Scaling RPL to Dense and Large Networks with Constrained Memory

Joakim Eriksson
RISE SICS and Yanzi Networks
joakim.eriksson@ri.se

Niclas Finne
RISE SICS and Yanzi Networks
niclas.finne@ri.se

Nicolas Tsiftes
RISE SICS
nicolas.tsiftes@ri.se

Simon Duquennoy
RISE SICS
simon.duquennoy@ri.se

Thiemo Voigt
Uppsala University and RISE SICS thiemo.voigt@ri.se

Abstract

The Internet of Things poses new requirements for reliable, bi-directional communication in low-power and lossy networks, but these requirements are hard to fulfill since most existing protocols have been designed for data collection. In this paper, we propose standard-compliant mechanisms that make RPL meet these requirements while still scaling to large networks of memory-constrained IoT devices, where the RAM size does not allow to store all neighbor and routing information. The only node that needs to have storage for all the routing entries is the RPL root node. Based on experimentation with large-scale commercial deployments, we suggest two mechanisms to make RPL scale under resource constraints: (1) end-to-end route registration for downwards traffic and (2) a novel policy for managing the neighbor table. By employing these mechanisms, we show that the bi-directional packet reception rate of RPL networks increases significantly, both in large and dense networks.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*wireless communication*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Routing Protocols*

General Terms

Algorithms, Design, Performance

Keywords

RPL, Scalability, Wireless Networking

1 Introduction

The need for reliable, bi-directional traffic in the Internet of Things (IoT) is evident from the number of applications that require interaction with the IoT devices. In lossy, multi-hop IoT networks, RPL [17] is the most prevalent standard routing protocol, but it has been designed based on data collection protocols such as CTP [10]. In application domains such as smart offices and facility management, there may be hundreds, or even thousands, of IoT devices monitoring and controlling all sorts of activities and equipment. The resource constraints of many types IoT devices—e.g., Class 1 devices with approximately 10 kB of RAM [2]—entail that there is insufficient memory available to store all relevant neighbor information and route entries. Table 1 shows the RAM usage of a Contiki configuration for a commercial setting that with 20 routing table entries and 10 neighbor table entries consumes almost 20% of the available RAM. When we cannot store all neighbor and routing information, the network topology cannot be structured optimally and the performance may suffer.

Table 1. RAM consumption per entry in the neighbor and routing tables in Contiki. The number of entries is taken from existing commercial deployments—this configuration consumes 1800 bytes of RAM for the tables.

Table	RAM / entry	Entries	Content
Routing	50 bytes	20	IPv6 address for route and next hop.
Neighbor	80 bytes	10	802.15.4 address, link stats, RPL info, IPv6 nbr info.

In this paper, we identify two problems of RPL that can severely degrade the performance in large-scale IoT networks, and propose two new mechanisms for mitigating the problems. We implement these mechanisms in ContikiRPL [16], a widely used open-source implementation of RPL which is distributed with the Contiki operating system.

Problem 1: Network size.

A key challenge is to keep a stable topology when the network size increases. As Figure 1 shows, in RPL's storing mode, all nodes have to store route entries for all nodes that have registered their IPv6 address for downward routing through them. This registration is done by sending an ICMPv6 message called Destination Advertisement Object (DAO). Nodes close to the RPL root have to store many route entries if they are responsible for forwarding to a large sub-

International Conference on Embedded Wireless Systems and Networks (EWSN) 2018
14–16 February, Madrid, Spain
© 2018 Copyright is held by the authors.
Permission is granted for indexing in the ACM Digital Library
ISBN: 978-0-9949886-2-1

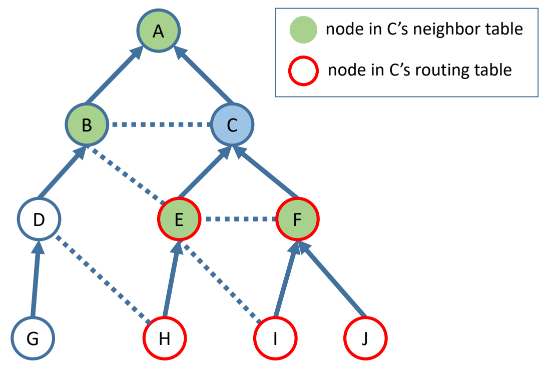


Figure 1. Node C must keep information about its neighbors (filled in green) as well as routing entries for its sub-graph (circled in red). Scaling to networks denser or larger than what the node can store in RAM is a challenge. We propose advanced neighbor table management policies to handle dense networks, and design a reliable, end-to-end route registration mechanism to scale to large networks.

set of the topology. When scaling up the network size, these nodes may not have enough memory to store all registered routes. RPL also defines a message type for acknowledging DAOs (DAO ACK), allowing feedback between the node that sends the DAO and the node that either accepts or rejects the registrations. If the node that receives a DAO has insufficient space to store another route, it can send a negative DAO ACK.

Whenever a node sends a DAO to register its IPv6 address, this DAO will be forwarded all the way to the RPL root. The root is the node that initiates the routing tree setup and is the top of the topology. The forwarding is done on a per-hop basis, which means that if a node is five hops away from the RPL root the DAO will be sent five times, and thus adding a route entry in each of the five nodes along the path to root, including the root node itself.

