ns-3 RPL module: IPv6 Routing Protocol for Low power and Lossy Networks

Lorenzo Bartolozzi
Student
Università di Firenze
Firenze, Italy
lore_barto@hotmail.it

Tommaso Pecorella
Assistant Professor
Università di Firenze
Firenze, Italy
tommaso.pecorella@unifi.it

Romano Fantacci
Full Professor
Università di Firenze
Firenze, Italy
romano.fantacci@unifi.it

ABSTRACT

This paper presents a framework for ns-3 (Network Simulator 3) of a particular wireless sensor network routing protocol, called RPL (IPv6 Routing Protocol for Low power and Lossy Networks). It will be presented with a design, its implementation and a simple example to demonstrate its operation. Finally, possible future developments are suggested and the appropriate conclusions will be drawn.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Routing protocols*; I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling methodologies*

General Terms

Algorithms, Design, Performance

Keywords

ns-3, RPL, IPv6, Wireless Sensor Networks

1. INTRODUCTION

Simulating a routing protocol is never an easy task. A number of factors have to be taken into account, especially when the routing protocol is trying to optimize routes based on dynamic parameters like energy, transmission delays and so on. The researcher should have a complete control over the simulated environment and a deep knowledge of the simulation tool in order to derive meaningful results. Of course 'meaningful' does not mean a 1:1 equivalence with a real implementation, it means something that can be generalized and is valid (not numerically but qualitatively) for a broad range of implementations.

This drives the authors to re-implement the RPL protocol in ns-3, even though a similar implementation does exist for Omnet++ Castalia [1] and ultimately RPL could be 'tested' in real nodes using for example Contiki [2] or TinyOS [3] or their built-in simulators (Cooja and TOSSIM). Castalia implementation is quite limited at the moment, while Cooja and TOSSIM are quite hard to modify in order to add new features or modify the existing ones. A fresh-made ns-3 implementation allows us to achieve the following goals: 1) a deep understanding of the protocol, including its (eventual) pitfalls, 2) a full control of the implementation and 3) most importantly, the capability to add non-existing hooks for new proposals to be submitted to the standardization body. Ns-3 had the right capabilities, such as full IPv6 support, while some missing parts (e.g., the 802.15.4 stack) were in development when we started the RPL implementation project.

RPL is the ROLL [10] IETF Working Group main outcome, and it is a completely new routing protocol especially tailored for Low Power (Sensor) Networks communicating in IPv6. The authors believe that if and when the so-called Internet of Things will become a reality, RPL will be its routing protocol. RPL has been developed considering Wireless Sensor Network (WSN, e.g., IEEE 802.15.4) as primary target, however it is not limited to WSN applications. It has been designed based on specific requirements.

RPL is a proactive routing protocol, so it builds the routes in advance. Due to the WSN dynamic nature, the protocol is very reactive to topology changes, and it can optimize the routes on the base of a number of different metrics. Moreover also a reactive mode of operation has been recently proposed.

The paper is organized as follows: in section two we outline the RPL routing protocol, in section three we describe the ns-3 module design, in section four its implementation, in section five we show the test results and section six covers the next steps and future work.

2. RPL ROUTING PROTOCOL

RPL [13] is the routing protocol developed by the ROLL Working Group specifically for the so-called 'Low Power and Lossy Networks'. This includes, but it is not limited to, Wireless Sensor Networks. The protocol is based on IPv6. IPv4 is not even considered as an option.

RPL was developed from four sets of requirements [5, 9, 4, 8] that represent the four main foreseen uses of WSN: Home Automation, Building Automation, Industrial and Urban environments. The documents highlight many differences in the various environments, but on one point they're mostly agreeing: the main use of WSNs is foreseen to be *data gathering* or *data distribution*. As a consequence, RPL focuses on building very efficient routes between one or more 'root'

WNS3 2012, March 23, Desenzano del Garda, Italy

DOI 10.4108/icst.simutools.2012.247718

nodes and all the other nodes in the network.

RPL supports actively three kinds of traffic: Point to Point, Point to Multipoint and Multipoint to Point, with four Mode of Operation (MOP). The multiple MOPs are due to the WSNs peculiarities. Even in 802.15.4, two different kinds of nodes are defined: Full Functional Devices (FFD) and Reduced Functional Devices (RFD), in order to accommodate the fact that some nodes might have little or no computational power. Moreover some RPL nodes might have more stringent constraints, hence the different MOPs. The four MOPs are: 'No downward routes', Non Storing, Storing, Storing with Multicast support. The differences will be clear in the following.

The base concept in RPL is the Destination Oriented Directed Acyclic Graph (DODAG). Assuming that the biggest part of the node's communication are flowing to or from a root node (the gateway between the WSN and the Internet usually), it is pretty straightforward that building a DODAG having as the destination the root node is equivalent to solve the routing problem. The whole RPL protocol tries to build, maintain and optimize DODAGs. The base idea is simple; the solution is not so simple.

