, 2006.

[10] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code maintenance and propagation in wireless sensor networks. In *Proceedings of the USENIX NSDI Conference*, pages 15–28, San Francisco, CA, USA, 2004.

[11] Y. Li, J. Harms, and R. Holte. Impact of lossy links on performance of multihop wireless networks. In *Computer Communications and Networks, 2005. ICCCN 2005*, pages 303–308, 2005.

[12] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. RFC 4944: Transmission of IPv6 packets over IEEE 802.14 networks, 2007.

[13] S. J. Roundy. *Energy scavenging for wireless sensor nodes with Focus on vibration to electricity conversion.* PhD thesis, University of California at Berkeley, 2003.

[14] Routing Over Low power and Lossy networks (ROLL). Description of Working Group. http://datatracker.ietf.org/wg/roll/charter/, 2011.

[15] The TinyOS Alliance. Poster abstract: TinyOS 2.1, adding threads and memory protection to TinyOS. In *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, Raleigh, NC, USA, 2008.

[16] TinyOS community. BLIP tutorial. http://docs.tinyos.net/index.php/BLIP_Tutorial, 2010.

[17] J. P. Vasseur. The Internet of Things: Dream or Reality. SENSORCOMM 2010: The Fourth International Conference on Sensor Technologies and Applications, 2010.

[18] J. P. Vasseur, N. Agarwal, J. Hui, Z. Shelby, P. Bertrand, and C. Chauvenet. RPL: IPv6 Routing Protocol for Low power and Lossy Networks. IPSO Alliance, 2011.

[19] T. Winter, P. Thubert, A. Brandt, T. Clausen, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, and J. P. Vasseur. RPL: IPv6 Routing Protocol for Low power and Lossy Networks. ROLL Working Group, 2011.

[20] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pages 1–13, New York, USA, 2003.

Constrained Application Protocol for Internet of Things

Xi Chen, chen857 (at) wustl.edu (A paper written under the guidance of Prof. Raj Jain)

Download

Abstract:

Internet of things (IoT) is an important part of a new generation of technology that every object no matter things or human could be connected to Internet. There are many wireless protocols (like IEEE 802.11 Series, 802.15 Series, Zigbee, etc) for communication between devices. However, considering a lot of small devices are unable to communicate efficiently with constrained resources, Internet Engineering Task Force (IETF) has developed a lightweight protocol: Constrained Application Protocol (CoAP). Firstly, this paper summarizes some wireless protocols. Then it introduce CoAP and corresponding security protocol DTLS. Finally, an application is given.

Keywords

IoT, CoAP, DTLS, wireless protocols, smart home, energy control system

Table of Contents:

1. Introduction

Internet of Things (IoT) is represented as a global network which intelligently connects all the objects no matter devices, systems or human, it is with self-configuring capabilities based on standard and interoperable protocols and formats [Petersburg 12]. Through smart sensing, identification technology and persuasive computing, IoT has been called the Third Wave in information industry following the computer and the Internet. There are hundreds of protocols supported by IoT. Of the many protocols, wireless protocols play an important role in IoT development.

In section 1, some wireless protocols in different layers of IoT are introduced. One latest protocol for application layer CoAP is given and its features and functions are summarized. By comparing it with Hypertext Transfer Protocol (HTTP), its advantages are presented. In section 2, some important CoAP models are explained in details, such as the message layer model, request/response layer model and message format. In sections 3, a security protocol Datagram Transport Layer Security (DTLS) for transmission protection is described. It achieves necessary elements for securing CoAP, like integrity, authentication and confidentiality. Then, an application of CoAP Smart Homes is described, it helps users to manage energy control systems, which reduce power consumption and prevent accidents.

2. Overview of Wireless Protocols for IoT

IoT needs to integrate various sensors, computer and communication equipment, which are using different communication protocols. Wireless Protocols are mainly used in three layers, which are PHY/MAC layer, Network/Communication layer and Application layer.

CoAP is one of the latest application layer protocol developed by IETF for smart devices to connect to Internet. As many devices exist as components in vehicles and buildings with constrained resources, it leads a lot of variation in power computing, communication bandwidth etc. Thus lightweight protocol CoAP is intended to be used and considered as a replacement of HTTP for being an IoT application layer protocol.

2.1 Protocols in Different Layers

IoT PHY/MAC Layers involve all the common wireless communication technology, such as IEEE 802.11 series, 802.15 series, HART (Highway Addressable Remote Transducer), etc. IEEE 802.15.4 standard specifies MAC/PHY part for long-range wireless personal area network (LR-WPAN). Zigbee, WirelessHART are based on IEEE802.15.4 by adding upper layers.

As TCP/IP lay the foundation for the Internet, thus IoT communication network mainly employ TCP and UDP protocols. Compared with UDP protocol, TCP protocol is more complex which makes it not easy to employ on resource-constrained devices. Now, most of IoT use UDP protocol. But UDP is not stable, which needs to combine with application layer to improve the stability.

Application layer usually employ HTTP to provide web service, but HTTP has high computation complexity, low data rate and high energy consumption. Therefore, IETF has developed several lightweight protocols, e.g., CoAP, Embedded Binary HTTP (EBHTTP), Lean Transport Protocol (LTP). The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks [Z.Shelby13]. EBHTTP is a binary-formatted, space-efficient, stateless encoding of the standard HTTP/1.1 protocol [G.Tolle13]. EBHTTP is primarily designed for transportation of small data between resource-constrained nodes, which is similar to CoAP. LTP is a lightweight Web Service transport protocol that allows the transparent exchange of Web Service messages between all kinds of resource constrained devices and server or PC class systems [Glombitza10].Table 1 summarizes protocols in different layers.

Table 1 protocols in different layers

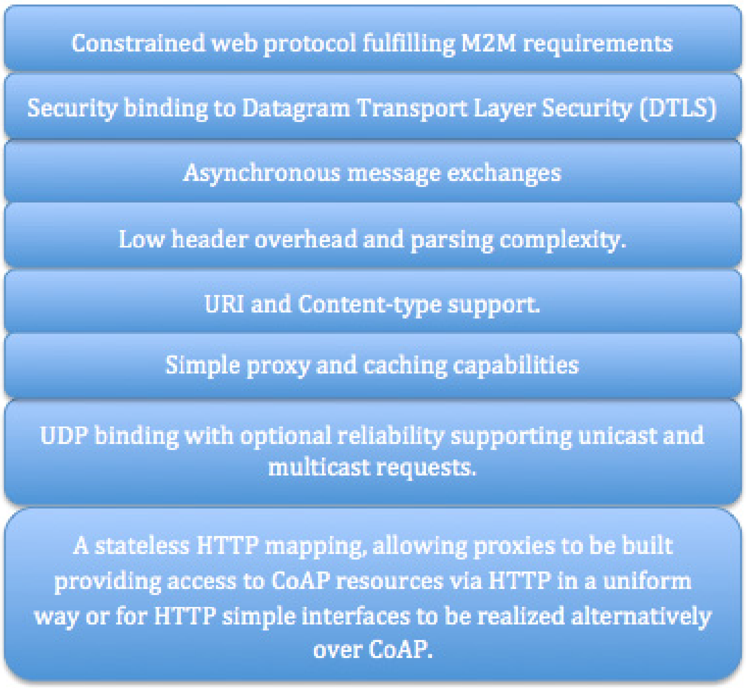
Application layer	HTTP, CoAP, EBHTTP, LTP, SNMP, IPfix, DNS, NTP, SSH, DLMS, COSEM, DNP, MODBUS
Network/Communication layer	IPv6/IPv4, RPL, TCP/UDP, uIP, SLIP, 6LoWPAN,
PHY/MAC layer	IEEE 802.11 Series, 802.15 Series, 802.3, 802.16, WirelessHART, Z-WAVE, UWB, IrDA, PLC, LonWorks, KNX

2.2 CoAP Features

With the completion of the CoAP specification, it is expected that there will be million of devices deployed in various application domains in the future. These applications range from smart energy, smart grid, building control, intelligent lighting control, industrial control systems, asset tracking, to environment monitoring. CoAP would become the standard protocol to enable interaction between devices and to support IoT applications [S.Keoh13]. The Constrained RESTful Environments (CoRE) is the workgroup in IEFT that is designing the CoAP protocol.

CoAP needs to consider optimizing length of datagram and satisfying REST protocol to support URI (Uniform Resource Identifier). It also needs to provide dependable communication based on UDP protocol. Table 2 shows the CoAP features [Petersburg12]

Table 2 CoAP features



- Constrained web protocol fulfilling M2M requirements
- Security binding to Datagram Transport Layer Security (DTLS)
- Asynchronous message exchanges
- Low header overhead and parsing complexity.
- URI and Content-type support.
- Simple proxy and caching capabilities
- UDP binding with optional reliability supporting unicast and multicast requests.
- A stateless HTTP mapping, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way or for HTTP simple interfaces to be realized alternatively over CoAP.

2.3 CoAP vs. HTTP

CoAP is network-oriented protocol, using similar features to HTTP but also allows for low overhead, multicast, etc. As HTTP protocol is a long-term successful standard, it can use small script to integrate various resources and services. Interoperation provided by HTTP is the key point of IoT, for this, HTTP is employed in application level. However, HTTP is based on TCP protocol using point to point (p2p) communication model that not suitable for notification push services. Also, for constrained devices, HTTP is too complex.

Unlike HTTP based protocols, CoAP operates over UDP instead of using complex congestion control as in TCP [Koojana11]. CoAP is based on REST architecture, which is a general design for accessing Internet resources. In order to overcome disadvantage in constrained resource, CoAP need to optimize the length of datagram and provide reliable communication. On one side, CoAP provides URI, REST method such as GET, POST, PUT, and DELETE. On the other side, based on lightweight UDP protocol, CoAP allows IP multicast, which satisfies group communication for IoT. To compensate for the unreliability of UDP protocol, CoAP defines a retransmission mechanism and provides resource discovery mechanism with resource description [Shelby11]. Fig 1 shows the HTTP and CoAP protocol stacks.

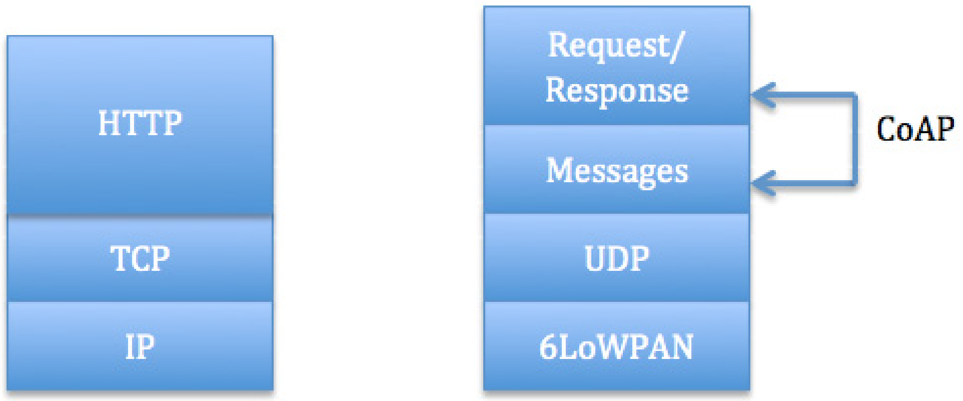
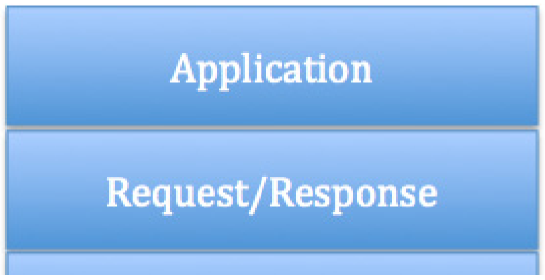


Fig 1: HTTP and CoAP protocol stacks

CoAP is not just a simply compression of HTTP protocol. Considering low processing capability and low power consuming demand of restrained resource, CoAP redesigned some features of HTTP to accommodate these limitations. HTTP used under unconstrained network and CoAP used under constrained network. Recently, HTTP-CoAP cross protocol proxy arouses scientific interest, it has and important role in solving congestion problem in the constrained environment [Castellani12].

3. CoAP Structure Model

CoAP interactive model is similar to HTTP's client/server model. Fig 2 shows that CoAP employs a two layers structure. The bottom layer is Message layer that has been designed to deal with UDP and asynchronous switching. The request/response layer concerns communication method and deal with request/response message.



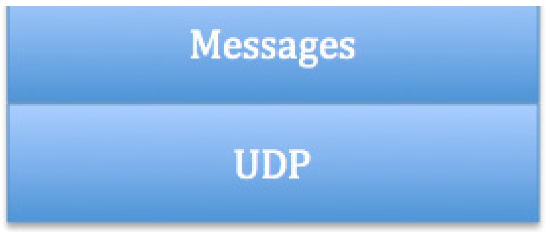


Fig. 2: Abstract Layer of CoAP [Z.Shelby13]

3.1 Message Layer model

Message Layer supports 4 types message: CON (confirmable), NON (non-confirmable), ACK (Acknowledgement), RST (Reset) [Z.Shelby13].

a) Reliable message transport: Keep retransmission until get ACK with the same message ID (like 0x8c56 in fig. 3). Using default time out and decreasing counting time exponentially when transmitting CON. If recipient fail to process message, it responses by replacing ACK with RST. Fig. 3 shows a reliable message transport.

