UNIVERSITÀ DEGLI STUDI DI TRENTO
Dipartimento di Ingegneria
e Scienza dell'Informazione

Harmonizing Actuation and Data Collection in Low-power and Lossy Networks: From Standard Compliance to Rethinking the Stack

Timofei Istomin

Trento, 2017

Abstract

Technology is evolving towards a higher degree of automation and connectivity, with concepts of Pervasive Computing, Smart Factories, Cyber-Physical Systems (CPS) and the Internet of Things (IoT) promising to integrate countless communicating devices into objects around us at home, in the streets, and on industrial sites. These embedded devices are often very small in size, autonomously-powered, and have restricted computational and communicational capabilities.

Low-power and Lossy Networks (LLNs) are multi-hop, typically wireless, self-organising networks aimed at interconnecting hundreds or thousands of such embedded devices. They inherit many techniques from wireless sensor networks, though going beyond their original task of collecting sensor readings. New applications comprising actuators, control loops, user interface devices and requiring connectivity of every "smart thing" with the Internet, pose new challenges to the network protocol stacks.

These stacks should not only efficiently support data collection from numerous low-power sensors, but provide scalable data forwarding in the opposite direction, making every single device in the LLN addressable and reachable from a central controller or from the Internet. This type of forwarding is needed to send commands to wireless actuators in the LLN or to enable request-response communication between a low-power device and a remote server. Control loops additionally require real-time guarantees from the communication system.

We demonstrate in this thesis that reconciling the battery lifetime with high reliability and low latency is still a challenge for existing protocols even at the scale of few hundreds of network nodes. Moreover, current techniques have a significant performance gap between their data collection and actuation forwarding components on memory-constrained platforms. This gap limits the applicability of the stacks, as the overall performance is determined by the weaker component.

Motivated by two real-life applications, we first study novel techniques that eliminate the performance gap in the standard IPv6 stack for LLNs, making the actuation traffic forwarding as performant as the data collection one in networks that are five times larger than what the original standard stack is able to support. Second, we demonstrate that the reliability of packet delivery in the standard-compliant solution is limited in practice at around 99% while its routing overhead causes significant inefficiency in energy consumption. Therefore, we change focus to a forwarding mechanism based on the principle of synchronous transmissions, made popular by Glossy. It is a recent and, thus, non-standard technique, known for excellent reliability, speed and energy efficiency of the flooding-based data dissemination service it provides. This service is a perfect match for actuation, but a similarly efficient data collection

protocol did not exist. To close this gap, we design CRYSTAL, a novel data collection protocol based on the same core principle of synchronous transmissions.
We show that, depending on the application, CRYSTAL reaches per-mille or even parts-per-million radio duty cycle. It does that with a packet loss rate lower than 10^{-5} under external Wi-Fi interference of a noisy office building, and provides a much higher reliability and energy efficiency than the state of the art under even stronger interference generated by JamLab.
We thoroughly evaluate the proposed solutions both in realistic simulations and two large-scale testbeds. We follow a principled approach based on understanding of the environment and the properties of the network topologies. The latter are acquired by our connectivity assessment tool Trident, which itself is one of the contributions of this thesis.
Through these contributions, this thesis pushes forward the applicability of LLNs, by improving their scalability, reliability, latency, energy efficiency and interference resilience, both in the context of an existing standard and in a clean-slate design. Further, by achieving this superior performance via network stacks that natively support both collection and actuation traffic, this thesis provides a stepping stone for applications that strongly rely on both, notably including the low-power wireless control applications.

Key words: low-power and lossy networks, wireless actuation, synchronous transmissions, routing protocols, internet of things, cyber-physical systems.

Contents

List of Figures

List of Tables

Introduction

Technology is evolving towards a higher degree of automation and connectivity, with concepts of Pervasive Computing, Smart Factories, Cyber-Physical Systems (CPS) and the Internet of Things (IoT) promising to integrate countless communicating devices into objects around us at home, in the streets, and on industrial sites.

Although connectivity has already become ubiquitous thanks to the undeniable success of Wi-Fi and cellular networks, it is still prohibitively expensive in terms of service subscription fees and/or energy consumption. An alternative technique for interconnecting numerous miniaturised autonomously-powered devices with highly constrained computational capabilities is to equip them with low-power short-range radios and organise them in a multi-hop network to cover distances larger than their communication range (tens of meters).

In the past, such multi-hop networks were known as *Wireless Sensor Networks* (WSNs) as they were mostly used for collecting sensor readings. Now they are advancing far beyond their original task and the term *Low-Power and Lossy Networks* (LLNs) is increasingly used to denote them, underlying the fact that the communication links in such networks are brittle, with relatively high packet loss rate and with properties varying in time. Unlike pure data collection, where individual sensor nodes might be even not addressable, in applications comprising wireless actuators or when the low-power devices interact with the Internet, bidirectional connectivity and individual addressability are required.

In the envisioned scenarios, there is typically a dedicated device that is either the source or the destination for *most* of the packets sent inside a LLN. This can be the central controller which accumulates knowledge about the system and commands actuators, and/or the *border router*, bridging all data flows between the "big" Internet and the LLN. In both cases, there is a need for organising a *multipoint-to-point* communication for the low-power devices to send data packets towards the dedicated node, and the opposite, *point-to-multipoint* communication from the central node towards the periphery of the LLN. Throughout this thesis we also use terms *collection* and *actuation* traffic, respectively, to denote these communication patterns, reflecting their common use in applications, or we distinguish them by the direction they follow in the routing hierarchy, namely, *upward* and *downward*.

New applications not only broaden the communication patterns carried out by the low-power wireless network, but also change the optimisation objectives for the underlying communication techniques. Unlike WSN research, where extending the battery lifetime was the main goal, the actuation component and the interactivity of modern applications bring reliability and timeliness of data delivery to the same level of importance. How strict these requirements are, depends on the application, as a delayed or omitted reaction of the system might be just annoying to the user [11] or, on the other hand, it might be disrupting the control loop dynamics, potentially causing damage to things or people [85]. Therefore, optimising for reliability and timeliness is becoming a trend in the research community that is pushing the low-power wireless technology towards more challenging control applications.

Still, whether these wireless systems are feasible is determined by their energy consumption. Ideally, deployed devices should be maintenance-free or at least guarantee a battery lifetime of several years [67, 85]. In this thesis, we argue that reconciling battery lifetime with high reliability and low latency is still a challenge even at the scale of few hundreds of devices, not to mention tens of thousands, the target for some application scenarios [29].

Motivation from real-world applications. This work is motivated by two application scenarios, both having at their core a multi-hop low-power wireless network for monitoring and control.

- The first scenario is built around an outdoor urban deployment comprising nearly a thousand wirelessly communicating devices on lampposts, densely covering several neighborhoods in the city of Trento, Italy. The lamppost devices are used to control the street lighting, but also provide a connectivity "backbone" for various wireless sensors and actuators located nearby, including pollution sensors, parking lot occupancy sensors and irrigation valves. In this scenario, the main challenge is the need for individual addressability of every device in combination with the scale and peculiarity of the physical topologies defined by the urban structure.

- The second scenario is characterised by its inclusion of a control loop. In our particular case, a WSN is used for adaptive lighting control in road tunnels[16]. The network collects periodic illuminance readings to inform the central controller, which dynamically maintains the desired light intensity along the tunnel by individually adjusting lamp brightness. Making the WSN satisfy the real-time requirements of the control loop while maximizing the WSN battery lifetime constitutes a challenge.

Both application scenarios suggest the use of a protocol stack for wireless LLNs that supports both data collection and actuation traffic. Although several such stacks exist, to the best of our knowledge, they were not evaluated in realistic scenarios as the ones we identified. Furthermore, as we show in this thesis, current techniques have a significant performance gap between their upward and downward forwarding components. This gap limits the applicability of the stacks, as the overall performance is determined by the weaker component.

Therefore, ***the goal of this thesis is to harmonize the support for data collection and actuation traffic in a unified stack of unprecedented reliability and performance.*** We achieve this goal both by improving existing protocols and creating novel ones.

We argue that protocol implementation is as important for the performance as its design, hence, we work with real implementations for a popular hardware platform, TMote Sky, which is commonly used in research as well as in publicly available network testbeds.

Contributions. The contributions of this thesis are grouped in three parts, which is reflected in the thesis structure. They mostly lie in the area of protocol design, but also include a deployment assessment tool for LLNs. We briefly describe the contributions next.

Part I. Working with real deployments requires understanding their connectivity properties: the quality of individual links, the background noise, the network degree and diameter and their variations over time. To systematically acquire these properties, we created Trident, a tool that allows running various types of deployment assessments, including week-long autonomous outdoor experiment campaigns, quick tests to guide new deployments, and testbed measurements.

Part II. RPL, the IPv6 routing protocol for LLNs, became our first choice for the lamppost network mentioned before thanks to the built-in support for both upward and downward communications and interoperability with the Internet. RPL is an IETF standard designed for a range of applications in home, office, urban and industrial automation [11, 85, 67, 29]. Moreover, there are several open-source implementations of RPL, among which we chose the ones bundled with Contiki OS and TinyOS.

RPL is optimised for data collection by design, however, we show that it is not only less efficient for actuation traffic, but it is often unable to deliver any packets to a part of the network. The reason is that the unicast downward forwarding used by RPL needs a significant amount of memory for keeping the network state, which limits the downward connectivity to around 50 devices on our hardware platform.

When the memory is insufficient to store the routes to all destinations, the only option to enable connectivity with every node in the network is to rely on a form of data dissemination. However, dissemination has scalability problems, too, though now in terms of network utilisation that grows with the network size. This increases energy consumption and potentially disrupts the data collection service of the stack due to increased contention in the medium.

Therefore, we propose a solution that mitigates the scalability limitations of RPL by balancing the memory use among devices and augmenting the unicast downward forwarding with various types of limited-scope dissemination. We implement this solution and demonstrate that its downward service scales up to hundreds of devices without affecting much the upward traffic nor the battery lifetime.

Part III. Even though we harmonized the upward and downward forwarding of the standard-based stack, the packet loss rate of both the original RPL implementation and our improved protocol did not go below one per cent in either traffic direction. Such packet loss rate is prohibitively high for many applications comprising control loops. We also identified that the routing overhead of RPL, as well as that of other popular protocols based on conventional multi-hop routing, constitutes a significant inefficiency in energy consumption.

Therefore, in Part III, we approach the problem from another direction and do so in the absence of any standard-imposed design constraints. We build on synchronous transmissions made popular by Glossy [38], a very fast, reliable and energy-efficient network flooding protocol. Although Glossy on its own provides an excellent point-to-multipoint dissemination service, we argue that a similarly efficient solution for multipoint-to-point traffic did not exist. This puts us in a symmetrical situation w.r.t. Part II, as now we need to optimise data collection to close the performance gap.

Indeed, even though Glossy has been used for data collection before, this is done by scheduling individual floods for every potential data source in the system. In the case of sparse, unpredictable, aperiodic collection traffic, such over-provisioning creates significant amount of idle listening when the sensor nodes do not have data to send. This type of collection traffic is generated, for example, by the data prediction technique in [89] which suppresses more than 99% of the packets in the tunnel application mentioned earlier.

To overcome this performance gap between efficient dissemination and expensive data collection, we designed CRYSTAL, a novel reliable data collection protocol based on synchronous transmissions. We demonstrate that CRYSTAL can dynamically adapt to the immediate traffic demands without requiring expensive scheduling or over-provisioning. Thanks to this, CRYSTAL is very fast at delivering occasional bursts of packets, consuming very little energy when the traffic is low, and it improves over the already remarkable reliability of Glossy.

The latency of data delivery in CRYSTAL is as low as 24 milliseconds per packet in a network of four hops. This time includes the end-to-end acknowledgement as CRYSTAL provides an *acknowledged data collection service* by design.

Regarding the reliability and energy efficiency, we show that, depending on the application, CRYSTAL reaches per-mille or even parts-per-million radio duty cycle. It does that with a packet loss rate lower than 10^{-5} under external Wi-Fi interference of a noisy office building.

Further, since industrial scenarios, which can benefit from CRYSTAL, are often characterised by the presence of interfering equipment, we devote a significant part of this work to understanding how particularly strong interference affects the properties of our protocol. For this, we use JamLab [9] to generate reproducible interference of various types, intensities and spatial distributions. We show that CRYSTAL sustains much higher levels of interference than state-of-the-art protocols, including Glossy itself, while maintaining its properties of being ultra reliable and ultra-low power.

Through these contributions, this thesis pushes forward the applicability of LLNs, by improving their scalability, reliability, latency, energy efficiency and interference resilience, both in the context of an existing standard and in a clean-slate design. Further, by achieving this superior performance via network stacks that natively support both collection and actuation traffic, this thesis provides a stepping stone for applications that strongly rely on both, notably including the low-power wireless control applications.

Part I
Assessing The Quality of Links and Topology Properties

1 Trident: A Connectivity Assessment Tool for Wireless Sensor Networks

Wireless sensor networks (WSNs) are finding their way into real-world applications, whose successful deployments are increasingly reported. This accrued experience has also evidenced how the task of deploying a WSN entails a number of challenges. One of the most important concerns the low-power wireless communication exploited by WSNs. This type of wireless communication has peculiar characteristics, that have been studied by many researchers; a summary is provided by a recent empirical study [92].

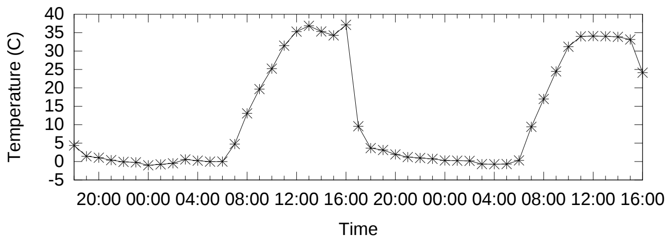
Motivation. Communication in the 2.4 GHz band of IEEE 802.15.4, commonly employed by WSN devices, is also affected by factors strongly dependent on the environment where the WSN is deployed, and hard to capture in the controlled setups typically used in the studies above. These factors include the shape of the deployment site [71], variations in temperature [7] and humidity [97] and their combination in outdoor [65, 103] or industrial [10] scenarios.

Figure 1.1 provides a concrete idea of the extent to which communication can be affected by these factors, albeit using a small-scale setup. A single link between two nodes communicating at 0 dBm and placed 40 m apart in an outdoor open field is observed over a 2-day period. Figure 1.1a shows the average temperature inside the plastic boxes holding the motes; Figure 1.1b shows that, for TMote Sky devices, this temperature is negatively correlated with the Received Signal Strength Indicator (*RSSI*) and Packet Delivery Rate (*PDR*) of the link. A 40°C temperature increase in the box (with a mere 11°C increase outside of it) causes a −13 dBm decrease in *RSSI*. This is enough to turn a perfect link ($PDR = 100\%$) into a dead one ($PDR = 0\%$).

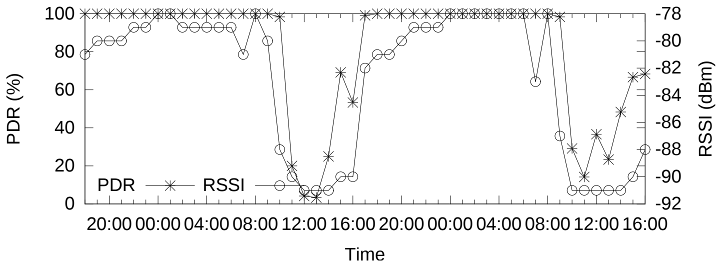
Assessing quantitatively the characteristics of low-power wireless links *in-field*, i.e., in the target environment where a WSN must be deployed, is key for several, intertwined goals:

- *supporting the WSN deployment*, by determining where to place the motes to ensure communication among them;
- *informing the selection (or design) of protocols*, to ensure they are well-suited to the target environment;

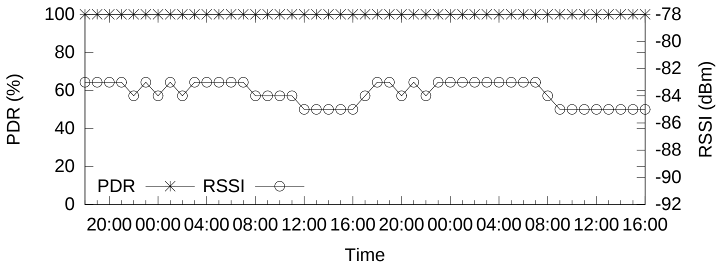
The contents of this chapter have been originally published in: **TRIDENT: In-field Connectivity Assessment for Wireless Sensor Networks**, Timofei Istomin, Ramona Marfievici, Amy L. Murphy, Gian Pietro Picco, In Proceedings of the 6th Extreme Conference on Communication and Computing (ExtremeCom), Galapagos Islands, Ecuador, August 2014, and have been slightly adapted for this thesis.



(a) Temperature.



(b) TMote Sky.



(c) Waspmote.

Figure 1.1 – Effect of temperature on *RSSI* and *PDR*.

- *deriving models*, to push the envelope of what can be predicted (or simulated) beforehand.

Unfortunately, the tools supporting connectivity assessment (e.g., SWAT [93], SCALE [18], RadiaLE [6]) were conceived to study the properties of low-power wireless links in controlled environments, e.g., an office testbed. Therefore, they require a secondary, wired networking infrastructure (e.g., USB cables) for gathering experimental data—a luxury one can rarely afford in real-world settings.

Contribution. This chapter presents TRIDENT [1], a tool expressly designed to simplify the chore of in-field connectivity assessment. TRIDENT relies only on the WSN nodes whose connectivity needs to be ascertained, without any external infrastructure.

TRIDENT is useful to WSN researchers and practitioners, who may use it towards any of the three aforementioned goals. However, the tool is designed to be easy to use also for domain experts who, being the “users”, have precise knowledge of where the WSN nodes should be deployed from the application standpoint, but typically do not have a very deep knowledge about the inner working of the WSN, and definitely do not take part in programming it. The connectivity experiments can be configured easily without requiring any coding, and the data collected with a straightforward procedure. Therefore, TRIDENT empowers domain experts with the ability to assess the connectivity of WSN nodes deployed in-field, without the need for a supporting WSN expert. The goals and requirements we set for TRIDENT are described in Section 1.1.

TRIDENT covers the entire workflow concerned with connectivity assessment experiments. After the WSN nodes are flashed with the TRIDENT software, appropriate user interfaces enable the operator to (re)configure the experiment settings, discover nodes, and download the results. The latter are stored in a database, simplifying the storage and analysis of the (typically large volumes of) data gathered. The main operation of TRIDENT is described in Section 1.2, along with the various components of the tool.

We originally developed TRIDENT for the popular research-oriented platform constituted by TMote Sky motes running TinyOS. However, connectivity assessment is relevant also to industry-oriented platforms, for which we chose Waspmote devices running the standard ZigBee stack as a representative. Interestingly, supporting the latter platform is not simply a matter of porting the code from the former; the fact that the ZigBee stack is "closed", unlike the TinyOS one, forced us to find ways to reliably measure the main metric of *PDR*, which cannot be derived directly otherwise.

Although both platforms are based on radio chips compliant with IEEE 802.15.4, their behavior is very different, as shown again in Figure 1.1. The same setup and observations we described earlier were also performed for a link between a pair of Waspmote nodes, whose behavior is shown in Figure 1.1c. The two experiments were performed in exactly the same conditions, by placing the two pairs of motes side-by-side on different channels. The chart shows that, for Waspmote, temperature is also negatively correlated with *RSSI*, but to a much lesser extent w.r.t. TMote Sky, allowing the *PDR* to remain at 100%. These sharp differences motivated different implementations of TRIDENT, discussed in Section 1.3. Finally, Section 1.4 ends the chapter with brief concluding remarks.

1.1 Requirements

In this section we describe the requirements we set for ourselves in designing TRIDENT, grouped according to their nature.

1.1.1 Type of Data Collected

Metrics. We want to support in-field collection of several metrics typically used to perform connectivity assessment.

The key metric provided is the *Packet Delivery Rate (PDR)*, i.e., the ratio of packets received over those sent. *PDR* provides a direct assessment of the ability of a link to reliably communicate packets.

A number of metrics are extracted directly from the *radio chip*: Received Signal Strength (*RSSI*), Link Quality Indicator (*LQI*), and *RSSI* noise floor, sampled after sending or receiving a packet. These metrics provide insights about physical layer parameters influencing *PDR*, and their correlation with it. Moreover, noise floor is useful to indirectly determine the presence of interference.

Finally, TRIDENT supports the acquisition, upon packet sending or receiving, of *environmental parameters* (e.g., temperature and humidity) from on-board sensors. These are useful to determine how the environment affects communication, as highlighted by the works mentioned earlier and concretely shown in Figure 1.1.

Aggregated vs. per-packet samples. The reason for which connectivity assessment is performed determines the nature of the data gathered. If a long-term observation is necessary, the amount of data recorded can rapidly become prohibitive for a resource-constrained platform like TMote Sky. Therefore, the tool should support the ability to store only an aggregate of the metrics collected, computed over a well-defined sequence of packet exchanges.

On the other hand, recording the individual metrics (e.g., *RSSI* and noise floor) associated with *each* packet provides a fine-grained detail that is necessary in some cases, e.g., when the goal is to ascertain the time-variant characteristics of links with the resolution necessary to inform the design of network protocols.

The choice between aggregated or per-packet samples should therefore be left to the user, balancing the goals of connectivity assessment against the storage limitations of the platform at hand.

Single packets vs. bursts. Another related dimension is the way the channel is observed. Connectivity assessment is often performed by sending *probe* packets with an inter-message interval (IMI) relatively high (e.g., seconds), representative of several WSN applications. Nevertheless, some applications (e.g., recording data from accelerometers [104, 17]) require the transmission of bursts of packets. Moreover, a well-known property of the wireless channel is that the transmission of packets sent with a small-enough IMI [94] is more reliable. The tool should allow the user to choose whether to use single packets or bursts of packets as probes.

1.1.2 Type of Experiments Supported

Interactive vs. batch. Connectivity assessment may be needed for reasons yielding different requirements, as described earlier.

If the ultimate goal is to support a WSN deployment by helping determine a node placement enabling communication, this is often achieved by performing tests of short duration (e.g., a few minutes). These are useful to quickly evidence which nodes experience low *PDR* values;

this information is used by the operator to relocate nodes and re-assess connectivity with another short test. To effectively support this process, the tool must provide a way to quickly (e.g., graphically) represent the *PDR* associated with WSN links.

On the other hand, connectivity assessment is often performed also to *characterize* the target environment. This requires long-term observations (e.g., days); the continuous presence of an operator would be impractical. The tool must provide the option to run *automatically* a battery of tests, including different settings, defined by the operator but executed without her involvement.

In our experience, the two modes of operation are often used in conjunction. Indeed, before starting a long-term batch experiment, a short-term interactive one is performed, to make sure that all nodes are functioning properly, and that the baseline connectivity is appropriate to the purpose of the experiment.

Mobile nodes. Mobile WSN applications, e.g., involving nodes placed on humans, animals, robots, or vehicles, are becoming increasingly popular. Therefore, the tool should support experiments where some of the nodes are mobile, to assess the connectivity between these and the fixed nodes. An interesting possibility, partially explored in our previous work[15], is to use mobile nodes as a way to perform a preliminary exploration of the deployment area. As the mobile node moves across the field and exchanges messages with fixed nodes, it samples the connectivity of a high number of locations, cumbersome to explore individually only with fixed nodes.

1.1.3 Support to Operators

In-field, wireless interaction with nodes. In-field operators must interact with the nodes for various purposes. The primary reason is to retrieve the results of experiments, stored on the nodes taking part in them. Another key operation is the re-tasking of the nodes with a new set of experiments. The operator may also need to interact with the nodes for the sake of monitoring the correct execution of the experiment, e.g., to retrieve statistics about the experiments performed or the battery level. Other useful operations are the ability to put selected nodes (or the entire WSN) in stand-by when they are not involved in an experiment, and wake them up later on.

In principle, some of these operations can be performed by directly accessing the node; for instance, data can be downloaded via USB. However, while this operation is trivial in a lab, it becomes cumbersome when nodes are deployed in-field in a harsh environments, e.g., outdoor in winter, or in places that are not easily accessible. Therefore, all of the interactions with the nodes should be performed over-the-air, by leveraging the wireless channel.

Data storage and processing. Connectivity assessment experiments may generate a huge quantity of data. Handling these as individual files becomes rapidly impractical. Further, the raw data gathered often needs to be processed in an automated way to simplify interpretation. The tool should therefore integrate a database for storing experimental data, enabling structured access and querying, and the definition of stored procedures providing a layer of abstraction in data manipulation and interpretation.

1.1.4 Non-functional Requirements

No infrastructure. As we already argued in the introduction, this non-functional requirement is a defining one. In-field experiments cannot afford the luxury of a secondary network; the experiment execution must rely only on the WSN nodes under test.

No coding required. Our desire to support domain experts implies that using the tool should *not* require writing even a single line of code. The configuration of experiments should occur entirely via the user interface; at most, domain experts must learn how to flash motes with a pre-canned binary before going in-field.

Ease of use. The logistics of in-field experiments makes them effort-demanding and time-consuming. The situation should not be exacerbated by a complex or cumbersome interaction with the tool. A graphical user interface, providing intuitive support for all the phases of the experiments, is therefore an obvious requirement.

Flexibility. The experiment settings, including number of nodes, their nature and role in the experiment, power and channel settings, number of messages, inter-message interval, number of test repetitions, etc., should be designed in a way that allows users to combine them freely, to explore different portions of the parameter space.

Decoupling from hardware platform. The tool should work on standard nodes without modifications to hardware. Nevertheless, as there are many WSN platforms available, supporting a new one in the tool should be simplified by its software architecture, by confining platform-specific details in well-identified components.

1.2 Design

Next we describe the design of TRIDENT. We begin with a description of the execution of connectivity assessment experiments, then provide an overview of the TRIDENT toolset.

1.2.1 Experiment Execution

Definitions. An entire TRIDENT experimental campaign is defined as a sequence of *rounds*, as shown in Figure 1.2. Each round has its own configuration parameters, detailed next, including whether it uses single-packet probes or burst probes, i.e., multiple packets transmitted in rapid sequence. The time between the beginning of two consecutive probes from the same node is the *inter-probe interval (IPI)*. For burst probes, we also define the *inter-message interval (IMI)* as the time between two messages belonging to the same burst. Both IPI and IMI are configurable on a per-round basis.

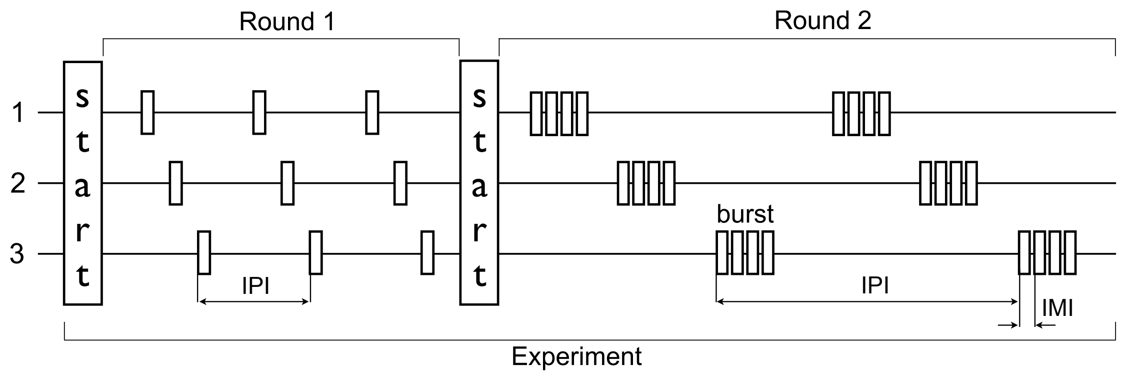


Figure 1.2 – Sample TRIDENT experiment showing two rounds, staggered transmissions using single-packet and burst probes, and per-round synchronization.

Probing the links. Connectivity is assessed by probing the communication link with packet transmissions and evaluating the received packets and their properties. Therefore, TRIDENT experiments must define precisely *when* each node transmits and listens.

All nodes behave the same: transmit a probe, pause for the IPI duration, repeat this process a configurable number of times. In between transmissions, nodes can be configured to listen for packets from other nodes. TRIDENT does not duty-cycle the radio during experiments, ensuring that no packets are lost due to the radio state. Moreover, to avoid collisions among simultaneously transmitted packets, which can confound the link evaluation, TRIDENT ensures that only one sender transmits at any given time.

This is achieved by having nodes begin their probe sequence in a staggered way based on their node identifiers. Specifically, the transmit time for the i-th probe of node n is defined as

$$t_{n,i} = t_{start} + nT_{IPI} + iNT_{IPI}$$

where t_{start} is the start time of the round, T_{IPI} is the value of the IPI, n is the node identifier, and N is the overall number of nodes. n and N are setup statically during the experiment design phase.

Staggering transmissions by assigning each nodes its transmit slot, requires the nodes to be time-synchronized. In TRIDENT this is achieved at the beginning of each round, as shown in Figure 1.2. This synchronization allows the system to compensate for clock drift during a long running experimental campaign.

Master node. Time synchronization is initiated by a special node, called the *master*. The latter acts in general as a coordinator towards the rest of the WSN nodes, as well as the “access point” enabling the operator to change the configuration of experiments. The master has the same binary code of the other nodes; its special role is determined by its identifier, $n = 0$.

The parameters describing the round configuration are also disseminated by the master node during synchronization at the beginning of each round. The option to change parameters

Table 1.1 – Per-round configuration parameters and their example values from [65].

Parameter	Example	Parameter	Example
# of nodes (N)	16	# of probes per sender	115
list of senders	0...15	single-packet probe	yes
list of listeners	0...15	inter-probe interval	16 s
transmit power	27(−1 dBm)	inter-message interval	N/A
probing channel	18	aggregate logging	yes

in each round allows the interleaving of rounds with different power levels, or interleaving single-packet and bursty rounds, as shown in Figure 1.2.

This choice has multiple consequences. First, only the master node is aware of the experimental campaign, and therefore is the only one affected by changes to the latter. As the master can receive an entire experimental campaign configuration over-the-air, physical access to nodes is not required to change or initiate a campaign. Second, prior to starting a long-running campaign, the operators can interactively run a number of small experiments, each time uploading the round description to the master, instructing the master to initiate the round, then collecting the results. After analysis, another short experiment can be carried out, or the long-running experiment can be initiated. This is all possible because the nodes are experiment-agnostic and the master can be controlled over-the-air.

Per-round configuration parameters. Table 1.1 shows the per-round configuration parameters available in TRIDENT, communicated by the master before starting a round. We already mentioned some of them, e.g., the overall number of nodes, the role (sender or listener), the type of probe, the values of IPI and IMI. Additional parameters include the radio channel, transmission power, overall number of probes transmitted per node, and number of packets

Table 1.2 – In-field commands.

Command	Target	Description
UPLOAD	master	load a campaign configuration
START	network	start the execution of an experiment
STOP	network	stop the execution of an experiment
POLL	1-hop	query battery level
SLEEP	1-hop	place nodes in low-power listening
WAKE-UP	1-hop	remove nodes from low-power listening
DOWNLOAD(n)	node	download logs from selected node
ERASE(n)	node	erase flash of selected node
SNIFF	operator	toggle packet sniffing

per burst. The logging method must also be defined, choosing between storing information about every packet, or only the average over the entire round, based on the needs of the experiment and the available storage. Finally, rounds and/or entire experiments can be repeated a configurable number of times, to increase statistical relevance.

As for metrics, not shown in the table, *PDR*, *RSSI*, and (if available) *LQI* are collected by default. If the platform allows, sender and receiver can acquire per-packet environmental conditions such as noise floor, temperature, and humidity. These values are recorded according to the per-round logging policy.

Interacting with the nodes under experiment. Changing the per-round configuration parameters is not the only option to interact in-field with nodes. Table 1.2 describes the commands available to the operator. As already mentioned, the operator can upload the description of an entire campaign configuration on the master, which then uses it to orchestrate the various rounds with the appropriate per-round configuration. However, the operator can also start and stop the execution of the experiment; these commands are propagated network-wide, with a mechanism similar to the one used to mark the start of a single round, described in Section 1.3.

The master node can instruct all nodes to automatically enter a sleep state upon the end of an entire experimental campaign, allowing them to save their remaining battery power. In TinyOS, nodes in this state use the default low-power listening (LPL) MAC with a long sleep interval, currently set to 1 s. By duty-cycling the radio, nodes save battery power, but can be woken up later, e.g., to initiate over-the-air data download or to start a new experimental campaign. Alternately, the operator can also put to (or wake-up from) sleep a subset of the nodes, and query for their battery level.

Other commands enable the operator to download the experiment logs from a selected node, and to erase its flash memory after successful data transfer is verified. Finally, passive packet sniffing can be activated on the node managed by the operator, enabling the latter to ascertain whether all nodes properly started the experiment.

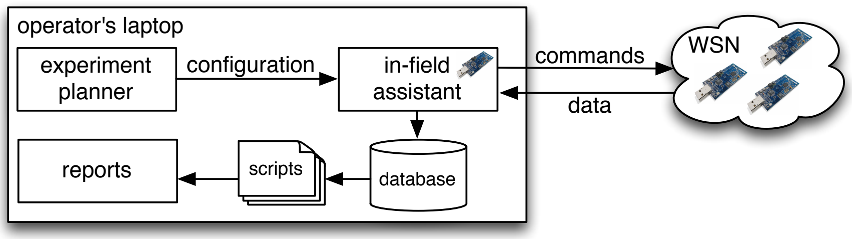


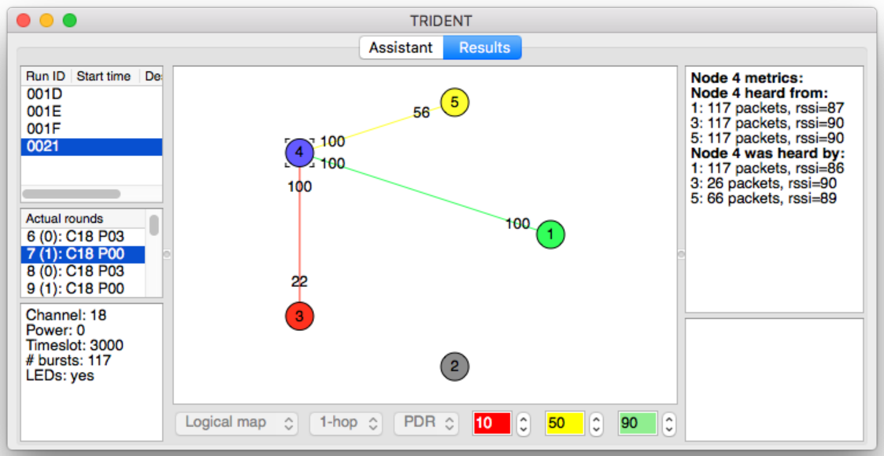
Figure 1.3 – The TRIDENT toolset.

1.2.2 Toolset Overview

Figure 1.3 depicts the main structure of TRIDENT. WSN nodes, the subject of the experiment, are programmed with a *platform-specific mote-level runtime* that is experiment-agnostic; its behavior is established by the operator without requiring any coding.

