Networking Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 7, 2010

T. Winter, Ed.

P. Thubert, Ed.
Cisco Systems
ROLL Design Team
IETF ROLL WG
October 4, 2009

RPL: Routing Protocol for Low Power and Lossy Networks
draft-ietf-roll-rpl-03

Status of this Memo

Copyright Notice

Abstract

Low Power and Lossy Networks (LLNs) are made largely of constrained nodes (with limited processing power, memory, and sometimes energy when they are battery operated). These routers are interconnected by lossy links, most of the time supporting only low data rates, that are usually fairly 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 a large number of nodes, up to several dozens or hundreds or more nodes in the network. 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. This document specifies the Routing Protocol for Low Power and Lossy Networks (RPL).

Requirements Language

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

Table of Contents

1. Introduction

Low Power and Lossy Networks (LLNs) are made largely of constrained nodes (with limited processing power, memory, and sometimes energy when they are battery operated). These routers are interconnected by lossy links, most of the time supporting only low data rates, that are usually fairly 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 a large number of nodes, up to several dozens or hundreds or more nodes in the network. 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], [I-D.ietf-roll-indus-routing-reqs], and [RFC5548]. This document specifies the 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], [I-D.ietf-roll-indus-routing-reqs], 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. (Note: it is intended that as this 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 instantiation of the protocol to be optimized in terms of required code space. It must be noted that RPL is not restricted to the aforementioned applications and is expected to be used in other environments. All "MUST" application requirements that cannot be satisfied by RPL will be specifically listed in the Appendix A, accompanied by a justification.

The core set of functionalities is to be capable of operating in the most severely constrained environments, with minimal requirements for memory, energy, processing, communication, and other consumption of limited resources from nodes. Trade-offs inherent in the provisioning of protocol features will be exposed to the implementer in the form of configurable parameters, such that the implementer can

further tweak and optimize the operation of RPL as appropriate to a specific application and implementation. Finally, RPL is designed to consult implementation specific policies to determine, for example, the evaluation of routing metrics.

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 Behavior

This specification does not rely on any particular features of a specific link layer technologies. It is anticipated that an implementer should be able to operate RPL over a variety of different low power wireless or PLC (Power Line Communication) link layer 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 terminology used in this document is consistent with and incorporates that described in `Terminology in Low power And Lossy Networks' [I-D.ietf-roll-terminology]. The terminology is extended in this document as follows:

Autonomous: The ability of a routing protocol to independently function without relying on any external influence or guidance. Includes self-organization capabilities.

DAG: Directed Acyclic Graph. A directed graph having the property that all edges are oriented in such a way that no cycles exist. In the RPL context, all edges are contained in paths oriented toward and terminating at a root node (a DAG root, or sink-typically a Low Power and Lossy Network Border Router (LBR)).

DAGID: DAG Identifier. A globally unique identifier for a DAG. All nodes who are part of a given DAG have knowledge of the DAGID. This knowledge is used to identify peer nodes within the DAG in order to coordinate DAG maintenance while avoiding loops.

DAG parent: A parent of a node within a DAG is one of the immediate successors of the node on a path towards the DAG root. For each DAGID that a node is a member of, the node will maintain a set containing one or more DAG parents. If a node is a member of multiple DAGs then it must conceptually maintain a set of DAG parents for each DAGID.

DAG sibling: A sibling of a node within a DAG is defined in this specification to be any neighboring node which is located at the same rank (depth) within a DAG. Note that siblings defined in this manner do not necessarily share a common parent. For each DAG that a node is a member of, the node will maintain a set of DAG siblings. If a node is a member of multiple DAGs then it must conceptually maintain a set of DAG siblings for each DAG.

DAG root: A DAG root is a sink within the DAG. All paths in the DAG terminate at a DAG root, and all DAG edges contained in the paths terminating at a DAG root are oriented toward the DAG root. There must be at least one DAG root per DAG, and in some cases there may be more than one. In many use cases, source-sink represents a dominant traffic flow, where the sink is a DAG root or is located behind the DAG root. Maintaining routes towards DAG roots is therefore a prominent functionality for RPL.

Grounded: A DAG is grounded if it contains a DAG root offering connectivity to an external routed infrastructure such as the public Internet or a private core (non-LLN) IP network.

Floating: A DAG is floating if is not grounded. A floating DAG is not expected to reach any additional external routed infrastructure such as the public Internet or a private core (non-LLN) IP network.

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

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

P2P: Point-to-point. This refers to traffic exchanged between two nodes.

P2MP: Point-to-Multipoint. This refers to traffic between one node and a set of nodes. This is similar to the P2MP concept in Multicast or MPLS Traffic Engineering ([RFC4461] and [RFC4875]). A common RPL use case involves P2MP flows from or through a DAG root outward towards other nodes contained in the DAG.

MP2P: Multipoint-to-Point; used to describe a particular traffic pattern. A common RPL use case involves MP2P flows collecting information from many nodes in the DAG, flowing inwards towards DAG roots. Note that a DAG root may not be the ultimate destination of the information, but it is a common transit node.

OCP: Objective Code Point. In RPL, the Objective Code Point (OCP) indicates which routing metrics, optimization objectives, and related functions are in use in a DAG. Instances of the Objective Code Point are further described in [I-D.ietf-roll-routing-metrics].

Note that in this document, the terms `node' and `LLN router' are used interchangeably.

3. Protocol Model

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

3.1. Protocol Properties Overview

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

3.1.1. 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).

3.1.2. 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).

3.1.3. Constraint Based Routing

The RPL design supports constraint based routing, based on a set of routing metrics. The routing metrics 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 and related objective functions in use in a particular implementation by means of an Objective Code Point (OCP). Both the routing metrics and the OCP help determine the construction of the Directed Acyclic Graphs (DAG) using a distributed path computation algorithm.

RPL supports the computation and installation of different paths in support of and optimized for a set of application and implementation specific constraints, as guided by an OCP. Traffic may subsequently be directed along the appropriate constrained path based on traffic marking within the IPv6 header. For more details on the approach towards constraint-based routing, see Section 4.

3.2. Protocol Operation

A LLN deployment will consist of a number of nodes and a number of edges (links) between them, whose characteristics will depend on implementation and link layer (L2) specifics. Due to the nature of the LLN environment the L2 links are expected to demonstrate a large degree of variance as to their availability, quality, and other related parameters. Certain links, demonstrating a viability above a confidence threshold for particular node and link metrics, as based on guidelines from [I-D.ietf-roll-routing-metrics], will be extracted from the L2 graph, and the resulting graph will be used as the basis

on which to operate the routing protocol. Note that as the characteristics of the L2 topology vary over time the set of viable links is to be updated and the routing protocol thus continues to evaluate the LLN. In RPL this process happens in a distributed manner, and from the perspective of a single node running RPL this process results in a set of candidate neighbors, with associated node and link metrics as well as confidence values.

Many of the dominant traffic flows in support of the LLN application scenarios are MP2P flows ([I-D.ietf-roll-building-routing-reqs], [I-D.ietf-roll-home-routing-reqs], [I-D.ietf-roll-indus-routing-reqs], and [RFC5548]). These flows are rooted at designated nodes that have some application significance, such as providing connectivity to an external routed infrastructure. The term "external" is used top refer to the public Internet or a core private (non-LLN) IP network. In support of this dominant flow RPL constructs Directed Acyclic Graphs (DAGs) on top of the viable LLN topology, selecting and orienting links among candidate neighbors toward DAG roots which root the MP2P flows.

LLN nodes running RPL will construct Directed Acyclic Graphs (DAGs) rooted at designated nodes that generally have some application significance, such as providing connectivity to an external routed infrastructure. The term "external" is used top refer to the public Internet or a core private (non-LLN) IP network. This structure provides the routing solution for the dominant MP2P traffic flows. The DAG structure further provides each node potentially multiple successors for MP2P flows, which may be used for, e.g., local route repair or load balancing.

Nodes running RPL are able to further restrict the scope of the routing problem by using the DAG as a reference topology. By referencing a rank property that is related to the positions in the DAG, nodes are able to determine their positions in a DAG relative to each other. This information is used by RPL in part to construct rules for movement relative to the DAG that endeavor to avoid loops. It is important to note that the rank property is derived from metrics, and not directly from the position in the DAG, as will be discussed further.

As DAGs are organized, RPL will use a destination advertisement mechanism to build up routing tables in support of outward P2MP traffic flows. This mechanism, using the DAG as a reference, distributes routing information across intermediate nodes (between the DAG leaves and the root), guided along the DAG, such that the routes toward destination prefixes in the outward direction may be set up. As the DAG undergoes modification during DAG maintenance, the destination advertisement mechanism can be triggered to update

the outward routing state.

A baseline support for P2P traffic in RPL is provided by the DAG, as P2P traffic may flow inward along the DAG until a common parent is reached who has stored an entry for the destination in its routing table and is capable of directing the traffic outward along the correct outward path. RPL also provides support for the trivial case where a P2P destination may be a `one-hop' neighbor. In the present specification RPL does not specify nor preclude any additional mechanisms that may be capable to compute and install more optimal routes into LLN nodes in support of arbitrary P2P traffic according to some routing metric.

3.2.1. DAG Construction

RPL constructs one or more DAGs, over gradients defined by optimizing cost metrics along paths rooted at designated nodes.

The DAG construction algorithm is distributed; each node running RPL invokes a set of DAG construction rules and objective functions when considering its role with respect to neighboring nodes such that the DAG structure emerges.

3.2.1.1. IP Router Advertisement - DAG Information Option (RA-DIO)

The IPv6 Router Advertisement (RA) mechanism (as specified in [RFC4861]) is used by RPL in order to build and maintain a DAG.

The IPv6 RA message is augmented with a DAG Information Option (DIO), forming an RA-DIO message, to convey information about the DAG including:

- o A DAGID used to identify the DAG as sourced from the DAG root. The DAGID must be unique to a single DAG in the scope of the LLN.

- o Objective Code Point (OCP) as described below.

- o Rank information used by nodes to determine their positions in the DAG relative to each other. This is not a metric, although its derivation is typically closely related to one or more metrics as specified by the OCP. The rank information is used to support loop avoidance strategies and in support of ordering alternate successors when engaged in path maintenance.

- o Sequence number originated from the DAG root, used to aid in identification of dependent sub-DAGs and coordinate topology changes in a manner so as to avoid loops.

- o Indications and configuration for the DAG, e.g. grounded or floating, administrative preference, ...

- o A vector of path metrics, as further described in [I-D.ietf-roll-routing-metrics].

- o List of additional destination prefixes reachable inwards along the DAG.

The RA messages are issued whenever a change is detected to the DAG such that a node is able to determine that a region of the DAG has become inconsistent. As the DAG stabilizes the period at which RA messages occur is configured to taper off, reducing the steady-state overhead of DAG maintenance. The periodic issue of RA messages, along with the triggered RA messages in response to inconsistency, is one feature that enables RPL to operate in the presence of unreliable links.

3.2.1.2. DAG Identification

Each DAG is identified by a particular identifier (DAGID) as well as its supported optimization objectives and available destination prefixes. The optimization objectives are conveyed as an Objective Code Point (OCP) as described further below. Available destination prefixes, which may include destinations available beyond the DAG root, multicast destinations, or IPv6 node addresses, are advertised outwards along the DAG and recipient nodes may then provision routing tables with entries inwards towards the destinations. The RPL implementation at each node will be provisioned by the application with sufficient information to determine which objectives and destinations are required, and thus the RPL implementation may determine which DAG to join.

The decision for a node to join a DAG may be optimized according to implementation specific policy functions on the node as indicated by one or more specific OCP values. For example, a node may be configured for one goal to optimize a bandwidth metric (OCP-1), and with a parallel goal to optimize for a reliability metric (OCP-2). Thus two DAGs, with two unique DAGIDs, may be constructed and maintained in the LLN: DAG-1 would be optimized according to OCP-1, whereas DAG-2 would be optimized according to OCP-2. A node may then maintain independent sets of DAG parents and related data structures for each DAG. Note that in such a case traffic may directed along the appropriate constrained DAG based on traffic marking within the IPv6 header. This specification will focus on the case where the node only joins one DAG; further elaboration on the proper operation of RPL in the presence of multiple DAGs, including traffic marking and related rules, are to be specified further in future revisions of

this or companion specifications.

3.2.1.3. Grounded and Floating DAGs

Certain LLN nodes may offer connectivity to an external routed infrastructure in support of an application scenario. These nodes are designated `grounded', and may serve as the DAG roots of a grounded DAG. DAGs that do not have a grounded DAG root are floating DAGs. In either case routes may be provisioned toward the DAG root, although in the floating case there is no expectation to reach an external infrastructure. Some applications will include permanent floating DAGs.

3.2.1.4. Administrative Preference

An administrative preference may be associated with each DAG root, and thereby each DAG, in order that some DAGs in the LLN may be more preferred over other DAGs. For example, a DAG root that is sinking traffic in support of a data collection application may be configured by the application to be very preferred. A transient DAG, e.g. a DAG that is only existing in support of DAG repair until a permanent DAG is found, may be configured to be less preferred. The administrative preference offers a way to engineer the formation of the DAG in support of the application.

3.2.1.5. Objective Code Point (OCP)

The OCP serves to convey and control the optimization objectives in use within the DAG. The OCP is further specified in [I-D.ietf-roll-routing-metrics]. Each instance of an allocated OCP indicates:

- The set of metrics used within the DAG
- The objective functions used for least cost path determination.
- The function used to compute DAG Rank
- The functions used to accumulate metrics for propagation within a RA-DIO message

For example, an objective code point might indicate that the DAG is using the Expected Number of Transmissions (ETX) as a metric, that the optimization goal is to minimize ETX, that DAG Rank is equivalent to ETX, and that RA-DIO propagation entails adding the advertised ETX of the most preferred parent to the ETX of the link to the most preferred parent.

By using defined OCPs that are understood by all nodes in a particular implementation, and by conveying them in the RA-DIO message, RPL nodes may work to build optimized LLN using a variety of application and implementation specific metrics and goals.

A default OCP, OCP 0, is specified with a well-defined default behavior. OCP 0 is used to define RPL behaviors in the case where a node encounters a RA-DIO message containing a code point that it does not support.

3.2.1.6. Distributed Algorithm Operation

- o Some nodes may be initially provisioned to act as DAG roots, either permanent or transient, with associated preferences.

- o Nodes will maintain a data structure containing their candidate (viable) neighbors, as based on guidelines in [I-D.ietf-roll-routing-metrics] This data structure will also track DAG information as learned from and associated with each neighbor.

- o Nodes who are members of a DAG, including DAG roots, will multicast RA-DIO messages as needed (when inconsistency is detected), to their link-local neighbors. Nodes will also respond to Router Solicitation (RS) messages.

- o Nodes who receive RA-DIO messages will take into consideration several criteria when processing the extracted DAG information. The node may discount the RA-DIO according to loop avoidance rules based on rank as described further below. Nodes will consider the information in the RA-DIO in order to determine whether or not that candidate neighbor offers a better attachment point to a DAG (which the node may or may not be a member of) according to the implementation specific optimization goals, OCP, and current metrics.

- o Nodes may join a new DAG or move within the current DAG, in response to the information contained in the RA-DIO message, and in accordance with loop avoidance rules described further in this specification. For the successors within the DAG, a node manages a set of DAG Parents. Joining, moving within, and leaving the DAG is accomplished through managing this set according to the rules specified by RPL.