A DODAG is 'built' by a root node, however in a network there might exist multiple root nodes. Moreover even a single root node could need to have multiple DODAGs due to (mainly) node's or traffic constraints. To make the problem more complex, the nodes might be moving, the links might be vanishing due to moving obstacles, etc. Hence, the protocol as a whole is very dynamic. On the other hand, ROLL group made an impressive work to maintain the protocol very simple yet extremely flexible. The base operation is: when a node is not connected to a DODAG it sends special messages named DODAG Information Solicitation (DIS) to a special IPv6 multicast group. Any other node already in a DODAG can send back a DODAG Information Object (DIO). The node will choose the more suitable node and will join its DODAG.

The main remaining issues are: 1) how to prevent loops, 2) how to rebuild a broken DODAG, 3) how to optimize the DODAG, 4) how to discover the 'downward' routes.

2.1 RPL loop prevention

Loop prevention should be ensured by the DODAG building mechanism. Every node has a *rank*, and each node must have a rank greater than its direct parent. Rare cases might exist where, due to delays in the DODAG updates, a loop occurs. In order to completely prevent loops, a loop detection mechanism is also implemented. It is not possible to describe it fully here due to space constraints. The full specification is in [13].

2.2 DODAG repair

Upon discovering a broken DODAG (i.e., with a loop or a fail upward link), a node will simply detach all its sub-DODAGs, basically forcing a disjoin for all the nodes connected to it. It's a bit harsh, but in this way the nodes will rapidly build a new and correct DODAG. A DODAG could also be repaired by switching to a secondary parent, if the implementation is supporting this feature.

2.3 DODAG optimization

A DODAG is built using the so-called Objective Functions (OF) [6, 11] and each OF will use a metric [12]. A metric can be additive, multiplicative, min or max and can involve the number of hops, the link quality, the node's energy and so on. The OF will determine how the DODAG is built and how, when the metric will change, the DODAG will have to be changed according to the new metrics. The process, at the end, is quite simple: the metric will be used to determine the relative rank of a node with respect to its parents or neighbors, and the OF will decide if the node should change its parent and, thus, its rank.

2.4 Downward routes

Upward (i.e., from a node to the root) is already handled by the DODAG. Hence, a node willing to send a packet 'up' has just to send it to its parent. For real, the parent and the DODAG parent don't have to be the same node (see [13]), but this part is a bit too complex to be explained here. Downward routes, viceversa, are the real difference between the various RPL MOPs.

Downward routes are maintained by sending periodic messages named Destination Advertisement Object (DAO) from each node toward its parents. The DAO will contain all the IPs the node can route, so its own IPs, plus all the IPs owned by the nodes belonging to its sub-DODAG. If the DODAG MOP is 0, ('No downward routes'), DAOs are not sent, and the root will have to figure out by itself the downward routes. Usually this implies to use a flooding to reach a node. MOP 1 is 'Non Storing', meaning that the DAOs are sent, but the nodes are unable to store downward routes. Hence, the root will have to act as a gateway, and each packet will reach the root and (eventually) will be re-sent to the destination. Downward packets will need a sort of source routing. MOP 2 and 3, 'Storing' and 'Storing with multicast', are the most powerful ones and will imply that each node can act as a router, i.e., is a Full Function Device (FFD). Hence, the nodes are able to send packets from one of their sub-DODAGS to another one, if the two are not connected.

In our implementation, we decided to focus on MOP 3, as it is the most flexible one. Moreover, with the actual microcontrollers' computing power and storage capability, we do expect that MOP 3 will be the preferred one.

This simple description is not exhaustive, and the definite source of information is the RPL draft [13] and the other drafts available at the ROLL Working Group. Other features do include P2P direct routes (in a reactive AODV-like fashion), direct neighbor communication (one-hop optimization), routes based on constraints, etc. Due to its flexibility and the separation between the routing protocol, the objective functions and the metrics, RPL is more than a simple routing protocol; it is instead a routing framework open to experiments and features development.

3. RPL MODULE DESIGN

This section will describe the design of the RPL ns-3 module. It will be shown a simplified diagram of the classes and the most significant sequence diagrams. Some design decisions taken during the protocol implementation will conclude this section.

3.1 Classes Diagram

Figure 1 shows the module's simplified UML class diagram. The design complies with the structure of a typical IPv6 ns-3 routing protocol.

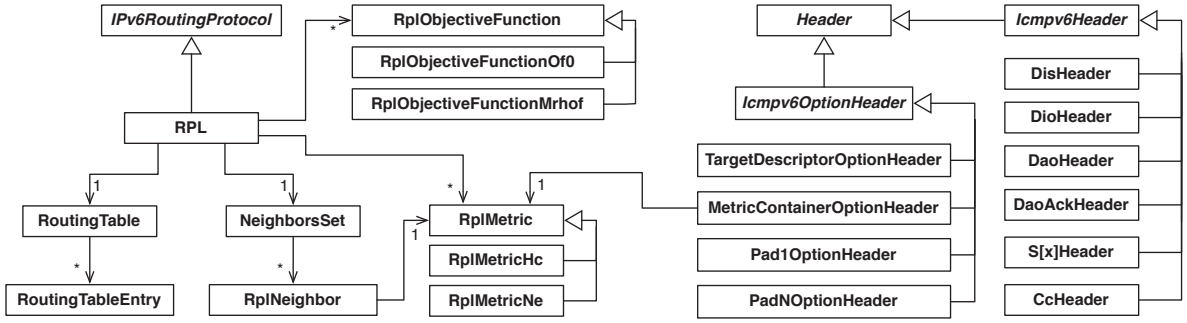


Figure 1: Simplified UML classes diagram.

