Experimental Comparison of Routing Protocols for Wireless Sensors Networks: Routing Overhead and Asymmetric Links

Henry-Joseph Audéoud and Martin Heusse
Univ. Grenoble Alpes, CNRS, Grenoble INP*, LIG, F-38000 Grenoble France
*Institute of Engineering Univ. Grenoble Alpes
Email: [henry-joseph.audeoud,martin.heusse]@univ-grenoble-alpes.fr

***Abstract*—RPL (the IETF Routing Protocol for Low-Power and Lossy Networks) and LRP (Lightweight Routing Protocol) have in common to build a collection tree (or, more precisely, a DODAG) and "downward" host routes in the wireless sensor network. Additionally, the objective of LRP is to keep control overhead as low as possible. To substantiate this claim, we compare RPL and LRP using 40 nodes of the IoT-LAB testbed — and the results are telling.**

We then introduce asymmetric links, which are unavoidable in most deployments, and measure their impact on the considered protocols in another set of experiments. We present the mechanisms that were introduced in LRP to deal with such cases and we report on extensive experiments involving RPL and LRP to analyze the behavior of both protocols when the links change or they exhibit asymmetry.

I. Introduction

RPL [1] has attained a mature state and its implementation in the Contiki OS (from version 3.0) conforms to the protocol specification. The implementation enables us to perform evaluations on the IoT-LAB[1] wireless sensor testbed to gain insights on the performance of various protocols with real wireless nodes, using the same code that could be used in field deployments and with much more realistic conditions than in simulation.

RPL is relatively generic and tackles many problems whereas it is often criticized for its complexity [2] and the amount of generated traffic [3]. On the contrary, Lightweight Routing Protocol (LRP) [4], [5] aims at generating as little traffic as possible while remaining simple. Both protocols consider the case of sensor networks in which the sensors need to send out data packets to the sink with limited topology changes at the time scale of the packet generation period; otherwise, a reactive approach would be more suitable [6], [7]. Also, the nodes should be accessible, so the routing protocol has to build "downward" routes. These are fairly standard objectives matching the assumptions behind the design of RPL, but also LOADng-CTP [8], [9]. This latter proposes to add a proactive collection tree creation mechanism to LOADng, precisely to address the situation in which multipoint to point (from nodes to the sink) traffic is salient. However, both RPL and LRP concurrently handles host and default routes whereas LOADng-CTP only deals with disjoint routes. Even if it creates default routes, we recall that LRP guarantees the routing is loop-free at all times without the need for piggybacking routing data in each data packet, as in RPL.

Based on our experience with LRP, we propose improvements (*cf.* Section IV) to further reduce the routing overhead, in particular when establishing (IV-A) and maintaining (IV-B) the collection tree or when maintaining the host routes (IV-C), as well as in the presence of unidirectional links (IV-D).

As expected, we find that, on a relatively large scale testbed with tens of nodes, both LRP and RPL manage to build similar collection trees, although the control overhead of LRP is only a fraction of that of RPL and even more so with the improvements presented in this paper.

We then turn to more specific and challenging situations with extensive experiments involving RPL and LRP on IoT-LAB with link quality changes or asymmetric links. We create such conditions by varying the transmission power and sensitivity of some nodes to assess the robustness of both protocols to the sort of disturbances that are unavoidable in real world deployments.

The paper starts with a short outline of RPL and LRP in Section II and III. Section IV describes the improvements introduced to further reduce the amount of control traffic with LRP. Section V analyzes the overhead incurred by each protocol whereas Section VI, focuses on the impact of asymmetric links.

II. Background on RPL

RPL (Routing Protocol for Low power and Lossy Networks) [1] is the standardized routing protocol for multi-hop sensor networks.

RPL builds a set of default routes forming a collection tree (more precisely a DODAG — Destination Oriented Directed Acyclic Graph, as several equivalent successors may be used concurrently) by means of a distributed Bellman Ford algorithm: a node select a default next hop (or parent, or successor) amongst its neighbors, based on the information they send in their broadcast DODAG Information Object messages (DIO).

[1] https://www.iot-lab.info

978-0-9883-0453-6/17 $31.00 © 2017 ITC
DOI 10.1109/ITC.2017.50

RPL is built upon the trickle mechanism [10] to spread DIO control messages throughout the network. Consequently, since DIOs may be sent relatively rarely, a node can probe its neighborhood by broadcasting a DODAG Information Solicitation message (DIS), causing the neighbor nodes to send DIO messages back.

The host routes are created and maintained by the end nodes, by sending regularly a Destination Advertisement Object message (DAO) to the preferred parent. The information contained in the DAO is relayed up along the DODAG, which creates the host route incrementally.