This configuration is performed through the GUI of a dedicated component, the *experiment planner*, which resides on the operator's laptop. The experiment planner essentially enables the operator to quickly and easily define the various details of the experiment, by properly setting the values for the parameters in Table 1.1. This step does not need to be performed in-field, as the planner enables only the definition and storage of experiments.

The actual upload of experiments to the master, and from there to the rest of the WSN, is instead supported by the *in-field assistant*, which also enables the execution of the other commands in Table 1.2. Communication with the WSN is enabled by a mote acting as a gateway, connected to the USB port of the operator's laptop.



Run ID	Start time	De
001D		
001E		
001F		
0021		

Actual rounds
6 (0): C18 P03
7 (1): C18 P00
8 (0): C18 P03
9 (1): C18 P00

Channel: 18
Power: 0
Timeslot: 3000
# bursts: 117
LEDs: yes

Node 4 metrics:
Node 4 heard from:
1: 117 packets, rssi=87
3: 117 packets, rssi=90
5: 117 packets, rssi=90
Node 4 was heard by:
1: 117 packets, rssi=86
3: 26 packets, rssi=90
5: 66 packets, rssi=89

Figure 1.4 – The "results view" of the TRIDENT in-field assistant, showing one-hop connectivity and *PDR* for node 4.

The in-field assistant provides also a simple visualization of the connectivity map built from available collected traces. An example is shown in Figure 1.4, visualizing the quality of the inbound links for node 4 according to an intuitive green/yellow/red color-coding, whose semantics in terms of *PDR* is configurable. This feature is particularly useful when TRIDENT is used for a short-term assessment, as it quickly informs the operator about areas with connectivity problems, whose nodes can then be re-arranged. A similar visualization is provided also for mobile nodes; once the in-field assistant is fed with a sequence of locations, it can "replay" the maps, showing to the operator how connectivity evolved due to mobility.

Finally, the *database* and the associated plotting *scripts* simplify the storage of the collected

Table 1.3 – Platform support in TRIDENT.

hardware	TMote Sky	Waspmote + XBee
software	TinyOS	Arduino
PHY layer	802.15.4 (2.4 GHz)	802.15.4 (2.4 GHz)
radio chip	CC2420	EM250
TX power	$-25\ldots0$ dBm	$-8\ldots2$ dBm
metrics	RSSI, LQI, noise	RSSI
burst probes	yes	no
storage	flash chip, 1 MB	microSD, 2 GB
aggregate logging	per round, on motes	on operator's laptop
sensors	temperature, humidity	—

data and its offline analysis. The database contains generic and customizable stored procedures for data manipulation. The set of pre-canned scripts allows the user to quickly plot trends derived from the data collected, e.g., currently including network-wide *PDR*, cumulative distribution functions of the links w.r.t. their *PDR*, spatial and temporal variations of the metrics, correlation of *PDR* with *RSSI* and *LQI*.

1.3 Platform-specific details

TRIDENT currently supports two hardware platforms: TMote Sky and Waspmote. The former directly integrates the CC2420 radio chip, while the latter relies on an extension module for radio communications, in our case the XBee S2 integrating the ZigBee-compliant EM250 system-on-chip. Both transceivers implement the 2.4 GHz IEEE 802.15.4 physical layer.

The software architecture of TRIDENT confines the differences mostly in the platform-specific runtime installed on motes, although a few changes are needed also to parts of the in-field assistant handling communication with the WSN and parsing the logs for visualization. We refer to these platform-specific portions of the software as the *backend*, and summarize the differences in Table 1.3. The Waspmote variant provides less features than the TMote Sky counterpart, as TinyOS allows low-level access to the radio chip while Waspmote provides only the high-level interface of the ZigBee application support sublayer (APS). These trade-offs are discussed in the rest of the section, along with other implementation details.

Available metrics. The two platforms provide different metrics. TinyOS records *RSSI* and *LQI* for each received message, while the XBee radio module reports only *RSSI*. Moreover, unlike ZigBee, the low-level API available to TinyOS applications allows requesting *RSSI* when the channel is idle to measure the noise floor.

The temperature and humidity sensor of TMote Sky provides important information for our studies of the environmental effects on connectivity. In principle, the same holds for Waspmote, given the wide range of sensors available for this platform. However, we have not yet implemented support for them in TRIDENT, although this does not pose any particular technical problem.

Experiment execution. On both platforms the experiment configuration is installed in the non-volatile storage of the master node. The TMote Sky backend supports uploading the configuration wirelessly or via USB, and stores it in a dedicated flash partition, while the Waspmote backend relies on a configuration file on the SD card.

As described in Section 1.2, the network is time-synchronized at the beginning of each round to avoid collisions among probes. The backends implement different techniques. In the case of TMote Sky, dissemination relies on a TDMA scheme where each node has its own time slots to repropagate commands, in a way similar to the mechanism outlined for probe transmission in Section 1.2. The common time reference needed for both the TDMA dissemination phase and to calculate t_{start} is established with TinyOS packet-level time synchronization service [66], yielding sub-millisecond precision.

As ZigBee does not provide such a synchronization service, we rely on the standard multi-hop broadcast feature, basically a network flooding. However, based on the ZigBee Pro feature set [109], broadcast packets are *always* sent 3 times in a row, increasing reliability at the expense of energy consumption. These broadcast packets are separated by a 500 ms interval plus a random delay between 0 and 40 ms. Therefore, in the worst case where the start synchronization message is received only upon third attempt, the time synchronization error goes slightly above 1 s per hop.

To secure a collision-free transmission schedule the IPI should be set long enough to compensate for these synchronization errors and also for the time drift of the nodes. On TMote Sky the synchronization error is negligible; the typical time drift of 100 ppm results in two nodes drifting apart by 36 ms in half an hour. Therefore the use of 250 ms time slots during 30-minute rounds can be considered safe, counting also the time needed for internal processing of the received packets. Instead, on Waspmote the second-per-hop error should be compensated; we achieve this by using 3 s time slots.

Moreover, transmission of probes as one-hop broadcast requires an additional second, again due to the triple transmission performed by the ZigBee network layer. Therefore, it is impossible to send bursts of packets with Waspmote; on TMote Sky, bursts are instead supported with a configurable IMI, set to 20 ms by default.

Another implication of ZigBee compliance is that nodes must join the wireless PAN (personal area network) before sending application data. A multi-hop ZigBee network is built around its coordinator with the standard join procedure, including channel scanning and handshaking, one hop at a time. This process requires up to minutes, depending on the network diameter. In case the channel is changed in between rounds, this affects the minimum interval between them, as the network topology must be rebuilt from scratch.

Determining the link-level PDR. In principle, the value of *PDR* can be obtained straightfor-

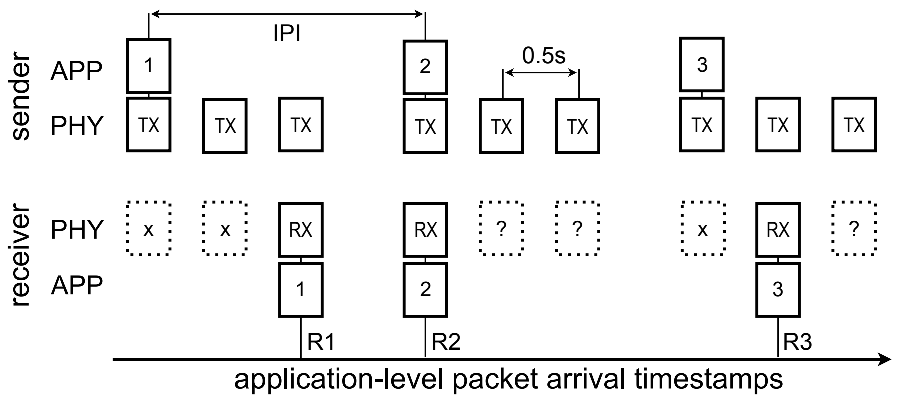


Figure 1.5 – ZigBee transmits each broadcast packet three times.

wardly, since the number of transmitted and received packets is known for each link. However, Waspmote introduce additional complexity due to the triple transmission of broadcast packets mandated by the ZigBee network layer. As shown in Figure 1.5, since duplicates are automatically filtered at the receiver, the link-level *PDR* cannot be determined by simply counting the delivered packets at the application layer. Consider two hypothetical experiment runs, one where each probe sent is always received upon first attempt, and one where it is received always upon the third attempt. The link-level *PDR* would be a meager 33% in the second case, yet the application-level *PDR* (the only directly measurable) would be 100% in both cases, providing a false representation of the quality of the wireless link.

Nevertheless, we can infer the number of failed attempts by looking at the packet arrival time, based on the fact that the three broadcast transmissions in ZigBee are spaced relatively far apart (500 ms). It is therefore possible to determine, upon receiving a broadcast packet, whether this was the first, second, or third transmission. We confirmed this experimentally: Figure 1.6 shows the distribution of the packet arrival interval (modulo the nominal IPI) measured at the application level for an intermediate-quality link. For a perfect link, all packets would be received with the same IPI, 15 s in this case; however, this is not the case when packets are lost. Consider an application-level packet i, received on first attempt. If

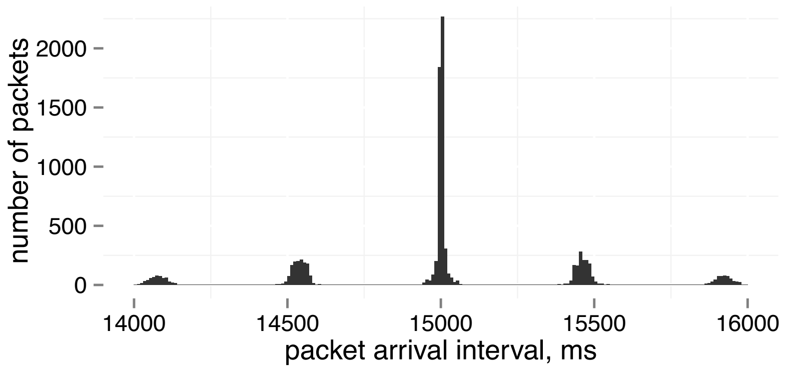


Figure 1.6 – Distribution of packet arrival interval.

the previous packet, $i-1$, was received only on second or third attempt, the IPI between the two packets is smaller than 15 s. A similar reasoning holds in case the next packet $i+1$ is not received upon first attempt, yielding an IPI greater than 15 s. Clearly, the histograms to the left and right of the central one in Figure 1.6 can be generated also by intermediate combinations, e.g., if i is received upon second attempt and $i+1$ upon third, the IPI will be around 14.5 s.

Packet loss can be inferred by comparing the application-level packet arrival timestamps (e.g., R_1 and R_2 in Figure 1.5), provided there is at least one packet in the received sequence known to have arrived upon first attempt. This can be stated certainly when at least one pair of packets (not necessarily consecutive) has either the minimum or maximum possible recorded arrival interval (modulo the IPI), i.e., placed leftmost or rightmost on the histogram of Figure 1.6. Indeed, this means that the one of the packets was delivered on first attempt and the other on last, as in the case of R_1 and R_2.

In principle, it may happen that no such IPI is recorded during the whole test. In practice, this is unlikely to happen for intermediate-quality links. However, one can still infer the characteristics of the link based on the application-level *PDR*. If the latter is very good, one can assume that the majority of the packets were received on the first attempt and base the analysis on this fact. For very bad links, it may be impossible to measure the actual *PDR* precisely when there are just few packets received from the whole sequence.

Storing the experiment data. Due to the storage limitations of TMote Sky, per-probe logging can be replaced by storing per-round averages of the recorded values, computed on the nodes themselves. Waspmote does not have strict storage capacity limitation; full logs are always stored and the log analysis performed on the operator's laptop. However, we plan to implement on-board log processing also on Waspmote, to reduce the downloading time of large logs.

1.4 Conclusions

In this chapter we presented TRIDENT, a tool for in-field connectivity assessment. Unlike similar tools in the literature, TRIDENT does not require any communication infrastructure besides the WSN nodes. Moreover, it is designed to be easy to use for domain experts, which can perform their connectivity experiments (e.g., to determine a correct placement of WSN nodes) without coding. TRIDENT supports several configuration parameters and metrics, enabling the investigation of many aspects of low-power communication. Moreover, the operator's interface provides a rich set of commands that simplify in-field management of experiments.

We report connectivity properties of multiple deployments we dealt with throughout this thesis. All of them were acquired by TRIDENT.

Part II
Downward Forwarding in Standard IP-based Protocol Stacks

2 RPL, the Routing Standard for the Internet of Things... Or Is It?

RPL, the IPv6 Routing Protocol for Low-Power and Lossy Networks [105], was standardized by the IETF in 2011 to establish a common ground for building interoperable commercial appliances in growing markets enabled by low-power and lossy networks (LLNs). Meanwhile, the Internet of Things (IoT) emerged, envisioning ubiquitous and global connectivity among the billions objects that we use in the everyday life.

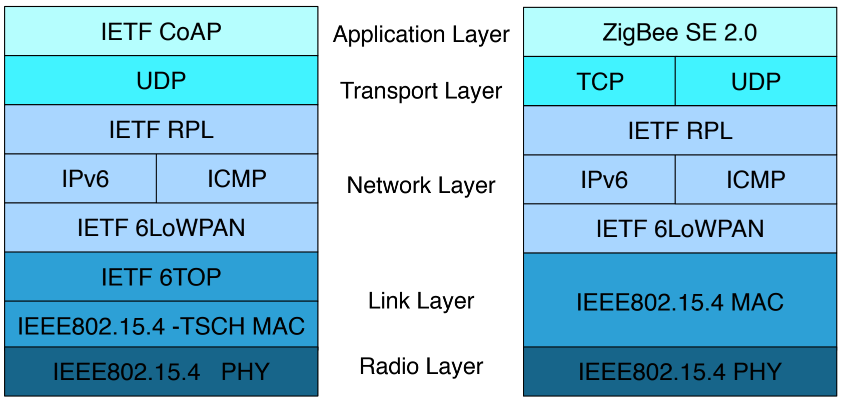
Given the significant overlapping between LLNs and IoT, and the fact that IPv6 is an essential feature for the latter, RPL has rapidly become *the* routing protocol for IoT, incorporating the protocol stack defined by IETF in this scope, atop the IEEE 802.15.4 MAC and PHY layers (Figure 2.1a). The success of RPL as an IoT standard is also witnessed by the fact that the companies part of the ZigBee Alliance have adopted RPL as their underlying technology (Figure 2.1b).

Therefore, RPL became our first choice protocol for one of the application scenarios that motivated this thesis, the urban lamppost network. This scenario is presented in detail in the next chapter, together with a thorough evaluation of the protocol performance and several proposed improvements. This chapter presents a high-level analysis of the main features and limitations of RPL.

The design goals behind RPL date back to the routing requirements determined by IETF in 2009 by considering the applications domains of home and building automation [11, 67] and of urban and industrial networks [29, 85]. After several years, it is important to ascertain the extent to which these requirements have been fulfilled by RPL; we offer our analysis of the state of the art in Section 2.2.

A similarly important question, however, is whether RPL is well-prepared to sustain today's rapidly-evolving field of IoT, which defines slightly different scenarios from those examined by IETF in 2009. In Section 2.3, we identify a few specific challenges that RPL must face to remain on the forefront of IoT technology. Finally, Section 2.4 contains concluding remarks.

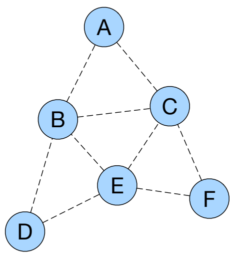
The contents of this chapter have been originally published in: **RPL, the Routing Standard for the Internet of Things... Or Is It?**, Oana Iova, Gian Pietro Picco, Timofei Istomin, Csaba Kiraly, In IEEE Communications Magazine – Feature Topic on Research to Standards – Next Generation IoT/M2M Applications, Networks and Architectures, pp. 16-22, vol. 54, no. 12, IEEE, December 2016, and have been slightly adapted for this thesis.



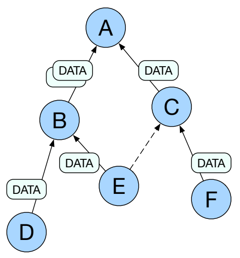
(a) IETF standardized stack.

(b) ZigbeeIP stack.

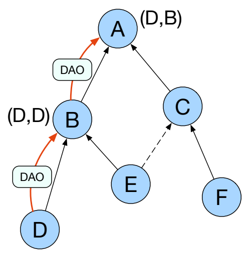
Figure 2.1 – Protocol stacks for the Internet of Things.



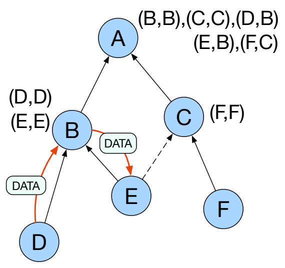
(a) A sample wireless network.



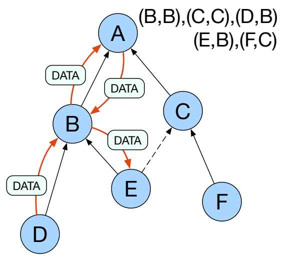
(b) Multipoint-to-point communication.



(c) Point-to-multipoint route construction: storing mode.



(d) Point-to-point communication: storing mode.



(e) Point-to-point communication: non storing mode.

Figure 2.2 – Routing in RPL. Existing routes are shown next to the network nodes.

2.1 RPL in a Nutshell

As described in the standard [105], RPL creates a routing topology in the form of a Destination-Oriented Directed Acyclic Graph (DODAG): a directed graph without cycles, oriented towards a *root* node, e.g., a border router. By default, each node maintains multiple parents towards the root; a *preferred* one is used for actually forwarding data packets upwards towards the root (Figure 2.2b), while the others are kept as backup routes. This scheme, called *multipoint-to-point communication* in RPL, naturally supports communication from the devices to the root with minimal routing state.

The topology is created and maintained via control packets called DODAG Information Objects (DIO), advertised by each node. The packet contains the routing metric (e.g., link quality, residual energy) and an objective function used by each node to select the parents among its neighbors. DIO packets are broadcast by each node according to an adaptive technique, the Trickle algorithm [62], which strikes a tradeoff between reactivity to topology changes and energy efficiency. Trickle ensures that DIOs are advertised aggressively when the network is unstable, and instead transmitted at an increasingly slow pace while the network is stable.

To support the dual traffic pattern from the root to the devices, called *point-to-multipoint communication* in RPL, the standard requires additional control messages and routing state. Specifically, each node in the network must announce itself as a possible destination to the root by sending Destination Advertisement Object (DAO) control packets. These messages are propagated "upwards" in the DODAG topology, via a parent that may coincide with the preferred parent (Figure 2.2c), therefore establishing "downwards" routes along the way.

As supporting this type of traffic could put more strain on the nodes memory due to the increased routing state, RPL defines two modes of operation: *storing* and *non-storing*. In storing mode, each node keeps a routing table containing mappings between all destinations reachable via its sub-DODAG and their respective next-hop nodes, learned while receiving DAOs. In non-storing mode, the root is the only network node maintaining routing information; the root exploits this global view for source routing, i.e., by including routing information directly into the packet itself.

These two modes of operation affect also the ability to support communication between any two nodes in the network, called *point-to-point communication* in RPL, and achieved as a combination of the techniques used for the previous two types of traffic. In storing mode, data packets travel upwards until they reach a node with routing information about their destination, i.e., a common ancestor (Figure 2.2d); from that point on, they proceed downwards, following the routes previously established by DAO packets. The same technique is used in non-storing mode; however, in this case, packets must travel upwards all the way to the root (the only node maintaining routing information) before being redirected to their destination (Figure 2.2e).

2.2 Did RPL Live Up to Expectations?

RPL was standardized based on requirements published 7 years ago[105], a relatively long time in the fast-paced world of networked computing in general and IoT in particular. This section revisits the original requirements for RPL stated by IETF, and concisely reports about the extent to which they are met by the state of the art. We frequently refer to the IETF documents that focused on the paradigmatic applications of home [11] and building automation [67], and urban [29] and industrial networks [85]. Due to space limitations, we focus on what we argue are the key dimensions, therefore omitting considerations on other relevant aspects, e.g., including zero-configuration or security.

2.2.1 Traffic Pattern

Original requirements. RPL was designed to account for different traffic patterns characteristic of the aforementioned reference applications. Multipoint-to-point communication is required by devices with sensing capabilities, which typically monitor the environment by periodically acquiring samples of physical quantities (e.g., temperature, pollution, humidity) and send them to a central unit. Applications may also need the dual point-to-multipoint pattern where communication is directed from the central unit to the rest of the network. This is often required to send queries to sensors or, when a control loop is present, to send actuation commands—e.g., switching lights or activating window blinds in a smart home [11]. This point-to-multipoint pattern may rely on multicast routing, which is actually a requirement in some reference scenarios [67, 29]. Finally, in alternative to a centralized controller gathering data and issuing commands, devices might cooperate with each other in a decentralized fashion by relying on point-to-point communication; for instance, lamps and appliances may automatically change their own state or the state of other appliances based on information gathered by sensor nodes nearby.

The routing protocol should support communication: *i)* from devices to a central unit (multipoint-to-point) *ii)* from the central unit to the rest of the network (point-to-multipoint) *iii)* directly among devices (point-to-point).

Current status. As we discussed in Section 2.1, all three types of traffic patterns above are supported by the RPL standard, although it emphasizes multipoint-to-point communication, naturally supported by the DODAG routing topology. On the contrary, point-to-point communication is inherently costly, as shown in Figures 2.2d–2.2e; if two nodes want to communicate with each other, packets must be sent upwards either to the first common ancestor (in storing mode), or all the way up to the root (in non-storing mode), from where they are re-routed downwards to their destination. This strategy creates congestion close to the root and greatly increases overhead and latency, causing problems especially in use cases involving replies, e.g., a query result or a command outcome. In RPL, this is the price to pay for reconciling different patterns in a single protocol, yielding the benefit of broader applicability at the expense of optimal performance.

These shortcomings motivated a new IETF standard protocol, *Reactive Discovery of Point-to-Point Routes in Low-Power and Lossy Networks (P2P-RPL)*, described in RFC 6697. This protocol is meant to complement RPL by allowing nodes to discover *on demand* routes to one or more nodes, via a temporary DODAG rooted at the node initiating communication. However, the efficiency of this combined solution is yet to be thoroughly studied.

As for multicast routing in point-to-multipoint communication, RPL supports it only in storing mode. Moreover, the standard only briefly describes how DAO messages should be used for group registration, without providing a full description of multicast operation. However, a complementary RFC 7731, *Multicast Protocol for Low-Power and Lossy Networks (MPL)*, has been recently standardized, defining multicast operation atop two protocols: Trickle and plain flooding. Evaluations in simulation (e.g., [20]) show that MPL is highly reliable, but can have high delay and energy consumption if Trickle parameters are not chosen carefully.

RPL supports all types of communication. However, for an efficient point-to-point communication and multicast routing, complementary RFCs must be implemented, increasing code complexity and memory footprint.

2.2.2 Mobility

Original requirements. In all the motivating scenarios analyzed by IETF it is foreseen that, although devices are currently mostly fixed, a variety of devices with reduced mobility (e.g., remote controls, vacuum cleaner robots, wearable healthcare devices) should be accommodated by RPL. Moreover, it is stated that industrial applications require the support of nodes located on vehicles or machines moving at speeds up to 35 km/h [85]. Devices should not act as routers while in motion; on the other hand, they should reestablish end-to-end communication with a static device within 5 s [67].

The routing protocol should allow devices that are moving to connect to the static routing topology.

Current status. The requirement above is fulfilled by RPL by allowing a mobile node to attach as a leaf to any node belonging to the routing topology. Communication from the mobile node to the root is in principle quite straightforward, as the former only requires a preferred parent, which may as well be the point of attachment. Point-to-multipoint communication, instead, is more complex; as the mobile node moves from a parent to another, information disseminated via DAOs may rapidly become obsolete.

Solutions for this case are not specified in the standard nor, to the best of our knowledge, realized in publicly-available reference implementations. Moreover, the under-specification in the standard is a problem also for multipoint-to-point; recent studies (e.g., [22]) show that currently implemented RPL mechanisms fail to rapidly detect when the preferred parent of a mobile node becomes unreachable as it moves, leading to high packet loss.

RPL allows a mobile node to attach/detach to/from the routing topology, but communication becomes highly unreliable.

2.2.3 Resource Heterogeneity

Original requirements. The devices targeted by RPL range from low-end battery-powered embedded devices severely constrained in memory and processing capabilities, up to high-end devices like smart watches or smartphones. This is in line with the reference scenarios; for example, home automation devices range from dumb, resource-constrained smoke alarms and temperature sensors to high-resource surveillance cameras. This diversity in hardware capabilities and application scenarios should be accounted for at the network layer. On the other hand, resource-constrained devices are envisioned to constitute the majority of network nodes; the need to limit battery replacement sets a lifetime goal of several years.

The routing protocol must take into account the heterogeneity of devices, and specifically address the memory and energy requirements of resource-constrained devices.

Current status. As mentioned in Section 2.1, RPL defines two modes—storing and non-storing—meant to address different resource capabilities of the network nodes. However, the RPL standard does *not* allow these two modes of operation to be mixed in the same network, severely undermining their practical use. Research solutions allowing nodes with different modes to operate in the same RPL network exist (e.g., [56]) but are yet to be standardized.

Even if this problem were to be solved, however, another threat to memory-constrained devices comes from the protocol footprint. RPL inherits the interoperability benefits of IPv6, but also its complexity; for instance, the need to construct and parse IPv6 packets with floating header options and using 6LoWPAN address compression increases significantly code memory needs. This is evident in Table 2.1, which compares the RPL implementations for TinyOS and Contiki, the most popular operating systems for low-power devices, with some well-known wireless sensor network protocols. The table shows that RPL requires almost twice the memory than a collection tree protocol [45]; the latter provides only multipoint-to-point communication, but it can be combined with the Trickle dissemination protocol (providing point-to-multipoint communication) by only marginally increasing the footprint, remaining significantly smaller than in RPL. On the popular TMote Sky platform, the footprint of RPL and of the variants discussed in Section 2.2.1 is very close to the 48 kB limit for code memory, leaving almost no memory to the application—despite the fact that these RPL implementations are far from supporting all the features specified in the RPL standard.

Code memory is not even the most severe problem; for example, in the Contiki implementation the 10 kB RAM of a TMote Sky limits the neighbor and routing tables to 20 and 50 entries, respectively, severely limiting scalability as discussed next.

Protocol name	Contiki	TinyOS
Collection tree	25.2	17.5
Trickle	21.2	14.2
Collection tree + Trickle	26.0	23.0
AODV	23.4	—
RPL	42.9	32.8
RPL + MPL	46.4	—
P2P-RPL	44.2	—

Table 2.1 – Code memory requirements (OS included), in kB.

RPL has too large of a footprint for resource-constrained devices, and requires all devices in a network to run the same mode of operation, limiting heterogeneity.

2.2.4 Scalability

Original requirements. Scalability is a very important characteristic of a routing protocol, bearing a direct impact on network reliability and performance. The applications studied by IETF have different scalability requirements, depending on the area where networks are deployed. For example, in home automation the routing protocol must support at least 250 devices, with larger numbers envisioned in the future [11]. Other scenarios imply even higher numbers of deployed devices; in smart cities, networks of $10^2 – 10^7$ nodes are envisioned, possibly organized into regions containing $10^2 – 10^4$ devices each [29]. While these numbers are very large, and unprecedented in the roll-outs prior to the issuing of the RPL standard, they align with deployment experiences[1] and projected trends[2].

The routing protocol should support very large networks, possibly organized in sub-networks up to thousands devices.

Current status. Recent real-world evaluations [33] show that the scalability of multipoint-to-point communication is quite good. It does, however, become questionable when unicast point-to-multipoint communication is used. In non-storing mode, the limiting factor is the size of the packet header containing the source routing information; this can include up to 8 IPv6 addresses (64 with a compressed header), but the longer the header the higher the overhead and the route repair latency. The impact on scalability, however, has yet to be evaluated; non-storing mode has been introduced only very recently in Contiki, and is currently not supported by TinyOS or industrial implementations (e.g., from Cisco[3]).

In storing mode, instead, the limiting factor is the memory available to store neighbor and routing tables. The nodes close to the root must store routing state for almost the entire

[1]For instance, e.g., http://www.smartsantander.eu

[2]An example are the recent statements by several companies about tens of billions of devices connected by the IoT in the near future.

[3]RPL configuration guide for Cisco IOS, http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/rpl/configuration/15-mt/rpl-15-mt-book.pdf.

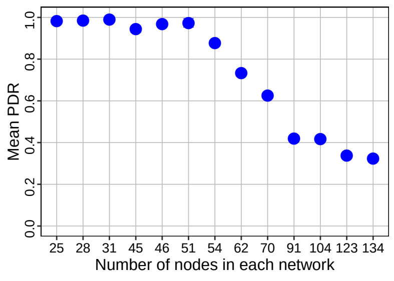


Figure 2.3 – As network size increases, the reliability of downward traffic decreases, as a consequence of more nodes having memory problems.

DODAG, which can be challenging for resource-constrained devices. Moreover, the RPL standard does not specify the action to take after a parent refuses to install a new downward route (e.g., because its routing table is full), therefore undermining the reliability of downward traffic. We recently analyzed the problem in our own work, where we focused on emulation of the Contiki RPL implementation over realistic topologies taken from a smart city deployment [52]. Figure 2.3 illustrates the problem: the reliability of downward traffic (e.g., for actuation) decreases severely as soon as the size of the network approaches the size of the routing table, which we set to 50 entries—the maximum allowed on TMote Sky, as already discussed. In practice, even for devices slightly more powerful than motes, this means that the scalability of RPL remains a far cry from the thousands of nodes mentioned above.

Until now we considered a flat address space. However, RPL can support prefix-based hierarchical routing, e.g., via the Prefix Information Option (PIO) mechanism, which allows routers to own an independent prefix and distribute it for autoconfiguration in their own sub-DODAG. However, the prefix assignment is not part of the standard, and can lead to performance deterioration [4].

RPL has serious scalability issues with point-to-multipoint traffic, especially when configured in storing mode. Prefix-based sub-netting can alleviate the problem, but requires better integration with RPL topology maintenance.

2.2.5 Reliability and Robustness

Original requirements. The quality of radio links among devices can be very unstable, especially in an urban setting, due to wireless channel effects (path loss, shadowing, fading), and interference from other communication systems. It is important for the routing protocol to react to changes in connectivity by rapidly reconfiguring the topology while maintaining a low control overhead and therefore energy expenditure. Specific metrics are stated for some of the reference scenarios considered; for instance, in home automation, the routing protocol is expected to converge within 0.5 s if no nodes have moved, and within 4 s otherwise [11].

Clearly, changes in connectivity ultimately affect communication reliability, whose critical role depends on the target application and functionality. For instance, it is expected that monitoring-only applications can tolerate a reliability as low as 90% [85], while those involving actuation require higher reliability [67]; in some application domains, message loss is even considered out of compliance and subject to fines [85], sometimes requiring a "five nines" or even "nine nines" reliability.

The routing protocol should be robust w.r.t. the inherent dynamicity of wireless links, quickly restoring communication to maintain high levels of reliability; some applications cannot afford data loss.

Current status. RPL relies on the Trickle algorithm [62] to efficiently maintain the routing topology, enabling quick reaction to connectivity changes while minimizing overhead during stable conditions. Nevertheless, when a link quality metric is used for routing, even small variations in link quality estimation can yield changes of preferred parent, as the objective function constantly searches for the best path towards the root; the consequent reaction of the Trickle algorithm generates unnecessary overhead.

The IETF addressed the problem in RFC 6719, *Minimum Rank with Hysteresis Objective Function (MRHOF)*, allowing a node to change its parent only when the new path towards the root differs significantly from the old one. The latter aspect is defined by a threshold, yielding a tradeoff between route stability and optimality; if the threshold is too high, parents are less likely to change and routes are more stable, but their quality may degrade significantly before they are reconfigured. The proper configuration of the threshold is therefore non-trivial, and not well investigated in the literature.

On the other hand, the overall reliability of RPL has been analyzed by several studies in real-world testbeds. A large 135-node experiment [33] showed values of 97% and 92% of delivered packets for the multipoint-to-point and point-to-multipoint traffic respectively, while the most challenging point-to-point traffic showed a very low reliability of 74%, due to scalability problems. For the last two traffic patterns, the authors even had to reduce by half the number of addressable nodes in the network for the routing tables to fit into the RAM, due to the aforementioned memory problems.

For multipoint-to-point traffic, RPL mechanisms deal effectively with the vagaries of wireless communication and yield good performance and reliability; the other two traffic patterns suffer from poor reliability.

2.3 RPL for IoT: The Challenges Ahead

Having examined how RPL fares w.r.t. its original requirements, we now turn our attention to current IoT trends and identify new requirements that may challenge future adoption of RPL as the routing protocol for IoT.

2.3.1 What Will Be the Dominant Traffic Pattern?

At the time of RPL standardization, multipoint-to-point communication was the main traffic pattern, motivated by the need to support distributed monitoring applications, and fueled by a decade of research on wireless sensor networks.

Today, *monitoring* is still a fundamental element of IoT architectures, but it is increasingly associated with *control* in a plethora of application domains. In its simplest incarnation, control entails the ability to send commands (e.g., for actuation) from a centralized controller via the same network used for monitoring; however, the most disruptive scenarios envision devices that interact and automatically take actions via in-network feedback loops. As a consequence, point-to-multipoint and point-to-point communication become even more critical than multipoint-to-point.

These two traffic patterns are addressed by RPL, which actually has a competitive advantage as it supports the networking requirements of both monitoring and control *i)* in an integrated fashion, and *ii)* in a large-scale setting. Unfortunately, as discussed in Section 2.2, these are exactly the traffic patterns for which RPL does *not* provide efficient solutions. Paradoxically, they are also the paradigms for which the integration with IP is key, as it enables direct addressing of each device.

Point-to-multipoint and point-to-point communication received significantly less attention in RPL, yielding implementations with poor performance; this may prevent future adoption of RPL in the ever-increasing IoT applications with a control component.

2.3.2 Mobile Devices: Norm or Exception?

Scenarios for IoT assume devices embedded in things, which can be relocated (e.g., in asset tracking), or carried and even worn by people, and therefore inherently mobile. In these ever-increasing mobile scenarios, the ability to opportunistically exploit the presence of a networking "backbone" to relay information regardless of the current location is fundamental, yet is largely overlooked by RPL, as discussed in Section 2.2.2.

Moreover, mobility often implies the need for context-aware interactions; for instance, the scope of a query or command issued by a node may need to be restricted to only a portion of the network surrounding it, based on system or application level properties (e.g., the floor building where the person carrying the device is walking).

Mobility is an unavoidable requirement in IoT scenarios, for which RPL currently provides unsatisfactory performance; further developments should also consider support for context-aware routing.

2.3.3 Resource-constrained Devices: Norm or Exception?

The devices enabling the IoT vision are inherently resource-constrained. This is unlikely to change in the future; technological developments will push the boundary of computing and communication power for the devices we know today, but at the same time generate new breeds of devices even smaller and therefore resource-constrained—a recurring trend in the last decades.

