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.

Interconnecting Smart Objects with IP. DOI: 10.1016/B978-0-12-375165-2.00017-X

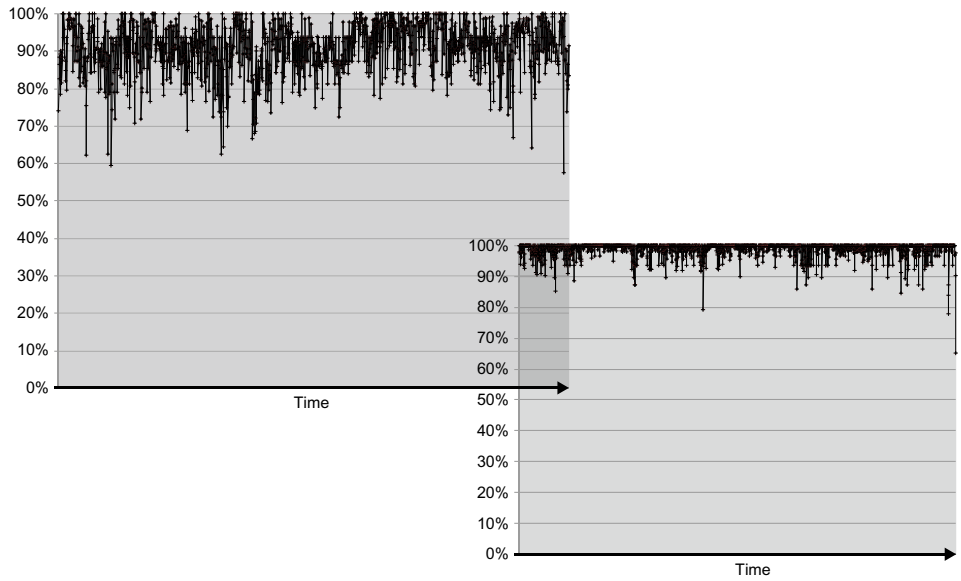

FIGURE 17.1

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.

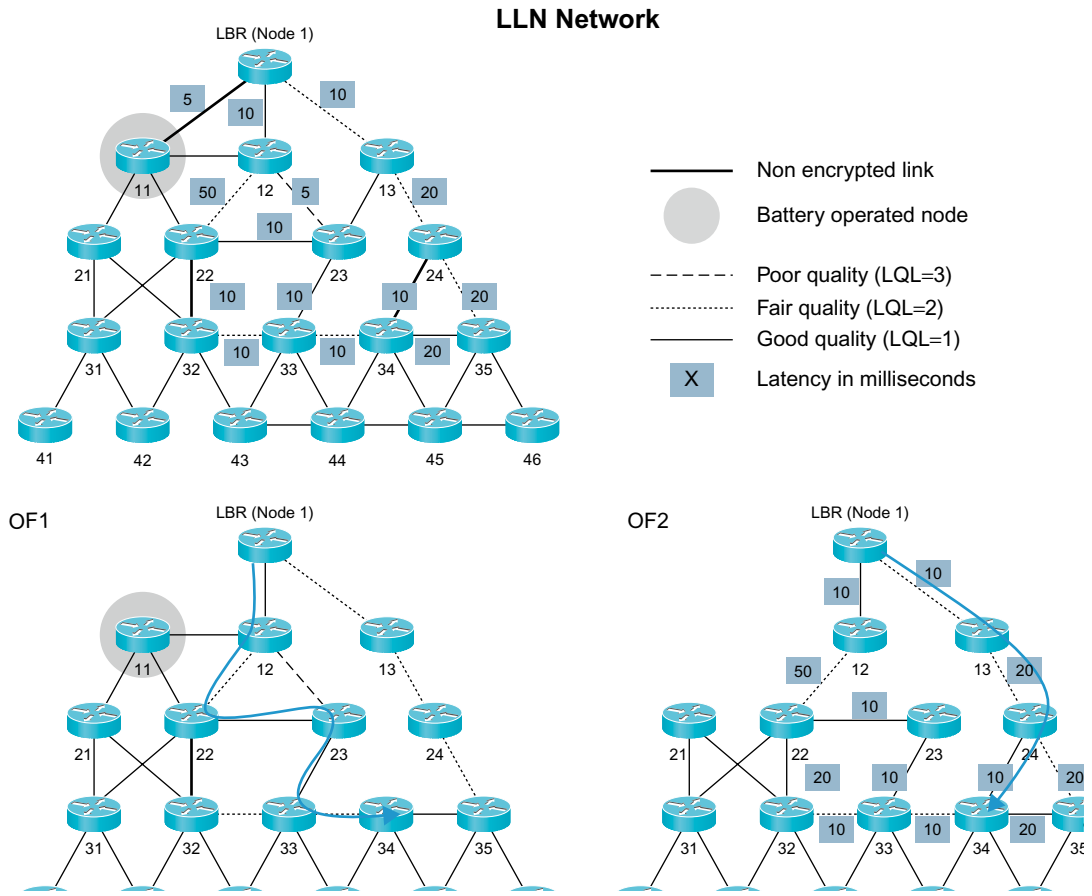

FIGURE 17.2

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

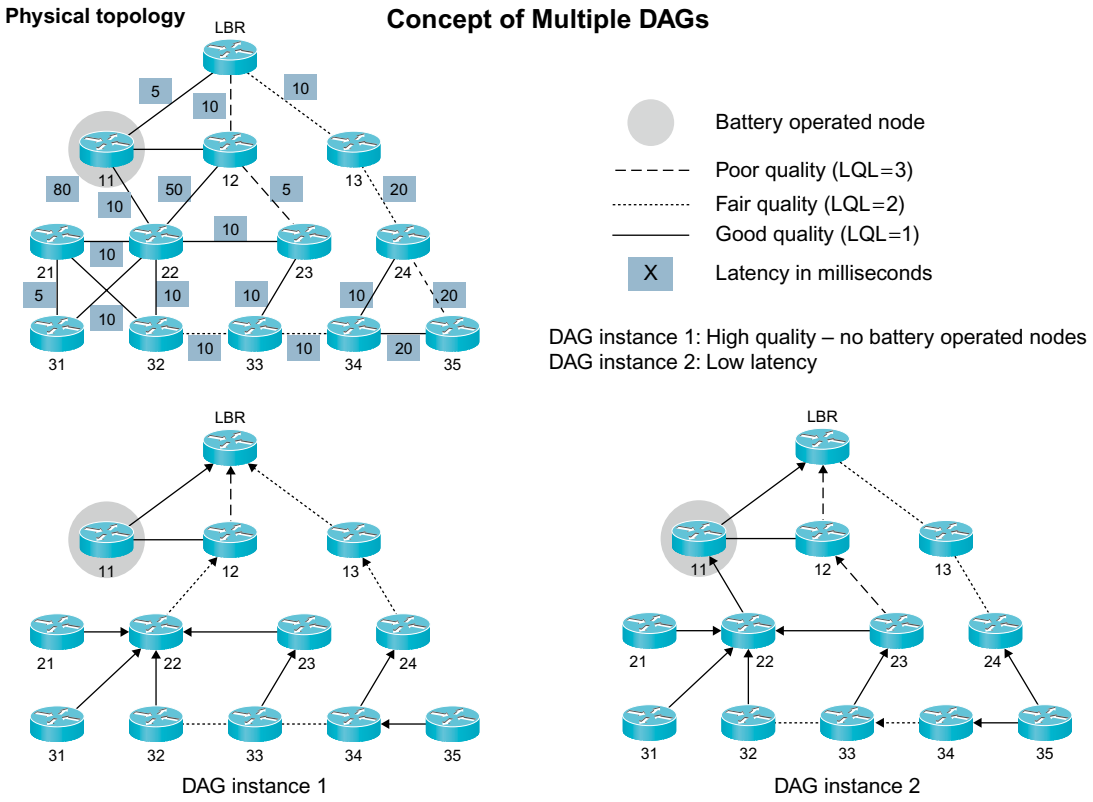

FIGURE 17.3

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).