RPL is known to generate a large number of packets when not tuned properly [11], [12], in part due to a constant instability of the preferred parent selection [13]. In fact, the protocol is in many respects overly complex, underspecified, and heavy as it adds overhead for each conveyed data packet while the RPL control packets themselves are excessively bulky. Moreover, one of the major shortcomings of RPL is that it does not allow the sink to request a route towards a specific node in the network [14].

Several papers compare RPL to LOADng, a lightweight reactive routing protocol for sensor networks, by simulations, and their conclusions are contradictory [7], [15], [16]. These conflicting results are not surprising, since the choice between the proactive or reactive behavior is primarily dictated by the network density and size, but also by the traffic pattern. So, the problem lies more in properly assessing the use case and then using the relevant approach.

III. Background on LRP

The sections below describe the current LRP's collection tree and host routes creation and maintenance mechanisms which appear with more details in a previous paper [5].

A. Collection tree

Just like RPL, LRP maintains a collection tree (or a DODAG when the nodes keep track of several successors concurrently) to extract traffic out of the network through the sink. The positions in the collection tree are computed using the same distributed Bellman-Ford algorithm. However, in contrast with RPL, a node broadcasts its position by sending a DIO message only when it changes its position or in response to a DIS solicitation. The successive algorithm executions supersede each other each time the sink originates a new sequence number to refresh the collection tree; this is the same global repair as in RPL.

To ensure there are no loops into the network, LRP imposes that a node must never accept a route that would put it further away from the sink than its current position, for a given sequence number, which is incremented at each global repair. In complement, a local repair mechanism allows nodes to reattach further down at any time; the procedure has a relatively low footprint, but there is no guarantee that all packets follow their shortest path to the sink anymore.

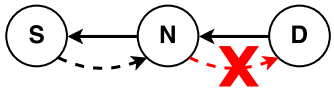


Figure 1: Example of a routing loop due to the coexistence of default and host routes. N deleted the host route to D (dashed arrows); there is now a routing loop between N and S as the packets destined to D now match only the default route in N and thus go back to S.

B. Host routes to the nodes in the network

LRP also allows the nodes to establish host routes to distribute traffic into the network. Host routes may be initiated by the hosts themselves (proactively), or at the sink's request (reactively).

In the proactive case, the node D advertises itself by sending a RREP (Route REPly) message to its successor. This message is forwarded towards the sink and the nodes along the way record the corresponding host route which eventually goes from the sink to D.

To find the node D in the network, either because the route was lost, expired, or it was not (successfully) proactively advertised by the destination, the sink floods the network with a RREQ (Route REQuest) message, containing D's address. When D receives it, it answers with a RREP message, as in the proactive case.

C. Data path validation

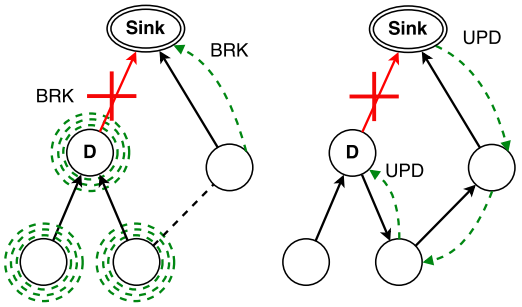
The coexistence of default routes and host routes may generate routing loops, even if the routing DODAG is effectively loop free at all time. For instance (see Figure 1), if an intermediate node N deletes a host route to D and receives a packet destined to D from a successor, this packet now matches only the default route and this node would resend it up to a successor.

To prevent the occurrence of loops, LRP has an implicit data path validation mechanism that detects routing loops whenever they are used. The situation is then solved by sending a RERR (Route ERRor) message that backtracks and removes the host route [5].

D. Local Repair: Tree Maintenance Algorithm

The local repair mechanism allows nodes to re-establish the collection tree without incurring a global repair. It is activated whenever a node loses all successors and thus has to turn to using a node in its own sub-tree as successor. The local repair comprises to steps (see Figure 2):

1) **Finding an alternate route to the sink:** A node D that can no longer use any neighbor as a successor broadcasts a BRK (BRoKen) message. This message is flooded by all the nodes into the sub-tree of D. Out of the sub-tree, the message makes its way as a unicast to the sink using the default routes.



(a) Finding an alternate route (b) Updating the collection tree

Figure 2: Example of a local repair. After node D asses that it has no more successor towards the sink, it floods the subtree with a BRK message to seek an alternative route to the sink (a) and then the collection tree is rebuilt (b).

2) **Updating the collection tree without creating a loop:** The sink then perform the actual local repair by sending a UPD (UPDate) message to D. To reach its destination, the UPD message backtracks the BRK message. Each intermediate node only keeps track of the best received BRK if it sees more than one. When a node receives an UPD message, it switches to using the UPD sender as successor.

This mechanism repairs the collection tree without any (even transient) routing loop.

IV. Reducing LRP control traffic

In this section, we detail the improvements to LRP that we introduced to reduce routing overhead in the general case as well as to appropriately deal with unidirectional links.