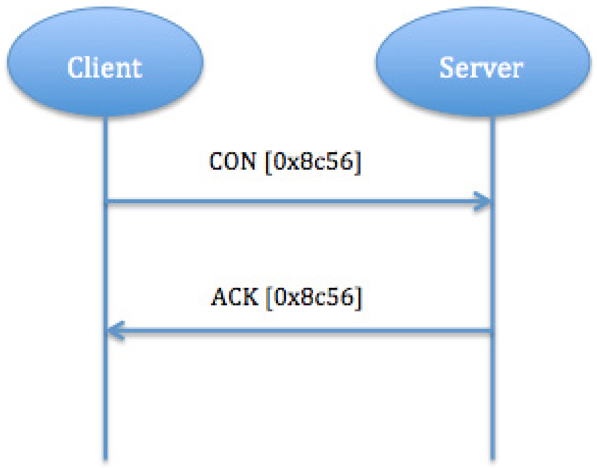


Fig. 3: Reliable Message Transport

b) Unreliable message transport: transporting with NON type message. It doesn't need to be ACKed, but has to contain message ID for supervising in case of retransmission. If recipient fail to process message, server replies RST. Fig. 4 shows unreliable message transport.

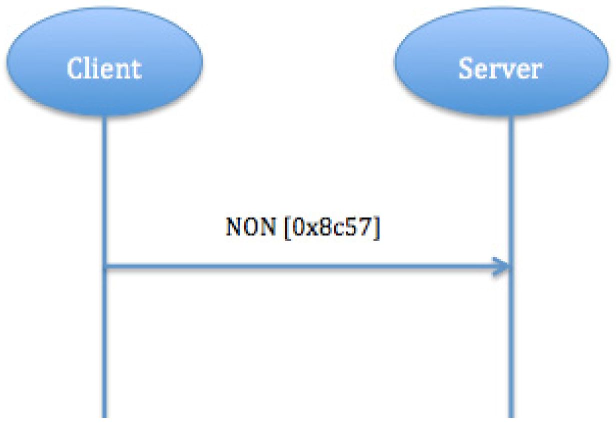


Fig. 4: Unreliable Message Transport

3.2 request/response Layer model

a) Piggy-backed: Client sends request using CON type or NON type message and receives response ACK with confirmable message immediately. In fig. 5, for successful response, ACK contain response message (identify by using token), for failure response, ACK contain failure response code.

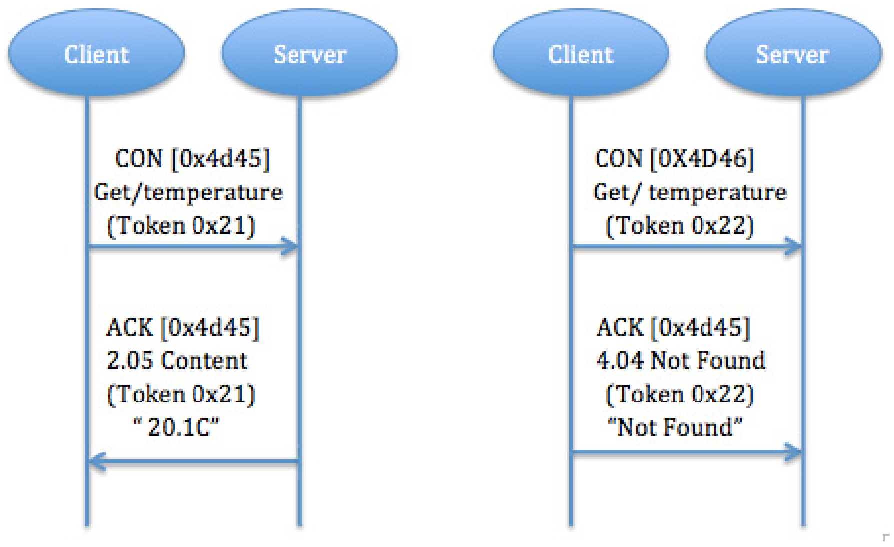


Fig. 5: The successful and failure response results of GET method

b) Separate response: If server receive a CON type message but not able to response this request immediately, it will send an empty ACK in case of client resend this message. When server ready to response this request, it will send a new CON to client and client reply a confirmable message with acknowledgment. ACK is just to confirm CON message, no matter CON message carry request or response (fig. 6).

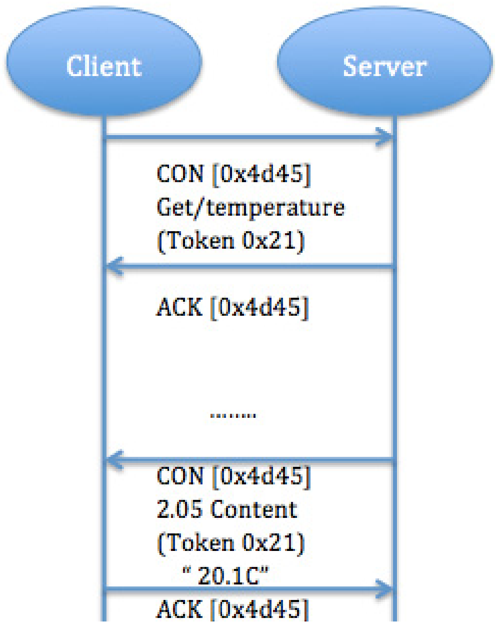


Fig. 6: A Get request with a separate response

c) Non confirmable request and response: unlike Piggy-backed response carry confirmable message, in Non confirmable request client send NON type message indicate that Server don't need to confirm. Server will resend a NON type message with response (fig. 7).

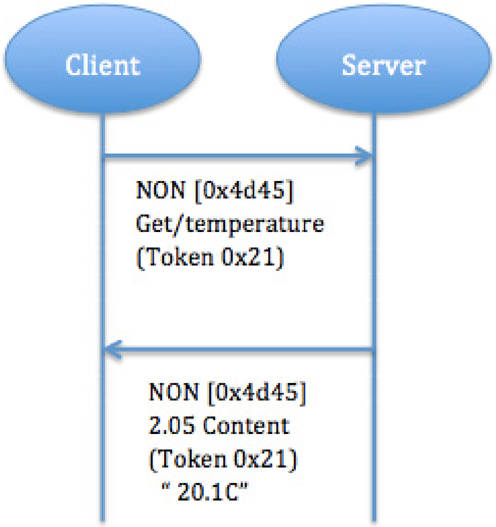


Fig. 7: Non confirmable request and response

3.3 Message Format

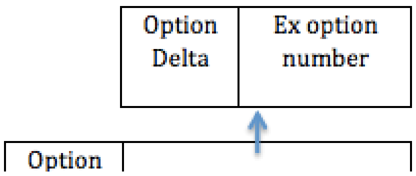
CoAP is based on the exchange of compact messages that, by default, are transmitted over UDP (i.e. each CoAP message occupies the data section of one UDP datagram) [Petersburg12]. Message of CoAP uses simple binary format. Message= fixed-size 4-byte header plus a variable-length Token plus a sequence of CoAP options plus payload. The format is shown in Table 3.

Table 3 Message Format

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

Ver	T	OC	Code	MessageID
Token (if any, TKL bytes)…				
Options (if any)…				
Payload (if any)…				

Option number=option delta+ ex option number. The format is shown in Fig. 8



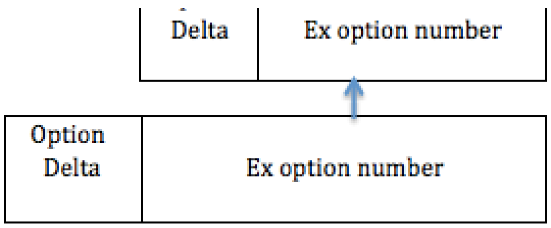


Fig. 8: CoAP option format

4. Security Protocol & Application for CoAP

CoAP is now becoming the standard protocol for IoT applications. Security is important to protect the communication between devices. In the following part, a security protocol DTLS is introduced. Also, one of CoAP application, Smart Homes, is described in this section.

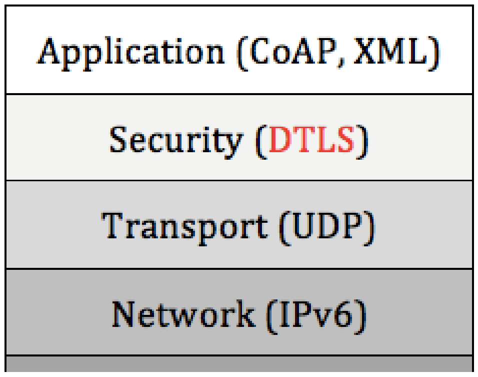
There are three main elements when considering security, namely integrity, authentication and confidentiality. DTLS can achieve all of them [Kothmayr 12]. IETF modifies DTLS to develop another protocol DTLS. DTLS employ TCP, which is too complex. DTLS solves two problems: reordering and packet lost. It adds three implements: 1 packet retransmission. 2 assigning sequence number within the handshake. 3 replay detection.

Unlike network layer security protocols, DTLS in application layer (fig.9) protect end-to-end communication. No end-to-end communication protection will make it easy for attacker to access to all text data that passes through a compromised node. DTLS also avoids cryptographic overhead problems that occur in lower layer security protocols.

4.1 Why use DTLS for CoAP Security

There are three main elements when considering security, namely integrity, authentication and confidentiality. DTLS can achieve all of them[Kothmayr12]. IETF modifies DTLS to develop another protocol DTLS. DTLS employ TCP, which is too complex. DTLS solves two problems: reordering and packet lost. It adds three implements: 1 packet retransmission. 2 assigning sequence number within the handshake. 3 replay detection.

Unlike network layer security protocols, DTLS in application layer (fig. 9) protect end-to-end communication. No end-to-end communication protection will make it easy for attacker to access to all text data that passes through a compromised node. DTLS also avoids cryptographic overhead problems that occur in lower layer security protocols.



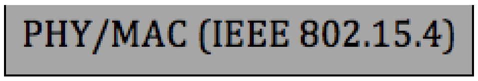


Fig. 9: DTLS in protocol stack

4.2 Structure of DTLS

There are two layers in DTLS. The bottom one contains Record protocol. The top one include three protocols which are Alert, Handshake and application data, in some condition Change Cipher Spec protocol may replace one of them. Change Cipher Spec message is used to notify Record protocol to protect subsequent records with just-negotiate cipher suite and keys. [Raza13]

Record protocol [E.Rescorla12]protects application data by using keys generated during Handshake. For outgoing message, protocol divide, compress, encrypt and apply Message Authentication Code (MAC) to them. For incoming message, protocol reassemble, decompress, decrypt and verify them. Record header consists of two parts, one is content type and another is fragment field. Content type decides what is contained in fragment field. It could be alert protocol, Handshake protocol or application data. Compare with DTLS Record, Handshake protocol is rather a complex one, which involves in a lot of exchange steps. Individual messages are grouped into message flights. Fig. 10 shows the process of Handshake.

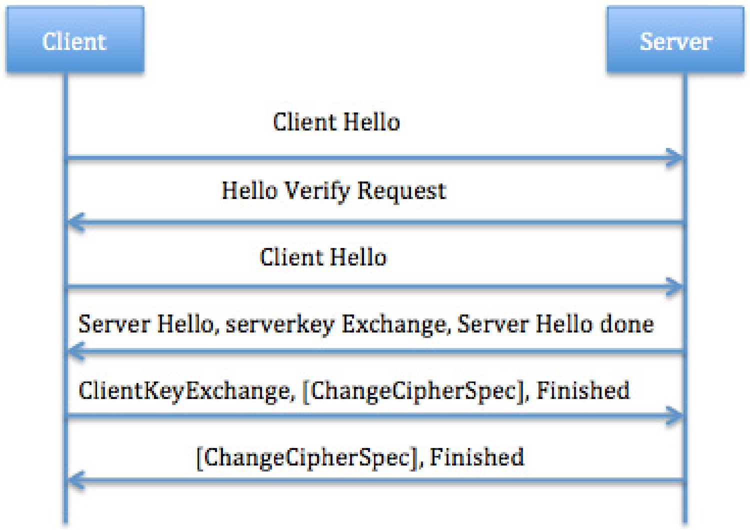
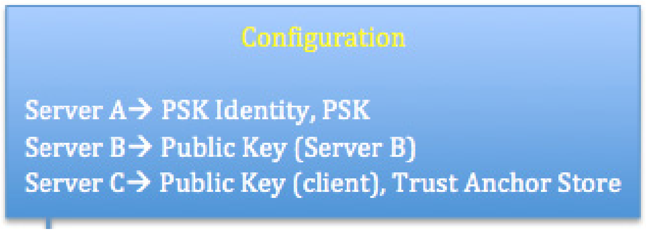

Client
Server
Client Hello
Hello Verify Request
Client Hello
Server Hello, serverkey Exchange, Server Hello done
ClientKeyExchange, [ChangeCipherSpec], Finished
[ChangeCipherSpec], Finished

Fig. 10: Process of Handshake

The Architecture show below (Fig. 11) is for uni-cast communication (client interacts with one or more servers). The client also needs to know which certificate or raw public key it has to use with a specific server [Hartke13].


Configuration
Server A→ PSK Identity, PSK
Server B→ Public Key (Server B)
Server C→ Public Key (client), Trust Anchor Store

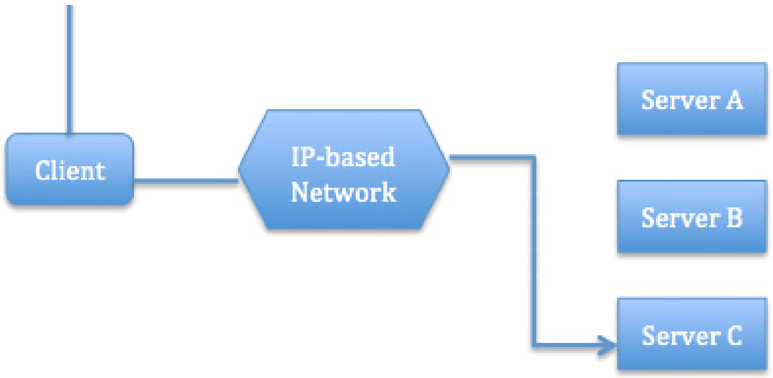


Fig. 11: Uni-cast communication model

4.3 CoAP Application for Smart Homes

Information appliance, control equipment and communication equipment in Smart home networks have the characters of low-cost and lightweight. Thus, CoAP could be seen as the best protocol choice for home communication networks.