- o As nodes join, move within, and leave DAGs they emit updated RA-DIOs which are received and acted on by neighboring nodes. When inconsistencies (such as caused by movement or link loss) are detected within the DAG structure, RA-DIO messages are emitted more frequently. When the DAG structure becomes consistent, RA-DIO messages taper off.

- As less preferred DAGs encounter more preferred DAGs that offer equivalent or better optimization objectives, the nodes in the less preferred DAGs may leave to join the more preferred DAGs, finally leaving only the more preferred DAGs. This is an illustration of the mechanism by which an application may engineer DAG construction.

- As the DAG construction operation proceeds, nodes accumulate onto the DAG in progressively outward tiers, centered around the DAG root.

- The nodes provision routing table entries for the destinations specified by the RA-DIO towards their DAG Parents. Nodes may provision a DAG Parent as a default gateway.

3.2.1.7. DAG Rank

When nodes select DAG parents, they will select the most preferred parent according to their implementation specific objectives, using the cost metrics conveyed in the RA-DIO messages along the DAG in conjunction with the related objective functions as specified by the OCP.

Based on this selection, the metrics conveyed by the most preferred DAG parent, the nodes own metrics and configuration, and a related function defined by the OCP, a node will be able to compute a value for its rank as a consequence of selecting a most preferred DAG parent.

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

It is important to note that the DAG Rank is not itself a metric, although its value is derived from and influenced by the use of metrics to select DAG parents and take up a position in the DAG. In other words, routing metrics and OCP (not rank directly) are used to determine the DAG structure and consequently the path cost. The only aim of the rank is to inform loop avoidance as explained hereafter. The computation of the DAG Rank MUST be done in such a way so as to maintain the following properties for any nodes M and N who are neighbors in the LLN:

For a node N, and its most preferred parent M, DAGRank(N) > DAGRank(M) must hold. Further, all parents in the DAG parent set must be of a rank less than self's DAGRank(N). In other words, the rank presented by a node N MUST be greater (deeper) than that presented by any of its parents.

If DAGRank(M) < DAGRank(N), then M is probably located in a more preferred position than N in the DAG with respect to the metrics and optimizations defined by the objective code point. In any fashion, Node M may safely be a DAG parent for Node N without risk of creating a loop.

> For example, a Node M of rank 3 is likely located in a more optimum position than a Node N of rank 5. A packet directed inwards and forwarded from Node N to Node M will always make forward progress with respect to the DAG organization on that link; there is no risk of Node M at rank 3 forwarding the packet back into Node N's sub-DAG at rank of 5 or greater (which would be a sufficient condition for a loop to occur).

If DAGRank(M) == DAGRank(N), then M and N are located positions of relatively the same optimality within the DAG. In some cases, Node M may be used as a successor by Node N, but with related chance of creating a loop that must be detected and broken by some other means.

> If Node M is at rank 3 and node N is at rank 3, then they are siblings; by definition Node M and N cannot be in each others sub-DAG. They may then forward to each other failing serviceable parents, making `sideways' progress (but not reverse progress). If another sibling or more gets involved there may then be some chance for 3 or more way loops, which is the risk of sibling forwarding.

If DAGRank(M) > DAGRank(N), then node M is located in a less preferred position than N in the DAG with respect to the metrics and optimizations defined by the objective code point. Further, Node (M) may in fact be in Node (N)'s sub-DAG. There is no advantage to Node (N) selecting Node (M) as a DAG parent, and such a selection may create a loop.

For example, if Node M is of rank 3 and Node N is of rank 5, then by definition Node N is in a less optimum position than Node N. Further, Node N at rank 5 may in fact be in Node M's own sub-DAG, and forwarding a packet directed inwards towards the DAG root from M to N will result in backwards progress and possibly a loop.

As an example, the DAG 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 code point being used within the DAG.

The DAG rank is subsequently used to restrict the options a node has for movement within the DAG and to coordinate movements in order to avoid the creation of loops.

3.2.1.8. Sub-DAG

The sub-DAG of a node is the set of other nodes of greater rank in the DAG, and thus might use a path towards the DAG root that contains this node. This is an important property that is leveraged for loop avoidance- if a node has lesser rank then it is not in the sub-DAG. (An arbitrary node with greater rank may or may not be contained in the sub-DAG). Paths through siblings are not contained in this set.

As a further illustration, consider the DAG examples in Appendix B. Consider Node (24) in the DAG Example depicted in Figure 9. In this example, the sub-DAG of Node (24) is comprised of Nodes (34), (44), and (45).

A frozen sub-DAG is a subset of nodes in the sub-DAG of a node who have been informed of a change to the node, and choose to follow the node in a manner consistent with the change, for example in preparation for a coordinated move. Nodes in the sub-DAG who hear of a change and have other options than to follow the node do not have to become part of the frozen sub-DAG, for example such a node may be able to remain attached to the original DAG through a different DAG parent. A further example may be found in Appendix B.8.

3.2.1.9. Moving up in a DAG

A node may safely move `up' in the DAG, causing its DAG rank to decrease and moving closer to the DAG root without risking the formation of a loop.

3.2.1.10. Moving down in a DAG

A node may not consider to move `down' the DAG, causing its DAG rank to increase and moving further from the DAG root. In the case where a node looses connectivity to the DAG, it must first leave the DAG before it may then rejoin at a deeper point. This allows for the node to coordinate moving down, freezing its own sub-DAG and poisoning stale routes to the DAG, and minimizing the chances of re-attaching to its own sub-DAG thinking that it has found the original

DAG again. If a node where allowed to re-attach into its own sub-DAG a loop would most certainly occur, and may not be broken until a count-to-infinity process elapses. The procedure of detaching before moving down eliminates the need to count-to-infinity.

3.2.1.11. DAG Jumps

A jump from one DAG to another DAG is attaching to a new DAGID, in such a way that an old DAGID is replaced by the new DAGID. In particular, when an old DAGID is left, all associated parents are no longer feasible, and a new DAGID is joined.

When a node in a DAG follows a DAG parent, it means that the DAG parent has changed its DAGID (e.g. by joining a new DAG) and that the node updates its own DAGID in order to keep the DAG parent.

3.2.1.12. Floating DAGs for DAG Repair

A DAG may also be floating. Floating DAGs may be encountered, for example, during coordinated reconfigurations of the network topology wherein a node and its sub-DAG breaks off the DAG, temporarily becomes a floating DAG, and reattaches to a grounded DAG. (Such coordination endeavors to avoid the construction of transient loops in the LLN).

A DAG, or a sub-DAG temporarily promoted to a DAG, may also become floating because of a network element failure. If the DAG parent set of the node becomes completely depleted, the node will have detached from the DAG, and may, if so configured, become the root of its own transient floating DAG with a less desirable administrative preference (thus beginning the process of establishing the frozen sub-DAG), and then may reattach to the original DAG at a lower point if it is able (after hearing RA-DIO messages from alternate attachment points).

3.2.2. Destination Advertisement

As RPL constructs DAGs, nodes may provision routes toward destinations advertised through RA-DIO messages through their selected parents, and are thus able to send traffic inward along the DAG by forwarding to their selected parents. However, this mechanism alone is not sufficient to support P2MP traffic flowing outward along the DAG from the DAG root toward nodes. A destination advertisement mechanism is employed by RPL to build up routing state in support of these outward flows. The destination advertisement mechanism may not be supported in all implementations, as appropriate to the application requirements. A DAG root that supports using the destination advertisement mechanism to build up routing state will

indicate such in the RA-DIO message. A DAG root that supports using the destination advertisement mechanism must be capable of allocating enough state to store the routing state received from the LLN.

3.2.2.1. IPv6 Neighbor Advertisement - Destination Advertisement Option (NA-DAO)

An IPv6 Neighbor Advertisement Message with Destination Advertisement Options (NA-DAO) is used to convey the destination information inward along the DAG toward the DAG root.

The information conveyed in the NA-DAO message includes the following:

- o A lifetime and sequence counter to determine the freshness of the destination advertisement.

- o Depth information used by nodes to determine how far away the destination (the source of the destination advertisement) is o Prefix information to identify the destination, which may be a prefix, an individual host, or multicast listeners

o Reverse Route information to record the nodes visited (along the outward path) when the intermediate nodes along the path cannot support storing state for Hop-By-Hop routing.

3.2.2.2. Destination Advertisement Operation

As the DAG is constructed and maintained, nodes are capable to emit NA-DAO messages to a subset, or all, of their DAG parents. The selection of this subset is according to an implementation specific policy.

As a special case, a node may periodically emit a link-local multicast IPv6 NA-DAO message advertising its locally available destination prefixes. This mechanism allows for the one-hop neighbors of a node to learn explicitly of the prefixes on the node, and in some application specific scenarios this is desirable in support of provisioning a trivial `one-hop' route. In this case, nodes who receive the multicast destination advertisement may use it to provision the one-hop route only, and not engage in any additional processing (so as not to engage the mechanisms used by a DAG parent).

When a (unicast) NA-DAO message reaches a node capable of storing routing state, the node extracts information from the NA-DAO message and updates its local database with a record of the NA-DAO message and who it was received from. When the node later propagates NA-DAO

messages, it selects the best (least depth) information for each destination and conveys this information again in the form of NA-DAO messages to a subset of its own DAG parents. At this time the node may perform route aggregation if it is able, thus reducing the overall number of NA-DAO messages.

When a (unicast) NA-DAO message reaches a node incapable of storing additional state, the node must append the next-hop address (from which neighbor the NA-DAO message was received) to a Reverse Route Stack carried within the NA-DAO message. The node then passes the NA-DAO message on to one or more of its DAG parents without storing any additional state.

When a node that is capable of storing routing state encounters a (unicast) NA-DAO message with a Reverse Route Stack that has been populated, the node knows that the NA-DAO message has traversed a region of nodes that did not record any routing state. The node is able to detach and store the Reverse Route State and associate it with the destination described by the NA-DAO message. Subsequently the node may use this information to construct a source route in order to bridge the region of nodes that are unable to support Hop-By-Hop routing to reach the destination.

In this way the destination advertisement mechanism is able to provision routing state in support of P2MP traffic flows outward along the DAG, and as according to the available resources in the network.

Further aggregations of NA-DAO messages prefix reachability information by destinations are possible in order to support additional scalability.

A further example of the operation of the destination advertisement mechanism is available in Appendix B.6

3.3. Loop Avoidance and Stability

The goal of a guaranteed consistent and loop free global routing solution for an LLN may not be practically achieved given the real behavior and volatility of the underlying metrics. The trade offs to achieve a stable approximation of global convergence may be too restrictive with respect to the need of the LLN to react quickly in response to the lossy environment. Globally the LLN may be able to achieve a weak convergence, in particular as link changes are able to be handled locally and result in minimal changes to global topology.

RPL does not aim to guarantee loop free path selection, or strong global convergence. In order to reduce control overhead, in

particular the expense of mechanisms such as count-to-infinity, RPL does try to avoid the creation of loops when undergoing topology changes. Further mechanisms to mitigate the impact of loops, such as loop detection when forwarding, are under investigation.

3.3.1. Greediness and Rank-based Instabilities

If a node is greedy and attempts to move deeper in the DAG, beyond its most preferred parent, in order to increase the size of the DAG parent set, then an instability can result. This is illustrated in Figure 11.

Suppose a node is willing to receive and process a RA-DIO messages from a node in its own sub-DAG, and in general a node deeper than it. In such cases a chance exists to create a feedback loop, wherein two or more nodes continue to try and move in the DAG in order to optimize against each other. In some cases this will result in an instability. It is for this reason that RPL mandates that a node never receive and process RA-DIO messages from deeper nodes. This rule creates an `event horizon', whereby a node cannot be influenced into an instability by the action of nodes that may be in its own sub-DAG.

A further example of the consequences of greedy operation, and instability related to processing RA-DIO messages from nodes of greater rank, may be found in Appendix B.9

3.3.2. Merging DAGs

The merging of DAGs is coordinated in a way such as to try and merge two DAGs cleanly, preserving as much DAG structure as possible, and in the process effecting a clean merge with minimal likelihood of forming transient DAG loops. The coordinated merge is also intended to minimize the related control cost.

When a node, and perhaps a related frozen sub-DAG, jumps to a different DAG, the move is coordinated by a set of timers (DAG Hop timers). The DAG Hop timers allow the nodes who will attach closer to the sink of the new DAG to `jump' first, and then drag dependent nodes behind them, thus endeavoring to efficiently coordinate the attachment of the frozen sub-DAG into the new DAG.

A further example of a DAG Merge operation may be found in Appendix B.10

3.3.3. DAG Loops

A DAG loop may occur when a node detaches from the DAG and reattaches to a device in its prior sub-DAG that has missed the whole detachment sequence and kept advertising the original DAG. This may happen in particular when RA-DIO messages are missed. Use of the DAG sequence number can eliminate this type of loop. If the DAG sequence number is not in use, the protection is limited (it depends on propagation of RA-DIO messages during DAG hop timer), and temporary loops might occur. RPL will move to eliminate such a loop as soon as a RA-DIO message is received from a parent that appears to be going down, as the child has to detach from it immediately. (The alternate choice of staying attached and following the parent in its fall would have counted to infinity and led to detach as well).

Consider node (24) in the DAG Example depicted in Figure 9, and its sub-DAG nodes (34), (44), and (45). An example of a DAG loop would be if node (24) were to detach from the DAG rooted at (LBR), and nodes (34) and (45) were to miss the detachment sequence. Subsequently, if the link (24)--(45) were to become viable and node (24) heard node (45) advertising the DAG rooted at (LBR), a DAG loop (45->34->24->45) may form if node (24) attaches to node (45).

3.3.4. DAO Loops

A DAO loop may occur when the parent has a route installed upon receiving and processing a NA-DAO message from a child, but the child has subsequently cleaned up the state. This loop happens when a no-DAO was missed till a heartbeat cleans up all states. The DAO loop is not explicitly handled by the current specification. Split horizon, not forwarding a packet back to the node it came from, may mitigate the DAO loop in some cases, but does not eliminate it.

Consider node (24) in the DAG Example depicted in Figure 9. Suppose node (24) has received a DA from node (34) advertising a destination at node (45). Subsequently, if node (34) tears down the routing state for the destination and node (24) did not hear a no-DAO message to clean up the routing state, a DAO loop may exist. node (24) will forward traffic destined for node (45) to node (34), who may then naively return it into a loop (if split horizon is not in place). A more complicated DAO loop may result if node (34) instead passes the traffic to it's sibling, node (33), potentially resulting in a (24->34->33->23->13->24) loop.

3.3.5. Sibling Loops

Sibling loops occur when a group of siblings keep choosing amongst themselves as successors such that a packet does not make forward progress. The current draft limits those loops to some degree by split horizon (do not send back to the same sibling) and parent preference (always prefer parents vs. siblings).

Consider the DAG Example depicted in Figure 9. Suppose that Node (32) and (34) are reliable neighbors, and thus are siblings. Then, in the case where Nodes (22), (23), and (24) are transiently unavailable, and with no other guiding strategy, a sibling loop may exist, e.g. (33->34->32->33) as the siblings keep choosing amongst each other in an uncoordinated manner.

