Route or Flood? Reliable and Efficient Support for Downward Traffic in RPL

Timofei Istomin, Oana Iova, Gian Picco, Csaba Kiraly

Route or Flood?
Reliable and Efficient Support for Downward Traffic in RPL

Timofei Istomin, University of Trento, Italy
Oana Iova, Université de Lyon, INSA Lyon, Inria, CITI, France
Gian Pietro Picco, University of Trento, Italy
Csaba Kiraly, Bruno Kessler Foundation, Italy

October 1, 2019

Abstract

Modern protocols for wireless sensor networks efficiently support multi-hop *upward* traffic from many sensors to a collection point, a key functionality enabling monitoring applications. However, the ever-evolving scenarios involving low-power wireless devices increasingly require support also for *downward* traffic, e.g., enabling a controller to issue actuation commands based on the monitored data. The IETF Routing Protocol for Low-power and Lossy Networks (RPL) is among the few tackling both traffic patterns. Unfortunately, its support for downward traffic is significantly unreliable and inefficient compared to its upward counterpart.

We tackle this problem by extending RPL with mechanisms inspired by opposed, yet complementary, principles. At one extreme, we retain the route-based operation of RPL and devise techniques allowed by the standard but commonly neglected by popular implementations. At the other extreme, we rely on flooding as the main networking primitive. Inspired by these principles, we define three base mechanisms, integrate them in a popular RPL implementation, analyze their individual and combined performance, and elicit the resulting tradeoffs in scalability, reliability, and energy consumption. The evaluation relies on simulation, using both real-world topologies from a smart city scenario and synthetic grid ones, as well as on testbed experiments validating our findings from simulation. Results show that the combination of *all* three mechanisms into a novel protocol, T-RPL *i)* yields high reliability, close to the one of flooding, *ii)* with a low energy consumption, similar to route-based approaches, and *iii)* improves remarkably the scalability of RPL w.r.t. downward traffic.

1 Introduction

The last two decades have seen a surge of research and industrial applications exploiting low-power wireless radio communication. This body of work has focused primarily on exploiting this technology for distributed monitoring, popularized by wireless sensor networks (WSNs), with the fundamental goal of enabling collection, at a designated root node, of the data sensed by many other network nodes. The crux of the problem has been to design protocols that provide *reliable* and *efficient* support for this *upward* traffic from the many nodes to the root. Indeed, these networks are often referred to as *low-power and lossy networks (LLNs)*, which concisely summarizes their challenges. On one hand, low-power operation unleashes network nodes from power cables; nodes are typically autonomously powered, e.g., through batteries. The efficiency of network protocols is therefore key, to avoid rapid depletion of the limited energy supply, and enable long-term operation. On the other hand, the hardware compromises enabling low-power operation define a breed of wireless technology, notably represented by the IEEE 802.15.4 standard, whose reliability is significantly lower w.r.t. common and energy-hungry radios, e.g., WiFi. Interestingly, increasing reliability requires extra communication, which is inherently at odds with limited energy consumption. However, research has significantly progressed, and numerous established solutions efficiently reconciling these conflicting goals for data collection exist nowadays.

Downward traffic and RPL: The Skeleton in the Closet. Nevertheless, the ever-evolving scenarios envisioned for the Internet of Things (IoT), increasingly require support also for the opposite, *downward* traffic from the root to the rest of the network. Needless to say, support for this traffic must be similarly reliable and efficient; actually, many scenarios require even *more* reliability, as in monitoring *and control* scenarios, where downward traffic carries commands to actuators.

In this context, the Routing Protocol for Low-power and Lossy Networks (RPL) [**?**], has attracted considerable attention. First of all, RPL is an IETF standard, and is therefore expected to enable seamless interconnection and interoperability across many devices, effectively enabling many Internet of Things (IoT) applications. In this respect, *"RPL was designed with the objective to meet the requirements spelled out in RFC5867, RFC5826, RFC5673, and RFC5548."* (p. 8, [**?**]). These RFCs distilled requirements from several application domains, notably home and building automation [**?**, **?**], as well as industrial and urban settings [**?**, **?**]. All these domains require support for both upward and downward traffic. For instance, the RFC 5548 about urban LLNs (U-LLNs) states (p. 4, [**?**]):

> *Unlike traditional ad hoc networks, the information flow in U-LLNs is highly directional. There are three main flows to be distinguished: 1. sensed information from the sensing nodes to applications outside the U-LLN, going through one or a subset of the LBR(s);*[1] *2. query requests from applications outside the U-LLN, going through the LBR(s) towards the sensing nodes; 3. control information from applications outside the U-LLN, going through the LBR(s) towards the actuators.*

where the first flow is upwards and the other two are downwards. As the RPL standard was inspired by the requirements of the above domains, it supports both patterns in a single protocol (§2).

Unfortunately, in practice support for downward traffic is significantly less developed in RPL than its upward counterpart. We provide concrete evidence for this statement in the context of a real-word smart city scenario that we retain as reference in this paper. As we describe in §3, we have access to the exact position of 864 nodes deployed on lampposts for monitoring and controlling public lighting, a representative application among those targeted by RPL. The nodes are organized in 13 clusters with different scale and physical topology, allowing us to explore diverse network configurations while remaining in the same application domain.

In §4 we use this extensive and realistic network setup to analyze the performance of the downward portion of ContikiRPL, arguably the most popular implementation available, and show that it provides unacceptable reliability for networks larger than 50 nodes, a far cry from the large scale envisioned by the aforementioned standard and IETF documents. We base our findings on the TMote Sky platform, a popular choice representative of the challenges imposed by the typical resource-scarce LLN devices targeted by RPL. However, the reason for the abrupt degradation in reliability lies precisely in the inability of RPL to cope with the scarcity of data memory on this target platform. In RPL storing mode, further discussed in §2, each node on the downward routing path from the root towards a destination must store an entry towards it. As the TMote Sky is equipped with 10kB of data memory, routing entries waste precious memory for the application; nevertheless, their allowed maximum number bears a direct impact on reliability. For instance, when set to 50 entries as in §4, this directly explains the abrupt degradation in reliability once the number of destinations to be served exceeds this limit. In other words, *RPL shows its shortcomings precisely when used with the resource-constrained devices it was designed for.*

Research Goals. A solution viable in some scenarios is to add extra memory, but this clashes with the trend towards smaller, cheaper, and increasingly resource-scarce devices enabled by ever-increasing miniaturization. Further, it is worth noting that the culprit is *not* the *absolute* number of routing entries available; rather, it is the *ratio* between them and the total number of destinations to be reliably served. In this paper, we argue that *this ratio can be significantly improved by making a more efficient use of memory and communication resources.*

Resorting to the non-storing mode offered by RPL (§2), which relies on source routing and directly encodes the route in the downward packet, is not a solution either, as the two modes of operation have

[1] Lossy Network Border Router (LBR), in this paper simply referred to as the *root*.

different use cases and tradeoffs [**?**]; for instance, in non-storing mode packets become larger and routes less reliable. Nevertheless, recent years have seen a surge of non-storing mode implementations, motivated by the lack of scalability shown by both open source (e.g., TinyOS and Contiki) and industrial (e.g., Cisco[2]) implementations of storing mode. In contrast, *our goal in this paper is to demonstrate that a reliable and efficient downward RPL in storing mode is possible.* In doing so, we push the applicability of storing mode well beyond what hitherto shown in the literature and redefine the tradeoffs and perceptions commonly associated to this mode of operation[3].

We achieve this goal by drawing inspiration from our own preliminary work [**?**] where we observed, in the same smart city scenario mentioned above, that a simple dissemination-based flooding protocol systematically provided near-perfect reliability even in scenarios where the route-based approach of ContikiRPL performed unacceptably. In that work, however, we did not consider energy consumption, which is the evident weakness of a flooding approach due to its communication overhead. This observation, however, triggered the following question:

> *Can we design a protocol that borrows from both route- and flood-based approaches, and finds the right balance by amplifying their benefits and mitigating their drawbacks?*

Research Contributions. The rest of the paper seeks an answer to this question. We illustrate (§5) three core mechanisms striking different tradeoffs in the design space between routing and flooding. The first mechanism, SWITCH, is fully route-based and exploits features of the RPL standard currently neglected by popular implementations. The other two mechanisms, ROOT and MCAST, instead depart from the standard by introducing increasing degrees of dissemination. ROOT replaces downward unicast messages originating at the root with 1-hop broadcast ones, significantly de-congesting the root, which normally is required to store *all* destinations. MCAST relies on multicast to support multi-hop dissemination while limiting its scope, and therefore communication overhead. Further, these mechanisms can be combined pairwise, or all together in a new protocol that we call T-RPL[4].

The style of our investigation is system-driven, and relies on an implementation of all these alternatives, described in §6, whose performance is then evaluated and compared in several ways. First, in §7 we compare individually the performance of the three base mechanisms and their combinations against the baselines representing the extremes of our design space: ContikiRPL as an example of a route-based protocol, pure flooding at the other extreme. This evaluation shows that indeed T-RPL is the variant that strikes the best tradeoffs between reliability and energy consumption, thanks to its peculiar combination of mechanisms. In §8 we compare T-RPL against representative protocols from the state of the art, including TinyRPL, arguably the other popular RPL implementation, demonstrating that none of them comes close to the performance of T-RPL and, as far as reliability is concerned, flooding. However, as mentioned above, in practice downward traffic and upward traffic coexist in real-world applications; a question is therefore whether and how they affect each other. We answer this question in §9. Given the breadth of the parameter space we consider, the evaluation in §7–§9 is carried out in simulation, using the popular Cooja [**?**] tool. However, in §10 we validate our simulation results via experiments performed in a 50-node testbed at our premises, whose results confirm the insights and trends we observed.

In §11 we distill the key lessons learned from these extensive simulation and real-world evaluations and offer a few reflections on how they can inspire new directions in the related research community, whose efforts are discussed in §12 before the concluding remarks in §13.

2 RPL: The Routing Standard for the Internet of Things

The standardized IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [**?**], has been designed to connect "thousands of constrained devices". The protocol builds a Destination-Oriented Directed Acyclic

[2] `http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/rpl/configuration/15-mt/rpl-15-mt-book.pdf`

[3] A comparison against non-storing mode is outside the scope of this paper; it would require an in-depth evaluation per se.

[4] T-RPL is pronounced as in "triple", hinting at the fact that it combines three mechanisms. Further, the "T" in the name is also the initial of Trento, the city where all authors were based at the time.

Graph (DODAG) topology with the border router as the root, which is the connection point between the low-power lossy network (LLN) and the Internet. The DODAG root is the destination for the upward (convergecast) traffic originating from the LLN and the source of downward traffic destined to the LLN nodes. The former traffic, also called *multipoint-to-point* in RPL terminology, is usually associated with sensor readings or events, and the latter *point-to-multipoint* traffic with configuration messages or actuation commands. RPL also support a *point-to-point* traffic between any two LLN nodes, not considered here.

Each DODAG node selects *multiple* parents, i.e., neighbors closer to the root, based on a routing metric. Nevertheless, only one of them (the *preferred* parent in RPL jargon) is used at any time to forward the upwards packets; the other parents serve as backup when the preferred one is unreachable. Figure 1a shows an example DODAG using minimum hop count as the routing metric. Although C has two parents, it uses only the preferred parent $B1$ for all packets destined to A.

This topology enables network nodes to send packets to the root (upward) while keeping only minimal routing state information. However, downward traffic requires much more routing state, since any node is a potential destination and the network needs to maintain paths from the root to all of them. To establish this routing state, RPL uses Destination Advertisement Object (DAO) messages. Each destination *announces* itself to the root by sending DAO messages, which are propagated upward in the topology through the so-called DAO parent. The standard specifies that the latter can be any of the node's parents; however, in popular implementations (e.g., Contiki, TinyOS) the DAO parent coincides with the preferred parent, for simplicity.

For downward traffic, RPL identifies two significantly different modes of operation: *storing* (our current focus) and *non-storing*. In the latter, the root is the only node in the network that has a routing table maintaining a global view on the network, learned from DAO packets containing the DAO parents of every node. When sending a data packet, source routing is employed; the root computes the end-to-end routing path and attaches it to the packet as a complete list of forwarders. In storing mode, instead, the routing state is distributed across the network; every node maintains a routing table listing the next-hop forwarders for all descendants (nodes in the sub-DODAG), learned from the received DAOs. In this mode of operation, nodes do not know the complete path towards the destination; at every hop a decision is taken on where to forward the packet. If at any point a forwarder receives a packet from the root with unknown destination, it must either drop it or return it to the sender, which could try another path (if any).

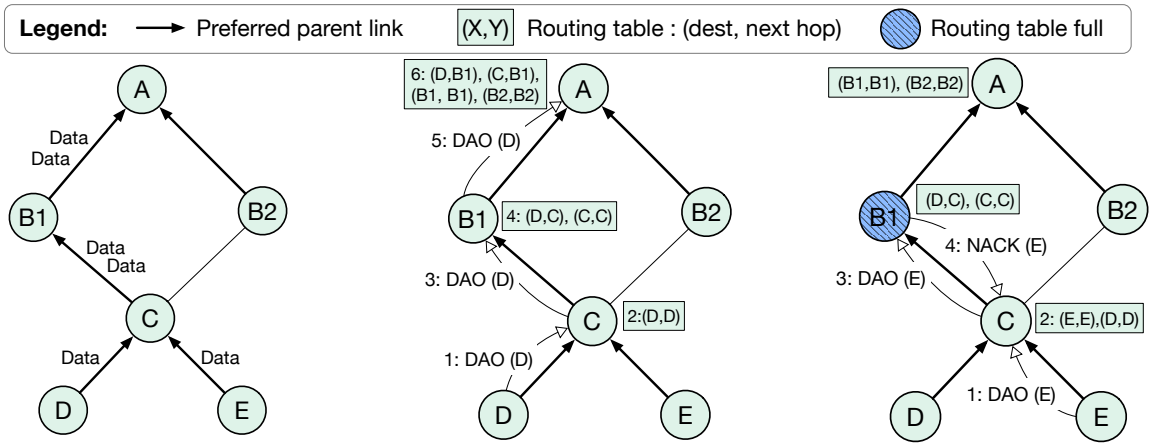
Each mode of operation has its own advantages and drawbacks; the choice between them must be performed on a case-by-case basis [**?**]. Storing mode enables the selection of more reliable routing paths, but suffers in large networks of resource-constrained devices. Non-storing mode ameliorates this problem, but induces overhead and unreliability due to the hefty headers associated to source routing. In this paper, we focus on storing mode with the declared goal (§1) of showing that it can achieve significantly higher scalability and reliability than hitherto shown in the literature.

Figure 1b exemplifies how routing tables are populated when storing mode is used. D announces itself as a possible destination to the root by sending to its DAO parent, C, a DAO packet with its own IP address as a target. C checks if sufficient memory allows the addition of D to its routing table; if this is the case, C forwards the DAO upward to $B1$. This process is repeated by every node that receives the DAO, until the message reaches the root.

3 Motivating Scenario: A Smart City LLN Deployment on Lampposts

The work described in this paper was originally motivated by our collaboration with a company in charge of deploying a large-scale IEEE 802.15.4 wireless infrastructure on the lampposts of Trento, Italy. This infrastructure is used to monitor and control public lighting and other "smart city" applications, in line with several other experiences around the world [**?**, **?**, **?**].

Through this collaboration, we obtained complete knowledge of the placement of 864 nodes and their grouping into 13 clusters, served by independent gateways connected to a remote control center. This knowledge is valuable, as datasets of this scale and detail are largely missing in the literature. Indeed,



(a) DODAG topology with hop count; C has B1 as its preferred parent.

(b) Construction of downward routes: D announces itself to the root.

(c) E announces itself to the root; B1 is out of memory.

Figure 1: RPL – upward and downward route construction.

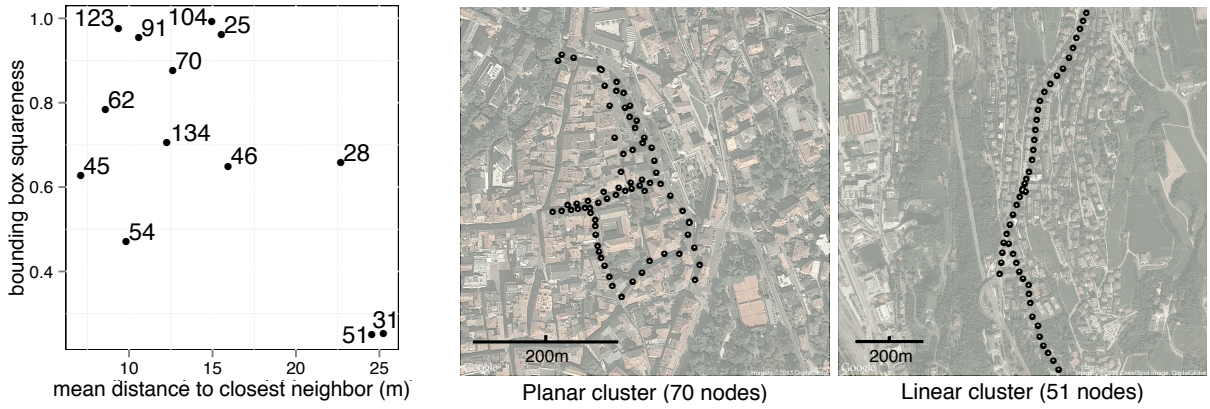
the urban environment determines peculiar topologies (e.g., multi-hop lines mixed with loops) and density patterns found neither in regular grids or random topologies commonly used in simulation nor in the indoor office networks commonly available in public testbeds. Interestingly, smart cities were among the reference scenarios [**?**] motivating the requirements set for RPL (§1).