The RPL standard does not fully specify how to handle DAO and DAO ACK messages. Hence, the RPL implementations typically follow the minimal requirements of the standard, and therefore cannot scale to large networks. In such implementations, a DAO is acknowledged by the next-hop parent only, which limits such message exchanges to a single hop rather than the end-to-end route to the root. A failure to add the route some hops away from the node sending the DAO will thus not be seen by the node, but only by the last node that forwarded the DAO. Since the registering node will assume that its registration was a success, a route from it to the root will not be established.

Mechanism 1: End-to-End DAO ACKs.

To address this problem, we design an end-to-end mechanism for DAO registration. We design this mechanism as standard-compliant, using unmodified messages but defining new rules and semantics for DAO and DAO ACK transmission. In addition, we make it possible to balance the topology by transmitting routing table capacity in the periodically transmitted DIO beacons of RPL. This feature further improves the performance of the end-to-end DAO mechanism and helps to reduce parent switching caused by full routing tables.

Problem 2: Network density.

The second challenge is keeping a stable topology when the network density increases. When density increases, each node in the network will have more neighbors, at some point more than can be stored in the nodes neighbor table [18]. Typically the limitation of storage in the table can be between tens to hundreds of entries. For example, in a deployment with of *Yanzi Networks*, the number of neighbors is ten, which means that any deployment with more than eleven nodes is a potential challenge. When a node's neighbor table is already full but it wants to add a neighbor to the table, it has to discard old entries and hence loses link statistics and other potentially useful information for the routing protocol.

Mechanism 2: Neighbor Selection Policy.

Our mitigation mechanisms consists of selecting the best neighbors neighbors to keep in the routing table from a potentially very large set of candidates. In RPL, all the neighbors that are used as next hop for routes will need to be kept, which limits the number of entries in the table for candidate parents. Our approach is to keep good neighbors in the neighbor table and carefully select which of the new neighbors are worth evaluating and adding to the neighbor table.

Contributions and Roadmap.

To address the challenges of routing reliably downwards in large RPL networks, we make the following research contributions.

- We identify and characterize the problem of scaling RPL to large and dense networks where the nodes in the network cannot store all requested route entries nor all neighbors.
- We introduce two practical mechanisms to manage routing tables and neighbor tables in large IoT networks.
- We evaluate the proposed mechanisms both in simulation and through a large-scale commercial deployment, demonstrating increased reliability in setups with hard memory constraints.

Our paper proceeds as follows. In the next section, we describe the context and real-world motivation in more detail. Section 3 discusses related work, and the necessary background on RPL required to follow the rest of the paper. In Section 4, we present our mechanisms to improve the scalability of downward routing in RPL with memory constraints. After a short discussion on the implementation, we evaluate these mechanisms in Section 6. Before concluding in Section 8, we briefly discuss how the implemented mechanisms have been used successfully in a large-scale commercial deployment in Section 7.

2 Context and Motivation

The RPL routing protocol is used in commercial solutions and is moving into several standards such as Zigbee Jupiter Mesh intended for large-scale deployments. It is crucial for RPL to scale to large IoT networks, where downward connectivity is required and where nodes are memory-constrained. Common low-cost IoT devices have significant resource limitations (e.g., 16-32 kB RAM)—both due to cost per device and energy consumption when adding RAM. While using Flash memory sometimes is an option to extend storage, it often has a lower cut-off voltage that will force it to fail earlier than other components as the battery depletes. Furthermore, the Flash itself has a limited lifetime due to memory wear. In the context of this paper, we define scalable routing as the ability to handle networks that result in more routing and neighbor table entries than nodes can store in RAM.

In this paper, we focus on RPL's storing mode of operation, where all nodes store routes to their sub-graph. An alternative is non-storing mode, where nodes do not store any routing entries and all routing decisions are made by the root. Non-storing mode, however, has its own limitations, in particular the added per-packet, per-hop overhead of carrying source-routing information. Also, many of today's commercial deployments are using storing mode, and some of them, such as smart utility networks, have many hops. Switching to non-storing mode in an already deployed network without causing downtime can be a complex task, and will lead to a higher packet header overhead when there are many hops. Furthermore, although non-storing mode can scale to large networks, neighbor table size restrictions may limit its performance in dense environments. Hence, our neighbor table management policies could be useful also for non-storing RPL networks.

Several issues arise when scaling RPL to large networks with bi-directional communication. First, the DAO route registration is not fully specified which leaves many decisions open for implementers. One of the decisions that are left open is whether the DAO ACK should be sent in an end-to-end manner or not. Typically implementers have interpreted that the DAO ACK should be sent between the each DAO sender along the path to the root and its closest parent. Hence, nodes accept a child without checking that routes can be installed along the entire path. The second issue comes from handling dense networks, i.e., networks where nodes have more neighbors than they can store in their neighbor table. These two issues limit the applicability of RPL, as the network will collapse as soon as it grows beyond the anticipated size or density. This paper presents practical solutions to both issues.

The work presented in the paper was motivated by a real use case from *Yanzi Networks*, an IoT company that deploys large RPL networks. *Yanzi Networks* uses Contiki OS as the base platform for their IoT devices. The starting point of using mainline ContikiRPL for routing was satisfactory at first. However, as soon as the network grew beyond what could be stored in the routing and neighbor tables the network started falling apart. A straightforward solution is to scale up the RAM beyond the expected size of the network, but this adds both costs and increased energy consumption to the devices while not solving the underlying issue. Further, the network sizes expected today might be quite small compared with what will be seen in the future with, for example, long range, smart city applications with ubiquitous sensors.