Smart home network provide controlling and monitoring energy of home devices. Energy control systems employ smart socket management and monitor power consuming equipment to provide voltage, current and other energy information. It could realize accident warning, remote control and dynamic energy saving. The system structure is shown in Fig. 12. Every data collection node with CoAP client could exchange information with other nodes. CoAP could both be installed in LAN or Internet. Unlike Many wireless protocols for home automotive devices, CoAP is designed not constrained in a local network but provide the fundamental basis of the web [Bergmann12]. In this system, CoAP-HTTP proxies are employed to provide HTTP client connection to CoAP resources and vice versa.

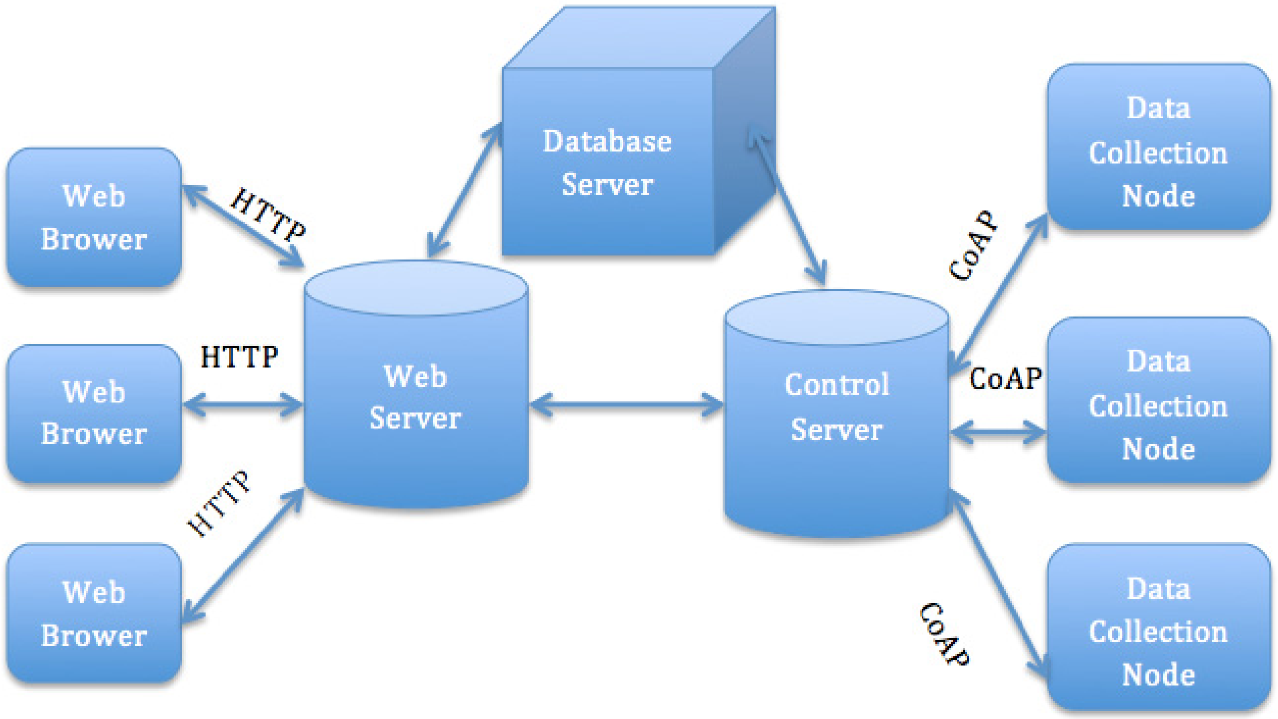


Fig. 12: energy control system

In system networking, date collection nodes consist of one proxy, smart socket and wireless data collection module. Energy information and environment information of equipment is collected by the smart socket and transported to data collection module through wireless channel, then send serial data to proxy to process and pack data. Control server analyzes all the data and stores them in database. The system integrate home network and Internet, users can access system webpage to remotely control switch, manage configuration, query energy consumption, etc. At the same timei1/4OEthis system will monitor environment situation. If any abnormal things happens, (like high temperature or voltage beyond the safety range) the system will analyze it and close relevant equipment.

5. Conclusion

This report summarizes some wireless protocols for IoT and introduces CoAP protocol in details with describing main features and modules. CoAP is based on HTTP protocol and is designed for constrained resource devices. Through comparing CoAP and HTTP together, the advantages of CoAP for IoT are analyzed. This report also provides a corresponding security protocol DTLS and one possible application of CoAP for IoT. As a number of protocols are being perfected, more and more smart homes will employ these wireless protocols for intelligent control.

6. References

- [Petersburg12] St. Petersburg, "Internet of Things, Smart Spaces, and Next Generation Networking," Russia, August 27-29, 2012 http://link.springer.com/book/10.1007%2F978-3-642-32686-8
- [Z. Shelby13] Z. Shelby, Sensinode, K. Hartke, "Constrained Application Protocol (CoAP)," draft-ietf-core-coap-18. [2013-06--28] http://tools.ietf.org/html/draft-ietf-core-coap-18
- [G.Tolle13] G.Tolle, Arch Rock Corporation. Embedded Binary HTTP (EBHTTP) draft-tolle-core-ebhttp-00.[2010-03-23] https://tools.ietf.org/html/draft-tolle-core-ebhttp-00
- [Glombitza10] Glombitza, N.; Pfisterer, D.; Fischer, S., "LTP: An Efficient Web Service Transport Protocol for Resource Constrained Devices," Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conference on , vol., no., pp.1,9, 21-25 June 2010 http://ieeexplore.ieee.org/xpl/login.jsptp=&arnumber=5508255&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5508255
- [S. Keoh13] S. Keoh, Philips Research, Z. Shelby,Sensinode. Profiling of DTLS for CoAP-based IoT Applications draft-keoh-dice-dtls-profile-iot-00 [2013-11-05] http://tools.ietf.org/html/draft-keoh-dice-dtls-profile-iot-00
- [Koojana11] Koojana Kuladinithi, Olaf Bergmann, Thomas Potsch Markus Becker, Carmelita Gorg, "Implementation of CoAP and its Application in Transport Logistics," In Proceedings of the Workshop on Extending the Internet to Low power and Lossy Networks (April 2011) http://hinrg.cs.jhu.edu/joomla/images/stories/coap-ipsn.pdf
- [Shelby11] Zach Shelby, "CoRE Link Format," draft-ietf-core-link- format-07 https://tools.ietf.org/html/draft-ietf-core-link-format-01
- [Castellani12] Castellani, A.; Fossati, T.; Loreto, S., "HTTP-CoAP cross protocol proxy: an implementation viewpoint," Mobile Adhoc and Sensor Systems (MASS), 2012 IEEE 9th International Conference on , vol.Supplement, no., pp.1,6, 8-11 Oct. 2012 http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6708523

- [Kothmayr12] A DTLS Based End-To-End Security Architecture for the Internet of Things with Two-way Authentication Thomas Kothmayr, Corinna Schimitt, Wen Hu, Michael Bruning. 2012 http://kothmayr.net/wp-content/papercite-data/pdf/kothmayr2012dtls.pdf
- [Raza13] Raza, S.; Shafagh, H.; Hewage, K.; Hummen, R.; Voigt, T., "Lithe: Lightweight Secure CoAP for the Internet of Things," Sensors Journal, IEEE , vol.13, no.10, pp.3711,3720, Oct. 2013 http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6576185
- [E. Rescorla12] E. Rescorlai, N. Modadugu, "Datagram Transport Layer Security Version 1.2," RFC Standard 6347, Jan. 2012. http://tools.ietf.org/html/rfc6347
- [Hartke13] Klaus Hartke, Hannes Tschofenig, A DTLS Profile for the Internet of Things draft-hartke-dice-profile-00, [2013-11-04] http://tools.ietf.org/html/draft-hartke-dice-profile-00
- [Bergmann12] Bergmann, O.; Hillmann, K.T.; Gerdes, S., "A CoAP-gateway for smart homes," Computing, Networking and Communications (ICNC), 2012 International Conference on, pp.446,450, Jan. 30 2012-Feb. 2 2012 http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6167461&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6167461

7. Acronyms

IoT	Internet of Things
IEFT	Internet Engineering Task Force
CoAP	Constrained Application Protocol
HTTP	Hypertext Transfer Protocol
HART	Highway Addressable Remote Transducer
LR-WPAN	Low-Rate Wireless Personal Area Network
EBHTTP	Embedded Binary HTTP
LTP	Lean Transport Protocol
CoRE	Constrained RESTful Environments
REST	Representation State Transfer
URI	Uniform Resource Identifier
P2P	Point to Point
DTLS	Datagram Transport Layer Security

Last Modified: April 30, 2014
This and other papers on current issues in Wireless and Mobile Networking are available online at http://www.cse.wustl.edu/~jain/cse574-14/index.html
Back to Raj Jain's Home Page

MQTT

Introducing MQTT

MQTT is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium.

MQTT characteristics

• Lightweight message queueing and transport protocol

• Asynchronous communication model with messages (events)

• Low overhead (2 bytes header) for low network bandwidth applications

• Publish / Subscribe (PubSub) model

• Decoupling of data producer (publisher) and data consumer (subscriber) through topics (message queues)

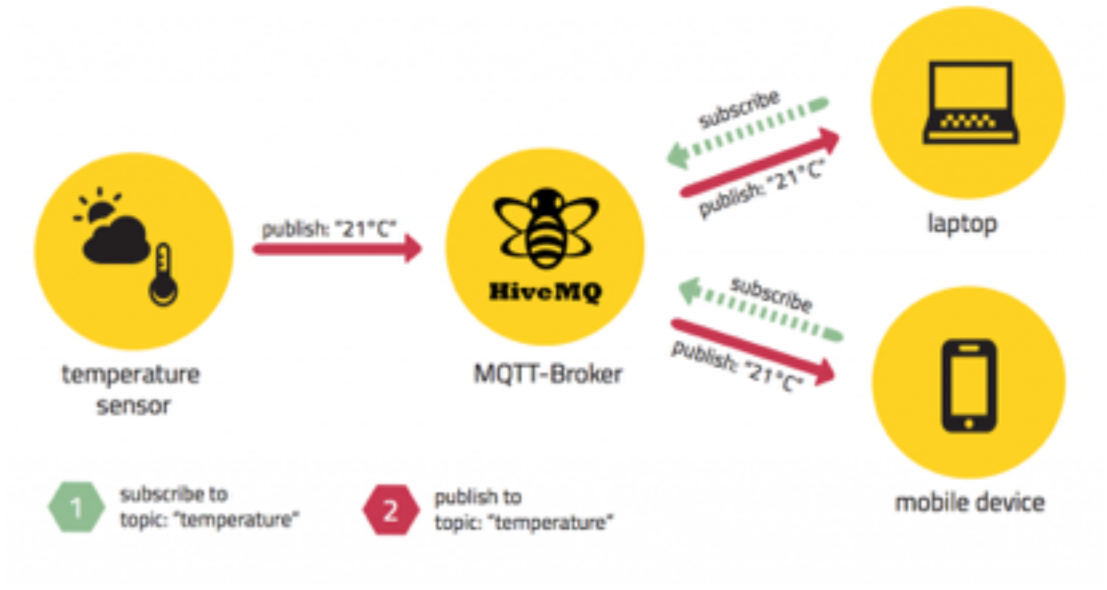
• Simple protocol, aimed at low complexity, low power and low footprint implementations (e.g. WSN - Wireless Sensor Networks)

• Runs on connection-oriented transport (TCP). To be used in conjunction with 6LoWPAN (TCP header compression)

• MQTT caters for (wireless) network disruptions

The publish/subscribe pattern

The publish/subscribe pattern (pub/sub) is an alternative to the traditional client-server model, where a client communicates directly with an endpoint. However, **Pub/Sub decouples a client, who is sending a particular message (called publisher) from another client (or more clients), who is receiving the message (called subscriber)**. This means that the publisher and subscriber don't know about the existence of one another. There is a third component, called broker, which is known by both the publisher and subscriber, which filters all incoming messages and distributes them accordingly. So let's dive into a little bit more details about the just mentioned aspects. Remember this is still the basic part about pub/sub in general, we'll talk about MQTT specifics in just a minute.



MQTT Publish / Subscribe

As already mentioned the main aspect in pub/sub is the decoupling of publisher and receiver, which can be differentiated in more dimensions:

- **Space decoupling:** Publisher and subscriber do not need to know each other (by ip address and port for example)
- **Time decoupling:** Publisher and subscriber do not need to run at the same time.
- **Synchronization decoupling:** Operations on both components are not halted during publish or receiving

In summary publish/subscribe decouples publisher and receiver of a message, through filtering of the messages it is possible that only certain clients receive certain messages. The decoupling has three dimensions: Space, Time, Synchronization.

