RPL Implementation for OMNeT++

Yevhenii Shudrenko

September 2020

1 Features overview

The list of implemented and prospected features of RPL are summarized in table below.

Table 1: RPL implementation status

Implemented	**TBD**
Control messages	
DIO, DAO	DIS
Modes of operation (MOP)	
• Storing • Non-storing	
Traffic flows	
• Multipoint-to-point (MP2P) • Point-to-multipoint (P2MP, storing only) • Point-to-point (P2P, storing only)	• P2MP (non-storing) • P2P (non-storing)
Additional features	
• Sub-DODAG poisoning • Loop detection	• Leaf nodes • Multiple RPL instances • DODAG, DAO repair • Floating DODAGs • Security

2 Showcase Scenarios

Below are the brief descriptions of simulation scenarios, showcasing a subset or particular feature of our RPL implementation. For all presented examples a typical UDP-based sensing application is taken as a baseline. Corresponding .ini configuration parameters are detailed in Table X.

Table 2: Simulation parameters

Parameter	**Value**
Communication range	150m
Distance between neighbors	$\tilde{1}$20m
UDP send interval	uniform(1s, 10s)
Payload	56B
Node 2 move speed (dynamic scenario)	2mps

2.1 Static Topology

Base scenario demonstrates RPL topology construction process in static topology, dissemination and processing of DIO, DAO messages as well as adaptive behavior of Trickle timer.

2.2 MP2P Traffic Flow Static

Following base scenario, this demonstrates MP2P traffic flow under static topology, typical for sensor networks without involved mobility. Data collection process is modelled using UDP basic client-server application. Nodes 1-5 (clients) are sending packets with sensor data payload at regular intervals to the sink node (server). Exact simulation parameters are provided in the table below.

2.3 MP2P Traffic Flow Dynamic

This scenario operates in the same MP2P communication model, but adds mobility to node 2 with the move speed of 2mps to incur following topology changes in RPL:

1. Upon leaving sink's communication range, node 2 loses connection to its only preferred parent, as shown in Fig. 3.
2. Because node 2 has no further candidate parents, it leaves the DODAG and clears corresponding internal state (rank, DODAG ID, RPL instance ID)
3. Node 2 can still hear messages from node 4 and eventually rejoins the former DODAG, but now as a child of node 4, which results in updated topology as shown in Fig. Y.