3.4. Local and Temporary Routing Decision

Although implementation specific, it is worth noting that a node may decide to implement some local routing decision based on some metrics, as observed locally or reported in the RA-DIO message. For example, the routing may reflect a set of successors (next-hop), along with various aggregated metrics used to load balance the traffic according to some local policy. Such decisions are local and implementation specific.

Routing stability is crucial in a LLN: in the presence of unstable links, the first option consists of removing the link from the DAG and triggering a DAG recomputation across all of the nodes affected by the removed link. Such a naive approach could unavoidably lead to frequent and undesirable changes of the DAG, routing instability, and high-energy consumption. The alternative approach adopted by RPL relies on the ability to temporarily not use a link toward a successor marked as valid, with no change on the DAG structure. If the link is perceived as non-usable for some period of time (locally configurable), this triggers a DAG recomputation, through the DAG discovery mechanism further detailed in Section 5.3, after reporting the link failure. Note that this concept may be extended to take into account other link characteristics: for the sake of illustration, a node may decide to send a fixed number of packets to a particular successor (because of limited buffering capability of the successor) before starting to send traffic to another successor.

According to the local policy function, it is possible for the node to order the DAG parent set from `most preferred' to `least preferred'. By constructing such an ordered set, and by appending the set with siblings, the node is able to construct an ordered list of preferred next hops to assist in local and temporary routing decisions. The use of the ordered list by a forwarding engine is loosely constrained, and may take into account the dynamics of the LLN. Further, a forwarding engine implementation may decide to perform load balancing functions using hash-based mechanisms to avoid packet re-ordering. Note however, that specific details of a

forwarding engine implementation are beyond the scope of this document.

These decisions may be local and/or temporary with the objective to maintain the DAG shape while preserving routing stability.

3.5. Maintenance of Routing Adjacency

In order to relieve the LLN of the overhead of periodic keepalives, RPL may employ an as-needed mechanism of NS/NA in order to verify routing adjacencies just prior to forwarding data. Pending the outcome of verifying the routing adjacency, the packet may either be forwarded or an alternate next-hop may be selected.

4. Constraint Based Routing in LLNs

This aim of this section is to make a clear distinction between routing metrics and constraints and define the term constraint based routing as used in this document.

4.1. Routing Metrics

Routing metrics are used by the routing protocol to compute the shortest path according to one of more defined metrics. IGPs such as IS-IS ([RFC5120]) and OSPF ([RFC4915]) compute the shortest path according to a Link State Data Base (LSDB) using link metrics configured by the network administrator. Such metrics can represent the link bandwidth (in which case the metric is usually inversely proportional to the bandwidth), delay, etc. Note that in some cases the metric is a polynomial function of several metrics defining different link characteristics. The resulting shortest path cost is equal to the sum (or multiplication) of the link metrics along the path: such metrics are said to be additive or multiplicative 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 ECMP (Equal Cost Multiple Paths). The optimization of multiple metrics is known as an NP complete problem and is sometimes supported by some centralized path computation engine.

In the case of RPL, it is virtually impossible to define *the* metric, or even a composite, that will fit it all:

- Some information apply when determining routes, other information may apply only when forwarding packets along provisioned routes.

- Some values are aggregated hop-by-hop, others are triggers from L2.

- Some properties are very stable, others vary rapidly.

- Some data are useful in a given scenario and useless in another.

- Some arguments are scalar, others statistical.

For that reason, the RPL protocol core is agnostic to the logic that handles metrics. A node will be configured with some external logic to use and prioritize certain metrics for a specific scenario. As new heterogeneous devices are installed to support the evolution of a network, or as networks form in a totally ad-hoc fashion, it will happen that nodes that are programmed with antagonistic logics and conflicting or orthogonal priorities end up participating in the same network. It is thus recommended to use consistent parent selection policy, as per Objective Code Points (OCP), to ensure consistent optimized paths.

RPL is designed to survive and still operate, though in a somewhat degraded fashion, when confronted to such heterogeneity. The key design point is that each node is solely responsible for setting the vector of metrics that it sources in the DAG, derived in part from the metrics sourced from its preferred parent. As a result, the DAG is not broken if another node makes its decisions in as antagonistic fashion, though an end-to-end path might not fully achieve any of the optimizations that nodes along the way expect. The default operation specified in OCP 0 clarifies this point.

4.2. Routing Constraints

A constraint is a link or a node characteristic that must be satisfied by the computed path (using boolean values or lower/upper bounds) and is by definition neither additive nor multiplicative. Examples of links constraints are "available bandwidth", "administrative values (e.g. link coloring)", "protected versus non-protected links", "link quality" whereas a node constraint can be the level of battery power, CPU processing power, etc.

4.3. Constraint Based Routing

The notion of constraint based routing consists of finding the shortest path according to some metrics satisfying a set of constraints. A technique consists of first filtering out all links and nodes that cannot satisfy the constraints (resulting in a sub-topology) and then computing the shortest path.


```
      Example 1:
         Link Metric:     Bandwidth
         Link Constraint: Blue
         Node Constraint: Mains-powered node

      Objective function 1:
         "Find the shortest path (path with lowest cost where the path
         cost is the sum of all link costs (Bandwidth)) along the path
         such that all links are colored `Blue' and that only traverses
         Mains-powered nodes."



      Example 2:
         Link Metric:     Delay
         Link Constraint: Bandwidth

      Objective function 2:
         "Find the shortest path (path with lowest cost where the path
         cost is the sum of all link costs (Delay)) along the path such
         that all links provide at least X Bit/s of reservable
```

bandwidth."

5. RPL Protocol Specification

5.1. DAG Information Option

The DAG Information Option carries a number of metrics and other information that allows a node to discover a DAG, select its DAG parents, and identify its siblings while employing loop avoidance strategies.

5.1.1. DAG Information Option (DIO) base option

The DAG Information Option is a container option carried within an IPv6 Router Advertisement message as defined in [RFC4861], which might contain a number of suboptions. The base option regroups the minimum information set that is mandatory in all cases.


```
     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      |    Length     |G|D|A|  00000  |   Sequence    |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    | DAGPreference |               BootTimeRandom                  |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |   NodePref.   |    DAGRank    |           DAGDelay            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    | DIOIntDoubl.  | DIOIntMin.    |     DAGObjectiveCodePoint     |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                          PathDigest                           |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
```

```
+                                                               +
|                            DAGID                              |
+                                                               +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   sub-option(s)...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: DIO Base Option

Type: 8-bit unsigned identifying the DIO base option. The suggested value is 140 to be confirmed by the IANA.

Length: 8-bit unsigned integer set to 4 when there is no suboption. The length of the option (including the type and length fields and the suboptions) in units of 8 octets.

Flag Field: Three flags are currently defined:

Grounded (G): The Grounded (G) flag is set when the DAG root is offering connectivity to an external routed infrastructure such as the Internet.

Destination Advertisement Trigger (D): The Destination Advertisement Trigger (D) flag is set when the DAG root or another node in the successor chain decides to trigger the sending of destination advertisements in order to update routing state for the outward direction along the DAG, as further detailed in Section 5.9. Note that the use and semantics of this flag are still under investigation.

Destination Advertisement Supported (A) : The Destination Supported (A) bit is set when the DAG root is capable to support the collection of destination advertisement related routing state and enables the operation of the destination advertisement mechanism within the DAG.

Unassigned bits of the Flag Field are considered as reserved.

They MUST be set to zero on transmission and MUST be ignored on receipt.

Sequence Number: 8-bit unsigned integer set by the DAG root, incremented according to a policy provisioned at the DAG root, and propagated with no change outwards along the DAG. Each increment SHOULD have a value of 1 and may cause a wrap back to zero.

DAGPreference: 8-bit unsigned integer set by the DAG root to its preference and unchanged at propagation. DAGPreference ranges from 0x00 (least preferred) to 0xFF (most preferred). The default is 0 (least preferred). The DAG preference 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 DAG while still meeting other optimization objectives, then the node will seek to join the more preferred DAG.

BootTimeRandom: A random value computed at boot time and recomputed in case of a duplication with another node. The concatenation of the NodePreference and the BootTimeRandom is a 32-bit extended preference that is used to resolve collisions. It is set by each node at propagation time.

NodePreference: The administrative preference of that LLN Node. Default is 0. 255 is the highest possible preference. Set by each LLN Node at propagation time. Forms a collision tiebreaker in combination with BootTimeRandom.

DAGRank: 8-bit unsigned integer indicating the DAG rank of the node sending the RA-DIO message. The DAGRank of the DAG root is typically 1. DAGRank is further described in Section 5.3.

DAGDelay: 16-bit unsigned integer set by the DAG root indicating the delay before changing the DAG configuration, in TBD-units. A default value is TBD. It is expected to be an order of magnitude smaller than the RA-interval. It is also expected to be an order of magnitude longer than the typical propagation delay inside the LLN.

DIOIntervalDoublings: 8-bit unsigned integer. Configured on the DAG root and used to configure the trickle timer governing when RA-DIO message should be sent within the DAG. DIOIntervalDoublings is the number of times that the DIOIntervalMin is allowed to be doubled during the trickle timer operation.

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

DAGObjectiveCodePoint: The DAG Objective Code Point is used to indicate the cost metrics, objective functions, and methods of computation and comparison for DAGRank in use in the DAG. The DAG OCP is set by the DAG root. (Objective Code Points are to be further defined in [I-D.ietf-roll-routing-metrics].

PathDigest: 32-bit unsigned integer CRC, updated by each LLN Node. This is the result of a CRC-32c computation on a bit string obtained by appending the received value and the ordered set of DAG parents at the LLN Node. DAG roots use a 'previous value' of zeroes to initially set the PathDigest. Used to determine when something in the set of successor paths has changed.

DAGID: 128-bit unsigned integer which uniquely identify a DAG. This value is set by the DAG root. The global IPv6 address of the DAG root can be used, however. the DAGID MUST be unique per DAG within the scope of the LLN. In the case where a DAG root is rooting multiple DAGs the DAGID MUST be unique for each DAG rooted at a specific DAG root.

The following values MUST NOT change during the propagation of RA-DIO messages outwards along the DAG: Type, Length, G, DAGPreference, DAGDelay and DAGID. All other fields of the RA-DIO message are updated at each hop of the propagation.

5.1.1.1. DAG Information Option (DIO) Suboptions

In addition to the minimum options presented in the base option, several suboptions are defined for the RA-DIO message:

5.1.1.1.1. Format

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

Figure 2: DIO Suboption Generic Format

Suboption Type: 8-bit identifier of the type of suboption. When processing a RA-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, continue to process the following suboption, correctly handling any remaining options in the message.

Suboption Length: 8-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 RA-DIO message suboptions which are currently defined for use in the DAG Information Option.

Implementations MUST silently ignore any RA-DIO message suboptions options that they do not understand.

RA-DIO message suboptions may have alignment requirements. Following the convention in IPv6, these options 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).

5.1.1.1.2. Pad1

The Pad1 suboption does not have any alignment requirements. Its format is as follows:

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

```
+-+-+-+-+-+-+-+-+
|   Type = 0    |
+-+-+-+-+-+-+-+-+
```

Figure 3: 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 octet of padding in the RA-DIO message to enable suboptions alignment. If more than one octet of padding is required, the PadN option, described next, should be used rather than multiple Pad1 options.

5.1.1.1.3. PadN

The PadN option does not have any alignment requirements. Its format is as follows:

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

Figure 4: Pad N

The PadN option is used to insert two or more octets of padding in the RA-DIO message to enable suboptions alignment. For N (N > 1) octets of padding, the Option Length field contains the value N-2, and the Option Data consists of N-2 zero-valued octets. PadN Option data MUST be ignored by the receiver.

5.1.1.1.4. DAG Metric Container

The DAG Metric Container suboption may be aligned as necessary to support its contents. Its format is as follows:

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

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - - -
|   Type = 2    | Container Len | DAG Metric Data
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+- - - - - - - - -
```

Figure 5: DAG Metric Container

The DAG Metric Container is used to report aggregated path metrics along the DAG. The DAG Metric Container may contain a number of discrete node, link, and aggregate path metrics as chosen by the implementer. The Container Length field contains the length in octets of the DAG Metric Data. The order, content, and coding of the DAG Metric Container data is as specified in

[I-D.ietf-roll-routing-metrics].

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

5.1.1.1.5. Destination Prefix

The Destination Prefix suboption has an alignment requirement of 4n+1. Its 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    |     Length    | Prefix Length |Resvd|Prf|Resvd|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Prefix Lifetime                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Destination Prefix (Variable Length)              |
.                                                               .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: DAG Destination Prefix

The Destination Prefix suboption is used when the DAG root, or another node located inwards along the DAG on the path to the DAG root, needs to indicate that it offers connectivity to destination prefixes other than the default. 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 DAG, a node MAY decide to join multiple DAGs in support of a particular application.

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

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

value of all one bits (0xFFFFFFFF) represents infinity. A value of all zero bits (0x00000000) indicates a loss of reachability.

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 RA-DIO message may need to specify connectivity to more than one destination, the Destination Prefix suboption may be repeated.

5.2. Conceptual Data Structures

The RPL implementation MUST maintain the following conceptual data structures in support of DAG discovery:

- o A set of candidate neighbors

- o For each DAG:

  * A set of candidate DAG parents

* A set of DAG parents (which are a subset of candidate DAG parents and may be implemented as such)

5.2.1. Candidate Neighbors Data Structure

The set of candidate neighbors is to be populated by neighbors who are discovered by the neighbor discovery mechanism and further qualified as statistically stable as per the mechanisms discussed in [I-D.ietf-roll-routing-metrics]. The candidate neighbors, and related metrics, should demonstrate stability/reliability beyond a certain threshold, and it is recommended that a local confidence value be maintained with respect to the neighbor in order to track this. Implementations MAY choose to bound the maximum size of the candidate neighbor set, in which case a local confidence value will assist in ordering neighbors to determine which ones should remain in the candidate neighbor set and which should be evicted.

If Neighbor Unreachability Detection (NUD) determines that a candidate neighbor is no longer reachable, then it shall be removed from the candidate neighbor set. In the case that the candidate neighbor has associated states in the DAG parent set or active DA entries, then the removal of the candidate neighbor shall be coordinated with tearing down these states. All provisioned routes associated with the candidate neighbor should be removed.

5.2.2. Directed Acyclic Graphs (DAGs) Data Structure

A DAG may be uniquely identified by within the LLN by its unique DAGID. When a single device is capable to root multiple DAGs in support of an application need for multiple optimization objectives it is expected to produce a different and unique DAGID for each of the multiple DAGs.

For each DAG that a node is, or may become, a member of, the implementation MUST keep a DAG table with the following entries:

o DAGID

o DAGObjectiveCodePoint

o A set of Destination Prefixes offered inwards along the DAG

o A set of candidate DAG parents

o A timer to govern the sending of RA-DIO messages for the DAG

o DAGSequenceNumber

When a DAG is discovered for which no DAG data structure is instantiated, and the node wants to join (i.e. the neighbor is to become a candidate DAG parent in the Held-Up state), then the DAG data structure is instantiated.

When the candidate DAG parent set is depleted (i.e. the last candidate DAG parent has timed out of the Held-Down state), then the DAG data structure SHOULD be suppressed after the expiration of an implementation-specific local timer. An implementation SHOULD delay before deallocating the DAG data structure in order to observe that the DAGSequenceNumber has incremented should any new candidate DAG parents appear for the DAG.