The main class, representing the routing logic, is obviously named "rpl::Rpl" (we used a specific namespace) and is derived from IPv6RoutingProtocol. There is as well a Helper class, not depicted in the figure.

All the RPL messages (e.g., DIS, DIO, DAO, etc.) are directly derived from Icmpv6Header, as they are all new ICMP types according to the RPL standard. The other headers, according to the standard, are ICMPv6 Option Headers, so they are as well in the model. They are used to add the required information to the new ICMPv6 messages.

The MetricContainerOptionHeader is worth describing further. Since the DIO messages (but not only those) can contain metric information, the header will contain an RplMetric instance. The class, however, is abstract, as the 'real' metrics are its subclasses. The header will contain as well an indication of the OF used by the DODAG, but just as an enumeration type.

The Rpl class itself, moreover, will need one or more RplMetric instances, to hold the values for the node, and one or more RplObjectiveFunction instances. This one as well is an abstract class and the working classes are its subclasses. These ones are effectively responsible for calculating the rank of a node compared to its neighbors, i.e., for choosing the parents and, eventually, for optimizing the DODAG. It is worth noticing that this allows to keep the Rpl class agnostic about the OF or the metric used. Depending on the metrics and OFs installed in a node during the node creation, a node can or cannot join a specific DODAG. The standard, indeed, does not assume that every node must support all the possible OFs and metrics; a node must discard all the MetricContainerOptionHeader objects containing unsupported metrics or OFs.

The last part is the routing table itself and the RplNeighbor set. The first is, of course, the routing table. We did not use the ns-3 one, as we needed some more fields not present in the ns3::Ipv6RoutingTableEntry. In section 4.2 we will discuss this point. The RplNeighbor set is needed to hold the data about the surrounding nodes. Each of them will have an associate metric, especially useful for parent selection. Moreover the RplNeighbor set is used to store the timers for node reachability check functions.

In principle the two structures (Routing Table and Neighbor Set) could be merged, but we decided to have them separate for efficiency purposes, as they are stored in a completely different way. In the future they could be merged into one single class, provided the lookup speed is not affected.

3.2 Sequence Diagrams

RPL operations are easily described using simple sequence diagrams. We will show the most relevant ones. It is worth noticing that RPL does use 'trickle timers' [7] to time its operations. Hence, timer's duration is adaptive.

3.2.1 DODAG join

This action takes place after the start of the protocol and also after the disjoin from a DODAG. The node will create a multicast DIS with the Solicited Information option and will wait for a response. The response will be a DIO with the DODAG Configuration option. This process is repeated until a DIO is received.

3.2.2 DODAG disjoin

When a node is part of a DODAG and for metric or administrative reasons wants to disassociate, it will perform a disjoin.

This procedure consists in disabling the trickle timer (if it was active), communicating the disjoin from the DODAG to the neighbors by sending a poisoning DIO to each of them and deleting all the rows from the routing table and the neighbors set. The poisoning DIO will remove that node from the routing table and the neighbors set. The node will then try to join a new DODAG.

3.2.3 DIS receive

Upon receiving a DIS, a node creates a DIO containing DODAG Configuration option and sends it back. This also triggers a trickle timer reset (if active). The trickle timer is used to send periodic, unsolicited, DIOs.

The DIS source node is also added to the neighbors set as 'NEIGHBOR_NOT_IN_DODAG' and a temporary route is created. The route will be discarded as soon as the DIO is sent.

3.2.4 DIO receive

The actions that follow the receipt of a DIO are different according to the state of the node. For convenience, given the considerable length of the diagram, we decided to omit it and insert just the description.

The node can be in one of these two states: (1) it is not yet joined to a DODAG or (2) it is already joined to one.

1) This case is the consequence of a DODAG join attempt. Hence, the node will copy the values contained in the DIO in his DODAG attributes, his rank will be calculated, its

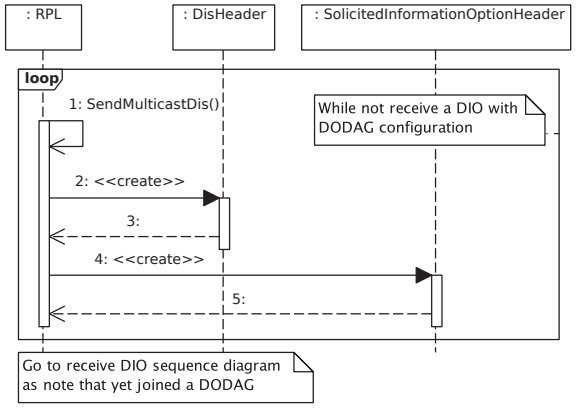


Figure 2: Join sequence diagram.