For these reasons, we use these smart city topologies as a reference throughout the paper. In §4 we use them to quantitatively analyze the problems of downward routing in RPL, and identify the key limiting factor: the tradeoff between the memory necessary to store network state (routing and neighbor tables) and the network scale. In §7–§9 we instead use these topologies to evaluate the effectiveness of our solutions to these problems, in comparison to the original RPL implementation as well as alternate approaches. However, we show that our results are of general applicability by evaluating them also with regular grids and in an indoor testbed.

We next provide further details about the key characteristics of the smart city reference scenario, and how different networks can be obtained by controlling the simulated noise.

Cluster topologies. Figure 2a offers a bird's eye view of the cluster topologies in our dataset. For simplicity, we identify clusters based on the number of nodes contained, and characterize them using three metrics: *i)* number of nodes in the cluster (point label) *ii)* distance to the closest neighbor, averaged over all nodes (x axis) *iii)* aspect ratio of a bounding box aligned with the largest span, indicating how "linear" a cluster is (y axis). Figure 2b shows two representative examples: a 70-node "planar" topology and a 51-node "linear" one. The latter is much less dense not only because of its linear shape but also due to larger distances between the nodes. Note that a narrow bounding box indicates a linear topology: such is the case for the 31- and 51-node clusters as shown on Figure 2a. A more squared bounding box, however, can still belong to a topology that is essentially linear, with the nodes positioned along a curved line. After analyzing the topologies, we have found the 28-node cluster to be such curvy but linear topology.

Nevertheless, we do not have access to the actual infrastructure deployment, and cannot perform protocol experiments directly on it. Simulation is essentially the only option to perform our evaluation on these topologies. We decided to use Cooja [**?**], which supports both Contiki [**?**] based implementations and others (e.g., TinyRPL) thanks to its hardware emulation feature. Emulated nodes run the exact same code (binary) as deployed nodes would, eliminating the typical problem of distortion in results due to implementation differences between a simulator and real world. Still, the use of simulation has well-known drawbacks, e.g., the approximations made w.r.t. the radio channel. In our study, in the absence of radio models or experimental traces expressly targeting a smart city environment, we resort to the multi-path ray tracing model (MRM) provided by Cooja and commonly used in the literature. It models radio hardware properties like transmission



(a) Topology statistics.

(b) Two representative topologies (note the different scale).

Figure 2: Topology of smart city clusters.

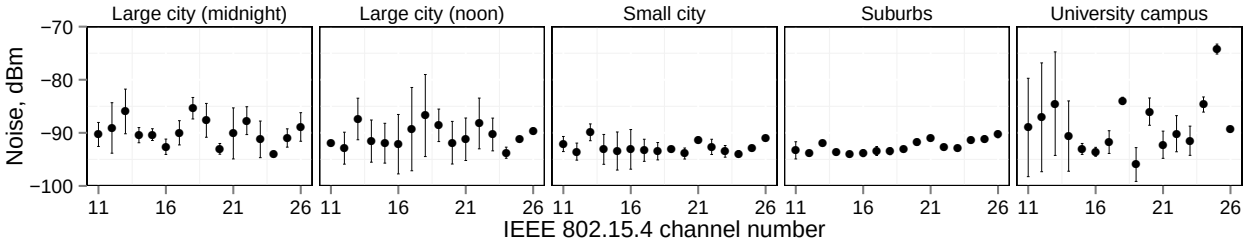


Figure 3: Background noise measured in various urban environments; mean and standard deviation (dBm).

power, sensitivity, and antenna gain, along with the effect of background noise and interference through signal-to-interference-and-noise ratio (SINR), as well as the capture and multipath effects.

Obtaining different network densities. On the upside, simulation allows us to explore many more parameters, to an extent not possible in real-world deployments. For example, using different noise levels enables studying different network densities, all originating from the same physical node deployment. The communication range changes under different noise levels, and so does the underlying network topology; as noise increases, the number of neighboring nodes decreases, resulting in a more sparse topology, and vice versa.

The noise floor in an urban environment can be relatively high, with abrupt short-term variations. Figure 3 shows measurements we performed on all IEEE 802.15.4 channels in several locations of Trento and Moscow, including suburbs, densely inhabited areas, and university campus. The mean noise is between −85 and −95 dBm, with a standard deviation of 0–5 dBm; occasionally, these reach −75 dBm and 10 dBm, respectively. Therefore, in our simulations we set the background noise as in Table 1, equivalent to scaling the density of clusters without modifying the physical location of their nodes.

Table 1: Background noise used in simulation with smart city topologies (dBm).

	noise	
density	mean	stddev
sparse	−80	2
intermediate	−85	2
dense	−90	2

4 The Problem with Downward Routing in RPL

We now analyze why downward RPL underperforms, using the smart city scenario (§3) as a case in point. We focus on the well-known TMote Sky as a representative LLN device, in line with the majority of the related literature. This device has limited *memory* (i.e., 48kB of

program memory and 10kB of RAM), leading to the problems we elicit below. Nevertheless, these problems appear in conjunction with *network scale*; therefore, the results are applicable also to other platforms, where performance problems may appear for a different combination of memory and network scale.

To offer quantitative evidence of the problems we observe, we run Cooja simulations of ContikiRPL over all the network clusters of our smart city scenario, with the topology densities in Table 1. We used the default size of 20 neighbor table entries, and allocated memory to 50 routing table entries. We simulate downward traffic only, where the root sends 500 isolated commands with an inter-message interval of 10s, each destined to a node chosen uniformly at random.

Figure 4 analyzes the simulation results, and specifically the relationship between reliability, memory usage, and the scale and density of clusters. Figure 4a plots the packet delivery rate (*PDR*), i.e., the number of messages received by destinations over the total number of messages sent by the root, against the cluster size. It is evident that as network scale increases beyond a given limit, *PDR* rapidly degrades and becomes unacceptable (<20%). We analyze the reasons next.

4.1 Too Many Routes

In relatively large networks like the clusters of our smart city scenario, some of the nodes, and especially those near the root, may not be able to store route entries for all destinations in their sub-DODAG as advertised via DAO messages by their children. The RPL standard specifies an optional DAO-NACK message with a *Rejection* status code to notify the DAO sender that the recipient is unable to act as a DAO forwarder, but does not define any mechanism to handle this rejection. In popular implementations of RPL, this DAO-NACK is not even sent and an incoming DAO from a new destination is silently dropped. The path is therefore partially built but useless, as the destination is unknown to all upward nodes including the root that, upon receiving an incoming packet for this unknown destination, has no other choice than to drop it.

This is illustrated in the example in Figure 1c where we assume, for the sake of argument, that routing tables are limited to 2 entries. Node E announces itself as a destination by sending a DAO to its preferred parent, as D did previously (Figure 1b). Upon receiving this DAO from E, C checks whether it can be added to the routing table; in this case, free memory allows C to accept the message and forward it to the preferred parent, $B1$. The latter, however, has both its routing entries occupied by C and D; therefore, it cannot store this new destination, and drops the DAO. Even if $B1$ were to send a DAO-NACK message to C to inform it of the situation, the behavior of C would be undefined according to the RPL standard. As a result, the root has no knowledge about how to reach E, and drops all packets destined to it.

Comparison of Figure 4a and 4b in the sparse scenario shows that indeed routing tables are the culprit for the degradation of reliability. *PDR* is near perfect until the cluster size (and therefore potential destinations) remains <50, i.e., the size of the routing table. Beyond this limit, Figure 4b shows (top) that the fraction of nodes with rejected DAOs jumps from zero to 15–20%. Figure 4b (bottom) also confirms that the problem is exacerbated for root neighbors; as the root must store all destinations, it is the first node to fill its routing table. However, the chart shows that several nodes away from the root also experience rejections; their number depends on the specific characteristics of the cluster topology. Similar trends hold for the intermediate density scenario.

As network *density* increases, however, the correlation of reliability with the number of nodes with a full routing table no longer holds. Figure 4a shows that *PDR* degrades earlier and more sharply in the dense topology. Nevertheless, Figure 4b shows that only the three biggest clusters contain nodes with full routing tables. Indeed, in dense scenarios the excessive number of routing entries is overshadowed by another issue, described next: the excessive number of neighbors.

4.2 Too Many Neighbors

ContikiRPL maintains information about nodes directly reachable in a dedicated neighbor table, whose presence is assumed in RPL implementations [**?**]. If an entry for a neighbor is not present in the table, forwarding towards it cannot happen. This is not a problem for upward traffic (e.g., data collection), since a node forwards packets to its preferred parent, and implementations guarantee that the corresponding entry

always remains in the neighbor table. However, it is a problem for the (many) downward routes constructed via DAO messages. A closer look at the process of handling a DAO for a new destination shows that, before checking the routing table and accepting the message, the receiving node must first check if the DAO sender is in the neighbor table. If it is not, and the neighbor table is full, the DAO packet is silently dropped. Notably, in this case a DAO-NACK is not even an option, again because the target node (the DAO sender) is not in the neighbor table.

The magnitude of the problem can be seen in Figure 4c, plotting the fraction of nodes with a full neighbor table. The latter is allocated 20 entries, the default for ContikiRPL on TMote Sky. As density increases, and the node degree approaches the size of the neighbor table, the fraction of nodes unable to store more entries approaches the totality of nodes. This phenomenon is responsible for the drop in *PDR* in dense scenarios, even for cluster sizes where routing tables are not full.

4.3 Summary

The relationship between memory size and network scale, reified in the size of routing and neighbor tables, is crucial in ensuring the reliability of downward traffic in RPL.

Although we elicited this issue in the context of the specific implementation we considered, ContikiRPL, these seemingly low-level details are representative of common choices made by state-of-the-art systems, bearing far-reaching consequences on route signaling. Nodes fail at constructing and maintaining downward paths as DAO messages get silently dropped, effectively rendering some network nodes unreachable. This compromises reliability and scalability, and ultimately clashes with the original goal of RPL, i.e., supporting large networks of resource-scarce devices.

These limitations may appear as poor implementation choices that, once replaced by better ones, could significantly ameliorate the situation. This is the spirit of the several mechanisms we introduce in the next section, to significantly improve the performance of RPL by making better use of the scarce resources available on the nodes. However, we also show in the rest of the paper that standard-compliant mechanisms only marginally ameliorate the situation, pointing to intrinsic limitations of the RPL *standard* w.r.t. reliability and scalability. We overcome these limitations with alternate dissemination-based mechanisms that, albeit non-standard, can be easily integrated in RPL without disrupting its core approach, as described next.

5 Rethinking RPL Downward Routing: Route or Flood?

The analysis in §4 shows that the memory problem appears on the path from the leaves to the DODAG root, when upward propagation of DAO messages is suddenly stopped. We identify two points in the protocol logic where measures can be taken to re-establish communication:

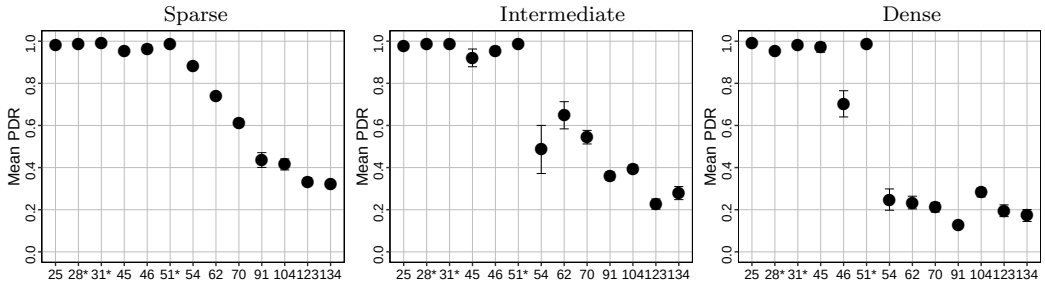
- when network nodes receive a DAO-NACK (rejection) from their DAO parent;
- when a packet with unknown destination is received at the DODAG root.

Consequently, two kinds of solutions can be envisioned:

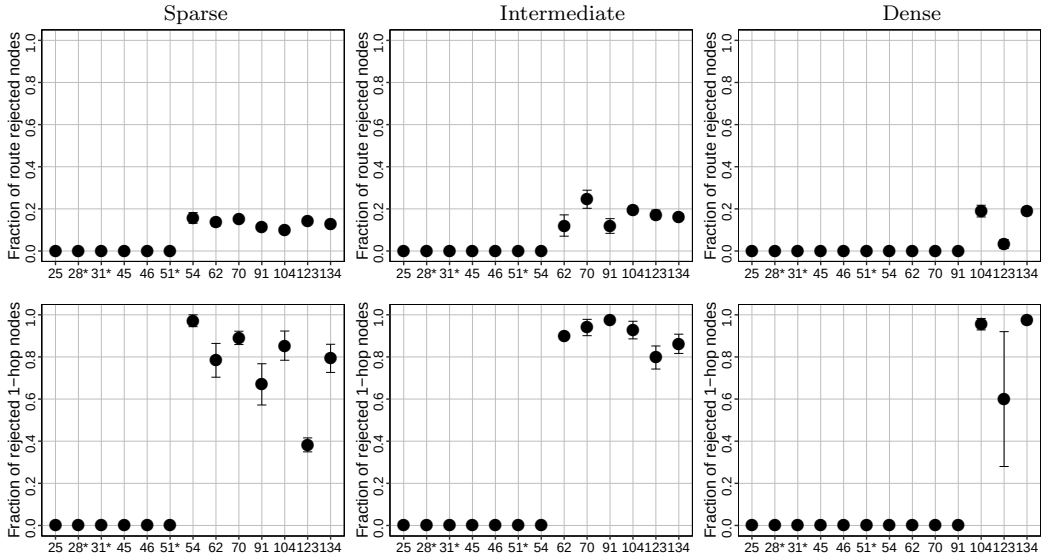
- *reactive* mechanisms that handle the DAO-NACK by attempting to find an alternative DAO parent, re-establishing a *route* towards the root;
- *forwarding* mechanisms that, in the absence of a valid route, *disseminate* the packet in an attempt to bypass the areas of the network in which the destination is unknown.

This paper revolves around several solutions stemming from the above observations. In particular, we introduce three base mechanisms:

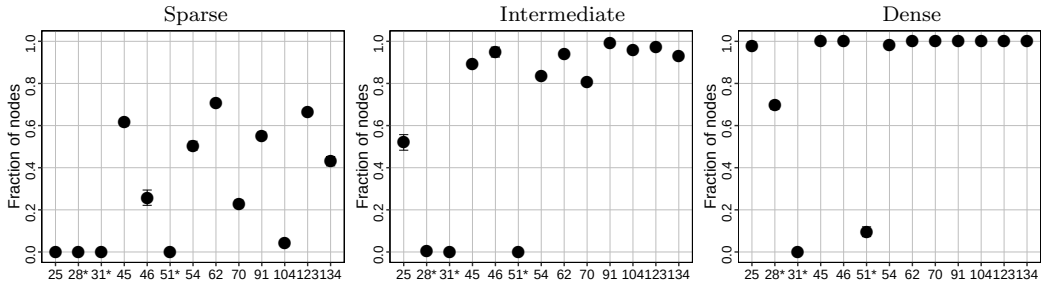
1. SWITCH: a pure reactive mechanism in which nodes switch to a different DAO parent when rejected by the current one;



(a) Packet delivery rate, *PDR*.

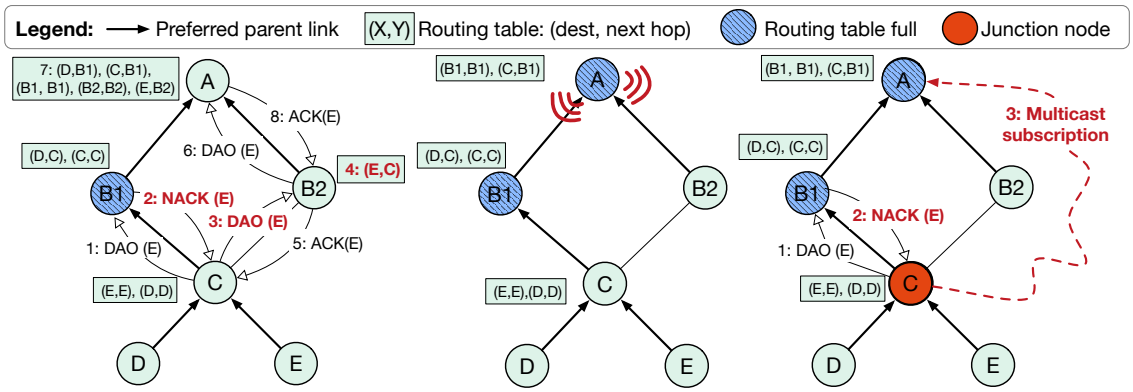


(b) Routing table: fraction of nodes rejected by their DAO parent due to a full routing table (top) and fraction of these rejected nodes that are neighbors of the root (bottom).