5.2.2.1. Candidate DAG Parents Structure

When the DAG is self-rooted, the set of candidate DAG parents is empty.

In all other cases, for each candidate DAG parent in the set, the implementation MUST keep a record of:

o a reference to the neighboring device which is the DAG parent

o a record of most recent information taken from the DAG Information Object last processed from the candidate DAG parent

o a state associated with the role of the candidate as a potential DAG parent {Current, Held-Up, Held-Down, Collision}, further described in Section 5.7

o A DAG Hop Timer, if instantiated

o A Held-Down Timer, if instantiated

5.2.2.1.1. DAG Parents

Note that the subset of candidate DAG parents in the `Current' state comprises the set of DAG parents, i.e. the nodes actively acting as parents in the DAG.

DAG parents may be ordered, according to the OCP. When ordering DAG parents, in consultation with the OCP, the most preferred DAG parent may be identified. All current DAG parents must have a rank less than or equal to that of the most preferred DAG parent.

When nodes are added to or removed from the DAG parent set the most preferred DAG parent may have changed and should be reevaluated. Any nodes having a rank greater than self after such a change must be placed in the Held-Down state and evicted as per the procedures described in Section 5.7

An implementation may choose to keep these records as an extension of the Default Router List (DRL).

5.3. DAG Discovery and Maintenance

DAG discovery locates the nearest sink, as determined according to some metrics and constraints, and forms a Directed Acyclic Graph towards that sink, by identifying a set of DAG parents. During this process DAG discovery also identifies siblings, which may be used later to provide additional path diversity towards the DAG root. DAG discovery enables nodes to implement different policies for selecting their DAG parents in the DAG by using implementation specific policy functions. DAG discovery specifies a set of rules to be followed by all implementations in order to ensure interoperation. DAG discovery also standardizes the format that is used to advertise the most common information that is used in order to select DAG parents.

One of these information, the DAG rank, is used by DAG discovery to provide loop avoidance even if nodes implement different policies. The DAG Rank is computed as specified by the Objective Code Point in use by the DAG, demonstrating the properties described in Section 3.2.1.7. The rank should be computed in such a way so as to provide a comparable basis with other nodes which may not use the same metric at all.

The DAG discovery procedures take into account a number of factors, including:

- o RPL rules for loop avoidance based on rank

- o The OCP function

- o The advertised metrics

- o Local policy functions (e.g. a bounded number of candidate neighbors).

5.3.1. DAG Discovery Rules

In order to organize and maintain loopless structure, the DAG discovery implementation in the nodes MUST obey to the following rules and definitions:

1. A node that does not have any DAG parents in a DAG is the root of its own floating DAG. It's rank is 1. A node will end up in that situation when it looses all of its current feasible parents, i.e. the set of DAG parents becomes depleted. In that case, the node SHOULD remember the DAGID and the sequence counter of the last RA-DIO message from the lost parents for a period of time which covers multiple RA-DIO messages. This is done so that if the node does encounter another possible attachment point to the lost DAGID within a period of time, the node may observe a sequence counter change by comparing the observed sequence counter to the last observed sequence counter and thus verify that the new attachment point is a viable and independent alternative to attach back to the lost DAGID.

2. A node that is attached to an infrastructure that does not support RA-DIO messages, is the DAG root of its own grounded DAG. It's rank is 1. (For example an LBR that is in communication with a non-LLN router not running RPL).

3. A (non-LLN) router sending a RA messages without DIO is considered a grounded infrastructure at rank 0. (For example, a router that is in communication with an LLN node but not running RPL such as a non-LLN public Internet router in communication with an LBR)

4. The DAG root exposes the DAG in the RA-DIO message and nodes propagate the RA-DIO message outwards along the DAG with the RAs that they forward over their LLN links.

5. A node MAY move at any time, with no delay, within its DAG when the move does not cause the node to increase its own DAG rank, as per the rank calculation indicated by the OCP.

6. A node MUST NOT move outwards along a DAG that it is attached to, causing the DAG rank to increase, except in a special case where the node MAY choose to follow the last DAG parent in the set of DAG parents. In the general case, if a node is required to move such that it cannot stay within the DAG without a rank increase, then it needs to first leave the DAG. In other words a node that is already part of a DAG MAY move or follow a DAG parent at any time and with no delay in order to be closer, or stay as close, to the DAG root of its current DAG as it already is, but may not move outwards. RAs received from other routers located at lesser rank in the same DAG may be considered as coming from candidate parents. RAs received from other routers located at the same rank in the same DAG may be considered as coming from siblings. Nodes MUST ignore RAs that are received from other routers located at greater rank within the same DAG.

7. A node may jump from its current DAG into any different DAG if it is preferred for reasons of connectivity, configured preference, free medium time, size, security, bandwidth, DAG rank, or whatever metrics the LLN cares to use. A node may jump at any time and to whatever rank it reaches in the new DAG, but it may have to wait for a DAG Hop timer to elapse in order to do so. This allows the new higher parts (closer to the sink) of the DAG to move first, thus allowing stepped DAG reconfigurations and limiting relative movements. A node SHOULD NOT join a previous DAG (identified by its DAGID) unless the sequence number in the RA-DIO message has incremented since the node left that DAG. A newer sequence number indicates that the candidate parents were not attached behind this node, as they kept getting subsequent RA-DIO messages with new sequence numbers from the same DAG. In the event that old sequence numbers (two or more behind the present value) are encountered they are considered stale and the corresponding parent SHOULD be removed from the set.

8. If a node has selected a new set of DAG parents but has not moved yet (because it is waiting for DAG Hop timer to elapse), the node is unstable MUST NOT send RA-DIOs for that DAG.

9. If a node receives a RA-DIO from one of its DAG parents, and if the parent contains a different DAGID, indicating that the parent has left the DAG, and if the node can remain in the

current DAG through an alternate DAG parent, then the node SHOULD remove the DAG parent which has joined the new DAG from

its DAG parent set and remain in the original DAG. If there is no alternate parent for the DAG, then the node SHOULD follow that parent into the new DAG.

10. When a node detects or causes a DAG inconsistency, as described in Section 5.3.4.2, then the node SHOULD send an unsolicited RA-DIO message to its one-hop neighbors. The RA-DIO is updated to propagate the new DAG information. Such an event MUST also cause the trickle timer governing the periodic sending of RA-DIO messages to be reset.

11. If a DAG parent increases its rank such that the node rank would have to change, and if the node does not wish to follow (e.g. it has alternate options), then the DAG parent SHOULD be evicted from the DAG parent set. If the DAG parent is the last in the DAG parent set, then the node SHOULD chose to follow it.

5.3.2. Reception and Processing of RA-DIO messages

When an RA-DIO message is received from a source device named SRC, the receiving node must first determine whether or not the RA-DIO message should be accepted for further processing, and subsequently present the RA-DIO message for further processing if eligible.

5.3.2.1. Determination of Eligibility for DIO Processing

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

If SRC is not a member of the candidate neighbor set, then the RA-DIO is not eligible for further processing. (Further evaluation/ confidence of this neighbor is necessary)

If the RA-DIO message advertises a DAG that the node is already a member of, then:

If the rank of SRC as reported in the RA-DIO message is lesser than that of the node within the DAG, then the RA-DIO message MUST be considered for further processing

If the rank of SRC as reported in the RA-DIO message is equal to that of the node within the DAG, then SRC is marked as a sibling and the RA-DIO message is not eligible for further processing.

If the rank of SRC as reported in the RA-DIO message is higher than that of the node within the DAG, and SRC is not a DAG parent, then the RA-DIO message MUST NOT be considered for further processing

If SRC is a DAG parent for any other DAG that the node is attached to, then the RA-DIO message MUST be considered for further processing (the DAG parent may have jumped).

If the RA-DIO message advertises a DAG that offers a better (new or alternate) solution to an optimization objective desired by the node, then the RA-DIO message MUST be considered for further processing.

5.3.2.2. Overview of RA-DIO Message Processing

If the received RA-DIO message is for a new/alternate DAG:

Instantiate a data structure for the new/alternate DAG if necessary

Place the neighbor in the candidate DAG parent set

If the node has sent an RA message within the risk window as described in Section 5.7.3 then perform the collision detection described in Section 5.7.3. If a collision occurs, place the candidate DAG parent in the collision state and do not process the RA-DIO message any further as described in Section 5.7.

If the SRC node is also a DAG parent for another DAG that the node is a member of, and if the new/alternate DAG satisfies an equivalent optimization objective as the other DAG, then the DAG parent is known to have jumped.

Remove SRC as a DAG parent from the other DAG (place it in the held-down state)

If the other DAG is now empty of candidate parents, then directly follow SRC into the new DAG by adding it as a DAG parent in the Current state, else ignore the RA-DIO message (do not follow the parent).

If the new/alternate DAG offers a better solution to the optimization objectives, then prepare to jump: copy the DIO information into the record for the candidate DAG parent, place the candidate DAG parent into the Held-Up state, and start the

DAG Hop timer as per Section 5.7.1.

If the RA-DIO message is for a known/existing DAG:

Process the RA-DIO message as per the rules in Section 5.3

As candidate parents are identified, they may subsequently be promoted to DAG parents by following the rules of DAG discovery as described in Section 5.3. When a node adds another node to its set of candidate parents, the node becomes attached to the DAG through the parent node.

In the DAG discovery implementation, the most preferred parent should be used to restrict which other nodes may become DAG parents. Some nodes in the DAG parent set may be of a rank less than or equal to the most preferred DAG 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).

5.3.3. RA-DIO Transmission

Each node maintains a timer that governs when to multicast RA messages. This timer is implemented as a trickle timer operating over a variable interval. Trickle timers are further detailed in

Section 5.3.4. The governing parameters for the timer should be configured consistently across the DAG, and are provided by the DAG root in the RA-DIO message. In addition to periodic RA messages, each LLN node will respond to Router Solicitation (RS) messages according to [RFC4861].

- When a node is unstable, because any DAG Hop timer is running in preparation for a jump, then the node MUST NOT transmit unsolicited RA-DIOs (i.e. the node will remain silent when the timer expires).

- When a node detects an inconsistency, it SHOULD reset the interval of the trickle timer to a minimum value, causing RA messages to be emitted more frequently as part of a strategy to quickly correct the inconsistency. Such inconsistencies may be, for example, an update to a key parameter (e.g. sequence number) in the RA-DIO message or a loop detected when a node located inwards along the DAG forwards traffic outwards. Inconsistencies are further detailed in Section 5.3.4.2.

- When a node enters a mode of consistent operation within a DAG, i.e. RA-DIO messages from its DAG parents are consistent and no other inconsistencies are detected, it may begin to open up the

interval of the trickle timer towards a maximum value, causing RAs to be emitted less frequently, thus reducing network maintenance overhead and saving energy consumption (which is of utmost importance for battery-operated nodes).

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

Note that if multiple DAG roots are participating in the same DAG, i.e. offering RA-DIO messages with the same DAGID, then they must coordinate with each other to ensure that their RA-DIO messages are consistent when they emit RA-DIO messages. In particular the Sequence number must be identical from each DAG root, regardless of which of the multiple DAG roots issues the RA-DIO message, and changes to the Sequence number should be issued at the same time. The specific mechanism of this coordination, e.g. along a non-LLN network between DAG roots, is beyond the scope of this specification.

5.3.4. Trickle Timer for RA Transmission

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

For each DAG that a node is part of, 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 RA-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 RA-DIO message as DIOIntervalDoublings. The default value is DEFAULT_DIO_INTERVAL_DOUBLINGS.

5.3.4.1. Resetting the Trickle Timer

The trickle timer for a DAGID is reset by:

1. Setting I_min and I_doublings to the values learned from the RA-

DIO message.

2. Setting C to zero.

3. Setting I to I_min.

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

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

When node learns about a DAG through a RA-DIO message and makes the decision to join it, it initializes the state of the trickle timer by resetting the trickle timer and listening. Each time it hears a consistent RA for this DAG from a DAG parent, it MAY increment C.

When the timer fires at time T, the node compares C to the redundancy constant, DEFAULT_DIO_REDUNDANCY_CONSTANT. If C is less than that value, the node generates a new RA and broadcasts 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.

5.3.4.2. Determination of Inconsistency

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

- The node joins a new DAGID
- The node moves within a DAGID
- The node receives a modified RA-DIO message from a DAG parent
- A DAG parent forwards a packet intended to move inwards, indicating an inconsistency and possible loop.

- A metric communicated in the RA-DIO message is determined to be inconsistent, as according to a implementation specific path metric selection engine.

o The rank of a DAG parent has changed.

5.4. DAG Heartbeat

The DAG root makes the sole determination of when to revise the DAGSequenceNumber by incrementing it upwards. When the DAGSequenceNumber is increased an inconsistency results, causing RA-DIO messages to be sent back outwards along the DAG to convey the change. The degree to which this mechanism is relied on may be determined by the implementation- on one hand it may serve as a periodic heartbeat, refreshing the DAG states, and on the other hand it may result in a constant steady-state control cost overhead which is not desirable.

Some implementations may provide an administrative interface, such as a command line, at the DAG root whereby the DAGSequenceNumber may be caused to increment in response to some policy outside of the scope of RPL.

Other implementations may make use of a periodic timer to automatically increment the DAGSequenceNumber, resulting in a periodic DAG Heartbeat at a rate appropriate to the application and implementation.

5.5. DAG Selection

The DAG selection is implementation and algorithm dependent. Nodes SHOULD prefer to join DAGs 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 candidate parent whose availability is detected as fluctuating, at least when more stable choices are available. Nodes MAY place the failed candidate parent in a Hold Down mode that ensures that the candidate parent will not be reused for a given period of time.

When connection to a fixed network is not possible or preferable for security or other reasons, scattered DAGs MAY aggregate as much as possible into larger DAGs 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 DAG parent.

5.6. Administrative rank

When the DAG is formed under a common administration, or when a node performs a certain role within a community, it might be beneficial to associate a range of acceptable rank with that node. For instance, 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 OCP in order to expose an exaggerated rank.

5.7. Candidate DAG Parent States and Stability

Candidate DAG parents may or may not be eligible to act as DAG parents depending on runtime conditions. The following states are defined:

Current — This candidate parent is in the set of DAG parents and may be used for forwarding traffic inward along the DAG. When a candidate parent is placed into the Current state, or taken out of the Current state, it is necessary to re-evaluate which of the remaining DAG parents is the most preferred DAG parent and its rank. At that time any remaining DAG parents of greater rank than this node must be placed in the Held-Down state, and the hold-down timer started, in order to be evicted as DAG parents. In the same fashion, siblings must also be reevaluated.

Held-Up — This parent can not be used until the DAG hop timer elapses.

Held-Down — This candidate parent can not be used till hold down timer elapses. At the end of the hold-down period, the candidate is removed from the candidate DAG parent set, and may be reinserted if it appears again with a RA-DIO message.

Collision — This candidate parent can not be used till its next RA-DIO message.

5.7.1. Held-Up

This state is managed by the DAG Hop timer, it serves 2 purposes:

Delay the reattachment of a sub-DAG that has been forced to detach. This is not as safe as the use of the sequence, but still covers that when a sub-DAG has detached, the RA-DIO message that is initiated by the new DAG root has a chance to spread outward

along the sub-DAG, ideally forming a frozen sub-DAG that is aware of the DAG change, such that two different DAGs have formed prior to an attempted reattachment.