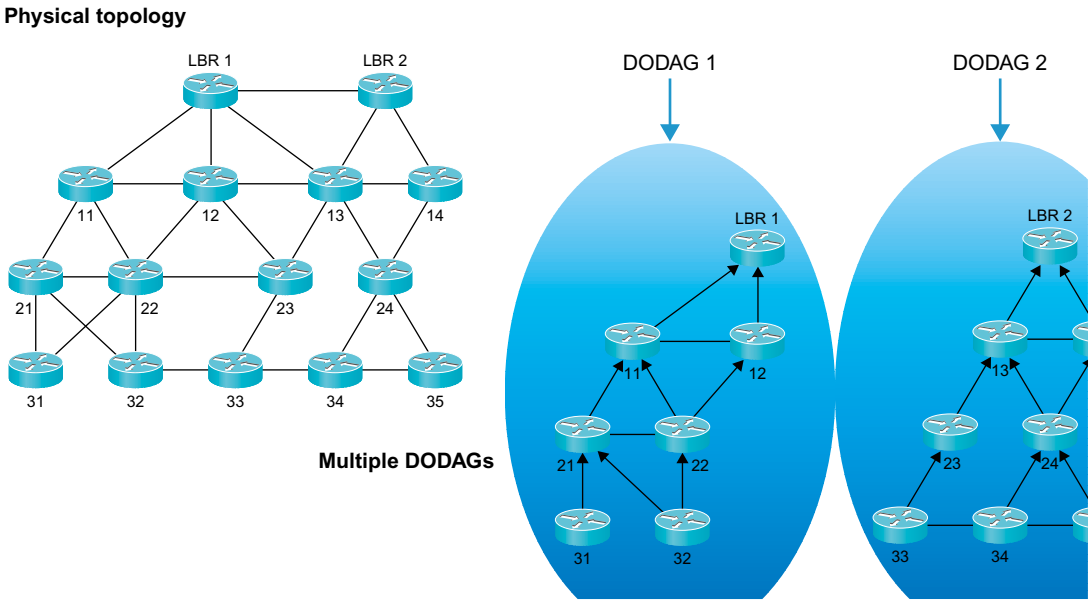

FIGURE 17.4

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.

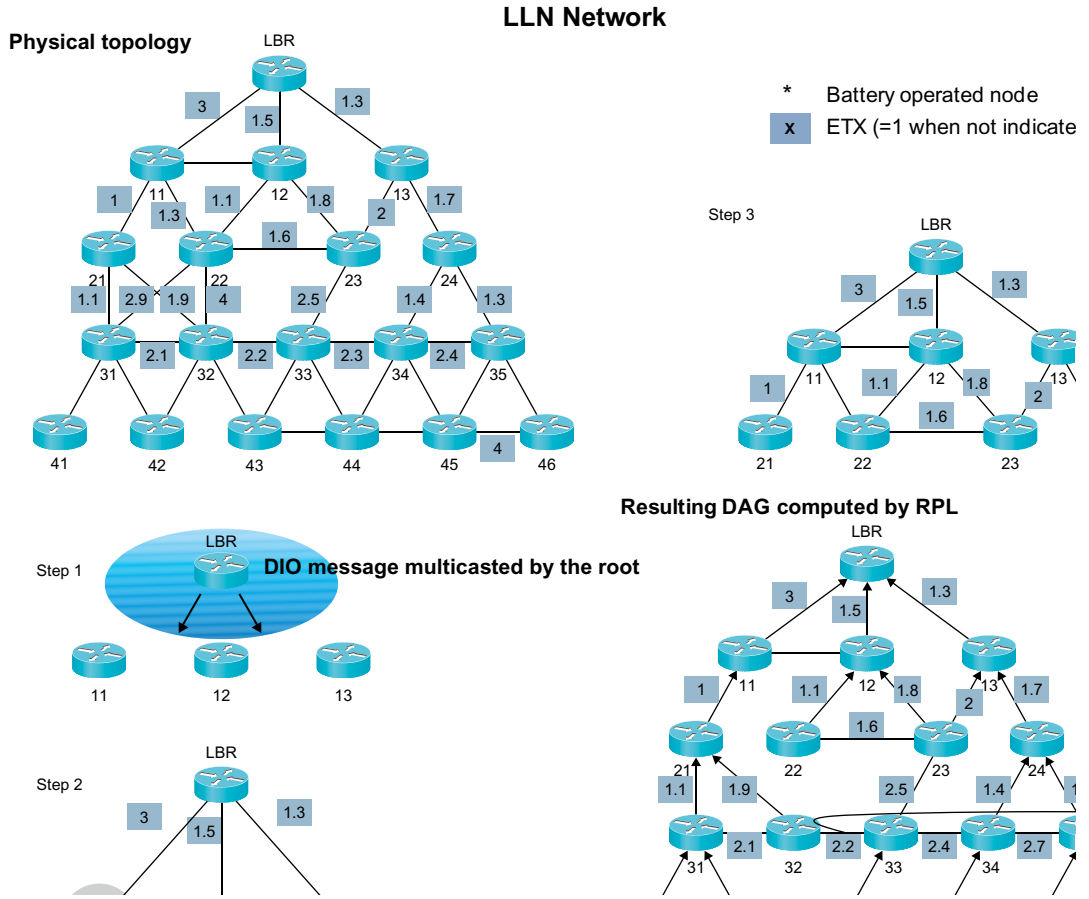

FIGURE 17.5

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

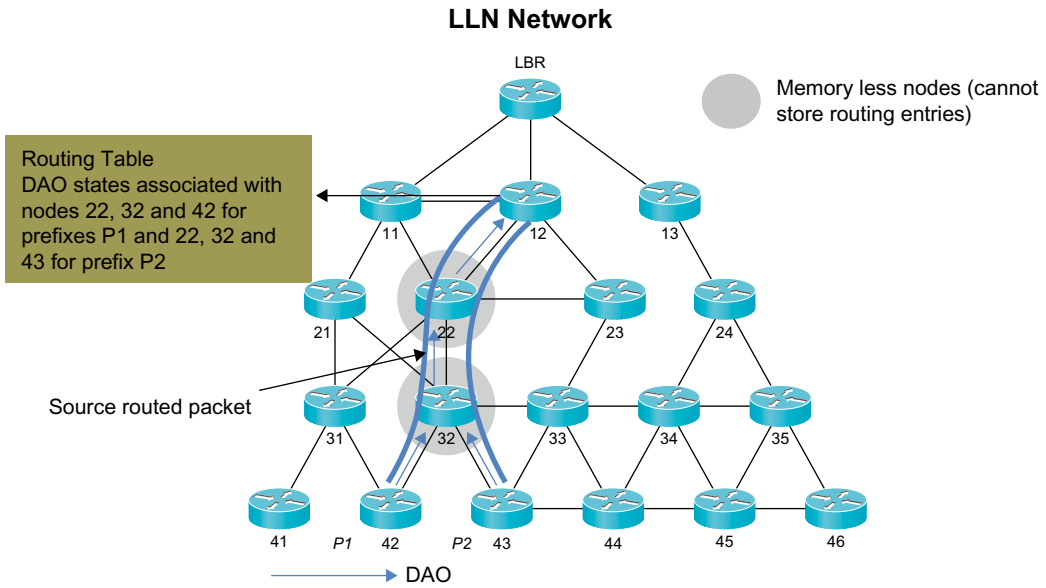

FIGURE 17.6

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.