(c) Neighbor table: fraction of nodes with a full table.

Figure 4: Tradeoff between reliability and memory necessary to store the network state in the smart city scenario (§3). Here and throughout the paper, results for this scenario report the size of each of the 13 clusters on the x-axis; a '*' next to the size denotes a linear topology.



(a) C tries another parent; It will remember B1 for D and B2 for E.

(b) Root broadcasts all data packets without destination.

(c) C joins the multicast group and becomes a junction node.

Figure 5: The three mechanisms proposed for downward traffic: a) SWITCH, b) ROOT, and c) MCAST.

2. ROOT: a pure forwarding mechanism in which packets with unknown destinations received at the root are simply broadcast to 1-hop neighbors, increasing the chances of packets reaching a node with a route to the destination;

3. MCAST: a hybrid mechanism in which *i)* rejected nodes react by joining a special multicast group, whose address is then *ii)* used by the root to disseminate packets with unknown destinations; upon reaching a rejected node, packets resume the normal route-based forwarding.

These three mechanisms are illustrated in §5.1-§5.3, while §5.4 shows that their combination strikes alternative trade-offs unattainable by each mechanism alone.

5.1 switch: Finding a New Route

Although the RPL standard allows nodes to use multiple parents for routing, existing implementations use only one—the preferred parent—both for upward routing and downward signaling (i.e., DAO messages). In contrast, SWITCH takes advantage of the DODAG topology by enabling a node to use parents other than the preferred one whenever necessary to create alternative downward routes. When a rejection is received from the preferred parent, a node *reacts* by sending DAOs to each of the others, until it finds a parent (if any) with neighbor and/or routing entries available.

Example. In Figure 1c, node C forwards a DAO to $B1$ containing destination E; however, $B1$'s routing table is full and the packet is dropped. In contrast, Figure 5a shows that in SWITCH, after $B1$ sends a DAO-NACK, C can try another parent by sending the DAO to $B2$. This node has available memory and therefore accepts to serve as forwarder towards E; $B2$ does so by sending a positive acknowledgement to C and forwarding E's DAO to the root. Hereafter, and until network conditions change, C has two parents; it forwards DAOs for destination D to $B1$ and those for E to $B2$.

Pros & Cons. SWITCH takes advantage of the DODAG topology already maintained by RPL, and is *standard compliant*. However, besides these major assets it also has limitations:

- the root does not have parents, therefore SWITCH is not applicable to it when the neighbor or routing table becomes full;
- the nodes close to the root are likely to quickly fill their routing tables, therefore finding good alternatives may become difficult;
- the same holds in sparse networks, due to inherent lack of neighbors;

- the routing cost and/or link quality of the route via the secondary parents may be worse than with the preferred parent, leading to a decrease in reliability.

5.2 root: Overcoming the Memory Bottleneck via Broadcast

As we discussed in §4, the root must keep an entry in its routing table for each downward destination in the network, and can therefore become a memory bottleneck. However, due to the hierarchy in the DODAG topology, this is not necessarily true for the root's neighbors. A destination unknown to the root may be known to one of its neighbors, from which a complete route to the destination exists; therefore, if a packet were able to bypass the root bottleneck, it could be routed to destination.

This is precisely what the ROOT mechanism achieves with a simple modification of the RPL *forwarding* rule at the root. If the packet destination is unknown, i.e., there is no entry for it in the routing table, the root simply performs a link-level broadcast of the packet to its neighbors. Otherwise, the root forwards the packet via unicast as usual, i.e., based on the routing table.

Consistently with this strategy, DAOs are never rejected by the root, effectively *creating the illusion that the root has neighbor and routing tables of infinite size.*

Example. Consider Figure 5b, where all nodes have a routing table limited to 2 entries. In the situation depicted, a packet not destined to $B1$ or C would be dropped at the root, as no other routes exist. The ROOT mechanism allows instead to bypass the root and move the forwarding decision to its neighbors. In the specific case, a packet destined to D would be received by $B1$ and consequently routed to destination via C. Moreover, a packet destined to $B2$ would be obviously received directly. On the other hand, the figure also shows how ROOT cannot solve all problems by itself; a packet destined to E would still be dropped at $B1$ and $B2$, since neither node has a route towards E.

Pros & Cons. The example highlights the main tradeoff in applying ROOT. On one hand, it is effective at relieving the memory problem at the root, consequently avoiding rejection of its neighbors. On the other hand, it does not ameliorate the situation of nodes beyond its neighborhood, which may be many (§4). Furthermore, IEEE 802.15.4 layer-two broadcast transmissions are not acknowledged, yielding a higher probability of packet loss w.r.t. the unicast ones normally employed.

5.3 mcast: Combining Route Recovery and Dissemination

As the name suggests, MCAST[5] relies on a special multicast group used by nodes to *react* to a DAO rejection *and* by the root as an alternate way to *forward* packets with unknown destinations.

This hybrid mechanism relies on DAO-NACKs to inform a node that it failed to advertise a downward route for itself or another destination in its sub-tree, as in SWITCH. However, in MCAST a rejected node reacts by joining the multicast group above, which contains all nodes whose DAOs have been similarly rejected by their parents. Note that it is not the DAO destination that joins the multicast group, but the node unable to forward the corresponding DAO, as discussed next.

The multicast group provides the root with a way to forward a packet with unknown destination to nodes that potentially have a route to it, as in ROOT. However, in MCAST the forwarding scope is no longer constrained to 1-hop neighbors in physical range, rather it extends in multi-hop to potentially anywhere in the network. As in ROOT, unicast forwarding is resumed when the multicast packet reaches a node that knows the original packet destination, i.e., it has a downward route to it.

We call the nodes in the multicast group *junction nodes*, as they are at the crossroads of multicast and unicast forwarding[6]. A junction node enables the delivery of packets from the root to all destinations in its routing table, including those for which it received a DAO-NACK. Note that a junction node may serve several rejected destinations. In this case, the junction node joins the group upon receiving the first DAO-NACK, but keeps track (using a 1-bit flag) of all destinations served. This information is used to know which

[5] A very preliminary design of MCAST appeared with the name D-RPL in [**?**].

[6] In a sense, the root neighbors in ROOT can also be considered junction nodes.

Table 2: Benefits and limitations of the proposed mechanisms.

	Pros	Cons
SWITCH	local, low-traffic repair of the DODAG topology	options for route recovery may be few and/or worse
	standard-compliant	not applicable at the root
ROOT	enables forwarding bypassing incomplete routes	relies on link-level broadcast, efficient but unreliable
	never rejects DAOs from root neighbors	applicable only at the root
MCAST	enables forwarding bypassing incomplete routes	potentially increases traffic overhead
	applicable to the entire network	

downward unicast routes begin at the junction node, i.e., which destinations are missing an upward path towards the root.

The junction node periodically re-sends DAO advertisements for its served destinations, in the background; some of them may be eventually accepted by the DAO parent, which may have freed memory due to changes in the RPL topology. If and when *all* destinations are accepted by the DAO parent, i.e., the junction node no longer has rejected destinations, it leaves the multicast group.

Apart from the 1-bit flag per routing entry, MCAST also requires state for handling the multicast group. The amount of memory required depends on the multicast protocol used; in our case, it amounts to adding a single route entry at the root and at the intermediate forwarding nodes (§6).

Example. Figure 5c illustrates the advantages of MCAST. In the example, node C learns via a DAO-NACK that its DAO advertisement for destination E has been rejected by $B1$. As such, C is now the highest node in the DODAG that knows about E, and therefore has the critical role of ensuring that it is reachable from the root, which is accomplished by C subscribing to the multicast group collecting rejected nodes. When the root has a packet for E, or any other destination for which it does not have a route, the root simply sends it to the multicast group; in the example, the packet will reach C via multicast, from where it will be forwarded to E based on the routing table.

Pros & Cons. MCAST reacts to a DAO rejection by enabling the root to bypass the area in which the rejected destination is unknown; this area can extend (and shrink) dynamically and arbitrarily through the network, thanks to the reliance on a multi-hop multicast layer.

The latter enabling factor, however, is also the main drawback of MCAST. Packets with an unknown destination are disseminated to all junction nodes, although only one has a downward route. If not properly mitigated, the use of multicast may increase the communication overhead, and consequently the energy consumption of the nodes.

5.4 Exploiting Multiple Mechanisms in Synergy

The three mechanisms we illustrated are based on different principles and tackle different aspects of the problem; they are essentially complementary to each other. Nonetheless, they also provide different tradeoffs, summarized in Table 2; therefore, a natural question is whether and how their strengths and weaknesses can be combined synergycally to improve the overall performance.

In this section, we concisely present four new protocols that encompass different degrees of route- and flood-based approaches, in an effort to find the best balance between their benefits and their drawbacks: three are the pairwise combination of the base mechanisms we just outlined, while the fourth includes them all. The relationship of these combinations w.r.t. the base protocols is summarized in Table 3. In §7, we quantitatively show that, indeed, the combined mechanisms provide better performance than the individual ones alone.

switch + root. The parent switching mechanism provided by SWITCH is not applicable at the root; dually, the 1-hop forwarding provided by ROOT is only applicable at the root. The fact that these two mechanisms are sharply complementary leads naturally to a simple implementation in which SWITCH runs on all nodes other than the root, which instead executes ROOT.

switch + mcast. The MCAST mechanism provides significant benefits at the price of a potential increase in network traffic, directly depending on the number of junction nodes in the multicast group. A junction

Table 3: Proposed mechanisms, and their combination.

	parent switching	1-hop broadcast at the root	multicast group for rejected nodes
SWITCH	X	—	—
ROOT	—	X	—
MCAST	—	—	X
SWITCH+ROOT	X	X	—
SWITCH+MCAST	X	—	X
ROOT+MCAST	—	X	X
T-RPL	X	X	X

node leaves the group only if the DAO parent somehow frees some memory, therefore finally accepting the destination being advertised. Replacing this default periodic advertisement towards the DAO parent with the local route recovery provided by SWITCH may help by rapidly finding alternative parents for the destinations managed by a junction node, enabling the latter to leave the multicast group and keep the corresponding traffic to the necessary minimum.

root + mcast. Another way to reduce the number of nodes in the multicast group, and therefore the traffic in MCAST, is to eliminate the junction nodes that are neighbors of the root. This can be accomplished by exploiting ROOT, and its ability to never reject a DAO from a root neighbor. Nevertheless, ROOT and MCAST implement different behaviors at the root; their combination cannot be simply achieved by choosing one or the other. In our combined protocol, nodes other than the root follow MCAST, while the root performs the following processing on a downward packet, escalating to increasing levels of dissemination if and when the packet cannot be forwarded:

1. it checks the packet destination against the routing table. If one exists, the packet is forwarded in unicast towards the next hop and processing ends; otherwise
2. it broadcasts the packet. If a corresponding layer 3 acknowledgment is received within a given timeout the processing ends; otherwise
3. it multicasts the packet on the special group containing junction nodes.

Step 1 is the default RPL behavior constituting the starting point for both ROOT and MCAST to exploit the routing table whenever possible. Step 2 is similar to ROOT, and is key in reducing the reliance on the multicast group (i.e., the number of destinations enabled only via MCAST), as it enables root neighbors with a downward route to a destination to use it, without the overhead induced by multicast. Multicast is triggered by a missing acknowledgment—a salient difference w.r.t. ROOT—and effectively provides the "safety net" for downward packets only if and when necessary. This layer 3 ACK is indeed not used to acknowledge the reception of the broadcast packet, rather as a confirmation that a route towards the destination was found.

switch + root + mcast = T-RPL. In addition to the pairwise combination of our base mechanisms illustrated thus far, we also design a protocol, T-RPL, *that combines them all*.

The intuition behind this design choice is that each base mechanism solves one aspect of the problem. What happens if all three mechanisms are present in the same protocol? Are they going to synergically complement each other and provide the best tradeoff or, instead, interfere with each other and degrade performance w.r.t. the individual mechanisms or their pairwise combinations?

Of course, this also depends on *how* their combination is performed. We design T-RPL as follows:

- the root behaves as in ROOT+MCAST, exploiting its forwarding strategy that escalates dissemination only if and when needed;
- the other nodes behave as in SWITCH+MCAST, exploiting its ability to attempt local route repair while providing an alternative means to ensure reachability from the route via multicast.

The result is a protocol that retains the routed approach at the core of RPL, represented by SWITCH, but caters for its limitations by complementing it with the dissemination offered by ROOT and MCAST. The

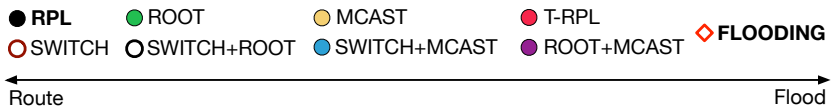


Figure 6: Covering the design space between pure routing and pure flooding.

Table 4: Memory usage of the protocols, in KiB.

Protocol	Code (ROM)	Δ(%)	Data (RAM)	Δ(%)
ContikiRPL	40.9	—	7.8	—
SWITCH	42.1	+2.9	8.7	+11.5
MCAST	43.3	+5.9	8.2	+5.1
ROOT	40.9	0	7.8	0
T-RPL	44.6	+9.0	9.3	+19.2
FLOODING	41.5	+1.5	6.7	-14.1

combination of routing and flooding in T-RPL is hitherto unexplored in the literature and, as we show in the rest of the paper, indeed achieves the best tradeoffs along several dimensions. Nevertheless, it is worth recalling that the contribution of the paper goes beyond the specific decisions we made in T-RPL, and points instead to the general exploration of a balanced combination of routing and flooding as a viable and effective approach for protocol design.

5.5 Summary

We presented three mechanisms and their combinations that, once integrated into RPL, can help improve its downward routing by exploiting a different mix of routing and dissemination. Figure 6 pictorially illustrates the concept by showing where each solutions lies w.r.t. two extremes: pure RPL routing and pure flooding. SWITCH provides a standard-compliant enhancement to RPL routing that does not rely at all on dissemination. ROOT and MCAST, on the other hand, rely on increasing degrees of dissemination to bypass areas where downward routes are not known. Bringing this concept to an extreme results in pure flooding, where downward routes do not exist and each packet is simply disseminated to the entire network.

These two extremes, pure RPL and pure flooding, are natural baselines for the experiments in §7 comparing the performance of our solutions, whose key implementation details are discussed next.

6 Implementation Highlights

We implemented the solutions discussed in §5 as extensions to ContikiRPL, the popular RPL implementation distributed as part of ContikiOS. Our modifications affect only the routing logic supporting downward traffic, and leave upward routing (e.g., for data collection) entirely unaffected. Next, we present the salient aspects of our implementations, whose memory footprint is shown in Table 4 for the configuration used in our experiments (Table 8).

switch. The SWITCH mechanism relies on DAO-NACKs, which are defined by the RPL standard to carry a rejection status. Accordingly, we define a status *Out of memory* that is sent in response to a DAO by a receiver that has reached its neighbor and/or routing table capacity. Further, we modified ContikiRPL as follows:

- the entries in the parents table include a new flag denoting whether the corresponding parent already replied with a DAO-NACK;
- the routing table contains a new field storing the DAO parent for each destination, effectively decoupling it from the preferred parent used for upward traffic.

As mentioned in §4.2, Contiki requires the address of a packet recipient to be in the neighbor table; if the latter is full and does not contain the DAO sender, it is impossible to send a DAO-NACK back to it.

To overcome this constraint, we reserve R neighbor table entries to temporarily store the addresses of DAO-NACK recipients; these entries are filled when a DAO is received from an unknown neighbor and removed right after sending the DAO-NACK. After several experiments, we determined that $R = 4$ offers the best reliability; this setting is assumed hereafter.

root. This mechanism requires only simple modifications at the root, to complement the default unicast forwarding used for known destinations with the link-layer broadcast used for unknown destinations. The latter consists of a MAC-level broadcast of the packet, which retains its IP destination, enabling resuming of the default routing at the receivers with a downward route. Consequently, ROOT does not affect memory consumption, as shown in Table 4.

mcast. At the other extreme, MCAST is the most complex mechanism as it relies on a multicast layer. Two well-known ones already interoperating with RPL are included in the Contiki distribution: MPL (Multicast Protocol for Low power and Lossy Networks) [**?**] and SMRF (Stateless Multicast RPL Forwarding) [**?**]. However, MPL essentially floods the entire network, and therefore clashes with the design principle at the core of MCAST and yields unnecessarily high overhead. Instead, SMRF relies on a multicast tree limiting the dissemination scope and greatly reducing traffic. Other protocols (e.g., BMRF [**?**]), although not readily available in Contiki, could easily support MCAST.