Limit RA-DIO message storms (control cost / churn) when two DAGs collide/merge. The idea is that between the nodes from DAG A that decide to move to DAG B, those that see the highest place (closer to the DAG root) in DAG B will move first and advertise their new locations before other nodes from DAG A actually move.

A new DAG is discovered upon receiving a RA message with or without a DIO. The node joins the DAG by selecting the source of the RA message as a DAG parent (and possibly installing the DAG parent as a default gateway). The node is then a member of the DAG and may begin to multicast RA-DIO messages containing the DIO for the DAG.

When a new DAG is discovered, the candidate parent that advertises the new DAG is placed in a held up state for the duration of a DAG Hop timer. If the resulting new set of DAG parents is more preferable than the current one, or if the node is intending to maintain a membership in the new DAG in addition to its current DAG, the node expects to jump and becomes unstable.

A node that is unstable may discover other candidate parents from the same new DAG during the instability phase. It needs to start a new DAG Hop timer for all these. The first timer that elapses for a given new DAG clears them all for that DAG, allowing the node to jump to the highest position available in the new DAG.

The duration of the DAG Hop timer depends on the DAG Delay of the new DAG and on the rank of candidate parent that triggers it: (candidates rank + random) * candidate's DAG_delay (where 0 <= random < 1). It is randomized in order to limit collisions and synchronizations.

5.7.2. Held-Down

When a neighboring node is 'removed' from the Default Router List, it is actually held down for a hold down timer period, in order to prevent flapping. This happens when a node disappears (upon expiration timer).

When the hold down timer elapses, the node is removed from the candidate DAG parent set.

5.7.3. Collision

A race condition occurs if 2 nodes send RA-DIO messages at the same time and then attempt to join each other. This might happen, for

example, between nodes which act as DAG root of their own DAGs. In order to detect the situation, LLN Nodes time stamp the sending of RA-DIO message. Any RA-DIO message received within a short link-layer-dependent period introduces a risk. To resolve the collision, a 32bits extended preference is constructed from the RA-DIO message by concatenating the NodePreference with the BootTimeRandom.

A node that decides to add a candidate to its DAG parents will do so between (candidate rank) and (candidate rank + 1) times the candidate DAG Delay. But since a node is unstable as soon as it receives the RA-DIO message from the desired candidate, it will restrain from sending a RA-DIO message between the time it receives the RA and the time it actually jumps. So the crossing of RA may only happen during the propagation time between the candidate and the node, plus some internal queuing and processing time within each machine. It is expected that one DAG delay normally covers that interval, but ultimately it is up to the implementation and the configuration of the candidate parent to define the duration of risk window.

There is risk of a collision when a node receives an RA, for another candidate that is more preferable than the current candidate, within the risk window. In the face of a potential collision, the node with lowest extended preference processes the RA-DIO message normally, while the router with the highest extended preference places the other in collision state, does not start the DAG hop timer, and does not become instable. It is expected that next RAs between the two will not cross anyway.

For example, consider a case where two nodes are each rooting their own transient floating DAGs and multicast RA-DIO messages towards each other in a close enough interval that the RA-DIO messages `cross'. Then each node may receive the RA-DIO message from the other node, and in some scenario decide to join each others DAG. RPL avoids this deadlock scenario via the collision mechanism described above - after each node sends the RA-DIO message they will enter the risk window. When the peer RA-DIO message is received in the risk window, the nodes will calculate the extended preferences as describe above and the node with the lowest extended preference will proceed to process the RA-DIO message, while the other node will defer, avoiding the deadlock scenario.

5.7.4. Instability

A node is instable when it is prepared to shortly replace a set of DAG parents in order to jump to a different DAGID. This happens typically when the node has selected a more preferred candidate parent in a different DAG and has to wait for the DAG hop timer to elapse before adjusting the DAG parent set. Instability may also

occur when the entire current DAG parent set is lost and the next best candidates are still held up. Instability is resolved when the DAG hop timer of all the candidate(s) causing instability elapse. Such candidates then change state to Current or Held- Down.

Instability is transient (in the order of DAG hop timers). When a node is unstable, it MUST NOT send RAs with the DIO message. This avoids loops when node A decides to attach to node B and node B decides to attach to node A. Unless RAs cross (see Collision section), a node receives RA-DIO messages from stable candidate parents, which do not plan to attach to the node, so the node can safely attach to them.

5.8. Guidelines for Objective Code Points

5.8.1. Objective Function

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

The Objective Function is specified in the RA-DIO message using an objective code point (OCP) and indicates the objective function that has been used to compute the DAG (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]. This document specifies the OCP 0, in support of default operation.

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 RA-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 The OF scans all the candidate neighbors on the possible interfaces to check whether they can act as an attachment router for a DAG. 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 The OF computes self's rank by adding the step of rank to that candidate to the rank of that candidate. The step of rank is estimated as follows:

  * The step of rank might vary from 1 to 16.

    + 1 indicates a unusually good link, for instance a link between powered devices in a mostly battery operated environment.

+ 4 indicates a `normal'/typical link, as qualified by the implementation.

+ 16 indicates a link that can hardly be used to forward any packet, for instance a radio link with quality indicator or expected transmission count that is close to the acceptable threshold.

* Candidate neighbors that would cause self's rank to increase are ignored

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 DAG are ignored

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

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

5.8.2. Objective Code Point 0 (OCP 0)

Here follows the specification for the default Objective Function corresponding to OCP codepoint 0. This is a very simple reference to help design more complex Objective Functions. In particular, the Objective Function described here does not use physical metrics as described in [I-D.ietf-roll-routing-metrics], but are only based on abstract information from the RA-DIO message such as rank and administrative preference.

This document specifies a default objective metric, called OF0, and using the OCP 0. OF0 is the default objective function of RPL, and can be used if allowed by the policy of the processing node when no objective function is included in the RA-DIO message, or if the OF indicated in the RA-DIO message is unknown to the node. If not allowed, then the RA-DIO message is simply ignored and not processed by the node.

5.8.2.1. OCP 0 Objective Function (OF0)

OF0 favors the connectivity. That is, the Objective Function is designed to find the nearest sink into a 'grounded' topology, and if there is none then join any network per order of administrative preference. The metric in use is the rank.

OF0 selects a preferred parent and a backup next_hop if one is available. The backup next_hop might be a parent or a sibling. All the traffic is routed via the preferred parent. When the link conditions do not let a packet through to the preferred parent, the packet is passed to the backup next_hop.

The step of rank is 4 for each hop.

5.8.2.2. Selection of the Preferred Parent

As it scans all the candidate neighbors, OF0 keeps the parent that is the best for the following criteria (in order):

1. The interface must be usable and the administrative preference (if any) applies first.

2. A candidate that would cause the node to augment the rank in the current DAG is not considered.

3. A router that has been validated as usable, e.g. with a local confidence that has exceeded some pre-configured threshold, is better.

4. If none are grounded then a DAG with a more preferred administrative preference is better.

5. A router that offers connectivity to a grounded DAG is better.

6. A lesser resulting rank is better.

7. A DAG for which there is an alternate parent is better. This check is optional. It is performed by computing the backup next_hop while assuming that this router won.

8. The DAG that was in use already is preferred.

9. The router with a better router preference wins.

10. The preferred parent that was in use already is better.

11. A router that has announced a RA-DIO message more recently is preferred.

5.8.2.3. Selection of the Backup next_hop

- The interface must be usable and the administrative preference (if any) applies first.

- The preferred parent is ignored.

- Candidate neighbors that are not in the same DAG are ignored.

- Candidate neighbors with a higher rank are ignored.

- Candidate neighbors of a better rank than self (non-siblings) are preferred.

o A router that has been validated as usable, e.g. with a local confidence that has exceeded some pre-configured threshold, is better.

o The router with a better router preference wins.

o The backup next_hop that was in use already is better.

5.9. Establishing Routing State Outward Along the DAG

The destination advertisement mechanism supports the dissemination of routing state required to support traffic flows outward along the DAG, from the DAG root toward nodes.

As a result of destination advertisement operation:

o DAG discovery establishes a DAG oriented toward a DAG root using extended Neighbor Discovery RS/RA flows, along which inward routes toward the DAG root are set up.

o Destination advertisement extends Neighbor Discovery in order to establish outward routes along the DAG. Such paths consist of:
   * Hop-By-Hop routing state within islands of `stateful' nodes.
   * Source Routing `bridges' across nodes who do not retain state.

Destinations disseminated with the destination advertisement mechanism may be prefixes, individual hosts, or multicast listeners. The mechanism supports nodes of varying capabilities as follows:

o When nodes are capable of storing routing state, they may inspect destination advertisements and learn hop-by-hop routing state toward destinations by populating their routing tables with the routes learned from nodes in their sub-DAG. In this process they may also learn necessary piecewise source routes to traverse regions of the LLN that do not maintain routing state. They may perform route aggregation on known destinations before emitting Destination Advertisements.

o When nodes are incapable of storing routing state, they may forward destination advertisements, recording the reverse route as the go in order to support the construction of piecewise source routes.

Nodes that are capable of storing routing state, and finally the DAG roots, are able to learn which destinations are contained in the sub-DAG below the node, and via which next-hop neighbors. The dissemination and installation of this routing state into nodes allows for Hop-By-Hop routing from the DAG root outwards along the

DAG. The mechanism is further enhance by supporting the construction of source routes across stateless `gaps' in the DAG, where nodes are incapable of storing additional routing state. An adaptation of this mechanism allows for the implementation of loose-source routing.

A special case, the reception of a destination advertisement addressed to a link-local multicast address, allows for a node to learn destinations directly available from its one-hop neighbors.

A design choice behind advertising routes via destination advertisements is not to synchronize the parent and children databases along the DAG, but instead to update them regularly to recover from the loss of packets. The rationale for that choice is time variations in connectivity across unreliable links. If the topology can be expected to change frequently, synchronization might be an excessive goal in terms of exchanges and protocol complexity. The approach used here results in a simple protocol with no real peering. The destination advertisement mechanism hence provides for periodic updates of the routing state, as cued by occasional RAs and other mechanisms, similarly to other protocols such as RIP [RFC2453].