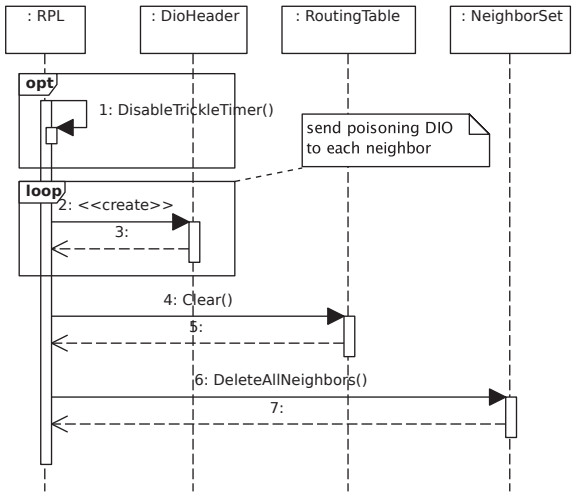


Figure 3: Disjoin sequence diagram.

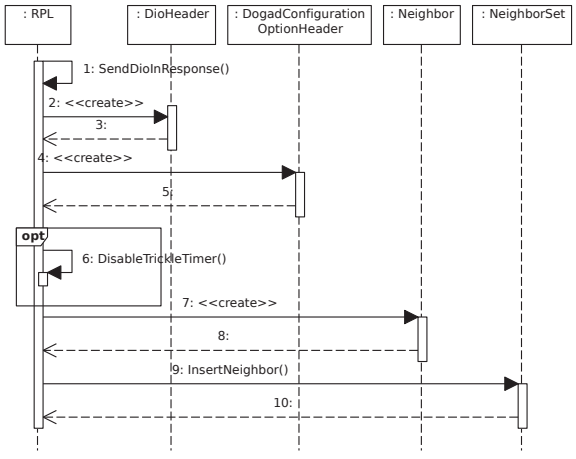


Figure 4: Receive DIS sequence diagram.

status updated and, if it is a 'ROUTER', the trickle timer is launched. The DIO sender will be stored in the neighbors set with type 'DODAG_PARENT_ONLY_PRF'. The sender, of course, will be automatically selected as the preferred DODAG parent. The routing table is updated accordingly.

2) In the second case we have two more possibilities: (a) the DIO is a poisoning one or (b) it is a periodic DIO sent to maintain the upward routes.

a) In this first case the node will need to delete all the sender's entry and all the routes that have the sender as next hop from the routing table. The sender is also removed from the neighbors set . If the sender was a preferred parent, another parent is searched in the neighbor set. If no parent is found, the node will disjoin himself from the DODAG.

b) In the event that the DIO is a periodic message then the node will check if the version of DODAG has been increased or not.

i) If so the node will join the new version of DODAG, so it will reset the routing table, the neighbors set and, if it is active, the trickle timer. This is an implementation choice, as the standard does not suggest anything about this point. Better choices could be tested in the future.

ii) If the DODAG version is unchanged, then the 'DTSN' field is checked. If it is different from the last one that the sending node sent, a DAO will be scheduled. Then, if the rank of the sender node is lesser than the one of the current node, the sender will be stored in neighbors set, checking if the new node is to become the preferred parent and, as a consequence, to update the node's rank. The sender node is in any case stored in the neighbor set and the routing table is updated accordingly.

3.2.5 DAO receive

DAO messages are used to inform the nodes about the downward routes. A DAO might require an acknowledgement; in this case a DAO-ACK is sent.

The DAO message is then processed and, for each entry contained in it (the exact DAO message structure is too complex to be described here) the routing table is updated accordingly, adding (or updating) each entry and setting the DAO sender as next hop.

After the update, the changed entries are stored in a new DAO message and this is immediately sent to the Dao parents. Usually the DAO parent is just the preferred parent, but the standard allows multiple DAO parents in order to optimize routes and/or have fallback downward paths. This immediate send is important to quickly update the downward routes of the upper nodes, i.e., the ones along the path towards the root.

3.3 Design Choices

During the design phase some choices have been made about many aspects of the protocol. The most relevant ones are: (1) the mode of operation, (2) the objective functions and metrics, (3) messages and security, and (4) messages options.

1) It was decided to develop only the storing operation mode with or without multicast. Those two MOPs differ only in minimal parts, and basically the non-multicast is achieved by preventing multicast routing. The other MOPs should be easy to add in the future, as they are a reduced functionality with respect to the storing one.