In SMRF, nodes keep no per-packet state and only minimal routing state: a single entry per multicast group, i.e., one in our case. A node joining the multicast group generates a multicast DAO message, propagated upwards in the DODAG. The nodes receiving the message further propagate it upwards, after storing the corresponding multicast address in their routing table and therefore becoming forwarders for the group even if they did not join it. Multicast messages are sent by the root, and re-propagated by all nodes that have an entry for the corresponding multicast address.

In the context of MCAST, support for an arbitrary number of unknown destinations can be added by reserving a single routing table entry for a multicast address dedicated to the group of nodes that do not have enough memory to store all downward destinations. At the root, this means that the packets with unknown destination must be sent using IP multicast instead of unicast. In our implementation, this is achieved by relying on a special MCAST IPv6 extension header which contains the original destination address, and it is inserted after the RPL hop-by-hop option header. Alternatively, the IPv6 destination option header or generic IPv6 tunneling could be used. Upon reaching a junction node, the packet is stripped of this special header and the original destination address is restored, resuming the default downward routing. Other modifications to ContikiRPL involve the bookkeeping necessary to keep track of the destinations a junction node is managing (i.e., it has received a rejection for), which determines whether the node should join, remain, or leave the multicast group. Although MCAST is the most complex of our three base mechanisms, its memory overhead is relatively limited, as shown in Table 4.

T-RPL and other combinations. The implementation of the combinations we outlined in §5.4 descends naturally from the base mechanisms and their implementations. The only significant exceptions are T-RPL and ROOT + MCAST, which introduce an acknowledgement used by a neighbor that has a downward route to the packet broadcasted by the root. This acknowledgement informs the root that escalating to multicast forwarding is not necessary, therefore limiting the dissemination of multicast packets. This acknowledgment is defined as a new ICMPv6 message.

Obviously, the combination of multiple mechanisms has a negative effect on memory consumption (Table 4), although this could be mitigated by an optimization on routing entries (§7.4).

flooding. Finally, we outline the implementation of the pure flooding we use as the second baseline in addition to ContikiRPL, as discussed in §5.5. Hereafter, FLOODING refers to the specific protocol we use, in which nodes do not keep any routing table for downward traffic. The root initiates the flooding of a new packet via broadcast to its neighbors; whenever a node receives a packet never seen before, it rebroadcasts it once after a short random delay. There are no MAC-level acknowledgements and re-transmissions; a time-to-live (TTL) is associated to messages to limit their rebroadcasting. Duplicates are filtered by checking against a cache of packet identifiers; a history of the last 100 received is accommodated in the data memory originally reserved by RPL for downward routes. The neighbor table is managed (and configured) as in the

other protocols.

In principle, the protocol we concisely described entirely replaces the RPL logic for downward traffic, notably including DAO dissemination. However, the latter is deeply intertwined with the logic for upward routing. Therefore, in our implementation we *added* support for FLOODING while leaving the code for DAO dissemination in place, though disabled during our experiments. This explains why we report a larger code size for FLOODING w.r.t. the original ContikiRPL, despite the fact that the downward logic of the former is significantly simpler.

7 Evaluation of Proposed Mechanisms and their Combinations

This section presents a detailed evaluation of the proposed mechanisms (SWITCH, ROOT, and MCAST) and their combinations (SWITCH+ROOT, SWITCH+MCAST, ROOT+MCAST, and T-RPL) as presented in §5, and summarized in Table 3 and Figure 6. We begin by describing our simulation setup and performance metrics. Then, we present the evaluation results, first on all the network clusters of our smart city reference scenario, and then on regular grid topologies with up to 225 nodes. Our baselines for comparison are ContikiRPL and FLOODING that, as shown in Figure 6, are at the extremes of the spectrum between route-based and flood-based approaches we consider.

7.1 Simulation Setup and Performance Metrics

Simulation tool. We used Cooja [**?**], a very accurate network simulator used by many works concerned with LLNs. Cooja enabled us to run directly the binaries of our protocols compiled for the TMote Sky platform, for which Cooja provides instruction-level emulation. The radio propagation was modeled by the multi-path ray tracing model (MRM) built into Cooja. We ran 20 repetitions for each simulated configuration. The evaluation results presented in this and the following sections are based on 52851 simulations and 23.5 years of CPU time, sometimes occupying over 500 cores in parallel for days. Nevertheless, we ran many more simulations to identify the relevant portions of the configuration space and verify the tradeoffs in all combinations; although not all results are shown here, due to readability concerns, these amount to 120704 individual simulations for a total of 168 CPU years, further reinforcing the confidence in the results we report.

Network topology. We use as a reference the realistic smart city scenario presented in §3, containing 864 nodes grouped into 13 clusters. Network density significantly affects protocol performance, as shown later in the paper; however, it depends not only on physical node density, but also on radio propagation characteristics. As in §3, we resort to noise levels as a simple, effective, and yet realistic means to condense this dimension of the simulation space into a single parameter, and use the sparse, intermediate, and dense networks resulting from the noise levels in Table 1.

Table 5 offers metrics characterizing the resulting topologies. Among these metrics we report also the average, minimum and maximum of the per-node sum of the *outgoing* link *PDR*. These values are useful to characterize the quality of communication in a node neighborhood. For instance, in the sparse scenario the average node degree is 37; however, the average sum of link *PDR* is only 11, showing that many of the neighbors are poorly connected.

The above 13 clusters represent a large variety of topologies, that are however all derived from the same smart city scenario. Therefore, to generalize our findings we also present results derived with synthetic topologies in the form of *regular* grids with $N \times N$ nodes, $N \in \{3, 5, 7, 9, 11, 13, 15\}$, with the root in the center. This also allows us to experiment with networks with up to 225 nodes, twice the largest smart city topology. We select the grid step (50m) and radio propagation properties (mean noise -90dBm, standard deviation 2) to obtain an intermediate density comparable to the one in the smart city scenario; for instance, the synthetic topology with 121 nodes has similar characteristics to the smart city one with 134 nodes (Table 5).

In principle, we could have used even larger networks. However, we verified that this would yield unacceptable performance of the basic operation of the upward RPL typically used for data collection, that we take as granted in this work. Figure 7 shows the results of our simulations of ContikiRPL with only upward

Table 5: Topology metrics for sample planar smart city topologies and synthetic grids.

	smart city									synthetic	
Topology density	sparse			intermediate			dense			intermediate	
#nodes	70	91	134	70	91	134	70	91	134	121	225
Avg degree	30	49	37	57	83	82	68	90	125	82	101
Max degree	48	76	62	69	90	116	69	90	133	118	161
Min degree	8	21	6	27	64	24	62	90	77	47	44
Avg sum of link *PDR*	11	16	11	24	37	28	46	69	63	27	29
Max sum of link *PDR*	17	26	21	36	55	44	59	83	87	36	38
Min sum of link *PDR*	3	3	1.6	7	13	5	20	43	20	12	11
Avg hops	3.6	3	5.3	2	1.6	2.6	1.2	1.1	1.6	2.3	3
Max hops	8	6	11	4	3	5	3	2	3	4	5
Avg path ETX	4.5	3.8	7.5	2.4	2	3.2	1.4	1.2	1.8	2.7	3.6
Max path ETX	11	8.2	22	5.2	4	6	3.1	2	3	4.7	6.3

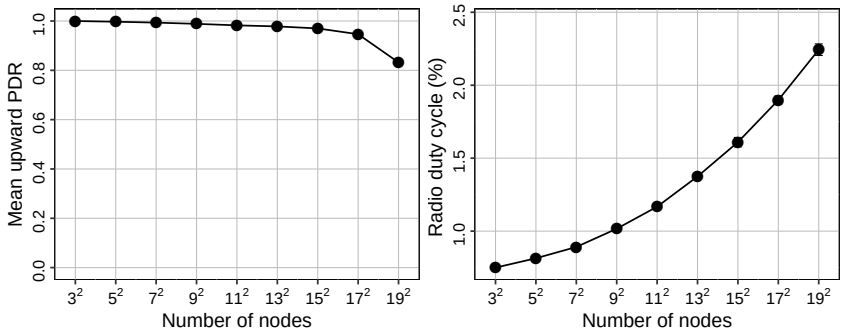


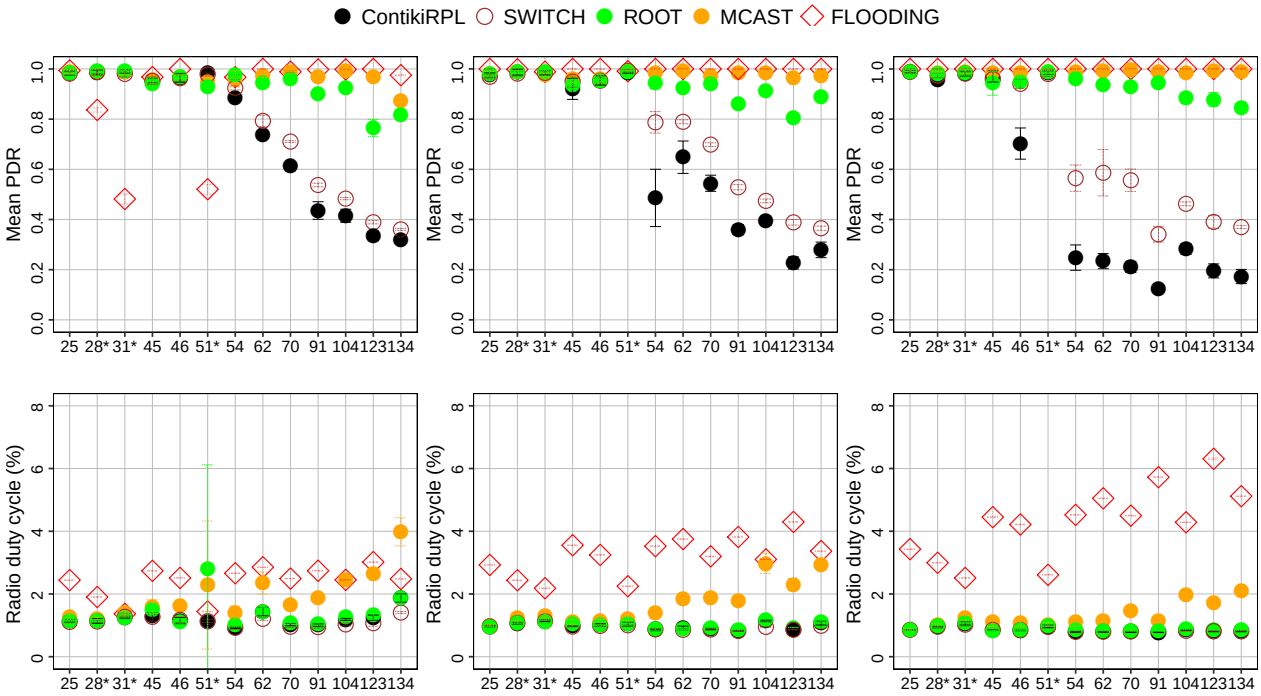
Figure 7: Reliability and duty cycle evaluation of ContikiRPL on synthetic topologies in the presence of upward traffic with an inter-message interval of 3 minutes.

collection traffic, an inter-message interval of 3 minutes (the same used in §9), and DAO messages enabled but no downward traffic. The *PDR* remains close to 100% for $N = 15$ but decreases abruptly after, reaching 80% with 361 nodes. This is due to the fact network becoming overloaded because *i)* the larger the network, the more packets are generated, and *ii)* the topology maintenance of ContikiRPL causes frequent DODAG reconfigurations increasing the number of DIO and DAO messages. Consequently, we consider networks with at most 225 nodes.

Protocol configuration. Table 8 presents the most important protocol settings used in our study. For ContikiRPL and its variants we use the Minimum Rank with Hysteresis Objective Function (MRHOF) [**?**] and Expected Transmission Count (ETX) routing metric, as these are the default and a popular choice in the literature. Similarly, we retain the default setting of 20 entries for the neighbor table, and set the routing table to a maximum of 50 entries, therefore almost filling the remaining RAM of the TMote Sky platform. Finally, we use ContikiMAC as the Media Access Control layer, configured with the default wake-up interval of 125ms.

Application traffic. We compare protocols using a downward traffic mimicking actuation commands sent by the root. Messages have a 6B payload, enough to fit a command code and 1–2 parameters. In each experiment, after a warm-up time (10 minutes) needed to stabilize the routing topology, the root sends 500 isolated commands, each destined to a node chosen uniformly at random and with an inter-message interval (IMI) of 10s, enough for a command to propagate to destination without interfering with the next. Indeed, actuation commands and, in general, downward traffic require higher reliability (§1) than upward traffic but are typically less frequent; we analyze the interaction with background traffic in §9.

Performance metrics. We measure and report about the following quantities:



(a) Sparse (b) Intermediate (c) Dense

Figure 8: Performance of the base mechanisms in the smart city reference scenario, under different densities.

- *Packet delivery ratio (PDR)*: the ratio of the valid packets received by intended destinations over the total number of packets sent by the root. This a measure of the protocol *reliability*.
- *Radio duty cycle (DC)*: the ratio between the time a node spends with its radio active (transmitting or listening) over the time interval under consideration. The *DC* value is directly reported by Cooja and is a proxy for the protocol *energy consumption*.

We also investigated the end-to-end delay. However, results are not insightful, as the various alternatives exhibit similar performance; in the interest of space and readability we omit them.

The charts shown hereafter report average values along with error bars denoting the 95% confidence interval.

7.2 Smart City Topologies

We begin our evaluation by analyzing how the base mechanisms in §5.1–§5.3 perform individually when compared to the baselines, in the smart city reference scenario. We then evaluate their combinations in §5.4 in the same scenario, to further explore tradeoffs among performance metrics.

Base mechanisms. Figure 8 shows the performance of SWITCH, ROOT, and MCAST under different topology densities. Table 6 offers an alternative view on *PDR*, enabling us to "zoom in" on values that are otherwise difficult to discern in Figure 8.

As already noted in Figure 4a, the performance of ContikiRPL degrades when the network size exceeds 50 nodes, due to the limited space in the routing and neighbor tables (Table 8), with a *PDR* as low as 20% in dense networks.

SWITCH improves the *PDR* of RPL in all topologies, while yielding the same low *DC* as ContikiRPL. Still, this improvement is not significant in all cases, as the *PDR* is bounded by the maximum number of routes that the root can keep (50 in our case) and by the link quality to alternative parents. As density

Table 6: *PDR* (%) for all smart city clusters under different network densities.