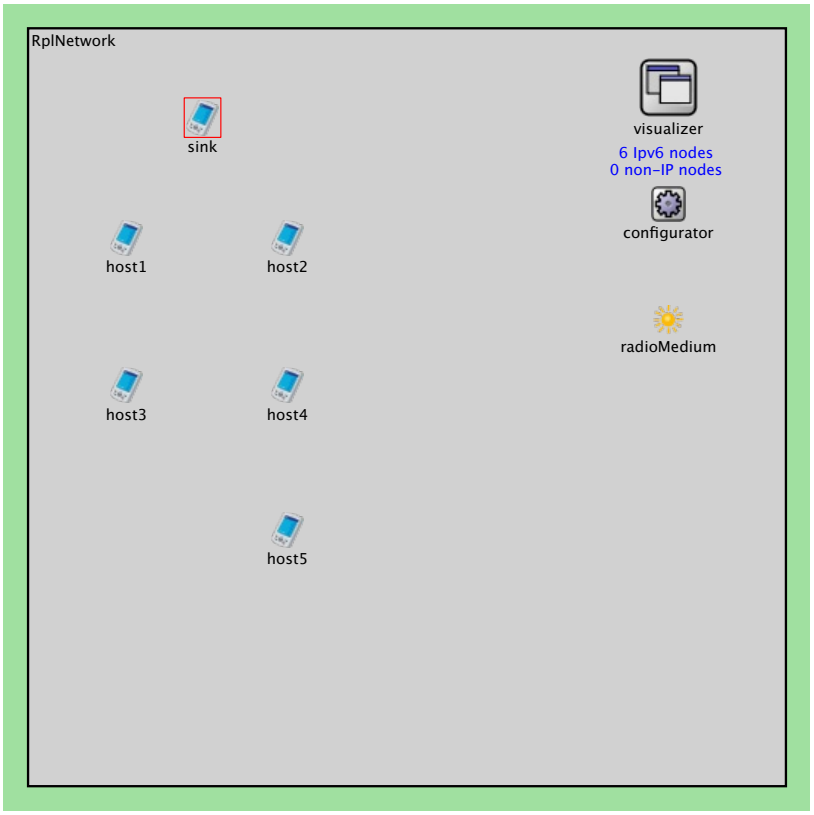

Figure 1: Static scenario topology overview

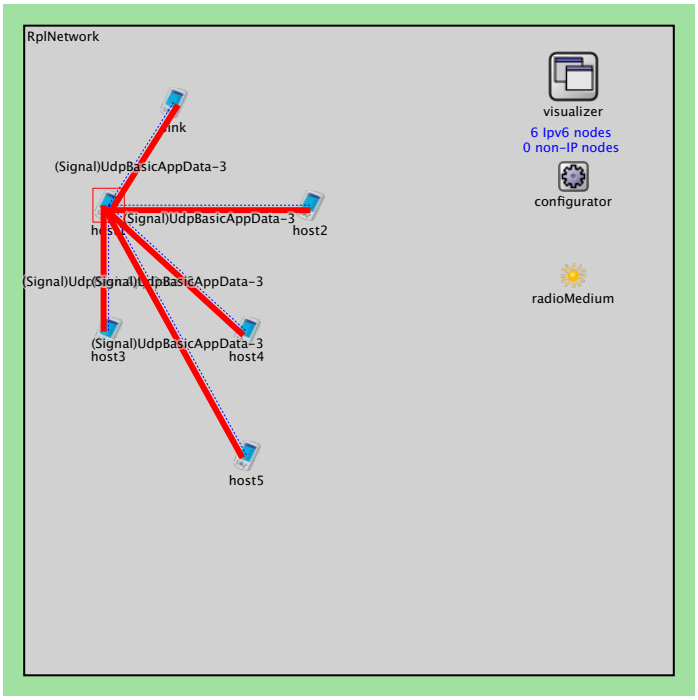

Figure 2: Sink gathering sensor data in multipoint-to-point traffic flow

2.4 Sub-DODAG Poisoning

One could notice in the result of previous scenario node 4 still keeping node 2 as its preferred parent, despite latter having detached from the DODAG. Such state will cause routing loops, where nodes 2 and 4 will 'bounce' packets at each other. To eradicate probability of such issues, *poisoning* mechanism is applied:

1. Upon leaving the DODAG node advertises an infinite rank in its final DIO, causing former children hearing this message to remove from this node from their parent sets
2. Node 4 thus removes node 2 as its preferred parent and tried to find a new one
3. Considering node 4 has no candidate parents, it also leaves the DODAG, but remains stationary
4. Upon receiving DIO from node 3, node 4 rejoins the DODAG at a higher rank and starts itelf broadcasting DIOs
5. Eventually node 2 hears a DIO from node 4 and also rejoins the same DODAG with an updated rank

In this scenario poisoning functionality is enabled on node 2, which results in a proper topology end-state as shown in Fig. 5.