3 Background and Related Work

This section describes the RPL protocol and reviews earlier work on scalability in low-power wireless networks.

3.1 RPL Background

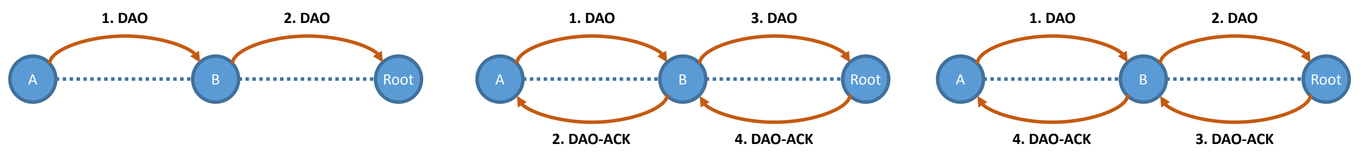
RPL is the standard protocol for low-power IPv6 routing defined by the IETF as RFC6550 [17]. It is a distance-vector protocol, where the routing topology is a Destination-Oriented Directed Acyclic Graph (DODAG) that is typically rooted at a network border router. Each node is attached a *rank* representing its distance to the root using some cost function (*e.g.*, the ETX metric).

The topology is built distributively with the pseudo-periodic transmission of beacons (so-called *DIO*, DODAG Information Object, messages). A node receiving a DIO may choose to join the network. It will then maintain a set of candidate parents, and will elect one preferred parent. The preferred parent is used for *upwards* routing, *i.e.*, routing towards the root. By using a *DIS* (DODAG Information Solicitation) message, nodes can solicit a DIO message from one or more RPL nodes. Nodes typically send DIS messages in multicast when joining a network, and in unicast when actively probing a particular node.

RPL also enables arbitrary traffic patterns: nodes can be targeted and reached from their IPv6 address. Routing from the root to any node, or *downward* routing, is done by using the reverse path in the DODAG. When routing downwards, a node selects any child that has the destination in its sub-graph, and uses this child as the next hop. Any pair of nodes can communicate by routing first upwards to the root (or any common ancestor) and then downwards to the destination.

For the next-hop selection during downward routing, RPL offers two distinct modes of operations: storing and non-storing mode. Storing mode is based on routing tables, while non-storing mode uses source routing. In this paper, we focus on RPL's storing mode of operation, for its fully distributed nature. Beside having a regular 1-hop neighbor table, each node maintains a routing table with one entry per node in the sub-graph, *i.e.*, all downward nodes. The routing table stores the next-hop neighbor to forward messages to each registered destination address prefix. Only the children that are in the neighbor table can be selected as next hops in the routing table because the neighbor table contains additional information required for communication such as the mapping between IPv6 and MAC addresses.

In storing mode, nodes maintain their routing table using *DAO* (Destination Advertisement Object) messages. Whenever switching parent, a node will send a *DAO* to the new parent to ensure proper registration and route installation. The parent will, in turn, relay that information upwards to trigger routing table updates along the path to the root. Similarly, *No-Path DAO* messages are used for route removal. RPL defines an optional *DAO-ACK* mechanism, for the nodes to acknowledge proper route installation/removal or notify failure. We call the message to achieve the latter *DAO-NACK*, although it is simply a *DAO-ACK* message with a particular status code.



(a) Without DAO-ACK **(b) With DAO-ACK** **(c) With end-to-end DAO-ACK**

Figure 2. (a) Without DAO-ACK, node A is registered to B and then to the root without any confirmation. Any failure in the path cause unreachability. (b) With DAO-ACK, node A gets a confirmation that B could register it, but does not know if the full path was successfully created. (c) With our end-to-end DAO-ACK, B makes sure the registration succeeded all the way to the root before sending an ACK back to A.

3.2 Related Work

Early work within sensor networking identified that sensor nodes cannot be expected to hold complete neighbor tables as the networks can be very dense and there will be more neighbors than can be stored in the limited RAM. Woo *et al.* identify the need for neighbor management policies, and suggest such a policy for data collection networks [18]. Their problem is similar to the one we address with our neighbor table policy, but their solution is not applicable to RPL because of differences in the underlying protocol, such as the focus on data collection while we tackle the challenge of bi-directional communication.

The Thread stack is an IPv6-based mesh solution, which has been designed by Nest and standardized by the Thread Group. Unlike RPL, however, it is designed for a maximum of 32 active routers. This limitation helps to make it memory-efficient and to make it possible to ensure that neighbor and routing tables contain fresh information. Thread networks can scale to a few hundred devices in total, but in such networks most of the devices cannot act as routers.

ORPL is an extension of RPL that performs opportunistic routing to achieve scalability in particular for downwards routing [6]. By representing a set of reachable nodes as either a bitmap or as a Bloom filter, ORPL can store this set in a much more compact and hence memory-efficient way than a routing table. In contrast to our work, ORPL is a departure from the RPL standard.

@scale [4] is a large-scale deployment with around 500 devices that employs HYDRO, a predecessor of RPL, for routing. To achieve scalability the authors use multiple load-balancing routers. They find that even in static deployments keeping the routing table dynamic and continuously discovering nodes is key to achieve good performance. We share these experiences, but we also provide concrete policies for neighbor table maintenance under memory constraints.