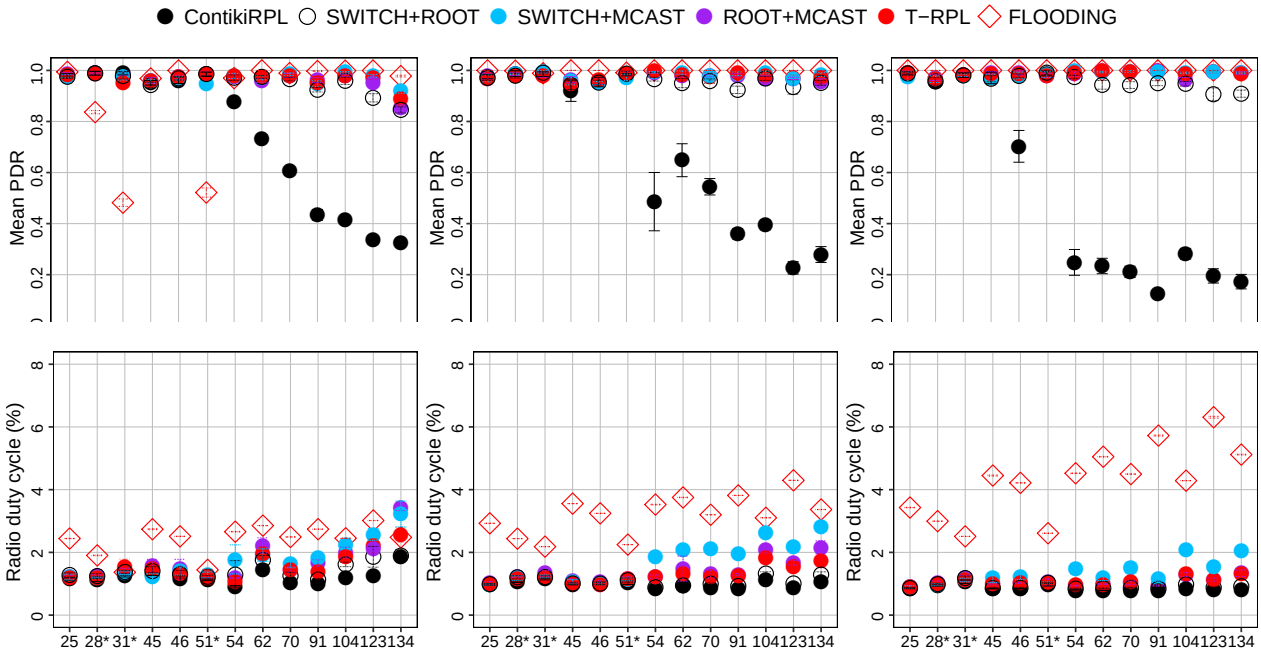
sparse													
	25	28∗	31∗	45	46	51∗	54	62	70	91	104	123	134
ContikiRPL	98.42	98.65	99.06	95.41	96.47	98.45	88.37	73.76	61.24	43.59	41.49	33.36	32.06
SWITCH	98.15	98.85	98.03	95.27	96.31	97.81	92.46	79.31	71.06	53.77	48.33	38.92	36.06
ROOT	98.58	**99.41**	**99.20**	94.08	97.04	92.77	97.72	94.31	95.98	90.18	92.37	76.46	81.72
MCAST	98.98	98.75	98.71	95.61	96.88	95.08	96.06	97.44	**98.71**	**97.02**	99.33	97.03	87.30
SWITCH+ROOT	97.68	99.31	98.18	93.52	97.15	98.85	95.83	**97.54**	97.00	92.47	96.49	89.32	84.60
SWITCH+MCAST	97.61	99.29	98.78	95.91	96.49	91.80	**98.11**	97.13	98.40	95.00	**99.56**	**98.44**	**92.37**
ROOT+MCAST	**99.04**	98.88	98.93	**96.40**	**97.59**	**99.59**	97.71	95.78	98.10	96.26	97.85	94.72	85.09
T-RPL	98.16	98.86	92.14	95.87	97.21	98.60	98.08	97.52	97.89	94.94	98.16	97.16	89.28
FLOODING	99.38	83.68	48.17	96.77	99.97	52.06	96.72	100	98.89	99.79	99.80	99.98	97.53
intermediate													
	25	28∗	31∗	45	46	51∗	54	62	70	91	104	123	134
ContikiRPL	97.58	98.72	98.49	92.09	95.43	98.45	48.61	64.83	54.44	35.97	39.46	22.67	27.89
SWITCH	96.89	98.16	98.44	94.98	95.32	99.29	78.73	78.95	69.86	52.86	47.42	38.88	36.53
ROOT	**98.05**	98.75	98.76	93.89	95.22	98.75	94.34	92.69	94.08	86.29	91.44	80.40	88.78
MCAST	98.02	**98.85**	97.80	95.95	95.93	**99.31**	98.55	99.11	97.40	98.33	98.55	96.49	97.27
SWITCH+ROOT	97.13	97.93	99.11	94.23	95.24	98.75	96.59	95.04	95.82	92.39	96.86	93.49	95.12
SWITCH+MCAST	96.87	98.59	**99.42**	95.83	95.05	97.31	99.28	**99.15**	97.76	98.73	**99.21**	96.88	**98.37**
ROOT+MCAST	97.78	98.58	98.34	**96.40**	95.85	98.77	99.26	97.85	**97.94**	98.11	96.60	96.89	95.68
T-RPL	96.78	98.10	97.80	94.88	**96.15**	98.72	**99.75**	98.44	97.68	**98.93**	98.03	**97.20**	97.16
FLOODING	100	99.86	98.88	100	100	99.11	100	100	100	100	100	100	100
dense													
	25	28∗	31∗	45	46	51∗	54	62	70	91	104	123	134
ContikiRPL	98.93	95.54	98.16	97.02	70.27	98.68	24.81	23.40	21.11	12.58	28.19	19.52	17.28
SWITCH	98.82	97.38	98.14	95.98	94.08	98.00	56.49	58.64	55.65	34.11	46.25	39.04	36.98
ROOT	98.75	97.97	98.39	94.31	94.80	98.74	96.22	93.53	92.71	94.64	88.41	87.93	84.54
MCAST	98.95	**98.61**	**99.01**	98.31	98.40	**98.97**	98.84	99.45	**99.63**	98.97	98.51	99.28	**99.08**
SWITCH+ROOT	98.80	95.71	98.05	96.75	97.86	99.14	97.42	94.36	94.25	95.01	94.77	90.68	90.89
SWITCH+MCAST	97.45	96.57	98.09	97.09	98.03	98.38	**99.85**	99.35	99.59	99.37	**99.03**	99.39	99.02
ROOT+MCAST	**99.09**	97.29	98.30	**99.24**	**99.02**	98.02	98.63	**99.88**	99.60	**99.89**	96.44	**99.50**	98.90
T-RPL	98.90	96.68	98.12	98.71	98.77	97.96	99.27	99.82	99.51	**99.89**	98.60	99.33	98.88
FLOODING	100	100	100	100	100	100	100	100	100	100	100	100	100

decreases, the diversity of available paths also decreases, leaving nodes with fewer alternative DAO parents and ultimately reducing the gap between SWITCH and ContikiRPL.

In contrast, ROOT achieves $PDR > 76\%$ with DC similar to ContikiRPL, showing the efficiency of the simple 1-hop broadcast at the root. As expected, the root is often a bottleneck, limiting the overall network performance while the rest of the nodes suffer less from the memory limitations. By fixing the problem only at the root we significantly improve the downward packet delivery ratio of RPL. However, ROOT does not achieve perfect reliability in any of the topologies (Table 6) because *i)* the link-layer broadcast is less reliable than unicast, and *ii)* the 1-hop broadcast only applies to the root; all other nodes with saturated memory still drop packets whenever they do not know where to forward them.

MCAST is the mechanism yielding the best results with the larger networks, approaching near-perfect reliability in dense topologies (Table 6), confirming the effectiveness of its scope-limited flooding. However, this comes at the expense of slightly increased DC, in part because MCAST delivers up to 4 times more data packets than ContikiRPL, but also because of multicast: the same data packet is forwarded on several paths, to reach all junction nodes.

On the other hand, FLOODING is the only protocol with a perfect reliability for all dense topologies and most of intermediate ones. This is possible because packets injected by the root are forwarded on all the possible paths, exploiting their high redundancy. However, the latter is significantly less marked in sparse scenarios, especially in linear topologies, as *PDR* drops below 49% (Table 6), due to the lack of diversity in network paths that FLOODING, unlike the other protocols, cannot compensate with MAC-level unicast retransmissions. Further, the higher reliability of FLOODING comes at the cost of a steep increase in energy consumption; in dense scenarios, DC is up to 6 times higher than in the other variants (Figure 8c), due to the



(a) Sparse (b) Intermediate (c) Dense

Figure 9: Performance of the mechanism combinations in the smart city scenario.

overhead induced by duplicate packets. In sparse networks, however, FLOODING yields a *DC* only marginally higher than MCAST, along with a marginally higher *PDR*. Moreover, MCAST performs significantly better in the aforementioned case of linear topologies, reaching an overall better tradeoff between reliability and energy efficiency.

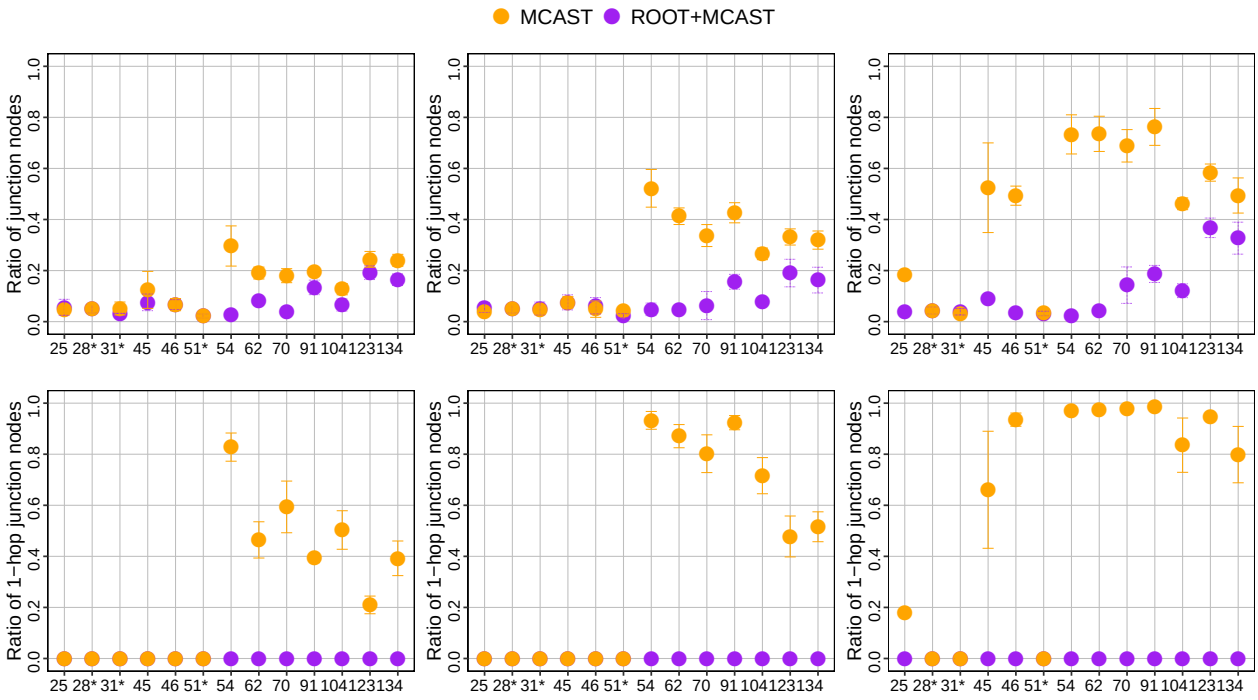
Combination of mechanisms. We now evaluate SWITCH+ROOT, SWITCH+MCAST, ROOT+MCAST, and T-RPL as outlined in §5.4. Figure 9 and Table 6 show the results.

As previously mentioned, SWITCH and ROOT are complementary to each other and their combination significantly increases reliability especially in larger networks (Figure 9) as now *all* nodes implement a mechanism to overcome the memory problem. Further, this is achieved with a very low *DC*. However, the worse link quality towards non-preferred parents, inherited from SWITCH, and the lack of acknowledgments for link-local broadcasts, inherited from ROOT, prevent their combination from achieving a perfect *PDR*.

SWITCH+MCAST achieves higher reliability than MCAST in most cases, especially in large networks with sparse density; further, this is achieved with a lower *DC*, as the presence of SWITCH mechanisms help reduce the scope of dissemination in MCAST. Nevertheless, in dense scenarios, SWITCH+MCAST performs worse than MCAST, as high density allows SWITCH to find alternative DAO parents that, however, have worse link quality than the preferred parent. Interestingly, in the larger cluster SWITCH+MCAST achieves the best reliability across all densities among all RPL variants considered.

ROOT+MCAST achieves a lower *DC* w.r.t. MCAST, while maintaining a very similar *PDR*. To understand the reason, we analyzed the number of junction nodes present in the network when MCAST is used alone and in combination with ROOT. As Figure 10 shows, the latter case shows a considerable decrease in the total number of junction nodes at all densities. Interesting, this is achieved mostly by eliminating those in the immediate vicinity of the root, as seen in the bottom charts; this type of junction nodes are totally absent in ROOT+MCAST, thanks to the 1-hop broadcast provided by ROOT. This means that in ROOT+MCAST the relatively expensive multicast dissemination is activated infrequently, considerably reducing the communication overhead and therefore *DC*.

The T-RPL variant, combining all three base mechanisms, achieves a *PDR* very close (and in a few cases coinciding) with the best one, often represented by ROOT+MCAST. However, compared to the latter, it has a



(a) Sparse (b) Intermediate (c) Dense

Figure 10: Number of junction nodes in the smart city scenario, under different densities. The top charts show the ratio of junction nodes over the entire network; the bottom charts show the ratio of junction nodes that are 1-hop from the root over the total number of junction nodes.

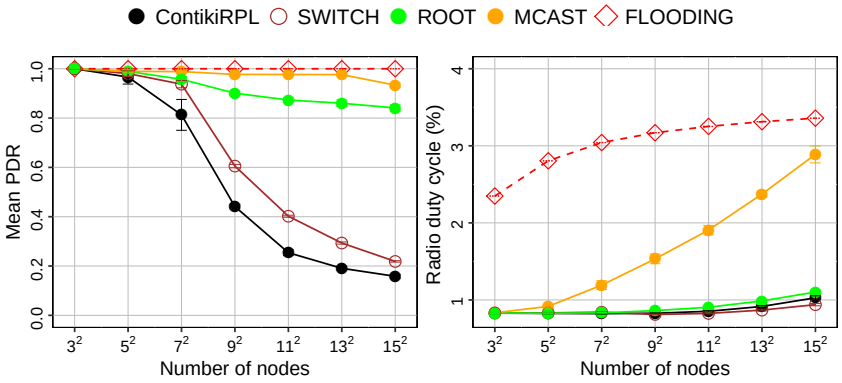


Figure 11: Performance of the base mechanisms on 2D regular grids, as a function of network size.

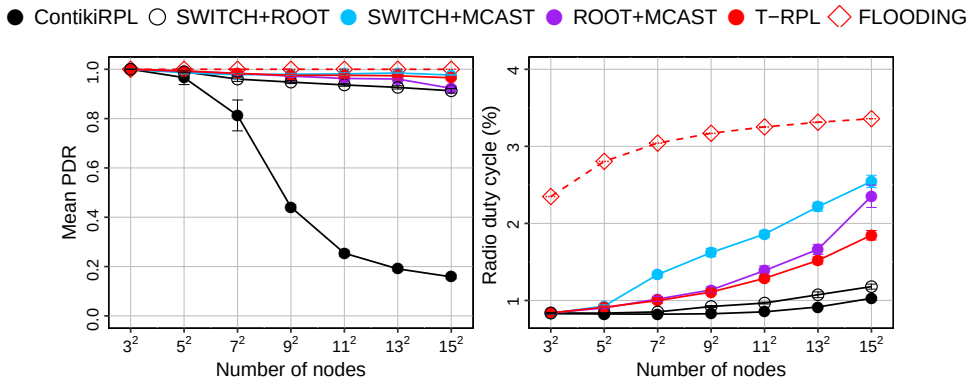


Figure 12: Performance of mechanism combinations on 2D regular grids, as a function of network size.

significantly lower DC, especially as the network size increases. The lowest $PDR = 89.28\%$ is achieved with sparse density and the largest cluster; in this situation, the 1-hop broadcast mechanism is less effective, as witnessed also by the performance of the variants involving ROOT. However, T-RPL still achieves higher PDR than these variants. In general, T-RPL is the most energy-efficient among MCAST-based variants, while providing a similar (or higher) PDR. Therefore, by combining the advantages of the three base mechanisms, T-RPL achieves the best tradeoff between reliability and energy consumption among RPL-based protocols.

7.3 Synthetic Grid Topologies

We continue the evaluation of our RPL variants, this time using 2D regular grid topologies. This allows us to: *i)* verify that the tradeoffs we observed in §7.2 are not specific to the smart city topologies considered, and *ii)* test the scalability of our solutions by almost doubling the size of the network. As before, we first compare the base mechanisms and then their combinations.

Base mechanisms. Figure 11 shows the comparison between the proposed mechanisms and the baselines, while Table 7 offers a more precise view of the PDR.

As in the smart city scenario, the scalability of ContikiRPL is severely affected by the routing and neighbor table problems. Instead, FLOODING has perfect reliability for all network sizes. However, its DC is 3 times higher than ContikiRPL, due to the overhead induced by duplicate packets.

SWITCH shows a trend similar to the smart city scenario, which is only worsened by the increase in network scale; as the number of nodes increases, the limitation on the maximum number of routes becomes

Table 7: *PDR* (%) for synthetic grid topologies.

#nodes	9	25	49	81	121	169	225
ContikiRPL	100	96.64	81.29	44.10	25.41	19.08	15.83
SWITCH	100	98.04	93.74	60.53	40.20	29.33	21.87
ROOT	100	98.93	95.72	90.06	87.33	86.01	84.09
MCAST	100	98.94	**98.84**	97.69	97.67	97.62	93.35
SWITCH+ROOT	100	98.97	96.02	94.80	93.65	92.66	91.27
SWITCH+MCAST	100	98.62	97.79	**97.99**	**98.14**	**98.49**	**97.67**
ROOT+MCAST	100	99.24	98.33	97.23	96.29	96.05	92.27
T-RPL	100	**99.30**	98.34	97.55	97.56	97.37	96.53
FLOODING	100	100	100	100	100	100	100

more marked and the performance of SWITCH approaches the one of ContikiRPL.

ROOT also shows results consistent with smart city topologies; reliability is much higher than ContikiRPL although lower than FLOODING and MCAST. Further, ROOT it is not affected by the significant increase in scale in grid topologies, as *PDR* remains almost constant after the network size reaches 11^2 nodes, roughly equivalent to the biggest smart city cluster. As in the latter scenario, the simple use of 1-hop broadcast at the root yields an impressive 4-fold reliability improvement w.r.t. ContikiRPL while maintaining $DC < 1\%$.