2) We chose to implement only two of the metrics defined

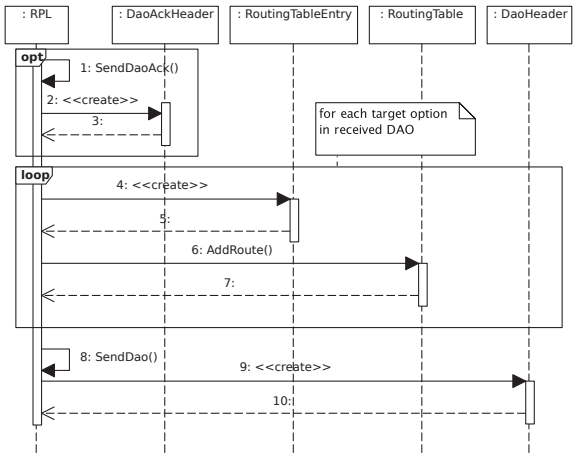


Figure 5: Receive DAO sequence diagram.

in [12], NE (Node Energy) and HC (Hop Count), and all the objective functions currently proposed, OF0 (Objective Function 0) [11] and MRHOF (Rank Hysteresis Minimum Objective Function) [6]. The missing metrics (e.g., Throughput, Latency, etc.) do require testing on an 802.15.4 MAC, so their development has been deferred.

3) We decided, on the other hand, to implement all messages types, even if some of them are not used by the protocol right now. The unused message types are all the secure ones (SDIS, SDIO, SDAO, SDAO-ACK, CC) plus the DAO-ACK. The DAO-ACK just needs the proper handling code, and will be added in the future. On the other hand it is not mandatory so we delayed its implementation. Secure messages have been implemented to allow an evaluation of the secure types overhead over the non-secure ones. At the moment their control logic is to be finalized, and it will be before the official module release.

4) Finally, regarding the messages options, the following ones have been implemented but they are not currently used (a) Prefix Information, (b) Route Information, and (c) Target Descriptor options.

a) The first one is used for the IPv6 prefixes auto-configuration, very useful in real networks (especially in the types of network with a huge number of nodes as those to which RPL is dedicated). How to use this option is strictly bound to how the IPv6 addresses are assigned to the nodes, so it needs some cross-layer work in order to setup the right use policy. Its logic is heavily dependent on, and overlaps with, DHCP, RADV and so on. This option is very interesting but its use might imply the definition of a common interface between all the node configuration options, so its full implementation is left for future work. Moreover it should be checked if this option will be effectively used or not by real networks.

b) Route information option is used by the root node to inform about external networks reachability. This option is useful in multi-root scenarios ('virtual roots' in the standard). Since this scenario has not yet been considered, and its testing is not in the priorities list, the option is actually not used. This does not affect the protocol, since the (single) root acts as the default gateway anyway.

c) The Target Descriptor option is used to give a description of the target nodes. The standard is quite vague about the use of Target Descriptors, but they might be used to prioritize routing over different nodes. We did implement the option as it could be useful for experimenting and research beyond the bare-bone standard.

4. RPL MODULE IMPLEMENTATION

This section describes some details of the ns-3 RPL module. A full class description is out of scope here, so we just focus on the most relevant classes and their main functions.

4.1 RPL Class

RPL class represents the routing protocol, as already discussed. Since RPL is an IPv6-only routing protocol, it will extend Ipv6RoutingProtocol and will implement its methods. Beside RouteInput and RouteOutput, where the real forwarding logic is handled, the important methods are RecvDis, RecvDio, RecvDao, Join and Disjoin. These five methods, as a matter of fact, are the ones driving the protocol behavior by implementing the actions described above in section 3.2.

All the logic for computing the rank is instead delegated to the RplObjectiveFunction class ComputeRank method, which, based on the rank of a given parent and its metric, sets the node's rank and decides if the node should change the actual parent node. This separation makes extending the Objective Functions very easy.

The RPL class also contains some methods for the management of the trickle timer for the DIO scheduling, the methods to send all the various management packet types and many other configuration methods. Almost all the parameters are set through Attributes, so the class is fully customizable.

4.2 RPL Routing Tables

The implementation of the routing table involves two classes: (a) RoutingTableEntry and (b) RoutingTable.

a) The class RoutingTableEntry represents a single row of the routing table. Beside the destination address, its network mask, the next-hop address and the output interface, it also contains some specific RPL parameters, such as the advertising neighbor information and the path sequence number. Ns-3 already has the IPv6RoutingTableEntry class, but it was not suitable for this use, as it contains elements that do not apply to RPL, such as the gateway and the default route. RPL, on the contrary, is much closer to a multi-hop routing system. The derived class would have almost nothing shared with the parent class.

b) The class RoutingTable represents the routing table. Hence, it contains a RoutingTableEntry list and several methods for searching, insertion and deletion of routes.

4.3 RPL Neighbors

The neighbors' set is made also by two classes: (a) RplNeighbor and (b) NeighborsSet.

a) The RplNeighbor class is very specific to the RPL protocol. It represents a node 'known' by the given node, either because they both are belonging to the same DODAG or because the node did receive some information from that node (e.g., a multicast DIS). So a neighbor node can be: (1) a node that is not in the same DODAG, (2) a node that belongs to the node's sub-DODAG (therefore with higher rank), (3) a DODAG parent (therefore with lower rank), (4) a DAO parent (a DODAG parent selected to send the DAOs to) or (5) a preferred DODAG or DAO parent. According to

the neighbor's kind, a different message processing is triggered. Beside the basic node data (address and interface on which it was detected), this class contains some needed node's parameters, such as the rank, the metric, the DTSN and, for the Neighbor Unreachability Detection, the NUD number.

b) The neighbor set is represented by the class NeighborsSet. This is simply an RplNeighbor list, but provides very useful methods for the neighbor management, such as search and update methods.