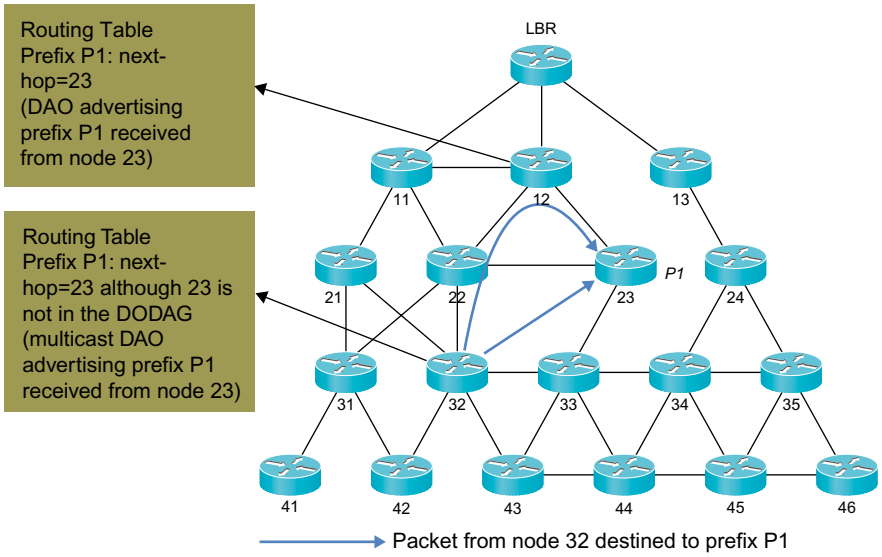

FIGURE 17.7

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).

The exact location where these flags are carried is not yet defined (e.g., flow label, existing, or even new IPv6 extended header). The idea is to set a flag in the packet header that is used to verify that the packet is making forward progress in order to detect loops, or to detect a DODAG inconsistency.

For example, when a packet is rerouted to a sibling, a flag is set in the packet header to indicate that the packet has been forwarded to a sibling. When it reaches the next hop, if the packet has to be forwarded again to another sibling because there is no available link toward the root, then the packet is dropped. In its current revision, RPL allows for a one-hop sibling path (only 1 bit is used) since it is believed that in most cases a one-hop sibling will provide a viable path to the root but that a single bit could be extended to a counter. The idea is to limit the number of hops along a sibling path to avoid sibling loops. Similarly, DAO loops can be detected by using a "down" bit. When a packet is sent in the down direction, the bit is set. Upon receiving a packet with the "down" bit set, if the routing table of the node indicates to send it in the up direction, the DODAG is inconsistent (there may be a loop) and the packet may be discarded. Such inconsistency triggers the resetting of the DIO trickle timers. As further optimization, the child that has received the packet in error can send it back to the parent with an "error" bit set to trigger the cleanup of the route by the parent that will in turn send the packet again to another child or sibling. That process allows recursive routing table cleanup. The same mechanism could be used for other types of loop detection and routing table cleanup.

17.6.8 Global and Local Repair

Repair mechanisms are key components of routing protocols. As the network topology changes because of link and node failures or link/node metric changes, it is imperative to dynamically update the routing decision to adapt to topological changes. To that end, various mechanisms have been defined to rebuild the DODAG upon network topology changes. The first case to handle is *DODAG repair* when a network element (e.g., such as a link or a node) fails. RPL must then rebuild a new DODAG according to the new topology. Repairs must be handled with care in lossy environments to avoid rebuilding a DODAG upon a transient failure, since rebuilding a DODAG has a global impact on the network and nodes resources. Overreacting would potentially compromise routing stability.

RPL specifies two complementary repair mechanisms: a *global* and a *local* repair technique. There are many other routing protocols that use local repair strategies to quickly find an alternate path (which may momentarily not be optimal) deferring the global repair of the entire topology. This is the approach taken by RPL: when a link is considered nonviable and an alternate path must be found (as opposed to being a transient failure that does not require any action), the node triggers a local repair to quickly find an alternate path, even if the alternate path is not optimum (local optimum). Then in a second step, which may be deferred, the DODAG is rebuilt for all the nodes in the network (global optimum).

- Local repair: To quickly find an alternate path when the most preferred path or all other alternate paths are no longer available with a minimal attempt to find an optimal path.
- Max_depth rule: A node cannot advertise a rank less than or equal to any of its parents. It may advertise a rank lower than in a previous advertisement if the node has jumped in the DODAG to improve its position. The max_depth rule also states that within a DODAG iteration a node must not advertise a rank deeper than L+DAGMaxRankIncrease, where L is the lowest rank that the node has advertised within the DODAG iteration. Note that 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.

Although this has already been discussed, let's re-explain why such as rule was introduced: one of the main risks when joining a parent is to be on the path of that node to the DODAG root (in other words, to attach to one of a children). Should that happen, a loop would be formed and the rank would then continuously increment until reaching the "infinite" value for the rank at which point the nodes would detach from each other. This is also referred to as the "count-to-infinity" problem that also exists in other distance vector routing protocols because with this type of routing protocol a node does not have global visibility of the network topology. Thus the idea is to introduce a mechanism that reduces the number of iterations of successive increments, in other words, avoid waiting to count until "infinity." A node triggering a local repair is allowed to choose as a parent a node whose rank is less than L+DAGMaxRankIncrease where L is the lowest rank value that has been advertised within the DODAG iteration. Once again, the DODAG root may decide to set the DAGMaxRankIncrease value to 0. If at some point the rank of the node exceeds L+DAGMaxRankIncrease, the rank is considered equal to infinity and the loop is broken. This mechanism is illustrated using Figure 17.8.