As for MCAST, its reliability is again similar to the smart city scenario, but the increase in scale induced by our grid topologies exacerbates the protocol weakness, namely, its energy consumption. *DC* increases linearly with the size of the network; for the 225-node network becomes 3 times higher than the other RPL variants, approaching the one of FLOODING. However, unlike FLOODING, MCAST is significantly more selective in triggering dissemination only when needed (Figure 11); the increase in *DC* with network size is a direct consequence of the fact that multicast is triggered more often to cope with the increasing number of nodes with overloaded routing and neighbor tables.

Combination of mechanisms. Figure 12 shows that SWITCH+ROOT achieves $PDR > 91\%$ for all network sizes, but without reaching 100%, consistent with results from smart city topologies. Still, it yields the lowest *DC* among all the proposed solutions.

Among MCAST variants, the reliability of SWITCH+MCAST and T-RPL is remarkable and quite stable w.r.t. the increase in network size; the difference in *PDR* between the two is $< 1\%$, while ROOT+MCAST performs worse due to the unreliable 1-hop broadcast at the root. Overall, the combination of mechanisms offered by T-RPL is significantly more energy-efficient, yielding the lowest *DC* among MCAST-based solutions. This is similar to what we noted for smart city topologies, but exacerbated here by the increase in scale; T-RPL consumes 28% less energy than SWITCH+MCAST, despite offering basically the same reliability.

7.4 A Note about Memory Consumption and Routing Table Size

We observed (§6) that the data memory requirements of our mechanisms, and in particular T-RPL, are slightly higher than ContikiRPL. As we configured all protocols, including FLOODING, with the default neighbor table size of 20, this difference in memory occupation is essentially determined by the routing tables, whose size is also the same for all protocols (Table 8). Indeed, while the routing entries of ContikiRPL occupy 34 B, those for MCAST and T-RPL require 36 B and 52 B, respectively[7].

The lower memory occupation of ContikiRPL raises a question: If ContikiRPL were granted a larger number of routing entries, would its reliability improve w.r.t. scale? Or, dually, if T-RPL were granted a *lower* number of routing entries, would this affect its reliability?

Figure 13 offers an answer to both these questions by comparing the performance of these two protocols in our 50-entry configuration (same as Figure 12) against one where *i)* ContikiRPL is allocated as many routes (95) as can be fit in the memory used by T-RPL (9.3 KiB) and, dually, *ii)* T-RPL is allocated only as many routes (23) as can be fit in the memory used by ContikiRPL (7.8 KiB). This configuration nearly *doubles* the routing entries assigned to ContikiRPL and *halves* those assigned to T-RPL. Nevertheless,

[7]The routing entries in T-RPL and SWITCH contain the 8B address of the alternative parent; these could be reduced to a 2B pointer to the address in the neighbor table.

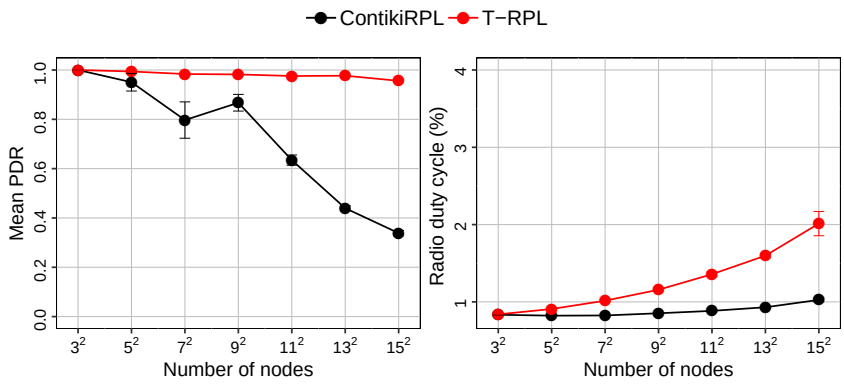


Figure 13: Impact of memory reserved to routing entries. Compared to Figure 12, where all protocols are configured with 50 entries, ContikiRPL and T-RPL here have 95 and 23 entries allocated, respectively. These values are based on each other's RAM occupation when the default 50-entry configuration is used (Table 4).

Figure 13 shows that this only marginally affects the tradeoffs of reliability vs. scale. The extra routing tables allow ContikiRPL to marginally improve reliability, which remains at $PDR \approx 85\%$ for 81 nodes but rapidly degrades for larger scales. In contrast, the reliability of T-RPL is essentially unaltered, even with half the routing tables. The reason is that T-RPL can compensate the scarce memory with alternate, route-less mechanisms, which ensure reliability with significantly lower memory costs. Interestingly, and desirably, these bear minimal impact on DC, which remains very close to the one in Figure 12. In other words, the mixed strategy embodied by T-RPL enables this protocol to exploit very efficiently the memory available, without sacrificing reliability and energy consumption.

7.5 Summary

Our results show that the pure route-based approach of SWITCH, albeit standard-compliant, brings only limited improvements. Instead, introducing flooding over different scopes significantly improves reliability, especially in dense scenarios, but with different energy tradeoffs. T-RPL stands out as its PDR improves considerably w.r.t. ContikiRPL, while offering a very low DC. In the smart city scenario, the reliability of T-RPL is generally very high and consistent across type of topology (linear vs. planar) and density. The results in grid topologies *i)* confirm these tradeoffs and *ii)* show that the negative characteristics of some RPL variants are exacerbated by the increase in scale.

Nevertheless, Table 6 shows that for applications requiring *perfect* reliability, FLOODING is the answer, due to its ability to exploit redundant network paths. However, the price to pay is a remarkably higher energy consumption and a dependency on the physical topology; FLOODING achieves unacceptable reliability on linear and sparse topologies due to the lack of diversity in network paths combined with the absence of MAC-level retransmissions. Instead, T-RPL achieves the best tradeoff between reliability and energy consumption by combining the advantages of all base mechanisms, yielding near-perfect PDR with a low DC.

8 Comparison against Existing Protocols

The evaluation in §7 shows that T-RPL is the RPL protocol variant that offers the best tradeoff between reliability and energy consumption among all those we considered. However, a question remains about how does T-RPL fare against other popular downward routing protocols, alternative to ContikiRPL. We answer this question in the following by considering protocols with very different characteristics and underlying mechanisms:

Table 8: Protocol configuration.

	MAC	Neighbor table size	Routing table size	Routing metric	Objective Function
ContikiRPL	ContikiMAC	20	50	ETX	MRHOF
FLOODING	ContikiMAC	20	–	ETX	MRHOF
TinyRPL	BoxMAC	20 (parent table)	50	ETX	MRHOF
ORPL	ContikiMAC (custom)	40	–	EDC	custom
TM	ContikiMAC	20	–	ETX	MRHOF
AODV	ContikiMAC	20	256	hop count	–
T-RPL	ContikiMAC	20 (4 reserved)	50	ETX	MRHOF

- *TinyRPL* is the reference implementation of RPL for TinyOS [**?**], one of the mainstream operating systems in the LLN community. Comparing different implementations of RPL helps us elicit differences in performance [**?**]. For instance, while ContikiRPL adapts the interval between DAO messages based on network dynamics, TinyRPL sends DAO messages with a fixed periodicity, regardless of the network status.

- *Opportunistic RPL (ORPL)* [**?**] is a modification of the RPL protocol which aims at improving the scalability of downward routing by replacing the RPL path building mechanism with a hash- or bitmap-based one. Moreover, it also supports multiple downlink paths by using opportunistic forwarding [**?**] at the link layer. ORPL uses a custom version of ContikiMAC [**?**] as its underlying duty-cycling MAC.

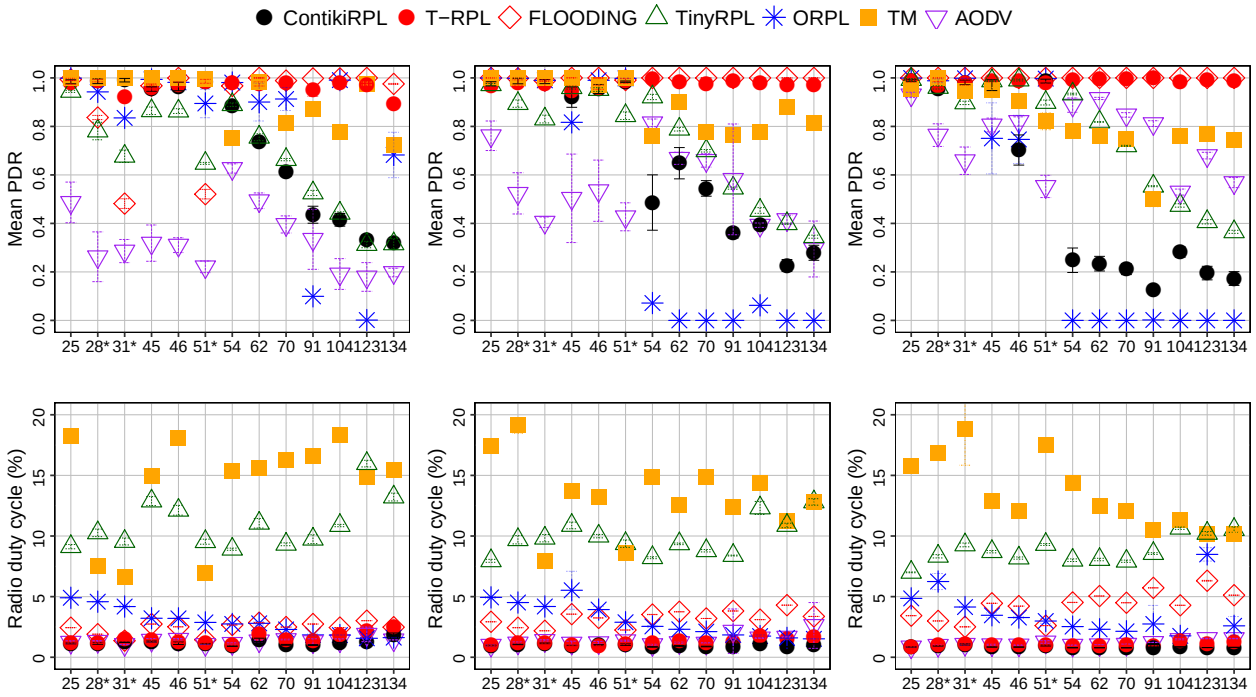
- *Trickle Multicast (TM)* [**?**], an IETF standard, supports downward traffic from the root to multiple nodes. TM relies on Trickle [**?**] to control packet retransmissions by a node based on the number of neighboring transmissions for the same packet. TM provides eventual delivery to all nodes—a property not guaranteed by pure FLOODING.

- *Ad Hoc On-Demand Distance Vector (AODV)* [**?**] is a routing protocol that builds and maintains individual unicast routes. To construct routes, AODV floods the network with RREQ (route request) control packets. Each node registers locally an entry for the originator of the RREQ, therefore building a reverse path. The destination replies (in unicast) with a RREP (route reply) message, therefore establishing a bi-directional route between source and destination. AODV is used by a significant body of literature, it is supported by ZigBee, and is naturally amenable to downward traffic from the root to individual nodes.

Hereafter we compare T-RPL against these protocols, in the same simulation setup described in §7.1, again distinguishing between the realistic smart city scenario and the synthetic grid one.

Protocol configuration. All these protocols are highly customizable through parameters such as buffer sizes, timeouts, retries and hop limit. Also, some of them are highly dependent on the underlying MAC protocol. Wherever possible, we used the default values, as these are likely to be first choice in a deployment and the ones tested the most. The value of the most important protocol parameters are summarized in Table 8. We note that ORPL can use a larger neighbor table size than other RPL-based protocols, as it does not require a routing table. Moreover, AODV has a considerably larger routing table, partly because it does not use IPv6 addresses, and partly because its implementation occupies less memory than RPL-based protocols.

As mentioned, the choice of the (fixed) DAO interval is crucial for TinyRPL, and therefore required extra effort. Specifically, we simulated TinyRPL on different types of smart city topologies to determine the best tradeoff between *PDR* and energy consumption. As the DAO interval increases, *PDR* becomes nearly constant, while *DC* decreases as less control packets are generated. We verified that the optimal value in our setup occurs for a DAO interval value of 180s.



(a) Sparse (b) Intermediate (c) Dense

Figure 14: Performance comparison against existing protocols, in the smart city scenario.

8.1 Smart City Topologies

Figure 14 shows the results in the smart city scenario. Interestingly, TinyRPL achieves higher *PDR* than ContikiRPL at high scale and/or density, an indication that TinyRPL is much less affected by memory limitations. The reason lies in an optimization in TinyRPL enabling forwarding data packets without consulting the neighbor table. The global IP address of a node is derived from its MAC address, making the opposite conversion possible without storing any state. However, the 50-route upper bound on addressable devices still holds, affecting the *PDR* on larger clusters. Moreover, due to the fixed DAO interval, the *DC* of TinyRPL is 10 times higher than ContikiRPL.

Instead, ORPL exhibits worse reliability than ContikiRPL, experiencing a sudden drop to $PDR = 0\%$ when the neighbor table becomes full, as no mechanism is in place to cater for this situation. Clearly, the problem is less severe in sparse topologies, as the number of neighbors is reduced.

TM improves over ContikiRPL, but its *PDR* remains ~80% in large or dense networks, despite its design in principle guarantees eventual packet delivery. Ironically, the reason is a network overload due to the high number of retransmissions injected in the network to provide this guarantee. This is also the reason why TM has the highest *DC* among all protocols considered, including FLOODING.

AODV, on the other hand, yields unacceptable reliability in almost all clusters, regardless of scale and density; the reason is the hop-count metric tends to use longer but weaker links to minimise the route length. Further, performance is especially dire in sparse networks; as nodes have less neighbors and paths are longer, RREQ and/or RREP packets are likely lost, and routes remain incomplete. The low *DC* is explained by the similarly low *PDR*.

The tradeoffs we observed in §7.2 are therefore still valid. T-RPL generally provides the best reliability across topologies, scale, and density among RPL-based solutions without a significant increase in *DC* w.r.t. the baseline ContikiRPL. This is in contrast to FLOODING, which achieves much higher reliability (except for linear topologies) at the cost of a significantly higher *DC*.

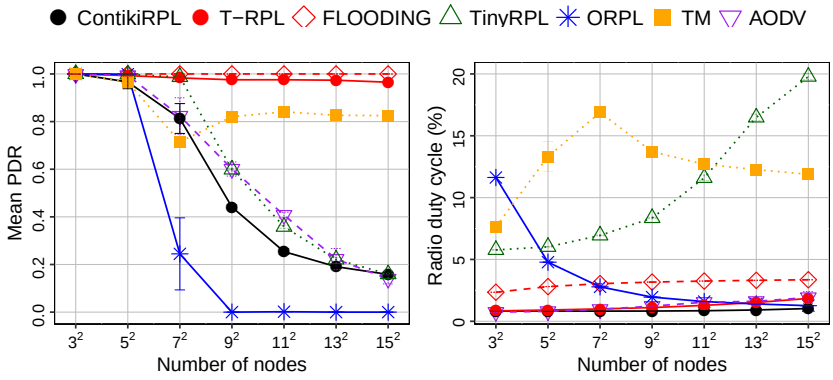


Figure 15: Performance comparison against existing protocols, on 2D regular grids.

8.2 Synthetic Grid Topologies

We now turn our attention to synthetic grids that, as noted in §7.3, allow us to investigate protocol performance in larger networks and a different topology. Figure 15 shows the results.

As in §7.3, this setup exacerbates the negative aspects of some of the protocols. For instance, TinyRPL achieves perfect *PDR*, unlike ContikiRPL, until the routing table is filled; for larger networks, *PDR* remains above ContikiRPL but eventually coincides with it on the largest topology tested. The reliability of AODV is very close to TinyRPL, but for a different reason, namely, the increased number of RREQ/RREP packets, which overload the network and ultimately prevent correctly building and maintaining routes. Finally, ORPL breaks down as soon as the root neighbor table reaches its maximum of 40 entries.

TM behaves similarly to the smart city scenario, showing a constant *PDR* ∼80% across all networks >49 nodes. Due to its multicast mechanism and the high number of retransmissions TM overloads the network. This can also be seen in the high *DC*, almost always >10%.

Finally, FLOODING achieves again perfect reliability but with a *DC* twice or more the one of T-RPL, which nevertheless achieves a $PDR = 96.53\%$ in the largest topology.

8.3 Summary

The comparison between ContikiRPL and TinyRPL shows that the problems identified in §4 are not implementation-dependent—or at least are shared by these two implementations, which are arguably the most widespread.

Moreover, *all* the popular downward routing protocols we considered provide unacceptable reliability in some (if not all) combination of topology, scale, and density. Interestingly, all route-based protocols (ContikiRPL, TinyRPL, AODV) are on the lower half of reliability performance, especially in larger networks; instead, flood-based protocols (FLOODING, T-RPL, TM) systematically achieve $PDR > 80\%$. ORPL, which does not fit precisely either category, unfortunately completely stops working when the neighbor table is full.