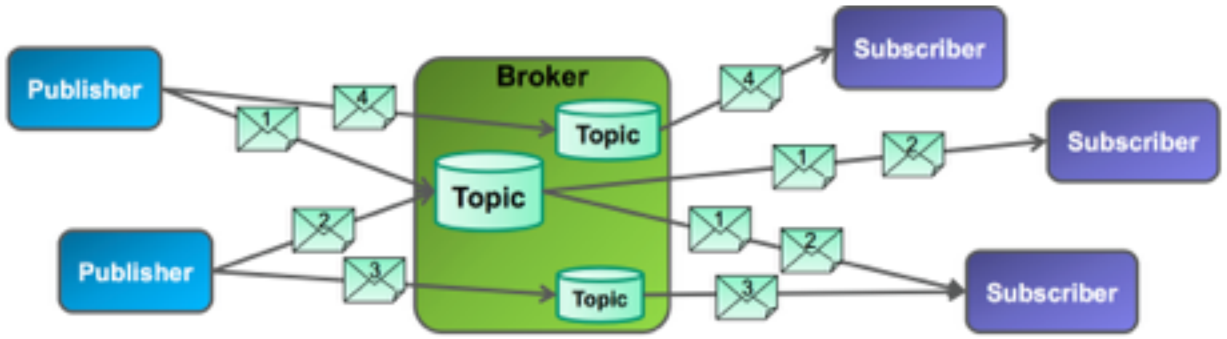
Scalability

Pub/Sub also provides a greater scalability than the traditional client-server approach. This is because operations on the broker can be highly parallelized and processed event-driven. Also often message caching and intelligent routing of messages is decisive for improving the scalability. But it is definitely a challenge to scale publish/subscribe to millions of connections. This can be achieved using clustered broker nodes in order to distribute the load over more individual servers with load balancers.

MQTT

MQTT decouples the space of publisher and subscriber. So they just have to know hostname/ip and port of the broker in order to publish/subscribe to messages. It also decouples from time, but often this is just a fall-back behavior, because the use case mostly is to delivery message in nearrealtime. Of course the broker is able to store messages for clients that are not online. (This requires two conditions: client has connected once and its session is persistent and it has subscribed to a topic with Quality of Service greater than 0). MQTT is also able to decouple the synchronization, because most client libraries are working asynchronously and are based on callbacks or similar model. So it won't block other tasks while waiting for a message or publishing a message.

MQTT uses subject-based filtering of messages. So each message contains a topic, which the broker uses to find out, if a subscribing client will receive the message or not. In order to handle the challenges of a pub/sub system in general, **MQTT has the quality of service (QoS) levels.** It is easily possible to specify that a message gets successfully delivered from the client to the broker or from the broker to a client.



Client, Broker and Connection Establishment

Client

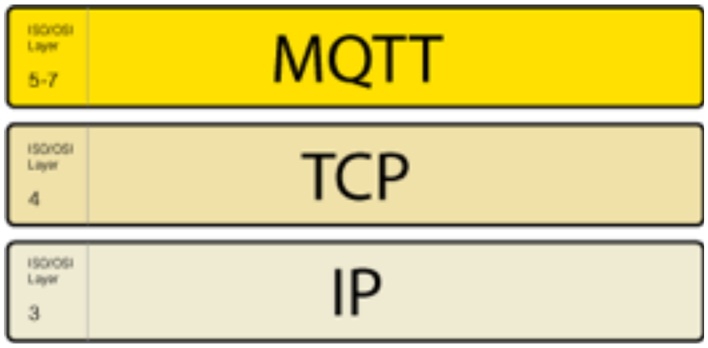
This includes publisher or subscribers, both of them label an MQTT client that is only doing publishing or subscribing. (In general a MQTT client can be both a publisher & subscriber at the same time). **A MQTT client is any device from a micro controller up to a full fledged server, that has a MQTT library running and is connecting to an MQTT broker over any kind of network.** This could be a really small and resource constrained device, that is connected over a wireless network and has a library strapped to the minimum or a typical computer running a graphical MQTT client for testing purposes, basically any device that has a TCP/IP stack and speaks MQTT over it. The client implementation of the MQTT protocol is very straight-forward and really reduced to the essence. That's one aspect, why MQTT is ideally suitable for small devices. **MQTT client libraries are available for a huge variety of programming languages, for example Android, Arduino, C, C++, C#, Go, iOS, Java, JavaScript, .NET.**

Broker

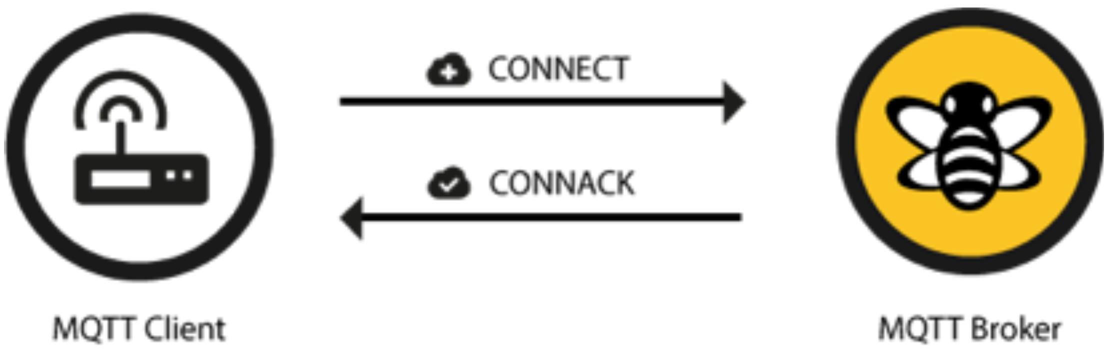
The counterpart to a MQTT client is the MQTT broker, which is the heart of any publish/subscribe protocol. Depending on the concrete implementation, a broker can handle up to thousands of concurrently connected MQTT clients. **The broker is primarily responsible for receiving all messages, filtering them, decide who is interested in it and then sending the message to all subscribed clients.** It also holds the session of all persisted clients including subscriptions and missed messages. Another responsibility of the broker is the authentication and authorization of clients. And at most of the times a broker is also extensible, which allows to easily integrate custom authentication, authorization and integration into backend systems. Especially the integration is an important aspect, because often the broker is the component, which is directly exposed on the internet and handles a lot of clients and then passes messages along to downstream analyzing and processing systems. Subscribing to all message is not really an option. All in all the broker is the central hub, which every message needs to pass. Therefore **it is important, that it is highly scalable, integratable into backend systems, easy to monitor and of course failure-resistant.**

MQTT Connection

The MQTT protocol is based on top of TCP/IP and both client and broker need to have a TCP/IP stack.



The MQTT connection itself is always between one client and the broker, no client is connected to another client directly. **The connection is initiated through a client sending a CONNECT message to the broker. The broker response with a CONNACK** and a status code. Once the connection is established, the broker will keep it open as long as the client doesn't send a disconnect command or it looses the connection.

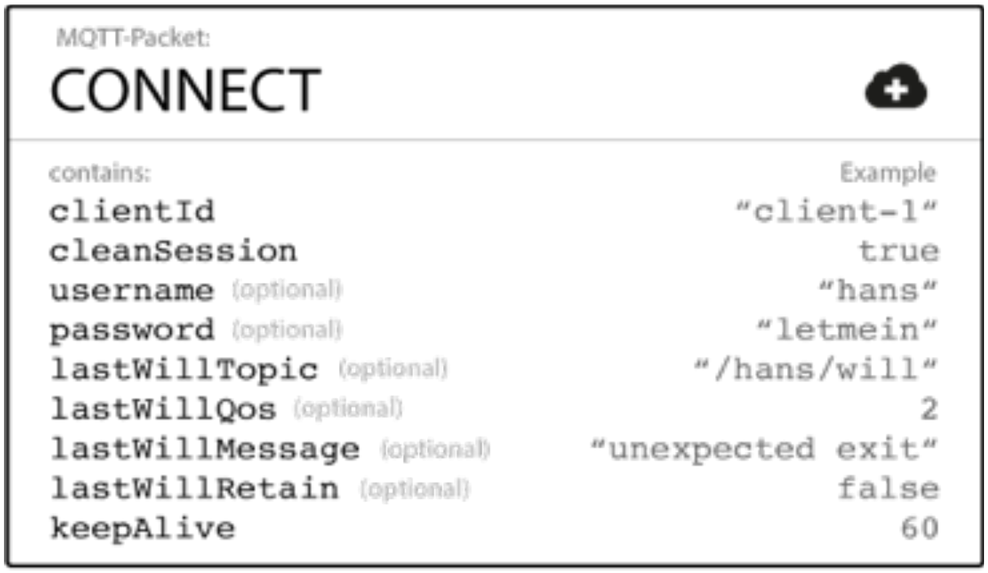


MQTT connection through a NAT

It is a common use case that MQTT clients are behind routers, which are using network address translation (NAT) in order to translate from a private network address (like 192.168.x.x, 10.0.x.x) to a public facing one. As already mentioned the MQTT client is doing the first step by sending a CONNECT message. So there is no problem at all with clients behind a NAT, because the broker has a public address and the connection will be kept open to allow sending and receiving message bidirectional after the initial CONNECT.

Client initiates connection with the CONNECT message

So let's look at the MQTT CONNECT command message. As already mentioned this is sent from the client to the broker to initiate a connection. If the CONNECT message is malformed (according to the MQTT spec) or it takes too long from opening a network socket to sending it, the broker will close the connection. This is a reasonable behavior to avoid that malicious clients can slow down the broker.
A good-natured client will send a connect message with the following content among other things:



MQTT-Packet:
CONNECT

contains:	Example
clientId	"client-1"
cleanSession	true
username (optional)	"hans"
password (optional)	"letmein"
lastWillTopic (optional)	"/hans/will"
lastWillQos (optional)	2
lastWillMessage (optional)	"unexpected exit"
lastWillRetain (optional)	false
keepAlive	60

So let's go through all these options one by one:

ClientId

The client identifier (short ClientId) is an **identifier of each MQTT client** connecting to a MQTT broker. As the word identifier already suggests, it should be unique per broker. The broker uses it for identifying the client and the current state of the client

Clean Session

The clean session flag indicates the broker, whether the **client wants to establish a persistent session or not**. A persistent session (CleanSession is false) means, that the broker will store all subscriptions for the client and also all missed messages, when subscribing with Quality of Service (QoS) 1 or 2. If clean session is set to true, the broker won't store anything for the client and will also purge all information from a previous persistent session.

Username/Password

MQTT allows to send a **username and password for authenticating the client and also authorization**. However, the password is sent in plaintext, if it isn't encrypted or hashed by implementation or TLS is used underneath

Will Message

The will message is part of the last will and testament feature of MQTT. **It allows to notify other clients, when a client disconnects ungracefully**. A connecting client will provide his will in form of an MQTT message and topic in the CONNECT message. If this clients gets disconnected ungracefully, the broker sends this message on behalf of the client.

KeepAlive

The keep alive is a time interval, the clients commits to by sending regular PING Request messages to the broker. The broker response with PING Response and this mechanism will allow both sides to determine if the other one is still alive and reachable. We'll talk about this in detail in a future post.

Broker responds with the CONNACK message

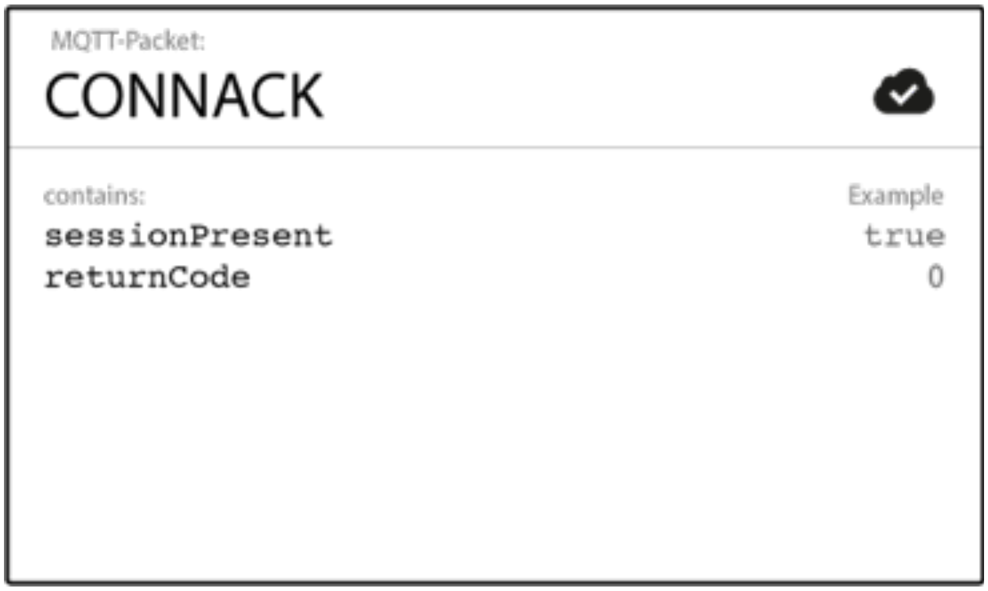
When a broker obtains a CONNECT message, it is obligated to respond with a CONNACK message. The CONNACK contains only two data entries: session present flag, connect return code.

Session Present flag

The **session present flag indicate, whether the broker already has a persistent session of the client from previous interactions**. If a client connects and has set CleanSession to true, this flag is always false, because there is no session available. If the client has set CleanSession to false, the flag is depending on, if there are session information available for the ClientId. If stored session information exist, then the flag is true and otherwise it is false. This flag was added newly in MQTT 3.1.1 and helps the client to determine, whether it has to subscribe to topics or if these are still stored in his session.

Connect acknowledge flag

The second flag in the CONNACK is the connect acknowledge flag. It signals the client, if the **connection attempt was successful and otherwise what the issue is**.



In the following table you see all return codes at a glance.

