Networking Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 9, 2010

T. Winter, Ed.

P. Thubert, Ed.
Cisco Systems
ROLL Design Team
IETF ROLL WG
March 8, 2010

RPL: IPv6 Routing Protocol for Low power and Lossy Networks
draft-ietf-roll-rpl-07

Abstract

Low power and Lossy Networks (LLNs) are a class of network in which both the routers and their interconnect are constrained: LLN routers typically operate with constraints on (any subset of) processing power, memory and energy (battery), and their interconnects are characterized by (any subset of) high loss rates, low data rates and instability. LLNs are comprised of anything from a few dozen and up to thousands of LLN routers, and support point-to-point traffic (between devices inside the LLN), point-to-multipoint traffic (from a central control point to a subset of devices inside the LLN) and multipoint-to-point traffic (from devices inside the LLN towards a central control point). This document specifies the IPv6 Routing Protocol for LLNs (RPL), which provides a mechanism whereby multipoint-to-point traffic from devices inside the LLN towards a central control point, as well as point-to-multipoint traffic from the central control point to the devices inside the LLN, is supported. Support for point-to-point traffic is also available.

Status of this Memo

Table of Contents

1. Introduction

Low power and Lossy Networks (LLNs) consist of largely of constrained nodes (with limited processing power, memory, and sometimes energy when they are battery operated). These routers are interconnected by lossy links, typically supporting only low data rates, that are usually unstable with relatively low packet delivery rates. Another characteristic of such networks is that the traffic patterns are not simply unicast, but in many cases point-to-multipoint or multipoint-to-point. Furthermore such networks may potentially comprise up to thousands of nodes. These characteristics offer unique challenges to a routing solution: the IETF ROLL Working Group has defined application-specific routing requirements for a Low power and Lossy Network (LLN) routing protocol, specified in [I-D.ietf-roll-building-routing-reqs], [I-D.ietf-roll-home-routing-reqs], [RFC5673], and [RFC5548]. This document specifies the IPv6 Routing Protocol for Low power and lossy networks (RPL).

1.1. Design Principles

RPL was designed with the objective to meet the requirements spelled out in [I-D.ietf-roll-building-routing-reqs], [I-D.ietf-roll-home-routing-reqs], [RFC5673], and [RFC5548]. Because those requirements are heterogeneous and sometimes incompatible in nature, the approach is first taken to design a protocol capable of supporting a core set of functionalities corresponding to the intersection of the requirements. As the RPL design evolves optional features may be added to address some application specific requirements. This is a key protocol design decision providing a granular approach in order to restrict the core of the protocol to a minimal set of functionalities, and to allow each implementation of the protocol to be optimized differently. All "MUST" application requirements that cannot be satisfied by RPL will be specifically listed in the Appendix A, accompanied by a justification.

A network may run multiple instances of RPL concurrently. Each such instance may serve different and potentially antagonistic constraints or performance criteria. This document defines how a single instance operates.

RPL is a generic protocol that is to be deployed by instantiating the generic operation described in this document with a specific objective function (OF) (which ties together metrics, constraints, and an optimization objective) to realize a desired objective in a given environment.

A set of companion documents to this specification will provide

further guidance in the form of applicability statements specifying a set of operating points appropriate to the Building Automation, Home Automation, Industrial, and Urban application scenarios.

1.2. Expectations of Link Layer Type

RPL does not rely on any particular features of a specific link layer technology. RPL is designed to be able to operate over a variety of different link layers, including but not limited to, low power wireless or PLC (Power Line Communication) technologies.

Implementers may find RFC 3819 [RFC3819] a useful reference when designing a link layer interface between RPL and a particular link layer technology.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Additionally, this document uses terminology from [I-D.ietf-roll-terminology], and introduces the following terminology:

DAG: Directed Acyclic Graph. A directed graph having the property that all edges are oriented in such a way that no cycles exist. All edges are contained in paths oriented toward and terminating at one or more root nodes.

DAG root: A DAG root is a node within the DAG that has no outgoing edges. Because the graph is acyclic, by definition all DAGs must have at least one DAG root and all paths terminate at a DAG root.

Destination Oriented DAG (DODAG): A DAG rooted at a single destination, i.e. at a single DAG root (the DODAG root) with no outgoing edges.

DODAG root: A DODAG root is the DAG root of a DODAG.

Rank: The rank of a node in a DAG identifies the nodes position with respect to a DODAG root. The farther away a node is from a DODAG root, the higher is the rank of that node. The rank of a node may be a simple topological distance, or may more commonly be calculated as a function of other properties as described later.

DODAG parent: A parent of a node within a DODAG is one of the immediate successors of the node on a path towards the DODAG root. The DODAG parent of a node will have a lower rank than the node itself. (See Section 3.6.2.1).

DODAG sibling: A sibling of a node within a DODAG is defined in this specification to be any neighboring node which is located at the same rank within a DODAG. Note that siblings defined in this manner do not necessarily share a common DODAG parent. (See Section 3.6.2.1).

Sub-DODAG The sub-DODAG of a node is the set of other nodes in the DODAG that might use a path towards the DODAG root that contains that node. Nodes in the sub-DODAG of a node have a greater rank than that node itself (although not all nodes of greater rank are necessarily in the sub-DODAG of that node). (See Section 3.6.2.1).

DODAGID: The identifier of a DODAG root. The DODAGID must be unique within the scope of a RPL Instance in the LLN.

DODAG Iteration: A specific sequence number iteration ("version") of a DODAG with a given DODAGID.

RPL Instance: A set of possibly multiple DODAGs. A network may have more than one RPL Instance, and a RPL node can participate in multiple RPL Instances. Each RPL Instance operates independently of other RPL Instances. This document describes operation within a single RPL Instance. In RPL, a node can belong to at most one DODAG per RPL Instance. The tuple (RPLInstanceID, DODAGID) uniquely identifies a DODAG.

RPLInstanceID: Unique identifier of a RPL Instance.

DODAGSequenceNumber: A sequential counter that is incremented by the root to form a new Iteration of a DODAG. A DODAG Iteration is identified uniquely by the (RPLInstanceID, DODAGID, DODAGSequenceNumber) tuple.

Up: Up refers to the direction from leaf nodes towards DODAG roots, following the orientation of the edges within the DODAG.

Down: Down refers to the direction from DODAG roots towards leaf nodes, going against the orientation of the edges within the DODAG.

Objective Code Point (OCP): An identifier, used to indicate which Objective Function is in use for forming a DODAG. The Objective Code Point is further described in [I-D.ietf-roll-routing-metrics].

Objective Function (OF): Defines which routing metrics, optimization objectives, and related functions are in use in a DODAG. The Objective Function is further described in [I-D.ietf-roll-routing-metrics].

Goal: The Goal is a host or set of hosts that satisfy a particular application objective / OF. Whether or not a DODAG can provide connectivity to a goal is a property of the DODAG. For example, a goal might be a host serving as a data collection point, or a gateway providing connectivity to an external infrastructure.

Grounded: A DODAG is said to be grounded, when the root can reach the Goal of the objective function.

Floating: A DODAG is floating if is not Grounded. A floating DODAG is not expected to reach the Goal defined for the OF.

As they form networks, LLN devices often mix the roles of 'host' and 'router' when compared to traditional IP networks. In this document, 'host' refers to an LLN device that can generate but does not forward RPL traffic, 'router' refers to an LLN device that can forward as well as generate RPL traffic, and 'node' refers to any RPL device, either a host or a router.

3. Protocol Overview

The aim of this section is to describe RPL in the spirit of [RFC4101]. Protocol details can be found in further sections.

3.1. Topology

This section describes how the basic RPL topologies, and the rules by which these are constructed, i.e. the rules governing DODAG formation.

3.1.1. Topology Identifiers

RPL uses four identifiers to track and control the topology:

o The first is a RPLInstanceID. A RPLInstanceID identifies a set of one or more DODAGs. All DODAGs in the same RPL Instance use the

same OF. A network may have multiple RPLInstanceIDs, each of which defines an independent set of DODAGs, which may be optimized for different OFs and/or applications. The set of DODAGs identified by a RPLInstanceID is called a RPL Instance.

o The second is a DODAGID. The scope of a DODAGID is a RPL Instance. The combination of RPLInstanceID and DODAGID uniquely identifies a single DODAG in the network. A RPL Instance may have multiple DODAGs, each of which has an unique DODAGID.

o The third is a DODAGSequenceNumber. The scope of a DODAGSequenceNumber is a DODAG. A DODAG is sometimes reconstructed from the DODAG root, by incrementing the DODAGSequenceNumber. The combination of RPLInstanceID, DODAGID, and DODAGSequenceNumber uniquely identifies a DODAG Iteration.

o The fourth is rank. The scope of rank is a DODAG Iteration. Rank establishes a partial order over a DODAG Iteration, defining individual node positions with respect to the DODAG root.

3.1.2. DODAG Information

For each DODAG that a node is, or may become, a member of, the implementation should conceptually keep track of the following information. The data structures described in this section are intended to illustrate a possible implementation to aid in the description of the protocol, but are not intended to be normative.

o RPLInstanceID

o DODAGID

o DODAGSequenceNumber

o DAG Metric Container, including DAGObjectiveCodePoint

o A set of Destination Prefixes offered by the DODAG root and available via paths upwards along the DODAG o A set of DODAG parents

o A set of DODAG siblings

o A timer to govern the sending of RPL control messages

3.2. Instances, DODAGs, and DODAG Iterations

Each RPL Instance constructs a routing topology optimized for a certain Objective Function (OF). A RPL Instance may provide routes to certain destination prefixes, reachable via the DODAG roots. A single RPL Instance contains one or more Destination Oriented DAG (DODAG) roots. These roots may operate independently, or may coordinate over a non-LLN backchannel.

Each root has a unique identifier, the DODAGID.

A RPL Instance may comprise:

o a single DODAG with a single root

   * For example, a DODAG optimized to minimize latency rooted at a single centralized lighting controller in a home automation application.

o multiple uncoordinated DODAGs with independent roots (differing DODAGIDs)

   * For example, multiple data collection points in an urban data collection application that do not have an always-on backbone suitable to coordinate to form a single DODAG, and further use the formation of multiple DODAGs as a means to dynamically and autonomously partition the network.

o a single DODAG with a single virtual root coordinating LLN sinks (with the same DODAGID) over some non-LLN backbone

   * For example, multiple border routers operating with a reliable

backbone, e.g. in support of a 6LowPAN application, that are capable to act as logically equivalent sinks to the same DODAG.

o a combination of the above as suited to some application scenario.

Traffic is bound to a specific RPL Instance by a marking in the flow label of the IPv6 header. Traffic originating in support of a particular application may be tagged to follow an appropriate RPL instance which enables certain (path) properties, for example to follow paths optimized for low latency or low energy. The provisioning or automated discovery of a mapping between a RPLInstanceID and a type or service of application traffic is beyond the scope of this specification.

An example of a RPL Instance comprising a number of DODAGs is depicted in Figure 1. A DODAG Iteration (two "versions" of the same

DODAG) is depicted in Figure 2.

```
     +----------------------------------------------------------------+
     |                                                                |
     | +--------------+                                               |
     | |              |                                               |
     | |     (R1)     |            (R2)                   (Rn)        |
     | |     /  \     |            /| \                  / |  \       |
     | |    /    \    |           / |  \                /  |   \      |
     | |  (A)    (B)  |         (C) |  (D)     ...    (F) (G)  (H)    |
     | |  /|\     |\  |         /   |   |\             |   |    |     |
     | | : : :    : : |        :   (E)  : :            :   :    :     |
     | |              |            / \                                |
     | +--------------+           :   :                               |
     |      DODAG                                                     |
     |                                                                |
     +----------------------------------------------------------------+
                                RPL Instance
```

Figure 1: RPL Instance

```
            +----------------+                +----------------+
```

```
            |                 |                 |                 |
            |       (R1)      |                 |       (R1)      |
            |      /   \      |                 |      /          |
            |     /     \     |                 |     /           |
            |   (A)     (B)   |        \        |   (A)           |
            |   /|\      |\   |  ------\        |   /|\           |
            |  : : (C)   : :  |         \       |  : : (C)        |
            |                 |         /       |        \        |
            |                 |  ------/        |         \       |
            |                 |        /        |         (B)     |
            |                 |                 |          |\     |
            |                 |                 |          : :    |
            |                 |                 |                 |
            +-----------------+                 +-----------------+
                Sequence N                         Sequence N+1
```

Figure 2: DODAG Iteration

3.3. Traffic Flows

3.3.1. Multipoint-to-Point Traffic

Multipoint-to-Point (MP2P) is a dominant traffic flow in many LLN applications ([I-D.ietf-roll-building-routing-reqs], [I-D.ietf-roll-home-routing-reqs], [RFC5673], [RFC5548]). The destinations of MP2P flows are designated nodes that have some application significance, such as providing connectivity to the larger Internet or core private IP network. RPL supports MP2P traffic by allowing MP2P destinations to be reached via DODAG roots.

3.3.2. Point-to-Multipoint Traffic

Point-to-multipoint (P2MP) is a traffic pattern required by several LLN applications ([I-D.ietf-roll-building-routing-reqs], [I-D.ietf-roll-home-routing-reqs], [RFC5673], [RFC5548]). RPL supports P2MP traffic by using a destination advertisement mechanism that provisions routes toward destination prefixes and away from roots. Destination advertisements can update routing tables as the underlying DODAG topology changes.

3.3.3. Point-to-Point Traffic

RPL DODAGs provide a basic structure for point-to-point (P2P) traffic. For a RPL network to support P2P traffic, a root must be able to route packets to a destination. Nodes within the network may also have routing tables to destinations. A packet flows towards a root until it reaches an ancestor that has a known route to the destination.

RPL also supports the case where a P2P destination is a 'one-hop' neighbor.

RPL neither specifies nor precludes additional mechanisms for computing and installing more optimal routes to support arbitrary P2P traffic.

3.4. Upward Routes and DODAG Construction

RPL provisions routes up towards DODAG roots, forming a DODAG optimized according to the Objective Function (OF) in use. RPL nodes construct and maintain these DODAGs through exchange of DODAG Information Object (DIO) messages. Undirected links between siblings are also identified during this process, which can be used to provide additional diversity.

3.4.1. DODAG Information Object (DIO)

A DIO identifies the RPL Instance, the DODAGID, the values used to compute the RPL Instance's objective function, and the present DODAG Sequence Number. It can also include additional routing and configuration information. The DIO includes a measure derived from the position of the node within the DODAG, the rank, which is used for nodes to determine their positions relative to each other and to inform loop avoidance/detection procedures. RPL exchanges DIO messages to establish and maintain routes.

RPL adapts the rate at which nodes send DIO messages. When a DODAG is detected to be inconsistent or needs repair, RPL sends DIO messages more frequently. As the DODAG stabilizes, the DIO message rate tapers off, reducing the maintenance cost of a steady and well-working DODAG.

This document defines an ICMPv6 Message Type "RPL Control Message", which is capable of carrying a DIO.

3.4.2. DAG Repair

RPL supports global repair over the DODAG. A DODAG Root may increment the DODAG Sequence Number, thereby initiating a new DODAG iteration. This institutes a global repair operation, revising the DODAG and allowing nodes to choose an arbitrary new position within the new DODAG iteration.