A. Speed up DIO transmission

Whenever a node S receives a DIO message coming from a lower neighbor N publishing a distance to the sink which is larger than S's plus the cost of link NS, then S may spontaneously send a (unicast) DIO back to N, in order to allow this latter move up the collection tree.

This mechanism allows N to easily probe its neighborhood by publishing its own distance to the sink in a DIO. It will receive replies only from neighbors that it may use as successor.

Indeed, S will not reply to N's DIO if its published distance to the sink is *the same* than it could have by using S as successor. In the special case where N is looking for an alternative successor, it may use the DIO option `DETECT_ALL_SUCCESSORS` to explicitly require that all possible successors respond with their position.

B. Reduce BRK flooding footprint

The original LRP local repair mechanism prompted the flooding of a BRK in the whole sub-tree below the detached node (*cf.* Section III-D), whereas in many cases the repair routes could be found through a not so distant predecessor. We add an expanding ring search scheme to reduce the number of unnecessary BRK broadcasts in this situation.

We add a `ring_size` field to the BRK messages, decremented each time it passes a node. In this way, the originator of the BRK controls the depth at which other routes to the sink will be sought. In the worst case, if the closest nodes cannot be used to repair the collection tree, nothing stops the detached node to set then ring size to a larger value. Note that outside of the sub-tree of the BRK originator, the BRK messages are sent unicast on the default route to the sink; the ring size limit is not taken into account anymore.

C. Host routes repair

A modification in the network topology does not only break the collection tree (and trigger a local repair) but also breaks all the host routes of the corresponding sub-tree. After having detected that host routes are broken (*cf.* Section III-C), the sink will reactively look for their destinations by flooding several RREQ into the whole network (*cf.* Section III-B). This may create a significant broadcast traffic that we want to avoid.

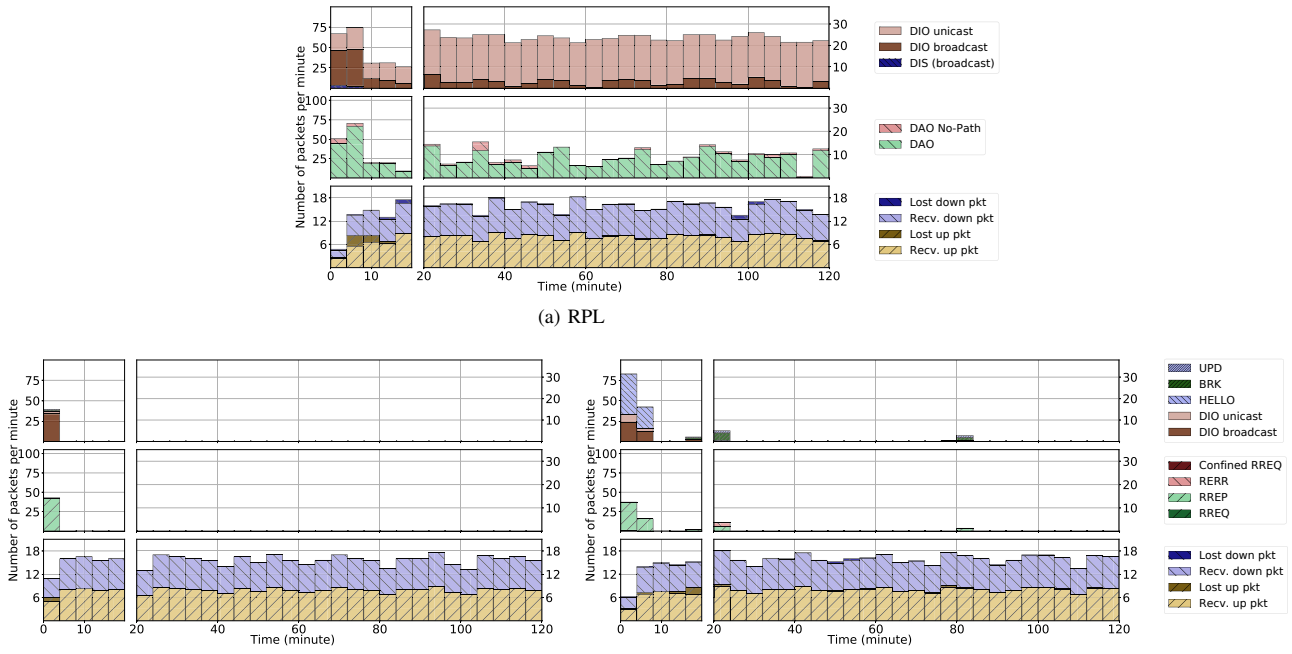
When the local repair finishes, all the repaired sub-tree is reattached to a node N. We know that N is the new point of attachment to the original tree because it receives the UPD repair message from a neighbor that was neither its successor nor its predecessor. We propose that N broadcasts a confined RREQ when it finishes a local repair. This special RREQ message should be flooded only in the sub-tree of N; if a node does not receive it from its successor, it drops this message without processing it. The confined RREQ does not target a specific address; rather, it means that the host route of any node R that receives this message should be rebuilt. R refreshes its host route by sending a RREP to the sink with a new sequence number (it will readily replace the old host route). It also re-broadcasts the confined RREQ, to forward it to its predecessors.