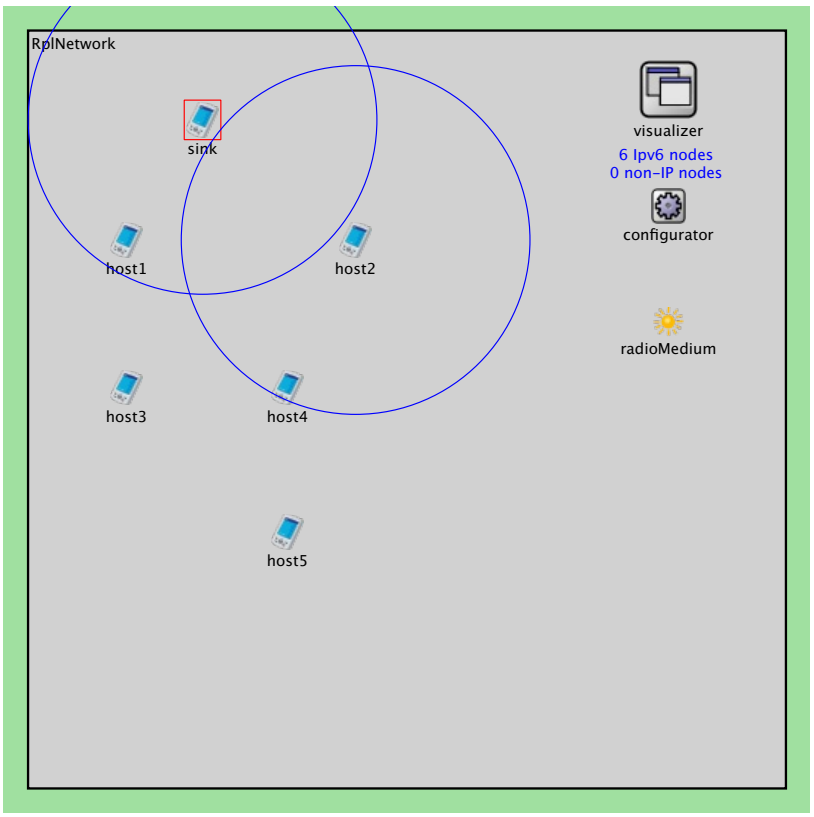

Figure 3: Node 2 moving out of its preferred parent (sink) communication range

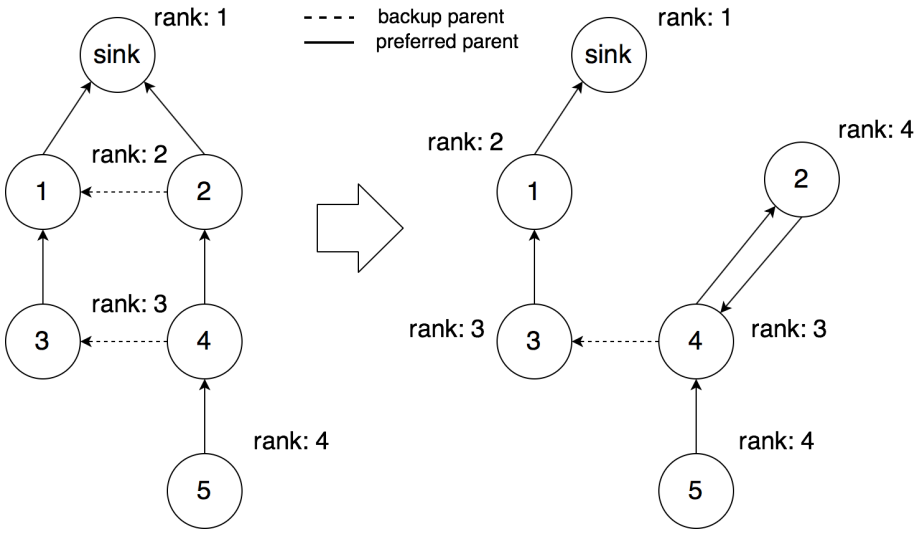

Figure 4: Topology changes in RPL instance incurred by node 2's mobility

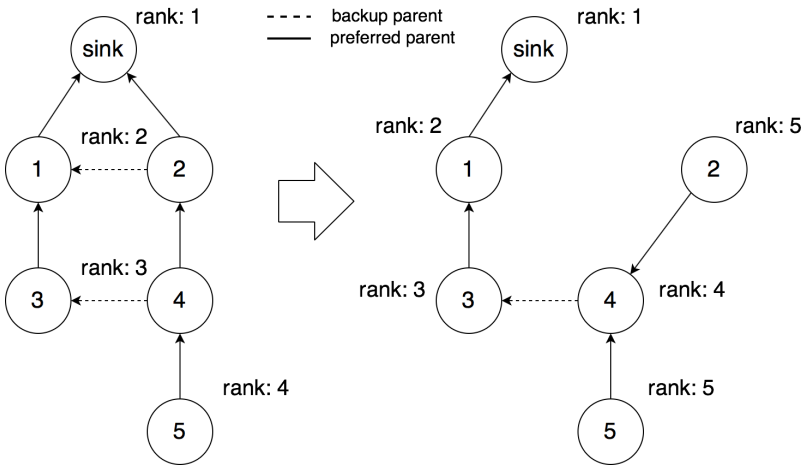

Figure 5: Topology changes in RPL instance incurred by node 2's mobility with enabled poisoning

2.5 P2MP Traffic Flow

This use-case serves as a basic example of actuator control data flow in point-to-multipoint fashion, under storing mode. Upon joining a DODAG nodes start unicasting DAO messages announcing their presence as possible destinations to the sink. Having collected DAO from all nodes, sink has a complete topology overview and starts sending actuator data to individual nodes. As they operate in storing mode, each node learns destinations from it's own sub-DODAG and is able to route packets on its own, as visualized in Fig. 6.