Dawans *et al.* pointed out the challenge in scaling RPL to dense networks, where not all neighbors can be stored in the neighbor table [3]. The problem discussed is twofold: (1) discarding information from good neighbors is obviously not desirable, as we lose the opportunity to use them as backup parent, but (2) discarding information from neighbors with bad links also comes at a price, as the neighbor might be added later again only to re-evaluate a bad quality link. The paper suggests the need for a neighbor replacement policy but does not investigate the topic further and rather focuses on link probing.

Istomin *et al.* have evaluated RPL in a large-scale smart city deployment where actuation, that is, downward routing is important [11]. Their findings showed that the RPL implementations available at that time were not able to fulfill the requirements. Our paper's goal is to improve the situation by making RPL scalable.

Judging the quality of links is an important issue for probing and adding links to the routing table. Fonseca *et al.* have presented the four-bit wireless link estimator that has shown to reduce packet delivery cost while maintaining a high delivery rate [9]. Baccour *et al.* provide a comprehensive overview of radio link estimation [1] whereas Ruckebusch *et al.* find out that the choice of link estimators for RPL depends on the scenario at hand [15].

Several research papers have studied reliability and load balancing in large-scale RPL networks [7, 12, 13] The papers above achieve promising results, but do not specifically address scenarios where the nodes' memory is insufficient to store all neighbors and routing entries.

4 Scaling RPL with Constrained Resources

RPL is designed to be flexible. In its most basic use case, it can be configured for pure data collection, which entails that any given node in the network needs to keep track of only its best parent in the network to forward the traffic to. This mode is very scalable because there is no need to store multiple routes or neighbors. At the other end of the configuration spectrum are fully bi-directional routable networks where all nodes can be addressed at any time. As long as every node can store a route to every descendant in its sub-graph in its routing table and there is an entry in the neighbor table for each neighbor, it is easy to get RPL to operate reliably. When these tables are too small, however, full connectivity can no longer be guaranteed and hence RPL does not scale.

In the following, we present mechanisms that enhance RPL's ability to scale to networks in which the constrained memory of the nodes is insufficient to store all the information that they would want in the best case.

4.1 Scaling with Network Size

In large networks, nodes are unable to store routes to all descendants in their routing tables. A major issue is the registration of new nodes that needs to be propagated all the way to the root node, and hence requires an entry in the routing tables on all nodes on the path to the root. The problem is exacerbated by the fact that nodes closer to the root have more descendants than their children.

4.1.1 *End-to-End Path Reservation*

The mechanism used for registering downward routes in RPL is the DAO, through which a node sends its routable address upwards all the way to the root node. This is either sent with best effort (see Figure 2a) or sent with request for acknowledgment (DAO-ACK). For scaling, we use the DAO-ACK (see Figure 2b). The RPL specification [17] is, however, unspecific about DAO-ACKs: it just describes that an ACK should be sent, and how it should be formatted.

In order to scale up and maintain reachability, it is crucial that a DAO-ACK is not just processed by a node and its direct parent, but that the full path up to the root is acknowledged. We propose to use an end-to-end DAO-ACK (illustrated in Figure 2c) where any node that gets a DAO forwards it as the specification states, but waits for the response from its parent before sending the DAO-ACK. This way of handling the DAO-ACK makes it possible for nodes to know whether the route was installed along the entire path to the root.

Initially, the ContikiRPL implementation always accepted new route registrations, by removing the oldest route in case the routing table was full. This behavior, however, scales poorly with network size. With our end-to-end DAO mechanism, nodes with a full routing table decline new route registrations and send a DAO-NACK back to the originator.

If a node does not receive a DAO ACK or NACK within a short time after sending a DAO, it will retransmit the DAO a few times. By contrast, if it gets a DAO-NACK, it will attribute that to the specific path used, and try to select a new parent.

The major new benefit with this end-to-end DAO is that as soon as the node gets a DAO-ACK for a sent DAO, it will have the guarantee that there is a downward route registered all the way up to the root, and it can perform bi-directional communication. The design of this end-to-end DAO mechanism complies with RFC 6550.

4.1.2 *Topology Balancing*

Reducing unnecessary traffic saves energy and bandwidth. This is of particular importance when networks are large or dense. In such situations, it is beneficial that nodes provide information about the number of free entries in the routing table to their children using RPL's aperiodic beacons (DIO). This prevents nodes from selecting parents whose routing table is already full and sending the corresponding DAO messages to them. For end-to-end reliability, not only the parent's routing table needs to have a free entry but also all the nodes on the path from the parent to the root need to have free entries. Therefore, nodes should not only announce their free routing table entries but the number of free entries of the most restricted routing table on the path to the root. This number is the minimum of a node's free entries and the number of free entries announced in the parent's DIO messages.

4.2 Scaling with Network Density

In dense deployments there will be nodes that cannot store all their neighbors in the neighbor table. When this happens, many network protocol implementations start to perform poorly—they either do no longer add new neighbors and therefore cannot maintain a good network topology, or they continuously add and replace neighbor table entries, causing undesirable churn.