4.4 RPL Helper

The class RplHelper provides several methods to create and install the RPL protocol on different nodes of a network. By using this class, with a few lines of code you can customize all the nodes, e.g., by setting which objective functions and metrics a node can use or deciding which nodes will act as DODAG roots, along with their OF and metric.

The Helper class does this by simply using the names of the intended OFs and metrics, hence it is automatically supporting new Objective Functions or metrics.

4.5 Extra classes

The implementation also has documentation and examples. However, at present both should be finalized. The manuals should be completed, as the module is still being written and the documentation has to follow the changes. The examples will be finalized once the 802.15.4 (and SixLowPAN) modules will be available, in order to offer the user a 'good' example and not the actual debugging scenario.

5. RPL TESTING

The design and implementation have been tested on a simple network, depicted in figure 6a. The test has been performed using the `--vis` ns-3 option, as is using the *visualizer* module. Since the 802.15.4 and SixLowPan ns-3 modules are still unfinished, the demonstration has been performed using point-to-point links. The solution is sub-optimal, but it allows understanding the RPL operations. The ns-3 visualizer tool has been used to show the network structure, the main steps of the test and its results.

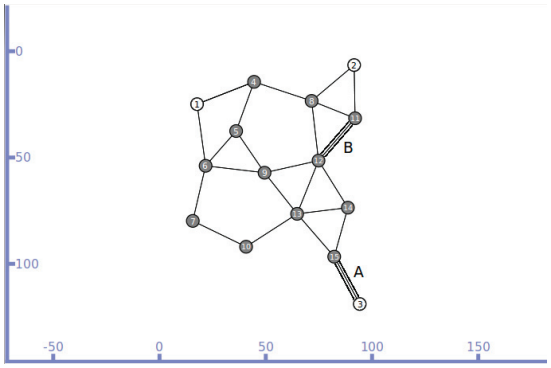
5.1 Test network

Figure 6a shows the test network. The nodes with the light background are the DODAG roots. Node 1 exports a DODAG with objective function OF0, node 2 a DODAG with MRHOF with NE metric, and node 3 a DODAG with MRHOF with HC metric. The links highlighted (link A and link B) will be 'broken' during the simulation in order to show the effects of link failures. Link A will stop at 40s and link B at 80s. The links have been chosen so to force nodes to re-join a different DODAG, as link A is the only one connecting root 3, while link B will disjoin a large part of the root 2 DODAG.

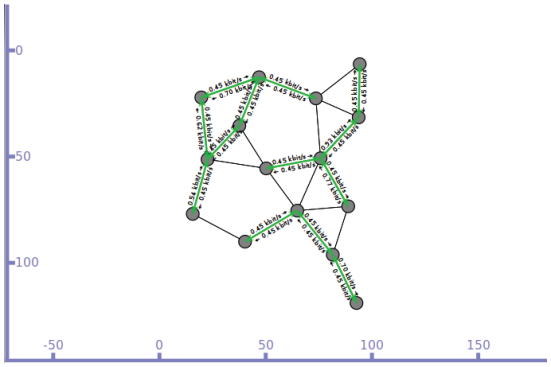
The various links have different bandwidth and delays, and the nodes have a different remaining energy, fixed at simulation start. The node's energy does not change during the simulation.

5.2 Test steps

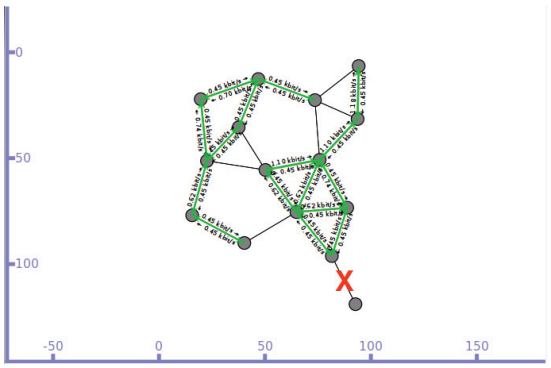
At the beginning of the simulation, the nodes start immediately to transmit the various RPL messages. The nodes



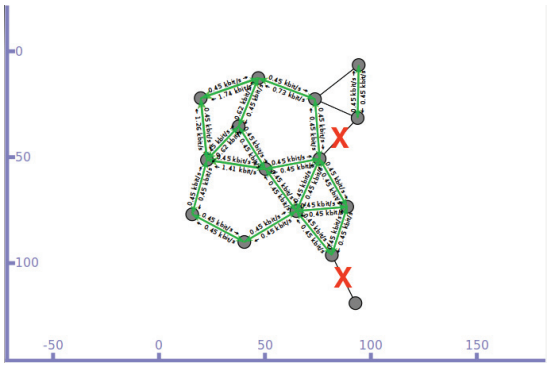
(a) Network layout



(b) Test network after the join



(c) Test network after link A break



(d) Test network after link B break

Figure 6: Test network node's communications.