RPL supports mechanisms which may be used for local repair within the DODAG iteration. The DIO message specifies the necessary parameters as configured from the DODAG root. Local repair options include the allowing a node, upon detecting a loss of connectivity to a DODAG it is a member of, to:

o Poison its sub-DODAG by advertising an effective rank of INFINITY to its sub-DODAG, OR detach and form a floating DODAG in order to preserve inner connectivity within its sub-DODAG.

o Move down within the DODAG iteration (i.e. increase its rank) in a limited manner, no further than a bound configured by the DODAG root via the DIO so as not to count all the way to infinity. Such a move may be undertaken after waiting an appropriate poisoning interval, and should allow the node to restore connectivity to the DODAG Iteration, if at all possible.

3.4.3. Grounded and Floating DODAGs

DODAGs can be grounded or floating. A grounded DODAG offers connectivity to to a goal. A floating DODAG offers no such connectivity, and provides routes only to nodes within the DODAG. Floating DODAGs may be used, for example, to preserve inner connectivity during repair.

3.4.4. Administrative Preference

An implementation/deployment may specify that some DODAG roots should be used over others through an administrative preference. Administrative preference offers a way to control traffic and engineer DODAG formation in order to better support application requirements or needs.

3.4.5. Objective Function (OF)

The Objective Function (OF) implements the optimization objectives of route selection within the RPL Instance. The OF is identified by an Objective Code Point (OCP) within the DIO, and its specification also indicates the metrics and constraints in use. The OF also specifies the procedure used to compute rank within a DODAG iteration. Further details may be found in [I-D.ietf-roll-routing-metrics], [I-D.ietf-roll-of0], and related companion specifications.

By using defined OFs that are understood by all nodes in a particular deployment, and by referencing these in the DIO message, RPL nodes may work to build optimized LLN routes using a variety of application and implementation specific metrics and goals.

In the case where a node is unable to encounter a suitable RPL Instance using a known Objective Function, it may be configured to join a RPL Instance using an unknown Objective Function - but in that case only acting as a leaf node.

3.4.6. Distributed Algorithm Operation

A high level overview of the distributed algorithm which constructs the DODAG is as follows:

- o Some nodes are configured to be DODAG roots, with associated DODAG configuration.

- o Nodes advertise their presence, affiliation with a DODAG, routing cost, and related metrics by sending link-local multicast DIO messages.

o Nodes may adjust the rate at which DIO messages are sent in response to stability or detection of routing inconsistencies.

o Nodes listen for DIOs and use their information to join a new DODAG, or to maintain an existing DODAG, as according to the specified Objective Function and rank-based loop avoidance rules.

o Nodes provision routing table entries, for the destinations specified by the DIO, via their DODAG parents in the DODAG iteration. Nodes may provision a DODAG parent as a default gateway.

o Nodes may identify DODAG siblings within the DODAG iteration to increase path diversity.

o Using DIOs, and possibly information in data packets, RPL nodes detect possible routing loops. When a RPL node detects a possible routing loop, it may adapt its DIO transmission rate to apply a local repair to the topology.

3.5. Downward Routes and Destination Advertisement

RPL constructs and maintains DODAGs with DIO messages to establish upward routes: it uses Destination Advertisement Object (DAO) messages to establish downward routes along the DODAG as well as other routes. DAO messages are an optional feature for applications that require P2MP or P2P traffic. DIO messages advertise whether destination advertisements are enabled within a given DODAG.

3.5.1. Destination Advertisement Object (DAO)

A Destination Advertisement Object (DAO) conveys destination information upwards along the DODAG so that a DODAG root (and other intermediate nodes) can provision downward routes. A DAO message includes prefix information to identify destinations, a capability to record routes in support of source routing, and information to determine the freshness of a particular advertisement.

Nodes that are capable of maintaining routing state may aggregate routes from DAO messages that they receive before transmitting a DAO message. Nodes that are not capable of maintaining routing state may attach a next-hop address to the Reverse Route Stack contained within the DAO message. The Reverse Route Stack is subsequently used to generate piecewise source routes over regions of the LLN that are incapable of storing downward routing state.

A special case of the DAO message, termed a no-DAO, is used to clear downward routing state that has been provisioned through DAO

operation.

This document defines an ICMPv6 Message Type "RPL Control Message", which is capable of carrying a DAO.

3.5.1.1. 'One-Hop' Neighbors

In addition to sending DAOs toward DODAG roots, RPL nodes may occasionally emit a link-local multicast DAO message advertising available destination prefixes. This mechanism allow provisioning a trivial 'one-hop' route to local neighbors.

3.6. Routing Metrics and Constraints Used By RPL

Routing metrics are used by routing protocols to compute shortest paths. Interior Gateway Protocols (IGPs) such as IS-IS ([RFC5120]) and OSPF ([RFC4915]) use static link metrics. Such link metrics can simply reflect the bandwidth or can also be computed according to a polynomial function of several metrics defining different link characteristics; in all cases they are static metrics. Some routing protocols support more than one metric: in the vast majority of the cases, one metric is used per (sub)topology. Less often, a second metric may be used as a tie-breaker in the presence of Equal Cost Multiple Paths (ECMP). The optimization of multiple metrics is known as an NP complete problem and is sometimes supported by some centralized path computation engine.

In contrast, LLNs do require the support of both static and dynamic metrics. Furthermore, both link and node metrics are required. In the case of RPL, it is virtually impossible to define one metric, or even a composite metric, that will satisfy all use cases.

In addition, RPL supports constrained-based routing where constraints may be applied to both link and nodes. If a link or a node does not satisfy a required constraint, it is 'pruned' from the candidate list, thus leading to a constrained shortest path.

The set of supported link/node constraints and metrics is specified in [I-D.ietf-roll-routing-metrics].

The role of the Objective Function is to specify which routing metrics and constraints are in use, and how these are used, in addition to the objectives used to compute the (constrained) shortest path.

Example 1: Shortest path: path offering the shortest end-to-end delay

Example 2: Constrained shortest path: the path that does not traverse any battery-operated node and that optimizes the path reliability

3.6.1. Loop Avoidance

RPL guarantees neither loop free path selection nor strong global convergence. In order to reduce control overhead, however, such as the cost of the count-to-infinity problem, RPL avoids creating loops when undergoing topology changes. Furthermore, RPL includes rank-based mechanisms for detecting loops when they do occur. RPL uses this loop detection to ensure that packets make forward progress within the DODAG iteration and trigger repairs when necessary.

3.6.1.1. Greediness and Rank-based Instabilities

Once a node has joined a DODAG iteration, RPL disallows certain behaviors, including greediness, in order to prevent resulting instabilities in the DODAG iteration.

If a node is allowed to be greedy and attempts to move deeper in the DODAG iteration, beyond its most preferred parent, in order to increase the size of the parent set, then an instability can result.

Suppose a node is willing to receive and process a DIO messages from a node in its own sub-DODAG, and in general a node deeper than itself. In this case, a possibility exists that a feedback loop is created, wherein two or more nodes continue to try and move in the DODAG iteration while attempting to optimize against each other. In some cases, this will result in instability. It is for this reason that RPL limits the cases where a node may process DIO messages from deeper nodes to some forms of local repair. This approach creates an 'event horizon', whereby a node cannot be influenced beyond some limit into an instability by the action of nodes that may be in its own sub-DODAG.

3.6.1.2. DODAG Loops

A DODAG loop may occur when a node detaches from the DODAG and reattaches to a device in its prior sub-DODAG. This may happen in particular when DIO messages are missed. Strict use of the DAG sequence number can eliminate this type of loop, but this type of loop may possibly be encountered when using some local repair mechanisms.

3.6.1.3. DAO Loops

A DAO loop may occur when the parent has a route installed upon receiving and processing a DAO message from a child, but the child has subsequently cleaned up the related DAO state. This loop happens when a no-DAO was missed and persists until all state has been cleaned up. RPL includes loop detection mechanisms that may mitigate the impact of DAO loops and trigger their repair.

In the case where stateless DAO operation is used, i.e. source routing specifies the down routes, then DAO Loops should not occur on the stateless portions of the path.

3.6.1.4. Sibling Loops

Sibling loops could occur if a group of siblings kept choosing amongst themselves as successors such that a packet does not make forward progress. This specification limits the number of times that sibling forwarding may be used at a given rank, in order to prevent sibling loops.

3.6.2. Rank Properties

The rank of a node is a scalar representation of the location of that node within a DODAG iteration. The rank is used to avoid and detect loops, and as such must demonstrate certain properties. The exact calculation of the rank is left to the Objective Function, and may depend on parents, link metrics, and the node configuration and policies.

The rank is not a cost metric, although its value can be derived from and influenced by metrics. The rank has properties of its own that are not necessarily those of all metrics:

Type: Rank is an abstract scalar. Some metrics are boolean (e.g. grounded), others are statistical and better expressed as a tuple like an expected value and a variance. Some OCPs use not one but a set of metrics bound by a piece of logic.

Function: Rank is the expression of a relative position within a DODAG iteration with regard to neighbors and is not necessarily a good indication or a proper expression of a distance or a cost to the root.

Stability: The stability of the rank determines the stability of the routing topology. Some dampening or filtering might be applied to keep the topology stable, and thus the rank does not necessarily change as fast as some physical metrics

would. A new DODAG iteration would be a good opportunity to reconcile the discrepancies that might form over time between metrics and ranks within a DODAG iteration.

Granularity: Rank is coarse grained. A fine granularity would prevent the selection of siblings.

Properties: Rank is strictly monotonic, and can be used to validate a progression from or towards the root. A metric, like bandwidth or jitter, does not necessarily exhibit this property.

Abstract: Rank does not have a physical unit, but rather a range of increment per hop, where the assignment of each increment is to be determined by the implementation.

The rank value feeds into DODAG parent selection, according to the RPL loop-avoidance strategy. Once a parent has been added, and a rank value for the node within the DODAG has been advertised, the nodes further options with regard to DODAG parent selection and movement within the DODAG are restricted in favor of loop avoidance.

3.6.2.1. Rank Comparison (DAGRank())

Rank may be thought of as a fixed point number, where the position of the decimal point between the integer part and the fractional part is determined by MinHopRankIncrease. MinHopRankIncrease is the minimum increase in rank between a node and any of its DODAG parents. When an objective function computes rank, the objective function operates on the entire (i.e. 16-bit) rank quantity. When rank is compared, e.g. for determination of parent/sibling relationships or loop detection, the integer portion of the rank is to be used. The integer portion of the Rank is computed by the DAGRank() macro as follows:

```
DAGRank(rank) = floor(rank/MinHopRankIncrease)
```

MinHopRankIncrease is provisioned at the DODAG Root and propagated in the DIO message. For efficient implementation the MinHopRankIncrease SHOULD be a power of 2. An implementation may configure a value MinHopRankIncrease as appropriate to balance between the loop avoidance logic of RPL (i.e. selection of eligible parents and siblings) and the metrics in use.

By convention in this document, using the macro DAGRank(node) may be interpreted as DAGRank(node.rank), where node.rank is the rank value

as maintained by the node.

A node A has a rank less than the rank of a node B if DAGRank(A) is less than DAGRank(B).

A node A has a rank equal to the rank of a node B if DAGRank(A) is equal to DAGRank(B).

A node A has a rank greater than the rank of a node B if DAGRank(A) is greater than DAGRank(B).

3.6.2.2. Rank Relationships

The computation of the rank MUST be done in such a way so as to maintain the following properties for any nodes M and N that are neighbors in the LLN:

DAGRank(M) is less than DAGRank(N): In this case, the position of M is closer to the DODAG root than the position of N. Node M may safely be a DODAG parent for Node N without risk of creating a loop. Further, for a node N, all parents in the DODAG parent set must be of rank less than DAGRank(N). In other words, the rank presented by a node N MUST be greater than that presented by any of its parents.

DAGRank(M) equals DAGRank(N): In this case the positions of M and N within the DODAG and with respect to the DODAG root are similar (identical). In some cases, Node M may be used as a successor by Node N, which however entails the chance of creating a loop (which must be detected and resolved by some other means).

DAGRank(M) is greater than DAGRank(N): In this case, the position of M is farther from the DODAG root than the position of N. Further, Node M may in fact be in the sub-DODAG of Node N. If node N selects node M as DODAG parent there is a risk to create a loop.

As an example, the rank could be computed in such a way so as to closely track ETX when the objective function is to minimize ETX, or latency when the objective function is to minimize latency, or in a more complicated way as appropriate to the objective function being used within the DODAG.

4. ICMPv6 RPL Control Message

This document defines the RPL Control Message, a new ICMPv6 message.

In accordance with [RFC4443], the RPL Control Message has the following format:

```
     0                   1                   2                   3
```

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                         Message Body                          +
|                                                               |
```

Figure 3: RPL Control Message

The RPL Control message is an ICMPv6 information message with a requested Type of 155.

The Code field identifies the type of RPL Control Message. This document defines three codes for the following RPL Control Message types:

- o 0x01: DODAG Information Solicitation (Section 5.2)
- o 0x02: DODAG Information Object (Section 5.1)
- o 0x04: Destination Advertisement Object (Section 6.1)

5. Upward Routes

This section describes how RPL discovers and maintains upward routes. It describes DODAG Information Objects (DIOs), the messages used to discover and maintain these routes. It specifies how RPL generates and responds to DIOs. It also describes DODAG Information Solicitation (DIS) messages, which are used to trigger DIO transmissions.

5.1. DODAG Information Object (DIO)

The DODAG Information Object carries information that allows a node to discover a RPL Instance, learn its configuration parameters, select a DODAG parent set, and maintain the upward routing topology.

5.1.1. DIO Base Format

DIO Base is an always-present container option in a DIO message. Every DIO MUST include a DIO Base.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|G|A|T|S|0| Prf |   Sequence    |             Rank              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| RPLInstanceID |     DTSN      |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
+                                                               +
|                            DODAGID                            |
+                                                               +
|                                                               |
+                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |   sub-option(s)...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: DIO Base

Control Field: The DAG Control Field has three flags and one field:

Grounded (G): The Grounded (G) flag indicates whether the upward routes this node advertises provide connectivity to the set of addresses which are application-defined goals. If the flag is set, the DODAG is grounded and provides such connectivity. If the flag is cleared, the DODAG is floating and may not provide such connectivity.

Destination Advertisement Supported (A): The Destination Advertisement Supported (A) flag indicates whether the root of this DODAG can collect and use downward route state. If the flag is set, nodes in the network are enabled to exchange destination advertisements messages to build downward routes (Section 6). If the flag is cleared, destination advertisement messages are disabled and the DODAG maintains only upward routes.

Destination Advertisement Trigger (T): The Destination Advertisement Trigger (T) flag indicates a complete refresh of downward routes. If the flag is set, then a refresh of downward route state is to take place over the entire DODAG. If the flag is cleared, the downward route maintenance is in its normal mode of operation. The further details of this process are described in Section 6.

Destination Advertisements Stored (S): The Destination Advertisements Stored (S) flag is used to indicate that a non-root ancestor is storing routing table entries learned from DAO messaging. If the flag is set, then a non-root ancestor is known to be storing routing table entries learned from DAO messages. If the flag is cleared, only the root node may be storing routing table entries learned from DAO messaging. This flag is further described in Section 6.

DODAGPreference (Prf): A 3-bit unsigned integer that defines how preferable the root of this DODAG is compared to other DODAG roots within the instance. DAGPreference ranges from 0x00 (least preferred) to 0x07 (most preferred). The default is 0 (least preferred). Section 5.3 describes how DAGPreference affects DIO processing.

Unassigned bits of the Control Field are reserved. They MUST be set to zero on transmission and MUST be ignored on reception.

Sequence Number: 8-bit unsigned integer set by the DODAG root. Section 5.3 describes the rules for sequence numbers and how they affect DIO processing.

Rank: 16-bit unsigned integer indicating the DODAG rank of the node sending the DIO message. Section 5.3 describes how Rank is set and how it affects DIO processing.

RPLInstanceID: 8-bit field set by the DODAG root that indicates which RPL Instance the DODAG is part of.

Destination Advertisement Trigger Sequence Number (DTSN): 8-bit unsigned integer set by the node issuing the DIO message. The Destination Advertisement Trigger Sequence Number (DTSN) flag is used as part of the procedure to maintain downward routes. The details of this process are described in Section 6.

DODAGID: 128-bit unsigned integer set by a DODAG root which uniquely identifies a DODAG. Possibly derived from the IPv6 address of the DODAG root.

5.1.2. DIO Base Rules

1. If the 'A' flag of a DIO Base is cleared, the 'T' flag MUST also be cleared.

2. For the following DIO Base fields, a node that is not a DODAG root MUST advertise the same values as its preferred DODAG parent (defined in Section 5.3.2). Therefore, if a DODAG root does not change these values, every node in a route to that DODAG root eventually advertises the same values for these fields. These fields are:
   1. Grounded (G)
   2. Destination Advertisement Supported (A)
   3. Destination Advertisement Trigger (T)
   4. DAGPreference (Prf)
   5. Sequence
   6. RPLInstanceID
   7. DODAGID

3. A node MAY update the following fields at each hop:
   1. Destination Advertisements Stored (S)
   2. DAGRank
   3. DTSN

4. The DODAGID field each root sets MUST be unique within the RPL Instance.

5.1.3. DIO Suboptions

This section describes the format of DIO suboptions and the five suboptions this document defines: Pad 1, Pad N, DAG Metric Container, DAG Destination Prefix, and DAG Configuration.

5.1.3.1. DIO Suboption Format

The Pad N, DAG Metric Container, DAG Destination Prefix, and DAG Configuration suboptions all follow this format:

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

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - -
|  Subopt. Type |       Suboption Length        | Suboption Data
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - -
```