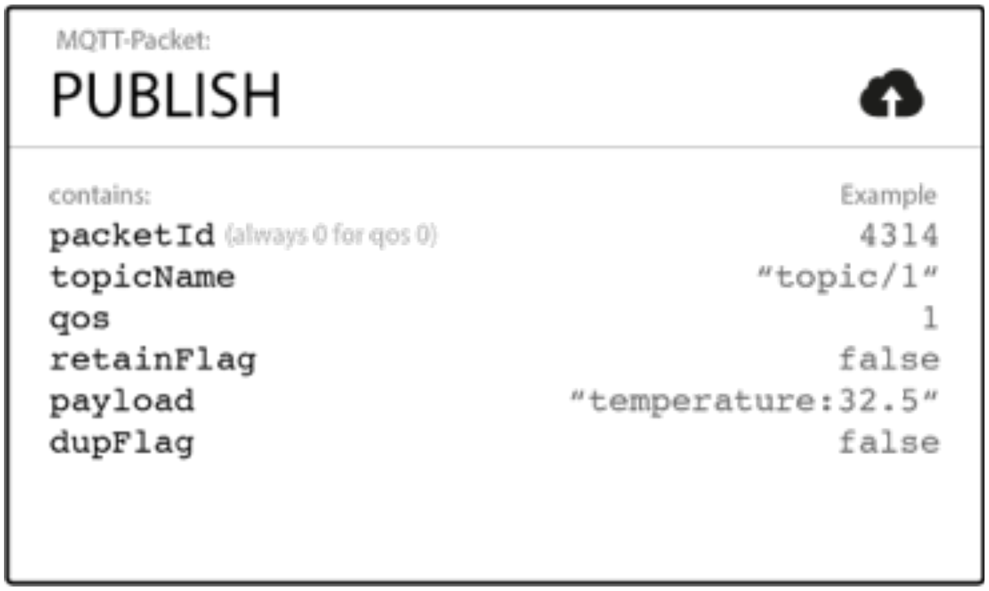
Return Code	Return Code Response
0	Connection Accepted
1	Connection Refused, unacceptable protocol version
2	Connection Refused, identifier rejected
3	Connection Refused, Server unavailable
4	Connection Refused, bad user name or password
5	Connection Refused, not authorized

MQTT Publish, Subscribe & Unsubscribe

1.Publish

After a MQTT client is connected to a broker, it can publish messages. MQTT has a topic-based filtering of the messages on the broker, so **each message must contain a topic, which will be used by the broker to forward the message to interested clients. Each message**

typically has a payload which contains the actual data to transmit in byte format. MQTT is data-agnostic and it totally depends on the use case how the payload is structured. It's completely up to the sender if it wants to send binary data, textual data or even full-fledged XML or JSON. A MQTT publish message also has some more attributes, which we're going discuss in detail:



MQTT-Packet:

PUBLISH

contains:	Example
packetId (always 0 for qos 0)	4314
topicName	"topic/1"
qos	1
retainFlag	false
payload	"temperature:32.5"
dupFlag	false

Topic Name
A simple string, which is hierarchically structured with forward slashes as delimiters. An example would be "myhome/livingroom/temperature" or "Germany/Munich/Octoberfest/people".

QoS
A Quality of Service Level (QoS) for this message. The level (0,1 or 2) determines the guarantee of a message reaching the other end (client or broker).

Retain-Flag
This flag determines if the message will be saved by the broker for the specified topic as last known good value. New clients that subscribe to that topic will receive the last retained message on that topic instantly after subscribing. More on retained messages and best practices in one of the next posts.

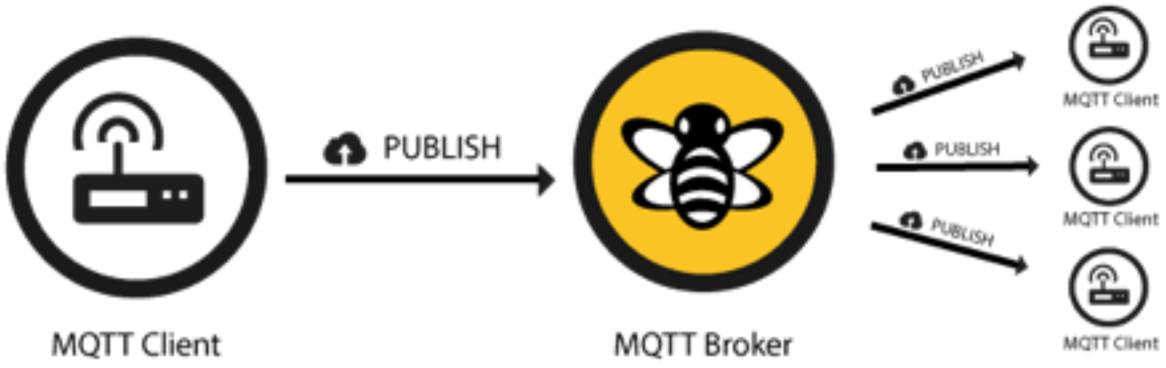
Payload
This is the actual content of the message. MQTT is totally data-agnostic, it's possible to send images, texts in any encoding, encrypted data and virtually every data in binary.

Packet Identifier
The packet identifier is a unique identifier between client and broker to identify a message in a message flow. This is only relevant for QoS greater than zero. Setting this MQTT internal identifier is the responsibility of the client library and/or the broker.

DUP flag
The duplicate flag indicates, that this message is a duplicate and is resent because the other end

didn't acknowledge the original message. This is only relevant for QoS greater than 0. This re-send/duplicate mechanism is typically handled by the MQTT client library or the broker as an implementation detail.

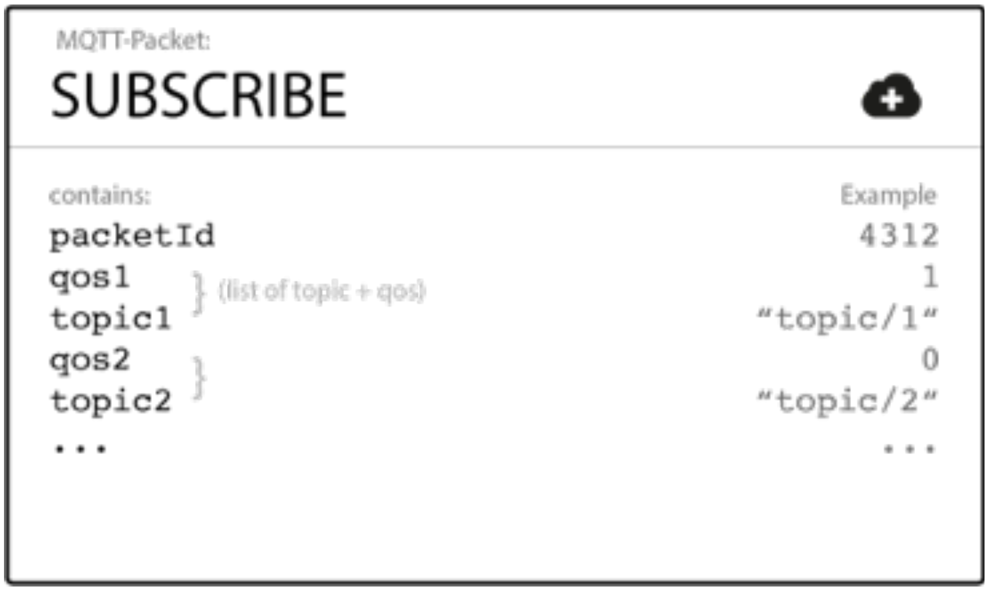
So when a client sends a publish to a MQTT broker, **the broker will read the publish, acknowledge the publish if needed (according to the QoS Level) and then process it**. Processing includes determining which clients have subscribed on that topic and then sending out the message to the selected clients which subscribe to that topic.



The client, who initially published the message is only concerned about delivering the publish message to the broker. From there on it is the responsibility of the broker to deliver the message to all subscribers. The publishing client doesn't get any feedback, if someone was interested in this published message and how many clients received the message by the broker.

2. Subscribe

Publishing messages doesn't make sense if no one ever receives the message, or, in other words, if there are no clients subscribing to any topic. A client needs to send a SUBSCRIBE message to the MQTT broker in order to receive relevant messages. A subscribe message is pretty simple, it just contains a unique packet identifier and a list of subscriptions.

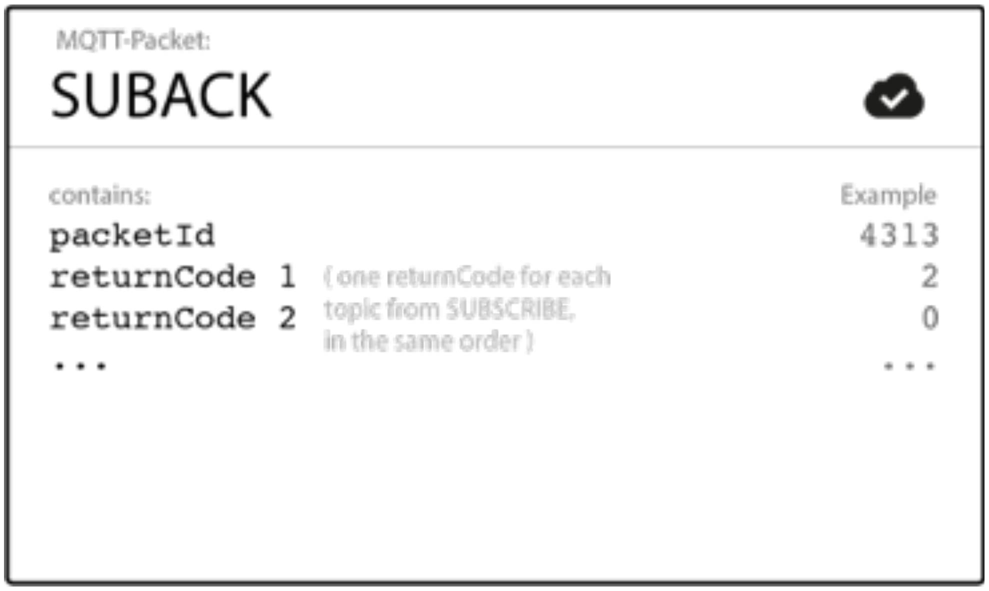


Packet Identifier
The packet identifier is a unique identifier between client and broker to identify a message in a message flow. This is only relevant for QoS greater than zero. Setting this MQTT internal identifier is the responsibility of the client library and/or the broker.

List of Subscriptions
A SUBSCRIBE message can contain an arbitrary number of subscriptions for a client. Each subscription is a pair of a topic topic and QoS level. The topic in the subscribe message can also contain wildcards, which makes it possible to subscribe to certain topic patterns. If there are overlapping subscriptions for one client, the highest QoS level for that topic wins and will be used by the broker for delivering the message.

3. Suback

Each subscription will be confirmed by the broker through sending an acknowledgment to the client in form of the SUBACK message. This message contains the same packet identifier as the original Subscribe message (in order to identify the message) and a list of return codes.



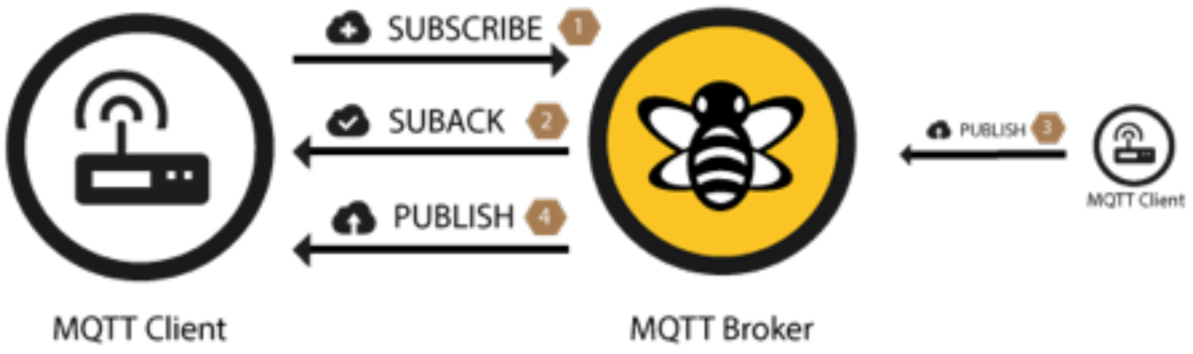
Packet Identifier

The packet identifier is a unique identifier used to identify a message. It is the same as in the SUBSCRIBE message.

Return Code

The broker sends one return code for each topic/QoS-pair it received in the SUBSCRIBE message. So if the SUBSCRIBE message had 5 subscriptions, there will be 5 return codes to acknowledge each topic with the QoS level granted by the broker. If the subscription was prohibited by the broker (e.g. if the client was not allowed to subscribe to this topic due to insufficient permission or if the topic was malformed), the broker will respond with a failure return code for that specific topic.

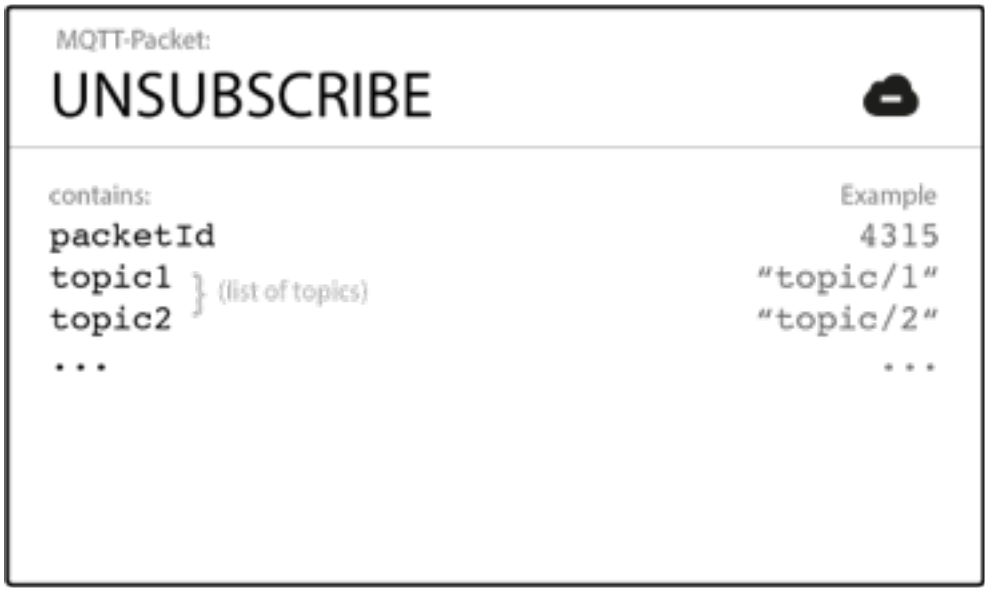
Return Code	Return Code Response
0	Success – Maximum QoS 0
1	Success – Maximum QoS 1
2	Success – Maximum QoS 2
128	Failure



After a client successfully sent the SUBSCRIBE message and received the SUBACK message, it will receive every published message matching the topic of the subscription.