5.9.1. Destination Advertisement Message Formats

5.9.1.1. DAO Option

RPL extends Neighbor Discovery [RFC4861] and RFC4191 [RFC4191] to allow a node to include a destination advertisement option, which includes prefix information, in the Neighbor Advertisement (NA) messages. A prefix option is normally present in RA messages only, but the NA is augmented with this option in order to propagate destination information inwards along the DAG. The option is named the Destination Advertisement Option (DAO), and an NA message containing this option may be referred to as a destination advertisement, or NA-DAO. The RPL use of destination advertisements allows the nodes in the DAG to build up routing state for nodes contained in the sub-DAG in support of traffic flowing outward along the DAG.

```
 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      |    Length     | Prefix Length |    RRCount    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          DAO Lifetime                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Route Tag                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   DAO Depth   |   Reserved    |         DAO Sequence          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Prefix (Variable Length)                    |
.                                                               .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Reverse Route Stack (Variable Length)            |
.                                                               .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7: The Destination Advertisement Option (DAO)

Type: 8-bit unsigned identifying the Destination Advertisement option. IANA had defined the IPv6 Neighbor Discovery Option Formats registry. The suggested type value for the Destination Advertisement Option carried within a NA message is 141, to be confirmed by IANA.

Length: 8-bit unsigned integer. The length of the option (including the Type and Length fields) in units of 8 octets.

Prefix Length: 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.

Route Tag: 32-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.

DAO Depth: Set to 0 by the node that owns the prefix and first issues the NA-DAO message. Incremented by all LLN nodes that propagate the NA-DAO message.

Reserved: 8-bit unused field. The reserved field MUST be set to zero on transmission and MUST be ignored on receipt.

DAO Sequence: Incremented by the node that owns the prefix for each new NA-DAO message for that prefix.

Prefix: Variable-length field containing an IPv6 address or a prefix of an IPv6 address. 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 who adds on to the Reverse Route Stack will append to the list and increment the RRCount.

5.9.2. Destination Advertisement Operation

5.9.2.1. Overview

According to implementation specific policy, a subset or all of the feasible parents in the DAG may be selected to receive prefix information from the destination advertisement mechanism. This subset of DAG parents shall be designated the set of DA parents.

As NA-DAO messages for particular destinations move inwards along the DAG, a sequence counter is used to guarantee their freshness. The sequence counter is incremented by the source of the NA-DAO message (the node that owns the prefix, or learned the prefix via some other means), each time it issues a NA-DAO message for its prefix. Nodes who receive the NA-DAO message and, if scope allows, will be forwarding a NA-DAO message for the unmodified destination inwards along the DAG, will leave the sequence number unchanged. Intermediate nodes will check the sequence counter before processing a NA-DAO message, and if the DAO is unchanged (the sequence counter has not changed), then the NA-DAO message will be discarded without additional processing. Further, if the NA-DAO message appears to be out of synch (the sequence counter is 2 or more behind the present value) then the DAO state is considered to be stale and may be purged, and the NA-DAO message is discarded. A depth is also added for tracking purposes; the depth is incremented at each hop as the NA-DAO message is propagated up the DAG. Nodes who are storing routing state may use the depth to determine which possible next-hops

for the destination are more optimal.

If destination advertisements are activated in the RA-DIO message as indicated by the `D' bit, the node sends unicast destination advertisements to its DA parents, and only accepts unicast destination advertisements from any nodes but those contained in the DA parent subset.

Every NA to a DA parent MAY contain one or more DAOs. Receiving a RA-DIO message with the `D' destination advertisement bit set from a DAG parent stimulates the sending of a delayed destination advertisement back, with the collection of all known prefixes (that is the prefixes learned via destination advertisements for nodes lower in the DAG, and any connected prefixes). If the Destination Advertisement Supported (A) bit is set in the RA-DIO message for the DAG, then a destination advertisement is also sent to a DAG parent once it has been added to the DA parent set after a movement, or when the list of advertised prefixes has changed. Destination advertisements may also be scheduled for sending when the PathDigest of the RA-DIO message has changed, indicating that some aspect of the inwards paths along the DAG has been modified.

Destination advertisements may advertise positive (prefix is present) or negative (removed) NA-DAO messages, termed as no-DAOs. A no-DAO is stimulated by the disappearance of a prefix below. This is discovered by timing out after a request (a RA-DIO message) or by receiving a no-DAO. A no-DAO is a conveyed as a NA-DAO message with a DAO Lifetime of 0.

A node who is capable of recording the state information conveyed in a unicast NA-DAO message will do so upon receiving and processing the NA-DAO message, thus building up routing state concerning destinations below it in the DAG. If a node capable of recording state information receives a NA-DAO message containing a Reverse Route Stack, then the node knows that the NA-DAO message has traversed one or more nodes that did not retain any routing state as it traversed the path from the DAO source to the node. The node may then extract the Reverse Route Stack and retain the included state in order to specify Source Routing instructions along the return path towards the destination. The node MUST set the RRCount back to zero and clear the Reverse Route Stack prior to passing the NA-DAO message information on.

A node who is unable to record the state information conveyed in the NA-DAO message will append the next-hop address to the Reverse Route Stack, increment the RRCount, and then pass the destination advertisement on without recording any additional state. In this way the Reverse Route Stack will contain a vector of next hops that must

be traversed along the reverse path that the NA-DAO message has traveled. The vector will be ordered such that the node closest to the destination will appear first in the list. In such cases, if it is useful to the implementation to try and build up redundant paths, the node may choose to convey the destination advertisement to one or more DAG parents in order of preference as guided by an implementation specific policy.

In some cases (called hybrid cases), some nodes along the path a destination advertisement follows inward along the DAG may store state and some may not. The destination advertisement mechanism allows for the provisioning of routing state such that when a packet is traversing outwards along the DAG, some nodes may be able to directly forward to the next hop, and other nodes may be able to specify a piecewise source route in order to bridge spans of stateless nodes within the path on the way to the desired destination.

In the case where no node is able to store any routing state as destination advertisements pass by, and the DAG root ends up with NA-DAO messages that contain a completely specified route back to the originating node in the form of the inverted Reverse Route Stack. A DAG root should not request (Destination Advertisement Trigger) nor indicate support (Destination Advertisement Supported) for destination advertisements if it is not able to store the Reverse Route Stack information in this case.

The destination advertisement mechanism requires stateful nodes to maintain lists of known prefixes. A prefix entry contains the following abstract information:

- A reference to the ND entry that was created for the advertising neighbor.
- The IPv6 address and interface for the advertising neighbor.
- The logical equivalent of the full destination advertisement information (including the prefixes, depth, and Reverse Route Stack, if any).
- A 'reported' Boolean to keep track whether this prefix was reported already, and to which of the DA parents.
- A counter of retries to count how many RA-DIO messages were sent on the interface to the advertising neighbor without reachability confirmation for the prefix.

Note that nodes may receive multiple information from different

neighbors for a specific destination, as different paths through the DAG may be propagating information inwards along the DAG for the same destination. A node who is recording routing state will keep track of the information from each neighbor independently, and when it comes time to propagate the NA-DAO message for a particular prefix to the DA parents, then the DAO information will be selected from among the advertising neighbors who offer the least depth to the destination.

The destination advertisement mechanism stores the prefix entries in one of 3 abstract lists; the Connected, the Reachable and the Unreachable lists.

The Connected list corresponds to the prefixes owned and managed by the local node.

The Reachable list contains prefixes for which the node keeps receiving NA-DAO messages, and for those prefixes which have not yet timed out.

The Unreachable list keeps track of prefixes which are no longer valid and in the process of being deleted, in order to send NA-DAO messages with zero lifetime (also called no-DAO) to the DA parents.

5.9.2.1.1. Destination Advertisement Timers

The destination advertisement mechanism requires 2 timers; the DelayNA timer and the RemoveTimer.

- o The DelayNA timer is armed upon a stimulation to send a destination advertisement (such as a RA-DIO message from a DA parent). When the timer is armed, all entries in the Reachable list as well as all entries for Connected list are set to not be reported yet for that particular DA parent.

- o The DelayNA timer has a duration that is DEF_NA_LATENCY divided by a multiple of the DAG rank of the node. The intention is that nodes located deeper in the DAG should have a shorter DelayNA timer, allowing NA-DAO messages a chance to be reported from deeper in the DAG and potentially aggregated along sub-DAGs before propagating further inwards.

- o The RemoveTimer is used to clean up entries for which NA-DAO messages are no longer being received from the sub-DAG.

  * When a RA-DIO message is sent that is requesting destination advertisements, a flag is set for all DAO entries in the routing table.

* If the flag has already been set for a DAO entry, the retry count is incremented.

* If a NA-DAO message is received to confirm the entry, the entry is refreshed and the flag and count may be cleared.

* If at least one entry has reached a threshold value and the RemoveTimer is not running, the entry is considered to be probably gone and the RemoveTimer is started.

* When the RemoveTimer elapse, NA-DAO messages with lifetime 0, i.e. no-DAOs, are sent to explicitly inform DA parents that the entries who have reached the threshold are no longer available, and the related routing states may be propagated and cleaned up.

o The RemoveTimer has a duration of min (MAX_DESTROY_INTERVAL, RA_INTERVAL).

5.9.2.2. Multicast Destination Advertisement messages

It is also possible for a node to multicast a NA-DAO message to the link-local scope all-nodes multicast address FF02::1. This message will be received by all node listening in range of the emitting node. The objective is to enable direct P2P communication, between destinations directly supported by neighboring nodes, without needing the RPL routing structure to relay the packets.

A multicast NA-DAO message MUST be used only to advertise information about self, i.e. prefixes in the Connected list or addresses owned by this node. This would typically be a multicast group that this node is listening to or a global address owned by this node, though it can be used to advertise any prefix owned by this node as well. A multicast NA-DAO message is not used for routing and does not presume any DAG relationship between the emitter and the receiver; it MUST NOT be used to relay information learned (e.g. information in the Reachable list) from another node; information obtained from a multicast NA-DAO MAY be installed in the routing table and MAY be propagated by a router in unicast NA-DAOs.

A node receiving a multicast NA-DAO message addressed to FF02::1 MAY install prefixes contained in the NA-DAO message in the routing table for local use. Such a node MUST NOT perform any other processing on the NA-DAO message (i.e. such a node does not presume it is a DA parent).

5.9.2.3. Unicast Destination Advertisement messages from child to parent

When sending a destination advertisement to a DA parent, a node includes the DAOs for prefix entries not already reported (since the last DA Trigger from an RA-DIO message) in the Reachable and Connected lists, as well as no-DAOs for all the entries in the Unreachable list. Depending on its policy and ability to retain routing state, the receiving node SHOULD keep a record of the reported NA-DAO message. If the NA-DAO message offers the best route to the prefix as determined by policy and other prefix records, the node SHOULD install a route to the prefix reported in the NA-DAO message via the link local address of the reporting neighbor and it SHOULD further propagate the information in a NA-DAO message.

The RA-DIO message from the DAG root is used to synchronize the whole DAG, including the periodic reporting of destination advertisements back up the DAG. Its period is expected to vary, depending on the configuration of the trickle timer that governs the RAs.

When a node receives a RA-DIO message over an LLN interface from a DA parent, the DelayNA is armed to force a full update.

When the node broadcasts a RA-DIO message on an LLN interface, for all entries on that interface:

o If the entry is CONFIRMED, it goes PENDING with the retry count set to 0.

o If the entry is PENDING, the retry count is incremented. If it reaches a maximum threshold, the entry goes ELAPSED If at least one entry is ELAPSED at the end of the process: if the Destroy timer is not running then it is armed with a jitter.

Since the DelayNA timer has a duration that decreases with the depth, it is expected to receive all NA-DAO messages from all children before the timer elapses and the full update is sent to the DA parents.

Once the RemoveTimer is elapsed, the prefix entry is scheduled to be removed and moved to the Unreachable list if there are any DA parents that need to be informed of the change in status for the prefix, otherwise the prefix entry is cleaned up right away. The prefix entry is removed from the Unreachable list when no more DA parents need to be informed. This condition may be satisfied when a no-DAO is sent to all current DA parents indicating the loss of the prefix, and noting that in some cases parents may have been removed from the set of DA parents.

5.9.2.4. Other events

Finally, the destination advertisement mechanism responds to a series of events, such as:

- o Destination advertisement operation stopped: All entries in the abstract lists are freed. All the routes learned from NA-DAO messages are removed.

- o Interface going down: for all entries in the Reachable list on that interface, the associated route is removed, and the entry is scheduled to be removed.

- o Loss of routing adjacency: When the routing adjacency for a neighbor is lost, as per the procedures described in Section 5.11, and if the associated entries are in the Reachable list, the associated routes are removed, and the entries are scheduled to be destroyed.

- o Changes to DA parent set: all entries in the Reachable list are set to not 'reported' and DelayNA is armed.

5.9.2.5. Aggregation of prefixes by a node

There may be number of cases where a aggregation may be shared within a group of nodes. In such a case, it is possible to use aggregation techniques with destination advertisements and improve scalability.

Other cases might occur for which additional support is required:

1. The aggregating node is attached within the sub-DAG of the nodes it is aggregating for.

2. A node that is to be aggregated for is located somewhere else within the DAG, not in the sub-DAG of the aggregating node.

3. A node that is to be aggregated for is located somewhere else in the LLN.

Consider a node M who is performing an aggregation, and a node N who is to be a member of the aggregation group. A node Z situated above the node M in the DAG, but not above node N, will see the advertisements for the aggregation owned by M but not that of the individual prefix for N. Such a node Z will route all the packets for node N towards node M, but node M will have no route to the node N and will fail to forward.

Additional protocols may be applied beyond the scope of this

specification to dynamically elect/provision an aggregating node and groups of nodes eligible to be aggregated in order to provide route summarization for a sub-DAG.

5.9.2.6. Default Values

DEF_NA_LATENCY = To Be Determined

MAX_DESTROY_INTERVAL = To Be Determined

5.10. Multicast Operation

This section describes further the multicast routing operations over an IPv6 RPL network, and specifically how unicast NA-DAOs can be used to relay group registrations inwards. Wherever the following text mentions MLD, one can read MLDv2 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 root of the DAG, MLD requests are mapped and transported as NA-DAO messages within the RPL protocol; each hop coalesces the multiple requests for a same group as a single NA-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 NA-DAO message to its preferred parent only, in which case multicast packets coming back might be lost for all of its sub-DAG if the transmission fails over that link. Alternatively the router might select to copy additional parents as it would do for NA-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 NA-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 RPL DAG 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 outwards

along the DAG based on the multicast routing table entries installed from the NA-DAO message.

For a source inside the DAG, the packet is passed to the preferred parents, and if that fails then to the alternates in the DAG. 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 DAG root has to further propagate the packet into the external infrastructure.

As a result, the DAG Root acts as an automatic proxy Rendez-vous 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 DAG is grounded or floating, the root can serve inner multicast streams at all times.

5.11. Maintenance of Routing Adjacency

The selection of successors, along the default paths inward along the DAG, or along the paths learned from destination advertisements outward along the DAG, 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.

5.12. Packet Forwarding

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

1. It is preferred to select a successor from a DAG who is supporting an OCP and related optimization that maps to an objective 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 outwards along the sub-DAG), then use that successor.

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

6. If there is a DAG offering a default route with a compatible OCP, then select one of those DAG parents as a successor.

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

8. Finally, if no DAG 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 who was the predecessor of the packet (split horizon), except in the case where it is intended for the packet to change from an inward to an outward flow, such as switching from DIO routes to DAO routes as the

destination is neared.

6. RPL Variables

DIO Timer One instance per DAG that a node is a member of. Expiry triggers RA-DIO message transmission. Trickle timer with variable interval in [0, DIOIntervalMin..2^DIOIntervalDoublings]. See Section 5.3.4

DAG Hop Timer Up to one instance per candidate DAG parent in the `Held-Up' state per DAG that a node is going to jump to. Expiry triggers candidate DAG parent to become a DAG parent in the `Current' state, as well as cancellation of any other DAG Hop timers associated with other DAG parents for that DAG. Duration is computed based on the rank of the candidate DAG parent and DAG delay, as (candidates rank + random) * candidate's DAG_delay (where 0 <= random < 1). See Section 5.7.1.

Hold-Down Timer Up to one instance per candidate DAG parent in the `Held-Down' state per DAG. Expiry triggers the eviction of the candidate DAG parent from the candidate DAG parent set. The interval should be chosen as appropriate to prevent flapping. See Section 5.7.

DAG Heartbeat Timer Up to one instance per DAG that the node is acting as DAG root of. May not be supported in all implementations. Expiry triggers revision of DAGSequenceNumber, causing a new series of updated RA-DIO message to be sent. Interval should be chosen appropriate to propagation time of DAG and as appropriate to application requirements (e.g. response time vs. overhead). See Section 5.4

DelayNA Timer Up to one instance per DA parent (the subset of DAG parents chosen to receive destination advertisements) per DAG. Expiry triggers sending of NA-DAO message to the DA parent. The interval is to be proportional to DEF_NA_LATENCY/(node rank), such that nodes of greater rank (further outward along the DAG) expire first, coordinating the sending of NA-DAO messages to allow for a chance of aggregation. See Section 5.9.2.1.1

DestroyTimer Up to one instance per DA entry per neighbor (i.e. those neighbors who have given NA-DAO messages to this node as a DAG parent) Expiry triggers a change in state for the DA entry, setting up to do unreachable (No-DAO) advertisements or

immediately deallocating the DA entry if there are no DA parents. The interval is min(MAX_DESTROY_INTERVAL, RA_INTERVAL). See Section 5.9.2.1.1

7. 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.

7.1. Control of Function and Policy

7.1.1. Initialization Mode

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

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

7.1.2. 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

NodePreference

DAGDelay

DIOIntervalDoublings

DIOIntervalMin:

DAGObjectiveCodePoint

PathDigest

DAGID

Destination Prefixes

DAG Root behavior: In some cases, a node may not want to permanently act as a DAG root if it cannot join a grounded DAG. For example a battery-operated node may not want to act as a DAG 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 DAG root for a configured period of time.

DAG Hop Timer: A RPL implementation MUST provide the ability to configure the value of the DAG Hop Timer, expressed in ms.

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

7.1.3. Trickle Timers

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

For each DAG, a RPL implementation MUST allow for the monitoring of the following parameters, further described in Section 5.3.4:

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.

7.1.4. DAG Heartbeat

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

7.1.5. The Destination Advertisement Option

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

- o The DelayNA timer
- o The Remove timer

7.1.6. Policy Control

DAG discovery enables nodes to implement different policies for selecting their DAG 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 DAG, 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 NA-DAO messages according to a set of rules defined by policy.

7.1.7. 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.

7.2. 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.

7.3. 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 5.2, an implementation must maintain a set of data structures in support of DAG discovery:

- The candidate neighbors data structure
- For each DAG:
    * A set of candidate DAG parents
    * A set of DAG parents (which are a subset of candidate DAG parents and may be implemented as such)

7.3.1. 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.

7.3.2. Directed Acyclic Graph (DAG) Table

For each DAG, a RPL implementation MUST keep track of the following DAG table values:

- DAGID
- DAGObjectiveCodePoint
- A set of Destination Prefixes offered inwards along the DAG
- A set of candidate DAG Parents

- timer to govern the sending of RA-DIO messages for the DAG
- DAGSequenceNumber

The set of candidate DAG 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 candidate DAG Parent
- a state associated with the role of the candidate as a potential DAG Parent {Current, Held-Up, Held-Down, Collision}, further described in Section 5.7
- A DAG Hop Timer, if instantiated
- A Held-Down Timer, if instantiated
- A flag reporting if the Parent is a DA Parent as described in Section 5.9

7.3.3. Routing Table

To be completed.

7.3.4. 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 DAG parent, e.g. if the DAGID has changed.

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

7.3.5. RPL Trickle Timers

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

- o The DIOIntervalMin expressed in ms

- o The DIOIntervalDoublings

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

7.4. 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.

7.5. Requirements on Other Protocols and Functional Components

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

7.6. Impact on Network Operation

To be completed.

8. 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].

9. IANA Considerations

9.1. DAG Information Option (DIO) Base Option

The DAG Information Option is a container option carried within an IPv6 Router Advertisement message as defined in [RFC4861], which might contain a number of suboptions. The base option regroups the minimum information set that is mandatory in all cases.

IANA had defined the IPv6 Neighbor Discovery Option Formats registry. The suggested type value for the DAG Information Option (DIO) Base Option is 140, to be confirmed by IANA.

9.2. New Registry for the Flag Field of the DIO Base Option

IANA is requested to create a registry for the Flag field of the DIO Base Option.

New bit numbers may be allocated only by an IETF Consensus action. Each bit 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

Three flags are currently defined:

Bit	Description	Reference
0	Grounded DAG	This document
1	Destination Advertisement Trigger	This document
2	Destination Advertisement Supported	This document

DIO Base Option Flags

9.3. DAG Information Option (DIO) Suboption

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

DAG Information Option (DIO) Base Option Suboptions

9.4. Destination Advertisement Option (DAO) Option

The RPL protocol extends Neighbor Discovery [RFC4861] and [RFC4191] to allow a node to include a Destination Advertisement Option, which includes prefix information in the Neighbor Advertisements messages. The Neighbor Advertisement messages are augmented with the Destination Advertisement Option (DAO).

IANA had defined the IPv6 Neighbor Discovery Option Formats registry. The suggested type value for the Destination Advertisement Option carried within a Neighbor Advertisement message is 141, to be confirmed by IANA.

9.5. Objective Code Point

This specification requests that an Objective Code Point registry, as to be specified in [I-D.ietf-roll-routing-metrics], reserve the Objective Code Point value 0x0000, for the purposes designated as OCP

0 in this document.

10. Acknowledgements

The ROLL Design Team would like to acknowledge the review, feedback, and comments from Dominique Barthel, Yusuf Bashir, Mathilde Durvy, Manhar Goindi, Mukul Goyal, Quentin Lampin, Philip Levis, Jerry Martocci, Alexandru Petrescu, and Don Sturek.

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

The ROLL Design Team 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.

11. Contributors

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

Email: jpv@cisco.com

Jonathan W. Hui
Arch Rock Corporation
501 2nd St. Ste. 410
San Francisco, CA 94107
USA

Email: jhui@archrock.com

Thomas Heide Clausen
LIX, Ecole Polytechnique, France

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

Richard Kelsey
Ember Corporation
Boston, MA
USA

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

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

Email: stevedh@cs.berkeley.edu

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

12. References

12.1. Normative References

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

12.2. Informative References

[I-D.ietf-bfd-base]
Katz, D. and D. Ward, "Bidirectional Forwarding

Detection", draft-ietf-bfd-base-09 (work in progress),

February 2009.

[I-D.ietf-manet-nhdp]
Clausen, T., Dearlove, C., and J. Dean, "MANET Neighborhood Discovery Protocol (NHDP)", draft-ietf-manet-nhdp-10 (work in progress), July 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-07 (work in progress), September 2009.

[I-D.ietf-roll-home-routing-reqs]
Brandt, A., Buron, J., and G. Porcu, "Home Automation Routing Requirements in Low Power and Lossy Networks", draft-ietf-roll-home-routing-reqs-08 (work in progress), September 2009.

[I-D.ietf-roll-indus-routing-reqs]
Networks, D., Thubert, P., Dwars, S., and T. Phinney, "Industrial Routing Requirements in Low Power and Lossy Networks", draft-ietf-roll-indus-routing-reqs-06 (work in progress), June 2009.

[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-00 (work in progress), April 2009.

[I-D.ietf-roll-terminology]
Vasseur, J., "Terminology in Low power And Lossy Networks", draft-ietf-roll-terminology-01 (work in progress), May 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>.

[RFC2453] Malkin, G., "RIP Version 2", STD 56, RFC 2453, November 1998.

[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.

[RFC4461] Yasukawa, S., "Signaling Requirements for Point-to-Multipoint Traffic-Engineered MPLS Label Switched Paths (LSPs)", RFC 4461, April 2006.

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

[RFC4875] Aggarwal, R., Papadimitriou, D., and S. Yasukawa, "Extensions to Resource Reservation Protocol - Traffic Engineering (RSVP-TE) for Point-to-Multipoint TE Label Switched Paths (LSPs)", RFC 4875, May 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.

Appendix A. 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

Consider the example LLN physical topology in Figure 8. In this example the links depicted are all usable L2 links. Suppose that all links are equally usable, and that the implementation specific policy function is simply to minimize hops. This LLN physical topology then yields the DAG depicted in Figure 9, where the links depicted are the edges toward DAG parents. This topology includes one DAG, rooted by an LBR node (LBR) at rank 1. The LBR node will issue RAs containing DIO, as governed by a trickle timer. Nodes (11), (12), (13), have selected (LBR) as their only parent, attached to the DAG at rank 2, and periodically advertise RA-DIO multicasts. Node (22) has selected (11) and (12) in its DAG parent set, and advertises itself at rank 3. Node (22) thus has a set of DAG parents {(11), (12)} and siblings {((21), (23)}.