D. Handling unidirectional links

In LRP, a node N elects to use a tentative successor T after receiving a DIO with a promising distance to the sink. Nevertheless, the reception of such a DIO only shows that T-to-N communication may be possible, but not in the reverse direction, from N to T.

To assess bidirectional link quality, we propose that N first attempts a unicast HELLO handshake with T. If it is successful, N starts using T as a successor. As HELLO messages contain the link cost as seen by the sender node, N is able to compute the cost of the link by taking both directions into account. It will revise this cost if numerous link layer losses subsequently happen. Otherwise, if the HELLO handshake fails, N notes that the link is suspicious and looks for other options. Each node keeps track of its unresponsive neighbors and blacklists them for some time, even when receiving new DIOs from them.

This procedure is more lightweight than in LOADng-CTP [8], where all links are checked to be bidirectional beforehand.



(a) RPL

(b) o-LRP

(c) LRP

Figure 3: 2-hour experiments with 40 client nodes and 1 sink. All along the experiment, RPL sends a relatively high number of messages for collection tree (top plots) and host route (middle plots) maintenance. At the start of the experiment, the number of messages is similar for both versions of LRP (b)(c), but they are of a different nature. In the original LRP, the links are used without been checked. Hopefully, in this case, no link as revealed to be faulty. The improved LRP (c) detects them during the HELLO handshake which adds a bit of unicast traffic. However, as soon as the routes stabilize, both versions of LRP send no routing traffic, except when some link is finally considered unusable (*e.g.* after 80 min. of experimentation in (c)).

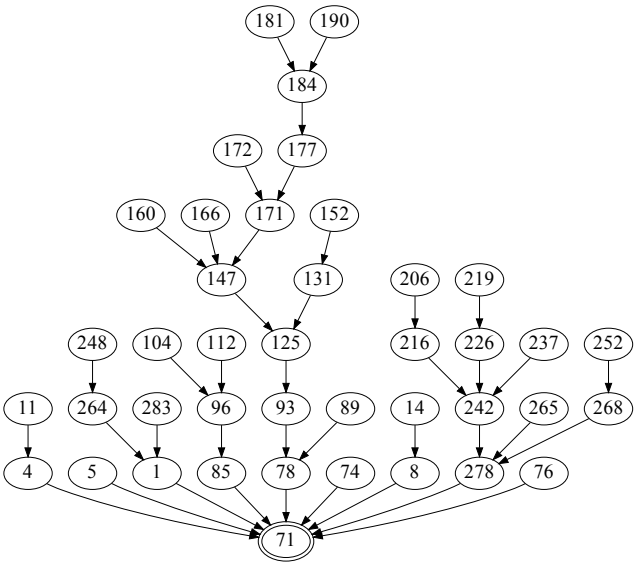


Figure 4: Final collection tree for the experiment described in Section V-B. 71 is the sink. The trees are similar with RPL, o-LRP and LRP.

Table I: Number of messages sent during the whole experiments described in Section V-B and Figure 3.

		RPL	o-LRP	LRP
Collection-tree maintenance	Broadcast	786	136	155
	Unicast	2497	22	399
Host routes maintenance	Broadcast	n/a	0	8
	Unicast	1586	172	233
Data packets successfully routed		97.2%	98.9%	98.0%

Table II: Parameters of RPL and LRP

Radio duty cycling	Contiki MAC
Check interval	125 ms
RPL	
I_{min}	4 s
I_{max}	1048 s
DIO redundancy	10
DAO re-generation period	[15-22] min.
Objective function	MRHOF with ETX
LRP	
Random delay for packet transmission following a broadcast	0.5 s
Frequency of neighborhood probing when disconnected	∼5 min.
Blacklist timeout	10 min.
Metric type	Hop count
# UDP application packet (per client)	1 every 5 min.

V. Protocol overhead

In this section, we discuss the overhead of the RPL and LRP with or without the above improvements.

A. Per-packet overhead

For data path validation purposes, RPL mandates to add a RPL header to each data packet, which adds at least 8 bytes to each packet. It is not desirable in networks in which this header represents as much as 7.8% of the MAC Service Data Unit size (102B in IEEE 802.15.4).

Even worse, for packets coming from outside of the sensor network, adding an additional Hop by Hop header incurs adding an outer IPv6 header to tunnel the external packet into the RPL domain. LRP does not need to attach any information to data packets.