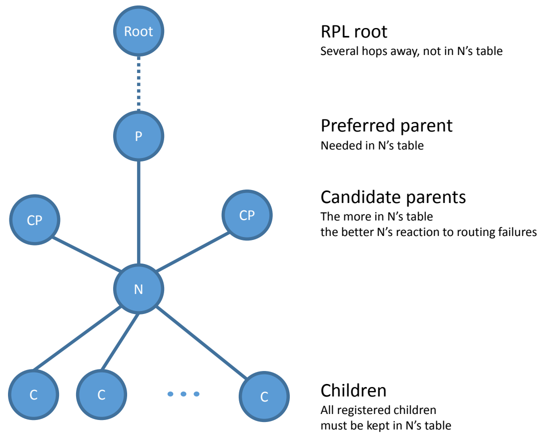


Figure 3. Illustration of a neighbor table policy. Here, the preferred parent, two candidate parents, and three children are stored. Whenever a child attempts to register, node N will check if there is space left in the neighbor table before ACKing. If there is no space, it may decide to free a slot by removing a candidate parent, depending on the policy in use.

4.2.1 *Neighbor Management*

To mitigate the network density problem, we suggest an approach where a new neighbor is added to the neighbor table only when there are indications that it is significantly better than the currently worst neighbor in the table. The goodness of a neighbor can be derived from a number of factors that are stored in the neighbor table.

The neighbor table of a low-power IPv6 implementation (such as Contiki's) typically contains, among other things, the following information:

- **Link ETX**, an estimate of the link quality to the neighbor;
- **Link ETX freshness** an indicator of how accurate the link estimate is (freshness, transmission count, etc.);
- **IPv6 reachability state** required by the IPv6 neighbor discovery protocol.

Whenever receiving a beacon (DIO) from a neighbor, the following information can be obtained:

- **Link RSSI**, the signal strength of the link from the neighbor;
- **RPL Rank**, the position of the neighbor in the RPL topology;
- **RPL Metrics** used to compute the node's rank in case it selects the neighbor as preferred parent.

4.2.2 *Neighbor Table Policy*

We describe next how to decide whether to add a neighbor to the neighbor table or not based on the above information. Typically a new neighbor is added when receiving

DIOs and IPv6 neighbor discovery messages. While in some implementations the decisions about adding new neighbors is distributed over different parts of the stack, we advocate to centralize the decisions about the addition of a neighbor to a new neighbor table management module. This makes it possible to avoid adding neighbors that either are likely to be worse than the ones in the table or that may not be useful for other reasons, such as when they intend to join another network.

The default neighbor policy in Contiki is to always add new neighbors as they appear. The neighbor that is removed from the table when it is full is the oldest neighbor that is not locked. Locking occurs when a neighbor is either a preferred parent (upward routing) or a child (downward routing). In dense networks, there are situations where nodes lock all of their entries to the preferred parent and current children. This makes it impossible to keep track of other candidate parents, which is essential for robust topology maintenance.

In many other 6LoWPAN implementations, the policy is to add neighbors as long as there is room and then no longer add new neighbors when the neighbor table is full.

We propose a novel neighbor policy for RPL that maintains one preferred RPL parent, a few neighbors that are good candidate parents for upward connectivity and then a set of children or next-hop neighbors used for routing traffic downwards. This allocation of the table makes it possible to switch the preferred parent at any time, as there are always several parents in the table. It also removes the problem that all entries are locked and hence a parent switch is impossible.

With our policy in place, nodes are able to switch parent even in dense environments. This however, also comes with a risk: a node might discover a promising parent, decide to switch to it, only to discover that the link is actually worse than initially estimated. To mitigate this issue, we extend our policy so as to only ever select parents whose link estimate is fresh. In order to help obtain fresh estimates, we extend our mechanism with a periodic link probing feature. Periodically (*e.g.*, every 2 min), nodes select a probing target (the current parent if not fresh, else another candidate parent) and sends a unicast DIS to it, so as to assess the link quality in both directions and obtain a rank update from the neighbor.

Using our neighbor policy we always keep the preferred parent in the table, provision some fixed space for children, and use the remaining space for candidate parents, used as backup should the preferred parent fail. In a neighbor table of size N, we can handle 1 preferred parent, M children, and $N - M - 1$ candidate parents, as illustrated in Figure 3. This policy avoids the scenario where the table gets filled with the preferred parent and children only, leaving no space for candidate parents. Such a scenario would be detrimental to the reliable operation of RPL, because it would not maintain link estimates to backup parents.

Our neighbor table policy operates as follows:

- When a neighbor table is empty new neighbors can always be added.
- When a table is full and there is a new candidate neighbor, we take different actions based on what type of message it makes itself known through.
  - DIO: Add the neighbor if there is indication that this is a potentially better parent than the worst of the current parents. E.g. RPL rank is better and the link is expected to be good based on its RSSI.
  - DAO: Add the neighbor if there is room for another child and send a DAO ACK. Else discard the DAO and send a DAO-NACK. Children already in the table are never evicted but may expire due to DAO lifetime timeout.
  - DIS: Add the neighbor if the DIS is a unicast transmission, and there is room for another child. Else, ignore the DIS.

The neighbor policy problem and parts of the proposed mechanism is also described in an IETF draft within the LWIG working group.

5 Implementation

We implement the proposed mechanisms in the Contiki operating system [5]. There is, however, nothing that prevents the implementation of the mechanisms in other operating systems. The end-to-end DAO ACK mechanism has been implemented as modifications to the Contiki RPL stack.

For the neighbor table policy mechanism, we have centralized neighbor management to a new module in ContikiRPL. This avoids letting different layers take decisions on which neighbor to keep based on information local to the layer only. Instead, the centralized module has the exclusive right to decide which neighbors to add or evict according to the current policy. In the evaluation we make use of a neighbor table with information from many modules including link statistics, RPL routing protocol state, IPv6 addresses and their states.