```
                  (LBR)
                  / | \
             .---`  |  `----.
            /       |        \
         (11)------(12)------(13)
          | \       | \       | \
          |  `----. |  `----. |  `----.
          |        \|        \|        \
         (21)------(22)------(23)       (24)
```

```
                             |        /|        /|         |
                             |  .----` |  .----` |         |
                             | /       | /       |         |
                            (31)------(32)------(33)------(34)
                             |        /| \       | \       | \
                             |  .----` |  `----. |  `----. |  `----.
                             | /       |        \|        \|        \
                   .--------(41)      (42)      (43)------(44)------(45)
                  /         /        /| \       | \
            .----`    .----`    .----` |  `----. |  `----.
           /         /         /       |        \|        \
        (51)------(52)------(53)------(54)------(55)------(56)
```

Note that the links depicted represent the usable L2 connectivity available in the LLN. For example, Node (31) can communicate directly with its neighbors, Nodes (21), (22), (32), and (41). Node (31) cannot communicate directly with any other nodes, e.g. (33), (23), (42). In this example these links offer bidirectional communication, and `bad' links are not depicted.

Figure 8: Example LLN Topology


```
                                     (LBR)
                                     / | \
                                .---`  |  `----.
                               /       |        \
                            (11)      (12)      (13)
                             | \       | \       | \
                             |  `----. |  `----. |  `----.
                             |        \|        \|        \
                            (21)      (22)      (23)      (24)
                             |        /|        /|         |
                             |  .----` |  .----` |         |
                             | /       | /       |         |
                            (31)      (32)      (33)      (34)
                             |        /| \       | \       | \
                             |  .----` |  `----. |  `----. |  `----.
                             | /       |        \|        \|        \
                   .--------(41)      (42)      (43)      (44)      (45)
                  /         /        /| \       | \
            .----`    .----`    .----` |  `----. |  `----.
```

```
         /          /          /          |         \|         \
       (51)       (52)       (53)       (54)       (55)       (56)
```

Note that the links depicted represent directed links in the DAG overlaid on top of the physical topology depicted in Figure 8. As such, the depicted edges represent the relationship between nodes and their DAG parents, wherein all depicted edges are directed and oriented `up' on the page toward the DAG root (LBR). The DAG may provide default routes within the LLN, and serves as the foundation on which RPL builds further routing structure, e.g. through the destination advertisement mechanism.

Figure 9: Example DAG

B.1. Moving Down a DAG

Consider node (56) in the example of Figure 8. In the unmodified example, node (56) is at rank 6 with one DAG parent, {(43)}, and one sibling (55). Suppose, for example, that node (56) wished to expand its DAG parent set to contain node (55), as {(43), (55)}. Such a change would require node (56) to detach from the DAG, to defer reattachment until a loop avoidance algorithm has completed, and to then reattach to the DAG with {(43), (55)} as it's DAG parents. When node (56) detaches from the DAG, it is able to act as the root of its own floating DAG and establish its frozen sub-DAG (which is empty). Node (56) can then observe that Node (55) is still attached to the original DAG, that its sequence number is able to increment, and deduce that Node (55) is safely not behind Node (56). There is then

little change for a loop, and Node (56) may safely reattach to the DAG, with parents {(43), (55)}. At reattachment time, node (56) would present itself with a rank deeper than that of its deepest DAG parent (node (55) at rank 6), rank 7.

B.2. Link Removed

Consider the example of Figure 8 when link (13)-(24) goes down.

o Node (24) will detach and become the root of its own floating DAG

o Node (34) will learn that its DAG parent is now part of its own floating DAG, will consider that it can remain a part of the DAG rooted at node (LBR) via node (33), and will initiate procedures to detach from DAG (LBR) in order to re-attach at a lower rank.

- o Node (45) will similarly make preparations to remain attached to the DAG rooted at (LBR) by detaching from Node (34) and re-attaching at a lower rank to node (44).

- o Node (34) will complete re-attachment to Node (33) first, since it is able to attach closer to the root of the DAG.

- o Node (45) will cancel plans to detach/reattach, keep node (34) as a DAG parent, and update its dependent rank accordingly.

- o Node (45) may now anyway add node (44) to its set of DAG parents, as such an addition does not require any modification to its own rank.

- o Node (24) will observe that it may reattach to the DAG rooted at node (LBR) by selecting node (34) as its DAG parent, thus reversing the relationship that existed in the initial state.

B.3. Link Added

Consider the example of Figure 8 when link (12)-(42) appears.

- o Node (42) will see a chance to get closer to the LBR by adding (12) to its set of DAG parents, {(32), (12)}

- o Node (42) may be content to leave its advertised rank at 5, reflecting a rank deeper than its deepest parent (32).

- o Node (42) may now choose to remain where it is, with two parents {(12), (32)}. Should there be a reason for Node (42) to evict Node (32) from its set of DAG parents, Node (42) would then advertise itself at rank 2, thus moving up the DAG. In this case,

Node (53), (54), and (55) may similarly follow and advertise themselves at rank 3.

B.4. Node Removed

Consider the example of Figure 8 when node (41) disappears.

- o Node (51) and (52) will now have empty DAG parent sets and be detached from the DAG rooted by (LBR), advertising themselves as the root of their own floating DAGs.

- o Node (52) would observe a chance to reattach to the DAG rooted at (LBR) by adding Node (53) to its set of DAG parents, after an appropriate delay to avoid creating loops. Node (52) will then advertise itself in the DAG rooted at (LBR) at rank 7.

- o Node (51) will then be able to reattach to the DAG rooted at (LBR) by adding Node (52) to its set of DAG parents and advertising itself at rank 8.

B.5. New LBR Added

Consider the example of Figure 8 when a new LBR, (LBR2) appears, with connectivity (LBR2)-(52), (LBR2)-(53).

- o Nodes (52) and Node (53) will see a chance to join a new DAG rooted at (LBR2) with a rank of 2. Node (52) and (53) may take this chance immediately, as there is no risk of forming loops when joining a DAG that has never before been encountered. Note that the nodes may choose to join the new DAG rooted at (LBR2) if and only if (LBR2) offers more optimum properties in line with the implementation specific local policy.

- o Nodes (52) and (53) begin to send RA-DIO messages advertising themselves at rank 2 in the DAGID (LBR2).

- o Nodes (51), (41), (42), and (54) may then choose to join the new DAG at rank 3, possibly to get closer to the DAG root. Note that in a more advanced case, these nodes also remain members of the DAG rooted at (LBR), for example in support of different constraints for different types of traffic.

- o Node (55) may then join the new DAG at rank 4, possibly to get closer to the DAG root.

- o The remaining nodes may choose to remain in their current positions within the DAG rooted at node (LBR), since there is no clear advantage to be gained by moving to DAG (LBR2).

B.6. Destination Advertisement

Consider the example DAG depicted in Figure 9. Suppose that Nodes (22) and (32) are unable to record routing state. Suppose that Node (42) is able to perform prefix aggregation on behalf of Nodes (53), (54), and (55).

- o Node (53) would send a NA-DAO message to Node (42), indicating the availability of destination (53).

- o Node (54) and Node (55) would similarly send NA-DAO messages to Node (42) indicating their own destinations.

- o Node (42) would collect and store the routing state for destinations (53), (54), and (55).

- o In this example, Node (42) may then be capable of representing destinations (42), (53), (54), and (55) in the aggregation (42').

- o Node (42) sends a NA-DAO message advertising destination (42') to Node 32.

- o Node (32) does not want to maintain any routing state, so it adds onto to the Reverse Route Stack in the NA-DAO message and passes it on to Node (22) as (42'):[(42)]. It may send a separate NA-DAO message to indicate destination (32).

- o Node (22) does not want to maintain any routing state, so it adds on to the Reverse Route Stack in the NA-DAO message and passes it on to Node (12) as (42'):[(42), (32)]. It also relays the NA-DAO message containing destination (32) to Node 12 as (32):[(32)], and finally may send a NA-DAO message for itself indicating destination (22).

- o Node (12) is capable to maintain routing state again, and receives the NA-DAO messages from Node (22). Node (12) then learns:
  - * Destination (22) is available via Node (22)
  - * Destination (32) is available via Node (22) and the piecewise source route to (32)
  - * Destination (42') is available via Node (22) and the piecewise source route to (32), (42').

- o Node (12) sends NA-DAO messages to (LBR), allowing (LBR) to learn routes to the destinations (12), (22), (32), and (42'). (42), (53), (54), and (55) are available via the aggregation (42'). It is not necessary for Node (12) to propagate the piecewise source routes to (LBR).

B.7. Example: DAG Parent Selection

For example, suppose that a node (N) is not attached to any DAG, and that it is in range of nodes (A), (B), (C), (D), and (E). Let all nodes be configured to use an OCP which defines a policy such that ETX is to be minimized and paths with the attribute `Blue' should be avoided. Let the rank computation indicated by the OCP simply reflect the ETX aggregated along the path. Let the links between node (N) and its neighbors (A-E) all have an ETX of 1 (which is learned by node (N) through some implementation specific method). Let node (N) be configured to send IPv6 Router Solicitation (RS) messages to probe for nearby DAGs.

- o Node (N) transmits a Router Solicitation.

- o Node (B) responds. Node (N) investigates the RA-DIO message, and learns that Node (B) is a member of DAGID 1 at rank 4, and not `Blue'. Node (N) takes note of this, but is not yet confident.

- o Similarly, Node (N) hears from Node (A) at rank 9, Node (C) at rank 5, and Node (E) at rank 4.

- o Node (D) responds. Node (D) has a RA-DIO message that indicates that it is a member of DAGID 1 at rank 2, but it carries the attribute `Blue'. Node (N)'s policy function rejects Node (D), and no further consideration is given.

- o This process continues until Node (N), based on implementation specific policy, builds up enough confidence to trigger a decision to join DAGID 1. Let Node (N) determine its most preferred parent to be Node (E).

- o Node (N) adds Node (E) (rank 4) to its set of DAG parents for DAGID 1. Following the mechanisms specified by the OCP, and given that the ETX is 1 for the link between (N) and (E), Node (N) is now at rank 5 in DAGID 1.

- o Node (N) adds Node (B) (rank 4) to its set of DAG parents for DAGID 1.

- o Node (N) is a sibling of Node (C), both are at rank 5.

o Node (N) may now forward traffic intended for the default destination inward along DAGID 1 via nodes (B) and (E). In some cases, e.g. if nodes (B) and (E) are tried and fail, node (N) may also choose to forward traffic to its sibling node (C), without making inward progress but with the intention that node (C) or a following successor can make inward progress. Should Node (C) not

have a viable parent, it should never send the packet back to Node (N) (to avoid a 2-node loop).

B.8. Example: DAG Maintenance