4. Unsubscribe

The counterpart of the SUBSCRIBE message is the UNSUBSCRIBE message which deletes existing subscriptions of a client on the broker. The UNSUBSCRIBE message is similar to the SUBSCRIBE message and also has a packet identifier and a list of topics.


MQTT-Packet:
UNSUBSCRIBE
contains:
Example
(list of topics)

packetId	4315
topic1	"topic/1"
topic2	"topic/2"
...	...

Packet Identifier

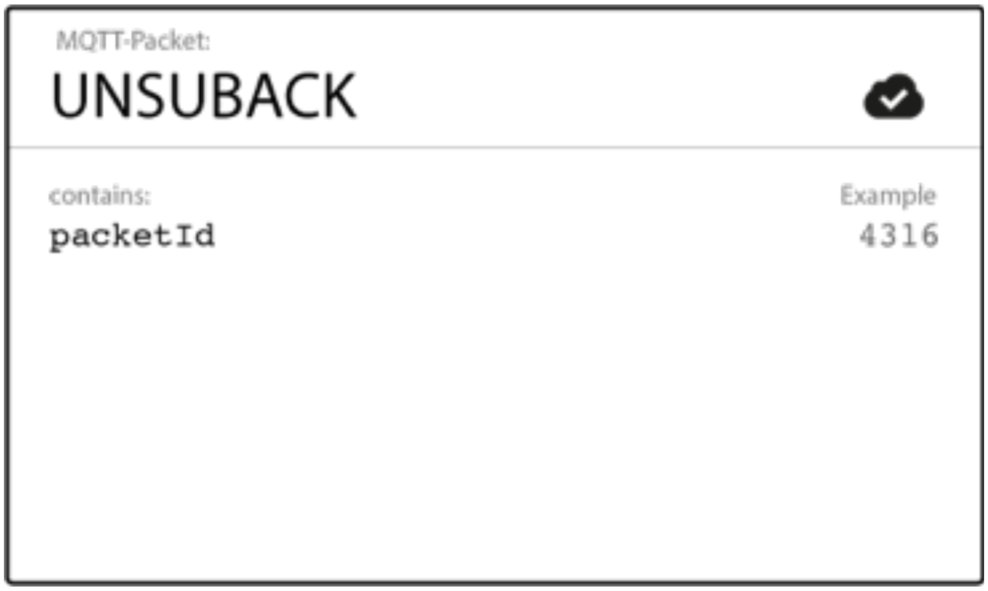
The packet identifier is a unique identifier used to identify a message. The acknowledgement of an UNSUBSCRIBE message will contain the same identifier.

List of Topic

The list of topics contains an arbitrary number of topics, the client wishes to unsubscribe from. It is only necessary to send the topic as string (without QoS), the topic will be unsubscribed regardless of the QoS level it was initially subscribed with.

Unsuback

The broker will acknowledge the request to unsubscribe with the UNSUBACK message. This message only contains a packet identifier.



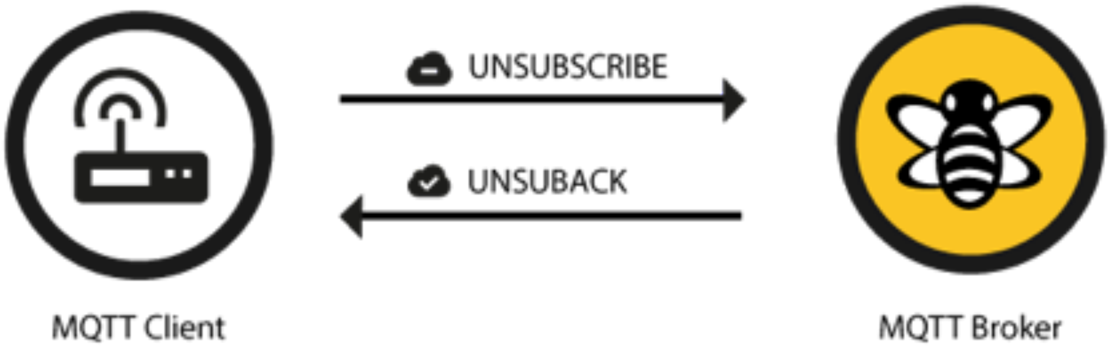
MQTT-Packet:

UNSUBACK

contains:	Example
packetId	4316

Packet Identifier

The packet identifier is a unique identifier used to identify a message. It is the same as in the UNSUBSCRIBE message.



After receiving the UNSUBACK from the broker, the client can assume the subscriptions in the UNSUBSCRIBE message are deleted.

MQTT Topics

Topics/Subscriptions

Messages in MQTT are published on topics. There is no need to configure a topic, publishing on it is enough. Topics are treated as a hierarchy, using a slash (/) as a separator. This allows sensible arrangement of common themes to be created, much in the same way as a filesystem. For example, multiple computers may all publish their hard drive temperature information on the following topic, with their own computer and hard drive name being replaced as appropriate:

- sensors/COMPUTER_NAME/temperature/HARDDRIVE_NAME

Clients can receive messages by creating subscriptions. A subscription may be to an explicit topic, in which case only messages to that topic will be received, or it may include wildcards. Two wildcards are available, `+` or `#`.

`+` can be used as a wildcard for a single level of hierarchy. It could be used with the topic above to get information on all computers and hard drives as follows:

- sensors/+/temperature/+

As another example, for a topic of "a/b/c/d", the following example subscriptions will match:

- a/b/c/d
- +/b/c/d
- a/+/c/d
- a/+/+/d
- +/+/+/+

The following subscriptions will not match:

- a/b/c
- b/+/c/d
- +/+/+

`#` can be used as a wildcard for all remaining levels of hierarchy. This means that it must be the final character in a subscription. With a topic of "a/b/c/d", the following example subscriptions will match:

- a/b/c/d
- #
- a/#
- a/b/#
- a/b/c/#
- +/b/c/#

Zero length topic levels are valid, which can lead to some slightly non-obvious behaviour. For example, a topic of "a//topic" would correctly match against a subscription of "a/+/topic". Likewise, zero length topic levels can exist at both the beginning and the end of a topic string, so "/a/topic" would match against a subscription of "+/a/topic", "#" or "/#", and a topic "a/topic/" would match against a subscription of "a/topic/+" or "a/topic/#".

MQTT Features

1. Quality of Service 0, 1 & 2

What is Quality of Service?

The **Quality of Service** (*QoS*) level is an agreement between sender and receiver of a message regarding the guarantees of delivering a message. There are 3 QoS levels in MQTT:

- *At most once* (0)
- *At least once* (1)
- *Exactly once* (2).

When talking about QoS there are always two different parts of delivering a message: publishing client to broker and broker to subscribing client. We need to look at them separately since there are subtle differences. The QoS level for publishing client to broker is depending on the QoS level the client sets for the particular message. When the broker transfers a message to a subscribing client it uses the QoS of the subscription made by the client earlier. That means, QoS guarantees can get downgraded for a particular receiving client if subscribed with a lower QoS.

Why is Quality of Service important?

QoS is a major feature of MQTT, it makes communication in unreliable networks a lot easier because the protocol handles retransmission and guarantees the delivery of the message, regardless how unreliable the underlying transport is. Also it empowers a client to choose the QoS level depending on its network reliability and application logic.

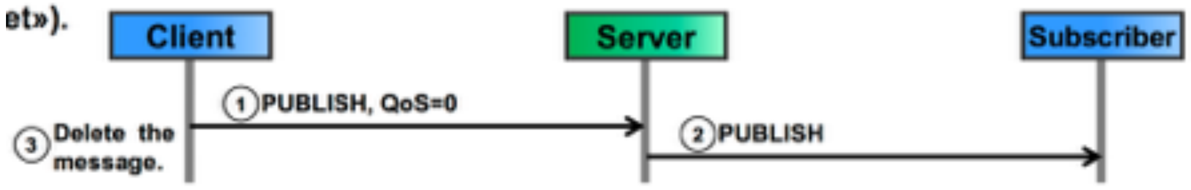
How does it work?

So how is the quality of service implemented in the MQTT protocol ? We will look at each level one by one and explain the functionality.

QOS 0 – AT MOST ONCE

The minimal level is zero and it guarantees a best effort delivery. A message won't be acknowledged by the receiver or stored and redelivered by the sender. This is often called "fire and forget" and provides the same guarantee as the underlying TCP protocol.

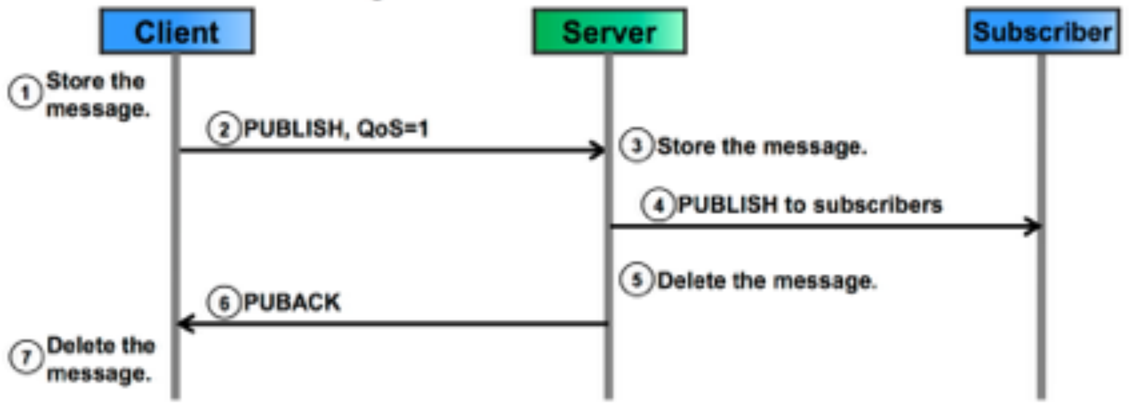
Example application: Temperature sensor data which is regularly published. Loss of an individual value is not critical since applications (consumers of the data) will anyway integrate the values over time and loss of individual samples is not relevant.



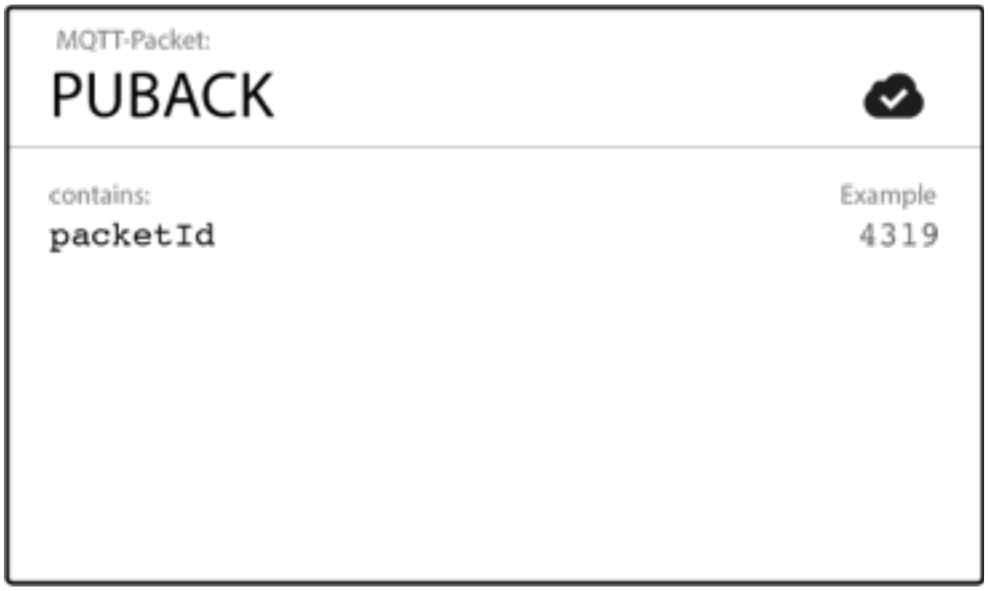
QOS 1 – AT LEAST ONCE

When using QoS level 1, it is guaranteed that a message will be delivered at least once to the receiver. But the message can also be delivered more than once.

Example application: A door sensor senses the door state. It is important that door state changes (closed->open, open->closed) are published losslessly to subscribers (e.g. alarming function). Applications simply discard duplicate messages by evaluating the message ID field.



The sender will store the message until it gets an acknowledgement in form of a PUBACK command message from the receiver.