Table 1: Simplified routing table and neighbors set, T=25s

Node	Rank	Destinations in the RT	Neighbors	Parents
1	1	4, 5, 6, 7, 8	4, 6	-
2	1	9, 11, 12, 14	11	-
3	1	10, 13, 15	15	-
4	1025	1, 5, 8	1, 5, 8	1
5	2049	4, 6	4, 6	4, (6)
6	1025	1, 5, 7	1, 5, 7	1
7	2049	6	6	6
8	2049	4	4	4
9	4	12	12	12
10	4	13	13	13
11	2	2, 9, 12, 14	2, 12	2
12	3	9, 11, 14	9, 11, 14	11
13	3	10, 15	10, 15	15
14	4	12	12	12
15	2	3, 10, 13	3, 13	3

join one of the three DODAGs exported by the root nodes (1, 2 and 3).

After a few seconds all the nodes have joined a DODAG. Figure 6b shows the RPL control traffic flowing into the network. Nodes belonging to the same DODAG are exchanging packets, while no traffic exist between nodes belonging to different DODAGs. The DODAGs are not easily recognized from this figure as the control traffic is also exchanged between neighbors with the same rank. The routing tables (Tables 1 and 2) can help on this.

At $T = 40s$ link A stops. Figure 6c shows the network structure after the nodes have detected root 3 unreachability and have joined one of the other DODAGs.

At $T = 80s$ also link B is disconnected. As shown in Figure 6d, the nodes that could no longer reach the DODAG root reorganized themselves. Root 2 still keeps one node in its DODAG.

5.3 Test results

The test described above has shown the behavior of the protocol implementation concerning the DODAGs creation and rebuilding capabilities after link failures.

Inspecting the routing table entries at the different nodes, along with the neighbor's sets, can better prove the behavior correctness. To achieve this goal, we printed each node's outing table and neighbors set with a period of five seconds. For convenience, we have reported in tables 1 and 2 just the two most significant: the one at $T = 25s$ and the $T = 125s$, i.e., two 'stable' points, after the transients due to DODAG's reorganization.

We can easily see how nodes with higher rank in a DODAG have all the nodes 'below' them in their routing table, along with their parents. In the neighbor's set, on the other hand, each node has all the one-hop nodes belonging to the same DODAG. We can see as well that some nodes have two parents, one of them being the preferred one and the other ones (in parentheses) kept as backup parents in case of link failure.

To further test the protocol, we used a large-scale network. The topology has been generated using a Poisson Spatial distribution with the RPL root in the center. The parameters are: Square area (1 Km x 1 Km) with root node in the center, 2000 nodes. coverage of each node 50m (ra-

Table 2: Simplified routing table and neighbors set, T=125s

Node	Rank	Destinations in the RT	Neighbors	Parents
1	1	4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15	4, 6	-
2	1	11	11	-
3	1			-
4	1025	1, 5, 8, 9, 12, 13, 14, 15	1, 5, 8	1
5	2049	4, 6, 9	4, 6, 9	4, (6)
6	1025	1, 5, 7, 9, 10, 12, 13, 14, 15	1, 5, 7, 9	1
7	2049	6, 10, 13	6, 10	6
8	2049	4, 12, 13, 14, 15	4, 12	4
9	2049	5, 6, 12, 13, 14, 15	5, 6, 12, 13	6
10	3073	7, 13	7, 13	7
11	2	2	2	2
12	3073	8, 9, 13, 14, 15	8, 9, 13, 14	8, (9)
13	3073	9, 10, 12, 14, 15	9, 10, 12, 14, 15	9
14	4097	12, 13, 15	12, 13, 15	12, (13)
15	4097	13, 14	13, 14	13

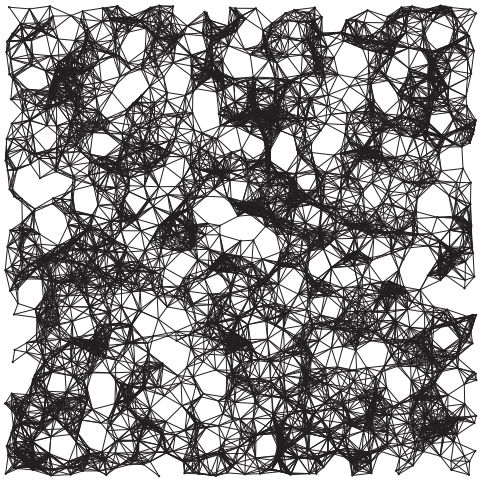


Figure 7: Large-scale network topology.

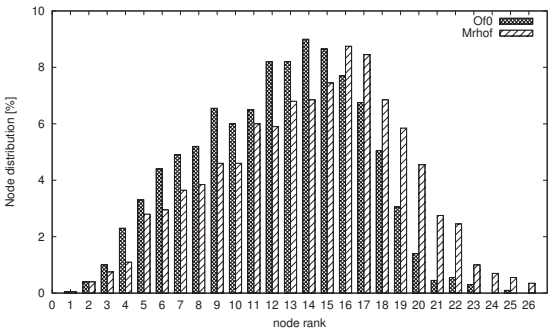


Figure 8: Node ranks with large-scale network.

dius), 14.967 links. The resulting layout is in Figure 7. The topology is fully connected.