Figure 5: DIO Suboption Generic Format

Suboption Type: 8-bit identifier of the type of suboption.

Suboption Length: 16-bit unsigned integer, representing the length in octets of the suboption, not including the suboption Type and Length fields.

Suboption Data: A variable length field that contains data specific to the option.

The following subsections specify the DIO message suboptions which are currently defined for use in the DODAG Information Object.

When processing a DIO message containing a suboption for which the Suboption Type value is not recognized by the receiver, the receiver MUST silently ignore the unrecognized option and continue to process the following suboption, correctly handling any remaining options in the message.

DIO message suboptions may have alignment requirements. Following the convention in IPv6, options with alignment requirements are aligned in a packet such that multi-octet values within the Option Data field of each option fall on natural boundaries (i.e., fields of width n octets are placed at an integer multiple of n octets from the start of the header, for n = 1, 2, 4, or 8).

5.1.3.2. Pad1

The Pad1 suboption format is as follows:

```
 0
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|   Type = 0    |
```

```
    +-+-+-+-+-+-+-+-+
```

Figure 6: Pad 1

NOTE! the format of the Pad1 option is a special case - it has neither Option Length nor Option Data fields.

The Pad1 option is used to insert one or two octets of padding in the DIO message to enable suboptions alignment. If more than two octets of padding is required, the PadN option, described next, should be used rather than multiple Pad1 options.

5.1.3.3. PadN

The PadN suboption format is as follows:


```
     0                   1                   2
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - -
    |   Type = 1    |        Suboption Length       | Suboption Data
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - -
```

Figure 7: Pad N

The PadN suboption is used to insert three or more octets of padding in the DIO message to enable suboptions alignment. For N (N > 2) octets of padding, the Suboption Length field contains the value N-3, and the Option Data consists of N-3 zero-valued octets. PadN Option data MUST be ignored by the receiver.

5.1.3.4. Metric Container

The Metric Container suboption format is as follows:

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

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - -
|   Type = 2    |       Suboption Length        | Metric Data
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - -
```

Figure 8: Metric Container

The Metric Container is used to report metrics along the DODAG. The Metric Container may contain a number of discrete node, link, and aggregate path metrics as chosen by the implementer. The Suboption Length field contains the length in octets of the Metric Data. The order, content, and coding of the Metric Container data is as specified in [I-D.ietf-roll-routing-metrics].

The processing and propagation of the Metric Container is governed by implementation specific policy functions.

5.1.3.5. Destination Prefix

The Destination Prefix suboption format is as follows:


```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 3    |      Suboption Length         |Resvd|Prf|Resvd|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Prefix Lifetime                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Prefix Length |                                               |
+-+-+-+-+-+-+-+-+                                               |
|             Destination Prefix (Variable Length)              |
.                                                               .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 9: DAG Destination Prefix

The Destination Prefix suboption is used to indicate that connectivity to the specified destination prefix is available from the DODAG root, or from another node located upwards along the DODAG on the path to the DODAG root. This may be useful in cases where more than one LBR is operating within the LLN and offering connectivity to different administrative domains, e.g. a home network and a utility network. In such cases, upon observing the Destination Prefixes offered by a particular DODAG, a node MAY decide to join multiple DODAGs in support of a particular application.

The Suboption Length is coded as the length of the suboption in octets, excluding the Type and Length fields.

Prf is the Route Preference as in [RFC4191]. The reserved fields MUST be set to zero on transmission and MUST be ignored on receipt.

The Prefix Lifetime is a 32-bit unsigned integer representing the length of time in seconds (relative to the time the packet is sent) that the Destination Prefix is valid for route determination. The lifetime is initially set by the node that owns the prefix and denotes the valid lifetime for that prefix (similar to AdvValidLifetime [RFC4861]). The value might be reduced by the originator and/or en-route nodes that will not provide connectivity for the whole valid lifetime. A value of all one bits (0xFFFFFFFF) represents infinity. A value of all zero bits (0x00000000) indicates a loss of reachability.

The Prefix Length is an 8-bit unsigned integer that indicates the number of leading bits in the destination prefix.

The Destination Prefix contains Prefix Length significant bits of the destination prefix. The remaining bits of the Destination Prefix, as

required to complete the trailing octet, are set to 0.

In the event that a DIO message may need to specify connectivity to more than one destination, the Destination Prefix suboption may be repeated.

5.1.3.6. DODAG Configuration

The DODAG Configuration suboption format is as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 4    |               Length          | DIOIntDoubl.  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  DIOIntMin.   |   DIORedun.   |   MaxRankInc  | MinHopRankInc |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 10: DODAG Configuration

The DODAG Configuration suboption is used to distribute configuration information for DODAG Operation through the DODAG. The information communicated in this suboption is generally static and unchanging within the DODAG, therefore it is not necessary to include in every DIO. This suboption MAY be included occasionally by the DODAG Root, and MUST be included in response to a unicast request, e.g. a unicast DODAG Information Solicitation (DIS) message.

The Length is coded as 5.

DIOIntervalDoublings is an 8-bit unsigned integer, configured on the DODAG root and used to configure the trickle timer (see Section 5.3.5.1 for details on trickle timers) governing when DIO message should be sent within the DODAG. DIOIntervalDoublings is the number of times that the DIOIntervalMin is allowed to be doubled during the trickle timer operation.

DIOIntervalMin is an 8-bit unsigned integer, configured on the DODAG root and used to configure the trickle timer governing when DIO message should be sent within the DODAG. The minimum configured interval for the DIO trickle timer in units of ms is 2^DIOIntervalMin. For example, a DIOIntervalMin value of 16ms is expressed as 4.

DIORedundancyConstant is an 8-bit unsigned integer used to configure suppression of DIO transmissions. DIORedundancyConstant is the minimum number of relevant incoming DIOs required to suppress a DIO transmission. If the value is 0xFF then the suppression mechanism is disabled.

MaxRankInc, 8-bit unsigned integer, is the DAGMaxRankIncrease. This is the allowable increase in rank in support of local repair. If DAGMaxRankIncrease is 0 then this mechanism is disabled.

MinHopRankInc, 8-bit unsigned integer, is the MinHopRankIncrease as described in Section 3.6.2.1.

5.2. DODAG Information Solicitation (DIS)

The DODAG Information Solicitation (DIS) message may be used to solicit a DODAG Information Object from a RPL node. Its use is analogous to that of a Router Solicitation; a node may use DIS to probe its neighborhood for nearby DODAGs. The DODAG Information Solicitation carries no additional message body. Section 5.3.5 describes how nodes respond to a DIS.

5.3. Upward Route Discovery and Maintenance

Upward route discovery allows a node to join a DODAG by discovering neighbors that are members of the DODAG and identifying a set of parents. The exact policies for selecting neighbors and parents is implementation-dependent. This section specifies the set of rules those policies must follow for interoperability.

5.3.1. RPL Instance

A RPLInstanceID MUST be unique across an LLN.

A node MAY belong to multiple RPL Instances.

Within a given LLN, there may be multiple, logically independent RPL instances. This document describes how a single instance behaves.

5.3.2. Neighbors and Parents within a DODAG Iteration

RPL's upward route discovery algorithms and processing are in terms of three logical sets of link-local nodes. First, the candidate neighbor set is a subset of the nodes that can be reached via link-local multicast. The selection of this set is implementation-dependent and OF-dependent. Second, the parent set is a restricted subset of the candidate neighbor set. Finally, the preferred parent, a set of size one, is an element of the parent set that is the preferred next hop in upward routes.

More precisely:

1. The DODAG parent set MUST be a subset of the candidate neighbor set.

2. A DODAG root MUST have a DODAG parent set of size zero.

3. A node that is not a DODAG root MAY maintain a DODAG parent set of size greater than or equal to one.

4. A node's preferred DODAG parent MUST be a member of its DODAG parent set.

5. A node's rank MUST be greater than all elements of its DODAG parent set.

6. When Neighbor Unreachability Detection (NUD), or an equivalent mechanism, determines that a neighbor is no longer reachable, a RPL node MUST NOT consider this node in the candidate neighbor set when calculating and advertising routes until it determines that it is again reachable. Routes through an unreachable neighbor MUST be eliminated from the routing table.

These rules ensure that there is a consistent partial order on nodes within the DODAG. As long as node ranks do not change, following the above rules ensures that every node's route to a DODAG root is loop-free, as rank decreases on each hop to the root. The OF can guide candidate neighbor set and parent set selection, as discussed in [I-D.ietf-roll-routing-metrics].

5.3.3. Neighbors and Parents across DODAG Iterations

The above rules govern a single DODAG iteration. The rules in this section define how RPL operates when there are multiple DODAG iterations:

5.3.3.1. DODAG Iteration

1. The tuple (RPLInstanceID, DODAGID, DODAGSequenceNumber) uniquely defines a DODAG Iteration. Every element of a node's DODAG parent set, as conveyed by the last heard DIO from each DODAG parent, MUST belong to the same DODAG iteration. Elements of a node's candidate neighbor set MAY belong to different DODAG Iterations.

2. A node is a member of a DODAG iteration if every element of its DODAG parent set belongs to that DODAG iteration, or if that node is the root of the corresponding DODAG.

3. A node MUST NOT send DIOs for DODAG iterations of which it is not a member.

4. DODAG roots MAY increment the DODAGSequenceNumber that they advertise and thus move to a new DODAG iteration. When a DODAG root increments its DODAGSequenceNumber, it MUST follow the conventions of Serial Number Arithmetic as described in [RFC1982].

5. Within a given DODAG, a node that is a not a root MUST NOT advertise a DODAGSequenceNumber higher than the highest DODAGSequenceNumber it has heard. Higher is defined as the greater-than operator in [RFC1982].

6. Once a node has advertised a DODAG iteration by sending a DIO, it MUST NOT be member of a previous DODAG iteration of the same DODAG (i.e. with the same RPLInstanceID, the same DODAGID, and a lower DODAGSequenceNumber). Lower is defined as the less-than operator in [RFC1982].

Within a particular implementation, a DODAG root may increment the DODAGSequenceNumber periodically, at a rate that depends on the deployment. In other implementations, loop detection may be considered sufficient to solve routing issues, and the DODAG root may increment the DODAGSequenceNumber only upon administrative intervention. Another possibility is that nodes within the LLN have some means by which they can signal detected routing inconsistencies or suboptimalities to the DODAG root, in order to request an on-demand DODAGSequenceNumber increment (i.e. request a global repair of the DODAG).

When the DODAG parent set becomes empty on a node that is not a root, (i.e. the last parent has been removed, causing the node to no longer be associated with that DODAG), then the DODAG information should not be suppressed until after the expiration of an implementation-specific local timer in order to observe if the DODAGSequenceNumber has been incremented, should any new parents appear for the DODAG.

As the DODAGSequenceNumber is incremented, a new DODAG Iteration spreads outward from the DODAG root. Thus a parent that advertises the new DODAGSequenceNumber can not possibly belong to the sub-DODAG of a node that still advertises an older DODAGSequenceNumber. A node may safely add such a parent, without risk of forming a loop, without regard to its relative rank in the prior DODAG Iteration. This is equivalent to jumping to a different DODAG.

As a node transitions to new DODAG Iterations as a consequence of following these rules, the node will be unable to advertise the

previous DODAG Iteration (prior DODAGSequenceNumber) once it has committed to advertising the new DODAG Iteration.

During transition to a new DODAG Iteration, a node may decide to forward packets via 'future parents' that belong to the same DODAG (same RPLInstanceID and DODAGID), but are observed to advertise a more recent (incremented) DODAGSequenceNumber.

5.3.3.2. DODAG Roots

1. A DODAG root that does not have connectivity to the set of addresses described as application-level goals, MUST NOT set the Grounded bit.

2. A DODAG root MUST advertise a rank of ROOT_RANK.

3. A node whose DODAG parent set is empty MAY become the DODAG root of a floating DODAG. It MAY also set its DAGPreference such that it is less preferred.

An LLN node that is a goal for the Objective Function is the root of its own grounded DODAG, at rank ROOT_RANK.

In a deployment that uses a backbone link to federate a number of LLN roots, it is possible to run RPL over that backbone and use one router as a "backbone root". The backbone root is the virtual root of the DODAG, and exposes a rank of BASE_RANK over the backbone. All the LLN roots that are parented to that backbone root, including the backbone root if it also serves as LLN root itself, expose a rank of

ROOT_RANK to the LLN, and are part of the same DODAG, coordinating DODAGSequenceNumber and other DODAG root determined parameters with the virtual root over the backbone.

5.3.3.3. DODAG Selection

The DODAGPreference (Prf) provides an administrative mechanism to engineer the self-organization of the LLN, for example indicating the most preferred LBR. If a node has the option to join a more preferred DODAG while still meeting other optimization objectives, then the node will generally seek to join the more preferred DODAG as determined by the OF. All else being equal, it is left to the implementation to determine which DODAG is most preferred, possibly based on additional criteria beyond Prf and the OF.