MQTT-Packet:

PUBACK

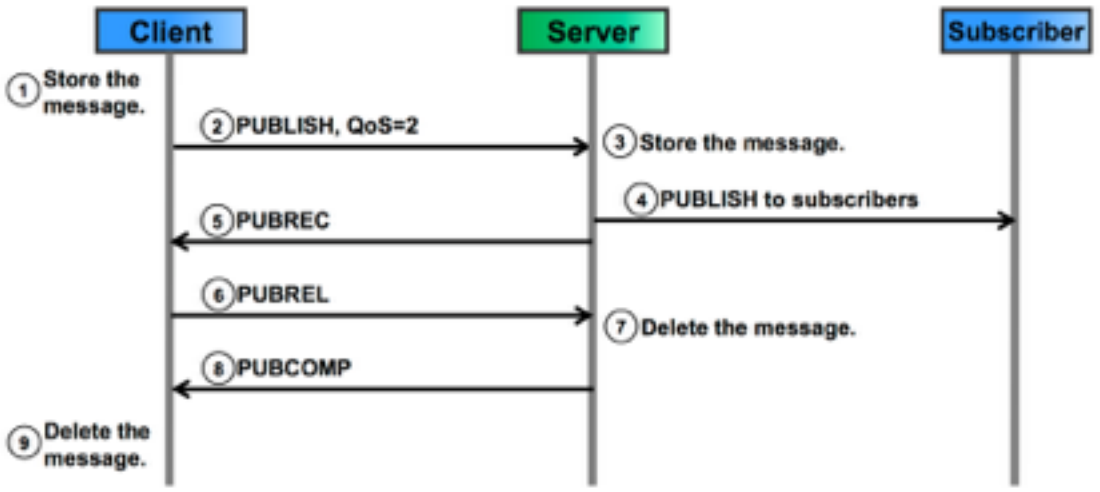
contains:	Example
packetId	4319

The association of PUBLISH and PUBACK is done by comparing the packet identifier in each packet. If the PUBACK isn't received in a reasonable amount of time the sender will resend the PUBLISH message. If a receiver gets a message with QoS 1, it can process it immediately, for example sending it to all subscribing clients in case of a broker and then replying with the PUB-ACK.

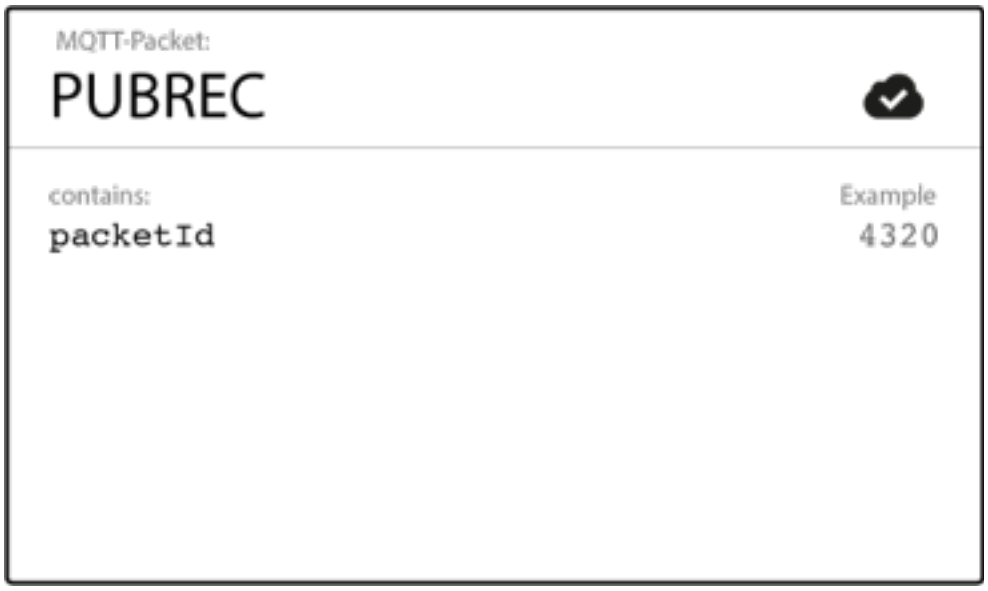
The duplicate (DUP) flag, which is set in the case a PUBLISH is redelivered, is only for internal purposes and won't be processed by broker or client in the case of QoS 1. The receiver will send a PUBACK regardless of the DUP flag.

QOS 2 - EXACTLY-ONCE DELIVERY

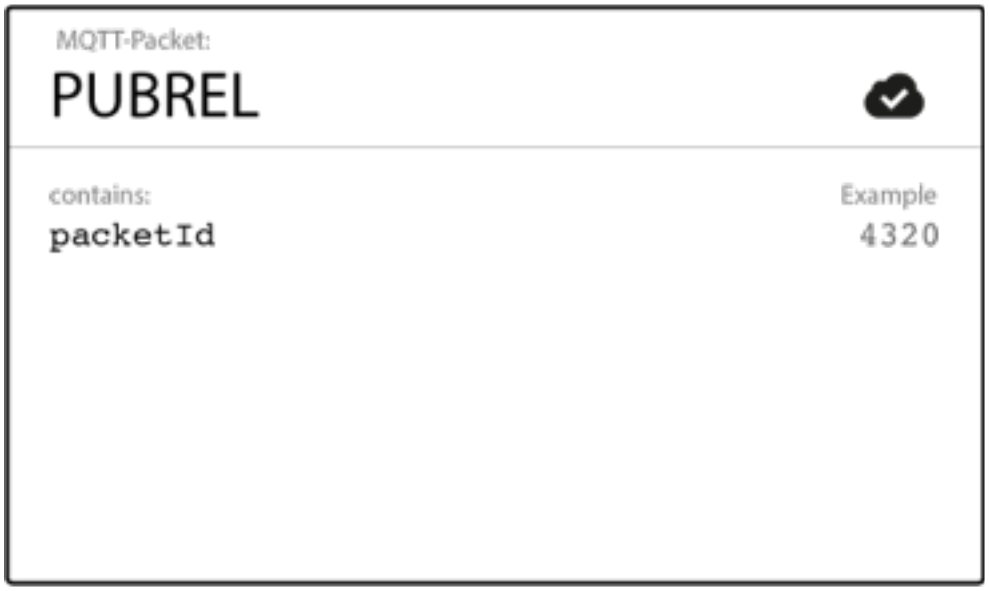
Exactly-once is a combination of at-least-once and at-most-once delivery guarantee.

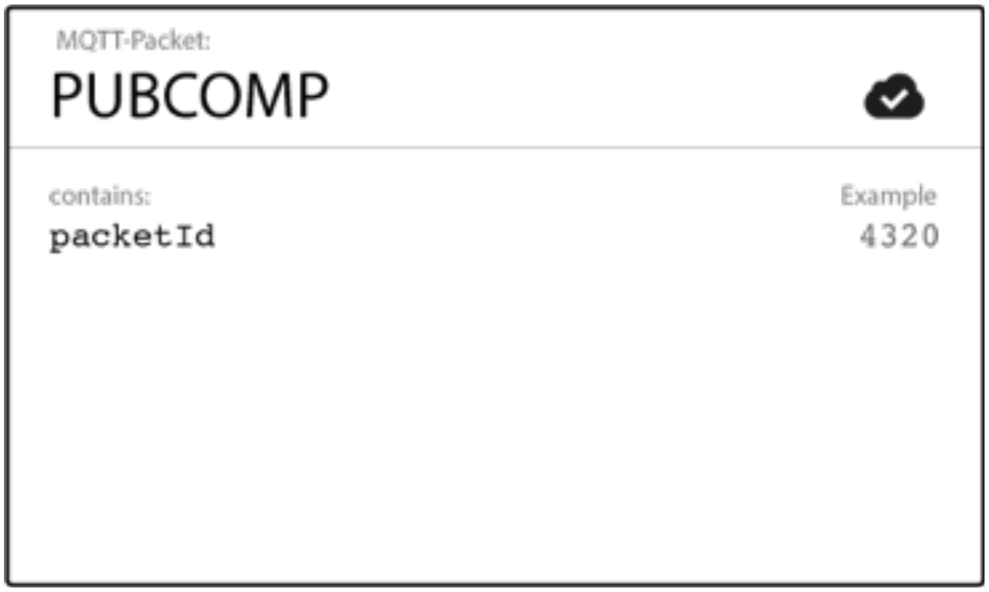


The message is always delivered exactly once. The message must be stored locally at the sender and the receiver until it is processed. QoS=2 is the safest, but slowest mode of transfer. It takes at least two pairs of transmissions between the sender and receiver before the message is deleted from the sender. The message can be processed at the receiver after the first transmission. In the first pair of transmissions, the sender transmits the message and gets acknowledgment(PUBREC) from the receiver that it has stored the message. If the sender does not receive an acknowledgment, the message is sent again with the DUP flag set until an acknowledgment is received.



In the second pair of transmissions, the sender tells the receiver that it can complete processing the message, PUBREL. If the sender does not receive an acknowledgment(PUBCOMP) of the PUBREL message, the PUBREL message is sent again until an acknowledgment is received. The sender deletes the message it saved when it receives the acknowledgment to the PUBREL message.





The receiver can process the message in the first or second phases, provided that it does not re-process the message. If the receiver is a broker, it publishes the message to subscribers. If the receiver is a client, it delivers the message to the subscriber application. The receiver sends a completion message back to the sender that it has finished processing the message.

This level could be used, for example, with billing systems where duplicate or lost messages could lead to incorrect charges being applied.

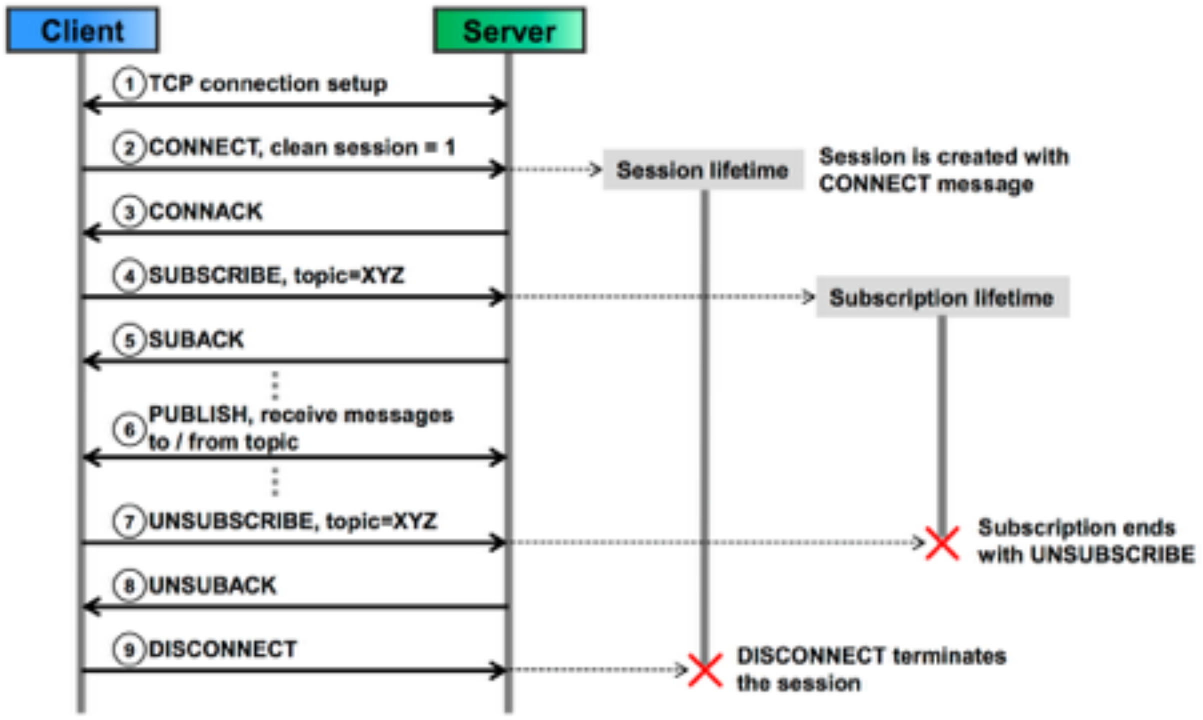
2. Persistent Session and Queuing Messages

Persistent Session

When a client connects to a MQTT broker, it needs to create subscriptions for all topics that it is interested in in order to receive messages from the broker. On a reconnect these topics are lost and the client needs to subscribe again. This is the normal behavior with no persistent session. But for constrained clients with limited resources it would be a burden to subscribe again each time they lose the connection. So a persistent session saves all information relevant for the client on the broker. The session is identified by the *clientId* provided by the client on connection establishment.