In this respect, the current demands of RPL in terms of code and data memory are still too high. In part, this is a consequence of IPv6 compliance. However, it is also the result of a standard that simply addresses (and requires) too much and, ironically, of implementations that neglect the few opportunities the standard allows to explicitly cater for resource-constrained devices—e.g., notably including non-storing mode. Specialized RPL variants must be devised, guaranteeing a better tradeoff between performance and flexibility and allowing the selection of the appropriate protocol components or variants based on the device (and application) constraints at hand.

Resource-constrained devices are here to stay. However, the current definition of RPL is often at odds with the requirements they pose, and the implementations are often ignoring the few provisions the standard already offers to ameliorate the situation.

2.3.4 New Approaches to Network Stack Design

When the RPL standard was in the making, the predominant routing approach was incarnated by the CTP protocol [45], from which RPL borrows several techniques. Around the time the standard was issued, however, two alternative routing paradigms emerged in the closely-related wireless sensor network research community: opportunistic routing and synchronous transmissions. The former approach, popularized by ORW [60], considers all neighbors as potential forwarders, therefore reducing delay and energy consumption and increasing robustness by exploiting spatial diversity. The second approach, popularized by Glossy [38], removes completely the need for a routing (and MAC) layer, by providing a reliable network flooding primitive; reliability is achieved by guaranteeing that rebroadcasting occurs within a very short time bound necessary to exploit constructive interference and the capture effect, and yielding also network-wide time synchronization.

Both approaches are reported to significantly improve over the state of the art, in terms of reliability, latency, and energy consumption. Opportunistic routing has already inspired a RPL variant [33] aimed at improving downward traffic; this is not yet the case for synchronous transmissions, whose even higher performance gains are currently obtained in networks of homogeneous devices using the same radio chip [38].

New approaches with significantly better performance have emerged since the RPL standard was defined, and should therefore receive attention.

2.3.5 New Wireless Technology

The new scenarios fostered by IoT motivated the development of new wireless technology tackling lower power consumption, increased range, or both. Some of these, e.g., Wi-Fi HaLow (IEEE 802.11ah), are a natural evolution of existing technologies, and likely to be easily accommodated into RPL. At the other extreme, Low-Power Wide-Area Networks (LPWANs) hold the potential to radically change the picture of IoT networking as currently defined by RPL. With a low-power budget similar to IEEE 802.15.4 devices, LPWANs provide long-range communication (2–5 km in urban environments, 30 km and higher otherwise) at the price of reduced bandwidth and data rates. At a minimum, LPWANs are likely to become an inescapable alternative for interconnecting different sub-networks at a geographical scale; however, they also hold the potential to induce a rethinking of multi-hop routing strategies. In between these extremes, IoT technologies hitherto applied mostly on consumer appliances are being optimized for ultra low-power consumption, as in the case of Bluetooth Low Energy (BLE), therefore becoming an appealing alternative for fulfilling goals similar to RPL.

The scenarios fostered by IoT are triggering a new wave of wireless technologies that can significantly redefine the goals, and therefore the mechanisms, of RPL.

2.4 Conclusion

In this chapter we analyzed the extent to which the RPL standard lived up to the expectations defined by IETF requirements, and highlighted other challenges that emerged since its definition as a standard. RPL had its quota of successes, clearly contributing to the advancement of communications in the world of tiny, embedded, networking devices.

The concise analysis we provided in this chapter, however, also highlighted several weaknesses that, in our opinion, may undermine a larger IoT adoption of RPL. The complexity of today's IoT makes it difficult for a single standard to paper over the significant differences in application requirements and heterogeneity in hardware constraints, both likely to be exacerbated in the near future.

In the next chapter we look at RPL at a much deeper level of detail, studying its performance in a set of IoT scenarios. We provide evidence backing up the identified weaknesses of RPL w.r.t. the downward traffic delivery, suggest techniques to overcome them and study the techniques at scale in simulation and on real hardware.

3 Route or Flood? Towards an Efficient and Reliable Downward RPL

In this chapter we demonstrate that the performance gap between data collection and actuation in RPL is significant because of the scalability problems of the downward routing identified in Chapter 2. This gap limits the overall reliability of the protocol in applications relying on bi-directional connectivity.

Various techniques have been proposed at the network layer to support downward traffic. Those can be divided into routing-based (proactive [19] or reactive [82]) and flooding-based [46], with their own pros and cons. Routing protocols are usually blamed for the extensive overhead introduced by maintaining the routing tables. Flooding, on the other hand, does not need to maintain any routing entries, but it is an energy-hungry approach with a high channel utilization.

In this chapter we show that mechanisms rooted in both approaches can help improve the downward routing in RPL, considering the specific needs of each application. More specifically, we take a deeper look at RPL [105], the only standard routing protocol for IoT at this date, we show its shortcomings when it comes to transmit information from a border router towards a node inside the network, and we propose several mechanisms to improve this communication, integrating both routing and dissemination-based elements.

3.1 Context and Contribution

The IPv6 Routing Protocol for Low-power and Lossy Networks, RPL [105], has been in the centre of attention in the last years as the standard routing protocol used to connect a plethora of applications in the flourishing Internet of Things. RPL was designed to enable efficient IPv6 routing in several application domains, notably home and building automation [11, 67], as well as industrial and urban settings [85, 29]. In these contexts, RPL is expected to reliably relay

This chapter is partially based on our published works: 1) **D-RPL: Overcoming Memory Limitations in RPL Point-to-Multipoint Routing**, Csaba Kiraly, Timofei Istomin, Oana Iova, Gian Pietro Picco In Proceedings of the 40th Annual IEEE Conference on Local Computer Networks (LCN), 2015, Clearwater Beach, Florida, USA, October 2015; 2) **Is RPL Ready for Actuation? A Comparative Evaluation in a Smart City Scenario**, Timofei Istomin, Csaba Kiraly, Gian Pietro Picco In Proceedings of the 12th European Conference on Wireless Sensor Networks (EWSN), Porto, Portugal, February 2015.

the information to and from small, autonomous devices that perform sensing and actuation, with unprecedented degrees of scalability and flexibility. However, it has been recently pointed out that much of RPL's expectations are not completely fulfilled in the current context of the Internet of Things [51].

Indeed, while RPL supports both upward communication necessary to funnel data from multiple sensors to a central data sink (e.g., a controller), and downward traffic necessary to send commands from the sink to actuation devices, its operation is optimized for the former type of traffic. Specifically, much more control packets have to be sent in the network in order to propagate the routing information needed for downward communication, which means more overhead and energy consumption.

RPL identifies two modes of operation when downward traffic is needed: *storing* and *non-storing*, based on whether routing information is stored at every node or instead is attached directly to (source-routed) packets. From an application point of view, non-storing mode appears to be better suited to home automation and building control, where the majority of nodes in the network have extremely constrained memory [12]. However, in advanced metering infrastructure and industrial networks, the use of storing mode is considered more energy efficient since it does not have to transport the hefty headers associated to source routing [86]. Also, the storing mode allows for better prediction of the latency associated with downward traffic flows [84], which is critical in most industrial networks, and very important in several smart city applications.

Both modes of operation have their own place, and the choice between them must be performed on a case-by-case basis. Still, no matter the mode of operation, RPL should offer efficient and reliable multi-hop data forwarding. In this chapter we focus on the storing mode, as it is by far the most used in both open source (e.g., TinyOS and Contiki) and industrial (e.g., Cisco[1]) implementations.

3.1.1 Motivation: Unexpected Findings from a Realistic Scenario

Our research team was sought after for collaboration by a company deploying a large-scale low-power wireless infrastructure in the city of Trento, Italy, constituted by 860+ IEEE 802.15.4 nodes mounted on lampposts. The collaboration objective stated by the company was to improve their current network stack, which was using a simple flooding algorithm with a few mechanisms to mellow the effect of collisions. Our first step was to compare their implementation against the state of the art protocols: RPL, and Trickle [47]. However, as we showed in our previous work [52] the results were somewhat unexpected: mainstream RPL implementations (ContikiRPL and TinyRPL) were outperformed in a mains powered scenario, both in terms of reliability and efficiency. Albeit duty cycling was not taken into consideration in this evaluation, our results showed that RPL has a network utilization comparable with

[1]RPL configuration guide for Cisco IOS, http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/rpl/configuration/15-mt/rpl-15-mt-book.pdf

flooding, as it cannot amortize the topology maintenance cost under the (realistic) traffic properties used in our tests.

While timely, reliable, and efficient transmission of commands to actuation devices is key in many scenarios, our findings suggest that this goal is currently not achieved by RPL. We identified the main factor that undermines the performance of RPL as the poor handling of both data and control packets when devices have scarce memory. Since in storing mode each device in the network has to keep routing information about all the other devices to which it has to relay packets, in dense or large-scale networks the resource-scarce sensor nodes quickly deplete the available entries in their neighbor and routing tables, causing packets to be dropped. This problem has been pointed out also in other contexts, e.g., when trying to run IEEE802.15.4 TSCH with RPL on TelosB nodes, the authors had to disable the support for downward routing, due to the limited memory [34]. In a sense, RPL shows its shortcomings precisely when applied to the constrained devices it was designed for. The standard actually acknowledges its incomplete specification mentioning that *"Future extensions to RPL may elaborate on refined actions/behaviors to manage this case"*. However, until now, no such extension has been provided, despite the fact that the number of constrained devices is expected to soar in the near future [37].

A possible solution for some scenarios would be adding extra memory, but this clashes with the trend towards smaller, cheaper and increasingly resource-scarce devices enabled by ever-increasing miniaturization. Switching completely to the non-storing mode is not a solution either, as the two modes of operation have different use cases. In consequence, we want to push the state of the art for the storing mode well beyond the current simplistic specifications. In our opinion, without extensions to RPL addressing the issues above, deployments would rely on implementation-specific mechanisms, lacking interoperability, possibly leading to undefined behavior, and ultimately jeopardizing the proper functioning of these solutions and undermining the credibility of RPL at large.

3.1.2 Contribution

The goal of this chapter is to explore the design tradeoffs, and implement a novel downward RPL protocol for the storing mode, significantly more reliable and efficient than the existing alternatives on highly constrained devices.

We first go into the details of RPL's underperformance, passing under the magnifying glass the results that we previously obtained [52]. We show that the core of the problem is the inability of RPL to react when devices remain without memory to store neighbor or routing information. As a result, not only control packets, but also data packets are dropped, severely impacting the reliability of the protocol.

To overcome these memory problems, we propose three new mechanisms, which incorporate different levels of routing and dissemination-like techniques:

1. SWITCH searches for alternative parents that can share the routing table load.
2. ROOT adds a 1-hop broadcast at the root to maximize the chance that the data packet reaches a node that knows its destination.
3. MCAST generalizes this 1-hop broadcast solution into a limited and directed multi-hop dissemination that uses multicast in its base.

While each of these solutions improves RPL in different ways, there is still a tradeoff to be made between reliability and energy efficiency. To obtain the best tradeoff, we combine the advantages of the proposed mechanisms into a new protocols T-RPL, which manages to have a reliability close to 100%, while keeping the radio duty cycle low, at around 1%.

Finally, we provide an extended evaluation of all the proposed solutions, along several dimensions:

- *Protocol competitors.* To offer a thorough evaluation we compare against the following protocols: RPL variants (ContikiRPL [24], TinyRPL [98], ORPL [33]), dissemination-based protocols (Trickle Multicast - TM [47], and Flooding) and a reactive Ad-Hoc On-Demand Distance Vector (AODV) routing protocol [82];
- *Simulation vs Testbed.* We evaluate the aforementioned protocols both in simulations and testbeds, as the presence of real-world factors can impact the reliability and energy efficiency of the protocols.
- *Physical topology.* While our reference scenario is the real placement of LLN nodes that we used in [52], we also experiment with synthetic, grid-like topologies that allow us to study other properties of the proposed solutions, like scalability;
- *Duty-cycling MAC.* The findings we reported in [52] assume an always-on radio, motivated by a mains-powered smart city scenario. Although this is a compelling case study, *i)* this assumption does not hold for all applications where point-to-multipoint RPL is envisioned, and *ii)* duty cycling may significantly change the tradeoffs among protocols, e.g., it is known to negatively affect dissemination protocols;
- *Different topology densities.* The staple applications we mentioned differ in their levels of topology densities, which affects differently the protocol performance (e.g., causing the aforementioned issues with neighborhood and routing tables);
- *Interplay with upward traffic.* Downward traffic (e.g., for actuation) is often generated in parallel to upward traffic (e.g., for data collection), which may heavily affect the former.

In the next section we make a detailed overview of RPL, then we present our motivating scenario (Section 3.3), and we highlight the reasons behind RPL's poor performance in point-to-multipoint communication (Section 3.4). We present our three proposed mechanisms in

Section 3.5, discussing also implementation details, their limitations, and possible combinations into stronger protocols. Section 3.7 presents an evaluation of all the proposed solutions, including their combinations against the baseline RPL and Flooding. Then, in Section 3.8, we evaluate in simulation the best of the proposed protocols (T-RPL) against a series of competitors, which shows the superiority of our solution in multiple scenarios. Then, in Section 3.10, we also evaluated the performance of T-RPL and Flooding in a testbed scenario with over 90 nodes. We present the related work in Section 3.11 and our conclusions in Section 3.12.

3.2 RPL - The Routing Standard for the Internet of Things

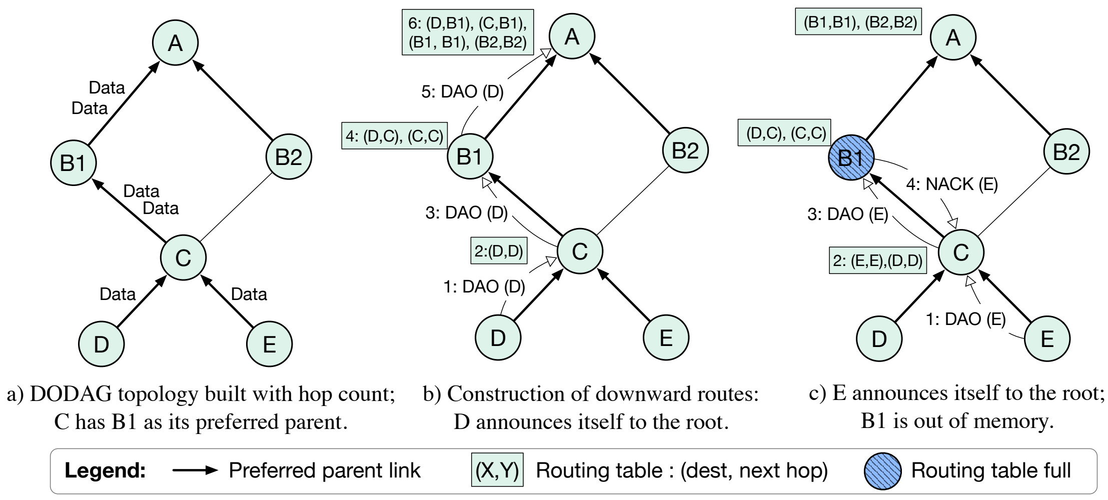
The standardized IPv6 Routing Protocol for Low-Power and Lossy Networks, RPL [105], has been designed to connect “thousands of constrained devices". The protocol builds a Destination-Oriented Directed Acyclic Graph (DODAG) logical topology with the border router as the root which is the connection point between the low-power lossy network to the Internet. In the scope of the LLN, the DODAG root is the destination for the convergecast (upward) traffic originating from the LLN and the source of downward traffic destined to the LLN nodes. The former is usually associated with sensor readings or events and the latter with configuration messages or actuation commands.

Unlike a tree topology, in a DODAG each node maintains multiple parents that are nodes closer to the root according to a routing metric. Nevertheless, at every moment in time RPL uses only a single (preferred) parent to forward the convergecast packets; the other parents have a backup purpose and can be used if the preferred one is not reachable anymore.

For example, we can see in Fig. 3.1a a DODAG constructed using minimum hop count as the routing metric. Even though C has two parents, it will only use the preferred parent $B1$ for all packets that have A as the destination.

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

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 keeping a global view on the network which is learnt from DAO packets containing the DAO parents of every node. When sending a data packet, the root computes the route and attaches it as a complete list of forwarders directly to the packet (source routing). In storing mode, however, the routing state is distributed across the network with every node



a) DODAG topology built 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 3.1 – RPL - upward and downward route construction

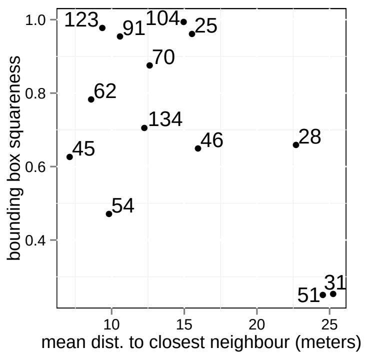
keeping a routing table that lists the next-hop forwarders for all descendants (nodes in its sub-DODAG) of the current node, learned from the received DAOs. In this mode of operation no node knows the complete path towards the destination but at every hop a decision is taken on where to forward the packet. If at any point a forwarder receives a packet from the root with unknown destination, it has to either drop it, or return it to the sender which could try another path (if any).

Figure 3.1b exemplifies how DAOs are propagated in the network and how the routing tables are populated, when the storing mode is used. D has to announce itself as a possible destination to the root. It starts by sending a DAO with its own IP address as a target, to its DAO parent, C. C checks if it has enough memory to add it to its routing table, and if that is the case, it forwards the DAO upward to $B1$. This process is repeated by every node that receives the DAO, until the message reaches the root.

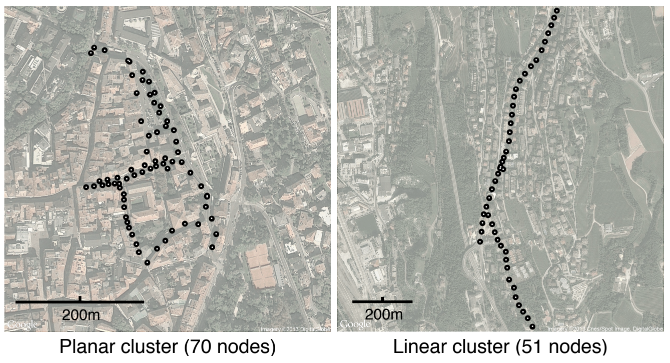
3.3 A Motivating Scenario: Smart City Lamppost Deployment

In this chapter we target the smart city scenario, arguably one of the most promising application domains for low-power wireless actuation, and one of the targeted applications by RPL [29]. Thanks to our collaboration with an IT company from Trento, Italy, we have complete knowledge of the placement of the 864 nodes on lampposts, and their grouping into 13 clusters (whose sizes varies from 25 to 134 nodes) served by independent gateways. Messages arriving from a command center through reliable links are relayed by these gateways to the nodes.

The variety of peculiar network topologies determined by the urban environment, containing semi-regular structures such as long multi-hop stripes, circles, dense and sparse areas, make this scenario different from studies in testbeds or simulations on regular grids or random topologies. Figure 3.2a shows a comparison of cluster geometries. For simplicity, we identify



(a) Topology statistics



(b) Two representative topologies

Figure 3.2 – Summary statistics about the geometry of the topology of all the 13 clusters (a) and topology of two representative clusters (b). Note the different scale of the maps.

clusters based on the number of nodes contained, and we 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 3.2b shows an example of two representative topologies: a 70-node "planar" one and a 51-node "linear" one. Note that the latter is much less dense not only because of its linear shape but also due to larger distances between the nodes.

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. We decided to use Cooja [78], which supports both Contiki [24] based implementations and others (e.g., TinyRPL) thanks to its hardware emulation feature. 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, such as transmission power, sensitivity, and antenna gain. It also models the effect of background noise and interference through signal-to-interference-and-noise ratio (SINR), the capture and multi-path effects.

On the upside, simulation also allows us to explore more parameters, which would otherwise not be possible in real life deployments. For example, using different noise levels makes it possible to study different topology densities, having as starting point the actual network deployment. The theoretical 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 a dense urban environment can be relatively high and with high short-term variations. To see which noise values to apply to our smart city scenario, we performed

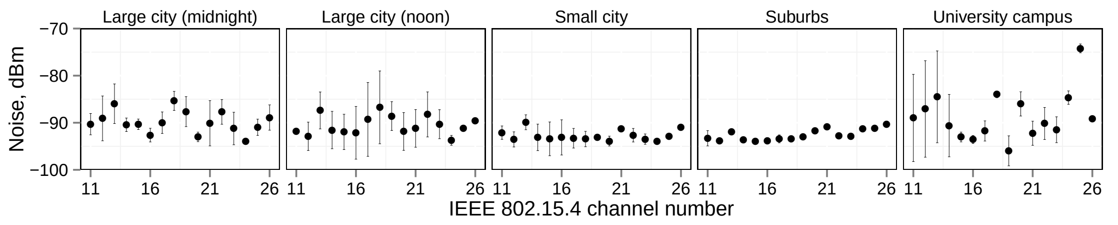


Figure 3.3 – Background noise measured in various urban environments; mean and standard deviation, dBm.

Table 3.1 – Background noise properties (mean and standard deviation) used in the simulations with the smart city topologies.

Density	Mean noise, dBm	Std.dev.
Sparse	−80	2
Intermediate	−85	2
Dense	−90	2

measurements on all IEEE 802.15.4 channels in several places of Trento and Moscow, including suburbs, densely inhabited areas and the university campus. As we can see in Fig. 3.3, the mean noise floor in the cities is usually −85 to −95 dBm, the standard deviation is typically 0–5 dBm. Occasionally these values reach −75 dBm and 10 dBm respectively.

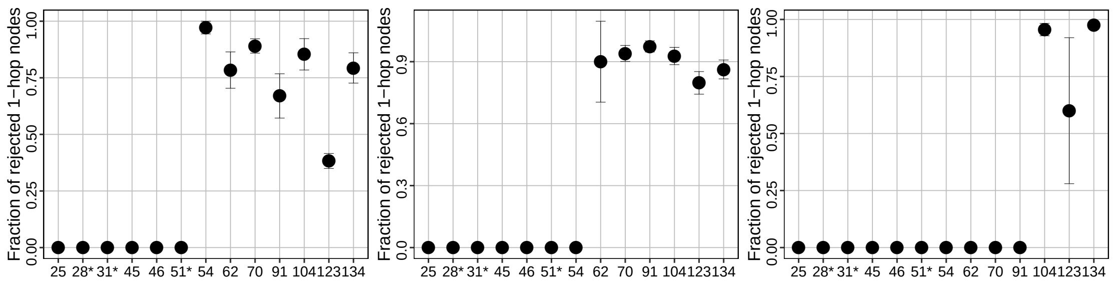
This makes it difficult for system designers to predict the properties of the deployed network such as the node neighborhood size and the number of hops, and demands from the protocols to perform well in a wide range of conditions, including dense and sparse networks.

Considering this and the noise measurements, we chose to vary the background noise parameter in our simulations, which is equivalent to scaling the density of the clusters without modifying the node coordinates. The values we used are presented in Table 3.1.

Building upon this smart city scenario, we generalize our network considering that it is formed of resource scarce devices, for example, TMote Sky having only 48K of program memory and 10K of RAM. We also assume that the root has similarly limited hardware capabilities as the rest of the nodes in the network.

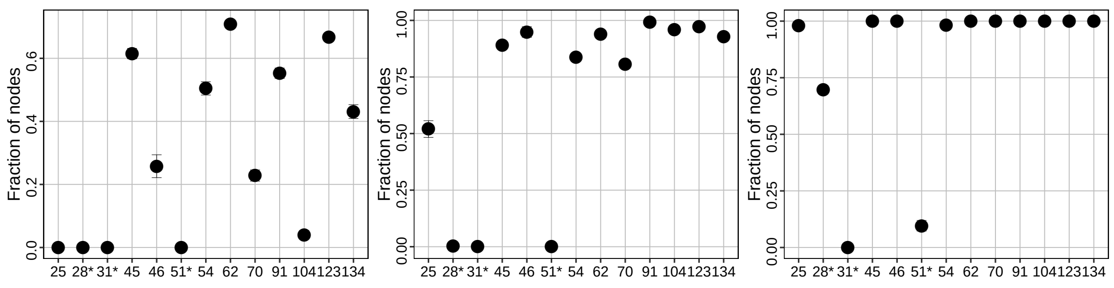
3.4 The Problem with Downward Routing in RPL

In this section we make a detailed analysis of the reasons for which RPL underperforms in the previous mentioned smart city scenario, when only point-to-multipoint traffic is used. Consequently, we present why RPL's routing and signaling mechanisms fail when operating on devices that have scarce memory.



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

Figure 3.4 – Routing table statistics on all smart cities topologies in function of different topology densities. Ratio of 1-hop nodes rejected by their parent (i.e., the DODAG root) due to full routing table.



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

Figure 3.5 – Neighbor table statistics on all smart city clusters in function of different topology densities (* marks the topologies that are linear). Ratio of nodes with the neighbor table full.

3.4.1 Too Many Routes

In large networks (e.g., the large clusters from our smart city scenario presented above) some of the nodes (especially the ones close to the root) might not be able to store all destinations situated in their sub-DODAG. When the routing table of a node becomes full, incoming DAOs from new destinations are simply dropped by current implementations. The RPL standard specifies an optional DAO-NACK message with a *Rejection* status code to notify the DAO sender that the recipient is unwilling to act as a DAO forwarder (e.g., because it has no more space in its routing table). Still, the standard does not define any mechanism to handle this rejection. In fact, in popular Contiki and TinyOS implementations of RPL, this DAO-NACK is not even sent. As a result, the DAO is ignored, and the path remains partially built, but useless, since the destination remains unknown to all nodes higher in the DODAG, including the root. Therefore, if there is an incoming packet for this unknown destination, the root has no choice than to drop it.

Let us take the example from Fig. 3.1c and suppose that the routing table of all nodes is limited to 2 entries. Node *E* wants to announce itself as a destination to the root, as *D* previously did, by sending a DAO to its preferred parent. Once C receives the DAO, it checks if it can add E to its routing table. Since it has enough memory, it accepts it, and forwards the message to its preferred parent. $B1$ follows the same reasoning. However, both of its routing table entries are occupied (by D and C), and it cannot store this new destination, so it drops the DAO. $B1$ should also send a DAO-NACK message to C announcing it that it cannot act as a DAO forwarder for destination E. Unfortunately, upon the reception of the DAO-NACK message, RPL does not specify any mechanism for E to respond accordingly. Consequently, the root has no way of finding out where E is in the DODAG, and will have to drop all the packets having this destination.

We need to understand whether this routing table overflow happens mostly at the root or also at other devices in the network. For this, we run ContikiRPL[2] over all the network clusters from our smart city scenario, with different topology densities. Figure 3.4 shows the ratio of nodes situated at one hop from the root whose parent (i.e., the root) is out of memory. In the sparse and intermediate scenarios we can notice that from the topologies with more than 50 nodes (i.e., more nodes than entries in the routing table), about 80% of them are situated at one hop from the root, showing that the routing table problem affects mostly the root.

As network density increases, it appears that the routing table problem disappears. In the dense scenario only 3 topologies still show nodes with full routing tables. Actually, the routing problem is overshadowed by another issue: too many neighbors. Indeed, at such density, a large number of nodes are connected directly to the root. This means that the root's neighbor table gets filled quicker than the routing table, causing existing routes to become unusable or preventing new routes from being built. We present more details about this problem next.

3.4.2 Too Many Neighbors

Some implementations of RPL (e.g., in Contiki) rely on the neighbor table to map IP addresses of the neighbors to their MAC addresses. If entries for some of the neighbors are not present in the table, no forwarding can be done to these nodes. If we look closer at what happens when a node receives a DAO for a new destination in its sub-DODAG we see that before checking the routing table and accepting the message, the node has to check first if the neighbor that forwarded this DAO already exists in its neighbor table. If such entry is not found and cannot be added (as the neighbor table is full), the DAO packet is immediately dropped, without further processing. In fact, a DAO-NACK cannot even be sent in this case, as instead of using the source MAC address of the DAO packet to send the reply, the node looks for it in the neighbor table, which is useless in this situation.

This problem can often be seen in dense environments, where a node has more neighbors than memory to store them, as we saw in Fig. 3.4. If only collection traffic is used, this is not a problem, since a node forwards all the received data packets to its preferred parent and the

[2]We used a configuration with 20 neighbor table entries (default for our platform) and 50 routing table entries (the maximum that fits into node's memory).

implementation guarantees that the entry for the preferred parent always stays in the neighbor table. However, for the point-to-multipoint traffic, downward routes have to be constructed beforehand, using DAO messages.

To show just how important this problem is, we resorted again to the simulation of ContikiRPL over our smart city networks. Figure 3.5 shows the results. As expected, the denser the networks are, the more nodes suffer from the overflow of their neighbor tables. Even in the sparse scenario, 2/3 of the topologies display neighbor table problems. This is no surprise as in ContikiRPL for TMote Sky platform, by default, a node can register maximum 20 neighbors and many topologies are denser than this. In the dense scenario, where the reception range increases, all the planar topologies present almost 100% of nodes with memory problems, illustrating the importance of this problem.

3.4.3 Summary

While these routing and neighbor table problems might be seen as technical details of the RPL standard and its implementations, from a routing perspective it means that one of its main mechanisms, the route signaling, is compromised: routers fail at constructing and maintaining the proper downward paths, as DAO messages get dropped, without any warning. As a result, the routing mechanisms cannot function properly, and some of the network nodes remain isolated from the root due to the lack of routing information. In the next section we present several mechanisms to solve this problem.

3.5 Rethinking Downward Routing for RPL

The memory problem appears on the path from the leaves of the routing topology to the root, when the propagation of DAO messages upwards in the DODAG is suddenly stopped. We distinguish two points where the protocol can react and try to re-establish the broken connectivity:

- when the problem is detected by network nodes due to a rejection by their DAO parent;
- when the packet with an unknown destination enters the DODAG at the root.

Consequently, different solutions can be designed: a *reaction* mechanism to handle the possible rejection of a node from its DAO parent, after the reception of a DAO-NACK packet, or a mechanism that can forward packets with unknown destinations. We propose next three solutions:

1. A reaction mechanism: SWITCH, a parent switching solution that allows rejected nodes to establish an alternative downward path when they receive a DAO-NACK from their parent.

2. A forwarding mechanism: ROOT, a 1-hop broadcast mechanism used to send data packets with unknown destinations to all the neighbors of the root, increasing thus the chances for the data packet to reach a node that has a route to its destination.
3. A hybrid mechanism: MCAST, a multicast approach where rejected nodes join a special multicast group, whose address is then used by the root to send downward traffic for unknown destinations.

The rest of the section presents in detail these three mechanisms improving downward routing in RPL.

3.5.1 Parent Switching

We recall here that even though the RPL standard allows nodes to use multiple parents for routing, existing implementations only use one parent (the preferred one) both for forwarding data packets, and for route signaling (i.e., DAO message transmission). We find this approach non-optimal and propose SWITCH, a reaction mechanism that takes advantage of the directed acyclic graph topology by using other parents than the preferred one to create alternative downward routes, when necessary.

Mechanism description. When a node receives a DAO-NACK message from its parent, it creates an alternative routing path through a secondary parent. Hence, the node sends DAO messages to other parents (one by one), until it finds one that still has neighbor / routing entries available. This approach not only takes advantage of the already existing topology, but it is also standard compliant.

Example. In Fig. 3.1c node C forwarded a DAO message to $B1$ containing destination E, but since $B1$'s routing table was full, the packet was dropped. As we can see in Fig. 3.6a, once C receives a DAO-NACK from $B1$, it tries another parent by sending the DAO to $B2$, which still has available memory. $B2$ accepts to be the forwarder for destination E, indicating this to C with a positive acknowledgement and forwarding E's DAO to the root. Starting from this moment, C keeps forwarding DAOs for destination D to $B1$ and for destination E to $B2$ until the conditions in the network change.

Limitations. This simple standard-compliant solution has the following limitations:

- Nodes close to the root can quickly fill their routing table, and even if a node tries to use other parents, these might also have their neighbor / routing table full;
- The root is the only node in the network that does not have any parents. When its neighbor or routing table gets full, none of the above methods can be applied;
- The routing cost through the secondary parents and, possibly, link quality, might be worse than that of the preferred parent, which will decrease reliability and increase energy consumption due to additional packet transmissions;

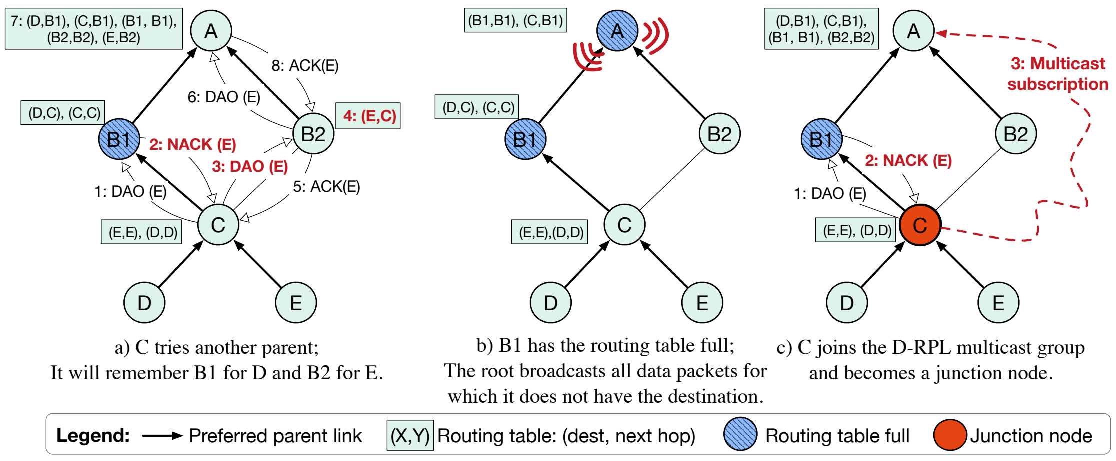

7: (D,B1), (C,B1), (B1, B1), (B2,B2), (E,B2)
A
8: ACK(E)
6: DAO (E)
(D,C), (C,C)
B1
2: NACK (E)
B2
4: (E,C)
3: DAO (E)
1: DAO (E)
5: ACK(E)
(E,E), (D,D)
C
D
E
a) C tries another parent;
It will remember B1 for D and B2 for E.
(B1,B1), (C,B1)
(D,C), (C,C)
(E,E),(D,D)
b) B1 has the routing table full;
The root broadcasts all data packets for which it does not have the destination.
(D,B1), (C,B1), (B1, B1), (B2,B2)
3: Multicast subscription
c) C joins the D-RPL multicast group and becomes a junction node.
Legend:
Preferred parent link
(X,Y) Routing table: (dest, next hop)
Routing table full
Junction node

Figure 3.6 – The three mechanisms proposed for downward traffic: a) SWITCH, b) ROOT, and c) MCAST.

- In sparse networks, a node might have no other parents that it could use.