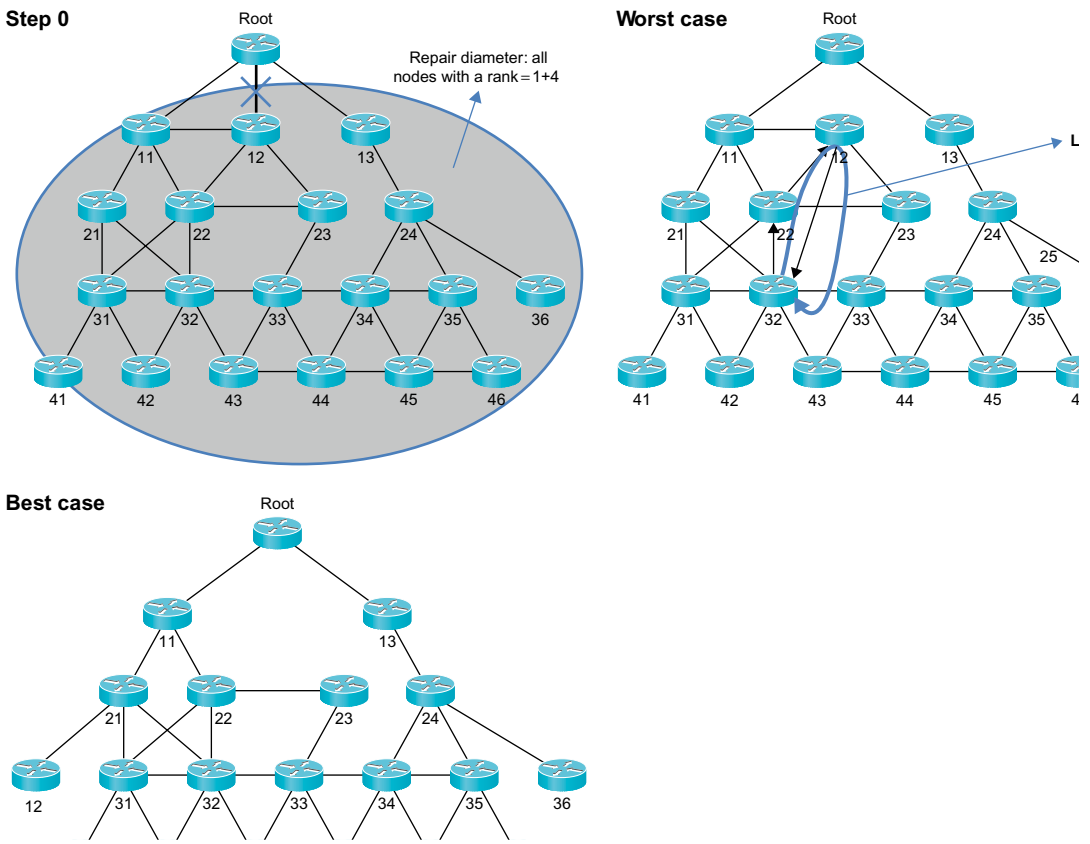

FIGURE 17.8

Illustration of the use of the DAGMaxRankIncrease value.

Suppose that the link between node 12 and the root fails and DAGMaxRankIncrease = 5. In this example suppose that node 12 has a rank of 2. That means node 12 can join any node with a maximum rank of (rank_of_node 12) + 5 = 7. This includes all nodes in the network (in this simple example). In the best case, node 12 selects a node that does not belong to its sub-DODAG and no loop is formed (e.g., if node 12 decides to join node 21). Let's now suppose that node 12 attaches to a node in its sub-DODAG, say node 32 (note that the line between node 12 and 32 is oriented in the 12->32 direction). What happens next is that node 12 sends an updated DIO reflecting its new rank 5. Node 22 in turn updates its rank to 6, node 32 updates its rank to 7, and node 12 updates its rank to 8, which exceeds the maximum allowed value. At this point the loop will be broken and the node will detach. This illustrates how the use of the DAGMaxRankIncrease avoids counting to infinity (0xFFFF).

In addition, RPL has defined another mechanism known as "poisoning," which is useful when performing local repair while trying to avoid loops.

The poison-and-wait mechanism considers the situation of a node running out of parents after a network element failure. According to the local policy the node may simply decide to root a new "floating" DODAG (in this case the G-bit of the DIO must be cleared) after having set its rank to 1 (it is the DODAG root) and the DODAGPreference (the node may decide to lower its preference). Alternatively, the node may decide to try to rejoin the DODAG by selecting a new parent. According to the RPL rules it cannot join a node if that makes its rank higher than L+DAGMaxRankIncrease in the DODAG iteration that it has left. Note that if the DAGMaxRankIncrease value is set to 0 by the DODAG root, the node cannot join any node that would increase its rank. The "poisoning" mechanism sends a poisoning DIO message to all children to be removed as a parent and trigger a new parent selection so the node is not an ancestor of any of the nodes in its sub-DODAG. This mechanism is illustrated in Figure 17.9.

Suppose link 24-13 fails. Node 24 does not have any alternate parent or sibling. In this case, it resets its trickle timer to trigger the sending of a new DIO, and upon expiration of the trickle timer it sends a DIO with Rank = Infinite (value = 0xFFFF). As the new DIO travels in the sub-DODAG, nodes act to potentially select another parent. For example, node 36 becomes isolated, node 35 starts using node 23 as a new parent, so does node 34, etc. The end result is that the former children of node 24 no longer use node 24 as an ancestor. Note that an implementation may choose to send multiple DIO poisoning messages should one of them get lost. After the expiration of a local timer (to give a better chance for all nodes in the sub-DODAG to change their next-hop decision), it becomes safe for node 24 to call the OF and select a new parent *regardless of its rank* as long as the max_depth rule is respected. The poisoning message may be lost resulting in attaching to a child, which may lead to a loop (but the max_depth rule would avoid counting to infinity). Step 2 in Figure 17.9 shows node 24 then joining the DODAG via node 34 before advertising its new rank.

⚠ Although the poisoning node advertises an infinite rank, it retains its original rank to be compliant with the max_depth rule exposed earlier.

The poisoning approach is not "guaranteed"; the poisoning DIO may be lost resulting in loop formation that could be broken faster because of the max-depth rule without having to count to infinity.

Global repair is achieved by RPL when the DODAG root generates a new DODAGSequenceNumber. As the DIO messages are propagated down the DODAG, each node detects the new DODAG SequenceNumber, the OF function is reevaluated, and nodes potentially select new parents.

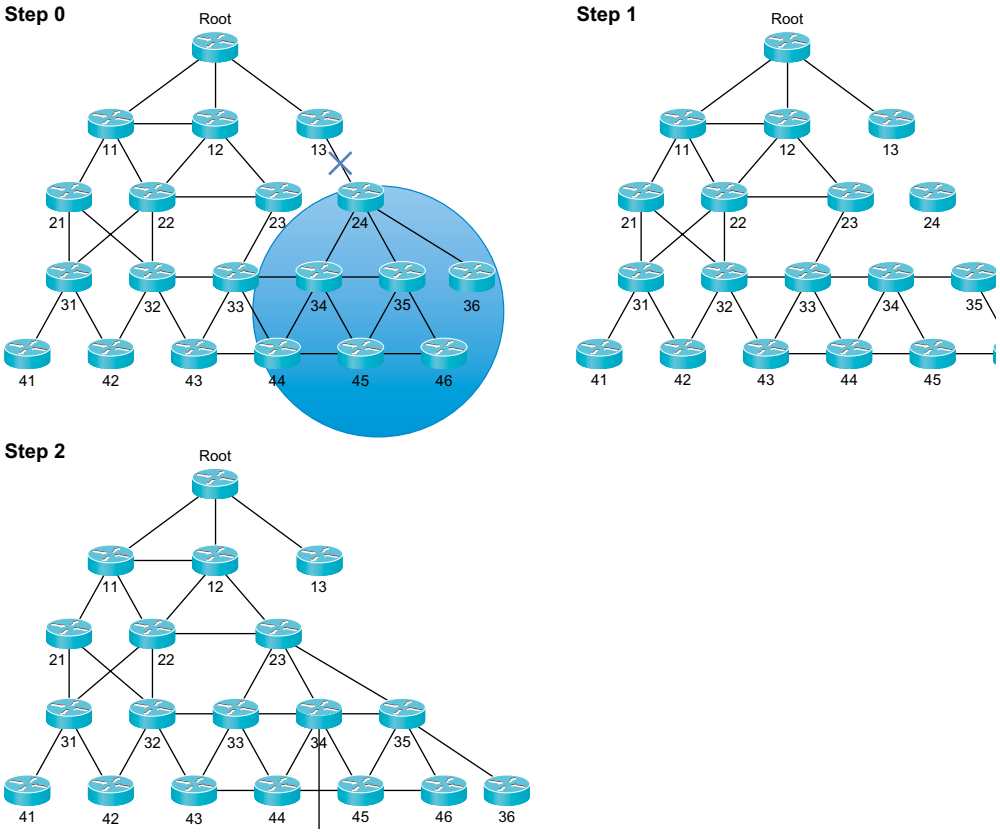

FIGURE 17.9

Illustration of the Poison-And-Wait Approach.