B. Protocol background traffic

RPL uses trickle to reduce routing traffic when the topology does not change. The benefits of trickle over the long term are obvious: less and less packets are sent. On the other hand, before the timers reach a large value, nodes send redundant information several times with little benefit. This is particularly costly for link layers like ContikiMAC [17] in which the energy cost of broadcasts is typically an order of magnitude greater than that of unicasts.

In RPL, DAO are also blindly retransmitted at regular intervals to refresh host routes from the sink. The period of DAO renewal is a tradeoff between the amount of control traffic they generate and the time during which a node may remain unreachable. In fact, if a route is erased on an intermediate node, the sink can only request all nodes in a sub-tree to resend their DAOs (by means of increasing the DTSN), or patiently wait for the node to refresh its DAO.

In LRP, all updates follow the detection of an inconsistency: the sink can trigger a global repair and the nodes take action when they loose their successors or when a host route appears broken. The only recurrent traffic from nodes is to probe their blacklisted neighbors or when detached nodes send infinite distance DIO to discover their environment.

Figure 3 and 4 show an experiment run on the IoT-LAB. During two hours, 40 client nodes send UDP data packets up to the sink with a period of 5 minutes, and the sink replies to each packet.

Table I gives the numbers of packets exchanged during these experiments. We note that both RPL, the original version of LRP (denoted *o-LRP*, *i.e.* before the improvements described in Section IV) and the improved version of LRP (denoted *LRP*) successfully route almost all data packets. However, RPL uses a lot more routing packets than LRP: if we consider that a broadcast packet occupies the radio channel 10 times as much as unicast packets (which is inherent to Contiki MAC or most preamble sampling duty cycling schemes), we find that the RPL radio channel occupancy is 530% that of the improved version of LRP and 225% that of the original version of LRP in this case. With the improvements, the LRP radio channel occupancy is 42% that of the original version.

Had the experiment lasted longer, the difference would only widen: from the two top plots of Figure 3 it is clear that both versions of LRP are mostly continuously quiet, whereas RPL creates a constant background traffic.

VI. Experiments with asymmetric and muted links

The quality of radio transmissions depends on the environment and on the radio hardware. Key environmental factors are ambient noise, physical obstacles that cause reflections and absorption, and interferences which generally do not have the same impact on all nodes. The radio hardware impacts the transmission when the gains and sensitivity varies between devices; these effects play an important role in wireless sensor networks where devices are low cost and the environment may be crowded.

In this situation, a link between two nodes may be asymmetric, in other words, it may not present the same packet delivery ratios (PDR) in both directions [18], [19]. In extreme cases, a node N may be **"deaf"** (*i.e.* other nodes receive N's packets, but N does not receive anything back) or **"muted"** (*i.e.* N receives properly, but is not heard). This section describes the experiments to analyze how RPL and LRP handle these situations.

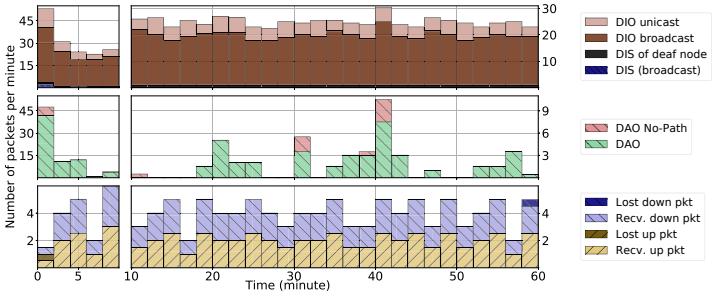
Table II presents the parameters used in the experiments. The trickle settings match the recommended practice, with I_{min} more than one order of magnitude greater than the broadcast duration (125 ms in ContikiMAC [17]). This parameter effectively avoids DIO collisions in our experiments with RPL. For the same reason, we randomize the transmissions of DIOs, BRK, HELLO, and RREQ in LRP.

A. Deaf nodes

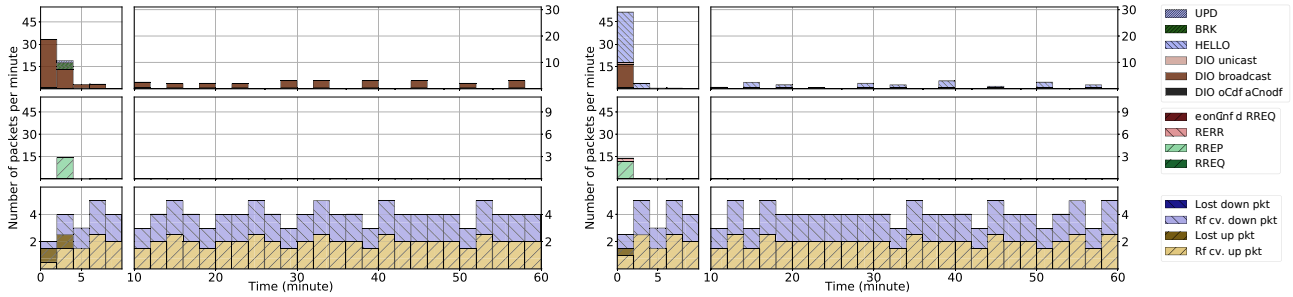
On the IoT-LAB testbed, we start 11 client nodes and a sink. As in Section V-B, each client sends one UDP packet every 5 minutes to the sink which replies with another UDP packet. One of the client nodes is deaf: its sensitivity is lowered and it does not receive any message from other nodes, while its packets are received by its neighbors. Figure 5 presents the results.