3.5.2 Broadcasting at the root

As we saw in Fig. 3.4, the root is the memory bottleneck in most cases, as it has to keep in its routing table an entry for all the nodes in the network. However, due to the hierarchy in the DODAG topology, there are chances that nodes in its 1-hop neighborhood have some or all routes missing at the root. In consequence, we propose a modification to the forwarding mechanism at the root that sends data packets with unknown destinations using link-local broadcast to all the neighbors of the root, increasing thus the chances of the data packet to reach a node that knows how to forward it down the DODAG.

Mechanism description. In a nutshell, when the root has a packet for which there is no entry in the routing table, it broadcasts this packet to all its neighbors. Since the root has this fallback broadcast mechanism to reach its 1-hop neighbors, it does not strictly need a neighbor and/or routing table entry to forward a packet. Therefore it never rejects incoming DAOs, *creating an illusion that its neighbor and routing tables are infinite.*

Example. This solution is exemplified in Fig. 3.6b, where all the nodes have the routing table limited to 2 entries. If the destination of data packets is situated at one hop from the root, the ROOT mechanism works very well: *B*2 receives the packets even though it does not exist in *A*'s routing table. However, since neither *B*1 nor *B*2 have a routing entry for *E*, all the packets having this destination are be dropped (not by the root, but by its 1-hop neighbors).

Limitations. The main disadvantage of this mechanism is that it only covers the neighbor and routing table problem at the root and does not help any other node in the network that has depleted their memory. Furthermore, IEEE 802.15.4 layer-two broadcasts are not acknowledged, which results in a higher probability of packet loss if compared to unicast propagation. Still, as we show in Section 3.7, this simple broadcast enhancement can have very good impact on delivering data packets with unknown destination over the first hop.

3.5.3 Multicast for rejected nodes

In essence, our proposed hybrid mechanism, MCAST [54], enhances RPL's storing mode by modifying the root's behavior, similar to the 1-hop broadcast mechanism, and by generalizing it to the rest of the nodes in the network that have to handle DAO-NACK messages. In a nutshell, all the nodes that failed to advertise a DAO (for themselves or for someone in their sub-DODAG) and received a DAO-NACK from their parent, join a special multicast group, whose address is then used by the root to send data with unknown destination. The normal RPL unicast operation is resumed as soon as the packet reaches a group member knowing the route to the destination.

Mechanism description. For each packet, the root first checks its routing table for normal IP forwarding. If there is no route entry for the destination address, instead of dropping the packet or broadcasting it, the root forwards it to a special multicast group for rejected nodes. The root first changes the packet's destination address to the multicast group address, and adds a special MCAST IPv6 extension header that contains the original destination address[3]. Unicast forwarding is resumed at the moment the packet reaches a node that knows the packet's original destination. We call this node a *junction node*, as it is situated at the crossroad of two delivery mechanisms: multicast dissemination and unicast forwarding.

By joining the multicast group for rejected nodes, and therefore becoming a junction node, the node enables the delivery of packets from the root to all destinations in its routing table, even those for which it received a DAO-NACK. If a node receives several DAO-NACKs for different destinations, it only has to join the group once. To avoid introducing duplicates in the network, a junction node should keep track of destinations for which it received DAO-NACK messages and switch to unicast forwarding only for these destinations.

To reduce traffic overhead in space and time to the necessary minimum, nodes that have no more marked route entries can leave the multicast group.

Example. We demonstrate this in Fig. 3.6c. MCAST goes into action when node C receives the DAO-NACK from $B1$, becoming the last node on the path knowing the route to E. As such, C has the critical role of ensuring E's reachability from the root, and hence subscribes to the multicast group of rejected nodes. When the root wants to send a packet to E, or to any other node for which it does not have a route, it simply sends it to all the nodes subscribed to this multicast group.

Now, let us assume that in Fig.3.6c node E has another parent C', which is a junction node for other destinations, but not for E (i.e., C' did not receive a DAO-NACK for destination E). Both

[3]The extension header is inserted after the RPL hop-by-hop option header. Other possibilities to achieve the same functionality could be the use of an IPv6 destination option header or to use generic IPv6 tunneling.

Table 3.2 – Characteristics of the proposed mechanisms and protocols.

	Parent switching	1-hop broadcasting at the root	Multicasting rejected nodes
SWITCH	X	—	—
ROOT	—	X	—
MCAST	—	—	X
SWITCH+ROOT	X	X	—
MCAST+SWITCH	X	—	X
MCAST+ROOT	—	X	X
T-RPL	X	X	X

C and C' receive the packets sent by the root via the multicast channel, but only C forwards them to E, as C is its junction node. This way we avoid that E receives all the packets in duplicate.

Limitations. The main drawback of MCAST is exactly the multicast mechanism that improves the point-to-multipoint communication of RPL. Its dissemination nature makes it send data packets to all the junction nodes, while, probably, only one of them has the route to the destination. As a consequence, it can increase the overhead in the network and the energy consumption of the nodes.

3.5.4 Mechanism combinations for downward RPL

Looking carefully at the proposed mechanisms for improving downward RPL, we notice that none of them are mutually exclusive. Consequently, by taking advantage of their strengths we can combine these mechanisms into new protocols, obtaining new solutions for downward RPL. As we can see in Table 3.2, this approach can lead to 4 new solutions that we are going to presents next into more details.

SWITCH+ROOT. A first natural combination consists in bringing together the forwarding mechanism encompassed by ROOT, and the reaction meachanism of SWITCH, as they are complementary to each other. ROOT has from the start the disadvantage that it only solves the memory problem at the root, while SWITCH suffers mostly at the root. With this new approach, all the nodes in the network (besides the root) can search for alternative DAO parents upon the reception of a DAO-NACK message, while the root falls back to the 1-hop broadcasting when it does not find any route entry for a given destination. Basically, the nodes in the network are divided into two types: nodes that follow the 1-hop broadcast mechanism (i.e., the root), and nodes that implement the parent switching mechanism (i.e., all the other nodes in the network).

MCAST+SWITCH. The MCAST and SWITCH mechanisms are not mutually exclusive either. Moreover, finding alternative downward routes can lead to a reduction of junction nodes, which in return can reduce the amount of multicast traffic. When a node receives a DAO-NACK

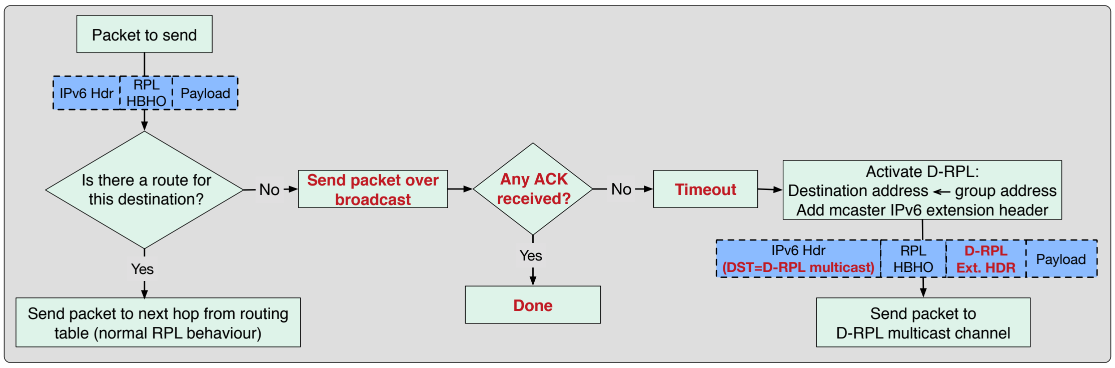

Packet to send
IPv6 Hdr
RPL HBHO
Payload
Is there a route for this destination?
No
Yes
Send packet to next hop from routing table (normal RPL behaviour)
Send packet over broadcast
Any ACK received?
No
Yes
Done
Timeout
Activate D-RPL:
Destination address ← group address
Add mcaster IPv6 extension header
IPv6 Hdr (DST=D-RPL multicast)
RPL HBHO
D-RPL Ext. HDR
Payload
Send packet to D-RPL multicast channel

Figure 3.7 – Flow chart of a RPL root that implements both ROOT and MCAST mechanisms (marked with red) as in the MCAST+ROOT and T-RPL solutions.

from its preferred parent, besides joining the MCAST multicast group, it can also start searching for alternative DAO parents in the background, as in SWITCH. When such a parent is found, the junction node can leave the multicast group because it is able to use the normal unicast forwarding for all nodes in its sub-DODAG. This approach should reduce the number of subscribers to the multicast channel, and hence, the number of packets sent using multicast.

MCAST+ROOT. A more significant reduction of the number of multicast packets can be obtained if we eliminate the junction nodes closest to the root. We recall here that a node situated at 1-hop from the root is a junction node if it has received a DAO-NACK from its parent (i.e., the root). By adding the ROOT mechanism to MCAST, we should eliminate all junction nodes situated at 1-hop from the root, as in this case the root never rejects incoming DAO packets.

More specifically, as we can see in Fig. 3.7, when the root has a data packet to send, it first checks if there is any routing entry for the given destination. If it is the case, it forwards the packet to the corresponding next hop. Otherwise, the data packet is transmitted as a link-local broadcast. If any of the root's neighbors has a routing table entry for that destination, forwards it to the next hop, resuming the standard unicast forwarding, but also sends a layer three acknowledgement to the root. If after a timeout the root did not receive an acknowledgement (i.e., the junction node is not a 1-hop neighbor), it will send the data packet over the multicast channel, using MCAST. All the other nodes in the network besides the root follow the MCAST mechanism.

The layer three acknowledgement that we introduced here is a newly defined ICMPv6 message that informs the root that the packet has been forwarded successfully and no further actions are required.

Switch+Root+Mcast, or T-RPL. Finally, we combine all of the proposed mechanisms (parent switching, 1-hop broadcasting at the root, and multicast for the rejected nodes) into one single, powerful, protocol: T-RPL. In a nutshell, T-RPL functions at the root exactly like MCAST+ROOT

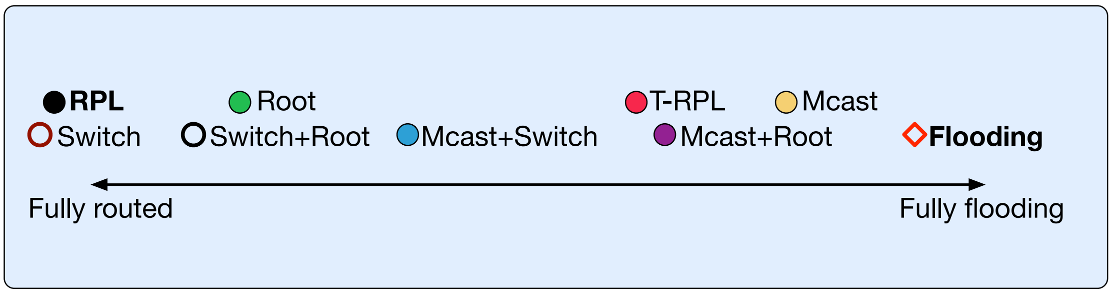


Figure 3.8 – Proposed mechanisms and solutions characteristics w.r.t. our baseline ContikiRPL and Flooding (with bold).

(Fig. 3.7), while all the other nodes in the network use the MCAST+SWITCH reasoning: upon rejection from its preferred parent, a node will both join the MCAST multicast group, and search for an alternative DAO parent in the background. This combination assures us that packets are delivered downstream using RPL's mechanism when there are no problems, and using the most efficient of our solutions, when nodes encounter the memory problem.

3.5.5 Summary

In this section we presented three mechanisms that can help improve downward routing in RPL: SWITCH, ROOT, and MCAST. They each have their strengths, as each of them has a different approach in solving the memory problem, ranging from a fully routed mechanism (parent switching in SWITCH) to different levels of data flooding (a broadcast reduced to only the 1-hop neighbors of the root in ROOT, and a limited dissemination using the multicast tree in MCAST). Furthermore, as we saw in Section 3.5.4, some of these mechanisms are complementary to each other, so a natural approach was to combine them, obtaining stronger solutions. Figure 3.8 shows where these new protocols fit w.r.t. our baseline, RPL, but also w.r.t. a full flooding approach, as each of them is characterized by different levels of routing and dissemination, based on the mechanisms that they encompass. We present in the next section a detailed evaluation of all the proposed mechanisms and protocols, comparing them with RPL and Flooding.

3.6 Implementation Details

We implemented all our proposed mechanisms and solutions in Contiki OS[4], starting from adding support for the negative DAO acknowledgements (DAO-NACKs) we mentioned before. Even though negative DAO-ACKs are defined by the RPL standard and carry a rejection code in the status field, no specific rejection codes have been allocated, leaving 128 such codes open for protocol implementations. In consequence, we define here a new status, *Out of memory*, which will be sent in response to a DAO, when the receiver has reached its neighbor and/or

[4]A snapshot of the official Contiki repository on github.com from 22 January 2015

routing table capacity.

Moreover, in Contiki OS, to be able to send these DAO-NACK messages, a node needs to have the destination in its neighbor table. However, since the memory may be full when receiving the DAO, its destination can not be added to the neighbor table, and hence, the DAO-ACK cannot be sent. To solve this problem, from the total number of neighbor table entries, we reserve N [5] of them to be used for sending DAO acknowledgements. This means that a node temporarily adds a new neighbor, which it deletes after sending the corresponding acknowledgement.

Next, we will present the implementation details for each mechanism separately.

SWITCH. Taking advantage of the DAO-NACK mechanism described above, the implementation of the SWITCH mechanism adds the following modifications to ContikiRPL:

- The parents table carries a new flag that tells whether a parent has already answered with a DAO-NACK, to avoid inquiring it again;
- The routing table has a new field to memorize the parent to which DAOs advertising each entry should be sent.

ROOT. In this case, we only need to modify the behavior of ContikiRPL at the root, all the other nodes in the network maintain the same implementation. As we mentioned before, first of all, the root accepts all incoming DAOs even if it has no space left in its tables. Later, when the root has to forward a packet to an unknown destination, it sends it as a layer two broadcast keeping the original layer three destination. Upon receiving a broadcast from the root, its neighbors check whether they know the route to the destination and use it to forward the packet in the standard unicast way. Otherwise they just drop the packet.

MCAST. In principle, MCAST could use any multicast protocol like MPL (Multicast Protocol for Low power and Lossy Networks) [48], or SMRF (Stateless Multicast RPL Forwarding) [77] to deliver the packets with unknown destination, however a combination with MPL would not make sense because this protocol floods the whole network with multicast packets so there is no need of an additional mechanism like MCAST to deliver them. SMRF, however, limits the scope of dissemination to the multicast tree and MCAST resumes the unicast forwarding between the junction node and the destination, reducing the overall amount of transmissions in the network.

There are other implementations of multicast MCAST could benefit from, for example BMRF [42], however, we chose to implement MCAST on top of SMRF, because it is already a part of the Contiki OS codebase. In SMRF, the nodes keep no per-packet state and only very little structure-related state. Multicast DAO messages are initiated by nodes joining the group and are propagated up the DODAG. Nodes receiving these messages, even if they did not join the group, store the multicast address in their routing table and forward the DAO message

[5]After several experiments, we fixed the value of N to 4 in our simulations, as this value offers the best reliability.

upwards. Thus, nodes that have exhausted their memory can still use SMRF, given that only a single multicast routing table entry is reserved for the MCAST multicast address.

The code specific to MCAST includes a hook in the IPv6 packet forwarding logic to divert packets on the multicast channel and to add the MCAST extension header; the handling of the MCAST extension header in the incoming packet processing code; an extension of routing table entries with DAO-ACK/NACK status field; and the corresponding logic in the RPL DAO-ACK handling code.

Flooding. As a second baseline in addition to ContikiRPL we use a combination of the standard RPL data collection and a simple whole-network flooding protocol to deliver actuation commands.

In this case, the nodes do not keep routing entries for downward traffic at all, they only maintain the default route used for data collection through their preferred parent; the neighbor table size is kept the same as for the other protocols. When the root has a packet to send, it broadcasts it to all its neighbors. When a node receives a packet that it did not see before, it rebroadcasts the packet only once after a short random delay. There are no acknowledgements and retries at the MAC layer, and duplicates are filtered by using a cache that stores message identifiers; a time-to-live (TTL) is associated to messages to limit their rebroadcasting. We use the memory space previously reserved for routing table to keep a history of the last 100 seen packets.

3.7 Evaluation of Downward Protocols

This section presents a detailed evaluation of all the proposed mechanisms and their combinations as presented in Section 3.5, and summarized in Table 3.2. Our baseline for comparison is ContikiRPL and Flooding. After testing these solutions on all the network clusters from our smart city scenario, we also evaluate their scalability by using unbiased 2D regular grid topologies where we increase the network size up to 225 nodes. We start by presenting our simulation setup.

3.7.1 Simulation Setup

We used Cooja [78] network simulator to run the real binaries of the protocols compiled for the TMote Sky platform, for which Cooja provides instruction-level emulation. The radio propagation was modelled by the Multi-Path Ray Tracer built into Cooja (MRM). For statistical relevance, we ran simulations 20 times per set of parameters.

In our evaluation we used the smart city scenario presented in Section 3.3: a deployment of 864 nodes on lampposts, grouped into 13 clusters (whose sizes varies from 25 to 134 nodes). In simulation, we used different noise levels to obtain different topology densities. The characteristics of some representative clusters are shown in Table 3.3.

Table 3.3 – Topology metrics for several planar smart city topologies in comparison with synthetic grids of 121 and 255 nodes.

	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 out PDR	11	16	11	24	37	28	46	69	63	27	29
Max sum out PDR	17	26	21	36	55	44	59	83	87	36	38
Min sum out 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

Moreover, to eliminate the bias due to the structure of the smart city clusters and to go beyond their scale, we also experimented with 2D regular grids. The nodes were situated on a $N \times N$ grid, where $N \in \{3,5,7,...,15\}$, obtaining networks that have up to 225 nodes, with the root in the center. We selected the grid step and the radio propagation properties so that the main topology metrics presented in Table 3.3 for the synthetic topology of 121 and 225 nodes are similar to the ones of the largest smart-city cluster with 134 nodes at the intermediate density.

In simulations we could go beyond networks that have 225 nodes to test the scalability of our solutions. However, since the goal of our study is to improve the performance of the downward traffic in RPL without hampering the data collection aspect of the protocol, we chose to scale the network size as long as the collection performance of RPL is still solid. To do so, we simulated ContikiRPL with collection only traffic, using an inter-packet interval of 3 minutes, and with DAO routing mechanism enabled, but no downward traffic.

As we can see in Fig. 3.9, the packet delivery ratio (PDR) of data collection starts decreasing once the network size reaches 169 nodes. This performance degradation was traced down to two problems. First, the larger number of nodes generate more packets that overload the network. Second, the topology maintenance mechanism of ContikiRPL causes frequent DODAG reconfigurations and thus generates an unbearable amount of DAO messages, showing that above this size the network topology is unstable and it would not make sense studying the downward routing at such scale.

Protocol setup. Table 3.4 presents the most important protocol settings used in this study. For ContikiRPL and all of the proposed solutions we kept the default Minimum Rank with Hysteresis Objective Function (MRHOF) [43] and the Expected Transmission Count (ETX) routing metric. For the neighbor table size we kept to the default setting of 20 entries and set the routing table size to 50, which occupied almost all remaining RAM on the TMote Sky platform.

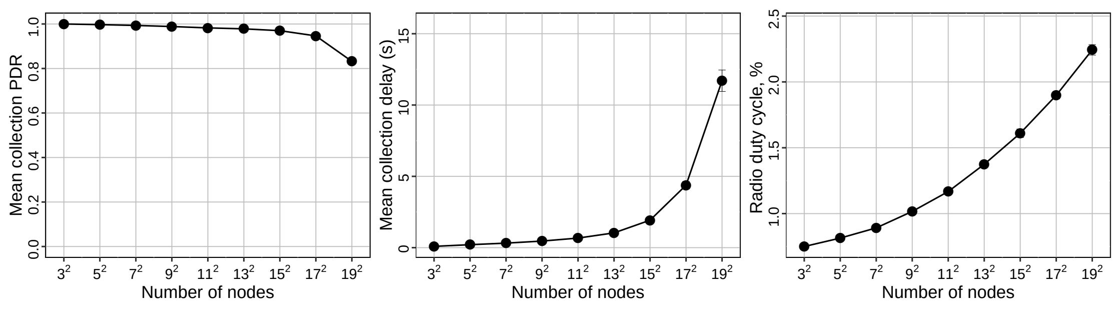


Figure 3.9 – Performance Evaluation of ContikiRPL on synthetic topologies in the presence of collection traffic with an inter-message interval of 3 minutes.

Table 3.4 – Protocol setup for ContikiRPL and all the proposed solutions.

Neighbor table size	20
Routing table size	50
Routing metric	ETX
Objective function	MRHOF

At the MAC layer we used ContikiMAC in all the simulations, with a wake up interval of $125ms$, its default value.

Application setup. We test the performance of ContikiRPL and our proposed solutions with an actuation traffic: sending commands from the gateway to other nodes in the cluster. Messages have a 6 B payload, enough to fit a command code and 1–2 parameters. In each experiment, after a warm-up time needed to stabilize the logical topology (10 minutes in our simulations), the gateway sends $B = 2000$ isolated commands, each destined to a node chosen with uniform random selection, with an inter-message interval (IMI) of $10s$; we focus on the reliability and timeliness of delivering isolated commands, rather than scalability in terms of traffic load, which we analyze later.

Performance metrics. The plots report the average value along with error bars denoting the confidence interval. Reliability and timeliness are quantified by measuring the *packet delivery ratio (PDR)* – computed as the ratio of valid received packets over the number of transmitted packets, and average *delivery delay* – time between the packet generation at the source and its reception at the destination, for each destination and further averaging over all nodes of the cluster. For the energy efficiency, we measured the *radio duty cycle* – ratio of time spent by the nodes with their radio on, either transmitting or listening, as reported by Cooja.

3.7.2 Evaluation on smart city topologies

We start our evaluation by taking a look at how our proposed mechanisms perform individually, when compared to our baseline, ContikiRPL and Flooding, over all the 13 clusters from our smart city scenario, using 3 different topology densities. Then, we evaluate all of the proposed combinations in the same scenario, to find the best tradeoff between reliability and network lifetime.

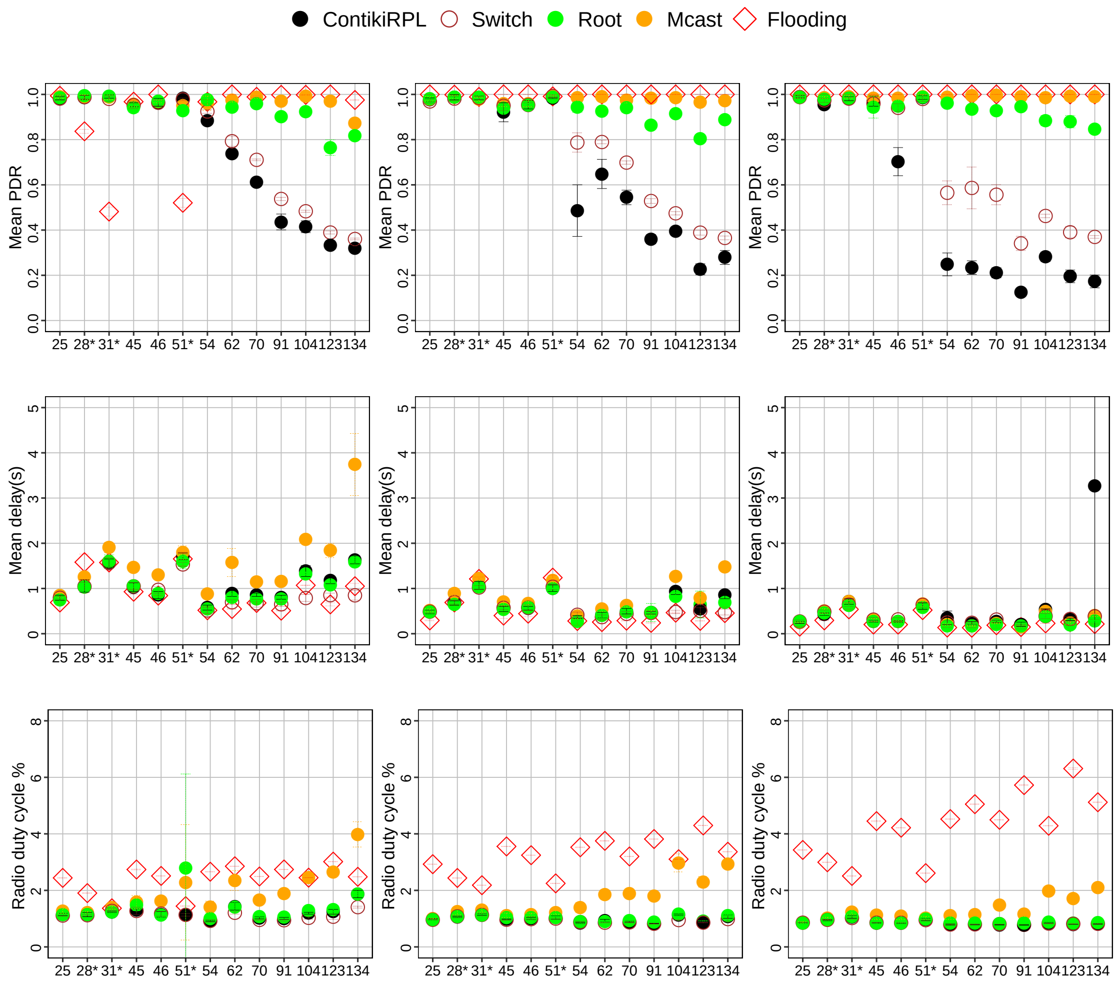
Evaluation of the mechanisms. Figure 3.10 shows the performance of our three mechanisms (SWITCH, ROOT, and MCAST) under different topology densities. The performance of ContikiRPL starts degrading once the network size gets over 50 nodes, due to the limited space in the routing and neighbor tables (a node can store maximum 20 neighbors and 50 routes, as we mentioned in Section 3.4). The PDR barely reaches 20% in some cases, when networks become dense and neighbor tables get saturated.

SWITCH manages to improve the reliability of RPL in all topologies, while keeping the same low duty cycle as ContikiRPL. Still, this improvement is not that 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 the alternative parents. If we look at the PDR figures from right to left we notice that as density decreases, the gap between SWITCH and ContikiRPL becomes smaller, as the diversity of paths also decreases, leaving the nodes with almost no alternative DAO parents.

For our second solution, ROOT, surprisingly, just by adding the simple 1-hop broadcast enhancement to ContikiRPL, the PDR gets above 80%, while keeping similar duty cycle and delay as ContikiRPL. Indeed, the duty cycle is always around 1%, showing the efficiency of the ROOT mechanism. Still, ROOT does not reach 100% of reliability on any of the topologies, as we can see also in Tables 3.5, 3.6, and 3.7. The reason is that the link-layer broadcast is less reliable than unicast and moreover, the 1-hop broadcast only applies to the root; all the other nodes in the network that have memory problems continue to drop packets if they don't know where to forward them. These results show that, as we 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 manage to significantly improve the downward packet delivery ratio of RPL.

MCAST on the other hand, is the only mechanism that manages to reach a reliability close to 100% of where ContikiRPL barely has 20%. As we can see in Tables 3.5, 3.6, and 3.7, MCAST's reliability is above 87% in sparse scenarios, above 95% in intermediate scenarios, and above 98% in dense scenarios. This improvement comes with slightly increased duty cycle and delay, partially because MCAST delivers up to 4 times more data packets than ContikiRPL, but also partially due to reliance on multicast: the same data packet is forwarded in several directions, reaching all the junction nodes. Contrary, in ContikiRPL and in the other two solutions, packets are forwarded only by nodes that have a route for that destination.

Finally, Flooding is the only protocol delivering 100% of packets in dense scenarios, and above



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

Figure 3.10 – Performance evaluation of the proposed mechanisms on all 13 clusters from our real-world smart city deployment, under different topology densities. The value on the $x-axis$ represents the size of the cluster (* marks the topologies that are linear).

98% in intermediate scenarios. In sparse scenarios, however, the reliability drops to 48% for the linear topologies, as we can see also in Table 3.5. However, the high reliability comes with an increase in duty cycle, which in dense scenarios is up to 6 times more than ContikiRPL. Indeed, even though Flooding does not use DAO messages for downward route construction, the overhead induced by duplicate packets is very high. In sparse networks however, the duty cycle is comparable to MCAST, as when the links are weak, the redundancy of flooding pays off.

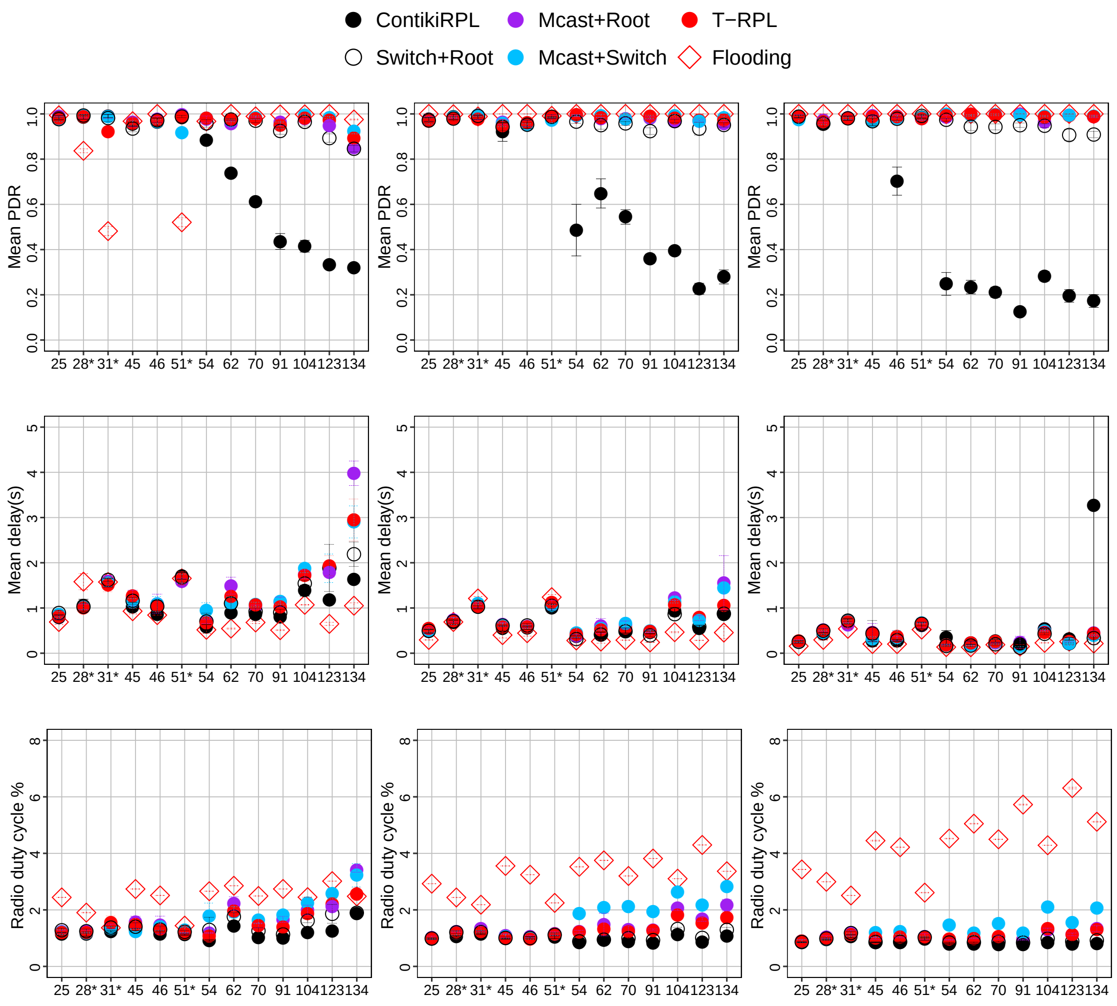
Evaluation of the combinations. We evaluate now all of the proposed combinations in the same scenario, and against the same baseline: ContikiRPL and Flooding, to find the best tradeoff between reliability and network lifetime.

As we previously mentioned, the SWITCH and ROOT mechanisms are complementary to each other, and as we can see in Fig. 3.11, combining them into SWITCH+ROOT manages to increase RPL's reliability close to 100%, as now all the nodes in the network implement a mechanism to overcome the memory problem. Still, the worse link quality on the secondary routes, inherited from SWITCH, and the lack of L2 acknowledgements for the link-local broadcasts prevent SWITCH+ROOT from reaching 100% of PDR on all topology densities. Still it is important to notice that this reliability is obtained while always keeping a duty cycle below 2%.

The combination of the ROOT and MCAST mechanisms into MCAST+ROOT leads to an overall lower duty cycle, while keeping similar PDR to MCAST, as can be seen in Tables 3.5, 3.6, and 3.7. To understand the reason behind that, we took a closer look at one of the smart city clusters. As Fig. 3.12 shows, thanks to the ROOT mechanism, there is a considerable decrease in the total number of junction nodes, mostly achieved by eliminating those from the immediate vicinity of the root. This can be clearly seen from the right-side chart of the figure, as the 1-hop junction nodes constitute the vast majority of those in the network for MCAST, while MCAST+ROOT does not have any 1-hop junction nodes. This means that the relatively expensive multicast mechanism is activated infrequently, considerably reducing the overhead in the network, and hence, the radio duty cycle.

The same reduction in duty cycle cannot be seen in MCAST+SWITCH. Indeed, even though SWITCH manages to restore some of the junction nodes to their normal behavior, these are situated lower in the network topology, and as Fig. 3.12 shows, their number is not very high.

Now, if we look at T-RPL, which combines all three mechanisms, we can notice that overall, it presents the lowest duty cycle, while having a reliability around 90%. As we can see in Tables 3.5, 3.6, and 3.7, T-RPL's reliability is above 89% in sparse scenarios, above 94% in intermediate scenarios, and above 96% in dense scenarios. The only case where it drops below 90% (to ~ 89%) is at sparse density and high number of nodes (134), as the 1-hop broadcast mechanism and the multicast are less reliable on large sparse topologies in general, and particularly in this cluster that has a group of nodes with very weak connectivity at high noise. The reason for which T-RPL does not reach 100% of PDR in all cases is that the 1-hop broadcast is less reliable than the unicast, as packets are not acknowledged at the MAC layer,



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

Figure 3.11 – Performance evaluation of the proposed combinations on all 13 clusters from our real-world smart city deployment, under different topology densities. The value on the $x-axis$ represents the size of the cluster (* marks the topologies that are linear).

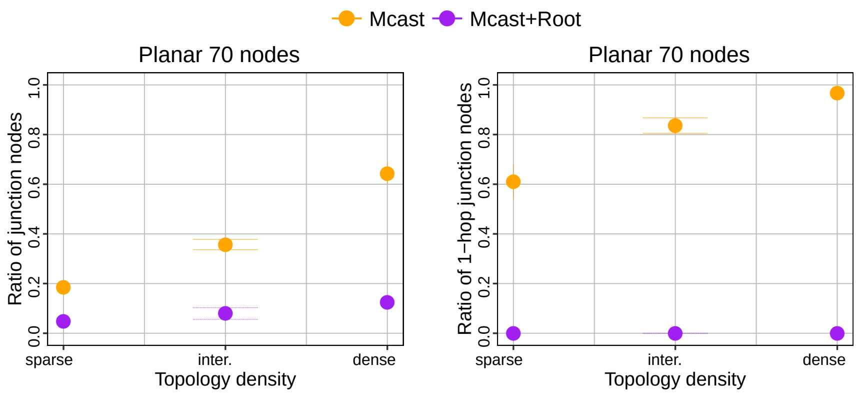


Figure 3.12 – Statistics with the number of junction nodes in the planar cluster with 70 nodes.