5.3.3.4. Rank and Movement within a DODAG Iteration

1. A node MUST NOT advertise a rank less than or equal to any member of its parent set within the DODAG Iteration.

2. A node MAY advertise a rank lower than its prior advertisement within the DODAG Iteration.

3. Let L be the lowest rank within a DODAG iteration that a given node has advertised. Within the same DODAG Iteration, that node MUST NOT advertise an effective rank higher than L + DAGMaxRankIncrease. INFINITE_RANK is an exception to this rule: a node MAY advertise an INFINITE_RANK at any time. (This corresponds to a limited rank increase for the purpose of local repair within the DODAG Iteration.)

4. A node MAY, at any time, choose to join a different DODAG within a RPL Instance. Such a join has no rank restrictions, unless that different DODAG is a DODAG Iteration of which that node has previously been a member, in which case the rule of the previous bullet (3) must be observed. Until a node transmits a DIO indicating its new DODAG membership, it MUST forward packets along the previous DODAG.

5. A node MAY, at any time after hearing the next DODAGSequenceNumber Iteration advertised from suitable DODAG parents, choose to migrate to the next DODAG Iteration within the DODAG.

Conceptually, an implementation is maintaining a DODAG parent set within the DODAG Iteration. Movement entails changes to the DODAG parent set. Moving up does not present the risk to create a loop but moving down might, so that operation is subject to additional constraints.

When a node migrates to the next DODAG Iteration, the DODAG parent and sibling sets need to be rebuilt for the new iteration. An implementation could defer to migrate for some reasonable amount of time, to see if some other neighbors with potentially better metrics but higher rank announce themselves. Similarly, when a node jumps into a new DODAG it needs to construct new DODAG parent/sibling sets for this new DODAG.

When a node moves to improve its position, it must conceptually abandon all DODAG parents and siblings with a rank larger than itself. As a consequence of the movement it may also add new siblings. Such a movement may occur at any time to decrease the rank, as per the calculation indicated by the OF. Maintenance of the parent and sibling sets occurs as the rank of candidate neighbors is observed as reported in their DIOs.

If a node needs to move down a DODAG that it is attached to, causing the rank to increase, then it MAY poison its routes and delay before moving as described in Section 5.3.3.5.

5.3.3.5. Poisoning a Broken Path

1. A node MAY poison, in order to avoid being used as an ancestor by the nodes in its sub-DODAG, by advertising an effective rank of INFINITE_RANK and resetting the associated DIO trickle timer to cause this INFINITE_RANK to be announced promptly.

2. The node MAY advertise an effective rank of INFINITE_RANK for an arbitrary number of DIO timer events, before announcing a new rank.

3. As per Section 5.3.3.4, the node MUST advertise INFINITE_RANK within the DODAG iteration in which it participates, if its revision in rank would exceed the maximum rank increase.

An implementation may choose to employ this poisoning mechanism when a node loses all of its current parents, i.e. the set of DODAG parents becomes depleted, and it can not jump to an alternate DODAG. An alternate mechanism is to form a floating DODAG.

The motivation for delaying announcement of the revised route through multiple DIO events is to (i) increase tolerance to DIO loss, (ii) allow time for the poisoning action to propagate, and (iii) to develop an accurate assessment of its new rank. Such gains are obtained at the expense of potentially increasing the delay before portions of the network are able to re-establish upwards routes. Path redundancy in the DODAG reduces the significance of either effect, since children with alternate parents should be able to utilize those alternates and retain their rank while the detached parent re-establishes its rank.

Although an implementation may advertise INFINITE_RANK for the purposes of poisoning, it is not expected to be equivalent to setting the rank to INFINITE_RANK, and an implementation would likely retain its rank value prior to the poisoning in some form, for purpose of maintaining its effective position within (L + DAGMaxRankIncrease).

5.3.3.6. Detaching

1. A node unable to stay connected to a DODAG within a given DODAG iteration MAY detach from this DODAG iteration. A node that detaches becomes root of its own floating DODAG and SHOULD immediately advertise this new situation in a DIO as an alternate to poisoning.

5.3.3.7. Following a Parent

1. If a node receives a DIO from one of its DODAG parents, indicating that the parent has left the DODAG, that node SHOULD stay in its current DODAG through an alternative DODAG parent, if possible. It MAY follow the leaving parent.

A DODAG parent may have moved, migrated to the next DODAG Iteration, or jumped to a different DODAG. A node should give some preference to remaining in the current DODAG, if possible, but ought to follow the parent if there are no other options.

5.3.4. DIO Message Communication

When an DIO message is received, the receiving node must first determine whether or not the DIO message should be accepted for further processing, and subsequently present the DIO message for further processing if eligible.

1. If the DIO message is malformed, then the DIO message is not eligible for further processing and is silently discarded. A RPL implementation MAY log the reception of a malformed DIO message.

2. If the sender of the DIO message is a member of the candidate neighbor set, then the DIO is eligible for further processing.

5.3.4.1. DIO Message Processing

As DIO messages are received from candidate neighbors, the neighbors may be promoted to DODAG parents by following the rules of DODAG discovery as described in Section 5.3. When a node places a neighbor into the DODAG parent set, the node becomes attached to the DODAG through the new DODAG parent node.

The most preferred parent should be used to restrict which other nodes may become DODAG parents. Some nodes in the DODAG parent set may be of a rank less than or equal to the most preferred DODAG parent. (This case may occur, for example, if an energy constrained device is at a lesser rank but should be avoided as per an optimization objective, resulting in a more preferred parent at a greater rank).

5.3.5. DIO Transmission

Each node maintains a timer, that governs when to multicast DIO messages. This timer is a trickle timer, as detailed in Section 5.3.5.1. The DIO Configuration Option includes the configuration of a RPL Instance's trickle timer.

o When a node detects or causes an inconsistency, it MUST reset the trickle timer.

o When a node migrates to a new DODAG Iteration it MUST reset the trickle timer to its minimum value

o When a node detects an inconsistency when forwarding a packet, as detailed in Section 7.2, the node MUST reset the trickle timer.

o When a node receives a multicast DIS message, it MUST reset the trickle timer to its minimum value.

o When a node receives a unicast DIS message, it MUST unicast a DIO message in response, and the response MUST include the DODAG Configuration Object. This provides a means that an interrogating node may be guaranteed to receive the DODAG Configuration Object, which otherwise might not be included at the option of the sender. In this case the node SHOULD NOT reset the trickle timer.

o If a node is not a member of a DODAG, it MUST suppress transmission of DIO messages.

o When a node is initialized, it MAY be configured to remain silent and not multicast any DIO messages until it has encountered and joined a DODAG (perhaps initially probing for a nearby DODAG with an DIS message). Alternately, it MAY choose to root its own floating DODAG and begin multicasting DIO messages using a default trickle configuration. The second case may be advantageous if it is desired for independent nodes to begin aggregating into scattered floating DODAGs, in the absence of a grounded node, for example in support of LLN installation and commissioning.

5.3.5.1. Trickle Timer for DIO Transmission

RPL treats the construction of a DODAG as a consistency problem, and uses a trickle timer [Levis08] to control the rate of control broadcasts.

For each DODAG that a node is part of (i.e. one DODAG per RPL Instance), the node must maintain a single trickle timer. The required state contains the following conceptual items:

I: The current length of the communication interval

T: A timer with a duration set to a random value in the range [I/2, I]

C: Redundancy Counter

I_min: The smallest communication interval in milliseconds. This value is learned from the DIO message as (2^DIOIntervalMin)ms. The default value is DEFAULT_DIO_INTERVAL_MIN.

I_doublings: The number of times I_min should be doubled before maintaining a constant rate, i.e. I_max = I_min * 2^I_doublings. This value is learned from the DIO message as DIOIntervalDoublings. The default value is DEFAULT_DIO_INTERVAL_DOUBLINGS.

5.3.5.1.1. Resetting the Trickle Timer

The trickle timer for a DODAG is reset by:

1. Setting I_min and I_doublings to the values learned from the DODAG root via a received DIO message.

2. Setting C to zero.

3. If I is not equal to I_min:

    1. Setting I to I_min.

    2. Setting T to a random value as described above.

    3. Restarting the trickle timer to expire after a duration T

When a node learns about a DODAG through a DIO message, and makes the decision to join this DODAG, it initializes the state of the trickle timer by resetting the trickle timer and listening. Each time it hears a redundant DIO message for this DODAG, it MAY increment C. The exact determination of what constitutes a redundant DIO message is left to an implementation; it could for example include DIOs that advertise the same rank.

When the timer fires at time T, the node compares C to the redundancy constant, DIORedundancyConstant. If C is less than that value, or if the DIORedundancyConstant value is 0xFF, the node generates a new DIO message and multicasts it. When the communication interval I expires, the node doubles the interval I so long as it has previously doubled it fewer than I_doubling times, resets C, and chooses a new T value.

5.3.5.1.2. Determination of Inconsistency

The trickle timer is reset whenever an inconsistency is detected within the DODAG, for example:

- o The node joins a new DODAG
- o The node moves within a DODAG
- o The node receives a DIO message from a DODAG parent that updates the information learned from a prior DIO message for that DODAG Parent
- o A DODAG parent forwards a packet intended to move up, indicating an inconsistency and possible loop.
- o A metric communicated in the DIO message is determined to be inconsistent, as according to a implementation specific path metric selection engine.
- o The rank of a DODAG parent has changed.

5.3.6. DODAG Selection

The DODAG selection is implementation and algorithm dependent. Nodes SHOULD prefer to join DODAGs for RPLInstanceIDs advertising OCPs and destinations compatible with their implementation specific objectives. In order to limit erratic movements, and all metrics being equal, nodes SHOULD keep their previous selection. Also, nodes SHOULD provide a means to filter out a parent whose availability is detected as fluctuating, at least when more stable choices are available.

When connection to a fixed network is not possible or preferable for security or other reasons, scattered DODAGs MAY aggregate as much as possible into larger DODAGs in order to allow connectivity within the LLN.

A node SHOULD verify that bidirectional connectivity and adequate link quality is available with a candidate neighbor before it considers that candidate as a DODAG parent.

5.4. Operation as a Leaf Node

In some cases a RPL node may attach to a DODAG as a leaf node only. One example of such a case is when a node does not understand the RPL Instance's OF. A leaf node does not extend DODAG connectivity but still needs to advertise its presence using DIOs. A node operating

as a leaf node must obey the following rules:

1. It MUST NOT transmit DIOs containing the DAG Metric Container.

2. Its DIOs must advertise a DAGRank of INFINITE_RANK.

3. It MAY transmit unicast DAOs as described in Section 6.2.

4. It MAY transmit multicast DAOs to the '1 hop' neighborhood as described in Section 6.2.9.

5.5. Administrative Rank

In some cases it might be beneficial to adjust the rank advertised by a node beyond that computed by the OF based on some implementation specific policy and properties of the node. For example, a node that has limited battery should be a leaf unless there is no other choice, and may then augment the rank computation specified by the OF in order to expose an exaggerated rank.

5.6. Collision

A race condition occurs if 2 nodes send DIO messages at the same time and then attempt to join each other. This might happen, for example, between nodes which act as DODAG root of their own DODAGs. In order to detect the situation, LLN Nodes time stamp the sending of DIO message. Any DIO message received within a short link-layer-dependent period introduces a risk. It left to the implementation to define the duration of the risk window.

There is risk of a collision when a node receives and processes a DIO within the risk window. For example, it may occur that two nodes are associated with different DODAGs and near-simultaneously send DIO messages, which are received and processed by both, and possibly result in both nodes simultaneously deciding to attach to each other. As a remedy, in the face of a potential collision, as determined by receiving a DIO within the risk window, the DIO message is not processed. It is expected that subsequent DIOs would not cross.

6. Downward Routes

This section describes how RPL discovers and maintains downward routes. Messages containing the Destination Advertisement Object (DAO), used to construct downward routes, are described. The downward routes are necessary in support of P2MP flows, from the DODAG roots toward the leaves. It specifies non-storing and storing behavior of nodes with respect to DAO messaging and DAO routing table

entries. Nodes, as according to their resources and the implementation, may selectively store routing table entries learned from DAO messages, or may instead propagate the DAO information upwards while adding source routing information. A further optimization is described whereby DAO messages may be used to populate routing table entries for the '1-hop' neighbors, which may be useful in some cases as a shortcut for P2P flows.

6.1. Destination Advertisement Object (DAO)

The Destination Advertisement Object (DAO) is used to propagate destination information upwards along the DODAG.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          DAO Sequence         |            DAO Rank           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| RPLInstanceID |   Route Tag   | Prefix Length |     RRCount   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          DAO Lifetime                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Destination Prefix (Variable Length)              |
.                                                               .
```