1) Deaf node in a RPL network: When a RPL node is not associated with any DODAG, it broadcasts a DIS message. This message resets the trickle timer that schedules the DIO message emissions in its neighborhood, thus generating many DIO broadcasts. Although this mechanism is useful to speed up the insertion of new nodes into the network, it is very harmful when a node N is deaf. Indeed, as N keeps on broadcasting DIS packets, its neighbors constantly send back DIOs. During the whole experiment depicted in Figure 5a, DIOs are constantly broadcast at the average rate of 1 every 3 seconds.

2) Deaf node in a LRP network: The original version of LRP (Figure 5b) makes all the neighbors of a deaf node broadcast a DIO message whenever the latter probes its

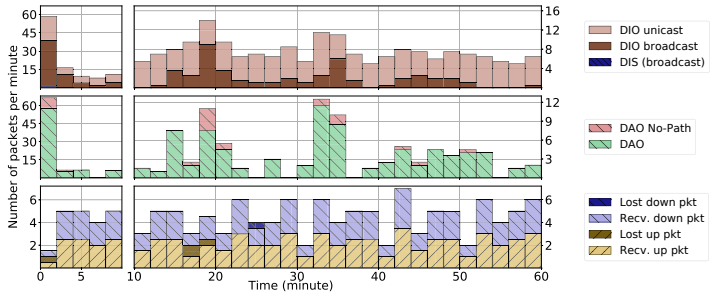


(a) RPL

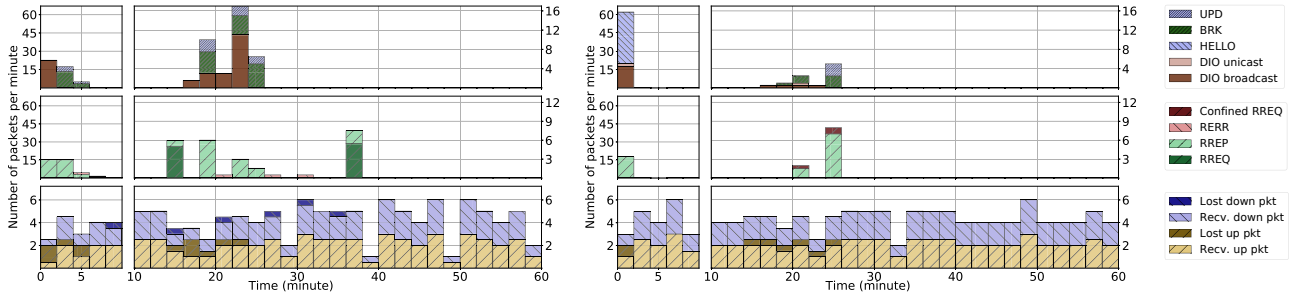


(b) o-LRP (c) LRP

Figure 5: 1-hour experiments of 11 client nodes and 1 sink. One of the client nodes is deaf. During the experiment, RPL is constantly sending broadcast DIOs (approximately 1 DIO every 3 seconds). o-LRP (the original version) sends broadcast DIOs whenever the deaf node probes its vicinity. The improved version of LRP sends parsimoniously unicast HELLOs.



(a) RPL



(b) o-LRP (c) LRP

Figure 6: 1-hour experiments with 11 client nodes and 1 sink. After 15 min., node 40 (*cf.* Figure 7) is muted. In all cases, some minutes later, connectivity is restored; however, RPL uses a lot more messages than LRP. We see that o-LRP (the original version) does not repair all host routes: some of them must be deleted and rebuilt (RERR messages and then RREQ floods and RREP messages, between 27 and 37 min.). With improvements, LRP directly rebuilds the host routes.

vicinity. This leads to an amount of 10 broadcast every 5 minutes in this setup.

With improved LRP, during the experiment depicted in Figure 5c, the nodes also answer to the deaf node. However, we notice two differences with the original version: first, the answers consist of HELLO unicast messages; indeed, before communicating with the deaf node, each neighbor probes the link to it. Secondly, the neighbors do not systematically answer to the deaf node probes. There is eventually less HELLO answers in improved LRP than DIO answers in the original one.