Note: This allows bypassing the RPL rule that states a first node must not process the DIO and select as a parent a second node that would result in the first node increasing its rank above L+DAGMaxRankIncrease. If a node using the old DODAGSequenceNumber receives a DIO with a new DODAGSequenceNumber from a second node with a rank too high according to the max_depth rule, there is no risk that the second node lies in the sub-DODAG of the first node, because the second node is in the new DODAG iteration and, consequently, there is no risk of loop. Thus the DODAG is recomputed entirely according to the OF creating an entirely new DODAG iteration. Such a global repair is not only used to effectively "repair" a DODAG but also to reoptimize it. Indeed, once a node has selected a parent, it continues to ignore DIO from other nodes in its current iteration resulting in an increase in its rank above L+DAGMaxRankIncrease. But what if one of these nodes effectively advertises a more optimal route according to the OF? That better path would then be ignored until a new DODAGSequenceNumber was originated by the DODAG root. Thus the global repair mechanism is not only used to repair a DODAG but also to reoptimize it.

Global repair rebuilds the DODAG. As such it is not only used as a repair mechanism but also a reoptimization technique for the DODAG. It requires extra cost regarding control traffic and is driven

by the root. Mechanisms could be added to request the DODAG root to trigger the global repair. Still, local repair is useful since the effect is localized and may occur more rapidly. As with any distance vector protocol, the risk of reattaching anywhere in the DODAG is forming loops. Thus the max_depth rule has been defined to limit the impact of forming loops, and to avoid counting to infinity should a loop be formed. Additionally, RPL supports the poisoning mechanism triggered by a node with no parent to avoid any node in its sub-DODAG to use it as an ancestor. At this point it would be safe for the node to locally repair by joining any node regardless of its rank as long as the max_depth rule is honored for that DODAG iteration. Finally, the ability of forming a floating DODAG upon losing connectivity with parents, in an attempt to preserve inner connectivity between a set of nodes in the network, is supported by RPL.

17.6.9 Routing Adjacency with RPL

Routing adjacency in RPL definitely deserves its own section. With routing protocols such as OSPF or IS-IS a routing adjacency between two neighbors is established once the neighbors have exchanged and agreed on various routing protocol parameters (e.g., protocol version, frequency of hellos, dead-timers) and once the LSDBs have been synchronized. From that point, routing adjacencies are maintained thanks to the exchange of "hello" packets sent every X seconds (X being configurable). If no hello is received after n * X seconds (n configurable) then the routing adjacency is considered as down, which triggers a routing protocol convergence. The situation in LLNs is radically different since the exchange of "hellos" between nodes would drain energy from the nodes as well as potentially cause congestion on limited bandwidth links in LLNs, which is highly undesirable when energy and bandwidth are scarce resources.

The approach taken by RPL recommends using a probing technique based on IPv6 Neighbor Discovery (ND); namely sending IPv6 solicitations messages (see Chapter 15). The use of ND implies neighbor reachability verification when data traffic is to be sent. The routing adjacency is then considered valid upon receiving a neighbor advertisement message with the "solicited" flag set. Other probing techniques could also be used. Alternatively and/or additionally other types of active probing are used according to the network characteristics and design.

If the most preferred parent is temporarily unavailable, then the node forwards the packet to an alternate parent (if available). In the absence of an alternate parent the node selects a sibling (if there is a sibling available).

Some implementations may choose to use algorithms to keep track of the number of recorded failed probes within a specific time window. It is important not only to consider the percentage of failed probes but also the time period during which the percentage of failed probes has been calculated in the presence of lossy links. It is not rare for a failure to be transient, which should not disqualify the parent. Thus an implementation should obverse the percentage of failed probes against the time frame. The reception of any message such as a DIO from a neighbor may be used as probes (failed or successful) if the link can be trusted to be symmetrical.

17.6.10 RPL Timer Management

Timer management is an important component of any protocol and RPL is no exception. The DIO timers used by RPL rely on the trickle algorithm proposed by [160], and other RPL timers may use the same algorithm in the future. Most routing protocols send keepalives to maintain routing

adjacency and any other control packets necessary to update their routing tables without explicitly trying to limit the control protocol overhead. This is done because the required bandwidth is negligible compared to the data traffic in "classic" IP networks. But such an approach would be problematic in LLNs where links are unstable and network resources are scarce. The issue is that limiting the control traffic also impacts the ability to maintain synchronization, the ability to quickly react to network changes, and so forth.

The trickle algorithm uses an adaptive mechanism to control the sending rates of control plane traffic such that nodes hear just enough packets to stay consistent under various circumstances. In the presence of change nodes send protocol control packets more often and control traffic rates are reduced when the network stabilizes. The trickle algorithm does not require complex code and states in the network. This is an important property considering the constrained resources on the nodes (some implementations only require 4–7 bytes of RAM for state maintenance).

RPL treats the DODAG construction as a consistency problem and makes use of trickle timers to decide when to multicast DIO messages. When an inconsistency is detected RPL messages are sent more often, and then as the network stabilizes RPL messages are sent less often.

Trickle behavior is controlled by several parameters:

- I: Current length of the communication interval.
- T: Timer value; T is in the range [I, I/2].
- C: Redundancy counter.
- K: Redundancy constant (learned from the DODAG root).
- I_{min}: Smallest value of I learned via the DIO message. $I_{min} = 2^{DIOIntervalMin}$ ms where DIOIntervalMin is advertised by the DODAG root in DIO messages.
- $I_{doubling}$: The number of times I may be doubled before maintaining a constant multicast rate. $I_{doubling}$ is advertised as DIOIntervalDoubling by the DODAG root in DIO messages.
- I_{max}: Largest value of $I_{max} = I_{min} * 2^{Idoubling}$.

In RPL trickle a node sets the trickle variable I_{min} and $I_{doubling}$ to the original values learned from the DIO messages, C = 0, I = I_{min}, and a random value is chosen for T in the range [I/2,I]. Each time a node receives a consistent DIO message from a DODAG parent, the C counter is incremented. When the timer expires, C is compared to the RPL constant (K = DEFAULT_DIO_REDUNDANCY_CONSTANT) to decide whether or not to multicast a DIO message. When the communication interval I expires, I is doubled, the C counter is reset, and a new value of T is chosen until I reaches the maximum value of I_{max}. The RPL specification explicitly states that the variable C may not be incremented. Indeed in some cases it may be beneficial not to increment C to avoid the suppression of some RPL control messages (this aspect is still under consideration).

When is the RPL trickle timer reset? The trickle timer has to be reset each time a DODAG inconsistency is detected to increase the frequency at which DIO messages are sent to quickly update the DODAG: when a new node joins the DODAG, when it receives a multicast DIS message from another node, when the node moves within a DODAG, when a node receives a modified DIO message from a DODAG parent reflecting some changes in the DODAG, when a potential loop is detected (e.g., a DODAG parent receives a packet that it would have forwarded inward), when the rank of a DODAG parent has changed, and so forth.