Table 3.5 – PDR (%) for all smart city clusters under the **sparse** topology density

	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
Mcast+Root	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
Mcast+Switch	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
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

and hence, they are not retransmitted. Still, such a low difference, might be not noticeable by most applications.

If we look at the duty cycle figures, we notice that T-RPL manages to significantly differentiate itself from the other multicast-based solutions, being the most energy efficient, while having roughly the same PDR. This shows us that by combining the advantages of the three techniques we obtain the best reliability among the RPL-based protocols while keeping the duty cycle low.

Summary. The results obtained until now show that T-RPL improves tremendously over ContikiRPL. Although T-RPL does not manage to maintain 100% of PDR in all cases, it gets very close (98%), while offering the lowest duty cycle. For applications that need 100% of reliability and have dense networks, Flooding might be a more suited protocol, however network lifetime has to suffer in this case.

3.7.3 Evaluation using synthetic topologies

In this section we continue the evaluation of the proposed solutions by experimenting with 2D regular grids, using the same approach as before: first we compare the proposed mechanisms individually, then their combinations. This will allow us to:

Table 3.6 – PDR (%) for all smart city clusters under the **intermediate** topology density

	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
Mcast+Root	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
Mcast+Switch	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
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

Table 3.7 – PDR (%) for all smart city clusters under the **dense** topology density

	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
Mcast+Root	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
Mcast+Switch	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
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

1. remove the bias of topologies using regular grids;
2. test the scalability of our solutions by almost doubling the size of the network;
3. provide an indirect validation of the smart city results.

Evaluation of the mechanisms. Figure 3.13 shows the comparison between the proposed mechanisms and our baseline: ContikiRPL and Flooding. We can notice that ContikiRPL does not scale at all, being severely affected by the routing and neighbor table problems. Flooding on the other hand, has perfect reliability on all network sizes, and the lowest delay among all protocols. However, its duty cycle is three times more than ContikiRPL, due to the overhead induces by duplicate packets.

SWITCH improves over ContikiRPL, but only slightly, barely managing to reach 20% of PDR when the network has 225 nodes. With the increase of network size SWITCH has problems scaling, as again, the PDR is bounded by the maximum number of routes that the root can keep.

ROOT also shows results consistent to what we saw on the smart city topologies, managing to maintain a PDR above 84%, but without reaching 100%. Still, it is impressive that a simple 1 hop broadcast mechanism can quadruple the reliability of ContikiRPL while maintaining a duty cycle below 1%.

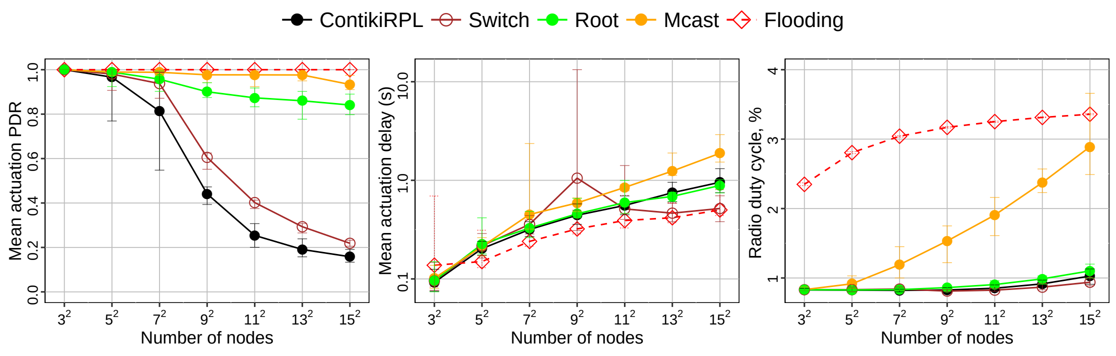


Figure 3.13 – Performance evaluation of proposed mechanisms on 2D regular grids, as a function of network size on the $x-axis$. Notice the logarithmic scale for the $y-axis$ on the figure representing the *delay*.

MCAST, on the other hand, has a slightly different behavior. Even if its reliability is similar to that on the smart city topologies, with a PDR above 93%, its duty cycle increases linearly with the size of the network, tripling w.r.t. ContikiRPL, and almost reaching Flooding. Indeed, as network size increase and more nodes overload their routing and neighbor tables, so does the number of data packets that are being sent using the multicast mechanism, increasing thus the duty cycle.

Evaluation of the combinations. Figure 3.14 shows the performance evaluation for the combinations of our mechanisms. SWITCH+ROOT manages to maintain a PDR above 91% on all network sizes, but without reaching 100%, which is consistent to what we saw on the smart city topologies. Still, it presents the lowest duty cycle among all the proposed solutions, remaining around 1%.

Among the multicast-based protocols, the performance of MCAST+SWITCH and T-RPL is remarkable, having a reliability above 97%, and respectively, 96% for all network sizes, as can be also seen from Table 3.8. The small difference in 1% between their PDR comes from the fact that T-RPL uses the slightly less reliable 1-hop broadcast mechanism at the root. However, even though the PDR is similar, T-RPL is more energy efficient, having the lowest duty cycle among them. Indeed, T-RPL consumes consistently less energy than MCAST and MCAST+SWITCH, as the addition of the root broadcast to the multicast solutions helps decrease the radio duty cycle.

Summary. The results obtained using unbiased grid topologies confirm the previous results from the smart city scenario: T-RPL manages to have the best tradeoff between reliability and energy efficiency, as it combines the advantages of all the proposed mechanisms. Although T-RPL does not manage to maintain 100% of PDR in all cases, it gets very close (98%), while offering the lowest duty cycle.

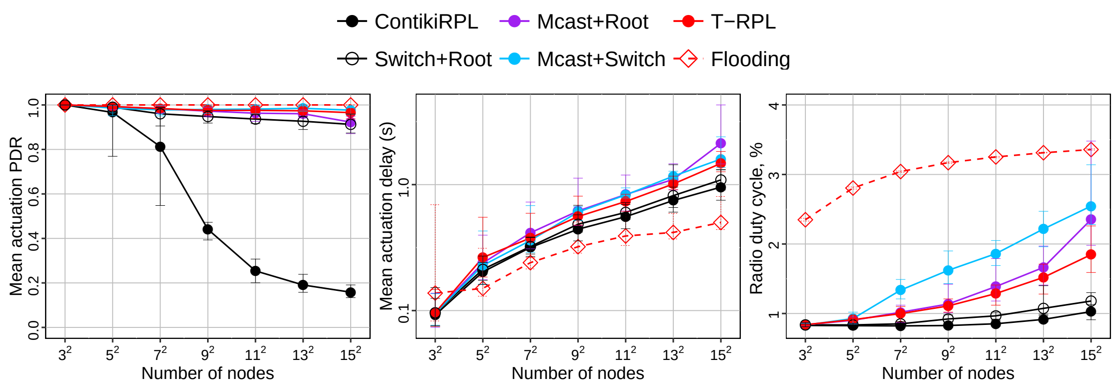


Figure 3.14 – Performance evaluation of proposed combinations on 2D regular grids, as a function of network size on the $x-axis$. Notice the logarithmic scale for the $y-axis$ on the figure representing the *delay*.

Table 3.8 – PDR (%) for synthetic topologies

	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
Mcast+Root	100	99.24	98.33	97.23	96.29	96.05	92.27
Mcast+Switch	100	98.62	97.79	97.99	98.14	98.49	97.67
T-RPL	100	99.30	98.34	97.55	97.56	97.37	96.53
Flooding	100	100	100	100	100	100	100

Flooding is the only protocol to reach 100% of reliability on all topologies, but with twice and in some cases even three times the duty cycle of T-RPL. Considering the redundancy of the data packets sent in the network, it is not surprising that in the end, Flooding manages to deliver everything, making it the best choice for applications that need 100% of reliability and do not care that much about energy consumption.

3.8 Evaluation against Existing Protocols

In this section we compare T-RPL, the protocol that offers the best tradeoff between reliability and duty cycle, against other routing protocols and implementations of the RPL standard, as they can bear significant differences [55].

3.8.1 State of the art protocols for downward routing

We compare our proposed solution against a series of routing protocols used for downward routing in WSNs, which have different characteristics:

- routing using individual unicast routes: Ad Hoc On-Demand Distance Vector (AODV) [82];
- opportunistic routing: Opportunistic RPL (ORPL) [33];
- dissemination/flooding: Trickle Multicast (TM) [47];
- different implementation of RPL: TinyRPL.

Ad Hoc On-Demand Distance Vector (AODV) is a reactive routing protocol that floods the network with $RouteRequest$ control packets in order to construct the routes. Each node registers locally an entry for the originator of the flooding, creating this way a reverse path. The destination replies (in unicast) with a $RouteReply$ message, and a bi-directional route is set between the source and the destination.

ORPL, among other benefits, aims at improving the scalability of downward routing in RPL by replacing the RPL path building mechanism with a hash- or bitmap-based one, adding also support for multiple downlink paths, and using the anycast-based opportunistic forwarding [60] at the link layer. ORPL also uses a custom version of ContikiMAC [31] as its duty-cycling MAC, the only one currently supported.

For comparison we also chose **TM**, which adaptively scales the number of retransmissions of the data packet depending on how many times the node has heard the same packet transmitted by its neighbors. The protocol allows nodes to resend data packets several times, if needed, and it should be less verbose than pure flooding, suppressing completely a retransmission if the data packet have been already sent by many neighbors. The retransmission period is also adaptive. When a new data message is received, the period is reset to the minimum, and doubled when there is no new data, until the maximum period is reached. This mechanism

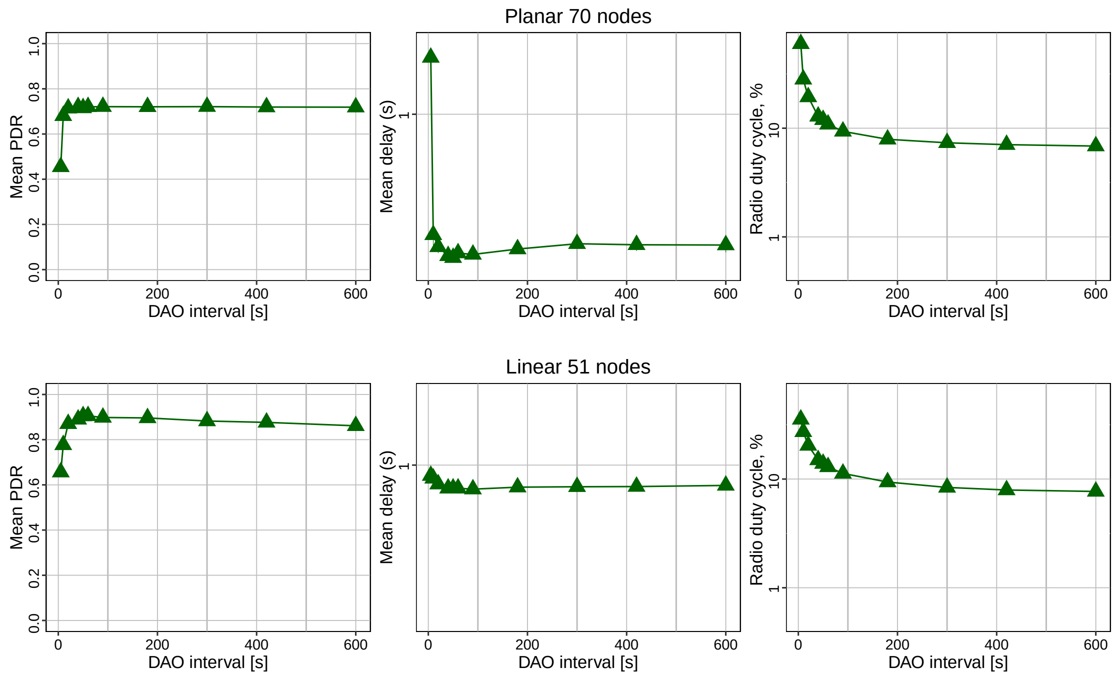


Figure 3.15 – TinyRPL performance as a function of different DAO interval values. Notice the logarithmic scale for the $y-axis$ on the figures representing the *delay* and the *duty cycle*.

provides eventual delivery of the message to all nodes, a guarantee not provided by pure flooding.

We also choose to compare our solution against **TinyRPL**, as this is the reference implementation of RPL for TinyOS, one of the mainstream operating system in the WSN community. When compared to ContikiRPL, TinyRPL has a significant difference that can severely impact the results, regarding the timer implementation for triggering DAO messages: instead of dynamically taking into account network changes (e.g., using Trickle [62] like ContikiRPL), TinyRPL sends DAO messages with a fixed periodicity, no matter the status of the network. This is less efficient and induces much more traffic in the network.

We experimented with a range of DAO interval values, to choose the best configuration for our scenario. Consequently, we simulated TinyRPL on two representative smart city topologies: a 70-node “planar" one and a 51-node “linear" one. We can see in Fig. 3.15 that as the interval increases, the PDR and delay stabilize, while the duty cycle decreases, as less control packets are sent in the network. In consequence, we chose to fix the value of the DAO interval to 180s for the rest of our simulations, as this offers the lowest duty cycle for the best PDR.

Protocols setup. 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 MAC protocol. Wherever possible, we used the default values, as these are likely to be

Table 3.9 – Protocol-specific parameters.

	MAC	Neighbor table size	Routing table size	Routing metric	Objective Function
ContikiRPL	ContikiMAC	20	50	ETX	MRHOF
T-RPL	ContikiMAC	20 (4 reserved)	50	ETX	MRHOF
Flooding	ContikiMAC	20	50	ETX	MRHOF
TM	ContikiMAC	20	–	ETX	MRHOF
ORPL	ContikiMAC [6]	40	–	EDC	custom
TinyRPL	BoxMAC	20 (parent table)	50	ETX	MRHOF
AODV	ContikiMAC	20	256	hop count	–

first choice in a deployment and the ones tested the most. The most important protocol parameters used in our study are summarized in Table 3.9. We can notice that ORPL can use a larger neighbor tables size than the other RPL-based protocols, as it does not use any routing table. Also, AODV has a considerably larger routing table, partly because it does not use IPv6 addresses, and partly because it's implementation occupies less memory than RPL's implementations.

3.8.2 Evaluation on smart city topologies

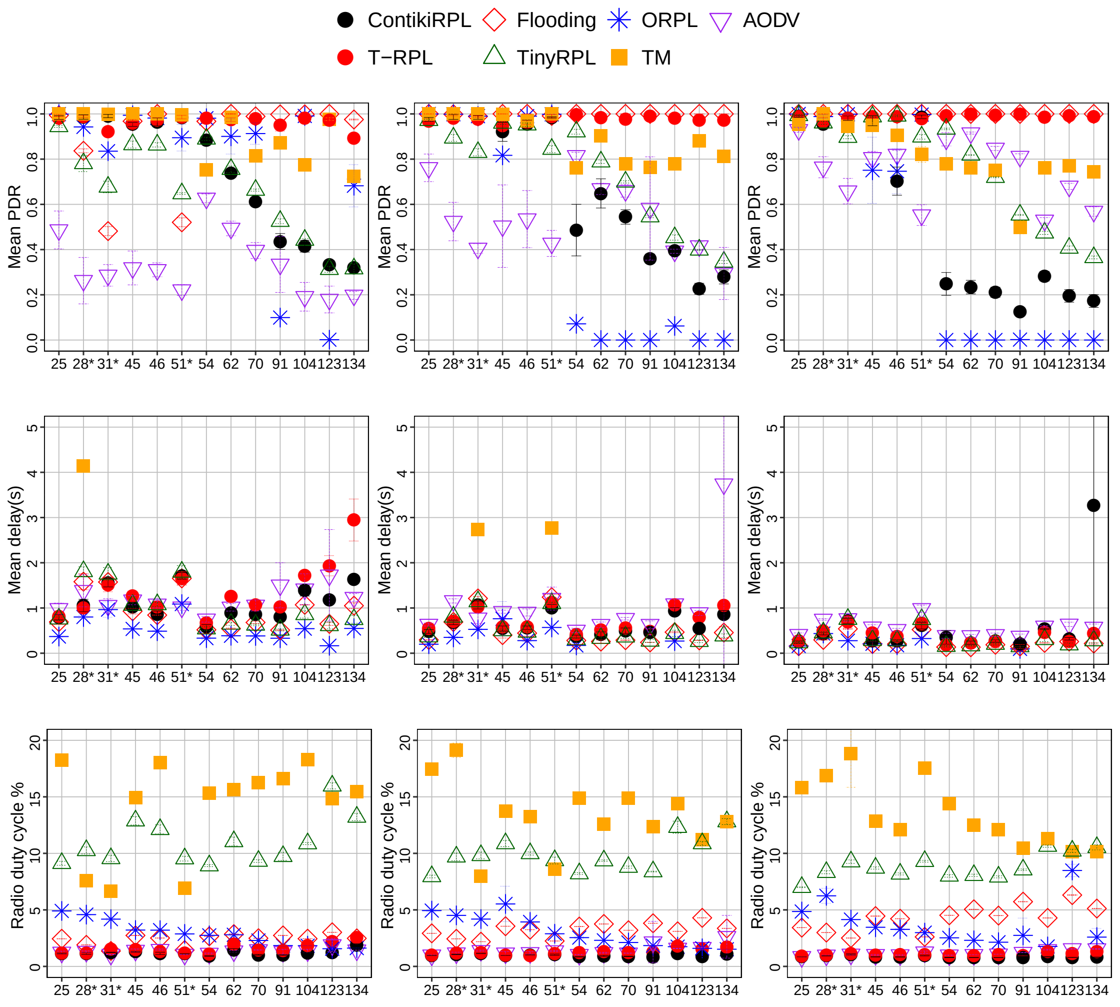
For the first step in our evaluation we keep the same scenario as before (Section 3.7.1): the root sends actuation commands towards the nodes with an inter-message interval of 10 seconds. We compare the selected protocols under different topology densities and on all the 13 network clusters from the smart city scenario.

We can see in Fig. 3.16 that TinyRPL also suffers from the memory problem, however, much less than ContikiRPL. The reason is in an optimisation implemented in TinyRPL that allows forwarding data packets without consulting the neighbor table for address conversion. In TinyRPL, the global IP addresses of the nodes are derived from their MAC addresses, making the opposite conversion possible without storing any state.

As a result, the neighbor table size does not limit forwarding and in the dense topologies the PDR goes up with respect to ContikiRPL. However, the upper bound of addressable devices still holds, with the maximum of 50 routes stored by the nodes, affecting the PDR on larger clusters. Further, because of the non-adaptive interval at which DAOs are sent in TinyRPL, its duty cycle is 10 times higher than that of ContikiRPL, significantly reducing the network lifetime.

ORPL shows the same density problems as ContikiRPL, but more accentuated, having a sudden drop of PDR to 0% when it reaches its memory limit in the neighbor table. This is the result of a complete lack of any mechanism for handling memory problems. In sparse topologies, as the number of neighbors is reduced, this problem is less severe.

[6] customized version



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

Figure 3.16 – Performance evaluation on all 13 clusters from our real-world smart city deployment, under different topology densities. The value on the $x - axis$ represents the size of the cluster (* marks the topologies that are linear).

AODV has a bad reliability throughout all density scenarios, and especially in the sparse case. Indeed, as nodes have less neighbors and paths are longer, there are more chances for the *RouteRequest* and/or *RouteResponse* messages to get lost, and hence the path remains partially not built. Furthermore, the hop-count metric tends to use longer but weaker links to minimise the path length. The low duty cycle in this is explained by the equally low PDR.

The reliability of TM is around 80% in large or dense networks. TM also has the highest duty cycle among all the protocols, even higher than that of Flooding, varying between 10 and 20%, as it uses a large number of broadcast control packets.

In what concerns the delay, there is a general trend of slightly reduced values for all protocols as density increases, due to a reduction of the network diameter.

T-RPL and Flooding are the only protocols that achieve a PDR close to 100% for all topologies, except for several points in sparse environments. However, Flooding is consistently less energy efficient, having 3 to 5 times the duty-cycle of T-RPL, due to the overhead induced by the link-layer broadcast.

3.8.3 Evaluation using synthetic topologies

In this section we continue our evaluation by removing the bias of the smart city topologies, using regular 2D grids, which also allows us to test the scalability of the protocols as we can easily increase the size of the networks.

We can see in Fig. 3.17 that the only protocol that manages to achieve perfect reliability is Flooding. Even though T-RPL does not reach 100% of reliability, its PDR remains above 97% (as mentioned in Table 3.8), while keeping the lowest duty cycle.

All the other protocols just do not scale, as already observed earlier. Even if TinyRPL does not suffer from the neighbor table problem, it still can hold maximum 50 route entries, which makes its reliability decrease once the network size is larger than 49 nodes. The performance of ORPL, again, suddenly drops from 100% to 0%, as the root's neighbor table reaches its maximum (40 entries), causing the protocol to stop working. The PDR of AODV seems to linearly decrease, due to an increased number of *RouteRequest* / *RouteReply* messages, which overload the network. Its duty cycle is lower than that of Flooding only because it delivers considerably fewer packets.

TM shows a constant PDR of 80% throughout all networks larger than 49 nodes, and the highest delay among all protocols. Indeed, due to its multicast mechanism and the high number of retransmissions TM overloads the network. This can also be seen in the high duty cycle, which is almost always above 10% (the only exception is the network with 9 nodes).

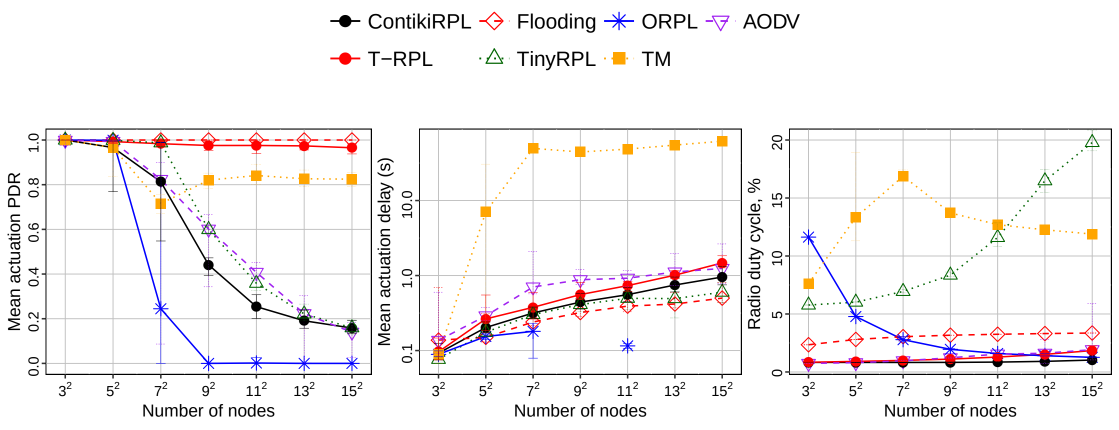


Figure 3.17 – Performance evaluation on 2D regular grids, as a function of network size on the $x-axis$. Notice the logarithmic scale for the $y-axis$ on the figure representing the *delay*.

3.8.4 Summary

The evaluation of our proposed protocol, T-RPL, against both ContikiRPL and TinyRPL shows that the problems highlighted in Section 3.4 are not implementation depended and a solution is greatly needed. Moreover, all of the downward routing protocols against which we compared showed the same inability to reliably deliver actuation commands in large networks. Our solution, T-RPL, and Flooding, are the only protocols able to reach a PDR close to 100%, offering different tradeoffs between reliability and energy efficiency.

3.9 Influence of Background Traffic

Finally, since our goal is to improve downward forwarding without hampering the data collection performance, we study the protocols when both actuation and data collection traffic is present. Note that all the protocols except for AODV and ORPL rely on the standard RPL mechanism to deliver upward traffic, i.e., TM, Flooding, and T-RPL, all use the unmodified version of ContikiRPL for data collection.

We keep the actuation traffic as before – the root sends actuation commands towards the nodes with an inter-message interval of 10 seconds, and we add a background collection traffic where each node sends a data packet to the root every 3 minutes. Although it might seem that the collection traffic is much less intensive, in reality it increases with scale. Indeed, for networks with more than 18 nodes (i.e., all the clusters from the smart city scenario), the data collection traffic is higher than the actuation traffic.

3.9.1 Evaluation on smart city topologies

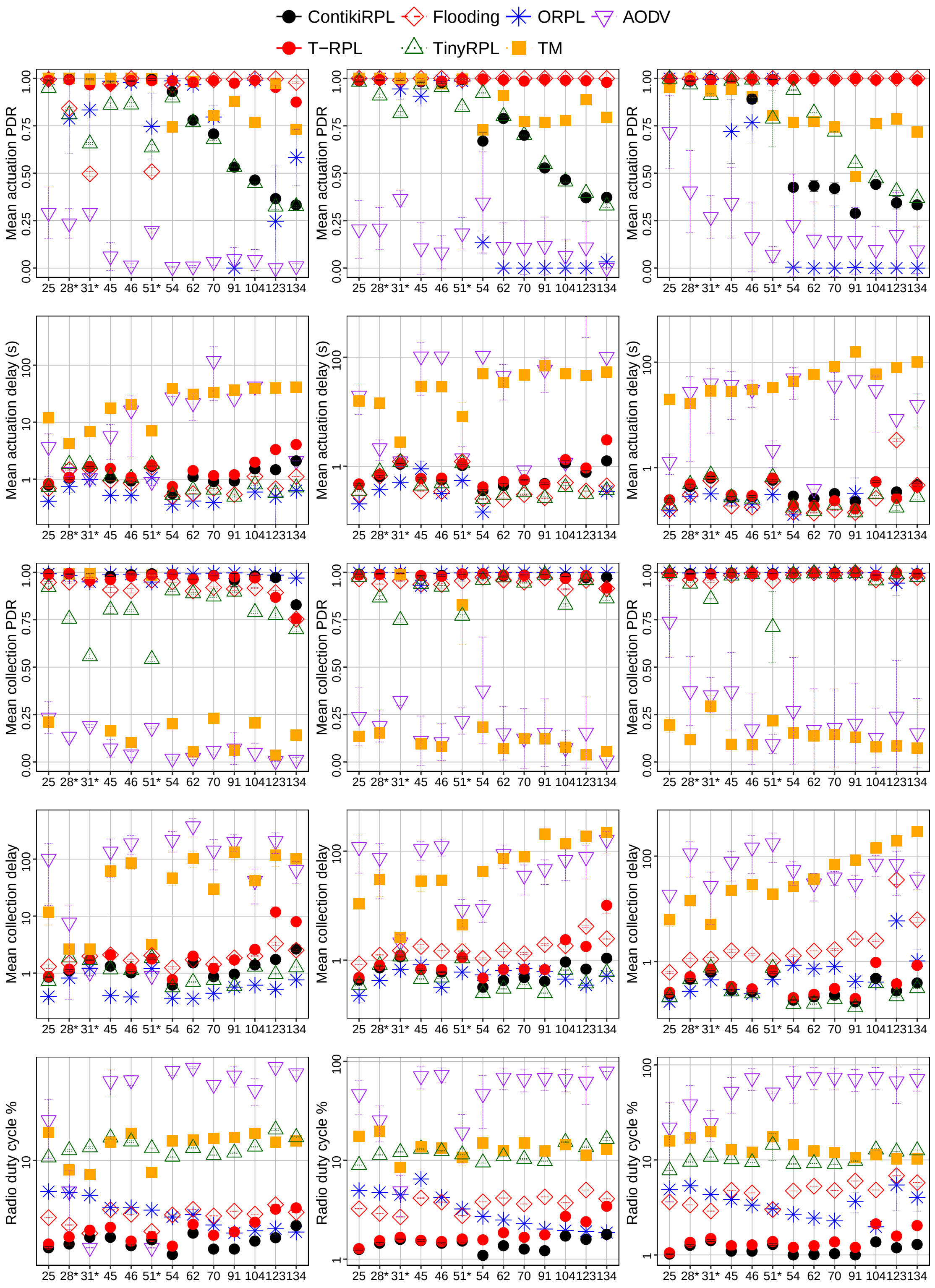
Figure 3.18 shows the results for all the smart city clusters, under three different topology densities, as before. If we compare the actuation PDR against the one from Fig. 3.16, where only actuation traffic was used, we can notice that the added collection traffic only impacts AODV, the only protocol that does not use RPL as the underlying mechanism for data collection. As a result, the data collection traffic and the corresponding control traffic for route discovery (*RouteRequest* / *RouteReply*) is overloading the network, and the protocol barely manages to deliver any packets (be it actuation or collection) with a duty cycle close to 100%.

A similar behavior can be seen also in TM, which although delivers the actuation traffic same as before, its collection PDR is below 25% in all clusters, for all topology densities. The high number of retransmissions needed by TM to deliver actuation traffic overloads the network and in consequence, the underlying RPL used for data collection fails. The network saturation affects also the time to deliver both actuation and collection packets, as TM and AODV present the highest delay.

Flooding also shows a sign of increased contention in the channel: the whole-network flooding used for actuation is overloading the network, hindering the collection. In consequence, Flooding reaches a PDR for the collection traffic only around 90% in the sparse scenario, and between 90 – 100% for the intermediate density. Even in the dense scenario, the reliability of collection does not manage to reach 100% on the linear topologies.

ORPL shows the best collection PDR among all protocols, throughout all network densities. However, it fails at delivering actuation traffic when networks become large and dense.

T-RPL is the only protocol delivering both actuation and collection traffic with high reliability, in all density scenarios, while having the lowest duty cycle.



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

Figure 3.18 – Performance evaluation on all the smart city topologies, combined actuation and data collection traffic. Note the logarithmic scale for the Delay and the Duty cycle.

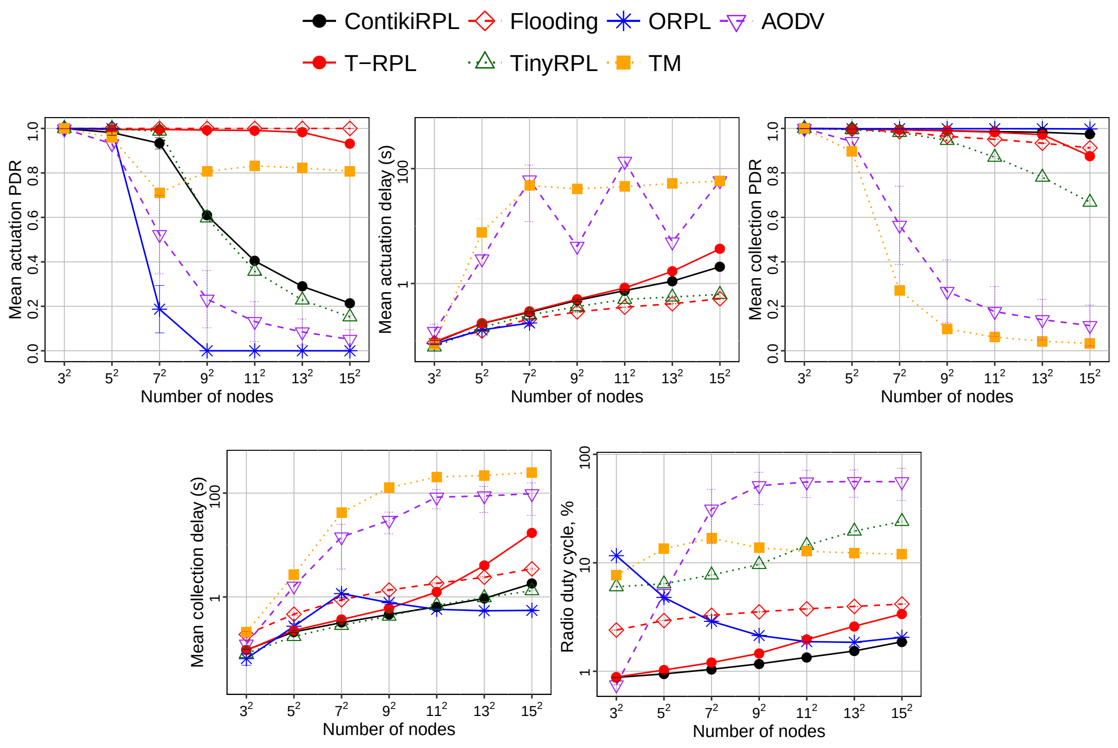


Figure 3.19 – Performance evaluation on the synthetic topologies with the **intermediate** density, combined actuation and data collection traffic. Note the logarithmic scale for the Delay and the Duty cycle.

3.9.2 Evaluation using synthetic topologies

Figure 3.19 shows the results of all the protocols over the synthetic topologies. Flooding performs very well, having a PDR of 100% for delivering actuation traffic. However, as we saw in the case of the smart city evaluation, its actuation forwarding overhead overloads the network to the detriment of the data collection. Indeed, as network size increases, collection data is increasingly dropped because of collisions and buffer saturation.

On the other hand, T-RPL maintains a PDR very close to 100% for both actuation and collection traffic, but it starts degrading faster than Flooding for actuation, and faster than ORPL for collection. Interestingly, the reliability of T-RPL goes below that of Flooding in the largest network.

If we look at the duty cycle results, we can notice that compared to Flooding, T-RPL does not spend energy when it is not needed: while Flooding shows an almost constant duty cycle over all network sizes, T-RPL increases its energy consumption only when the collection traffic intensifies (which happens the network grows).

On the opposite side, ORPL is the best for collection, having a duty cycle close to our pro-

Table 3.10 – Topology metrics for Indriya testbed at various transmission power levels.

TX Power	7 (–15 dBm)	15 (–7 dBm)	31 (0 dBm)
Avg degree	7	12	14
Max degree	15	26	25
Min degree	2	4	5
Avg sum out PDR	5.3	9	12
Max sum out PDR	13	18	21
Min sum out PDR	1.2	3.2	5
Avg hops	4.3	2.9	2.5
Max hops	8	5	4
Avg path ETX	5.5	3.6	2.9
Max path ETX	10.2	6	4.9

posed solution, T-RPL, but it does not manage to deliver any actuation traffic as soon as the neighborhood of the root exceeds the number of entries in the neighbor table.

Finally, the behavior that we saw for AODV and TM in the smart city scenario is exacerbated in this environment, as the collection PDR drops close to 0% and duty cycle rises close to 100% for AODV.

3.9.3 Summary

We showed in this section that no matter the topology, T-RPL offers the best tradeoff between reliability (both for actuation and collection traffic) and energy efficiency. On the other hand, while Flooding excels at delivering actuation traffic, though at a cost of a significantly increased energy consumption and reduced reliability of data collection.