Compared to RPL, this amounts to sending one unicast packet from each neighbor every 10 minutes instead of 1 broadcast packet every 3 seconds.

B. Muted nodes

To test the behavior of both protocols to muted nodes, we use the same kind of experiments as for deaf nodes: a 1-hour experiment on IoT-LAB, where each of the 11 clients nodes send one UDP packet to the sink every 5 minute; the sink answers with another UDP packet to each of them (there are no deaf nodes here). Obviously, if we completely mute one node from the start, it will not disturb its neighbors much as all its messages would be lost. So we start node 40 (*cf.* Figure 7) as a normal node and let it associate itself with the network and communicate with other nodes. Then, after 15 minutes, we reduce its transmission power: it is muted and cannot reach its former neighbors anymore — excepted one of them, node 33, in order to not to be totally isolated from the rest of the network.

Figure 6 shows the messages exchanged during the experiment. The collection trees of both RPL and LRP are approximately the same and they are presented in Figure 7.

1) Muted nodes in a RPL network: Even if RPL limits the count to infinity process by defining a maximum rank increase, this creates a transient loop in the network (40 uses 33 as a successor, which in turn uses 40 as a successor — note that the traffic does not loop around, thanks to the data path validation header) until node 33 is able to use 55 as a successor. All this generates extra DIO traffic between minutes 18 and 21 of the experiment (visible in Figure 6a). This experiment shows that RPL eventually handles this situation correctly: few data packets are lost and the DODAG is repaired quickly.

2) Muted nodes in a LRP network: In both versions, LRP nodes use a local repair algorithm (described in Section III-D) to solve this situation. However, in contrast with the improved version, the original one does not repair host routes as part of the local repair, which leads to RREQ flood and RREPs later on (at ~ 37 min.).

After node 40 is muted, the local repair algorithm is not launched immediately because of the delay in the neighbor unreachability detection algorithm.

In the three experiments, the connectivity is restored within minutes. However, as shown in Table III, the improved LRP uses one fifth of the broadcast messages that either RPL or the original LRP use, and few unicast messages (9 times less than RPL, 2 times less than the original LRP). This is another example of the huge background traffic of RPL described in Section V-B and the reduced traffic of LRP.

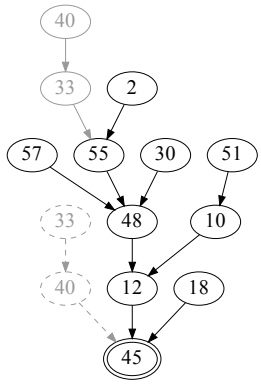


Figure 7: Typical collection tree of the experiments depicted in Figure 6. Node 45 is the sink. During the initial collection tree construction, nodes 40 and 33 (dashed) are connected directly to it. After node 40 is muted, the two nodes can only re-attach to the network through node 55. Note that the link between them is reversed.

Table III: Number of control messages sent during the experiment depicted in Figure 6, between 15 and 35 min.

	RPL		o-LRP	LRP
Broadcast	DIO	66	DIO	37	4
			BRK	7	5
			RREQ	11	3
Unicast	DIO	146	DIO	n/a	1
			BRK	20	4
			UPD	12	5
			HELLO	n/a	2
	DAO No-Path	15	RERR	3	-
	DAO	95	RREP	25	17

VII. Conclusion

With the improvements described above, LRP is still much simpler than RPL with approximately 3000 lines of code in Contiki *vs.* well above 5000 for RPL. The general approach of LRP is to build routes proactively and then repair them quickly as soon as an inconsistency emerges. In contrast, RPL sticks to its proactive philosophy and generates a constant and significat background control traffic. Even worse, a loss of DAO may cause a node to remain unreachable for a long time, due to the tradeoff between an intense DAO traffic and refreshing DAOs with much delay.

We note that, for the amount of control traffic RPL generates in a static network, LRP could perform a local repair every 5 minutes or more. This result holds as long as there is no deaf node in the network. Otherwise, RPL traffic is off the scale.

Finally, although no global repair takes place in the experiments described in this paper, we stress that an LRP sink has all the information it needs to decide if a global repair

is necessary or not, depending on the number of local repairs performed since the last global repair. In contrast, in RPL, the sink has little information about the changes happening in the network and thus, it is left triggering global repairs on a regular basis.

There are many directions in which to further develop LRP and we intend to explore more advanced metrics for LRP as well as load balancing schemes, thus making better use of the DODAG structure, similarly to what is done in the RPL context [20], [21]. Another idea is to explore the case of distributing several external routes with various prefix lengths. Even though the design philosophy of LRP is to make it as silent as possible, it can also be viewed as a sound and low overhead base for more elaborate purposes.

VIII. Acknowledgments

This work has been partially supported by the French Ministry of Research projects DataTweet under contract ANR-13-INFR-0008-01 and the PERSYVAL-Lab under contract ANR-11-LABX-0025-01.