By tuning the values of I_{min} and I_{max} it becomes possible to achieve some trade-off between the need for consistent DODAG, speed to propagate changes, and the protocol overhead. Some

simulations indicate that by setting I_{min} and I_{max} to a few dozen milliseconds and 1 hour, respectively, the control traffic could be reduced to up to 75% compared to a fix beacon value of 30 s. Knowing this, applicability statement documents combined with the specifics of the network where RPL is deployed should provide further guidance. For example, in some cases it may not be advantageous to set the I_{min} value too small (e.g., with low power MAC layers) to avoid simultaneous sending of DIO messages. The expected effect of using the trickle algorithm on control traffic is shown via simulation in the next section.

17.6.11 Simulation Results

Although only real-life deployments provide actual data on the efficiency of a protocol, there are a number of tools that a protocol designer can use during the design process, and simulation is undoubtedly one of the most useful. Although simulators are not "formal" mathematical proofs, they do provide useful data and help improve the level of confidence on the design choices. Furthermore, in most cases, there is no mathematical model that can be used to simulate the level of complexity of the protocol and real-life conditions.

During the design process of RPL, a number of simulations were performed. [239] is undoubtedly one of the major contributions in this area. A discrete event simulator has been developed based on OMNET++ [254] and the Castalia module for Wireless Sensor Networks within OMNET++.

One of the major challenges when developing a smart object network simulator is model link behavior. With lossy links such as low-power wireless links or PLC links, none of the mathematical models such as Markov Chains are applicable. Thus the approach taken in [239] uses real-life link traces as input to get high-fidelity results representative of real networks. Hundreds of link traces were gathered to build a link failure model database for both indoor and outdoor low-power lossy links. Each trace provided the PDR at different times. For some links, the received signal strength indication (RSSI) was available and due to the correlation between the RSSI and the PDR [254], it was possible to derive the PDR from the RSSI.

The simulator reads a topology database and randomly selects real-life traces when simulating RPL, thus providing very useful results that can be trusted. When a packet is to be transmitted by a node, the PDR of the link is read from the database and the packet is dropped with a probability equal to 1-PDR (different random number generators are used for all links to avoid link correlation).

Several networks have been simulated with consistent results and the results for one of them are provided in this section (the simulated network is depicted in Figure 17.10).

Data traffic is "constant bit rate" with a configurable rate. In the simulation run, the constant data traffic rate was set to 5 packets per second (a fairly high traffic rate for LLNs, but the idea was to stress the network to exacerbate some protocol characteristics).

Link failures are directly read from the link behavior database to which random failures were added according to an M/M/1 Markov Chain model (the interarrival times were set to a mean of 1 per hour).

In these simulations, 25% of packets were destined to the root and 75% to other nodes. In most networks a good proportion of the traffic is sent to the root or sink behind the root. In these simulations we chose to have a fairly high proportion of P2P traffic to study the efficiency of P2P routing with RPL.

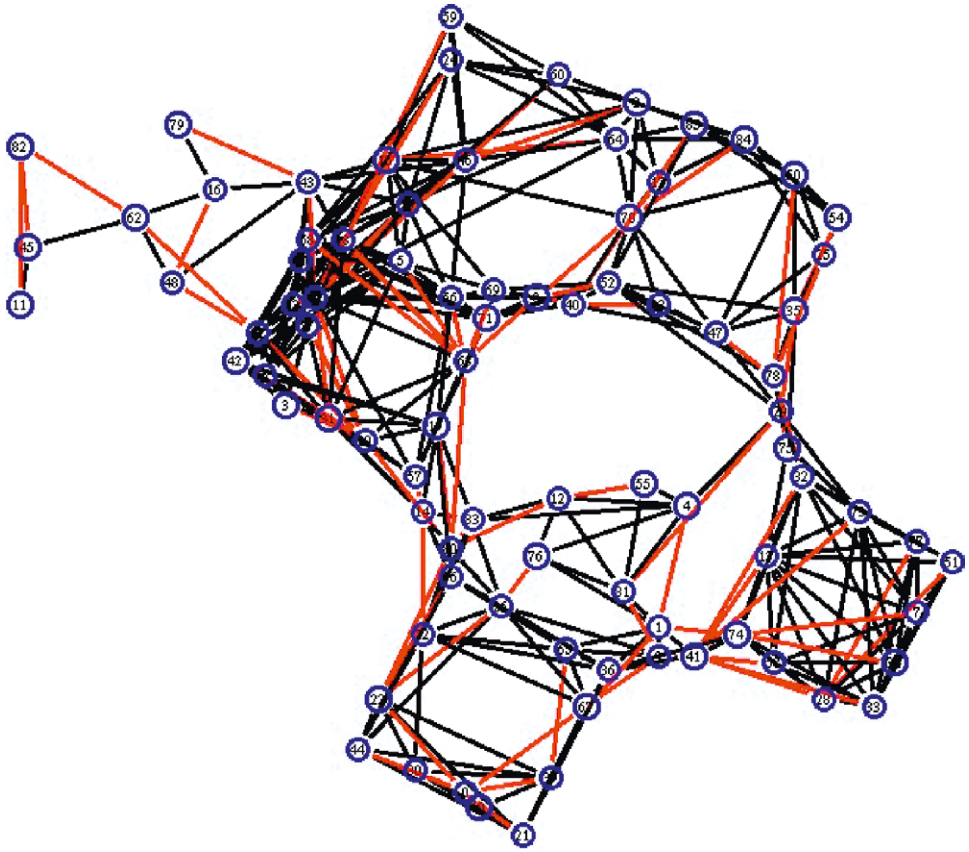

FIGURE 17.10

Topology of the simulated network.

The objective was to observe RPL behavior in a number of conditions (steady state, high stress) regarding several metrics for a single DODAG instance computed using the network topology shown in Figure 17.11. The RPL metric is the ETX (as described earlier) and the OF consists of minimizing the ETX path cost.

Several RPL characteristics were studied: control traffic, routing table size, path efficiency, and failure handling.

17.6.11.1 Control Traffic

In "classic" IP networks, the control traffic overhead (the routing protocol in this case) is generally not problematic considering the bandwidth available on high-speed links and is negligible compared to the data traffic. This is in contrast with LLNs where it is imperative to minimize the control traffic overhead and try to bound the control traffic to the data traffic. It is also imperative to reduce the traffic control load as the network stabilizes, which is the main motivation for using dynamic trickle

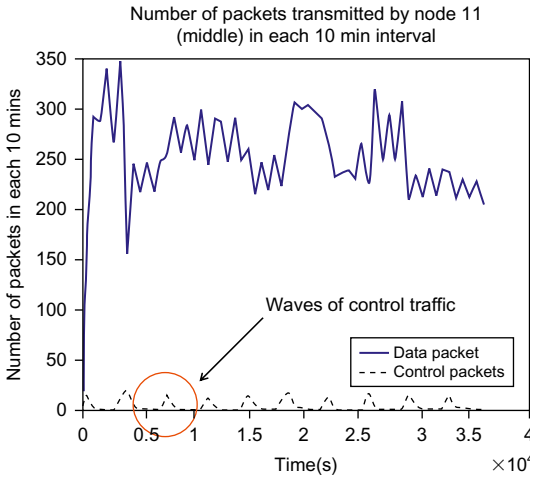

FIGURE 17.11

RPL control versus data traffic.