```
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Reverse Route Stack (Variable Length)             |
.                                                               .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   sub-option(s)...
+-+-+-+-+-+-+-+-+
```

Figure 11: The Destination Advertisement Object (DAO)

DAO Sequence: 16-bit unsigned integer. Incremented by the node that owns the prefix for each new DAO message for that prefix.

DAO Rank: 16-bit unsigned integer indicating the DAO Rank associated with the advertised Destination Prefix. The DAO Rank is analogous to the Rank in the DIO message in that it may be used to convey a relative distance to the Destination Prefix as computed by the Objective Function in use over the DODAG. It serves as a mechanism by which an ancestor node may order alternate DAO paths.

RPLInstanceID: 8-bit field indicating the topology instance associated with the DODAG, as learned from the DIO.

Route Tag: 8-bit unsigned integer. The Route Tag may be used to give a priority to prefixes that should be stored. This may be useful in cases where intermediate nodes are capable of storing a limited amount of routing state. The further specification of this field and its use is under investigation.

Prefix Length: 8-bit unsigned integer. Number of valid leading bits in the IPv6 Prefix.

RRCount: 8-bit unsigned integer. This counter is used to count the number of entries in the Reverse Route Stack. A value of '0' indicates that no Reverse Route Stack is present.

DAO Lifetime: 32-bit unsigned integer. The length of time in seconds (relative to the time the packet is sent) that the prefix is valid for route determination. A value of all one bits (0xFFFFFFFF) represents infinity. A value of all zero bits (0x00000000) indicates a loss of reachability.

Destination Prefix: Variable-length field identifying an IPv6 destination address, prefix, or multicast group. The Prefix Length field contains the number of valid leading bits in the prefix. The bits in the prefix after the prefix length (if any) are reserved and MUST be set to zero on transmission and MUST be ignored on receipt.

Reverse Route Stack: Variable-length field containing a sequence of RRCount (possibly compressed) IPv6 addresses. A node that adds on to the Reverse Route Stack will append to the list and increment the RRCount.

6.1.1. DAO Suboptions

The DAO message may optionally include a number of suboptions.

The DAO suboptions are in the same format as the DIO Suboptions described in Section 6.1.1.

In particular, a DAO message may include a DAG Metric Container suboption as described in Section 5.1.3.4. This suboption may be present in implementations where the DAO Rank is insufficient to optimize a path to the DAO Destination Prefix.

6.2. Downward Route Discovery and Maintenance

6.2.1. Overview

Destination Advertisement operation produces DAO messages that flow up the DODAG, provisioning downward routing state for destination prefixes available in the sub-DODAG of the DODAG root, and possibly other nodes. The routing state provisioned with this mechanism is in the form of soft-state routing table entries. DAO messages are able to record loose source routing information as by propagate up the DODAG. This mechanism is flexible to support the provisioning of paths which consist of fully specified source routes, piecewise source routes, or hop-by-hop routes as according to the implementation and the capabilities of the nodes.

Destination Advertisement may or may not be enabled over a DODAG rooted at a DODAG root. This is an a priori configuration determined by the implementation/deployment and not generally changed during the operation of the RPL LLN.

When Destination Advertisement is enabled:

1. Some nodes in the LLN MAY store at least one routing table entry for a particular destination learned from a DAO. Such a node is termed a 'storing node', with respect to that particular destination.

2. Some nodes are capable to store at least one routing table entry for every unique destination observed from all DAOs that pass through. Such a node is termed a 'fully storing node'.

3. DODAG roots nodes SHOULD be fully-storing nodes.

4. Other nodes in the DODAG are not required to store routing table entries for any particular destinations observed in DAOs. Nodes that do not store routing table entries from DAOs are termed 'non-storing nodes', with respect to a particular destination.

5. Non-storing nodes MUST participate in the construction of piecewise source routes as they propagate the DAO message, as described in Section 6.2.5.

6. Storing nodes MUST store any source route information received from the DAO (RRStack) in the routing table entry entry. If a node is not capable to do this then it must act as a non-storing node with respect to that particular destination.

7. Storing nodes MUST use piecewise source routes in order to forward data across a non-storing region of the LLN. The source routing mechanism is to be described in a companion specification. (If a node is not capable to do this, then that

node MUST NOT operate as a storing node).

6.2.2. Mode of Operation

- o DAO Operation may not be required for all use cases.

- o Some applications may only need support for collection/upward/MP2P flow with no acknowledgement/reciprocal traffic.

- o Some DODAGs may not support DAO Operation, which could mean that DAO Operation is wasteful overhead.

- o As a special case, multicast DAO operation may be used to populate 'one-hop' neighborhood routing table entries, and is distinct from the unicast DAO operation used to establish downward routes along the DODAG.

1. The 'A' flag in the DIO as conveyed from the DODAG root serves to enable/disable DAO operation over the entire DODAG. This flag should be administratively provisioned a priori at the DODAG root as a function of the implementation/deployment and not tend to change.

2. When DAO Operation is disabled, a node SHOULD NOT emit DAOs.

3. When DAO Operation is disabled, a node MAY ignore received DAOs.

6.2.3. Destination Advertisement Parents

- o Nodes will select a subset of their DODAG Parents to whom DAOs will be sent

  * This subset is the set of 'DAO Parents'

  * Each DAO parent MUST be a DODAG Parent. (Not all DODAG parents need to be DAO parents).

  * Operation with more than DAO Parent requires consideration of such issues as DAO fan-out and path diversity, to be elaborated in a future version of this specification.

- o The selection of DAO parents is implementation specific and may be based on selecting the DODAG Parents that offer the best upwards cost (as opposed to downwards or mixed), as determined by the metrics in use and the Objective Function.

- When DAO messages are unicast to the DAO Parent, the identity of the DAO Parent (DODAGID x DAGSequenceNumber) combined with the RPLInstanceID in the DAO message unambiguously associates the DAO message, and thus the particular destination prefix, with a DODAG Iteration.

- When DAO messages are unicast to the DAO Parent, the DAO Rank may be updated as according to the implementation and Objective Function in use to reflect the relative (aggregated) cost of reaching the Destination Prefix through that DAO parent. As a further extension, a DAO Suboption for the Metric Container may be included.

6.2.4. Operation of DAO Storing Nodes

6.2.4.1. DAO Routing Table Entry

A DAO Routing Table Entry conceptually contains the following elements:

- Advertising Neighbor Information
  - IPv6 Addr
  - Interface ID
- To which DAO Parents has this entry been reported
- Retry Counter
- Logical equivalent of DAO Content:
  - DAO Sequence
  - DAO Rank
  - DAO Lifetime
  - Route tag (used to prioritize which destination entries should be stored)
  - Destination Prefix (or Address or Mcast Group)
  - RR Stack*

The DAO Routing Table Entry is logically associated with the following states:

CONNECTED This entry is 'owned' by the node - it is manually configured and is considered as a 'self' entry for DAO Operation

REACHABLE This entry has been reported from a neighbor of the node. This state includes the following substates:

               CONFIRMED This entry is active, newly validated, and
                         usable

               PENDING   This entry is active, awaiting validation, and
                         usable. A Retry Counter is associated with
                         this substate

   UNREACHABLE This entry is being cleaned up. This entry may be
               suppressed when the cleanup process is complete.

When an attempt is to be made to report the DAO entry to DAO Parents, the DAO Entry record is logically marked to indicate that an attempt has not yet been made for each parent. As the unicast attempts are completed for each parent, this mark may be cleared. This mechanism may serve to limit DAO entry updates for each parent to a subset that needs to be reported.

6.2.4.1.1. DAO Routing Table Entry Management

```
           +---------------------------------+
           |                                 |
           |            REACHABLE            |    +-------------+
           |                                 |    |             |
           |        +-----------+            |    |  CONNECTED  |
     (*)----------->|           |-------+    |    |             |
           |        | Confirmed |       |    |    +-------------+
           |    +-->|           |---+   |    |
           |    |   +-----------+   |   |    |
           |    |                   |   |    |
           |    |                   |   |    |
           |    |                   |   |    |
           |    |   +-----------+   |   |    |    +-------------+
           |    |   |           |<--+   +--------->|             |
           |    +---|  Pending  |            |    | UNREACHABLE |
           |        |           |---------------->|             |--->(*)
           |        +-----------+            |    +-------------+
           |                                 |
           +---------------------------------+
```

DAO Routing Table Entry FSM

6.2.4.1.1.1. Operation in the CONNECTED state

1. CONNECTED DAO entries are to be provisioned outside of the context of RPL, e.g. through a management API. An implementation SHOULD provide a means to provision/manage CONNECTED DAO entries,

including whether they are to be redistributed in RPL.

6.2.4.1.1.2. Operation in the REACHABLE state

1. When a REACHABLE(*) entry times out, i.e. the DAO Lifetime has elapsed, the entry MUST be placed into the UNREACHABLE state and no-DAO SHOULD be scheduled to send to the node's DAO Parents.

2. When a no-DAO for a REACHABLE(*) entry is received with a newer DAO Sequence Number, the entry MUST be placed into the UNREACHABLE state and no-DAO SHOULD be scheduled to send to the node's DAO Parents.

3. When a REACHABLE(*) entry is to be removed because NUD or equivalent has determined that the next-hop neighbor is no longer reachable, the entry MUST be placed into the UNREACHABLE state and no-DAO SHOULD be scheduled to send to the node's DAO Parents.

4. When a REACHABLE(*) entry is to be removed because an associated Forwarding Error has been returned by the next-hop neighbor, the entry MUST be placed into the UNREACHABLE state and no-DAO SHOULD be scheduled to send to the node's DAO Parents.

5. When a DAO (or no-DAO) for a REACHABLE(*) entry is received with an older or unchanged DAO Sequence Number, then the DAO (or no-DAO) SHOULD be ignored and the associated entry MUST NOT be updated with the stale information.

6.2.4.1.1.2.1. REACHABLE(Confirmed)

1. When a DAO for a previously unknown (or UNREACHABLE) destination is received and is to be stored, it MUST be entered into the routing table in the REACHABLE(Confirmed) state, and a DAO SHOULD

be scheduled to send to the node's DAO Parents. Alternately the node may behave as a non-storing node with respect to this destination.

2. When a DAO for a REACHABLE(Confirmed) entry is received with a newer DAO Sequence Number, the entry MUST be updated with the logical equivalent of the DAO contents and a DAO SHOULD be scheduled to send to the node's DAO Parents.

3. When a DAO for a REACHABLE(Confirmed) entry is expected, e.g. because a DIO to request a DAO refresh is sent, then the DAO entry MUST be placed in the REACHABLE(Pending) state and the associated Retry Counter MUST be set to 0.

6.2.4.1.1.2.2. REACHABLE(Pending)

1. When a DAO for a REACHABLE(Pending) entry is received with a newer DAO Sequence Number, the entry MUST be updated with the logical equivalent of the DAO contents and the entry MUST be placed in the REACHABLE(Confirmed) state.

2. When a DAO for a REACHABLE(Pending) entry is expected, e.g. because DAO has (again) been triggered with respect to that neighbor, then the associated Retry Counter MUST be incremented.

3. When the associated Retry Counter for a REACHABLE(Pending) entry reaches a maximum threshold, the entry MUST be placed into the UNREACHABLE state and no-DAO SHOULD be scheduled to send to the node's DAO Parents.

6.2.4.1.1.3. Operation in the UNREACHABLE state

1. An implementation SHOULD bound the time that the entry is allocated in the UNREACHABLE state. Upon the equivalent expiry of the related timer (RemoveTimer), the entry SHOULD be suppressed.

2. While the entry is in the UNREACHABLE state a node SHOULD make a reasonable attempt to report a no-DAO to each of the DAO parents.

3. When the node has completed an attempt to report a no-DAO to each of the DAO parents, the entry SHOULD be suppressed.

6.2.5. Operation of DAO Non-storing Nodes

1. When a DAO is received from a child by a node who will not store a routing table entry for the DAO, the node MUST schedule to pass the DAO contents along to its DAO parents. Prior to passing the DAO along, the node MUST process the DAO as follows, in order that information necessary to construct a loose source route may be accumulated within the DAO payload as it moves up the DODAG:

    1. The most recent addition to the RRStack (the 'next waypoint') is investigated to determine if the node already has a route provisioned to the waypoint. If the node already has such a route, then it is not necessary to add additional information to the RRStack. The node SHOULD NOT modify the RRStack further.

    2. If the node does not have a route provisioned to the next waypoint, then the node MUST append the address of the child to the RRStack, and increment RRCount.

6.2.6. Scheduling to Send DAO (or no-DAO)

1. An implementation SHOULD arrange to rate-limit the sending of DAOs.

2. When scheduling to send a DAO, an implementation SHOULD equivalently start a timer (DelayDAO) to delay sending the DAO. If the DelayDAO timer is already running then the DAO may be considered as already scheduled, and implementation SHOULD leave the timer running at its present duration.

- When computing the delay before sending a DAO, in order to increase the effectiveness of aggregation, an implementation MAY allow time to receive DAOs from its sub-DODAG prior to emitting DAOs to its DAO Parents.

    * The scheduled delay in such cases may be, for example, such that DAO_LATENCY/DAGRank(self_rank) <= DelayDAO < DAO_LATENCY/DAGRank(parent_rank), where DAGRank() is defined as in

Section 3.6.2, such that nodes deeper in the DODAG may tend to report DAO messages first before their parent nodes will report DAO messages. Note that this suggestion is intended as an optimization to allow efficient aggregation -- it is not required for correct operation in the general case.

6.2.7. Triggering DAO Message from the Sub-DODAG

Triggering DAO messages from the Sub-DODAG occurs by using the following control fields with the rules described below:

The DTSN field from the DIO is a sequence number that is part of the mechanism to trigger DAO messages. The motivation to use a sequence number is to provide some means of reliable signaling to the sub-DODAG-- whereas a control flag that is activated for a short time may be unobserved by the sub-DODAG if the triggering DIO messages are lost, the DTSN increment may be observed later even if some DIO messages have been lost since the sequence number increment.

The 'T' flag provides a way to signal the refresh of DAO information over the entire DODAG iteration. Whereas a DTSN increment may only trigger a DAO refresh as far as the nearest storing node (because a storing node will not increment its own DTSN in response, as described in the rules below), the assertion of the 'T' flag in conjunction with an incremented DTSN will 'punch through' storing nodes to elicit a DAO refresh from the entire DODAG Iteration.

The 'S' flag provides a way to signal to a sub-DODAG that there is at least one non-root node somewhere in the set of DODAG ancestors,

where that non-root node is a storing node. This allows for an optimization-- when it is clear to a non-storing node that the root node can be the only storing ancestor, then that node does not necessarily need to trigger updates from its sub-DODAG when it modifies its DAO parent set. The motivation here is that the root node should be able to update its stored source routing information for the affected sub-DODAG based only on receiving DAO information concerning the link that changed. In the other case, when the 'S' flag is set, the non-storing node does not have a means to determine which DAO information may (or may not) need to be updated in the intermediate storing node so it must trigger DAO messages in order to update the intermediate storing node. Please note that some aspects of the proper use of the 'S' flag remain under investigation.

Further examples of triggering DAO messages are contained in Appendix B.

The control fields are used to trigger DAO messages as follows:

1. The DODAG root MUST clear the 'S' flag when it emits DIO messages.

2. Non-root nodes that store routing table entries learned from DAOs MUST set the 'S' flag when they emit DIO messages.

3. A node that has any DAO Parent with the 'S' flag set MUST also set the 'S' flag when it emits DIO messages.

4. A node that has all DAO Parents with cleared 'S' flags, and does not store routing table entries learned from DAOs, MUST clear the 'S' flag when it emits DIO messages.

5. A DAO Trigger Sequence Number (DTSN) MUST be maintained by each node per RPL Instance. The DTSN, in conjunction with the 'T' flag from the DIO message, provides a means by which DAO messages may be reliably triggered in the event of topology change.

6. The DTSN MUST be advertised by the node in the DIO message.

7. A node keeps track of the DTSN that it has heard from the last DIO from each of its DAO Parents. Note that there is one DTSN maintained per DAO Parent- each DAO Parent may independently increment it at will.

8. A node that is not a fully-storing node SHOULD increment its own DTSN when it adds a new parent, that parent having the 'S' flag set, to its DAO Parent set. It MAY defer advertising the

increment as long as it has a DAO parent that already provides adequate connectivity.

9. A node that is not a fully-storing node MUST increment its own DTSN when it receives a DIO from a DAO Parent that contains a

newly incremented DTSN. (The newly incremented DTSN is detected by comparing the value received in the DIO with the value last recorded for that DAO parent).

10. A fully-storing node MUST increment its own DTSN when it receives a DIO from a DAO Parent that contains a newly incremented DTSN and a set 'T' flag.

11. When a storing or non-storing node joins a new DODAG iteration, it SHOULD increment its DTSN as if the 'T' flag has been set.

12. DAO Transmission SHOULD be scheduled when a new parent is added to the DAO Parent set.

13. A node that receives a newly incremented DTSN from a DAO Parent MUST schedule a DAO transmission.

- When a node that is not fully-storing sees a DTSN increment, it will increment its own DTSN. This will cause the DTSN increment to extend down the DODAG to the first fully-storing node, which will send its DAOs back up, rebuilding source routes information along the way to the first node that incremented the DTSN, who then may report the new DAO information to its new parent.

- When a fully-storing node sees a DTSN increment, it is caused to reissue its entire set of routing table entries learned from DAOs (or an aggregated subset thereof), but will not need to increment its own DTSN. The 'DTSN increment wave' stops when it encounters fully-storing nodes.

- When a fully-storing node sees a DTSN increment AND the 'T' flag is set, it does increment its own DTSN as well. The 'T' flag 'punches through' all nodes, causing all routing tables in the entire sub-DODAG to be refreshed.

6.2.8. Sending DAO Messages to DAO Parents

1. When storing nodes send DAO messages for stored entries the RRStack SHOULD be cleared in the DAO message.

2. DAO Messages sent to DAO Parents MUST be unicast.

      * The IPv6 Source Address is the node sending the DAO message.

      * The IPv6 Destination Address is DAO parent.

3. When the appointed time arrives (DelayDAO) for the transmission of DAO messages (with jitter as appropriate) for the requested entries, the implementation MAY aggregate the the entries into a reduced numbers of DAOs to be reported to each parent, and perform compression if possible.

4. Note: it is NOT RECOMMENDED that a DAO Transmission (No-DAO) be scheduled when a DAO Parent is removed from the DAO Parent set.

6.2.9. Multicast Destination Advertisement Messages

A special case of DAO operation, distinct from unicast DAO operation, is multicast DAO operation which may be used to populate '1-hop' routing table entries.

1. A node MAY multicast a DAO message to the link-local scope all-nodes multicast address FF02::1.

2. A multicast DAO message MUST be used only to advertise information about self, i.e. prefixes directly connected to or owned by this node, such as a multicast group that the node is subscribed to or a global address owned by the node.

3. A multicast DAO message MUST NOT be used to relay connectivity information learned (e.g. through unicast DAO) from another node.

4. Information obtained from a multicast DAO MAY be installed in the routing table and MAY be propagated by a node in unicast DAOs.

5. A node MUST NOT perform any other DAO related processing on a received multicast DAO, in particular a node MUST NOT perform the actions of a DAO parent upon receipt of a multicast DAO.

o The multicast DAO may be used to enable direct P2P communication, without needing the RPL routing structure to relay the packets.

o The multicast DAO does not presume any DODAG relationship between the emitter and the receiver.

7. Packet Forwarding and Loop Avoidance/Detection

7.1. Suggestions for Packet Forwarding

When forwarding a packet to a destination, precedence is given to selection of a next-hop successor as follows:

1. In the scope of this specification, it is preferred to select a successor from a DODAG iteration that matches the RPLInstanceID marked in the IPv6 header of the packet being forwarded.

2. If a local administrative preference favors a route that has been learned from a different routing protocol than RPL, then use that successor.

3. If there is an entry in the routing table matching the destination that has been learned from a multicast destination advertisement (e.g. the destination is a one-hop neighbor), then use that successor.

4. If there is an entry in the routing table matching the destination that has been learned from a unicast destination advertisement (e.g. the destination is located down the sub-DODAG), then use that successor.

5. If there is a DODAG iteration offering a route to a prefix matching the destination, then select one of those DODAG parents as a successor.

6. If there is a DODAG parent offering a default route then select that DODAG parent as a successor.

7. If there is a DODAG iteration offering a route to a prefix matching the destination, but all DODAG parents have been tried and are temporarily unavailable (as determined by the forwarding procedure), then select a DODAG sibling as a successor.

8. Finally, if no DODAG siblings are available, the packet is dropped. ICMP Destination Unreachable may be invoked. An inconsistency is detected.

TTL MUST be decremented when forwarding. If the packet is being forwarded via a sibling, then the TTL MAY be decremented more aggressively (by more than one) to limit the impact of possible loops.

Note that the chosen successor MUST NOT be the neighbor that was the predecessor of the packet (split horizon), except in the case where it is intended for the packet to change from an up to an down flow, such as switching from DIO routes to DAO routes as the destination is

neared.

7.2. Loop Avoidance and Detection

RPL loop avoidance mechanisms are kept simple and designed to minimize churn and states. Loops may form for a number of reasons, from control packet loss to sibling forwarding. RPL includes a reactive loop detection technique that protects from meltdown and triggers repair of broken paths.

RPL loop detection uses information that is placed into the packet in the IPv6 flow label. The IPv6 flow label is defined in [RFC2460] and its operation is further specified in [RFC3697]. For the purpose of RPL operations, the flow label is constructed as follows:

```
 0                   1                   2
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |O|S|R|F|  SenderRank    | RPLInstanceID |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 12: RPL Flow Label

Down 'O' bit: 1-bit flag indicating whether the packet is expected to progress up or down. A router sets the 'O' bit when the packet is expect to progress down (using DAO routes), and resets it when forwarding towards the root of the DODAG iteration. A host MUST set the bit to 0.

Sibling 'S' bit: 1-bit flag indicating whether the packet has been forwarded via a sibling at the present rank, and denotes a risk of a sibling loop. A host sets the bit to 0.

Rank-Error 'R' bit: 1-bit flag indicating whether a rank error was detected. A rank error is detected when there is a mismatch in the relative ranks and the direction as indicated in the 'O' bit. A host MUST set the bit to 0.

Forwarding-Error 'F' bit: 1-bit flag indicating that this node can not forward the packet further towards the destination. The 'F' bit might be set by sibling that can not forward to a parent a packet with the Sibling 'S' bit set, or by a child node that does not have a route to destination for a packet with the down 'O' bit set. A host MUST set the bit to 0.

SenderRank: 8-bit field set to zero by the source and to DAGRank(rank) by a router that forwards inside the RPL network. (Note that the case where DAGRank(rank) does not fit into 8 bits is under investigation.)

RPLInstanceID: 8-bit field indicating the DODAG instance along which the packet is sent.

7.2.1. Source Node Operation

A packet that is sourced at a node connected to a RPL network or destined to a node connected to a RPL network MUST be issued with the flow label zeroed out, but for the RPLInstanceID field.

If the source is aware of the RPLInstanceID that is preferred for the flow, then it MUST set the RPLInstanceID field in the flow label accordingly, otherwise it MUST set it to the RPL_DEFAULT_INSTANCE.

If a compression mechanism such as 6LoWPAN is applied to the packet, the flow label MUST NOT be compressed even if it is set to all zeroes.

7.2.2. Router Operation

7.2.2.1. Conformance to RFC 3697

[RFC3697] mandates that the Flow Label value set by the source MUST be delivered unchanged to the destination node(s).

In order to restore the flow label to its original value, an RPL router that delivers a packet to a destination connected to a RPL network or that routes a packet outside the RPL network MUST zero out all the fields but the RPLInstanceID field that must be delivered without a change.

7.2.2.2. Instance Forwarding

Instance IDs are used to avoid loops between DODAGs from different origins. DODAGs that constructed for antagonistic constraints might contain paths that, if mixed together, would yield loops. Those loops are avoided by forwarding a packet along the DODAG that is associated to a given instance.

The RPLInstanceID is placed by the source in the flow label. This RPLInstanceID MUST match the RPL Instance onto which the packet is placed by any node, be it a host or router.

When a router receives a packet that specifies a given RPLInstanceID

and the node can forward the packet along the DODAG associated to that instance, then the router MUST do so and leave the RPLInstanceID flag unchanged.

If any node can not forward a packet along the DODAG associated to the RPLInstanceID in the flow label, then the node SHOULD discard the packet.

7.2.2.3. DAG Inconsistency Loop Detection

The DODAG is inconsistent if the direction of a packet does not match the rank relationship. A receiver detects an inconsistency if it receives a packet with either:

  the 'O' bit set (to down) from a node of a higher rank.

  the 'O' bit reset (for up) from a node of a lesser rank.

  the 'S' bit set (to sibling) from a node of a different rank.

When the DODAG root increments the DODAGSequenceNumber a temporary rank discontinuity may form between the next iteration and the prior iteration, in particular if nodes are adjusting their rank in the next iteration and deferring their migration into the next iteration. A router that is still a member of the prior iteration may choose to forward a packet to a (future) parent that is in the next iteration. In some cases this could cause the parent to detect an inconsistency because the rank-ordering in the prior iteration is not necessarily the same as in the next iteration and the packet may be judged to not be making forward progress. If the sending router is aware that the chosen successor has already joined the next iteration, then the sending router MUST update the SenderRank to INFINITE_RANK as it forwards the packets across the discontinuity into the next DODAG iteration in order to avoid a false detection of rank inconsistency.

One inconsistency along the path is not considered as a critical error and the packet may continue. But a second detection along the path of a same packet should not occur and the packet is dropped.

This process is controlled by the Rank-Error bit in the Flow Label. When an inconsistency, is detected on a packet, if the Rank-Error bit was not set then the Rank-Error bit is set. If it was set the packet is discarded and the trickle timer is reset.

7.2.2.4. Sibling Loop Avoidance

When a packet is forwarded along siblings, it cannot be checked for forward progress and may loop between siblings. Experimental

evidence has shown that one sibling hop can be very useful but is generally sufficient to avoid loops. Based on that evidence, this specification enforces the simple rule that a packet may not make 2 sibling hops in a row.

When a host issues a packet or when a router forwards a packet to a non-sibling, the Sibling bit in the packet must be reset. When a router forwards to a sibling: if the Sibling bit was not set then the Sibling bit is set. If the Sibling bit was set then then the router SHOULD return the packet to the sibling that that passed it with the Forwarding-Error 'F' bit set.

7.2.2.5. DAO Inconsistency Loop Detection and Recovery

A DAO inconsistency happens when router that has an down DAO route via a child that is a remnant from an obsolete state that is not matched in the child. With DAO inconsistency loop recovery, a packet can be used to recursively explore and cleanup the obsolete DAO states along a sub-DODAG.

In a general manner, a packet that goes down should never go up again. If DAO inconsistency loop recovery is applied, then the router SHOULD send the packet to the parent that passed it with the Forwarding-Error 'F' bit set. Otherwise the router MUST silently discard the packet.

7.2.2.6. Forward Path Recovery

Upon receiving a packet with a Forwarding-Error bit set, the node MUST remove the routing states that caused forwarding to that neighbor, clear the Forwarding-Error bit and attempt to send the packet again. The packet may its way to an alternate neighbor. If that alternate neighbor still has an inconsistent DAO state via this node, the process will recurse, this node will set the Forwarding-Error 'F' bit and the routing state in the alternate neighbor will be cleaned up as well.

8. Multicast Operation

This section describes further the multicast routing operations over an IPv6 RPL network, and specifically how unicast DAOs can be used to relay group registrations up. Wherever the following text mentions Multicast Listener Discovery (MLD), one can read MLDv2 ([RFC3810]) or v3.

As is traditional, a listener uses a protocol such as MLD with a router to register to a multicast group.

Along the path between the router and the DODAG root, MLD requests are mapped and transported as DAO messages within the RPL protocol; each hop coalesces the multiple requests for a same group as a single DAO message to the parent(s), in a fashion similar to proxy IGMP, but recursively between child router and parent up to the root.

A router might select to pass a listener registration DAO message to its preferred parent only, in which case multicast packets coming back might be lost for all of its sub-DODAG if the transmission fails over that link. Alternatively the router might select to copy additional parents as it would do for DAO messages advertising unicast destinations, in which case there might be duplicates that the router will need to prune.

As a result, multicast routing states are installed in each router on the way from the listeners to the root, enabling the root to copy a multicast packet to all its children routers that had issued a DAO message including a DAO for that multicast group, as well as all the attached nodes that registered over MLD.

For unicast traffic, it is expected that the grounded root of an DODAG terminates RPL and MAY redistribute the RPL routes over the external infrastructure using whatever routing protocol is used there. For multicast traffic, the root MAY proxy MLD for all the nodes attached to the RPL routers (this would be needed if the multicast source is located in the external infrastructure). For such a source, the packet will be replicated as it flows down the DODAG based on the multicast routing table entries installed from the DAO message.

For a source inside the DODAG, the packet is passed to the preferred parents, and if that fails then to the alternates in the DODAG. The packet is also copied to all the registered children, except for the one that passed the packet. Finally, if there is a listener in the external infrastructure then the DODAG root has to further propagate the packet into the external infrastructure.

As a result, the DODAG Root acts as an automatic proxy Rendezvous Point for the RPL network, and as source towards the Internet for all multicast flows started in the RPL LLN. So regardless of whether the root is actually attached to the Internet, and regardless of whether the DODAG is grounded or floating, the root can serve inner multicast streams at all times.

9. Maintenance of Routing Adjacency

The selection of successors, along the default paths up along the

DODAG, or along the paths learned from destination advertisements down along the DODAG, leads to the formation of routing adjacencies that require maintenance.

In IGPs such as OSPF [RFC4915] or IS-IS [RFC5120], the maintenance of a routing adjacency involves the use of Keepalive mechanisms (Hellos) or other protocols such as BFD ([I-D.ietf-bfd-base]) and MANET Neighborhood Discovery Protocol (NHDP [I-D.ietf-manet-nhdp]). Unfortunately, such an approach is not desirable in constrained environments such as LLN and would lead to excessive control traffic in light of the data traffic with a negative impact on both link loads and nodes resources. Overhead to maintain the routing adjacency should be minimized. Furthermore, it is not always possible to rely on the link or transport layer to provide information of the associated link state. The network layer needs to fall back on its own mechanism.

Thus RPL makes use of a different approach consisting of probing the neighbor using a Neighbor Solicitation message (see [RFC4861]). The reception of a Neighbor Advertisement (NA) message with the "Solicited Flag" set is used to verify the validity of the routing adjacency. Such mechanism MAY be used prior to sending a data packet. This allows for detecting whether or not the routing adjacency is still valid, and should it not be the case, select another feasible successor to forward the packet.

10. Guidelines for Objective Functions

An Objective Function (OF) allows for the selection of a DODAG to join, and a number of peers in that DODAG as parents. The OF is used to compute an ordered list of parents. The OF is also responsible to compute the rank of the device within the DODAG iteration.

The Objective Function is indicated in the DIO message using an Objective Code Point (OCP), as specified in [I-D.ietf-roll-routing-metrics], and indicates the method that must be used to construct the DODAG (e.g. "minimize the path cost using the ETX metric and avoid 'Blue' links"). The Objective Code Points are specified in [I-D.ietf-roll-routing-metrics], [I-D.ietf-roll-of0], and related companion specifications.

Most Objective Functions are expected to follow the same abstract behavior:

o The parent selection is triggered each time an event indicates that a potential next hop information is updated. This might happen upon the reception of a DIO message, a timer elapse, or a trigger indicating that the state of a candidate neighbor has changed.

- o An OF scans all the interfaces on the device. Although there may typically be only one interface in most application scenarios, there might be multiple of them and an interface might be configured to be usable or not for RPL operation. An interface can also be configured with a preference or dynamically learned to be better than another by some heuristics that might be link-layer dependent and are out of scope. Finally an interface might or not match a required criterion for an Objective Function, for instance a degree of security. As a result some interfaces might be completely excluded from the computation, while others might be more or less preferred.

- o An OF scans all the candidate neighbors on the possible interfaces to check whether they can act as a router for a DODAG. There might be multiple of them and a candidate neighbor might need to pass some validation tests before it can be used. In particular, some link layers require experience on the activity with a router to enable the router as a next hop.

- o An OF computes self's rank by adding to the rank of the candidate a value representing the relative locations of self and the candidate in the DODAG iteration.

    - * The increase in rank must be at least MinHopRankIncrease. (This prevents the creation of a path of sibling links connecting a child with its parent.)

    - * To keep loop avoidance and metric optimization in alignment, the increase in rank should reflect any increase in the metric value. For example, with a purely additive metric such as ETX, the increase in rank can be made proportional to the increase in the metric.