6 Evaluation

In this section we evaluate the proposed mechanisms described in Section 4. For this purpose, we use the COOJA simulator [14, 8] since it allows us to compare scenarios with different network stack settings under repeatable conditions, that is, without any external effects that would introduce randomness. COOJA executes deployable Contiki firmwares. The simulated devices are Arago Systems' WiSMotes, which include a Texas Instruments MSP430 microcontroller and a CC2520 radio that is compliant with IEEE 802.15.4. The communication stack configuration includes an 802.15.4 CSMA MAC protocol in non-beacon enabled mode, 6LoWPAN, and and IPv6 with RPL operating in non-storing mode.

6.1 End-to-End DAO

The end-to-end route registration mechanism is designed to make RPL handle the case when nodes cannot store all the routes needed in the routing table. This is the case when the network is large and dense. Typically the nodes that are closer to the RPL root have too small tables when the network grows since these nodes have to forward the downward traffic to all nodes below in the graph and therefore need to store many routing entries in the table. The first experiment evaluates the effectiveness of the end-to-end registration mechanism and its impact on the reliability of bidirectional communication.

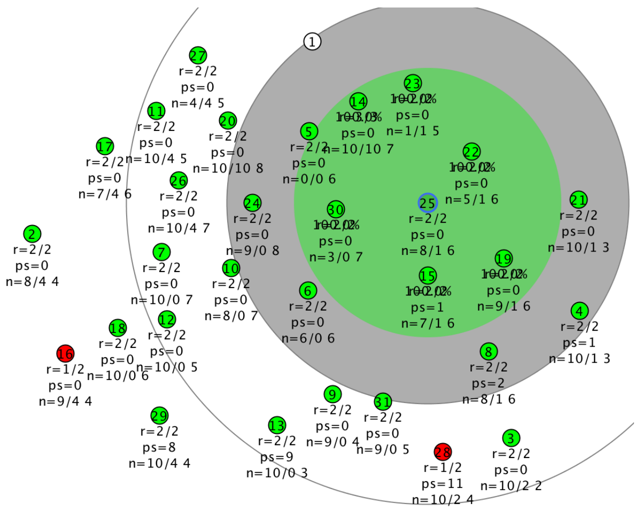


Figure 4. The topology used in our simulations. The root is node 1. The green disk shows node 25's radio range, while the gray disk shows its area of interference.

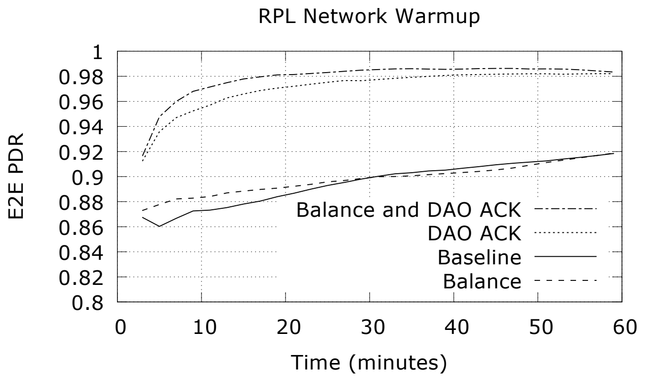


Figure 5. RPL network warmup time using four different configurations. Both experiments using DAO ACK reach 98 percent E2E PDR. Adding information about free routes and neighbors in the DIOs makes it reach that level faster (Balance and DAO ACK).

6.1.1 Setup

The first evaluation of the end-to-end DAO registration is made in COOJA to be able to control the communication range.

All simulations contain one RPL root that has memory to store all nodes in the network so that the root will never limit scalability. All other nodes use a limited routing table that can store 10 routes. In our experiments all nodes send a short UDP message to the root once per minute and the root responds with a reply. Successful reception of the reply is counted as packet delivery.

In the first experiment the network consist of one RPL root and 30 nodes, as depicted in Figure 4.

We compare four different RPL configurations, each for one hour:

- **Baseline** No scalability mechanism enabled
- **Balance** With DIOs containing topology balancing information (see §4.1.2)
- **DAO-ACK** With our end-to-end DAO-ACK (see §4.1.1)
- **Balance and DAO-ACK** With both topology balancing and end-to-end DAO-ACK

The metric for the evaluation is the bi-directional packet delivery ratio (E2EPDR) for all the messages sent from the node (including the response from the RPL root).

6.1.2 Results

Figure 5 shows the simulation results. As expected, the effect of using balancing together the baseline configuration is insignificant. The baseline by itself converges to an E2EPDR of 92 percent after one hour, while the second configuration, with balancing information in DIOs, converges to an E2EPDR of around 92 percent as well. The main reason for packet loss during the warmup period in RPL in this configuration is due to routing topology inconsistency. This happens when a node switches between one parent and another. During the switch it will first send a deregistration - a DAO with zero lifetime - and then a new registration. During this time the node itself will shortly be disconnected. Also the all children below this node will also be disconnected and possibly not even realizing it immediately and thus getting a longer disconnect before re-registering.

The final two configurations both use the end-to-end DAO ACK. They reach an E2EPDR of 98 percent. The use of balance information is of higher value when the DAO ACK is turned on. This is due to the lack of feedback of failure when DAO ACK is not available which means that the nodes do not pick a new parent but stay with a parent that could not store its downward route. In the other case, when DAO ACK is enabled, the information from the DIO can be used to avoid picking another parent does not have room for routes and to slightly speed up the selection.