3.10 Testbed Experiments

So far we focused on simulation to support our findings and evaluate the proposed protocol improvements. Simulation is a convenient tool that allows studying scalability and assess the impact of various parameters in isolation. However, to validate the simulation results we also ran experiments on real devices. We present in next our setup the performance evaluation of T-RPL against our baseline, ContikiRPL and Flooding.

3.10.1 Experiments setup

We tested our protocols in the Indriya testbed [27], which consists of around 100 TelosB motes located on three floors of an office building. We have varied the TX power to affect the topology density, and, consequently the link quality among the nodes (the equivalent of using different noise levels in simulation).

Table 3.10 summarizes the topology characteristics measured in Indriya at the selected three TX power levels. We can notices that while the average number of neighbors are lower than what we obtained in simulation, the average number of hops and path ETX are still comparable with the sparse and intermediate densities of the smart city networks.

We evaluated the protocols under the same metrics as before: reliability (mean actuation PDR), delay, and duty cycle. However, unlike simulation where Cooja reported the percentage of time the radio of the nodes was active, on real devices we relied on Energest [32], the built-in energy profiler of Contiki to estimate the duty cycle metric. Unlike the charts we used before, here the error bars present the minimum and the maximum of those metrics observed across several runs, because the number of testbed runs was not enough to accurately compute the confidence intervals.

As before, we studied the protocol performance under two traffic patterns, downward (actuation) traffic only, and a combination of actuation and collection traffic. We kept the same inter-message interval (IMI) as before: one actuation packet generated by the root with a fixed interval of 10 seconds, and one collection packet generated by each node in the network every 3 minutes. In addition, we also ran tests with a lower intensity of the collection traffic, with each node sending a packet once every 10 minutes, as the collection IMI of 3 minutes turned out to be challenging for the protocols. Indeed, in this case, in a network of 100 nodes, a collection packet reaches the sink roughly once every two seconds.

3.10.2 Downward traffic evaluation

Figure 3.20 shows the result of the ROOT and MCAST mechanisms, as well as our main solution, T-RPL, when compared to the baseline, ContikiRPL and Flooding. In general, the results of the testbed experiments match well the ones from the simulations. ContikiRPL shows poor actuation performance at all TX powers, due to the limitations of the routing and neighbor table sizes.

Interestingly, even in much less dense scenarios, ROOT goes a long way, keeping a PDR close to 90%, still not reaching 100%. Its less reliable 1-hop broadcast mechanism is also what keeps T-RPL from having a perfect reliability in all the scenarios. However, T-RPL manages to overcome the storage problem of ContikiRPL showing a PDR above 96%, while keeping a low duty cycle (around 1%).

Again, Flooding is the only protocols that reaches 100% of PDR in all the scenarios, even in sparser topologies. However, when TX power increases, so does the duty cycle, as in denser topologies every broadcast disturbs and awakens more nodes.

3.10.3 Influence of background traffic

Figures 3.21 and 3.22 show the performance of the studied protocols under two intensities of the background data collection traffic. Thanks to the inherent redundancy, the flooding-based forwarding shows the best actuation reliability among the three protocols even under the intense collection traffic.

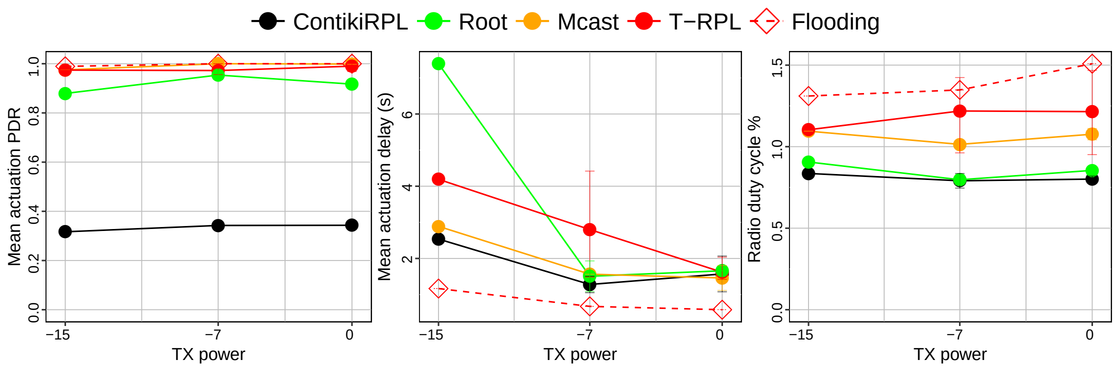


Figure 3.20 – Actuation performance in Indriya testbed with different TX powers (dBm).

Table 3.11 – Actuation PDR (%) in Indriya testbed with different TX powers (dBm)

	−15	−7	0
ContikiRPL	31.68	34.25	34.32
Root	87.91	95.39	91.71
Mcast	97.50	100	99.86
T-RPL	97.41	97.23	99.00
Flooding_with_dao	98.92	100	100

Flooding is also the fastest protocol, as it requires several times less time to forward the actuation traffic to the destination. At the downside, at high network density (high TX power) it negatively affects data collection traffic, resulting in slightly lower values of the PDR due to increased contention in the channel. This is also reflected in the increased collection delays coinciding with the flooding-based actuation.

T-RPL seems to suffer from the background data traffic more than Flooding, as it does not reach 100% of PDR, not even when the collection IMI is 10 minutes. This could be again related to the unreliability of the ROOT mechanism, as the upward traffic can create more collisions, and the actuation traffic will not be retransmitted by the root due to the lack of layer two broadcast acknowledgements. Still, the duty cycle of the T-RPL is consistently lower than that of Flooding, sometimes reaching two-fold difference.

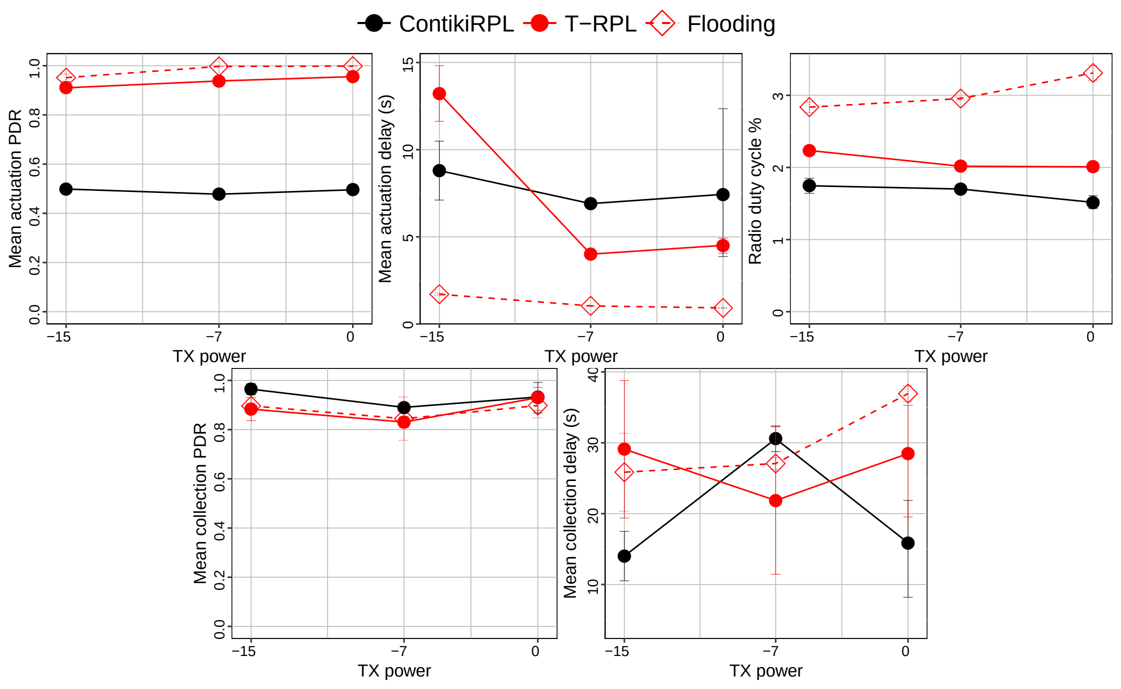


Figure 3.21 – Protocol performance with the combined actuation and collection traffic in Indriya testbed with different TX powers (dBm) and the collection IMI of 3 minutes.

Table 3.12 – Actuation and collection PDR (%) in Indriya testbed with different TX powers (dBm) and collection IMI of 3 minutes.

	Actuation			Collection		
	−15	−7	0	−15	−7	0
ContikiRPL	49.86	47.78	49.57	96.45	89.04	93.27
T-RPL	91.02	93.80	95.57	88.34	83.07	93.01
Flooding	95.15	99.71	99.85	89.65	84.44	89.80

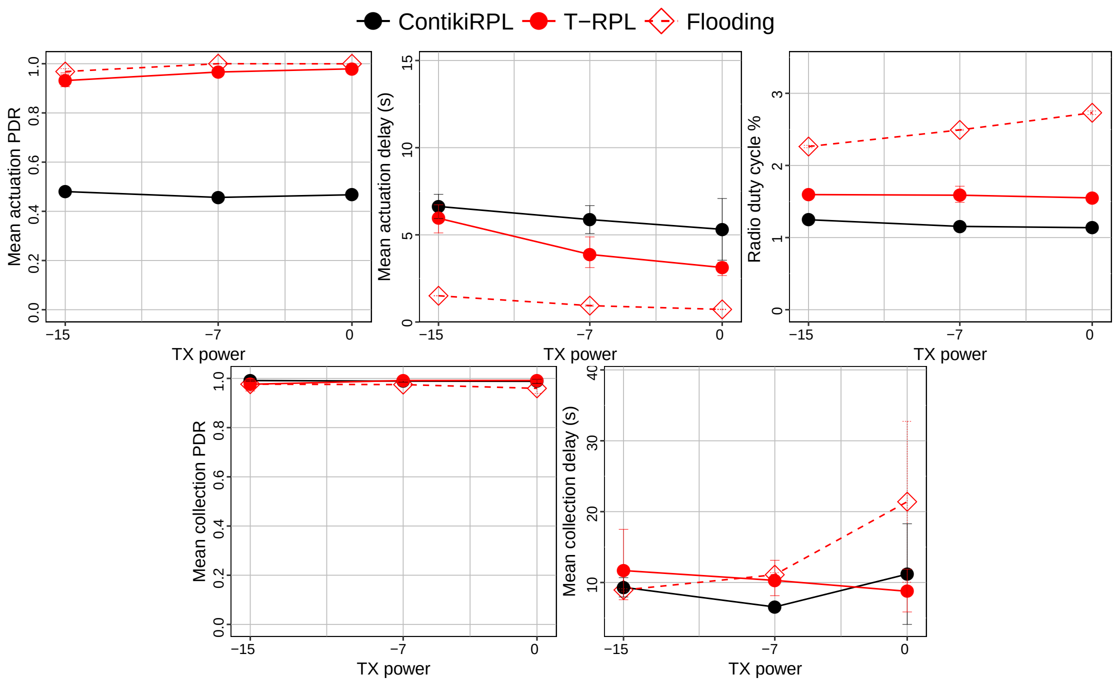


Figure 3.22 – Protocol performance with the combined actuation and collection traffic in Indriya testbed with different TX powers (dBm) and the collection IMI of 10 minutes.

Table 3.13 – Actuation and collection PDR (%) in Indriya testbed with different TX powers (dBm) and collection IMI of 10 minutes.

	Actuation			Collection		
	−15	−7	0	−15	−7	0
ContikiRPL	48.03	45.65	46.72	99.14	98.90	98.83
T-RPL	93.15	96.63	97.92	97.63	99.15	99.19
Flooding	96.82	100	99.96	97.66	97.50	95.96

3.11 Related work

Since RPL was standardized by IETF, a lot of studies have explored its data collection performance, topology stability, and energy efficiency [40], [50], [55], [88]. Still, the evaluation of point-to-multipoint communication required for actuation has been marginalized.

Clausen *et al.* criticized the design of RPL in what concerns its approach for point-to-multipoint communication: DAO transmission generates too much control traffic overhead, nodes close to the root have to store routing information for too many destinations, and elements such as the DAO timer or the DAO-ACK messages are underspecified [21]. While this paper shows RPL's shortcomings, the analysis was made just by studying the RPL standard, lacking any type of experimental evaluation.

Ko *et al.* compared RPL head to head with the Collection Tree Protocol (CTP) [45], the de-facto standard routing protocol in TinyOS [57]. Even though the experiments were carried out on an indoor testbed with only 30 nodes, the authors warn the readers that in order to support routes to all nodes in the network, the network size must be limited. This observation matches our findings for small clusters under low noise: nodes quickly fill their memory, not being able to support routes to all destinations.

The neighbor table problem that we described in this chapter was also reported in [25] and [100]. The authors observed that reaching the limit of the neighbor table size creates undesirable churn in the network because some of the entries are being dropped just to be later rediscovered with a non-accurate link quality estimation. We saw during our tests that the excessive churn hampers the protocol performance, especially the downward forwarding, as unneeded DAO traffic is generated, removing the routes from the old parents (by sending No-Path DAOs) and re-establishing them through the new parent. These additional control packets not only occupy the medium, but also increase the chances that the routing tables are not consistent if some of the DAO packets get lost, causing partially built routes and, consequently, drops of downward data packets.

An answer to the many problems encountered by the point-to-multipoint communication came from Duquennoy *et al.*, who proposed a new opportunistic routing protocol, ORPL, that uses bitmaps and Bloom filters to represent and propagate the routing information [33]. This representation is very compact and therefore allows networks to grow larger. In our tests we confirm the superiority of ORPL over RPL in many cases, however, we also shown that the current implementation of the protocol does not handle the situation when the network becomes denser than the neighbor table threshold.

In an effort to get the advantages of both the storing and non-storing modes, researchers have tried to mix both modes of operation in the same network even though this is not allowed by the RPL standard. The authors of MERPL [41] proposed to use source routing (like in the non-storing mode of RPL) for a subset of nodes that cannot be reached in a purely storing mode due to memory limitations of the forwarders along the path. However, this requires an assumption that the sink potentially needs to store the whole topology of its DODAG, furthermore the evaluation was done in simulation without a realistic radio channel model and it is unknown how this solution will perform with unreliable and/or saturated links. Another way of mixing the modes of operation was implemented in DualMOP-RPL [56], however that RPL modification aimed at improving interoperability of co-located RPL devices of different capabilities, and, as a consequence, connectivity and resilience to failures, but did not provide a solution for insufficient storage.

Low-power wireless stacks for industrial networks like WirelessHART and ISA100.11a [83], although interesting for the scope of this work, were excluded from our study. They both use TSCH-based data link layers and a dedicated network device, Network Manager, that collects link statistics from the network, computes routes and the timeslot schedules for the individual links along the routes, based on the connectivity graph. However, up to our knowledge, there is now open implementation of the routing and scheduling functionality of the Network Manager.

3.12 Discussion and Conclusion

RPL, the IPv6 routing protocol for LLNs, became our first choice for the smart city application scenario described in Section 3.1.1 thanks to the built-in support for both upward and downward communications. On the other hand, RPL is optimised for data collection by design, and we show that it is not only less efficient for actuation traffic, but it is often unable to deliver any packets to a part of the network. The reason is that the unicast downward forwarding used by RPL needs a significant amount of memory for keeping the network state, which limits the downward connectivity to around 50 devices on a typical hardware platform. Thus, RPL shows its shortcomings precisely when applied to the constrained devices it was designed for.

We based our study on two RPL implementations, ContikiRPL and TinyRPL, that are substantially different in the underlying implementation choices. Still, in our tests, they both were severely affected by the memory constraints, illustrating that the identified problem is not implementation-specific. Hence, this work was targeted at finding a solution that would improve downward forwarding in RPL by restoring connectivity with network devices in much larger networks, on the same memory-constrained platform.

We introduced three techniques for dealing with the limited device memory. The first one, SWITCH, tries to balance the memory used by the nodes by making them search for an alternative parent that still can store the routing information. However, when all potential parents have saturated their routing or neighbor tables, this technique alone does not help. When routing cannot be done, another possibility to enable connectivity with all destinations is to rely on a form of dissemination.

Therefore, the idea behind the second proposed mechanism, ROOT, comes from the observation that the root is the first node in the network to saturates its routing table, since it has to store the next-hop address for all of the network destinations. In the cases when the routing entry is not present and the next-hop node is not known for a data packet, the root just broadcasts it to all its neighbors. This technique alone proved to be quite efficient: reliability on our network topologies has improved fourfold (from 20% to 80% and more) while keeping the same duty cycle as that of our baseline ContikiRPL.

In larger networks, the neighbors of the root may also have their memory saturated, therefore, the third proposed mechanism, MCAST, generalizes the idea of ROOT and extends it over multiple hops. It uses multicast forwarding to overcome areas with nodes that have exhausted their memory. This increases both the protocol reliability and the scope of dissemination, still keeping it limited, as the normal unicast forwarding is resumed as soon as the packet reaches a node knowing the path to destination.

We studied the proposed mechanisms (SWITCH, ROOT, MCAST) in separation and in combination to understand their trade-offs and limitations. As a second baseline besides RPL we used a simple flooding, representative of dissemination approaches. In general, we demonstrated that extending the scope of data dissemination improved the protocol reliability but negatively affected energy efficiency. In fact, the unlimited flooding provided the best reliability in most of our tests. At the same time, on very sparse topologies Flooding is the least reliable, and it is consistently the worst in terms of the radio duty cycle. Flooding is also too aggressive at forwarding the actuation traffic, which hampers data collection.

Both simulation and testbed experiments showed that our ultimate solution, T-RPL, that combines the three proposed techniques, offers the best trade-off between reliability and energy efficiency. Even though T-RPL is not best in every single aspect, its main feature is versatility. It manages to provide first class performance according to several metrics and maintain this performance consistently when facing challenging scenarios without the need for manual protocol parameter tuning, on different types of topologies, with and without background traffic. In this sense, we have engineered it to be robust to topology extremes in terms of structure, density and scale, and to overcome device memory constraints. We demonstrated that the downward service of T-RPL scales up to hundreds of devices without affecting much the upward traffic nor the battery lifetime.

We argue that the proposed techniques can be applied to any RPL implementation, providing benefits similar to what we have shown in this work. Being layer-three techniques they are expected to unleash their full potential when combined with the time-slotted channel hopping

(TSCH) MAC protocol that achieves remarkable improvements at layer two [34]. Recently, an implementation of IEEE802.15.4 TSCH was made available for Contiki, and it is a part of our future plans to test it in combination with T-RPL.

The techniques presented in this chapter all build on top of a standard-compliant IPv6 stack for LLNs. In the next chapter we explore possibilities of achieving even better performance in a novel stack based on synchronous transmissions and designed without constraints imposed by the standards.

Part III
Efficient and Reliable Data Collection Based on Synchronous Transmissions

4 Data Prediction + Synchronous Transmissions = Ultra-low Power Wireless Sensor Networks

On our way towards a reliable, fast and energy efficient stack supporting bidirectional connectivity we dedicated the first part of this thesis to improving the point-to-multipoint (actuation) service of a standard-compliant IoT protocol stack. We managed to alleviate its scalability problems that had been limiting the number of reachable devices in the network to around half a hundred on memory-constrained devices. The solution partially relied on a limited-scope data dissemination and enabled the reachability, though, at a cost of elevated network utilisation and energy spendings. Regarding the reliability, the proposed protocols showed packet loss rate of around 1% in a network of more than 200 devices, bringing the actuation component of the stack on par with the data collection one. However, even higher delivery rates are required for certain control applications, while further reduction of the energy consumption is always favourable.

Therefore, in this chapter we approach the problem from the opposite direction. We rely on synchronous transmissions made popular by Glossy [38] to design a novel data collection protocol that can complement existing excellent actuation services based on Glossy (e.g., LWB [39], Blink [110] and sharing their remarkable properties.

Indeed, these recent (and thus non-standard) protocols were shown to be highly energy-efficient, fast, extremely reliable and have a very modest needs for program and data memory.

In implementing the data collection service we could just reuse one of existing protocols either based on conventional multi-path forwarding or on scheduled Glossy floods, however, as we show next, the existing approaches were not adaptive enough to the immediate traffic demands of the application thus requiring significant over-provisioning and, consequently, high energy consumption. This becomes especially important for applications that generate aperiodic traffic, for example when *data prediction*, introduced next, is used.

The contents of this chapter have been originally published in: **Data Prediction + Synchronous Transmissions = Ultra-low Power Wireless Sensor Networks**, Timofei Istomin, Amy L. Murphy, Gian Pietro Picco, Usman Raza, In Proceedings of the 14th ACM Conference on Embedded Networked Sensor Systems (SenSys 2016), Stanford (CA, USA), November 2016, and have been slightly adapted for this thesis.

Data prediction [49] has emerged as an application-level technique to reduce the amount of data generated and transmitted by WSN nodes. In this approach, each node constructs a mathematical model, shared by the node and the sink, to approximate future sensed data. As long as sensor readings fall within an application-defined tolerance of the value predicted by the model, no data is transmitted. When significant deviations occur, a new model is generated and sent to the sink. Data prediction is particularly effective when applied to environmental data such as light and temperature, suppressing up to 99% of message transmissions.

Recent work [89], however, has shown that this significant reduction of application messages does not lead to analogous savings when considering the lifetime of the system as a whole; e.g., when applying data prediction over a typical stack of CTP and BoX-MAC, only a 7-fold lifetime was achieved. Idle listening in the MAC layer and routing overhead prohibit further lifetime improvements.

Protocols based on synchronous transmissions neither maintain a topology nor rely on a duty-cycled MAC, making it tempting to place synchronous transmissions at the core of the collection aspect of the network stack.

In fact, work has been done to exploit Glossy for collection of periodic data [39, 95], the traffic resulting from data prediction is aperiodic, making these approaches inapplicable.

Indeed, inspection of the traffic profiles induced by data prediction (Section 4.1) reveals long periods of inactivity when models accurately predict the data; occasionally, these models must be updated and are transmitted by nodes to the sink. However, the time interval between updates and the number of updates to be communicated concurrently are irregular, not known globally, and ultimately unpredictable.

These observations lead to two key requirements (Section 4.2) for our new stack, CRYSTAL[1]. First, when there is nothing to transmit, the *network overhead must be minimized.* Second, model updates themselves must be delivered in both a *timely and reliable manner* despite their unpredictable nature in terms of distribution over time and number of concurrent transmissions.

CRYSTAL approaches these conflicting requirements by using Glossy network flooding as a primitive to build reliable data collection with data prediction at the topmost layer. CRYSTAL inherits the aforementioned properties of Glossy to broadcast a single message, and offers a simple but effective mechanism to provide reliability of the unpredictable and possibly concurrent model transmissions arising from data prediction. The core of CRYSTAL is a periodic, flexible sequence of synchronized slots organized in pairs, providing a network-wide transport protocol of sorts. In the first slot, all nodes with data to transmit send it by initiating a Glossy flood. In the second slot, it is the sink that initiates a Glossy flood, acknowledging which packet it has received, if any. Due to the capture effect [61], a property of IEEE 802.15.4 radios, the sink is highly likely to receive one of the packets even when there are multiple, concurrent

[1]Crystal balls are often associated with the ability to predict the future; further, a crystal is a beautiful, *glossy* object.

transmitters. The alternate execution of these two slots is repeated continuously inside a reporting interval; a distributed termination policy allows all network nodes to determine when the transmission sequence is complete, i.e., all data has been received by the sink, and nodes can safely go to sleep.

As CRYSTAL relies on Glossy, we offer a concise primer about it (Section 4.3) followed by a complete CRYSTAL protocol description (Section 4.4). An analytical model (Section 4.5) provides the foundation to analyze the energy consumption of CRYSTAL. An extensive set of 90-node experiments in the Indriya testbed [27] enable us to characterize the operation of CRYSTAL (Section 4.6) by determining experimentally a few key parameters that determine the accuracy of the model, allowing us to identify a good configuration and experimentally validate the model itself. Finally, we close the circle by using *both* our model *and* our implementation of CRYSTAL to determine the duty cycle it can achieve on 7 publicly-available real-world datasets (Section 4.7). We confirm our claim that CRYSTAL *achieves per-mille duty cycle* and lower, and show experimentally that this translates into improvements up to 80x over the CTP + BoX-MAC baseline, therefore bringing low energy consumption to levels hitherto possible only via specialized hardware.

We end the chapter by surveying related work (Section 4.8), followed by brief concluding remarks (Section 4.9).

4.1 Data Prediction: Network Implications

Data prediction enables the suppression of a remarkable number of periodic data reports, greatly reducing the need for communication in WSNs and improving significantly their lifetime. Nevertheless, the potential benefits brought by this application-level technique can be reaped only to some extent when applied to mainstream network stacks; these are designed for periodic traffic and therefore are ill-suited for the aperiodic, sparse traffic induced by data prediction. In this section we discuss qualitatively and quantitatively these issues, for a specific data prediction technique.

4.1.1 Derivative-Based Prediction

Several prediction techniques exist [49], with varying degrees of complexity and accuracy. In this work, we adopt Derivative Based Prediction (DBP) [89], in which each node constructs a linear model to predict the data. The model is formed by taking a sequence of m sensor values and approximating the slope (the derivative) of the data by a line formed by two points, respectively the average of the initial and final l values in the sequence. This model is used to predict the subsequent sensor values. As long as the actual, sensed value is within a certain value tolerance of the predicted value, no data is sent. Instead, if the sensed value falls outside the value tolerance for a given time tolerance, a new model is generated from the last m sensed values and sent to the sink.

We choose DBP as it is the most recent in the literature, shown to perform equivalently or better than the state of the art. Further, it is the only one that has been evaluated on a real WSN platform and for which the interplay with the underlying network stack has been evaluated. As summarized next, this constitutes the motivation for the work presented in this chapter.

4.1.2 Data Prediction on a Staple WSN Stack

The authors of [89] report message suppression rates up to 99% w.r.t. periodic collection, based on several publicly-available datasets we also use here for comparison; they are summarized in Table 4.1 and discussed in Section 4.1.3.

Nevertheless, in the same work the authors also show that the savings on the system as a whole are not as significant when the staple WSN stack constituted by CTP [45] and BoX-MAC [70] is used. The maximum improvement is seen by exploiting the characteristics of the sparse traffic induced by data prediction inside the configuration of the underlying network stack. Specifically, the infrequent transmissions generated by model updates enable in BoX-MAC the use of a sleep interval much longer than in the periodic case. This sleep interval can be further increased if, at the same time, a much longer maximum beaconing interval is used in CTP.

Differently from [89], in this work we use the Indriya testbed. Therefore, to establish a baseline for CRYSTAL, we apply the same experimental methodology and datasets in Indriya, validating the results of [89] in this setting. Figure 4.1 shows results for the INDOOR temperature dataset. The lowest duty cycle $DC = 4.778\%$ for periodic reporting is achieved with a MAC sleep interval of 500 ms, while data prediction yields the lowest $DC = 1.146\%$ (4.17x improvement) with a sleep interval of 2.5 s. The cross-layer configuration of data prediction, MAC, and routing achieves the lowest $DC = 0.743\%$ (6.4x improvement) with a sleep interval of 3 s and a maximum beaconing interval of 4000 s, instead of the default 500 s. Both data prediction configurations achieve 100% data yield, thanks to reduced contention, while the plain periodic configuration achieves 98.3%.

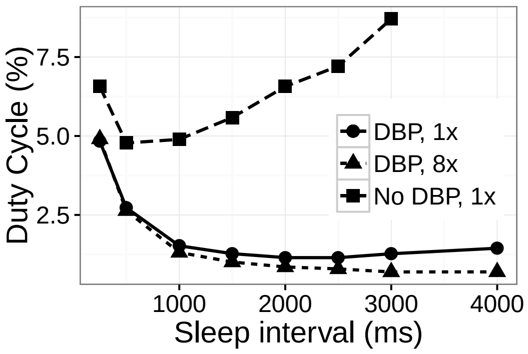


Figure 4.1 – Exploring the best configuration for CTP, BoX-MAC, and DBP-based data prediction in Indriya.

The sparse traffic induced by data prediction implies that energy consumption is dominated

Table 4.1 – Datasets characteristics.

Application & Dataset		Epoch	Nodes	Samples	Description
INDOOR	temperature	30 s	54	2,303,255	A 36-day dataset from an Intel Berkeley Research Lab WSN deployment; used by many papers on data prediction, e.g., [99, 80].
	humidity	30 s	54	2,303,255	
	light	30 s	54	2,303,255	
SOIL	air temperature	10 min.	10	225,360	A 225-day dataset from the Life Under Your Feet project [63]; the WSN is deployed in forests to study soil properties.
	soil temperature	10 min.	4	77,904	
TUNNEL	light	30 s	40	5,414,400	The 47-day dataset used in the WSN-based closed-loop control system for road tunnel lighting described in [16].
WATER	chlorine	5 min.	166	715,460	A dataset from a sensor network monitoring a water distribution system, simulated via the EPANET 2.0 [35] tool, used in several previous works (e.g., [81, 8]).

Table 4.2 – Data prediction applied to the INDOOR temperature dataset.

	updates sent in a given epoch															
	TOT	0–1	≥ 2	0	1	2	3	4	5	6	7	8	9	10	11,12	13
epoch occurrences	103K	99.8K	2.9K	84.3K	15.5K	2.2K	432	131	43	21	13	5	3	4	0	1
% over #epochs	100	97.2	2.8	82.1	15.1	2.2	0.4	0.1	0.04	0.02	0.01	0.005	0.003	0.004	0	0.001
#updates	22.3K	15.5K	6.8K	N/A	15.5K	4.4K	1.3K	524	215	126	91	40	27	40	0	13
% over #updates	100	69.4	30.6	N/A	69.4	19.9	5.8	2.3	1	0.6	0.4	0.2	0.1	0.2	0	0.1

by network overhead. In the best CTP/DBP configuration, 65% of the energy is spent in idle listening, 25% in transmitting beacons for tree maintenance, and only 10% in transmitting model updates. This observation motivates the approach presented in this chapter that, based on synchronous transmissions, entirely removes the need for a duty-cycling MAC and the maintenance of a routing topology.

4.1.3 Traffic Patterns with Data Prediction

We mentioned that data prediction induces a very sparse traffic w.r.t. periodic collection. We now quantify this statement based on the same publicly-available datasets used in [89], concisely summarized in Table 4.1.

Without data prediction, each WSN node reports a sample during each reporting period, hereafter called *epoch*. Epochs are not synchronized; nevertheless, if we were to discretize time based on their duration, each epoch would "see" a number of messages equal to the number of nodes, e.g., 54 in the INDOOR dataset. The message suppression achieved by data prediction in DBP dramatically changes this behavior, as shown in Figure 4.2. Discretizing the model updates in our traces over the epoch size shows that the vast majority of epochs contains *no* message transmission (Figure 4.2a); further, epochs with *at most* one message transmission are common in several datasets (Figure 4.2c). On the other hand, epochs with *multiple* message transmissions in the same epoch do exist, although their frequency depends on the phenomena at hand. For instance, Figure 4.2a shows that most datasets show a sort of "exponential decay", where the maximum number of transmissions in the same epoch is between 2 (SOIL temperature) and 19 (INDOOR light). The only exception is the WATER dataset (Figure 4.2b) where, due to the dynamics of the underlying physical phenomena sensed and the long epoch duration of 5 minutes, *i)* a negligible number of epochs contain at most one update, and *ii)* up to 38 message transmissions in the same epoch are observed.

Table 4.2 offers a closer look at the number of epochs with multiple updates for the INDOOR temperature dataset, the one analyzed in-depth in [89]. The table also illustrates the number of updates occurring with some other in the same epoch—an important factor for protocol design, as discussed next. For instance, the table shows that although only 2.80% of the epochs see $u > 2$ updates, these “concurrent” updates are 30.57% of all those to be disseminated. Figure 4.2c provides a similar view for all datasets, showing the fraction of updates that are *not* isolated in an epoch. Again, the WATER dataset is the exception: almost all updates occur concurrently with at least one other update.

4.2 Crystal: Design Rationale

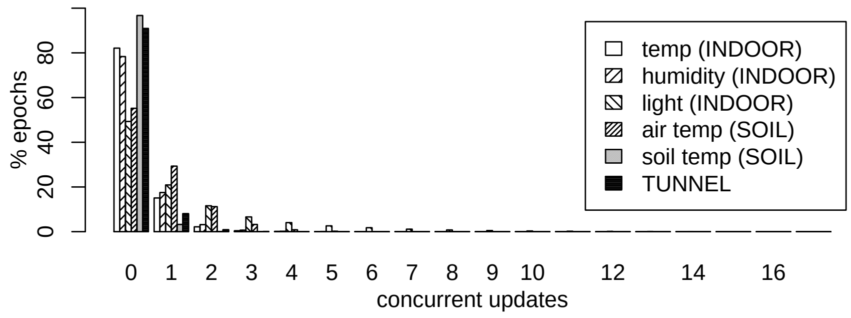
The quantitative considerations in Section 4.1.2–4.1.3 allow us to distill the **goals** that inspired the design of CRYSTAL:

Goal 1. *Minimal network overhead in the control plane.* An obvious goal of our design is to harvest the potential gains offered by the message suppression of data prediction, which drastically reduces data transmissions. As discussed in Section 4.1.3, this creates a traffic pattern with *no* message transmissions during the majority of reporting epochs. Section 4.1.2 shows this clashes with the operation of a perfectly tuned, mainstream WSN stack due to control overhead.

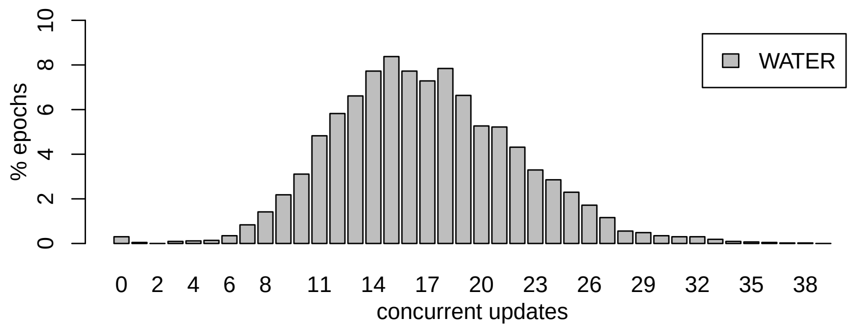
Goal 2. *Timely and reliable dissemination of unpredictable model updates.* We mentioned that the distribution of model updates across epochs is not known a priori; at the beginning of an epoch we cannot know if there will be several updates or none. Still, in the former case, applications demand that *all* pending updates are disseminated *within the epoch in which they were generated.* Deferring the update to a later epoch or, worse, losing an update, causes the sink to become unaware of changes in the actual data sensed at the nodes. This is exacerbated if the network is part of a control system, whose actions may be delayed or even incorrect.

Our **solution** to tackle these goals is a network stack based on synchronous transmissions that, requiring neither a MAC layer nor topology maintenance, is in line with Goal 1. The fundamental communication primitive is network-wide flooding, performed by exploiting physical properties of wireless communication (i.e., constructive interference and the capture effect [61]) yielding rapid and reliable packet dissemination, in line with Goal 2.