    - * Candidate neighbors that would cause self's rank to increase are not considered for parent selection

- o Candidate neighbors that advertise an OF incompatible with the set of OF specified by the policy functions are ignored.

- o As it scans all the candidate neighbors, the OF keeps the current best parent and compares its capabilities with the current candidate neighbor. The OF defines a number of tests that are critical to reach the objective. A test between the routers determines an order relation.

* If the routers are roughly equal for that relation then the next test is attempted between the routers,

* Else the best of the 2 becomes the current best parent and the scan continues with the next candidate neighbor

* Some OFs may include a test to compare the ranks that would result if the node joined either router

o When the scan is complete, the preferred parent is elected and self's rank is computed as the preferred parent rank plus the step in rank with that parent.

o Other rounds of scans might be necessary to elect alternate parents and siblings. In the next rounds:

* Candidate neighbors that are not in the same DODAG are ignored

* Candidate neighbors that are of greater rank than self are ignored

* Candidate neighbors of an equal rank to self (siblings) are ignored for parent selection

* Candidate neighbors of a lesser rank than self (non-siblings) are preferred

11. RPL Constants and Variables

Following is a summary of RPL constants and variables.

BASE_RANK This is the rank for a virtual root that might be used to coordinate multiple roots. BASE_RANK has a value of 0.

ROOT_RANK This is the rank for a DODAG root. ROOT_RANK has a value of 1.

INFINITE_RANK This is the constant maximum for the rank. INFINITE_RANK has a value of 0xFFFF.

RPL_DEFAULT_INSTANCE This is the RPLInstanceID that is used by this protocol by a node without any overriding policy. RPL_DEFAULT_INSTANCE has a value of 0.

DEFAULT_DIO_INTERVAL_MIN TBD (To be determined)

DEFAULT_DIO_INTERVAL_DOUBLINGS TBD (To be determined)

DEFAULT_DIO_REDUNDANCY_CONSTANT TBD (To be determined)

DIO Timer One instance per DODAG that a node is a member of. Expiry triggers DIO message transmission. Trickle timer with variable interval in [0, DIOIntervalMin..2^DIOIntervalDoublings]. See Section 5.3.5.1

DAG Sequence Number Increment Timer Up to one instance per DODAG that the node is acting as DODAG root of. May not be supported in all implementations. Expiry triggers revision of DODAGSequenceNumber, causing a new series of updated DIO message to be sent. Interval should be chosen appropriate to propagation time of DODAG and as appropriate to application requirements (e.g. response time vs. overhead).

DelayDAO Timer Up to one instance per DAO parent (the subset of DODAG parents chosen to receive destination advertisements) per DODAG. Expiry triggers sending of DAO message to the DAO parent. See Section 6.2.6

RemoveTimer Up to one instance per DAO entry per neighbor (i.e. those neighbors that have given DAO messages to this node as a DODAG parent) Expiry triggers a change in state for the DAO entry, setting up to do unreachable (No-DAO) advertisements or immediately deallocating the DAO entry if there are no DAO parents. See Section 6.2.4.1.1.3

12. Manageability Considerations

The aim of this section is to give consideration to the manageability of RPL, and how RPL will be operated in LLN beyond the use of a MIB module. The scope of this section is to consider the following aspects of manageability: fault management, configuration, accounting and performance.

12.1. Control of Function and Policy

12.1.1. Initialization Mode

When a node is first powered up, it may either choose to stay silent and not send any multicast DIO message until it has joined a DODAG, or to immediately root a transient DODAG and start sending multicast DIO messages. A RPL implementation SHOULD allow configuring whether

the node should stay silent or should start advertising DIO messages.

Furthermore, the implementation SHOULD to allow configuring whether or not the node should start sending an DIS message as an initial probe for nearby DODAGs, or should simply wait until it received DIO messages from other nodes that are part of existing DODAGs.

12.1.2. DIO Base option

RPL specifies a number of protocol parameters.

A RPL implementation SHOULD allow configuring the following routing protocol parameters, which are further described in Section 5.1.1:

DAGPreference
RPLInstanceID
DAGObjectiveCodePoint
DODAGID
Destination Prefixes
DIOIntervalDoublings
DIOIntervalMin
DIORedundancyConstant

DAG Root behavior: In some cases, a node may not want to permanently act as a DODAG root if it cannot join a grounded DODAG. For example a battery-operated node may not want to act as a DODAG root for a long period of time. Thus a RPL implementation MAY support the ability to configure whether or not a node could act as a DODAG root for a configured period of time.

DODAG Table Entry Suppression A RPL implementation SHOULD provide the ability to configure a timer after the expiration of which logical equivalent of the DODAG table that contains all the records about a DODAG is suppressed, to be invoked if the DODAG parent set becomes empty.

12.1.3. Trickle Timers

A RPL implementation makes use of trickle timer to govern the sending of DIO message. Such an algorithm is determined a by a set of configurable parameters that are then advertised by the DODAG root along the DODAG in DIO messages.

For each DODAG, a RPL implementation MUST allow for the monitoring of the following parameters, further described in Section 5.3.5.1:

I
T
C
I_min
I_doublings

A RPL implementation SHOULD provide a command (for example via API, CLI, or SNMP MIB) whereby any procedure that detects an inconsistency may cause the trickle timer to reset.

12.1.4. DAG Sequence Number Increment

A RPL implementation may allow by configuration at the DODAG root to refresh the DODAG states by updating the DODAGSequenceNumber. A RPL implementation SHOULD allow configuring whether or not periodic or event triggered mechanism are used by the DODAG root to control DODAGSequenceNumber change.

12.1.5. Destination Advertisement Timers

The following set of parameters of the DAO messages SHOULD be configurable:

- o The DelayDAO timer

- o The Remove timer

12.1.6. Policy Control

DAG discovery enables nodes to implement different policies for selecting their DODAG parents.

A RPL implementation SHOULD allow configuring the set of acceptable or preferred Objective Functions (OF) referenced by their Objective Codepoints (OCPs) for a node to join a DODAG, and what action should be taken if none of a node's candidate neighbors advertise one of the configured allowable Objective Functions.

A node in an LLN may learn routing information from different routing protocols including RPL. It is in this case desirable to control via administrative preference which route should be favored. An implementation SHOULD allow for specifying an administrative preference for the routing protocol from which the route was learned.

A RPL implementation SHOULD allow for the configuration of the "Route Tag" field of the DAO messages according to a set of rules defined by policy.

12.1.7. Data Structures

Some RPL implementation may limit the size of the candidate neighbor list in order to bound the memory usage, in which case some otherwise viable candidate neighbors may not be considered and simply dropped from the candidate neighbor list.

A RPL implementation MAY provide an indicator on the size of the candidate neighbor list.

12.2. Information and Data Models

The information and data models necessary for the operation of RPL will be defined in a separate document specifying the RPL SNMP MIB.

12.3. Liveness Detection and Monitoring

The aim of this section is to describe the various RPL mechanisms specified to monitor the protocol.

As specified in Section 3.1, an implementation is expected to maintain a set of data structures in support of DODAG discovery:

- o The candidate neighbors data structure

- o For each DODAG:

    * A set of DODAG parents

12.3.1. Candidate Neighbor Data Structure

A node in the candidate neighbor list is a node discovered by the some means and qualified to potentially become of neighbor or a sibling (with high enough local confidence). A RPL implementation SHOULD provide a way monitor the candidate neighbors list with some metric reflecting local confidence (the degree of stability of the neighbors) measured by some metrics.

A RPL implementation MAY provide a counter reporting the number of times a candidate neighbor has been ignored, should the number of candidate neighbors exceeds the maximum authorized value.

12.3.2. Directed Acyclic Graph (DAG) Table

For each DAG, a RPL implementation is expected to keep track of the following DODAG table values:

- o DODAGID

- DAGObjectiveCodePoint
- A set of Destination Prefixes offered upwards along the DODAG
- A set of DODAG Parents
- timer to govern the sending of DIO messages for the DODAG
- DODAGSequenceNumber

The set of DODAG parents structure is itself a table with the following entries:

- A reference to the neighboring device which is the DAG parent
- A record of most recent information taken from the DAG Information Object last processed from the DODAG Parent
- A flag reporting if the Parent is a DAO Parent as described in Section 6

12.3.3. Routing Table

For each route provisioned by RPL operation, a RPL implementation MUST keep track of the following:

- Destination Prefix
- Destination Prefix Length
- Lifetime Timer
- Next Hop
- Next Hop Interface
- Flag indicating that the route was provisioned from one of:
    * Unicast DAO message
    * DIO message
    * Multicast DAO message

12.3.4. Other RPL Monitoring Parameters

A RPL implementation SHOULD provide a counter reporting the number of a times the node has detected an inconsistency with respect to a DODAG parent, e.g. if the DODAGID has changed.

A RPL implementation MAY log the reception of a malformed DIO message along with the neighbor identification if avialable.

12.3.5. RPL Trickle Timers

A RPL implementation operating on a DODAG root MUST allow for the configuration of the following trickle parameters:

- o The DIOIntervalMin expressed in ms
- o The DIOIntervalDoublings
- o The DIORedundancyConstant

A RPL implementation MAY provide a counter reporting the number of times an inconsistency (and thus the trickle timer has been reset).

12.4. Verifying Correct Operation

This section has to be completed in further revision of this document to list potential Operations and Management (OAM) tools that could be used for verifying the correct operation of RPL.

12.5. Requirements on Other Protocols and Functional Components

RPL does not have any impact on the operation of existing protocols.

12.6. Impact on Network Operation

To be completed.

13. Security Considerations

Security Considerations for RPL are to be developed in accordance with recommendations laid out in, for example, [I-D.tsao-roll-security-framework].

14. IANA Considerations

14.1. RPL Control Message

The RPL Control Message is an ICMP information message type that is to be used carry DODAG Information Objects, DODAG Information Solicitations, and Destination Advertisement Objects in support of RPL operation.

IANA has defined a ICMPv6 Type Number Registry. The suggested type value for the RPL Control Message is 155, to be confirmed by IANA.

14.2. New Registry for RPL Control Codes

IANA is requested to create a registry, RPL Control Codes, for the Code field of the ICMPv6 RPL Control Message.

New codes may be allocated only by an IETF Consensus action. Each code should be tracked with the following qualities:

- o Code
- o Description
- o Defining RFC

Three codes are currently defined:

Code	Description	Reference
0x01	DODAG Information Solicitation	This document
0x02	DODAG Information Object	This document
0x04	Destination Advertisement Object	This document

RPL Control Codes

14.3. New Registry for the Control Field of the DIO Base

IANA is requested to create a registry for the Control field of the

DIO Base.

New fields may be allocated only by an IETF Consensus action. Each field should be tracked with the following qualities:

- o Bit number (counting from bit 0 as the most significant bit)
- o Capability description

- o Defining RFC

Four groups are currently defined:

Bit	Description	Reference
0	Grounded DODAG (G)	This document
1	Destination Advertisement Supported (A)	This document
2	Destination Advertisement Trigger (T)	This document
3	Destination Advertisements Stored (S)	This document
5,6,7	DODAG Preference (Prf)	This document

DIO Base Flags

14.4. DODAG Information Object (DIO) Suboption

IANA is requested to create a registry for the DIO Base Suboptions

Value	Meaning	Reference
0	Pad1 - DIO Padding	This document
1	PadN - DIO suboption padding	This document
2	DAG Metric Container	This Document
3	Destination Prefix	This Document
4	DAG Timer Configuration	This Document

DODAG Information Option (DIO) Base Suboptions

15. Acknowledgements

The authors would like to acknowledge the review, feedback, and comments from Emmanuel Baccelli, Dominique Barthel, Yusuf Bashir, Phoebus Chen, Mathilde Durvy, Manhar Goindi, Mukul Goyal, Anders Jagd, Quentin Lampin, Jerry Martocci, Alexandru Petrescu, and Don Sturek.

The authors would like to acknowledge the guidance and input provided by the ROLL Chairs, David Culler and JP Vasseur.

The authors would like to acknowledge prior contributions of Robert Assimiti, Mischa Dohler, Julien Abeille, Ryuji Wakikawa, Teco Boot, Patrick Wetterwald, Bryan Mclaughlin, Carlos J. Bernardos, Thomas Watteyne, Zach Shelby, Caroline Bontoux, Marco Molteni, Billy Moon,

and Arsalan Tavakoli, which have provided useful design considerations to RPL.

16. Contributors

RPL is the result of the contribution of the following members of the ROLL Design Team, including the editors, and additional contributors as listed below:

JP Vasseur
Cisco Systems, Inc
11, Rue Camille Desmoulins
Issy Les Moulineaux, 92782
France

Email: jpv@cisco.com

Thomas Heide Clausen
LIX, Ecole Polytechnique, France

Phone: +33 6 6058 9349
EMail: T.Clausen@computer.org
URI: http://www.ThomasClausen.org/

Philip Levis
Stanford University
358 Gates Hall, Stanford University
Stanford, CA 94305-9030
USA

Email: pal@cs.stanford.edu

Richard Kelsey
Ember Corporation
Boston, MA
USA

Phone: +1 617 951 1225
Email: kelsey@ember.com

Jonathan W. Hui
Arch Rock Corporation
501 2nd St. Ste. 410

San Francisco, CA 94107
USA

Email: jhui@archrock.com

Kris Pister
Dust Networks
30695 Huntwood Ave.
Hayward,   94544
USA

Email: kpister@dustnetworks.com

Anders Brandt
Zensys, Inc.
Emdrupvej 26

Copenhagen, DK-2100
Denmark

Email: abr@zen-sys.com

Stephen Dawson-Haggerty
UC Berkeley
Soda Hall, UC Berkeley
Berkeley, CA 94720
USA

Email: stevedh@cs.berkeley.edu

17. References

17.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

17.2. Informative References

[I-D.ietf-bfd-base]
Katz, D. and D. Ward, "Bidirectional Forwarding Detection", draft-ietf-bfd-base-11 (work in progress), January 2010.

[I-D.ietf-manet-nhdp]
Clausen, T., Dearlove, C., and J. Dean, "Mobile Ad Hoc Network (MANET) Neighborhood Discovery Protocol (NHDP)", draft-ietf-manet-nhdp-11 (work in progress), October 2009.

[I-D.ietf-roll-building-routing-reqs]
Martocci, J., Riou, N., Mil, P., and W. Vermeylen, "Building Automation Routing Requirements in Low Power and Lossy Networks", draft-ietf-roll-building-routing-reqs-09 (work in progress), January 2010.

[I-D.ietf-roll-home-routing-reqs]
Brandt, A. and J. Buron, "Home Automation Routing Requirements in Low Power and Lossy Networks", draft-ietf-roll-home-routing-reqs-11 (work in progress), January 2010.

[I-D.ietf-roll-of0]
Thubert, P., "RPL Objective Function 0", draft-ietf-roll-of0-01 (work in progress), February 2010.

[I-D.ietf-roll-routing-metrics]
Vasseur, J. and D. Networks, "Routing Metrics used for Path Calculation in Low Power and Lossy Networks", draft-ietf-roll-routing-metrics-04 (work in progress), December 2009.

[I-D.ietf-roll-terminology]
Vasseur, J., "Terminology in Low power And Lossy Networks", draft-ietf-roll-terminology-02 (work in progress), October 2009.