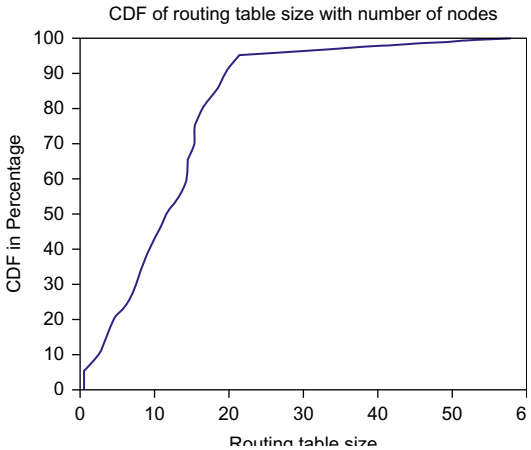

FIGURE 17.12

RPL routing table sizes.

timers. The values of the trickle timers for I_{min} and $I_{doubling}$ in these simulations were 1 and 16 seconds, respectively. Figure 17.11 shows how the data and control traffic varies over time.

The first observation is that the control traffic is clearly negligible compared to the data traffic (for that data traffic assumption, which is more true as we get closer to the DODAG root), but more importantly as the DODAG stabilizes the control traffic decreases significantly. This illustrates the desired effect of the trickle timers. We can observe waves of the control traffic. Each time an inconsistency is detected in the DODAG such as a path cost change, new parent after a failure, or a global repair mechanism triggered by the DODAG root (the DODAG root increments the DODAGSequenceNumber), the DIO changes and the trickle timers are reset. These factors explain the waves of control traffic. As expected and desired, when the DODAG stabilizes the traffic control is reduced accordingly as expected because of the trickle timers.

17.6.11.2 Routing Table Size

Nodes in LLNs have constrained memory. In extreme cases, some nodes cannot even store a routing table. RPL supports the insertion of such nodes in the network as discussed in the previous sections. In other cases routing tables may potentially contain dozens of entries, but nodes have limited memory for the storage of the routing table compared to IP core routers that can easily store hundreds of thousands of BGP routes. Thus, it is interesting to observe the memory requirements of RPL regarding routing table sizes. Figure 17.12 shows the Cumulative Distribution Function (CDF) for the number of required routing table entries (the number of routing entries increases as we get closer to the sink). Note: these results are in the absence of route aggregation in the network. There is tremendous interest in coupling RPL with route aggregation to limit the routing table sizes, and this work is currently in progress.

17.6.11.3 Path Efficiency

The DODAG computed by RPL is a sub-topology of the physical connectivity graph just like any other routing protocol. In other words, there are paths that the traffic has to follow along the DODAG

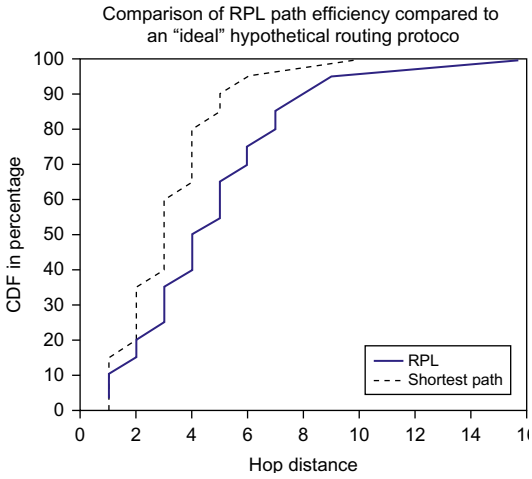

FIGURE 17.13

Path efficiency.

although a more optimal path may actually exist (outside of the DODAG) in the physical connectivity. This is particularly true for P2P traffic where the traffic from node A to node B must meet a common ancestor before being redirected down to the DODAG toward the destination (node B), with the exception of the P2P traffic between two nodes that are in direct range because of the use of multicast DAO. Thus the idea is to see how "suboptimal" the path computed by the DODAG for P2P traffic is compared to an "ideal" routing protocol that would systematically compute the best available path between A and B based on the actual connectivity.

⚠ Note: it is critical to remember that although RPL builds a DODAG this is not a P2MP or MP2P routing protocol: RPL fully supports P2P routing. RPL has even been enhanced with mechanisms such as multicast DAO to provide shortcuts for nodes in direct range and optimized P2P routing.

RPL provides a good quality path for the majority of cases. Still additional mechanisms may be added in the future with regards to P2P routing. As a reminder, these are simulations results and as such cannot be generalized. The results are shown in Figure 17.13. Other simulations on that particular subject are in progress.

17.6.11.4 Failure Handling

The ability of a routing protocol to compute an alternate path in the presence of network element failures has always been a critical characteristic of a routing protocol. Unfortunately, there is always tension between the control traffic cost, the environment, and the ability to quickly reroute the traffic. In a highly stable high-speed network, routing protocols use fast failure detection mechanisms to quickly detect a failure and reroute the traffic along a backup path. In contrast, in lossy environments in the presence of frequent failures the routing protocol should not constantly recompute paths (thus leading to high control traffic, oscillations, etc.), which is what RPL achieves as explained in detail earlier in this chapter. RPL makes use of two different repair mechanisms that have been discussed in Section 17.6.8: global repair triggered by the DODAG root and local repair where nodes locally handle the failure. We provide several simulation results showing both mechanisms. The metric used to illustrate the effect of RPL repair mechanisms is the amount of time during which no path was available when having to send a packet during the course of the simulation. For example, Figure 17.14 shows that in 80% of the cases the period of time without path was around 20 s **for the specific RPL parameters used in these simulations, of course**. In Figure 17.14 we also show the CDF for the failure period when first using global repair only.

Figure 17.14 shows the failure time for two different frequencies of global repair: in the first case, global repair (generation of new DODAGSequenceNumber) is set to 1 hour and in the second case it is reduced to 1 mn. As expected, this allows reduction in the failure time at the cost of increasing the control traffic cost (we can observe an increase of the control traffic, looking at node 11 in the