Nevertheless, this choice has consequences. Synchronous transmissions require all nodes to be simultaneously awake to help disseminate the flooded packet. For us, this requires a global schedule to ensure nodes are awake when an update must be disseminated. However, this schedule must consider that Goal 1 and Goal 2 pose conflicting concerns. On one hand, the desire to minimize control overhead (Goal 1) implies that *nodes should normally be awake as briefly as possible during an epoch, as in most cases no transmissions occur.* This argues for



(a) Percentage of epochs in which a given number of concurrent updates occur (all datasets except WATER).



(b) Percentage of epochs in which a given number of concurrent updates occur (only WATER).

Application & Dataset		%epochs with $u \leq 1$	%updates *not* isolated
INDOOR	temperature	97.19	30.57
	humidity	95.84	36.32
	light	70.19	84.13
SOIL	air temperature	84.51	55.52
	soil temperature	99.94	3.42
TUNNEL	light	99.00	20.77
WATER	chlorine	0.34	99.99

(c) Percentage of epochs where at most one update occurs and, dually, fraction of updates occurring with others in a given epoch.

Figure 4.2 – Distribution of model updates over epochs, in the datasets of Table 4.1.

a very short schedule. At the other extreme, the need for timely and reliable dissemination of unpredictable model updates (Goal 2) demands that *nodes with a pending update have enough opportunities to transmit and recover from rare packet loss within a single epoch.* This argues for a long-enough schedule accommodating all nodes with updates, whose number or even presence is impossible to ascertain without (very expensive) global knowledge.

CRYSTAL reconciles these conflicting goals with the mechanics of synchronous transmissions by essentially providing a *network-wide transport protocol* atop Glossy. In a nutshell, nodes with data simultaneously attempt to transmit their updates. After each transmission, the sink must acknowledge which packet it has received, if any. When all transmissions have completed, the network returns to sleep. Before going into the details of CRYSTAL in Section 4.4, we offer a concise primer on synchronous transmissions.

4.3 Synchronous Transmissions

Simply put, Glossy offers network-wide packet flooding and high-accuracy synchronization. In Glossy, a single node initiates a flood with a single transmission. Neighboring nodes receive it and immediately retransmit it, with their neighbors doing the same. While such a straightforward approach seems to lead to an inordinate number of collisions with many nodes transmitting simultaneously, Glossy observes that such concurrent transmissions need not be negative. In fact, they can be exploited due to a phenomenon of IEEE 802.15.4 radios called the *capture effect.* In these radios, a node can receive a packet despite interference from other transmitters when the signal of that packet is stronger than other signals or when the node begins to receive the packet sufficiently earlier than other signals. Further, if multiple transmissions initiate with a tiny temporal difference (smaller than 0.5 µs), the transmissions *constructively interfere*, increasing the probability of reception.

Glossy builds on these two phenomena, carefully controlling the timing of retransmissions to encourage constructive interference and to reliably flood a packet from a single initiator throughout the network. Experiments show that, with a single initiator, a Glossy flood reaches all network nodes with reliability >99% in few milliseconds, depending on the configuration parameters. To increase the flooding reliability, Glossy allows nodes to retransmit packets multiple times, denoting this with N.

4.4 Crystal: Protocol Description

We recall that our goals are to keep nodes asleep as much as possible and to disseminate the model updates in a timely, reliable fashion. CRYSTAL accomplishes these goals by using Glossy for rapid, highly reliable flooding. CRYSTAL itself is periodic, with the epoch determining when communication is possible toward the sink. Each epoch is formed by a very short active portion in which all nodes participate in data collection, and a much longer sleep portion when nodes consume very little power. The intricacies of CRYSTAL lie in how we guarantee that

Table 4.3 – CRYSTAL parameters.

Parameter	**Description**
E	Epoch duration (reporting period)
W_S, W_T, W_A	Glossy maximum listening interval (slot duration) for the **S**, **T**, **A** phases
N_S, N_T, N_A	Number of Glossy transmissions in **S**, **T**, **A** phases
G	Guard time before the **S**, **T**, **A** slots
R	Number of consecutive silent **T** slots triggering the termination of the active portion of the epoch at the sink
Y	Number of consecutive **TA** pairs with zero packets causing a network node without data go to sleep
Z	Number of consecutive missing acknowledgements causing a network node with data go to sleep

all updates are reliably received during the active portion using only Glossy transmissions.

In a nutshell. A CRYSTAL epoch starts with a synchronizing Glossy transmission from the sink, ensuring all nodes are temporally aligned and ready to participate in data collection. Subsequently, any node with a data packet to send transmits it with a Glossy flood. Due to the capture effect, at least one of these packets is highly likely to reach the sink, which then sends an acknowledgement via a Glossy flood, announcing the ID of the packet it received. With high reliability, all senders receive this acknowledgement, and if their data packet was not acknowledged, they simply try again by transmitting data then listening for the acknowledgement. This repeats until all transmitting nodes have received an acknowledgement for their data, then all nodes go to sleep.

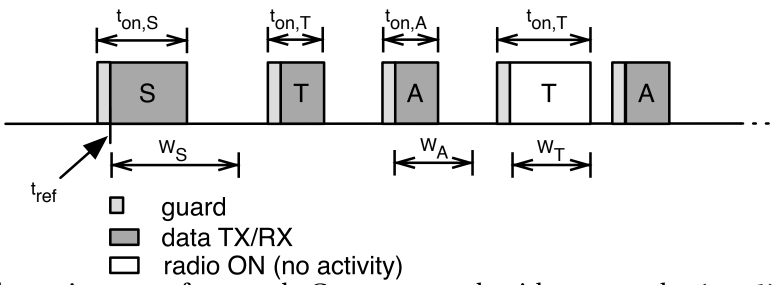


Figure 4.3 – The active part of a sample CRYSTAL epoch with one sender ($u = 1$) whose data is immediately acknowledged by the sink. For simplicity, $R = 1$.

An example. Figure 4.3 offers a sample of the active portion of a single CRYSTAL epoch, shown as a sequence of Glossy transmissions. Table 4.3 offers the key parameters. The first slot, **S**, contains a synchronization message from the sink, and serves the purpose of preparing the network nodes for communication following the previous, long sleep interval. This is followed by some number of **TA** pairs in which **T** represents a data transmission slot for use by nodes transmitting data and **A** is an acknowledgement slot for use by the sink. The number of **TA** pairs in each epoch varies depending on the number of nodes with data to transmit and the desired reliability. To identify the end of the active portion of the epoch, we define a distributed termination policy, detailed later. Note that although we refer to this as the *active* portion, when a node is not involved in communication (either receiving or transmitting) its radio is off. Slots have a duration W, defining the maximum interval a node listens on the channel to detect an ongoing Glossy flood. When the latter occurs, it normally completes before the end of W, as seen by comparing the two **T** slots in Figure 4.3.

Detailing TA. Inside a single **TA** pair, all nodes with data to send become Glossy initiators, meaning they initiate floods in the **T** slot, which are then carried out concurrently throughout the network. All non-initiators act as forwarders. Although Glossy is a flooding protocol, we focus on packet reception *at the sink*, as our goal is data collection. Glossy was designed to work with only a single initiator, but our experiments in Section 4.6 show that, due to the capture effect, one of the concurrent transmissions reaches the sink with a probability close to 1. In this case, the **T** slot is *successful*, and the sink floods a positive acknowledgement in the next **A** slot. If, instead, the sink does not receive a packet, it floods a negative acknowledgement. Turning our attention back to the network, if the positive acknowledgement reaches the corresponding sender, its data is known to have been received at the sink and the sender will not attempt retransmission.

Distributed termination. All **TA** pairs follow this structure with the expectation that, in each subsequent **TA**, there will be fewer initiators until, eventually, some number of consecutive **T** slots have no data, triggering termination of the active portion of the epoch. We call these **TA** pairs without data *silent pairs*.

The number R of *consecutive* silent pairs is key in determining termination, based on three conditions. First, the sink goes to sleep after R consecutive **T** slots without data. Therefore, if a data packet did not get through in an earlier **T** slot, a node has $R-1$ additional attempts to transmit before the sink stops listening. Increasing R decreases the probability of falsely detecting the end of the transmissions, at the expense of energy consumption.

To indicate the end of the active part of the epoch, the sink piggybacks the sleep command in its last negative acknowledgement by setting its *sleep bit* to 1. When such an acknowledgement arrives to a network node, it goes to sleep immediately, as there is no reason to stay awake when the sink is already sleeping. This is the second termination condition.

While these two conditions are nearly always sufficient, due to the occasional loss of acknowledgements we cannot rely on the second condition alone to put network nodes to sleep. Therefore, we define a third condition, used at the network nodes, that *i)* puts a node to sleep if it is mistakenly awake due to the loss of the last acknowledgement (the one carrying the sleep bit), but *ii)* simultaneously keeps a node awake for some additional time which is helpful in noisy conditions when it is unable to detect activity in the network due to high interference.

We distinguish whether the node still has unacknowledged data to send. If yes, it goes to sleep when it misses Z acknowledgements from the sink in a row. In this case, the node's data might remain undelivered during the epoch, however transmission can be attempted again in the next epoch. Alternatively, if the node has no unacknowledged messages, it goes to sleep only when it detects Y consecutive **TA** pairs with *zero* packets, i.e., neither data in **T** nor acknowledgement in **A**.

Intuitively, this third condition expresses the fact that a node "keeps trying" until the sink is likely asleep or inaccessible and there are no nodes around trying to deliver their packets. The last part of the condition increases overall reliability in situations when the node's neighbors have better connectivity and, therefore, might be still receiving sink's acknowledgements. In this case, the current node will stay awake to serve as a forwarder as long as needed.

Synchronization. It is critical that CRYSTAL ensures all nodes are properly aligned to wake up and participate in data collection. This is particularly challenging in applications with long epochs, e.g., WATER. CRYSTAL accomplishes time alignment by beginning the epoch with a Glossy synchronizing packet and prepending this **S** slot with a sufficiently long guard time G to compensate for clock drift. For applications with long epochs, or systems composed of nodes with significant clock drift, this approach can lead to large guard times and increased consumption. Therefore, our CRYSTAL implementation includes a mechanism to *learn* the clock skew at each node, and adjust the wake-up period accordingly. While this works remarkably well, guards are still needed due to imperfect estimation and changes in clock skew over time. In addition to a guard at the beginning of the epoch, each **T** and **A** slot also includes a guard time; this compensates for clock drift should the synchronization packet be lost.

All nodes expect to receive the synchronizing Glossy message from the sink within $G + W_S$. By starting the CRYSTAL epoch with a synchronization packet from the sink, we expect to spread with high probability the correct reference start time t_{ref} to all nodes. Nevertheless, our implementation allows both G and W_S to grow with the number of consecutive losses of this synchronization packet.

Further, if the number of consecutive transmitters is large, the **TA** sequence can become similarly long, increasing the risk that nodes lose synchronization in the middle. To combat this, CRYSTAL makes every **A** slot a synchronizing Glossy slot, bringing all nodes back in line.

Glossy reliability. Inside a single flood, the Glossy protocol allows packets to be repeated a variable number of times N. A higher number of repetitions increases reliability but also power consumption, as we show in Section 4.6. CRYSTAL leaves the number of repetitions for each slot type, N_S, N_T, and N_A, to be configured to meet application requirements. For instance, Figure 4.3 shows the transmission in **S** longer than the transmissions in the **T** and **A** slots, a choice ensuring that the synchronization message at the beginning of the epoch is more reliable than the other transmissions, as discussed in Section 4.6.3.

4.5 Analytical Model

We derive an analytical model for CRYSTAL, estimating the average, network-wide radio-on time T_{on} within an epoch, a key constituent to estimate duty cycle and therefore lifetime. We leverage this model in Section 4.7, to compute accurately the performance of CRYSTAL over the data profiles described in Section 4.1.3 and elicit the associated tradeoffs, without replaying month-long datasets. We validate the model in Section 4.6.5, based on the key parameters we measure in the rest of Section 4.6. The validation in Section 4.6.5 shows that the model we present in this section is very accurate. From Section 4.4, it is evident that T_{on} depends on the number u of concurrent updates, as this determines the minimum number of **TA** pairs necessary for their dissemination. We estimate $T_{on}(u)$ in two ways: an upper bound that uses only CRYSTAL's configuration parameters, and a much more accurate model that requires a few measurements of some constituents of CRYSTAL.

Upper bound. The average radio-on time across the entire network is (over)approximated by:

$$T_{on}(u) = W'_S + (u+R)(W'_T + W'_A) \tag{4.1}$$

where $W'_x = G + W_x$ is the slot duration in Glossy augmented by the short guard time discussed in Section 4.4, and R is the number of silent **TA** pairs. The equation above is a strict upper bound for T_{on} because, when an actual transmission takes place in a slot, the actual average per-slot radio-on time is $t_{on} < W'$. Knowing this value t_{on} for each slot type enables us to derive a much more accurate estimate, discussed next. Eq. (4.1) assumes perfectly reliable dissemination, potentially underestimating T_{on} when retransmissions occur. However, as discussed next and shown empirically in Section 4.6.4, retransmissions are extremely rare; the underestimation caused by neglecting them is amply overcome by the overestimation caused by considering W' in place of t_{on}.

Model. Determining the values of t_{on} for each slot type, hereafter referred to as $t_{on,S}$, $t_{on,T}$, $t_{on,A}$, is necessary to obtain accurate estimates of $T_{on}(u)$. The values of $t_{on,S}$ and $t_{on,A}$ can be safely assumed constant w.r.t. u, as transmission in the **S** and **A** slots is performed only by the sink; they are effectively a normal Glossy dissemination. This does not hold for **T** slots, in which multiple update senders may compete, and for which the value of u affects the radio-on time $t_{on,T}(u)$, as shown experimentally in Section 4.6.2.

Interestingly, the value of t_{on} (regardless of the slot type) also implicitly depends on W. Indeed, while the strict inequality $t_{on} < W'$ holds on average, this is not true for a single update transmission; if the expected number of packets N is not received, a node remains awake for the entire slot. If we know the fraction σ of nodes that complete their flood before the end of the slot, and the average time $\hat{t}_{on}$ it takes, we can formalize the dependency of t_{on} on W as:

$$t_{on} = \sigma \hat{t}_{on} + (1-\sigma)W' \tag{4.2}$$

Empirical knowledge of all these parameters, which we acquire in Section 4.6.2, allows us to determine an accurate estimate of the average per-epoch radio-on time as:

$$T_{on}(u) = t_{on,S} + u\rho(t_{on,T}(u) + t_{on,A}) + R(W'_T + t_{on,A}) \tag{4.3}$$

where ρ is the average number of **TA** pairs required to successfully deliver an update to the sink. This parameter also implicitly defines the probability that an update is successfully disseminated in a single pair, easily computed as $\frac{1}{\rho} = p_T p_A$, where p_T is the probability that the update is received at the sink in a **T** slot, and p_A the probability that the acknowledgement sent by the sink is received in the subsequent **A** slot. In practice, as we show in Section 4.6.2, these probabilities *i)* depend on the number N of Glossy retransmissions *ii)* are both very high, causing ρ to be very small.

4.6 Characterization

We have implemented CRYSTAL atop the original publicly-available version of Glossy based on ContikiOS for the TMote Sky platform. In this section, we analyze the operation of our CRYSTAL implementation with the double goal of identifying and quantifying the main factors affecting its performance, as well as of measuring the parameters necessary to inform the model in Eq. (4.3).

After describing our experimental setup (Section 4.6.1) we focus on the mechanics of the **S**, **T**, **A** slots (Section 4.6.2). We then use this information to identify the best configuration for running CRYSTAL in the Indriya testbed (Section 4.6.3) and use it in Section 4.6.4 to characterize the performance of CRYSTAL at the level of a single round of execution within an epoch. This data is used in Section 4.6.5 to validate our model, which is then exploited in Section 4.7 to derive duty cycle estimates based on the datasets of Section 4.1.3.

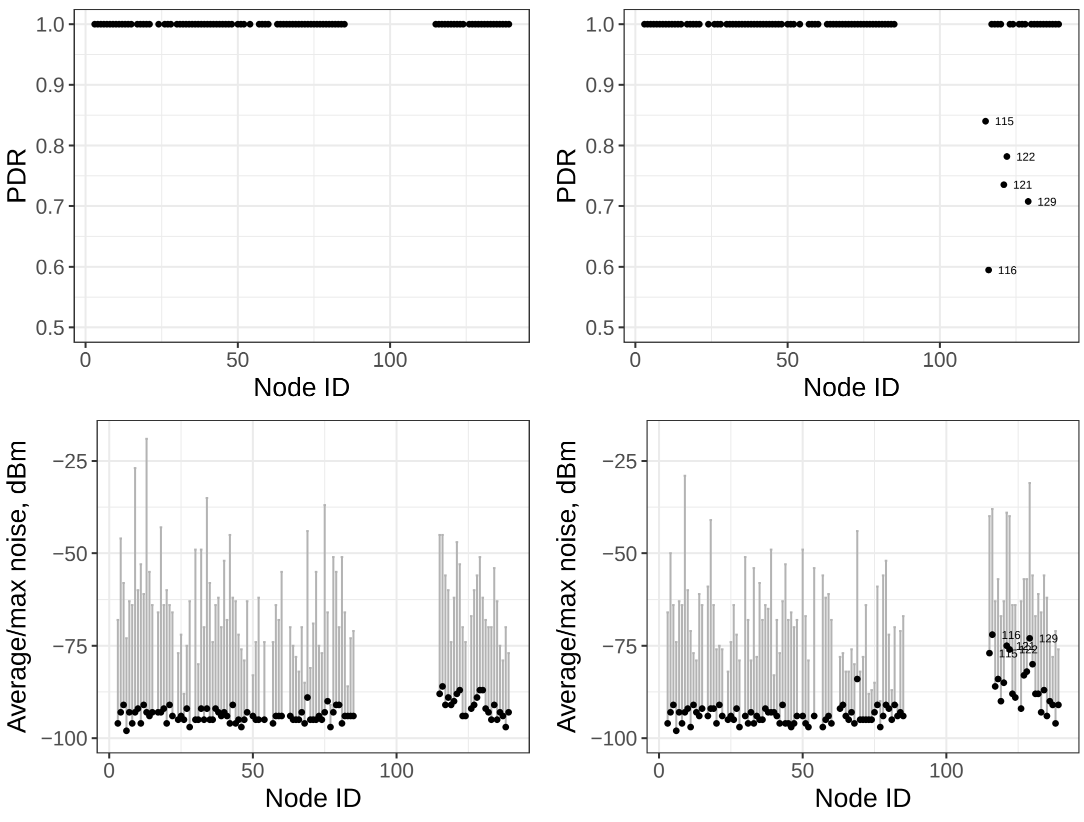
4.6.1 Experimental Setup

We ran our experiments in the Indriya testbed that, at that time of writing, had 88–92 operational nodes. We generated two topologies with different transmit power levels, 0 dBm (power 31) and -15 dBm (power 7), yielding an average network diameter of 4 and 7 hops, respectively.

We tested CRYSTAL during the night and also during the day, when the interference from Wi-Fi networks is significantly higher. We chose two channels, 20 and 26, which are believed to have respectively high and low influence from Wi-Fi. To assess the actual interference during the experiments, our CRYSTAL test application sampled and logged noise (RSSI) in the inactive portion of each epoch. Additionally, we ran regular, isolated network connectivity tests probing all links individually, without any concurrent transmissions. This identified two nodes (71 and 83) unpredictably losing connectivity, which we removed from our analysis.

CRYSTAL showed very similar performance on both channels during the night runs; however, the daytime results were inconsistent and difficult to assess. For example, while the majority of tests on channel 20 during the day yielded perfect reliability as in nighttime runs, in some others the packets from a handful of nodes were sometimes lost on the way to the sink. For instance, in one run 5 nodes showed a packet loss between 25% and 40% (Figure 4.4b). Closer inspection revealed that these nodes were exposed to an average noise of around

−70 dBm. Although we conjecture that the resilience built into CRYSTAL is an asset in these harsh conditions, a full analysis and comparison w.r.t. the state of the art requires the ability to control and reproduce interference patterns. Therefore, in this chapter we report the results only from night runs on channel 26. This is not to say that these experiments are interference-free: the average noise is between −90 and −95 dBm, while the maximum noise is often above −70 dBm and as high as −30 dBm for several nodes. This holds for both channel 20 and 26, despite the fact that the latter is often purported to be interference-free. In the next chapter we study in detail how strong interference affects CRYSTAL.



(a) Results at a lower noise (b) Results at a higher noise

Figure 4.4 – Examples of Crystal runs on a busy channel 20 during the day with inconsistent results. PDR (top) and average/maximum noise (bottom) registered for each node.

As for the scheduling of transmissions, for every test a unique table of nodes that should send packets in any given epoch was randomly generated and “replayed” cyclically by the nodes. Logging was performed via serial line during the inactive portion of an epoch. The logs contained information about transmission and reception of every message, and other vital information for each node and epoch. Node-level statistics (over all epochs) and network-wide ones (over all epochs and nodes) have been calculated offline from the logs.

4.6.2 Dissecting a Crystal Slot

Knowledge about key metrics of CRYSTAL slot types is fundamental to inform the model we defined in Section 4.5, enabling a correct configuration of the system as well as accurate duty cycle estimates given an update traffic profile.

Setup. To measure these parameters, we run specialized "benchmarks", where each slot type is measured in isolation. We used **S** phases only (pure Glossy) or a combination of **S** and a single **TA** pair. We used $E = 250$ ms to increase the sampling rate. We set $W = 20$ ms to ensure that all floods have enough time to complete. The results we show here are based on 1500 epochs for each individual point on the plot.

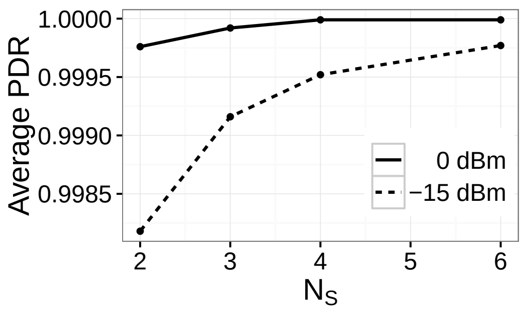
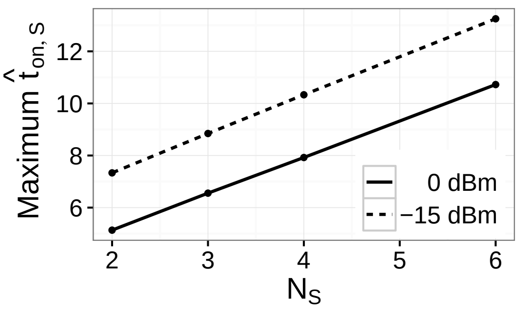
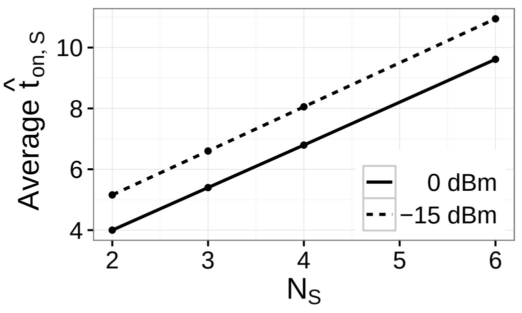


Figure 4.5 – **S** phase: average PDR.



(a) Maximum.



(b) Average.

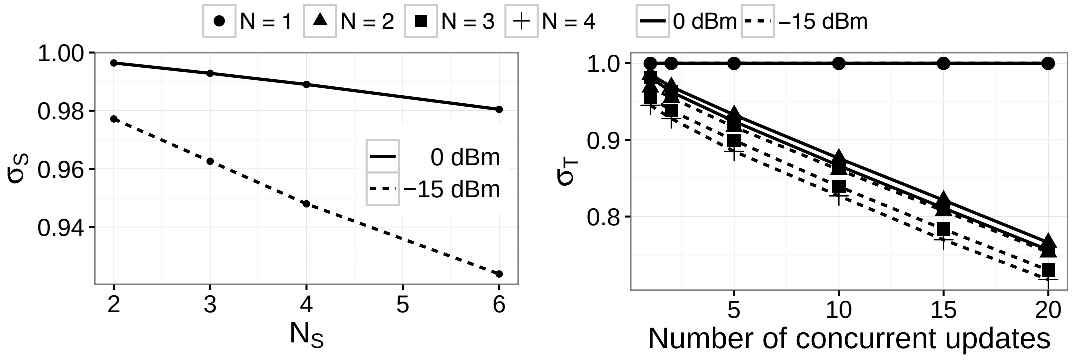
Figure 4.6 – **S** phase: per-slot radio-on time, $\hat{t}_{on,S}$. Note the different y-axis scale.

S and A phases. These phases are pure Glossy disseminations, as they are always performed by the sink. Figure 4.5 shows that the packet delivery rate (PDR) of the **S** phase, defined as the percentage of nodes that correctly receive the packet sent by the sink; PDR is very high, in line with results reported in the literature [38, 39, 59]. Furthermore, Figure 4.5 also confirms that, given a value of N, the corresponding reliability decreases as the network diameter increases, and is therefore lower in the low power case.

Figure 4.6 shows the per-slot radio-on time $\hat{t}_{on,S}$, computed only for the fraction σ_S of nodes receiving *all* N Glossy transmissions. $\hat{t}_{on,S}$ is crucial to dimension properly the slot duration W_S; if the latter is shorter than $\hat{t}_{on,S}$, the Glossy dissemination may not reach distant nodes.

Therefore, W_S should be larger than the maximum value of $\hat{t}_{on,S}$, shown in Figure 4.6a.

On the other hand, W_S cannot be too large. Even if the disseminated packet is received with high probability (as shown in Figure 4.5), some of the individual N transmissions may be lost, causing the corresponding nodes to stay awake for the entire W_S, wasting energy. As shown in Figure 4.7a, the fraction σ_S of nodes for which this does *not* happen (i.e., those for which $\hat{t}_{on,S}$ is computed) decreases as N increases and is lower for the larger-diameter low power case, because both these factors increase the chance of individual packet losses. The average value of $\hat{t}_{on,S}$, shown in Figure 4.6b, is relevant for computing the overall $t_{on,S}$ according to Eq. (4.2).



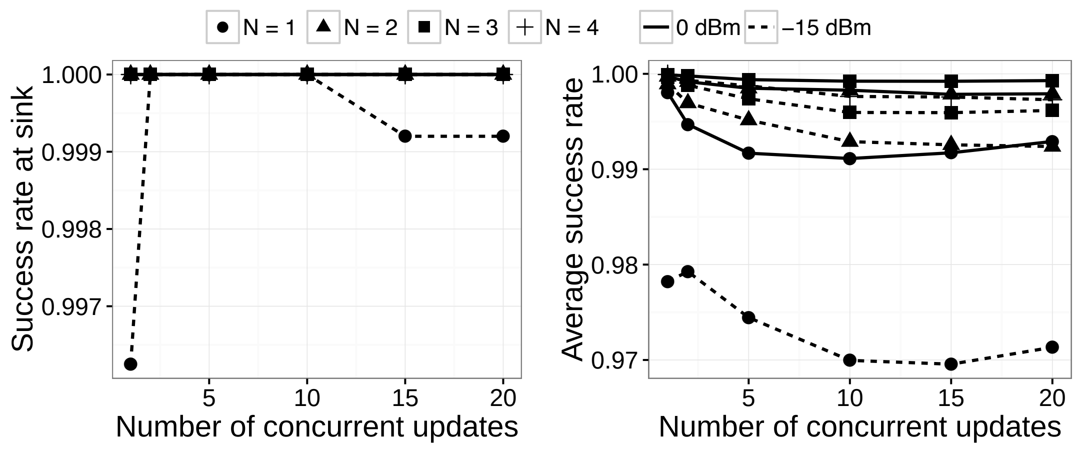
(a) **S** phase: σ_S. (b) **T** phase: σ_T.

Figure 4.7 – Fraction of nodes completing transmission within the slot duration W. Note the different scale on both axes.

The results above hold also for **A** phases, as their mechanics is exactly the same, apart from the different packet size, 8 B for **S** vs. 9 B for **A**. This yields a minimal difference on the radio-on time: on average, $\hat{t}_{on,A} = \hat{t}_{on,S} \times 1.020$.

T phase. Unlike **S** and **A**, the **T** phase behaves as in Glossy only if $u = 1$, i.e., there is only one sender in the network. Otherwise, if $u \geq 2$, multiple senders act as Glossy initiators, and compete during dissemination; as described in Section 4.3, the capture effect determines which packet among those concurrently broadcast is received, if any.

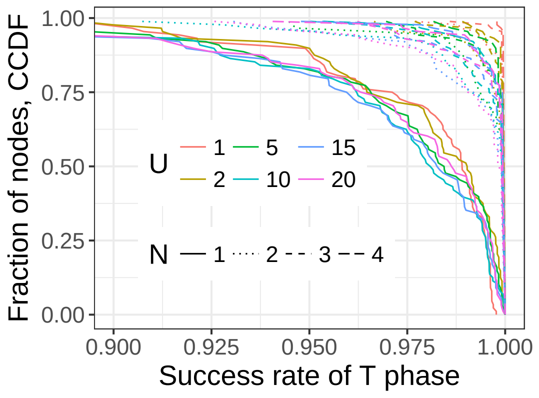
This mode of dissemination is inherently more unreliable than standard Glossy; however, the built-in redundancy inherited from the latter still yields a rather high probability that *at least one* update among those concurrently sent in the **T** slot is correctly received at all nodes. Figure 4.8b shows this probability of success, computed across the entire network, as a function of N and the number of concurrent updates u. The chart shows that, as the number of concurrent updates increases from $u = 1$, competition among senders causes transmissions to increasingly fail—up to a given point. As u increases, in fact, the probability that a node is close to one of the senders, and therefore receives its packet with high probability, increases. The chart also includes curves with $N = 1$ that, not surprisingly, provides the worst reliability especially in the low power case. This value was not included in the analysis of the **S** and **A** phases; these are used for time synchronization, and in this case Glossy requires $N \geq 2$.



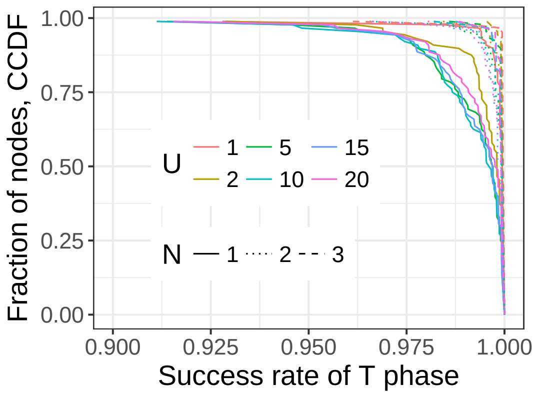
(a) To the sink, only.

(b) Across the entire network.

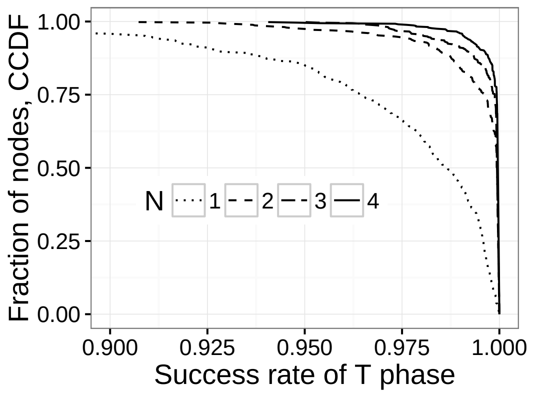
Figure 4.8 – **T** phase: average probability of successful transmission. Note the different y-axis scale.



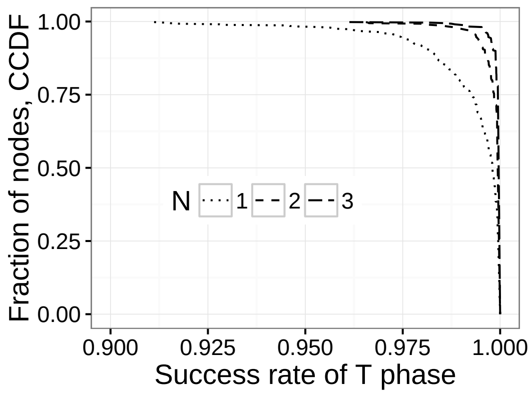
(a) Low power (-15 dBm).



(b) High power (0 dBm).



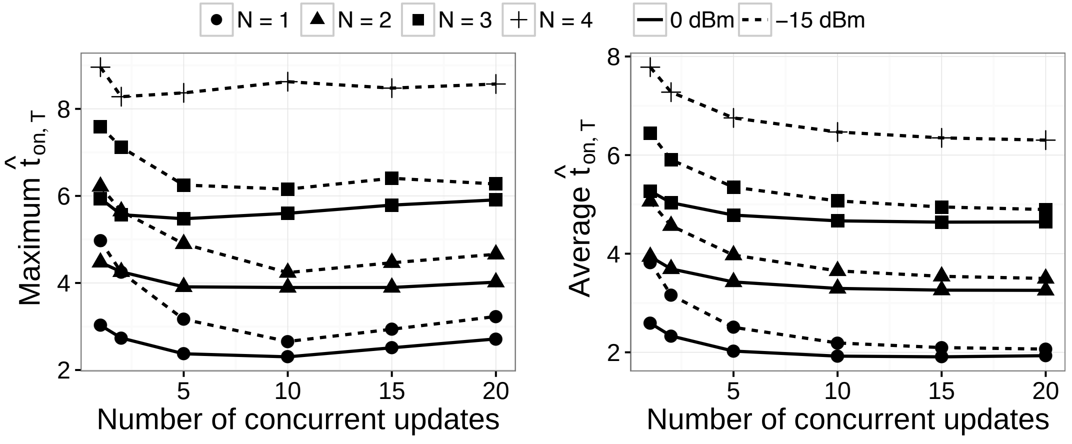
(c) Low power (-15 dBm), aggregated.



(d) High power (0 dBm), aggregated.

Figure 4.9 – CCDF for Figure 4.8b, individually for each value of U (top) and aggregated across all values of U (bottom).

On the other hand, the **T** phase in CRYSTAL is devoted to communication *towards the sink*. The probability of successful transmission to the sink in our experiments, shown in Figure 4.8a, is always at 100% except for the configuration with low power and $N = 1$. The reader may be led to think that this is the result of conveniently selecting the sink node. However, Figure 4.9 shows that a significant fraction of nodes similarly enjoy 100% reception rate when chosen as a sink. The chart shows the complementary cumulative distribution function (CCDF) of the probability of success per node (over all values of u), based on the same data shown in Figure 4.8b; effectively, this allows us to compute the probability of success for each node, in case it were chosen as the sink. For $N = 2$ and high power, in the worst case ($u = 15$) 43% of the nodes have perfect reception rates, and 53% have > 99.9%; for $N = 3$ at low power, 61% of the nodes have 100% reception. This confirms that our choice of sink is not biased.



(a) Maximum.

(b) Average.

Figure 4.10 – **T** phase: per-slot radio-on time, $\hat{t}_{on,T}$.