6.2 Neighbor policy

One of the main challenges when scaling to dense networks is to handle the neighbor tables. The assumption is that the neighbor table can only hold a limited number of neighbors and that sometimes there are more neighbors than entries in this table. The problem that occurs when replacing a neighbor in the table with a new one that might be better is that it is hard to understand the quality of the new neighbor before we discard the old one. Picking wrong neighbors might cause RPL to switch to what might look like a better upstream parent but that fails to be one in the long run.

In the experiments in this section we evaluate the effectiveness of the neighbor table policy described in Section 4.2.2.

6.2.1 Setup

To isolate the impact of the limited neighbor table this setup includes a large routing table but a neighbor table with a limited number of entries. In each simulation, the root node has large enough tables but all the other nodes are limited to ten neighbors in the neighbor table. The simulated network consists of the RPL root and 30 other RPL nodes. We vary the network density by changing the range of the transmissions. We evaluate the end-to-end PDR.

We compare four different neighbor table policies:

- **Hard-Lock** - add neighbors until the table is full - then keep them until timeout;
- **LRU Overwrite** - always add neighbors based on least recently used model;
- **Soft-Lock / Contiki Baseline** Using Contiki's out-of-the-box neighbor table, which basically locks the preferred parent and children, but does not provision space for candidate parents;
- **Our neighbor policy** Our novel policy, which keeps a fixed number of entries for candidate parents and that evaluates each new parent based on the expected quality before adding.

All four described neighbor table policies are used in simulations with the same set of nodes. Nodes run the same application as in the experiment in the previous section: one node acts as the RPL root and runs an UDP echo server and the other nodes send one packet per minute to the RPL root.

In the simulation we increase the transmission range in order to increase the density.

6.2.2 Results

Figure 6 presents the results. The two best policies are the Soft-Lock and our novel policy. Since the simulation is static and the quality of the links does not change over time these two are expected to behave similar - with the possibility of the soft-lock policy to perform slightly better as it will not need to reserve any alternative parents but can use all the neighbor table entries for downward routing (e.g. used as next hop). The hard lock policy locks the first ten nodes it sees and therefore it will end up with less possibilities of adding children in cases where the table is already filled with parents only. Finally the LRU policy only works well for low densities when the number of neighbors is low and all can be stored in the table. As soon as the network gets dense and there are more neighbors than can be stored in the table, the LRU policy starts to remove neighbors that are used for downward routing - causing failures. As we increase density by increased range this policy will recover slightly when most nodes can reach the root node as they will register immediately to the root, thus minimizing the risk of losing the downward route due to another node on the path removing the route (or the next hop).

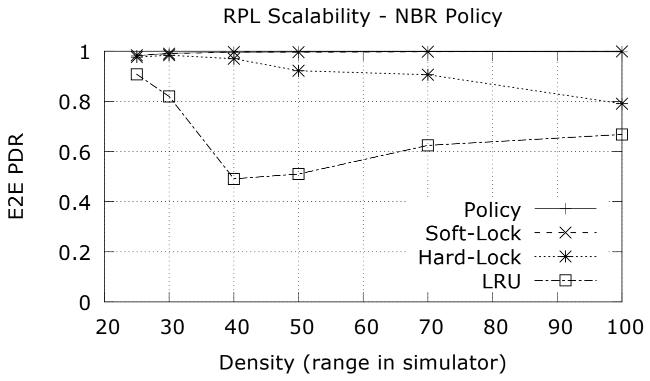


Figure 6. End to end PDR of an RPL network using varying implementation of neighbor table policy. The density of the network is varied by changing the communication range from a setting giving less than 10 neighbors in the table to a setting where most nodes are in range - e.g, giving 25-30 neighbors.

7 Large-Scale Commercial Deployment

This section reports on large-scale deployments we performed in collaboration with *Yanzi Networks*. The first deployment is a smart office application, with a total of 1000 nodes split in several networks of 120 or more nodes. The nodes are sensors and actuators such as motion detectors, temperature and humidity sensors, and smart plugs. The sensors feed data into cloud applications that analyze the work environment and visualize the data in various ways. We use an ARM Cortex M3 SoC, which features 16 kB RAM, 256 kB flash and a built-in IEEE 802.15.4 radio. Since the application layer, security protocols, IPv6 and MAC-layers buffers take a large portion of the available memory, we were constrained to run with only 10 neighbor entries and 20 route table entries, way below the density and size of the different physical networks. With the help of the mechanisms in this paper, we deployed all networks successfully, with all nodes becoming addressable in spite of the harsh memory constraints.

The second deployment is an even more dense installation. Also this networks includes the scalability improvements in this paper. This network shown in Figure 7 consists of more than 500 nodes on one single gateway. In this figure, the red nodes have recently (last 30 minutes) changed their parent in the topology, yellow nodes changed a while ago (up to four hours ago), and the blue nodes are very stable and have not changed their parents during the last four hours. Since there is a significant risk of packet loss while switching parent in a storing mode RPL network, a stable topology is typically also giving less packet loss.

8 Conclusion

This paper sheds some light on the RPL scalability problem for bi-directional traffic and in constrained memory. We introduce an end-to-end DAO registration mechanism, which enables scaling to large networks, using a constant space for routing entries. We also present a neighbor table management policy, to scale to dense networks, using a constant space for neighbor entries. Our simulations and commercial deployment show that with these two mechanisms, RPL can scale far beyond the number of neighbors and routes that can be stored in RAM.