[I-D.tsao-roll-security-framework]
Tsao, T., Alexander, R., Dohler, M., Daza, V., and A. Lozano, "A Security Framework for Routing over Low Power and Lossy Networks", draft-tsao-roll-security-framework-01 (work in progress), September 2009.

[Levis08] Levis, P., Brewer, E., Culler, D., Gay, D., Madden, S., Patel, N., Polastre, J., Shenker, S., Szewczyk, R., and A. Woo, "The Emergence of a Networking Primitive in Wireless

Sensor Networks", Communications of the ACM, v.51 n.7, July 2008, <http://portal.acm.org/citation.cfm?id=1364804>.

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, August 1996.

[RFC3697] Rajahalme, J., Conta, A., Carpenter, B., and S. Deering, "IPv6 Flow Label Specification", RFC 3697, March 2004.

[RFC3810] Vida, R. and L. Costa, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, June 2004.

[RFC3819] Karn, P., Bormann, C., Fairhurst, G., Grossman, D., Ludwig, R., Mahdavi, J., Montenegro, G., Touch, J., and L. Wood, "Advice for Internet Subnetwork Designers", BCP 89, RFC 3819, July 2004.

[RFC4101] Rescorla, E. and IAB, "Writing Protocol Models", RFC 4101, June 2005.

[RFC4191] Draves, R. and D. Thaler, "Default Router Preferences and More-Specific Routes", RFC 4191, November 2005.

[RFC4443] Conta, A., Deering, S., and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, March 2006.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007.

[RFC4915] Psenak, P., Mirtorabi, S., Roy, A., Nguyen, L., and P. Pillay-Esnault, "Multi-Topology (MT) Routing in OSPF", RFC 4915, June 2007.

[RFC5120] Przygienda, T., Shen, N., and N. Sheth, "M-ISIS: Multi Topology (MT) Routing in Intermediate System to Intermediate Systems (IS-ISs)", RFC 5120, February 2008.

[RFC5548] Dohler, M., Watteyne, T., Winter, T., and D. Barthel, "Routing Requirements for Urban Low-Power and Lossy Networks", RFC 5548, May 2009.

[RFC5673] Pister, K., Thubert, P., Dwars, S., and T. Phinney, "Industrial Routing Requirements in Low-Power and Lossy Networks", RFC 5673, October 2009.

Appendix A. Requirements

A.1. Protocol Properties Overview

RPL demonstrates the following properties, consistent with the requirements specified by the application-specific requirements documents.

A.1.1. IPv6 Architecture

RPL is strictly compliant with layered IPv6 architecture.

Further, RPL is designed with consideration to the practical support and implementation of IPv6 architecture on devices which may operate under severe resource constraints, including but not limited to memory, processing power, energy, and communication. The RPL design does not presume high quality reliable links, and operates over lossy links (usually low bandwidth with low packet delivery success rate).

A.1.2. Typical LLN Traffic Patterns

Multipoint-to-Point (MP2P) and Point-to-multipoint (P2MP) traffic flows from nodes within the LLN from and to egress points are very common in LLNs. Low power and lossy network Border Router (LBR) nodes may typically be at the root of such flows, although such flows are not exclusively rooted at LBRs as determined on an application-specific basis. In particular, several applications such as building or home automation do require P2P (Point-to-Point) communication.

As required by the aforementioned routing requirements documents, RPL supports the installation of multiple paths. The use of multiple paths include sending duplicated traffic along diverse paths, as well as to support advanced features such as Class of Service (CoS) based routing, or simple load balancing among a set of paths (which could be useful for the LLN to spread traffic load and avoid fast energy depletion on some, e.g. battery powered, nodes). Conceptually, multiple instances of RPL can be used to send traffic along different topology instances, the construction of which is governed by different Objective Functions (OF). Details of RPL operation in support of multiple instances are beyond the scope of the present specification.

A.1.3. Constraint Based Routing

The RPL design supports constraint based routing, based on a set of routing metrics and constraints. The routing metrics and constraints for links and nodes with capabilities supported by RPL are specified in a companion document to this specification,

[I-D.ietf-roll-routing-metrics]. RPL signals the metrics, constraints, and related Objective Functions (OFs) in use in a particular implementation by means of an Objective Code Point (OCP). Both the routing metrics, constraints, and the OF help determine the construction of the Directed Acyclic Graphs (DAG) using a distributed path computation algorithm.

A.2. Deferred Requirements

NOTE: RPL is still a work in progress. At this time there remain several unsatisfied application requirements, but these are to be addressed as RPL is further specified.

Appendix B. Examples

B.1. DAO Operation When Only the Root Node Stores DAO Information

Consider the example of Figure 13. In this example only the root node, (LBR*), will store DAO information. This is not known, nor is it required to be known, to all nodes a priori. Rather, each node is able to observe from the state of the 'S' flag that no ancestor, with the exception of the root node, stores DAO information.

```
                            (LBR*)
                             /  \
                            /    \
                           /      \
                       (11)        (12)
                        |           |
                        |           |
                        |           |
                       (21)        (22)
                           \
                            \
                             \
                             (31)
                             /  \
                            /    \
```

```
                      /       \
                  (41)         (42)
                    :            :
```

Figure 13: Only Root Node Stores DAOs

In this example:

o The 'S' flag is cleared in DIO messages emitted by (LBR*), because (LBR*) is the DODAG root.

o The 'S' flag is cleared in all DIO messages emitted by all other nodes, because no other node stores DAO information.

o (LBR*) has learned from DAO messages how to reach node (31) with a source route via {(11) (21)}.

o All source routes to nodes in the sub-DODAG of node (31), including nodes (41), (42), and others will include the prefix {(11) (21) (31)}

o Node (31) maintains a DTSN, (31).DTSN, that it will advertise in DIO messages.

Suppose now that there is a topology change within the same DODAG iteration, causing node (31) to evict node (21) as a DAO parent and add node (22) as a DAO parent:

1. Node (31) will schedule a DAO transmission because it has added a new node (22) to its DAO parent set.

2. Node (31) need not increment (31).DTSN at this event, because in this example no DAO parents have the 'S' flag set. Specifically this indicates to Node (31) that there are no intermediate storing nodes that may need to be explicitly updated with DAO information from it's sub-DODAG. Hence nodes (41), (42), and by extension the sub-DODAG of node (31) will not subsequently observe an incremented (31).DTSN and the sub-DODAG will not emit DAOs.

3. A new flow of DAOs for node (31) reaches the (LBR*), updating the

source route information for node (31) to include the new path {(12) (22)}.

4. (LBR*) may implicitly update all source routes that must transit node (31), i.e. the sub-DODAG of node (31), to use the updated source route prefix {(12) (22)} instead of {(11) (21)}.

Thus the use of the 'S' flag in the case where only the root node stores DAO information has allowed an optimization whereby only a DAO update for the node that changed its DAO parent set, (31), needs to be sent to the DODAG root.

B.2. DAO Operation When All Nodes Fully Store DAO Information

Consider the example of Figure 14. In this example all nodes will fully store DAO information.

```
              (LBR*)
               /  \
              /    \
             /      \
        (11*)        (12*)
           |           |
           |           |
           |           |
        (21*)        (22*)
             \
              \
               \
              (31*)
               /  \
              /    \
             /      \
        (41*)        (42*)
           :           :
```

Figure 14: All Nodes Store DAOs

In this example:

- The 'S' flag is cleared in DIO messages emitted by (LBR*), because (LBR*) is the DODAG root.

- The 'S' flag is set in DIO messages emitted by all non-root nodes because each non-root node stores DAO information.

- Source routing state is effectively not provisioned in this example, because each node has been able to store hop-by-hop routing state for each destination, possibly aggregated, as learned from DAOs. For example, node (11*) will have learned and stored information from a DAO to the effect that node (41*) is routable through a next hop of node (21*). Node (12*) on the other hand does not necessarily have a route provisioned to node (41*).

Suppose now that there is a topology change within the same DODAG iteration, causing node (31*) to evict node (21*) as a DAO parent and add node (22*) as a DAO parent:

1. Node (31*) will schedule a DAO transmission because it has added a new node (22*) to its DAO parent set.

2. Node (31) need not increment (31).DTSN, because it is a fully storing node and does not need to trigger DAO information from its sub-DODAG.

3. Node (31) gives a DAO Update to node (22*). Presuming that node (22*) has received the update, node (22*) will store the new entries for routes including the sub-DODAG of node (31*), including nodes (41*) and (42*). Node (22*) will schedule a DAO transmission for the new entries.

4. Similarly, node (22*) updates node (12*) and node (12*) updates (LBR*). Hop-by-hop routing state for the sub-DODAG of node (31*) is now provisioned at nodes (12*) and (22*).

Thus the addition to the DAO Parent set at the fully storing node (31*) does not elicit additional DAO-related traffic from its sub-

DODAG. The intermediate nodes along the 'new' downward path are updated by DAO messages along the new path.

Suppose next that the DODAG root triggers a refresh of DAO information over the same DODAG Iteration. (Note that the DODAG root might also trigger a DAO refresh but allow other topology changes at the same time by incrementing the DODAG Sequence Number to cause a move to the next DODAG Iteration).:

1. (LBR*) will increment its DTSN and issue a DIO with the 'T' flag set.

2. Nodes (11*) and (12*) will increment their own DTSNs in response to observing in the DIO from LBR a new DTSN and the 'T' flag being set. They will reset their trickle timers to cause the issue of new DIOs with the 'T' flag set. These nodes will also schedule a DAO transmission in response to observing a new DTSN from their DAO Parent, (LBR*). (This DAO transmission may be scheduled with a sufficient delay computed based on rank to allow a chance for the sub-DODAGs of the nodes to report DAO messages prior to the nodes reporting their own DAO information to (LBR*). This is implementation specific and may allow a chance for DAO aggregation.).

3. Node (21*) receives a DIO from node (11*) and observes the new (11*).DTSN as well as the set 'T' flag. Node (21*) increments its own DTSN, resets the trickle timer, and schedules a DAO transmission.

4. Similarly, as each node observes the incremented DTSN with the 'T' flag set from each of its parents, each node will increment its own DTSN, reset the DIO trickle timer, and schedule a DAO transmission.

Thus the entire DODAG iteration has been re-armed to send DAO messages based on the (LBR*)'s assertion of the 'T' flag. Note that normally a DTSN increment would cause no further action in a sub-DODAG beyond the first fully storing node that is encountered, but that in this case the 'T' flag effectively provides a means to 'punch through' all fully storing nodes.

B.3. DAO Operation When Nodes Have Mixed Capabilities

Consider the example of Figure 15. In this example some nodes are capable of storing DAO information and some are not.

```
                       (LBR*)
                        /  \
                       /    \
                      /      \
                   (11)        (12*)
                     |           |
                     |           |
                     |           |
                   (21)        (22)
                      \
                       \
                        \
                        (31)
                        /  \
                       /    \
                      /      \
                   (41)        (42*)
                     :           :
```

Figure 15: Mixed Capability DAO Operation

In this example:

o The 'S' flag is cleared in DIO messages emitted by (LBR*), because (LBR*) is the DODAG root.

o The 'S' flag is set in DIO messages emitted by (12*), because it is a storing node.

o The 'S' flag will be set in DIO messages emitted by nodes that contain node (12*) (or more generally any non-root storing node) as a DAO parent/ancestor. This indicates that somewhere along the DAO path there is a non-root storing node that may need to have its state updated (by a DAO refresh) in certain conditions.

Suppose that there is a topology change within the same DODAG iteration, causing node (31) to add node (22) as a DAO parent:

1. Node (31) will schedule a DAO transmission because it has added a new node (22) to its DAO parent set. Node (31) will increment (31).DTSN because node (22) has set the 'S' flag in its DIO messages. Node (31) will reset its DIO trickle timer.

2. Node (31)'s trickle timer will then expire and a DIO is issued and received by node's (41) and (42*).

3. Node (41) is a non-storing node. It will increment (41).DTSN in response to observing the increment in (31).DTSN, and reset its trickle timer. This results finally in the reliable (thanks to the DTSN) triggering of a DAO update from node (41)'s sub-DODAG.

4. As node (41) receives DAO updates from its sub-DODAG it updates the DAOs with source routing information as necessary and passes them on to node (31), along with its own (node (41)) DAO update.

5. Meanwhile, node (42*) is a fully storing node. It observes the increment to (31).DTSN and schedules a DAO update. Node (42*) does not need to increment (42*).DTSN, since it is a fully storing node it does not need to solicit DAO updates from its sub-DODAG in this case. At the scheduled time Node (42*) reissues its DAO information to node (31).

6. Node (31) receives the DAO messages from its sub-DODAG, adds source routing information as necessary, and issues DAO updates to node (22).

7. Node (22) similarly receives DAO messages from node (31), updates source routing information as necessary, and issues DAO updates to node (12*).

8. The intermediate storing node (12*) has now received from DAO messages the information necessary to provision routing state for node (31) and its sub-DODAG. As downwards traffic is routed through node (12*) it is able to consult source routing information that was learned from the DAO messages as needed to specify routes down the DAG across the non-storing nodes (22), (31), ...

Appendix C. Outstanding Issues

This section enumerates some outstanding issues that are to be addressed in future revisions of the RPL specification.

C.1. Additional Support for P2P Routing

In some situations the baseline mechanism to support arbitrary P2P traffic, by flowing upwards along the DODAG until a common ancestor is reached and then flowing down, may not be suitable for all application scenarios. A related scenario may occur when the down paths setup along the DODAG by the destination advertisement mechanism are not the most desirable downward paths for the specific application scenario (in part because the DODAG links may not be symmetric). It may be desired to support within RPL the discovery and installation of more direct routes 'across' the DAG. Such mechanisms need to be investigated.

C.2. Destination Advertisement / DAO Fan-out

When DAO messages are relayed to more than one DODAG parent, in some cases a situation may be created where a large number of DAO messages conveying information about the same destination flow upwards along the DAG. It is desirable to bound/limit the multiplication/fan-out of DAO messages in this manner. Some aspects of the Destination Advertisement mechanism remain under investigation, such as behavior in the face of links that may not be symmetric.

In general, the utility of providing redundancy along downwards routes by sending DAO messages to more than one parent is under investigation.

C.3. Source Routing

In support of nodes that maintain minimal routing state, and to make use of the collection of piecewise source routes from the destination advertisement mechanism, there needs to be some investigation of a mechanism to specify, attach, and follow source routes for packets traversing the LLN.

C.4. Address / Header Compression

In order to minimize overhead within the LLN it is desirable to perform some sort of address and/or header compression, perhaps via labels, addresses aggregation, or some other means. This is still under investigation.

C.5. Managing Multiple Instances

A network may run multiple instances of RPL concurrently. Such a network will require methods for assigning and otherwise managing RPLInstanceIDs. This will likely be addressed in a separate document.

Authors' Addresses

Tim Winter (editor)

Email: wintert@acm.org

Pascal Thubert (editor)
Cisco Systems
Village d'Entreprises Green Side
400, Avenue de Roumanille
Batiment T3
Biot - Sophia Antipolis 06410
FRANCE

Phone: +33 497 23 26 34
Email: pthubert@cisco.com

ROLL Design Team
IETF ROLL WG

Email: rpl-authors@external.cisco.com