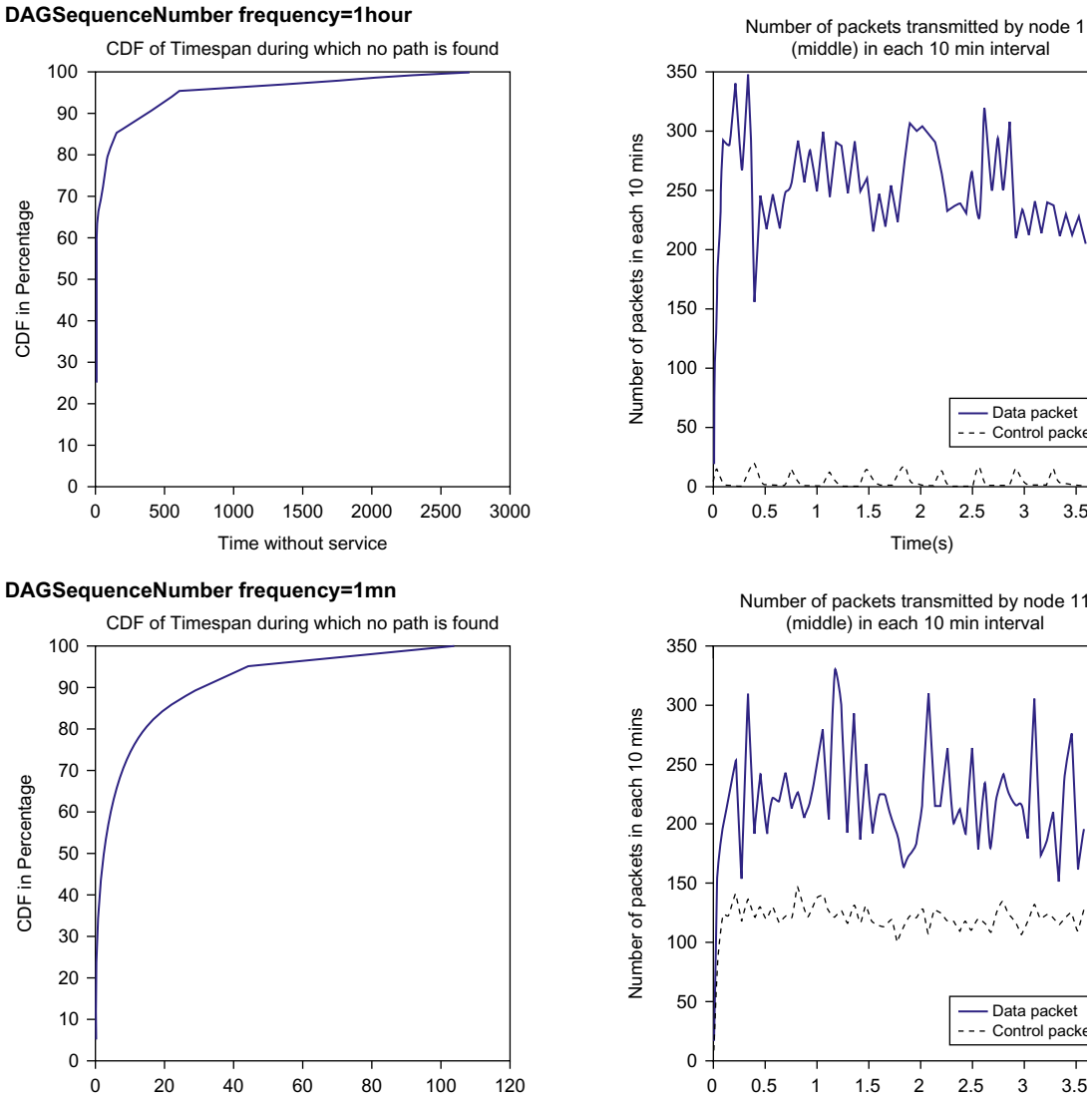

FIGURE 17.14

Time without service with global repair only.

middle of the network). As discussed earlier, it was decided to add a local repair mechanism to reduce the failure time. This way, the local repair mechanism quickly provides an alternate path followed by global repair to further reoptimize the DODAG. This is shown in Figure 17.15 where the global repair mechanism is set to 1 hour and local repair is activated. We observe that the failure time is reduced *dramatically*. The traffic control is *slightly* increased with local repair but localized (not even visible on the simulation run).

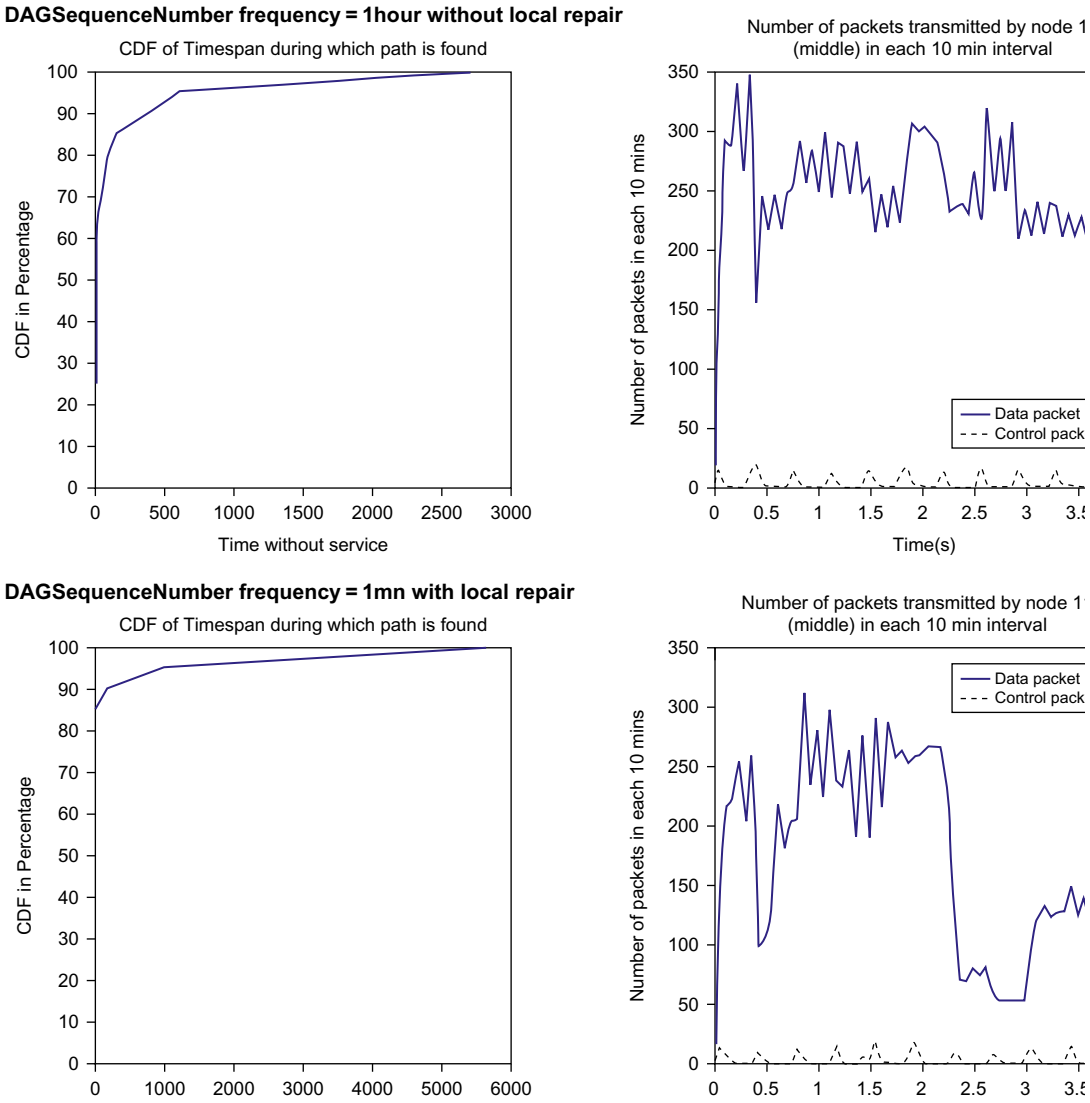

FIGURE 17.15

Time without service with global and local repair.

17.7 CONCLUSIONS

This chapter was entirely devoted to RPL, the new routing protocol for IP smart object networks developed by the IETF ROLL Working Group. A series of novel mechanisms have been designed to make RPL an efficient distance vector routing protocol for smart object networks in support of

P2P, MP2P, and P2MP traffic designed for LLNs. RPL has been designed as highly modular, with a very small footprint, and able to support a wide range of metrics and constraints according to the environments of interest while operating in constrained environments thus reducing the control traffic whenever possible. RPL can even be deployed to support multiple routing topologies according to the objective function (e.g., optimize reliability, minimize latency, etc.). Furthermore, several mechanisms referred to as global and local repair have been designed to provide alternate paths in the presence of failures and to reoptimize the routing topology on a configurable periodic basis while ensuring a high degree of robustness and flexibility. Early implementations show that RPL will only require a few kilobytes of Flash and a very few KB of RAM in its current specification.