9 Acknowledgments

Thanks to the development team at *Yanzi Networks* for the intense discussions during the course of the work. The work is partly supported by the distributed environment Ecare@Home funded by the Swedish Knowledge Foundation, 2015-2019.

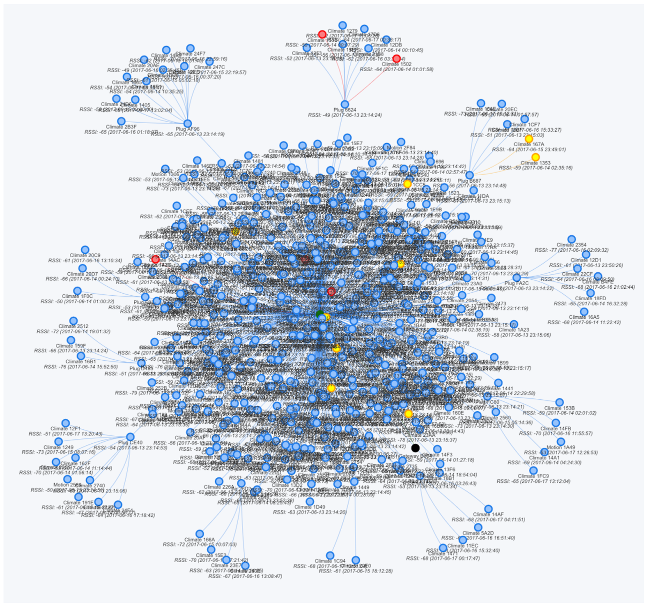


Figure 7. A deployed, very dense RPL network with more than 500 nodes. The nodes have room for 10 entries in the neighbor table and 20 entries in the routing table.

10 References

[1] N. Baccour, A. Koubâa, L. Mottola, M. A. Zúñiga, H. Youssef, C. A. Boano, and M. Alves. Radio link quality estimation in wireless sensor networks: a survey. *ACM Transactions on Sensor Networks (TOSN)*, 8(4):34, 2012.

[2] C. Bormann, M. Ersue, and A. Keranen. RFC 7228: Terminology for Constrained-Node Networks, May 2014.

[3] S. Dawans and S. Duquennoy. On Link Estimation in Dense RPL Deployments. In *Proceedings of the International Workshop on Practical Issues in Building Sensor Network Applications (IEEE SenseApp 2012)*, Clearwater, FL, USA, Oct. 2012.

[4] S. Dawson-Haggerty, S. Lanzisera, J. Taneja, R. Brown, and D. Culler. @ scale: Insights from a large, long-lived appliance energy WSN. In *Proceedings of the International Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, Beijing, China, 2012.

[5] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the IEEE Workshop on Embedded Networked Sensor Systems (IEEE Emnets)*, Tampa, Florida, USA, Nov. 2004.

[6] S. Duquennoy, O. Landsiedel, and T. Voigt. Let the Tree Bloom: Scalable Opportunistic Routing with ORPL. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys 2013)*, Rome, Italy, Nov. 2013.

[7] S. Duquennoy, B. A. Nahas, O. Landsiedel, and T. Watteyne. Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys 2015)*, Seoul, South Korea, Nov. 2015.

[8] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P. J. Marrón. COOJA/MSPSim: Interoperability Testing for Wireless Sensor Networks. In *SIMUTools 2009*, Rome, Italy, Mar. 2009.

[9] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis. Four-bit wireless link estimation. In *Proceedings of the Workshop on Hot Topics in Networks (ACM HotNets)*, Atlanta, Georgia, USA, Nov. 2007.

[10] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, Berkeley, CA, USA, 2009.

[11] T. Istomin, C. Kiraly, and G. P. Picco. Is RPL ready for actuation? A comparative evaluation in a smart city scenario. In *European Conference on Wireless Sensor Networks*, pages 291–299. Springer, 2015.

[12] H. Kim, J. Paek, and S. Bahk. QU-RPL: queue utilization based RPL for load balancing in large scale industrial applications. In *12th Annual IEEE International Conference on Sensing, Communication, and Networking, SECON 2015, Seattle, WA, USA, June 22-25, 2015*, pages 265–273, 2015.

[13] M. Michel, S. Duquennoy, B. Quoitin, and T. Voigt. Load-Balanced Data Collection through Opportunistic Routing. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems (IEEE DCOSS 2015)*, Fortaleza, Brazil, June 2015.

[14] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Proceedings of the First IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006)*, Tampa, Florida, USA, Nov. 2006.

[15] P. Ruckebusch, J. Devloo, D. Carels, E. De Poorter, and I. Moerman. An evaluation of link estimation algorithms for rpl in dynamic wireless sensor networks. In *6th EAI International Conference on Sensor Systems and Software*, Oct. 2015.

[16] N. Tsiftes, J. Eriksson, and A. Dunkels. Low-Power Wireless IPv6 Routing with ContikiRPL. In *Proceedings of the International Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, Stockholm, Sweden, Apr. 2010.

[17] T. Winter (Ed.), P. Thubert (Ed.), and RPL Author Team. RPL: IPv6 Routing Protocol for Low power and Lossy Networks, Mar. 2012. RFC 6550.

[18] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, pages 14–27, Los Angeles, California, USA, 2003. ACM Press.