FLOODING and our solution, T-RPL, are the only protocols capable of achieving near-perfect *PDR*, albeit striking different tradeoffs between reliability and energy consumption.

9 Influence of Background Upward Traffic

In the previous sections we have evaluated the performance of our solutions by considering only downward traffic; this is reasonable, as this is the traffic we are aiming to improve. However, in practice, most applications contain *both* upward and downward traffic; a staple example is the case where upward data collection and downward actuation commands co-exist in a single monitoring and control application [**?**, **?**, **?**, **?**, **?**]. After all, RPL is designed precisely to support both traffic patterns. However, when both are present, upward and downward traffic affect each other.

Therefore, in this section we take the unmodified upward ContikiRPL and evaluate the mutual impact it has w.r.t. the protocols we considered thus far. Specifically, we verify that *i)* the tradeoffs among downward protocols we hitherto elicited do not significantly change when upward traffic is present, and *ii)* the reliability and energy consumption of ContikiRPL when supporting upward traffic is not significantly hampered when used in conjunction with one of the downward protocols.

This evaluation goal requires a slight adaptation of the simulation setup in §7.1, described next, followed by the usual evaluation in both the realistic smart city scenario and the synthetic grid one. We focus on the two baselines, downward ContikiRPL and FLOODING, along with the popular protocols we considered in §8. As for our solutions, we consider only T-RPL; we omit the comparison against base mechanisms and their combination, as we verified that T-RPL offers the best tradeoff between *PDR* and *DC* also in the presence of upward traffic.

9.1 Simulation Setup and Performance Metrics

The characteristics of downward traffic, mimicking actuation commands, are the same as in §7.1. Here, we add a background upward traffic mimicking data collection, in which *each* node sends a packet to the root every 3 minutes; sending times are random and not synchronized. Although this configuration of upward traffic may appear to yield lighter traffic than its downward counterpart, in practice it increases with scale; for networks larger than 18 nodes (i.e., all clusters in the smart city scenario) the accrued upward traffic is actually higher than downward traffic.

To concisely assess the *combined* performance of upward and downward protocols, we define two additional performance metrics, derived from those in §7.1:

- *Overall PDR*, defined as $PDR_{all} = PDR_{up} \times PDR_{down}$. This is a measure of the reliability achieved by the network in the presence of both upward and downward traffic.
- *Overall efficiency*, defined as $E = PDR_{all} \times \frac{1}{DC}$, relates the overall reliability with the energy cost incurred for achieving it.

Of course, in principle both the overall PDR_{all} and efficiency E should have a high value.

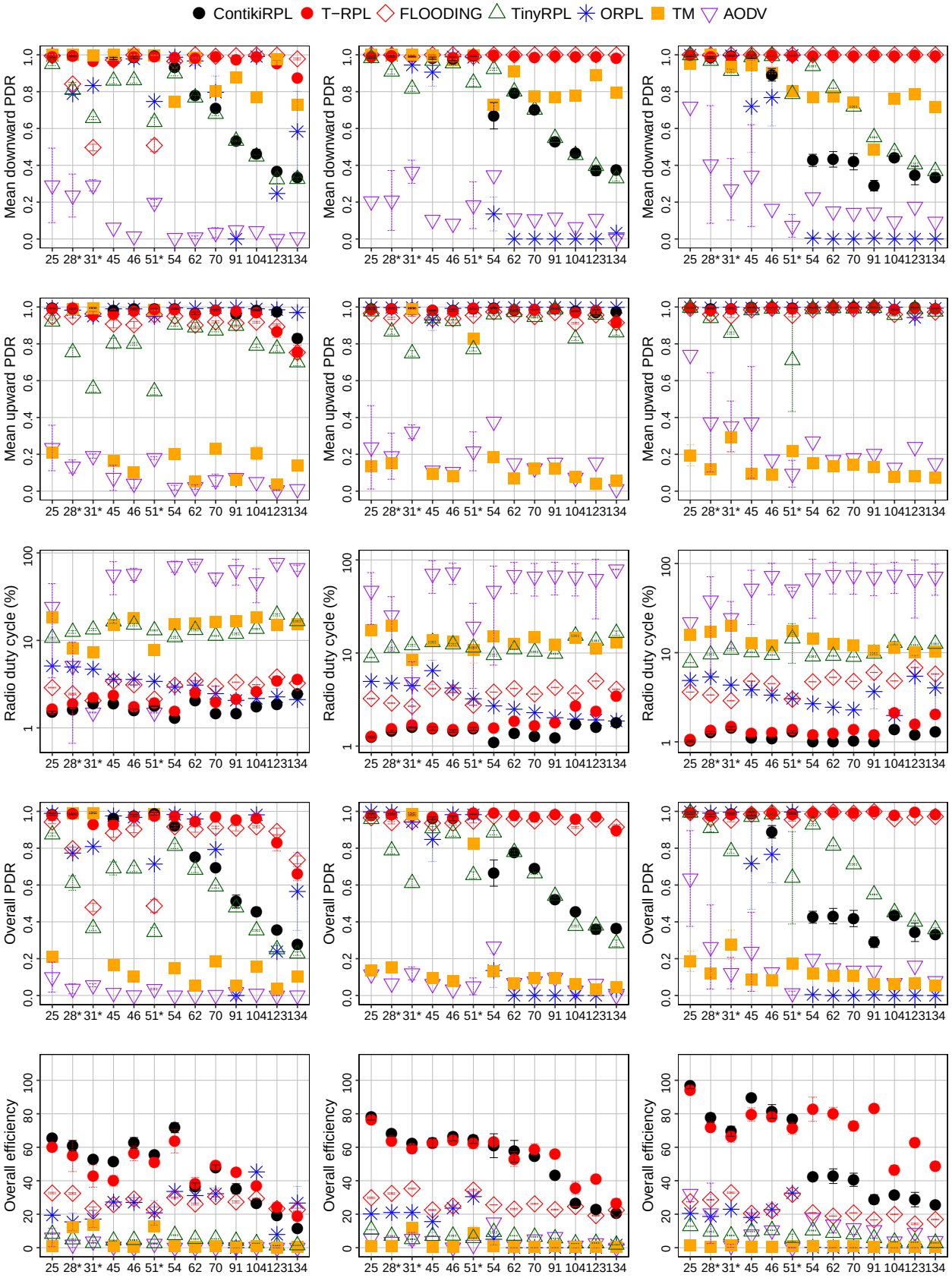
9.2 Smart City Topologies

Figure 16 shows the values of PDR_{down}, PDR_{up}, and DC for all smart city clusters under different topology densities, as in earlier sections, along with the values of PDR_{all} and E derived from them.

It is worth comparing the PDR_{down} in Figure 16 with the corresponding *PDR* in Figure 14, where downward traffic occurred in isolation. For ContikiRPL and ORPL the difference between the two is minimal; background upward traffic bears almost no impact on these protocols. Further, upward traffic is similarly unaffected by downward one. This is a very desirable property, that however is thwarted by the unacceptable downward reliability shown by both protocols. Notably, this property does not come for granted in RPL implementations; in TinyRPL, downward traffic is similarly unaffected, but upward one shows poor reliability in several clusters.

At the other extreme, AODV and TM break down when both upward and downward traffic are present. The heavy overhead they impose not only yields a low PDR_{down}, but also prevents upward RPL from achieving acceptable reliability.

As in previous sections, FLOODING and T-RPL stand out w.r.t. downward reliability; their PDR_{down} is largely unaffected by background upward traffic. However, the reverse is not true: for both protocols (and especially FLOODING), increased contention in the larger clusters causes a slight decrease in PDR_{up} at sparse and, to a less extent, intermediate densities. However, T-RPL is significantly more efficient than FLOODING, as the gap in *DC* between the two protocols is widened by the presence of upward traffic. Instead, T-RPL is as efficient as ContikiRPL in the scenario combinations where the latter performs well. These considerations are further evidence that T-RPL is capable of balancing the benefits of FLOODING by effectively mitigating its drawbacks via alternative mechanisms.



(a) Sparse (b) Intermediate (c) Dense

Figure 16: Performance with combined upward/downward traffic, in the smart city scenario. Note the logarithmic scale for DC.

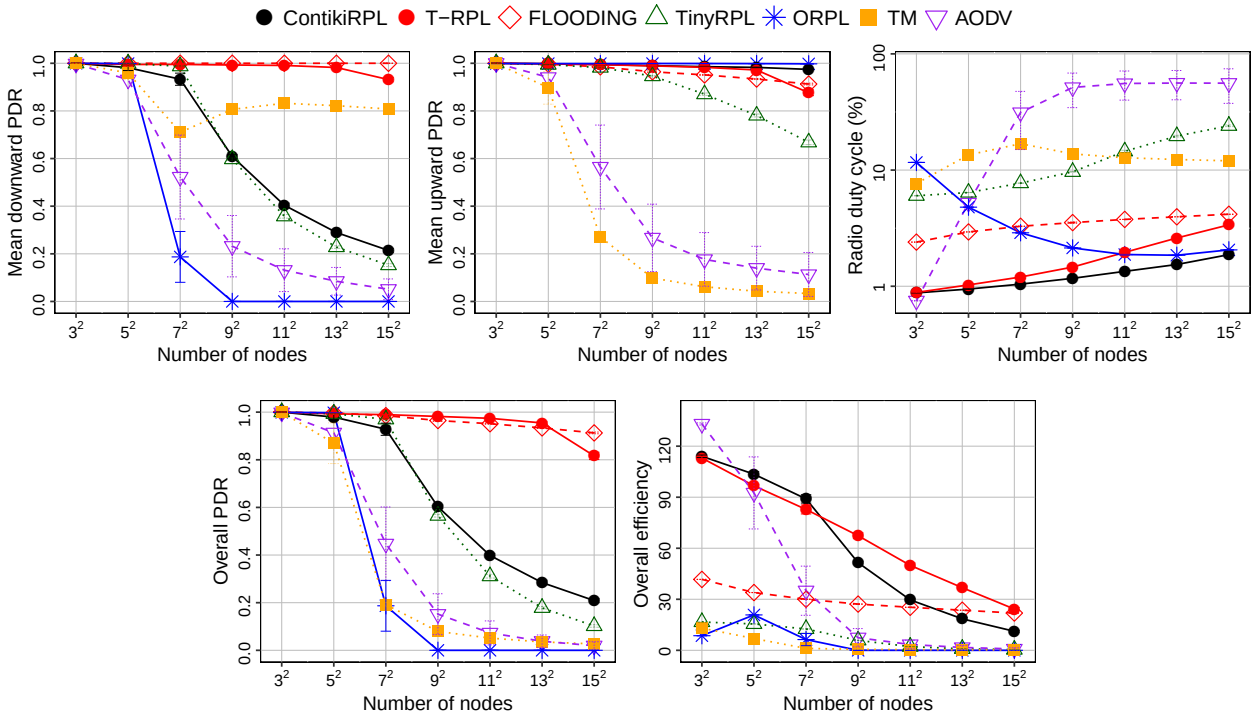


Figure 17: Performance with combined upward/downward traffic, on 2D regular grids. Note the logarithmic scale for *DC*.

9.3 Synthetic Grid Topologies

Figure 17 shows the results in synthetic grid topologies, essentially confirming the observations in both §8.2 and §9.2. Hereafter, we focus on T-RPL and FLOODING, as the other protocols generally yield unacceptable performance except for very small networks. A comparison between Figure 17 and Figure 15 reveals that, again, the performance of these two protocols is essentially unaffected by upward traffic. However, as in the smart city scenario (§9.2), the reverse does not hold, due to increased contention. Moreover, in regular grids FLOODING affects upward ContikiRPL slightly more than T-RPL, except for the largest network.

As for energy consumption, the regular topology enables us to ascertain better the fact that T-RPL is significantly more efficient, avoiding communication unless strictly needed. In contrast with the nearly-constant and relatively high *DC* imposed by FLOODING, the *DC* of T-RPL is considerably lower. It does increase with network size as the amount of upward traffic increases and downward forwarding paths get longer, but it only gets close to FLOODING's *DC* at the largest studied size, when the accrued upward traffic starts to saturate the network.

9.4 Summary

We observed (§7–§8) that FLOODING excels at reliably delivering downward traffic, although with significantly higher energy consumption than T-RPL, whose *PDR* remains very close to FLOODING.

However, in the realistic scenario where background upward traffic coexists with the downward one, our results show that the overall reliability of both protocols becomes very similar, due to the increased contention affecting the upward traffic, supported by unmodified ContikiRPL. FLOODING is slightly more reliable in the largest network; on the other hand, T-RPL is generally more scalable and efficient than FLOODING, thanks to its peculiar and synergistic combination of techniques, yielding the ability to borrow the benefits of flooding while limiting their drawbacks.

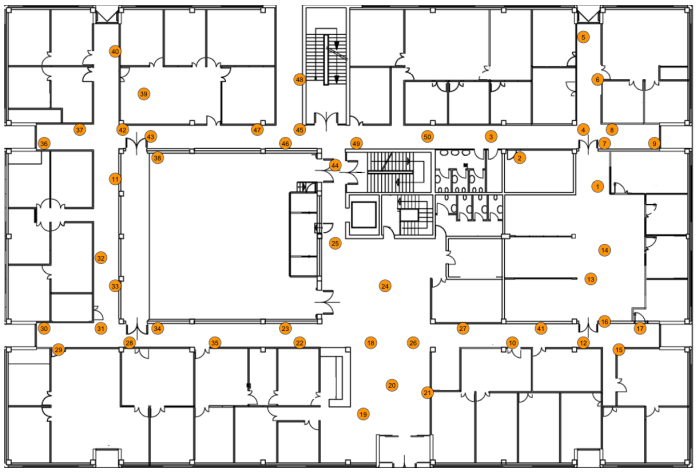


Figure 18: Testbed node deployment.

Table 9: Topology metrics for the testbed experiments.

Avg degree	11.7
Max degree	19
Min degree	7
Avg hops	2.6
Max hops	5
Avg sum of link *PDR*	10.8
Max sum of link *PDR*	17.8
Min sum of link *PDR*	5.4
Avg path ETX	2.6
Max path ETX	5.1

10 Testbed Experiments

We have hitherto exploited simulation to evaluate our proposed protocol mechanisms and distill our findings. Simulation is a convenient tool that allows one to fully control and assess the impact of protocol and network parameters in isolation. However, the findings from simulation require a validation in a real-world setup, which is the goal of this section.

We discuss next the similarities and differences of our testbed setup w.r.t. the simulation one, followed by a discussion of the results. Based on the findings in §7–§9, we focus on the base mechanisms proposed (SWITCH, ROOT, and MCAST) and their synergistic combination into T-RPL, and compare them against the same baselines used in simulation, namely, ContikiRPL and FLOODING.

10.1 Experimental Setup

We evaluate the above protocols in a 50-node 60×40 m^2 indoor testbed at our premises (Figure 18), with node 1 as the sink. To reduce differences across runs due to variables out of our control (e.g., noise in the ISM band) we executed test at night and during weekends on the IEEE 802.15.4 channel 26, the least affected by WiFi networks. Table 9 shows the metrics characterizing the testbed. A comparison with Table 5 shows its network topology properties are roughly similar to those of intermediate density topologies for both the smart city and synthetic grid scenarios.

However, the two tables also show that there is a significant disparity in the size of the networks we experimented with. The lack of large-enough testbeds and the difficulty to directly control density (e.g., via limited transmission power settings) prevent us from reproducing the impact of an increasing network size as we did in previous section. On the other hand, as mentioned in §4 a fundamental factor limiting the scalability of ContikiRPL *is the size of the routing table in relation to the size of the network*. In previous sections, we fixed the former to the maximum allowed and experimented with an increasing network size. Here, due to the inability to push the latter beyond the size of our testbed, we fix the network size and experiment with decreasing values of the routing table size. Specifically, we experiment with values ranging from the maximum of 50 entries we used throughout the paper, enough to accommodate *all* nodes in our testbed, and decrease this value in steps of 10 entries. The minimum value of 10 entries therefore stores $\frac{1}{5}$ of the nodes, a ratio similar to the one used for the largest 225-node network in synthetic grid experiments.

We focus on combined upward and downward traffic, the situation of most practical relevance, using the same IMI in §7.1 and §9.1, i.e., 1 downward packet generated at the root towards a random destination every 10s, and 1 upward packet generated by each non-root node every 3 minutes.

We report the same performance metrics investigated thus far. For reliability, we report the individual

Table 10: Testbed experiments: *PDR* vs. routing table size.