References

[1] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J.P. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low power and Lossy Networks," RFC 6550, IETF, Mar. 2012.

[2] Thomas Clausen, Ulrich Herberg, and Matthias Philipp, "A critical evaluation of the IPv6 routing protocol for low power and lossy networks (RPL)," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on*. IEEE, 2011, pp. 365–372.

[3] Timofei Istomin, Csaba Kiraly, and Gian Pietro Picco, "Is RPL ready for actuation? A comparative evaluation in a smart city scenario," in *European Conference on Wireless Sensor Networks*. Springer, 2015, pp. 291–299.

[4] C-A. La, Martin Heusse, and Andrzej Duda, "Link Reversal and Reactive Routing in Low Power and Lossy Networks," in *Proceedings of PIMRC'13*, London, UK, June 2013, IEEE.

[5] Henry-Joseph Audéoud, Michał Król, Martin Heusse, and Andrzej Duda, "Low overhead Loop-free Routing in Wireless Sensor Networks," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2015 IEEE 11th International Conference on*. IEEE, 2015, pp. 443–451.

[6] Floriano De Rango, Juan-Carlos Cano, Marco Fotino, Carlos Calafate, Pietro Manzoni, and Salvatore Marano, "OLSR vs DSR: A comparative analysis of proactive and reactive mechanisms from an energetic point of view in wireless ad hoc networks," *Computer Communications*, vol. 31, no. 16, pp. 3843–3854, 2008.

[7] Joydeep Tripathi, Jaudelice C. De Oliveira, and Jean-Philippe Vasseur, "Proactive versus reactive routing in low power and lossy networks: Performance analysis and scalability improvements," *Ad Hoc Networks*, vol. 23, pp. 121–144, 2014.

[8] Jiazi Yi and Thomas Clausen, "Collection Tree Extension of Reactive Routing Protocol for Low-Power and Lossy Networks," *International Journal of Distributed Sensor Networks*, vol. 2014, pp. 1–12, 2014.

[9] Saida Elyengui, Riadh Bouhouchi, and Tahar Ezzedine, "A comparative performance study of the routing protocols RPL, LOADng and LOADng-CTP with bidirectional traffic for AMI scenario," in *Intelligent Computer Communication and Processing (ICCP), 2015 IEEE International Conference on*. 2015, pp. 561–568, IEEE.

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

[11] K. Heurtefeux, H. Menouar, and N. AbuAli, "Experimental evaluation of a Routing Protocol for WSNs: RPL robustness under study," in *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct 2013, pp. 491–498.

[12] A. Yushev, P. Lehmann, and A. Sikora, "6LoWPAN with RPL performance measurements in an Automated Physical Testbed," in *Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS-SWS), 2014 2nd International Symposium on*, Sept 2014, pp. 29–33.

[13] Oana Iova, Fabrice Theoleyre, and Thomas Noel, "Stability and efficiency of RPL under realistic conditions in Wireless Sensor Networks," in *PIMRC*, 2013, pp. 2098–2102.

[14] T. Clausen, U. Herberg, and M. Philipp, "A critical evaluation of the IPv6 Routing Protocol for Low Power and Lossy Networks (RPL)," in *2011 IEEE 7th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct 2011, pp. 365–372.

[15] Mališa Vučinić, Bernard Tourancheau, and Andrzej Duda, "Performance comparison of the RPL and LOADng routing protocols in a home automation scenario," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2013, pp. 1974–1979.

[16] Jiazi Yi, Thomas Clausen, and Yuichi Igarashi, "Evaluation of routing protocol for Low power and Lossy Networks: LOADng and RPL," in *Wireless Sensor (ICWISE), 2013 IEEE Conference on*. IEEE, 2013, pp. 19–24.

[17] Adam Dunkels, "The ContikiMAC Radio Duty Cycling Protocol," Tech. Rep., Swedish Institute of Computer Science, 2011.

[18] Ana Bildea, *Link Quality in Wireless Sensor Networks*, Ph.D. thesis, Université de Grenoble, Nov. 2013.

[19] Kannan Srinivasan, Prabal Dutta, Arsalan Tavakoli, and Philip Levis, "Understanding the causes of packet delivery success and failure in dense wireless sensor networks," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, New York, NY, USA, 2006, SenSys '06, pp. 419–420, ACM.

[20] H. S. Kim, H. Kim, J. Paek, and S. Bahk, "Load Balancing under Heavy Traffic in RPL Routing Protocol for Low Power and Lossy Networks," *IEEE Transactions on Mobile Computing*, vol. PP, no. 99, pp. 1–1, 2016.

[21] M. Michel, S. Duquennoy, B. Quoitin, and T. Voigt, "Load-Balanced Data Collection through Opportunistic Routing," in *2015 International Conference on Distributed Computing in Sensor Systems*, June 2015, pp. 62–70.