2.6 P2P Traffic Flow

Extending the previous P2MP scenario, nodes can unicast messages to other nodes within the DODAG. This message travels upwards in the topology tree until a common ancestor is reached and then routed downwards to destination, as shown in Fig. 7 for the storing mode of operation.

2.7 Loop Detection

To detect and repair loops RPL performs several checks on RPL Packet Information header (RPI), appended to each application packet. Among other fields RPI contains:

- Sender rank
- 'O' flag indicating whether packet progresses down (1) or up (0)
- 'F' flag signaling *forwarding* error
- 'R' flag signaling *rank* error

Forwarding error

Forwarding error occurs only in storing mode, when node cannot find a route for the packet with 'O' flag set, indicating downwards direction. Upon setting the flag, the packet is returned to the sender, clearing faulty or outdated DAO routes.

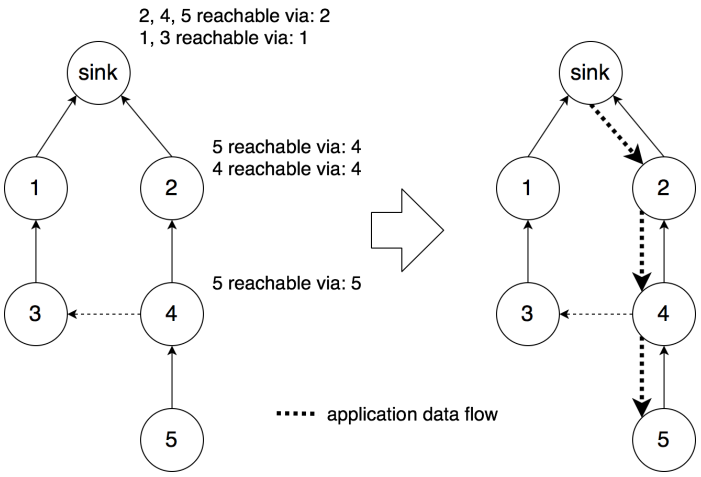

Figure 6: Application data forwarding in downwards direction

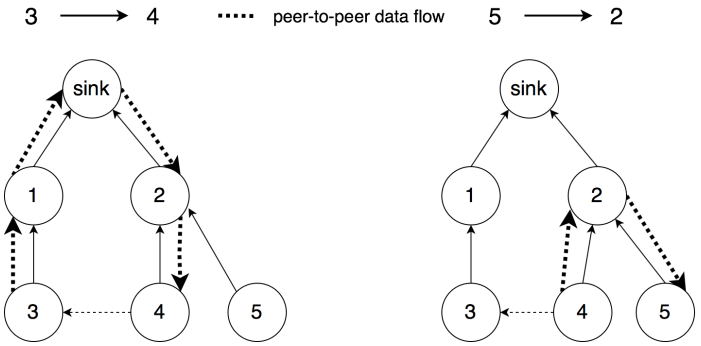

Figure 7: Application data forwarding in peer-to-peer communication

In first loop detection scenario, by combining dynamic part of 2.3 and P2MP traffic flow from 2.5 forwarding error is induced to trigger loop detection mechanism as follows:

1. Node 2 leaves the sink's communication range in the same manner as depicted in Fig. 3
2. Node 2 detaches from the DODAG due to empty candidate parent set and clears DAO routes it learned alongside RPL state
3. Sink still keeps an entry in its routing table for node 5 being reachable via node 2
4. Node 2 rejoins the DODAG as a child of node 4
5. Node 2 moves back and appears within sink's communication range again
6. Sink tries to send a packet to node 5 via node 2
7. Node 2 receives a packet with 'O' flag set, but doesn't have an entry in its routing table for the node 5 anymore resulting in a forwarding error flag set

Rank error

Rank error occurs whenever node fails to verify expected relationship between its own rank and that of a packet sender. Precisely, if either of the below conditions resolves to true, a possible inconsistency\loop is detected:

- 'O' flag is set (packet travels down) but senderRank $>$ rank
- 'O' flag is not set (packet travels up) but senderRank $<$ rank

These scenarios are recreacted based on the 2.3 case, where a loop between nodes 2 and 4 is formed due to the absence of poisoning DIO from node 2. Indeed, after the loop is formed as depicted in Fig. 4, a packet to the sink originating at node 2 will be forwarded to node 4 first. Then node 4 tries to pass the packet further 'upwards' by sending it back to node 2. When latter performs rank test as described above, the second condition will set rank error flag and trigger loop detection mechanism correspondingly.