	Downward *PDR*					Upward *PDR*				
Routing table size	50	40	30	20	10	50	40	30	20	10
ContikiRPL	99.62	79.27	57.21	38.40	18.41	99.96	99.97	99.92	99.91	99.93
SWITCH	99.83	83.14	62.34	42.02	20.83	99.97	99.98	99.95	99.98	99.92
ROOT	99.24	98.31	97.61	97.01	73.20	100	99.88	100	99.95	99.86
MCAST	99.33	99.29	99.34	98.72	98.75	99.86	99.92	99.97	99.93	99.98
T-RPL	99.24	99.72	99.84	98.99	98.38	99.86	99.88	99.96	99.87	99.77
FLOODING	100	100	99.96	99.96	99.91	99.95	99.93	99.97	99.93	99.86

PDR_{up} and PDR_{down} along with the overall PDR_{all}. Similarly, for energy consumption we report both the radio duty cycle *DC* and the overall efficiency *E*. However, *DC* can no longer be computed automatically by Cooja; therefore, we compute *DC* on the real devices using Energest [**?**], the built-in energy profiler of Contiki, a common choice to estimate *DC*.

For each combination of protocols and configurations, the results shown are based on 5 runs with a 1.5-hour duration. The error bars in the charts refer to the minimum and maximum value of the metrics observed across runs.

10.2 Results

Figure 19 and Table 10 present the results of our testbed experiments. First, we notice the near-perfect reliability of all the protocols in upward traffic. This is consistent with what we observed in §9, given the smaller scale of our testbed if compared to the network sizes at which contention begins to hamper reliability of upward traffic.

Moreover, the downward *PDR* of ContikiRPL and SWITCH are very similar, again in line with our previous observations. Their value closely approximates the number of destinations stored in the routing table; in other words, when the route is known, the protocols deliver downward traffic well. ROOT, with its simple 1-hop broadcasting, overcomes the memory limitations for all cases except for the most challenging one with only 10 routing entries. Interestingly, in this worst case the value of *PDR* is similar to the worst-case of 225 nodes in Figure 15. In these situations where the ratio between memory entries and network scale is unfavorable, not only the root suffers from the routing table overflow, but also the next-hop nodes. On the other hand, MCAST and T-RPL solve the issue also at non-root nodes, thanks to their scope-limited dissemination; they achieve near-perfect downward reliability, a result in accordance with previous results.

As for energy consumption, a comparison against results in previous sections (e.g., Figure 11 and 17 with 7^2 nodes) shows that simulation estimates for dissemination-based protocols are more pessimistic than testbed results. For instance, FLOODING is 1.5 times more energy-efficient in the testbed; the same holds also for MCAST, although to a lesser extent. The other protocols instead show similar results in both simulation and the testbed experiments, with a *DC* around 1%. Despite these (somewhat expected) discrepancies, the general trends we observed are however confirmed. FLOODING is almost twice more expensive than the other protocols; T-RPL is the only protocol that has the highest overall reliability *and* highest overall efficiency.

11 Lessons Learned and Outlook

The premise of this paper was to challenge the current route-based design of RPL by showing that flooding enables, in alternative or synergy with RPL, the design of protocols that efficiently support reliable downward traffic. A key question, however, is what are the tradeoffs at stake. Can flooding by itself efficiently and reliably support downward traffic, and under what conditions? If not, can a hybrid protocol borrowing from both route- and flood-based approaches bring any benefits?

After the exhaustive evaluation in the previous sections we are now in the position to distill a few answers and lessons learned:

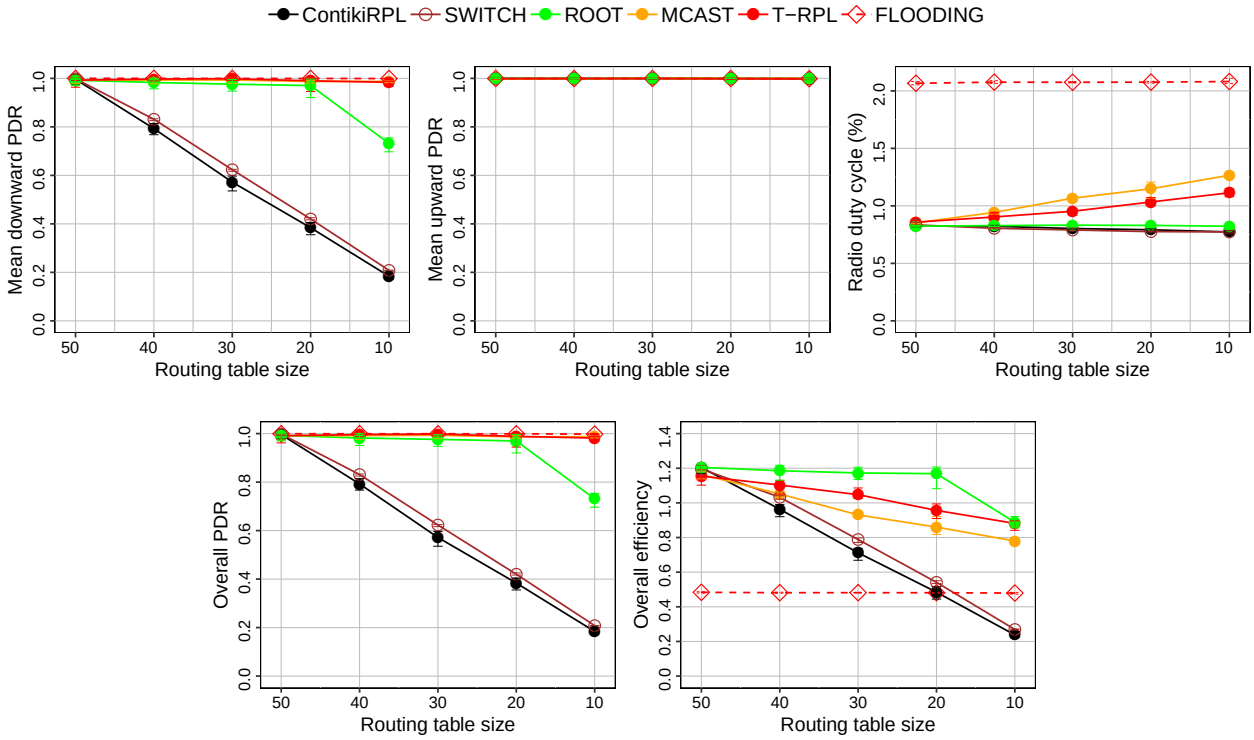


Figure 19: Testbed experiments: performance of combined upward/downward traffic vs. routing table size.

- **If energy consumption is not a concern (and the topology is "right") flooding is the answer.** Our results show that FLOODING generally achieves very high (and often perfect) reliability even in networks up to 225-nodes, i.e., 5 times more than what can be tolerated by ContikiRPL. However, they also evidence that energy consumption, which for FLOODING is essentially invariant w.r.t. network size, is much higher than in all other protocols considered. This said, if energy consumption is not a concern (e.g., because nodes are mains-powered) then FLOODING is arguably the best solution. One exception, however, is when the path diversity in the underlying network topology is limited, as in the linear topologies of our smart city scenario. In this case, FLOODING is not an option, as it yields unacceptable reliability.

- **T-RPL strikes the best tradeoff between reliability, energy consumption.** On the other hand, our T-RPL protocol is remarkably effective. It provides a near-perfect reliability, very close to FLOODING but for a fraction of the energy cost. Indeed, T-RPL *automatically* scales with the network size, exploiting the benefits of flood-based dissemination and, at the same time, limiting its drawbacks. Therefore, if energy consumption is a concern, T-RPL is arguably the protocol of choice.

- **T-RPL is the most versatile.** Compared to FLOODING, T-RPL is significantly less affected by the structure of the underlying network topology; in the topologies where dissemination is ineffective, the route-based operation T-RPL inherits from base RPL and SWITCH guarantees very high reliability. The same argument suggests that T-RPL is preferable over some of its constituent mechanisms or their pairwise combinations. For instance, our results show that in some cases ROOT+MCAST or even ROOT achieve similar (or even marginally better) results with less complexity. However, this occurs only in some topologies or some combinations of the target setup. In contrast, T-RPL is highly versatile, and guarantees similar levels of performance *without assumptions*, e.g., about the scale or structure of the physical network. This is a key asset when applying a protocol in practical contexts.

These distilled findings and lessons learned are also the opportunity for a few reflections about how the

contribution put forth by this paper affects related research efforts.

We believe that the key contribution provided by this paper, beyond the concrete implementation of the T-RPL protocol and its quantitative evaluation, is the fundamental insight that *a mix of deterministic routing and non-deterministic dissemination unlocks tradeoffs hitherto unexplored in protocols supporting downward routing.* This may pave the way to a new breed of protocols. Indeed, we explored only some of the points in the design space between routing and flooding (Figure 6). Although the performance of T-RPL is remarkably good if compared with the base ContikiRPL, alternate designs that combine these two extremes in a different mix may strike even better tradeoffs.

Moreover, we argue that the insight above, and the general contribution of the paper, goes beyond the specific protocols and setup shown here. For instance, although we based our experimental study on a CSMA-based MAC protocol, a recent trend is to exploit RPL in combination with the TDMA- and FDMA-based protocol framework offered by IEEE 802.15.4 TSCH [**?**]. This has a twofold impact in relation with the work presented here. On one hand, we expect that benefits similar to those shown here can be harvested also when RPL is used atop TSCH, given that the crux of the matter lies within the *routing* layer. In other words, we expect that the good performance shown by some network stacks combining RPL and TSCH [**?**] can be further improved by borrowing some of the ideas presented here, if not directly the T-RPL protocol. On the other hand, TSCH may also be beneficial to our routing protocols. Indeed, one of the reasons undermining reliability in several of our protocol variants, including T-RPL, is the unreliability of the layer 2 broadcast used by the ROOT mechanism, exacerbated in the presence of high contention. As the latter is essentially removed by design in TSCH, we expect T-RPL to achieve a near-perfect *PDR*, removing the need to consider pure FLOODING among viable alternatives.

12 Related Work

Since RPL was standardized by IETF, several studies [**?**, **?**, **?**, **?**] have analyzed its support for upward traffic by focusing, e.g., on topology stability [**?**], energy efficiency [**?**, **?**], cross-layer optimization [**?**], security [**?**, **?**], and reducing control overhead [**?**]. In comparison, the literature on RPL downward communication is significantly less abundant.

Clausen *et al.* [**?**] criticized the very design of RPL in this respect, arguing that DAO transmissions generate too much control traffic overhead, nodes close to the root must store routing information for too many destinations, and relevant standard elements (e.g., DAO timers or DAO-ACK messages) are under-specified. Nevertheless, their arguments were entirely qualitative; a quantitative and experimental analysis of the impact of these shortcomings was not provided. In contrast, Kim *et al.* [**?**] evaluated the performance of RPL under different traffic patterns, showing that both reliability and energy consumption are negatively affected when downward traffic becomes dominant. As a solution, the authors proposed a novel link quality estimator better suited for downward traffic. Although potentially beneficial, this solution does not remove the fundamental limitation concerned with memory vs. network scale we addressed in this paper.

The problem of limited storage for routing tables was mentioned by Ko *et al.* in their comparison [**?**] between TinyRPL and the Collection Tree Protocol (CTP) [**?**] from which RPL inherits several techniques. The authors noted that the default size of the routing table in TinyRPL is 30, which implicitly becomes the maximum number of nodes reachable by downward traffic. However, the implications of this constraint on reliability were not investigated further.

The limited storage for routing tables has been addressed by Ghaleb *et al.* [**?**] via a DAO-NACK mechanism enabling rejected nodes to search for another DAO parent, similar to our SWITCH mechanism. However, the authors rely on hop count as the criteria to select the DAO parent without taking into account the quality of the links between nodes, unlike our solution. Also, their solution is evaluated only with the unrealistic unit disk graph model (UDGM) and on a simulated network of unspecified size. The latest version (3.0) of ContikiOS also introduced a DAO-NACK mechanism; however, the reaction of a node upon its reception is completely different w.r.t. SWITCH. Instead of searching for an alternative DAO parent, the node artificially increases the ETX value of the link towards the preferred parent, and performs a local repair. This is a dangerous operation, as its final result changes the preferred parent, which *i)* induces a lot of unnecessary

network overhead, and *ii)* does not solve the limited storage problem, rather simply redirects it to another node. Finally, the Bounding Degrees RPL (BD-RPL) [**?**] follows the opposite approach of limiting in advance the number of children a parent can accept. None of these solutions solve the problem for the nodes close to the root.

The similar problem concerning the size of neighbor tables was also noticed in [**?**] and [**?**]. The authors observed that when these tables become full an undesirable churn appears in the network; entries are dropped only to be later re-discovered via lower-quality links. We observed a similar behavior in our experiments and, in contrast with these works, proposed effective solutions that significantly reduce the occurrence of this phenomenon.

The interplay between neighbor and routing tables was first recognized explicitly in [**?**], whose findings are at the core of this paper. Eriksson *et al.* [**?**] also recognized both problems, and addressed them with a technique that, similarly to SWITCH, uses a DAO-NACK to trigger a search for an alternative DAO parent. However, their approach is slightly more rigid than ours, in that *i)* they introduce an end-to-end DAO-(N)ACK mechanism in which a node accepts a new downward route only if all upward nodes on the path to the root have accepted it *ii)* the neighbor table is pre-provisioned with a fixed number of entries dedicated to children (for downward routing) and another for neighbors to be considered as backup candidates for the preferred parent.

Some researchers addressed the scalability problem by combining the storing mode we focused on with the non-storing mode also offered by RPL. The standard prevents co-existence of the two modes in the same network; however, their combination is meaningful given that they have complementary tradeoffs (§2). MERPL [**?**] resorts to source routing for nodes that became unreachable due to the inability of forwarders along the path to store all required routing entries. However, this approach assumes that the root has global, up-to-date knowledge of the DODAG rooted at it; this aspect is not considered in the evaluation, which relies on a custom Python simulator whose channel model is not specified. Resource aware hierarchical RPL (H-RPL) [**?**] sets the mode of operation of a node based on the resources available to it and its neighbors. However, the (potentially high) overhead of maintaining this information is neglected. DualMOP-RPL [**?**] explores a different angle by improving the interoperability of devices with different capabilities and resources. The techniques employed are proven effective in improving resilience to failures and therefore connectivity; however, they do not solve the memory vs. scale problem in situations where only resource-constrained devices are available.

Several works [**?**, **?**, **?**, **?**] have also addressed the problem of routing in the presence of mobility, as RPL restricts a mobile node to join the network only as a leaf. Although T-RPL does not address mobility directly, we observe that its use of dissemination as an alternate channel to bypass broken downward routes in principle could be beneficial also in this context.

Finally, in recent years the notion of synchronous transmissions made popular by Glossy [**?**] has been the basis for several protocols that rely on its energy-efficient, low-latency, highly reliable network flooding [**?**, **?**, **?**]. This technique could be an alternative to the conventional one in FLOODING, likely unlocking additional improvements in reliability and lifetime. On the other hand, we observe that a current limitation of Glossy and derivatives is that their implementation requires fine control of low-level hardware timing issues tied to the target radio and MCU chips, rendering interoperability difficult and also complicating integration in the RPL stack. In contrast, although the modifications we proposed to RPL are outside the standard, they do not depend on low-level hardware features and are easily implemented across different platforms.

13 Conclusions

RPL is among the few existing protocols catering for both upward and downward traffic, therefore targeting the needs of modern applications of low-power wireless. Nevertheless, downward traffic in RPL is significantly less optimized than its upward counterpart, as we showed quantitatively in this paper. As the culprit is the amount of memory devoted to neighbor and routing tables, which grows with the size and density of the network, RPL under-performs in large networks of resource-scarce devices—ironically, the scenario it originally targeted.

The basic premise of this paper is that a flood-based strategy may yield significant advantages w.r.t. the route-based one adopted by RPL. Therefore, we investigate three mechanisms tackling the aforementioned memory problem. The first one (SWITCH) improves route maintenance in RPL by exploiting aspects of the standard commonly neglected by popular implementations, and therefore provides a more accurate baseline of what can be achieved by the RPL standard. At the other extreme, we investigate the impact of exploiting increasing degrees of route-less dissemination, by *i)* replacing unicast with broadcast at the root (ROOT), often the most overloaded node, and *ii)* exploiting multicast (MCAST) to "patch" routes broken at nodes with saturated neighbor and/or routing tables. Our evaluation of the individual and combined impact of these mechanisms on the performance of RPL, carried out both in simulation and in a real-world testbed, shows that the best tradeoffs among scalability, reliability, and energy consumption is provided when these three mechanisms are combined. The resulting protocol, T-RPL, achieves a very high reliability, close to flooding but at a fraction of the energy cost. Further, while specific combinations of the aforementioned mechanisms perform marginally better in specific setups, T-RPL is the only protocol that yields top performance across all system configurations we tested. Our implementation of T-RPL is available as open source at `https://github.com/d3s-trento/T-RPL`.

The results above are, to the best of our knowledge, unprecedented in the RPL literature. However, the contribution of the paper goes beyond the concrete protocol(s) we design, implement, and evaluate, as it shows that a carefully balanced mix of routing and flooding strikes performance tradeoffs not achievable by either individual approach. This design strategy, which proved effective in improving popular implementations of RPL, can find applicability in more recent developments (e.g., TSCH-based network stacks) or even protocols other than RPL, where it may enable hitherto impossible tradeoffs between the efficiency of routing and the resilience of flooding.