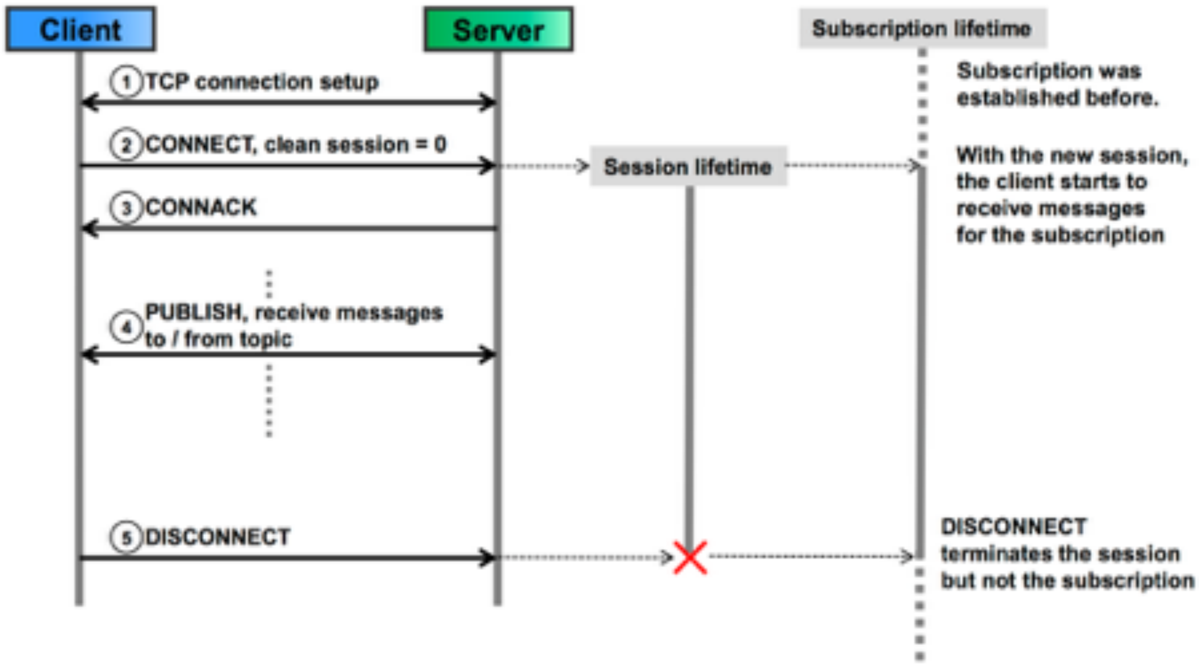
CONNECT and SUBSCRIBE message sequence (1/2)

Case 1: Session and subscription setup with clean session flag = 1 («transient» subscription)


Client
Server
1 TCP connection setup
2 CONNECT, clean session = 1
Session lifetime
Session is created with CONNECT message
3 CONNACK
4 SUBSCRIBE, topic=XYZ
Subscription lifetime
5 SUBACK
6 PUBLISH, receive messages to / from topic
7 UNSUBSCRIBE, topic=XYZ
Subscription ends with UNSUBSCRIBE
8 UNSUBACK
9 DISCONNECT
DISCONNECT terminates the session

CONNECT and SUBSCRIBE message sequence (2/2)

Case 2: Session and subscription setup with clean session flag = 0 («durable» subscription)


Client
Server
Subscription lifetime
1 TCP connection setup
Subscription was established before.
2 CONNECT, clean session = 0
Session lifetime
With the new session, the client starts to receive messages for the subscription
3 CONNACK
4 PUBLISH, receive messages to / from topic
5 DISCONNECT
DISCONNECT terminates the session but not the subscription

How to start/end a persistent session?

A persistent session can be requested by the client on connection establishment with the broker. The client can control, if the broker stores the session using the *cleanSession* flag If the clean session is set to true then the client does not have a persistent session and all information are lost when the client disconnects for any reason. When clean session is set to false, a persistent session is created and it will be preserved until the client requests a clean session again. If there is already a session available then it is used and queued messages will be delivered to the client if available.

How does the client know if there is already a session stored?

Since MQTT 3.1.1, the *CONNACK* message from the broker contains the *session present flag*, which indicates to the client if there is a session available on the broker.

Retained Messages

A retained message is a normal MQTT message with the retained flag set to true. The broker will store the last retained message and the corresponding QoS for that topic. Each client that subscribes to a topic pattern, which matches the topic of the retained message, will receive the message immediately after subscribing. For each topic only one retained message will be stored by the broker.

The subscribing client doesn't have to match the exact topic, it will also receive a retained message if it subscribes to a topic pattern including wildcards. For example client A publishes a retained message to *myhome/livingroom/temperature* and client B subscribes to *myhome/#* later on, client B will receive this retained message directly after subscribing. Also the subscribing client can identify if a received message was a retained message or not, because the broker sends out retained messages with the retained flag still set to true. A client can then decide on how to process the message.

So retained messages can help newly subscribed clients to get a status update immediately after subscribing to a topic and don't have to wait until a publishing clients send the next update.

In other words a retained message on a topic is the *last known good value*, because it doesn't have to be the last value, but it certainly is the last message with the retained flag set to true.

It is important to understand that a retained message has nothing to do with a persistent session of any client, which we covered in the last episode. Once a retained message is stored by the broker, the only way to remove it is explained below.

Send a retained message

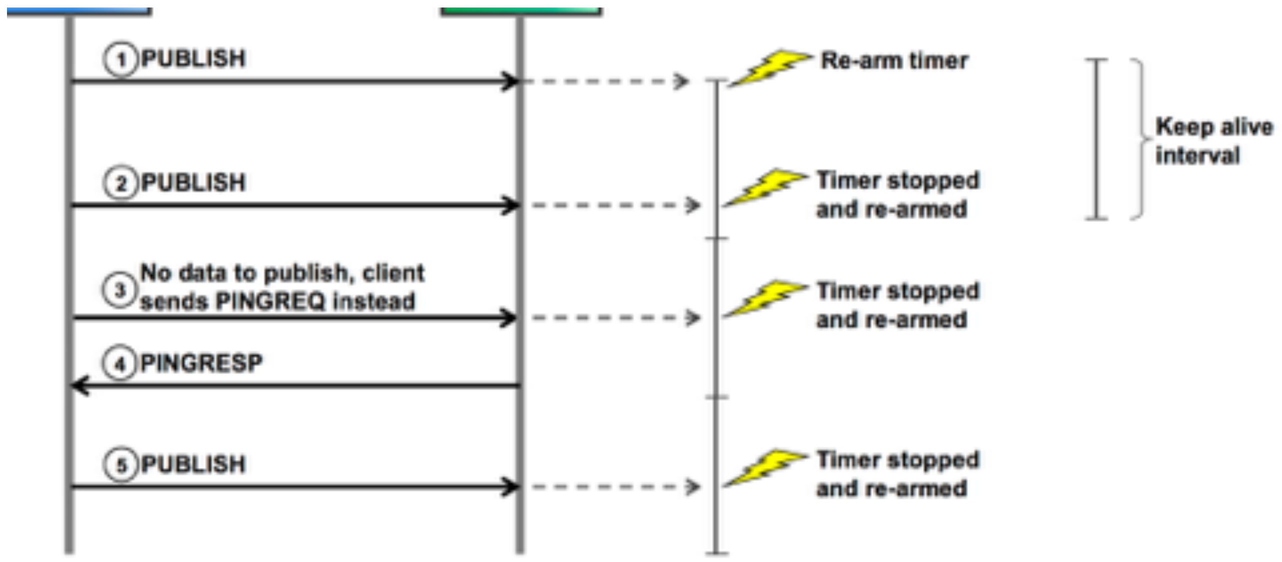
Sending a retained message from the perspective of a developer is quite simple and straight-forward. You just need to set the *retained flag* of a MQTT publish message to true. Each client library typically provides an easy way to do that.

Delete a retained message

There is also a very simple way for deleting a retained message on a topic: Just send a retained message with a zero byte payload on that topic where the previous retained message should be deleted. The broker deletes the retained message and all new subscribers won't get a retained message for that topic anymore. Often deleting is not necessary, because each new retained message will overwrite the last one.

Keep Alive and Client Take-Over

As we already know MQTT is based on TCP and that includes a certain guarantee that packets over the internet are transferred "reliable, ordered and error-checked". Nevertheless it can happen that one of the communicating parties gets out of sync with the other, often due to a crash of



1 PUBLISH
Re-arm timer
Keep alive interval
2 PUBLISH
Timer stopped and re-armed
3 No data to publish, client sends PINGREQ instead
Timer stopped and re-armed
4 PINGRESP
5 PUBLISH
Timer stopped and re-armed

one side or because of transmission errors. This state is called a half-open connection. The important point is that the still functioning end is not notified about the failure of the other side and is still trying to send messages and wait for acknowledgements.

Although TCP/IP in theory notifies you when a socket breaks, in practice, particularly on things like mobile and satellite links, which often "fake" TCP over the air and put headers back on at each end, it's quite possible for a TCP session to "black hole", i.e. it appears to be open still, but in fact is just dumping anything you write to it onto the floor.

MQTT Keep Alive timer

In order to solve the problem of half-open connection, the keep alive timer defines the maximum allowable time interval between client messages. The timer is used by the server to check client's connection status.

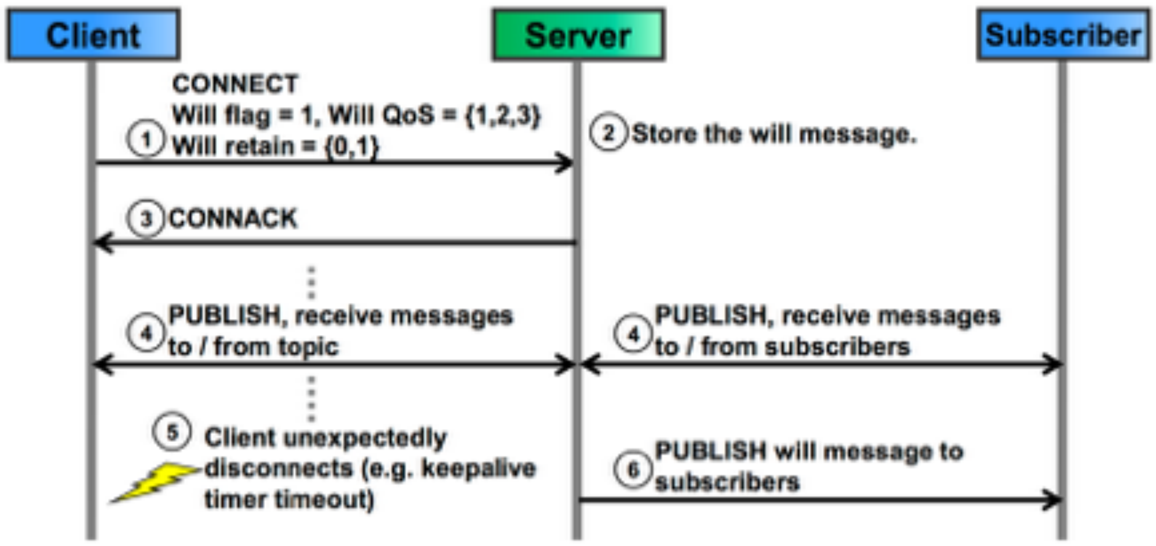
After 1.5 * keepalive-time is elapsed, the server disconnects the client (client is granted a grace period of an additional 0.5 keepalive-time).

In the absence of data to be sent, the client sends a PINGREQ message instead.

Typical value for keepalive timer are a couple of minutes.

Last Will and Testament

The Last Will and Testament (LWT) feature is used in MQTT to notify other clients about an ungracefully disconnected client. Each client can specify its last will message (a normal MQTT message with topic, retained flag, QoS and payload) when connecting to a broker. The broker

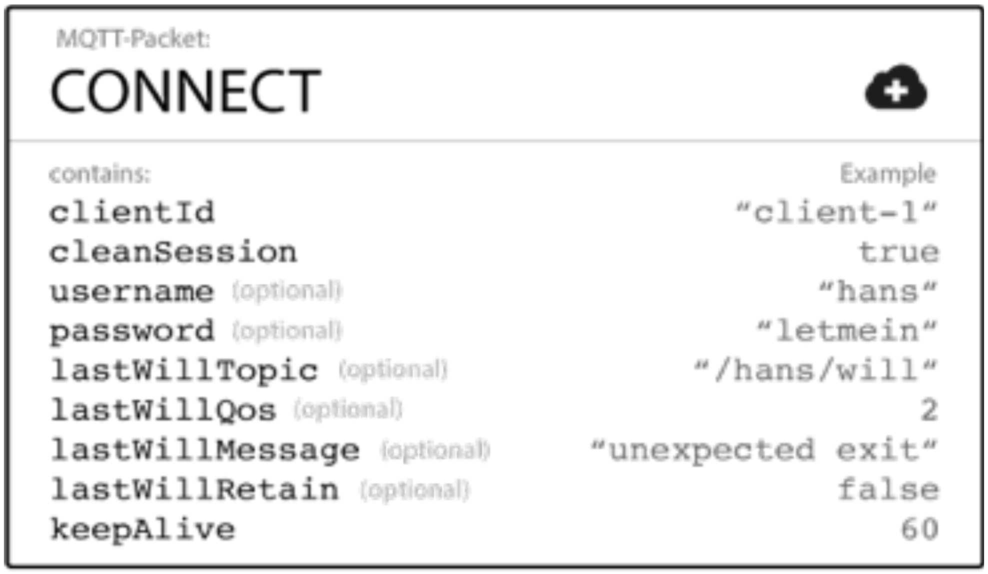

Client
Server
Subscriber
CONNECT
Will flag = 1, Will QoS = {1,2,3}
1 Will retain = {0,1}
2 Store the will message.
3 CONNACK
4 PUBLISH, receive messages to / from topic
4 PUBLISH, receive messages to / from subscribers
5 Client unexpectedly disconnects (e.g. keepalive timer timeout)
6 PUBLISH will message to subscribers

will store the message until it detects that the client has disconnected ungracefully. If the client disconnect abruptly, the broker sends the message to all subscribed clients on the topic, which was specified in the last will message. The stored LWT message will be discarded if a client disconnects gracefully by sending a DISCONNECT message.

LWT helps to implement strategies when the connection of a client drops or at least to inform other clients about the offline status.

How to specify a LWT message for a client?

The LWT message can be specified by each client as part of the CONNECT message, which serves as connection initiation between client and broker.



MQTT-Packet:

CONNECT

contains:	Example
`clientId`	“client-1“
`cleanSession`	true
`username` (optional)	“hans“
`password` (optional)	“letmein“
`lastWillTopic` (optional)	“/hans/will“
`lastWillQos` (optional)	2
`lastWillMessage` (optional)	“unexpected exit“
`lastWillRetain` (optional)	false
`keepAlive`	60

When will a broker send the LWT message?

According to the MQTT 3.1.1 specification the broker will distribute the LWT of a client in the following cases:

- An I/O error or network failure is detected by the server.
- The client fails to communicate within the Keep Alive time.
- The client closes the network connection without sending a DISCONNECT packet first.
- The server closes the network connection because of a protocol error.