```
          :                      :                      :
          :                      :                      :
         (A)                    (A)                    (A)
          |\                     |                      |
          | `-----.              |                      |
          |        \             |                      |
         (B)       (C)          (B)       (C)          (B)
                    |                      |             \
                    |                      |              `-----.
                    |                      |                     \
                   (D)                    (D)                    (C)
                                                                  |
                                                                  |
                                                                  |
                                                                 (D)

              -1-                    -2-                    -3-
```

Figure 10: DAG Maintenance

Consider the example depicted in Figure 10-1. In this example, Node (A) is attached to a DAG at some rank d. Node (A) is a DAG parent of Nodes (B) and (C). Node (C) is a DAG parent of Node (D). There is also an undirected sibling link between Nodes (B) and (C).

In this example, Node (C) may safely forward to Node (A) without creating a loop. Node (C) may not safely forward to Node (D), contained within it's own sub-DAG, without creating a loop. Node (C) may forward to Node (B) in some cases, e.g. the link (C)->(A) is temporarily unavailable, but with some chance of creating a loop (e.g. if multiple nodes in a set of siblings start forwarding `sideways' in a cycle) and requiring the intervention of additional mechanisms to detect and break the loop.

Consider the case where Node (C) hears a RA-DIO message from a Node (Z) at a lesser rank and superior position in the DAG than node (A). Node (C) may safely undergo the process to evict node (A) from its DAG parent set and attach directly to Node (Z) without creating a loop, because its rank will decrease.

Now consider the case where the link (C)->(A) becomes nonviable, and

node (C) must move to a deeper rank within the DAG:

- o Node (C) must first detach from the DAG by removing Node (A) from its DAG parent set, leaving an empty DAG parent set. Node (C) becomes the root of its own floating, less preferred, DAG.

- o Node (D), hearing a modified RA-DIO message from Node (C), follows Node (C) into the floating DAG. This is depicted in Figure 10-2. In general, any node with no other options in the sub-DAG of Node (C) will follow Node (C) into the floating DAG, maintaining the structure of the sub-DAG.

- o Node (C) hears a RA-DIO message from Node (B) and determines it is able to rejoin the grounded DAG by reattaching at a deeper rank to Node (B). Node (C) starts a DAG Hop timer to coordinate this move.

- o The timer expires and Node (C) adds Node (B) to its DAG parent set. Node (C) has now safely moved deeper within the grounded DAG without creating any loops. Node (D), and any other sub-DAG of Node (C), will hear the modified RA-DIO message sourced from Node (C) and follow Node (C) in a coordinated manner to reattach to the grounded DAG. The final DAG is depicted in Figure 10-3

B.9. Example: Greedy Parent Selection and Instability

```
            (A)                    (A)                    (A)
```

```
        |\                          |\                          |\
        | `-----.                   | `-----.                   | `-----.
        |        \                  |        \                  |        \
       (B)       (C)               (B)        \                 |        (C)
                                     \         |                |        /
                                      `-----.  |                | .-----`
                                             \|                 |/
                                             (C)               (B)

            -1-                         -2-                         -3-
```

Figure 11: Greedy DAG Parent Selection

Consider the example depicted in Figure 11. A DAG is depicted in 3 different configurations. A usable link between (B) and (C) exists in all 3 configurations. In Figure 11-1, Node (A) is a DAG parent for Nodes (B) and (C), and (B)--(C) is a sibling link. In Figure 11-2, Node (A) is a DAG parent for Nodes (B) and (C), and Node (B) is also a DAG parent for Node (C). In Figure 11-3, Node (A) is a

DAG parent for Nodes (B) and (C), and Node (C) is also a DAG parent for Node (B).

If a RPL node is too greedy, in that it attempts to optimize for an additional number of parents beyond its preferred parent, then an instability can result. Consider the DAG illustrated in Figure 11-1. In this example, Nodes (B) and (C) may most prefer Node (A) as a DAG parent, but are operating under the greedy condition that will try to optimize for 2 parents.

When the preferred parent selection causes a node to have only one parent and no siblings, the node may decide to insert itself at a slightly higher rank in order to have at least one sibling and thus an alternate forwarding solution. This does not deprive other nodes of a forwarding solution and this is considered acceptable greediness.

o Let Figure 11-1 be the initial condition.

o Suppose Node (C) first is able to leave the DAG and rejoin at a lower rank, taking both Nodes (A) and (B) as DAG parents as depicted in Figure 11-2. Now Node (C) is deeper than both Nodes (A) and (B), and Node (C) is satisfied to have 2 DAG parents.

- o Suppose Node (B), in its greediness, is willing to receive and process a RA-DIO message from Node (C) (against the rules of RPL), and then Node (B) leaves the DAG and rejoins at a lower rank, taking both Nodes (A) and (C) as DAG parents. Now Node (B) is deeper than both Nodes (A) and (C) and is satisfied with 2 DAG parents.

- o Then Node (C), because it is also greedy, will leave and rejoin deeper, to again get 2 parents and have a lower rank then both of them.

- o Next Node (B) will again leave and rejoin deeper, to again get 2 parents

- o And again Node (C) leaves and rejoins deeper...

- o The process will repeat, and the DAG will oscillate between Figure 11-2 and Figure 11-3 until the nodes count to infinity and restart the cycle again.

- o This cycle can be averted through mechanisms in RPL:

  - * Nodes (B) and (C) stay at a rank sufficient to attach to their most preferred parent (A) and don't go for any deeper (worse)

alternate parents (Nodes are not greedy)

  - * Nodes (B) and (C) do not process RA-DIO messages from nodes deeper than themselves (because such nodes are possibly in their own sub-DAGs)

B.10. Example: DAG Merge

```
                          :
                          :
                         (A)       (D)
                          |         |
                          |         |
```

```
                      |        |
                     (B)      (E)
                      |        |
                      |        |
                      |        |
                     (C)      (F)
```

Figure 12: Merging DAGs

Consider the example depicted in Figure 12. Nodes (A), (B), and (C) are part of some larger grounded DAG, where Node (A) is at a rank of d, Node (B) at d+1, and Node (C) at d+2. The DAG comprised of Nodes (D), (E), and (F) is a floating, less preferred, DAG, with Node (D) as the DAG root. This floating DAG may have been formed, for example, in the absence of a grounded DAG or when Node (D) had to detach from a grounded DAG and (E) and (F) followed. All nodes are using compatible objective code points.

Nodes (D), (E), and (F) would rather join the more preferred grounded DAG if they are able than to remain in the less preferred floating DAG.

Next, let links (C)--(D) and (A)--(E) become viable. The following sequence of events may then occur in a typical case:

o Node (D) will receive and process a RA-DIO message from Node (C) on link (C)--(D). Node (D) will consider Node (C) a candidate neighbor and process the RA-DIO message since Node (C) belongs to a different DAG (different DAGID) than Node (D). Node (D) will note that Node (C) is in a grounded DAG at rank d+2, and will begin the process to join the grounded DAG at rank d+3. Node (D) will start a DAG Hop timer, logically associated with the grounded DAG at Node (C), to coordinate the jump. The DAG Hop timer will

have a duration proportional to d+2.

o Similarly, Node (E) will receive and process a RA-DIO message from Node (A) on link (A)--(E). Node (E) will consider Node (A) a candidate neighbor, will note that Node (A) is in a grounded DAG at rank d, and will begin the process to join the grounded DAG at rank d+1. Node (E) will start a DAG Hop timer, logically associated with the grounded DAG at Node (A), to coordinate the jump. The DAG Hop timer will have a duration proportional to d.

- Node (F) takes no action, for Node (F) has observed nothing new to act on.

- Node (E)'s DAG Hop timer for the grounded DAG at Node (A) expires first. Node (E), upon the DAG Hop timer expiry, removes Node (D) as its parent, thus emptying the DAG parent set for the floating DAG, and leaving the floating DAG. Node (E) then jumps to the grounded DAG by entering Node (A) into the set of DAG parents for the grounded DAG. Node (E) is now in the grounded DAG at rank d+1. Node (E), by jumping into the grounded DAG, has created an inconsistency by changing its DAGID, and will begin to emit RA-DIO messages more frequently.

- Node (F) will receive and process a RA-DIO message from Node (E). Node (F) will observe that Node (E) has changed its DAGID and will directly follow Node (E) into the grounded DAG. Node (F) is now a member of the grounded DAG at rank d+2. Note that any additional sub-DAG of Node (E) would continue to join into the grounded DAG in this coordinated manner.

- Node (D) will receive a RA-DIO message from Node (E). Since Node (E) is now in a different DAG, Node (D) may process the RA-DIO message from Node (E). Node (D) will observe that, via node (E), it could attach to the grounded DAG at rank d+2. Node (D) will start another DAG Hop timer, logically associated with the grounded DAG at Node (E), with a duration proportional to d+1. Node (D) now is running two DAG hop timers, one which was started with duration proportional to d+1 and one proportional to d+2.

- Generally, Node (D) will expire the timer associated with the jump to the grounded DAG at node (E) first. Node (D) may then jump to the grounded DAG by entering Node (E) into its DAG parent set for the grounded DAG. Node (D) is now in the grounded DAG at rank d+2.

- In this way RPL has coordinated a merge between the more preferred grounded DAG and the less preferred floating DAG, such that the nodes within the two DAGs come together in a generally ordered manner, avoiding the formation of loops in the process.

Appendix C. Additional Examples

Consider the expanded example LLN physical topology in Figure 13. In this example an additional LBR is added. Suppose that all nodes are configured with an implementation specific policy function that aims to minimize the number of hops, and that both LBRs are configured to root different DAGIDs. We may now walk through the formation of the two DAGs.

```
                             (LBR)                    (LBR2)
                             / | \                    /    \
                        .---`  |  `----.             /      \
                       /       |        \            |      |
                    (11)------(12)------(13)      (14)      (15)
                     | \       | \       | \       |        /|
                     |  `----. |  `----. |  `----. |  .----` |
                     |        \|        \|        \| /       |
                    (21)------(22)------(23)      (24)      (25)
                     |        /|        /|         |        / /
                     |  .----` |  .----` |  .-----]|[------` /
                     | /       | /       | /       |        /
                    (31)------(32)------(33)------(34)-----`
                     |        /| \       | \       | \
                     |  .----` |  `----. |  `----. |  `----.
                     | /       |        \|        \|        \
           .--------(41)      (42)      (43)------(44)------(45)
          /         /         /| \       | \
    .----`    .----`    .----` |  `----. |  `----.
   /         /         /       |        \|        \
(51)------(52)------(53)------(54)------(55)------(56)
```

Figure 13: Expanded LLN Topology

```
                                     (LBR)                    (LBR2)
                                     / | \                    /    \
                                .---`  |  `----.             /      \
                               /       |        \            |      |
                            (11)      (12)      (13)      (14)      (15)



                            (21)      (22)      (23)      (24)      (25)



                            (31)      (32)      (33)      (34)



                            (41)      (42)      (43)      (44)      (45)



        (51)      (52)      (53)      (54)      (55)      (56)
```

Figure 14: DAG Construction Step 1

```
                                     (LBR)                    (LBR2)
                                     / | \                    /    \
                                .---`  |  `----.             /      \
                               /       |        \            |      |
                            (11)      (12)      (13)      (14)      (15)
                             | \       | \       |         |        /|
                             |  `----. |  `----. |         |  .----` |
                             |        \|        \|         | /       |
                            (21)      (22)      (23)      (24)      (25)



                            (31)      (32)      (33)      (34)
```

```
                            (41)      (42)      (43)      (44)      (45)



        (51)      (52)      (53)      (54)      (55)      (56)
```


Figure 15: DAG Construction Step 2

```
                                     (LBR)                    (LBR2)
                                     / | \                    /    \
                                .---`  |  `----.             /      \
                               /       |        \            |      |
                            (11)      (12)      (13)      (14)      (15)
                             | \       | \       |         |        /|
                             |  `----. |  `----. |         |  .----` |
                             |        \|        \|         | /       |
                            (21)      (22)      (23)      (24)      (25)
                             |        /|        /          |        / /
                             |  .----` |  .----`    .-----]|[------` /
                             | /       | /         /       |        /
                            (31)      (32)      (33)      (34)-----`


                            (41)      (42)      (43)      (44)      (45)



        (51)      (52)      (53)      (54)      (55)      (56)
```

Figure 16: DAG Construction Step 3

```
                                     (LBR)                    (LBR2)
                                     / | \                     /    \
                                .---`  |  `----.             /      \
                               /       |        \            |      |
                            (11)      (12)      (13)      (14)      (15)
                             | \       | \       |         |        /|
                             |  `----. |  `----. |         |  .----` |
                             |        \|        \|         | /       |
                            (21)      (22)      (23)      (24)      (25)
                             |        /|        /          |        / /
                             |  .----` |  .----`    .-----]|[------` /
                             | /       | /         /       |        /
                            (31)      (32)      (33)      (34)-----`
                             |        /|         | \       | \
                             |  .----` |         |  `----. |  `----.
                             | /       |         |        \|        \
                            (41)      (42)      (43)      (44)      (45)



        (51)      (52)      (53)      (54)      (55)      (56)
```

Figure 17: DAG Construction Step 4

```
                                     (LBR)                    (LBR2)
                                     / | \                     /    \
```

```
                                .---`  |  `----.             /      \
                               /       |        \            |      |
                            (11)      (12)      (13)      (14)      (15)
                             | \       | \       |         |       /|
                             |  `----. |  `----. |         |  .----` |
                             |        \|        \|         | /       |
                            (21)      (22)      (23)      (24)      (25)
                             |        /|        /          |        / /
                             |  .----` |  .----`    .-----]|[------` /
                             | /       | /         /       |        /
                            (31)      (32)      (33)      (34)-----`
                             |        /|         | \       | \
                             |  .----` |         |  `----. |  `----.
                             | /       |         |        \|        \
                   .--------(41)      (42)      (43)      (44)      (45)
                  /         /         /|         | \
            .----`    .----`    .----` |         |  `----.
           /         /         /       |         |        \
        (51)      (52)      (53)      (54)      (55)      (56)
```


Figure 18: DAG Construction Step 5

Appendix D. Outstanding Issues

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

D.1. Additional Support for P2P Routing

In some situations the baseline mechanism to support arbitrary P2P traffic, by flowing inward along the DAG until a common parent is reached and then flowing outward, may not be suitable for all application scenarios. A related scenario may occur when the outward paths setup along the DAG by the destination advertisement mechanism are not be the most desirable outward paths for the specific application scenario (in part because the DAG 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.

D.2. Loop Detection

It is under investigation to complement the loop avoidance strategies provided by RPL with a loop detection mechanism that may be employed when traffic is forwarded.

D.3. Destination Advertisement / DAO Fan-out

When NA-DAO messages are relayed to more than one DAG parent, in some cases a situation may be created where a large number of NA-DAO messages conveying information about the same destination flow inward along the DAG. It is desirable to bound/limit the multiplication/ fan-out of NA-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.

D.4. Source Routing

In support of nodes who 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.

D.5. 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.

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: dtroll@external.cisco.com