The topology has been generated using a separate program to calculate the node positions and the available links. The TopologyRead ns-3 module has been used to setup the needed point to point links. The point to point is not the best choice to evaluate a real network, however it is sufficient to evaluate the routing protocol under ideal conditions (i.e., no packet loss, no congestion).

The simulations have been run for 500 seconds, with a simulation execution time of about 3 minutes (no code optimizations). After 50 seconds the network routing was stable and, due to the RPL metrics used, it did not change. For the test we used two different Objective Functions: OF0 and MRHOF, the latter with a HopCount metric.

The node ranks are shown in Figure 8. We can observe, as expected, that OF0 is more aggressive on finding the minimum hop count. A more detailed analysis (left to future work) should evaluate the behavior of the two Objective Functions with node mobility, i.e., when the topology stability is hindered by external factors.

6. NEXT STEPS AND FUTURE WORK

RPL is a new protocol, and its standardization is not yet complete. Many new extensions are possible and actively proposed in the ROLL working group. The ns-3 implementation is meant to both follow the standard changes and to be used to test new features to be proposed for standardization.

The module itself needs some code cleaning and has to be peer-reviewed in order to be included in ns-3, and we don't doubt that many improvement will be highlighted during this process. From this point of view, some improvements can be: a better integration with the NDIS cache, the wiser use of NS/ND mechanism and the use of Energy Module, just to name a few.

Some RPL features have yet to be implemented, for example the P2P special routing (used when no root is available) and some metrics, like the min-max ones.

Extending the protocol beyond the standard is also a possible future work, in order to test new objective functions and metrics to optimize the protocol performances in the cases not covered by the intended case studies, e.g., for high mobility cases.

Finally, another future development may be the realization of the missing MOPs so to compare the network performances in mixed environments where some nodes cannot store the routing table entries.

7. CONCLUSIONS

The ns-3 RPL module was a challenge we took with great pleasure. Its creation took quite a long time (about one year), mainly due to the authors' inability to be dedicated to it full time. It also needed a complete refactoring, that basically lead to a complete rewriting of a large part of the code, in order to eliminate some mistakes contained in the first design.

Overall, we are quite satisfied of the module, even if it is not yet complete and ready to be integrated in the ns-3 main repository. The main remaining blocking issues are related to the 802.15.4 module (still in progress) and the SixLowPAN one (still in progress as well), so we decided to hold on until those two will be available and stabilized enough to be used. Another ns-3 module that might be interesting to integrate is the Energy one, in order to fully support the energy-related metrics.

The module design and implementation gave us a deep understanding about RPL itself, way deeper than the one possible just by reading the standard or studying another implementation. Hence, it was more fulfilling both from the student's point of view and from the academic one.

The very next steps, now, are to test the protocol against a real implementation, in order to assess its conformance. We plan to use the Contiki's implementation.

Last but not least, we plan to use the developed module to go beyond the standard and, eventually, to submit our results and proposals to the IETF ROLL Working group.

8. ACKNOWLEDGMENTS

We thank Maroua Boumessouer of INRIA Nancy - GRAND EST (France) `maroua.boumessouer@inria.fr` for the constructive and helpful comments about the RPL implementation.

This paper has been developed as part of the IMPULSO Project. The IMPULSO project is focused on integrated metropolitan logistics and is co-financed by the Italian Ministry of Economic Development in the framework of the INDUSTRIA 2015 Program.

9. REFERENCES

[1] Castalia - A simulator for WSNs. http://castalia.npc.nicta.com.au.

[2] The Contiki OS - The Operating System for the Internet of Things. http://www.contiki-os.org.

[3] TinyOS. http://www.tinyos.net.

[4] A. Brandt, J. Buron, and G. Porcu. Home Automation Routing Requirements in Low-Power and Lossy Networks. RFC 5826, 2010.

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

[6] O. Gnawali and P. Levis. The Minimum Rank Objective Function with Hysteresis. draft-ietf-roll-minrank-hysteresis-of-04, 2011.

[7] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko. The Trickle Algorithm. RFC 6206, 2011.

[8] J. Martocci, P. D. Mil, N. Riou, and W. Vermeylen. Building Automation Routing Requirements in Low-Power and Lossy Networks. RFC 5867, 2010.

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

[10] Routing Over Low power and Lossy networks IETF WG. http://tools.ietf.org/wg/roll.

[11] P. Thubert. RPL Objective Function Zero. draft-ietf-roll-of0-20, 2011.

[12] J. Vasseur, M. Kim, K. Pister, N. Dejean, and D. Barthel. Routing Metrics used for Path Calculation in Low Power and Lossy Networks. draft-ietf-roll-routing-metrics-19, 2011.

[13] T. Winter, P. Thubert, A. Brandt, T. Clausen, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, and J. Vasseur. RPL: IPv6 Routing Protocol for Low power and Lossy Networks. draft-ietf-roll-rpl-19, 2011.