Finally, Figures 4.10 and 4.7b show the average values of $\hat{t}_{on,T}$ and σ_T for the **T** phase. Figure 4.10b shows that the value of $\hat{t}_{on,T}$ decreases rapidly as concurrent updates increase from $u = 1$; this is due to the increased likelihood of finding a closer sender, which therefore yields a shorter radio-on time. As u further increases, however, the density of senders increases and therefore the likelihood that their transmissions result in a packet loss. Since, as already mentioned, in Glossy a node remains awake (up to the end of W) until N transmissions of the same packets have been received, the increase in packet loss yields an increase in the average and, especially, maximum values of $\hat{t}_{on,T}$. This phenomenon is mirrored by the linear decrease in the fraction σ_T of nodes that complete dissemination within W, shown in Figure 4.7b.

4.6.3 Configuring Crystal

The results in Section 4.6.2 enable us to determine a reasonable configuration for CRYSTAL, i.e., one that strikes an appropriate balance between reliability and energy consumption. We later use this configuration, shown in Table 4.4, to further analyze the inner characteristics of CRYSTAL and its overall performance against our datasets. We distinguish the configuration based on the power, as we have seen that this is the parameter that most affects the configuration.

Slot configuration: High power. N is the critical parameter affecting reliability, as already discussed. For the **S** and **A** phases, Figure 4.5 shows that $N = 3$ provides 99.99% reliability. Larger values further approach perfect reliability but induce higher energy costs, due to higher radio-on time $\hat{t}_{on,S}$ (Figure 4.6) and fraction $1 - \sigma_S$ of nodes remaining awake for the entire slot (Figure 4.7a). Therefore, $N = 3$ is a good tradeoff for both **S** and **A**. As for **T**, similar considerations motivate $N_T = 2$. In principle, $N_T = 1$ could further reduce $\hat{t}_{on,T}$ and energy consumption; however, this would make sink selection more critical, as shown in Figure 4.9.

The slot duration W should be chosen for each phase by looking at the maximum radio-on time t_{on}. For the **S** and **A** phases, a value $N = 3$ implies $W \geq 7$ ms (Figure 4.6a). We use this value for W_A, while we use a higher $W_S = 10$ ms, given that the **S** phase is crucial for synchronization. In any case, unlike for the **T** and **A** phases, the impact of W_S on duty cycle is limited, given that *i)* the **S** phase occurs less frequently than **T** and **A**, for $u > 0$, and *ii)* $\sigma_S > 99\%$ for $N_S = 3$ (Figure 4.7a), therefore the impact of W_S (which comes into play only for the remaning 1% of the **S** phases) is negligible. A similar reasoning for the **T** phase, based on Figure 4.10a and the chosen $N_T = 2$, yields $W_T = 5$ ms.

Slot configuration: Low power. The configuration for low power is determined based on analogous reasoning. For the **S** and **A** phases, we choose $N = 4$. Although it does not yield reliability as good as its high power counterpart (Figure 4.5), higher values of N would significantly increase energy consumption due to the dissemination time $\hat{t}_{on}$. For the **T** phase, $N_T = 1$ is not an option, as it does not guarantee reliable transmission to the sink (Figure 4.8a). Our choice of $N_T = 3$ approaches the overall reliability of the high power counterpart (Figure 4.8b) and limits the impact of the sink placement.

Table 4.4 – CRYSTAL configuration parameters. W_x and G values are expressed in ms.

Power	N_S	N_T	N_A	W_S	W_T	W_A	G	R	Y	Z
High	3	2	3	10	5	7	0.15	2	2	4
Low	4	3	4	14	8	12	0.15	2	2	4

For **S** and **A**, a slot duration $W = 12$ ms is sufficient to guarantee that $W > \max(\hat{t}_{on})$ (Figure 4.6a). Therefore, we use this value for the **A** phase, and a higher value $W_S = 14$ ms, coherently with the high power case. We then set $W_T = 8$ ms based on similar reasonings (Figure 4.10a).

Slot configuration: Guards. Selecting W is not enough; we must determine also the duration of the guard G preceding it. We verified that $G = 150$ μs yields good reliability for all types, even with $E = 5$ minutes, as discussed in Section 4.6.4.

Terminating a CRYSTAL round. As discussed in Section 4.4, three additional parameters govern the behavior of CRYSTAL, specifically concerning the termination of the sequence of **TA** pairs in a single round: the number of **T** slots with no data R, causing the sink to piggyback the sleep command in its last acknowledgement; and the local termination parameters that have an effect only when a node has lost the final acknowledgement: Y and Z.

We use $R = 2$ as we determined experimentally that *i)* higher values do not bring additional benefits w.r.t. reliability, while they obviously greatly and negatively affect energy consumption *ii)* using $R = 1$ in general negatively affects reliability, although we show in Section 4.6.4 that the energy savings it enables can be exploited in some cases.

Finally, we use $Y = 2$ and $Z = 4$ as we determined experimentally that this yields good reliability, and hardly affects the duty cycle of CRYSTAL since almost always all the nodes receive all the acknowledgements.

4.6.4 Dissecting a Crystal Epoch

We now focus on the mechanics of CRYSTAL operation inside an epoch, i.e., the entire sequence of **S**, **T**, **A** phases necessary to disseminate a given number u of updates.

Setup. We use $E = 2$ s to accommodate long **TA** sequences, as we need to explore u values in the range 0–40. We use $R = 2$ and the slot durations in Table 4.4. For every point in the parameter space, we gathered data for 450 epochs.

Reliability. For both high and low power, the configuration in Table 4.4 yields 100% reliability, for all values of u we consider. In other words, despite the fact that at most one out of the u updates is delivered in a single **T** slot, and that occasional packet loss may occur even for $u = 1$, the network-wide transport mechanism we devised is very effective in ensuring reliability. Re-transmissions do occur however, as shown in Table 4.5; these are more frequent with high power, consistent with the larger collision domain. Their number is however very small; even for the maximum $u = 40$ considered, only 7 times in 450 epochs an extra **TA** pair was needed.

Table 4.5 – Average **TA** pairs required for each update, $\rho(u)$.

	1, 2, 5, 10	15	20	30	40
High power	1	1.0004	1.0003	1.0001	1.0004
Low power	1	1	1	1	1.0002

Impact of R on the reliability of TA chains. The results we just presented are derived with $R = 2$ silent pairs; for a **TA** chain to break prematurely (i.e., before u **TA** pairs have been executed) it must happen twice in a row that the packet transmitted in a **T** slot is not received at the sink, which therefore replies with an empty acknowledgement in the corresponding **A** slot. The probability of this event is extremely low in practice, yielding the aforementioned very high reliability.

Nevertheless, while it does not make sense to explore $R > 2$, the question remains about the impact of a lower value $R = 1$, which would enable energy savings. We focus again on $u = 40$ concurrent updates, as this defines a challenging test to reliability. Figure 4.11 shows the results for high power, with $R = 1$, over 1800 epochs. The chart shows that the majority

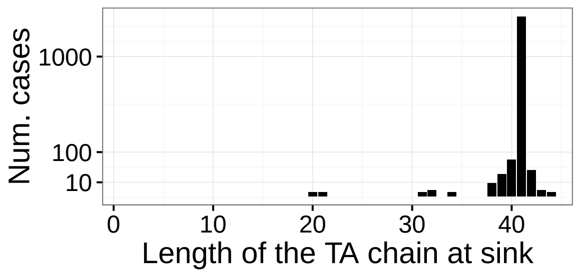


Figure 4.11 – Analyzing **TA** chains, $u = 40$, $R = 1$.

of chains terminate correctly upon the 41st **TA** pair; a few terminate slightly after, due to retransmissions. However, a few chains terminate earlier, therefore causing the loss of one or more updates; the probability of this happening increases towards the end of the chain. This is explained by the fact that the "stronger" a sender is the earlier its transmission succeeds; therefore, the end of the chain is usually populated by the "weakest" senders, for which the probability of packet loss is higher. Nevertheless, in a few cases, the **TA** chain breaks even halfway, around the 20th position, causing the loss of half the updates. Therefore, a value $R = 1$ cannot be used *for an entire chain*.

Opportunity for optimization: Dynamic R. On the other hand, the positive side of the previous argument is that the **TA** chain in Figure 4.11 does not break until the 20th position, with $R = 1$. This observation enables significant savings in energy consumption without prejudicing reliability.

Recall from Section 4.1.3 that the message suppression achieved by data prediction yields traffic profiles where the majority of epochs see at most one update. This holds for all our profiles except WATER (Figure 4.2c).

Therefore, optimizing the case with $u = 0$ is of paramount importance. In the $R = 2$ configuration we used thus far, CRYSTAL unfolds a transmission schedule with 5 slots, arranged in a **STATA** sequence. This was motivated by the reasoning that when using $R = 1$ (i.e., **STA**), the loss of a sent update would cause the sink to mistakenly believe that no update is disseminated, and send an empty acknowledgement in the **A** phase, effectively putting the entire network to sleep. However, Figure 4.11 shows that, in practice, *a chain is extremely unlikely to break on the first* **TA** *pair*. This is corroborated by Figure 4.8a showing that, at both powers, when any number u of updates are sent concurrently in a **T** slot, one *always* reaches the sink.

This leads to a strategy with a *dynamic* R value, using $R = 1$ for the first **TA** pair and, if a transmission occurs in it, switch to $R = 2$ for the rest of the epoch. In the cases where no update is actually transmitted, this dynamic assignment of R saves an entire **TA** pair, enabling substantial savings in energy consumption, as we further illustrate in Section 4.7.

Assessing the impact of the epoch duration. Until now, we executed experiments with an epoch $E = 2$ s, motivated by the need to reduce experiment time while exploring several

Table 4.6 – Validating the model: per-epoch radio-on time $T_{on}(u)$, in milliseconds.

	number of concurrent updates						
	0	1	2	5	10	15	20
	High power						
Maximum from CRYSTAL runs	30.6	41.51	51.82	81.54	130.18	180.13	231.33
Upper bound	34.75	47.05	59.35	96.25	157.75	219.25	280.75
Over-approximation (%)	13.57	13.34	14.52	18.04	21.18	21.71	21.36
Average from CRYSTAL runs	27.41	37.01	46.26	73.95	119.15	164.74	210.08
Model from benchmarks	26.82	36.32	45.38	72.25	117.52	163.96	211.69
Model from CRYSTAL runs	27.26	36.87	46.16	73.9	119.18	164.86	210.25
Error vs. benchmarks (%)	-2.15	-1.87	-1.91	-2.3	-1.37	-0.48	0.77
Error vs. runs (%)	-0.56	-0.38	-0.23	-0.07	0.03	0.07	0.08
	Low power						
Maximum from CRYSTAL runs	48.86	71.59	85.37	138.4	224.44	305.04	395.61
Upper bound	54.75	75.05	95.35	156.25	257.75	359.25	460.75
Over-approximation (%)	12.05	4.84	11.69	12.89	14.84	17.77	16.47
Average from CRYSTAL runs	41.81	58.23	73.62	118.14	189.88	259.08	327.64
Model from benchmarks	41.75	56.81	70.92	112.61	182.81	254.47	328.06
Model from CRYSTAL runs	41.65	56.76	71.43	114.79	185.79	254.7	323.1
Error vs. benchmarks (%)	-0.68	-0.65	-1.13	-2.08	-1.61	-0.03	1.63
Error vs. runs (%)	-0.92	-0.74	-0.42	-0.18	0.003	0.06	0.09

combination of parameters. However, this is a rather small duration if compared with the epoch typically adopted in sensing applications; Table 4.1 shows that, in our real-world datasets, E ranges between 30 s and 10 minutes. Are the results we derived thus far applicable to these long epochs?

In principle, the inner working of CRYSTAL is determined by the execution of the schedule, which is the same irrespective of the epoch length. Therefore, all of our estimates still hold unchanged; we verified this experimentally, although we omit the results due to space limitations.

A threat to this statement comes from time synchronization, to ensure the correct operation of the underlying Glossy layer. The **S** phase serves this purpose; however, if E is very large, the clock drift among nodes may grow to a point where the synchronization packet in the **S** slot is lost, causing inefficiencies due to node de-synchronization. However, this can be easily solved by choosing a larger guard for the **S** slot.

In our experiments, we verified that the value $G = 150\ \mu$s we use for all slot types is enough to reliably maintain time synchronization for epochs up to 5 minutes. This enables us to apply the model of Section 4.5 to the duty cycle computation by simply changing the value of the epoch. An investigation of longer epochs is beyond the scope of this work.

4.6.5 Validating the Model

We now focus on validating the accuracy of the CRYSTAL model we presented in Section 4.5. The upper bound of Eq. (4.1) can be determined solely by knowledge of the CRYSTAL configuration parameters shown in Table 4.4. Instead, the more accurate estimate of Eq. (4.3) requires also knowledge of the parameters $t_{on,S}$, $t_{on,T}$, $t_{on,A}$, and ρ.

We have two options for determining these parameters. The first one is to derive them from the slot-centric analysis in Section 4.6.2. The parameter values in this case are less accurate, as they are derived from specialized, slot-centric *benchmarks* instead of a full CRYSTAL run. These benchmarks are faster to gather than CRYSTAL runs and, as shown in Section 4.6.3, useful to choose the CRYSTAL configuration; it is therefore interesting to see what error they introduce in estimating T_{on}. The second option is instead to acquire these parameters directly from the full CRYSTAL experimental *runs* in Section 4.6.4. In this case, the parameters are obviously more accurate. These two model variants, derived from benchmarks and from runs, are then compared to the actual T_{on} in the CRYSTAL runs of Section 4.6.4.

The results are shown in Table 4.6, for both high and low power, as a function of the number u of concurrent updates. In the top part of the table, for both high and low power, we find confirmation that Eq. (4.1) is indeed an upper bound for T_{on}, by comparing against the maximum T_{on} in the CRYSTAL runs. The over-approximation introduced by neglecting the fact that $t_{on} < W$ grows with u, as expected. However, it always remains below 22%; therefore, the upper bound is still a valid design tool to get a first rough estimate of T_{on}.

In the rest of the table we assess (both variants of) the model by comparing its estimates against the average $T_{on}(u)$. We consider the error $\epsilon = \frac{real-model}{real}$, in percentage. As expected, the error of the benchmarks variant is higher, yet $|\epsilon| \leq 2.3\%$. The runs variants has much higher accuracy, as expected, always achieving $|\epsilon| < 1\%$.

Therefore, we can conclude that our model is a very good approximation of the real behavior of CRYSTAL, and we can exploit it next for computing the duty cycle over long datasets, without introducing a significant error.

4.7 Ultra-low Power Wireless Sensor Networks: A Reality

The premise of this work is that by combining the power of data prediction with a network stack efficiently supporting the traffic patterns it induces, it is possible to achieve ultra low-power WSNs. To verify the extent to which we achieve this goal we need to ascertain the duty cycle CRYSTAL achieves on the datasets we illustrated in Section 4.1.3. We divide our evaluation in two complementary parts. In the first one, we apply our model to the datasets, therefore estimating for all of them the duty cycle achievable over a long time span—impractical to reproduce in a testbed. In the second one, we instead measure *directly* the duty cycle in 2-hour experimental sessions where we compare the performance of CRYSTAL against the staple CTP-based stack, concerning not only duty cycle but also reliability.

In both cases, the duty cycle for each dataset is given by:

$$DC = \frac{\sum_{u=0}^{N} T_{on}(u)e(u)}{E\sum_{u=0}^{N} e(u)} \tag{4.4}$$

where $e(u)$ is the number of epochs in which u updates are transmitted concurrently, and $T_{on}(u)$ is the per-epoch radio-on time. The value of $e(u)$ is known for all datasets; see Table 4.2 for the INDOOR temperature one. As for $T_{on}(u)$, in the first part we use the model estimates, while in the second part we use directly the measured value. All results, computed and measured, are reported for the CRYSTAL variants with fixed and dynamic R, and for both high and low power.

Computing **the duty cycle from datasets**. Table 4.7 shows the results of applying the model to our datasets, using the more accurate estimates resulting from the parameters derived from experimental runs, as described in Section 4.6.5.

For the INDOOR dataset and high power, the upper bound estimate already places the duty cycle of CRYSTAL around our per-mille target. For the temperature and humidity datasets the upper bound for DC is slightly above 0.1% (i.e., 1‰) with a fixed R, and slightly below with a dynamic one. The light dataset has a slightly higher DC, as it has the highest number of concurrent updates among INDOOR datasets. However, the more accurate estimates provided by the model show that, by using a dynamic R, DC is reduced to slightly above 1‰ for light and as low as 0.7‰ for temperature and humidity. These best DC values translate to remarkable improvements w.r.t. CTP: up to 70x for temperature.

The DC of TUNNEL is ~0.1‰ lower than the INDOOR temperature and humidity datasets; this is reasonable, as these three datasets have a similar epoch and traffic pattern, as illustrated in Section 4.1.3. However, given the very small values at stake, this minuscule difference translates in an additional 8–10x improvement w.r.t. CTP.

The SOIL datasets also have a similar traffic pattern but a much longer (~20x) epoch. This brings the upper bound of DC for soil temperature to reach a stunning 0.06‰—i.e., 60 ppm. The more accurate model estimate brings this value down to 50 ppm, and dynamic R further reduces it to a tiny DC = 30 ppm. In this case, the table does not report any comparison against CTP. This would require finding the right configuration for epochs this long, and would anyway yield an exorbitant amount of control traffic w.r.t. the application one. However, our model reports an improvement of 955.6x for air temperature and 1592.67x for soil temperature.

The WATER dataset is somehow in the middle w.r.t. the other datasets. Its epoch is 5 minutes (half of SOIL, 10x more than INDOOR and TUNNEL) but, as discussed in Section 4.1.3, it exhibits a much higher frequency of concurrent updates. Therefore, the upper bound is DC = 0.8‰, while the actual one is as low as DC = 0.60‰. Interestingly, using a fixed or dynamic R bears a negligible impact on WATER. This is not surprising given that only 0.34% of the epochs have $u \leq 1$ (Figure 4.2c), i.e., 15 epochs out of the 4310 in the dataset.

Table 4.7 – CRYSTAL duty cycle for the datasets of Table 4.1, and comparison with plain CTP (i.e., without data prediction). The datasets with the "*" are artificially normalized to a 30-second epoch.

Application & Dataset		**Epoch**	**Duty Cycle (%), high power**					**Duty Cycle (%), low power**				
			upper bound		model (runs)		**vs.**	upper bound		model (runs)		**vs.**
			$R = 2$	$R = 1,2$	$R = 2$	$R = 1,2$	**CTP**	$R = 2$	$R = 1,2$	$R = 2$	$R = 1,2$	**CTP**
INDOOR	temperature	30 s	0.125	0.091	0.098	0.068	**70.26x**	0.197	0.142	0.15	0.104	**64.9x**
	humidity	30 s	0.127	0.095	0.1	0.071	**67.3x**	0.201	0.148	0.153	0.109	**61.93x**
	light	30 s	0.17	0.15	0.132	0.114	**41.91x**	0.272	0.238	0.204	0.176	**38.35x**
SOIL	air temperature	600 s	0.007	0.006	0.006	0.005	N/A	0.011	0.009	0.009	0.007	N/A
	soil temperature	600 s	0.006	0.004	0.005	0.003	N/A	0.009	0.006	0.007	0.004	N/A
TUNNEL	light	30 s	0.12	0.083	0.094	0.061	**78.33x**	0.189	0.128	0.144	0.093	**72.58x**
WATER	chlorine	300 s	0.081	0.081	0.061	0.061	N/A	0.133	0.133	0.094	0.094	N/A
SOIL*	air temperature	30 s	0.143	0.12	0.112	0.092	**51.93x**	0.227	0.19	0.172	0.141	**47.87x**
	soil temperature	30 s	0.117	0.078	0.092	0.057	**83.82x**	0.185	0.119	0.141	0.087	**77.59x**
WATER*	chlorine	30 s	0.809	0.809	0.608	0.608	**7.86x**	1.327	1.326	0.939	0.939	**7.19x**

Table 4.8 – Measuring reliability and duty cycle from Indriya experiments. The model values in italics are those that exhibit a (very small) change w.r.t. those in Table 4.7, when scaling down to a 2-hour traffic profile.

Network Stack	INDOOR, **temperature**				INDOOR, **light**				WATER*			
	yield (%)	**DC (%)**	**gain vs. CTP no DBP**	**gain vs. CTP DBP**	**yield (%)**	**DC (%)**	**gain vs. CTP no DBP**	**gain vs. CTP DBP**	**yield (%)**	**DC (%)**	**gain vs. CTP no DBP**	**gain vs. CTP DBP**
	High power											
CTP (no DBP)	98.33	4.778	1x	N/A	98.33	4.778	1x	N/A	98.33	4.778	1x	N/A
CTP (DBP, best configuration)	100	0.743	6.4x	1x	99.60	0.912	5.2x	1x	99.09	2.372	2x	1x
CRYSTAL (model, fixed *R*)	N/A	*0.097*	49.3x	7.7x	N/A	*0.131*	36.5x	7x	N/A	0.608	7.9x	3.9x
CRYSTAL (measured, fixed *R*)	100	0.098	49.0x	7.6x	100	0.131	36.4x	7x	100	0.606	7.9x	3.9x
CRYSTAL (model, dynamic *R*)	N/A	0.068	70.3x	10.9x	N/A	*0.113*	42.3x	8.1x	N/A	0.608	7.9x	3.9x
CRYSTAL (measured, dynamic *R*)	100	0.068	70.3x	10.9x	100	0.114	42x	8x	100	0.606	7.9x	3.9x
	Low power											
CTP (no DBP)	97.13	6.750	1x	N/A	97.13	6.750	1x	N/A	97.13	6.750	1x	N/A
CTP (DBP, best configuration)	100	0.825	8.2x	1x	99.60	1.087	6.2x	1x	98.99	2.997	2.3x	1x
CRYSTAL (model, fixed *R*)	N/A	*0.149*	45.3x	5.5x	N/A	*0.202*	33.4x	5.4x	N/A	0.939	7.2x	3.2x
CRYSTAL (measured, fixed *R*)	100	0.154	43.8x	5.3x	100	0.211	31.9x	5.1x	100	0.923	7.3x	3.2x
CRYSTAL (model, dynamic *R*)	N/A	*0.103*	65.5x	8x	N/A	*0.175*	38.6x	6.2x	N/A	0.939	7.2x	3.2x
CRYSTAL (measured, dynamic *R*)	100	0.106	63.7x	7.8x	100	0.173	39x	6.3x	100	0.923	7.3x	3.2x

In illustrating the results, we focused for simplicity on the high power ones; low power increases the network diameter, leading to a slightly higher *DC*. However, the right-hand side of Table 4.7 clearly shows that CRYSTAL achieves a *DC* around our 1‰ target and improves significantly over CTP.

This analysis allows us to put the duty cycle that can be achieved by CRYSTAL into the context of typical parameters for the real-world applications from which the datasets were obtained. In this respect, *a duty cycle below per-mille, and in some cases of parts-per-million, is several orders of magnitude smaller than what is achieved by the state of art.*

On the other hand, the epoch duration has a strong impact on the overall duty cycle. Therefore, the bottom of Table 4.7 offers an alternative view where the duty cycle of the various datasets is normalized to the lowest 30-second epoch of INDOOR and TUNNEL. Clearly, this is *purely speculative*, as in reality changing the epoch duration would actually change the probability of concurrent updates: the shorter the epoch, the lower this probability. In other words, we are *artificially defining a more challenging setup for* CRYSTAL.

Table 4.7 shows that, with this artificial normalization, the *DC* achievable for SOIL is in line with the other datasets that are not normalized; SOIL temperature actually achieves an improvement of 83.82x over CTP, the highest in our comparison. This is not the case for WATER, whose upper bound is 8‰, and best *DC* is 6‰. Nevertheless, considering the peculiar pattern shown in Figure 4.2, where essentially *every* epoch in WATER has concurrent updates, and the fact that even in these conditions CRYSTAL achieves a 7.86x improvement w.r.t. CTP, we argue this is actually a remarkable result.

***Measuring* the duty cycle (and reliability) from testbed experiments**. We now report about experiments that enable us to *measure* the duty cycle, therefore validating the findings we obtained by computing *DC* from the datasets with our model. In addition, this enables us to also evaluate the reliability of CRYSTAL, not captured by our model.

For these experiments, we "scale down" the traffic profiles of our datasets to reproduce their trends over a much shorter interval. The latter is determined by the maximum length of Indriya experiments (2 hours, i.e., 240 epochs of 30 s) minus a "burn-in" time for the CTP topology to stabilize, yielding experiments that are 200 epochs long.

Given a traffic profile, scaling is performed by simply multiplying by 200 the fraction of epochs with a given number u of updates. For instance, for INDOOR temperature in Table 4.2, the number of epochs with $u = 0$ updates becomes $0.8211 \times 200 = 164$. This obviously removes some values of u for which very few epochs exist (e.g., $u = 13$ in Table 4.2) but faithfully preserves the dominant trends of the profile. The latter is then reproduced by choosing, at each epoch, u nodes at random to serve as the update senders.

We focus only on three representative datasets, based on the estimates in Table 4.7: *i)* INDOOR temperature, for which CRYSTAL achieves good performance; *ii)* INDOOR light, the most challenging among the INDOOR datasets; *iii)* WATER normalized to a 30 s epoch because, albeit artificially generated, serves as a very challenging case for CRYSTAL.

We repeat the experiments for each dataset with both high and low power. For each combination, we compare the estimate given by our model against the sum of the T_{on} values we measure in each epoch divided by 200, the total number of epochs. Moreover, we measure reliability as the data yield at the sink. We repeat all experiments both with a fixed and dynamic R. Finally, for each combination we also compare against plain CTP (no prediction) as in Table 4.7, and also with the best configuration (for the combination of power and dataset) for CTP with DBP data prediction.

Table 4.8 shows the results. In all of our experiments CRYSTAL achieved 100% reliability. Plain CTP achieved the lowest reliability, as expected due to the higher traffic. However, even the CTP/DBP combination, despite the much sparser traffic induced by data prediction, achieved 100% only in 1/6 of the cases, namely, INDOOR temperature at high power.

The duty cycle values in Table 4.8 agree closely with those in Table 4.7. The scaled down profiles induce tiny changes across the two tables, marked in italics in Table 4.8. The measured *DC* is always very close to the model estimates, therefore corroborating the results derived in Table 4.7, including the improvement over plain CTP. Moreover, Table 4.8 shows that, in all combinations, CRYSTAL significantly improves (up to 10.9x) also against the best configuration of CTP/DBP; even in the challenging WATER* dataset, CRYSTAL achieves a 3.2x improvement, confirming that CRYSTAL offers a significant advancement w.r.t. the state of the art.

4.8 Related Work

Data prediction [90, 49, 3] is applicable to a large number of real-world applications, in which it greatly abates data traffic. Prior work [89] showed the limitations of a staple network stack in taking full advantage of data prediction, opening the way for CRYSTAL to remove limitations from idle listening and collection topology maintenance.

Routing Optimization. Accurate link estimation with broadcast beacons is a major overhead source in CTP. Some protocols mitigate this by using overhearing to estimate link quality [60, 87] or queue status [68] but, by relying on periodic traffic to update estimates, they compromise their accuracy and therefore usefulness in the extreme, low-traffic scenarios CRYSTAL targets. CRYSTAL also removes the low-power listening MAC, thus abating idle listening costs.

Ultra low-power data collection. A number of protocols achieve extremely low duty cycles by crossing the line between MAC and routing. Dozer [13] and Koala [72] do so by accepting extremely long latencies, minutes or days respectively, allowing nodes to sleep as long as possible between communication events. DISSense [23] reduces latency with a synchronous wake-up mechanism somewhat similar to CRYSTAL, but it only reaches per-mille duty cycle for reporting intervals of 60 minutes. CRYSTAL, instead, achieves per-mille duty cycle even with a short 30 s reporting interval, making it applicable to control-loop applications.

Concurrent Transmissions. Glossy [38] pioneered work on exploiting constructive interference to achieve millisecond level network-wide flooding from a single sender, and is at the core of other protocols supporting multiple senders.

In LWB [39] and Choco [95], Glossy floods are used to collect node requests for transmission slots and to distribute a global transmission schedule computed at the sink. Changes in traffic require regeneration and dissemination of the schedule. These solutions are effective for periodic data streams where a schedule is used for several transmissions, but inapplicable to the unpredictable and aperiodic traffic induced by data prediction, which would require a continuous rescheduling for each new transmission. Instead, CRYSTAL exploits the capture effect to build a network-wide transport protocol that involves sink-based acknowledgements, effectively allowing transmissions to compete and "self-schedule" based on the contingent communication needs.

Chaos [59] also supports multiple senders but allows concurrent, unscheduled transmissions by relying on the capture effect. Packets from different senders are merged before forwarding to compute, over several iterations, a network-level aggregate (e.g., the maximum value transmitted). CRYSTAL also relies on the capture effect, but with the opposite goal of delivering reliably to the sink *all* the data transmitted by nodes, bringing a different set of challenges and solutions.

Studies [101, 75] show that the Glossy flooding reliability degrades as the density and number of nodes increases. This problem is mitigated by limiting the number of concurrent transmitters [101, 14, 106, 107], or improving synchronization between the transmitters by compensating for radio processing and signal propagation delays [102]. These enhancements are orthogonal to CRYSTAL and can improve its performance.

Concurrent transmissions have also been explored for bulk data transfer [26, 28, 30]. The use of channel and spatial diversity allows them to push larger amounts of data, however, their complexity and higher power consumption make them unsuitable for the ultra-low data rates of data prediction.

4.9 Conclusions

We demonstrated that the synergy between the high message suppression rates offered by data prediction and the lightweight, reliable, energy-efficient, fast communication enabled by synchronous transmissions bring the performance of WSNs to unprecedented levels. Our system, CRYSTAL, can deliver multiple concurrent packets—the model updates generated infrequently, unpredictably, and aperiodically by data prediction—very fast and with perfect reliability, making it amenable to applications where WSNs are part of a control loop. Further, CRYSTAL achieves per-mille duty cycle, improving on the staple WSN stack (CTP + BoX-MAC) up to a factor of 80x. This remarkable reduction in energy consumption is achieved by neither compromising on the data reporting period nor using specialized hardware. On the contrary, the very small duty cycle achieved by CRYSTAL may offer a large leap towards energy-neutrality, e.g., in combination with energy harvesting techniques hitherto considered insufficient to power WSN nodes under real-world profiles like those we considered in this chapter.

We showed that CRYSTAL almost always provides perfect reliability in moderate noise conditions of a typical office environment characterised by fluctuating Wi-Fi interference. However, we noticed that during rare and unpredictable peaks of external interference some packets from the nodes under the highest noise exposure were lost. This motivated a more thorough study of CRYSTAL under high but controllable noise that is presented in the next chapter.

5 Interference-resilient Aperiodic Data Collection in Wireless Sensor Networks with Synchronous Transmissions

As we showed in the previous chapter, CRYSTAL, our protocol based on synchronous transmissions, provides ultra-low power reliable data collection service for WSNs. Although we tested it at day and night, the unpredictable external interference present in office buildings during the day resulted in inconsistent results that are difficult to interpret. This chapter is thus dedicated to a thorough study of synchronous transmissions and CRYSTAL in particular under strong and reproducible interference.

Motivation and goals. Although synchronous transmissions are commonly considered highly resilient to interference due to inherent redundancy (e.g., multiple transmissions propagating along multiple paths) and reliance on PHY-level properties (i.e., constructive interference and capture effect), to the best of our knowledge, the interference resilience of synchronous transmissions has never been *systematically* evaluated, and the extent to which it holds is unknown. A commonly-accepted methodology to test against interference is to run experiments in office testbeds both at night, when WiFi interference is lower, and day, where instead it is higher due to human presence. This approach is adopted for the evaluation of several well-known systems [39, 59, 106, 53] including Glossy [38], which was the first to exploit synchronous transmissions and is often used as the core dissemination layer in other systems.

Unfortunately, these conditions are a far cry from those typically found, e.g., in industrial settings [76, 79], where noise levels are much higher than those induced by WiFi, and monitoring tasks sometimes require nodes to be placed next to the noise-generating equipment. Worse, the interference patterns in testbeds, especially during the day, often fluctuate randomly. For instance, in Indriya [27] we sometimes observed nodes exposed to an average noise of -60 dBm and higher, which would appear on any of the 16 channels offered by the IEEE 802.15.4 radios. In principle, this random and high noise allows one to reproduce realistic conditions. However, it prevents systematic testing; the performance of a protocol is jeopardized by random high noise whose source remains unknown and whose pattern cannot be reproduced. Considering the high reliability of synchronous transmissions, a single "unlucky" run is often enough to put a dent in reliability and, at the same time, this run cannot be reproduced and analyzed to devise effective countermeasures.

Our goal in this chapter is therefore *to provide, for the first time, the systematic analysis of the performance of a protocol based on synchronous transmissions under interference.*

Choice of protocol. Many synchronous transmission protocols exist [39, 59, 106, 53] that build upon Glossy [38] by relying on a custom schedule of its network-wide floods to achieve different goals. Therefore, analyzing only the Glossy protocol would be limiting, as the different protocols exploit it with different performance and application targets. At the same time, a comparative evaluation across all existing Glossy-based protocols is prohibitive due to their number and the many dimensions at stake.

We focus on the CRYSTAL protocol, described in the previous chapter. Apart from being a recent proposal with remarkable lifetime gains, our choice is motivated by these characteristics:

- *Heavy reliance on the capture effect.* The core operation of CRYSTAL relies on concurrent Glossy floods from different senders, each with unique packets, to "compete". Thanks to the capture effect and the redundancy of Glossy floods, the packet of *one* sender is received at the sink with high probability; a network-wide acknowledgment informs other senders that they must retransmit, while also providing for the (unlikely) case in which no data is received at the sink.
- *Challenging target traffic pattern.* Originally driven by the needs of higher-level data prediction schemes, CRYSTAL targets *aperiodic* and *sparse* traffic. This traffic pattern is particularly challenging because *i)* with fewer packets than periodic approaches, the loss of a single one has a much larger impact on reliability, and *ii)* packet transmissions are separated by long periods of inactivity, during which energy consumption should be minimized.

These characteristics define very challenging requirements w.r.t. interference, as it may undermine the very operation of CRYSTAL by significantly lowering the benefits of capture effect, jeopardizing reception at the sink. Moreover, interference exacerbates the clash between reliability and energy consumption; the extra protocol effort necessary to ensure the former increases the latter.

Experimental Methodology and Comparison Baselines. The results reported in this chapter are derived from real-world experiments in a 49-node testbed available at our office premises, allowing exclusive and continuous access to the infrastructure. Our testbed exhibits its own *natural* interference, mostly due to WiFi. Nevertheless, the aforementioned goal of a systematic analysis of interference demands the ability to *generate* and reproduce interference patterns. To this end, we exploit JamLab [9], summarized in 5.1, to create realistic and reproducible noise patterns, both more disruptive and extensive than natural ones. We focus on interference emulating the behavior of WiFi devices and microwave ovens, as these show the greatest network disruption. Details of the experimental setup and the associated noise levels appear in 5.1.

We evaluate CRYSTAL in terms of packet delivery rate (*PDR*) and duty cycle (*DC*) as indicators of reliability and energy consumption, respectively. Moreover, as CRYSTAL relies on unmodified

Glossy, we indirectly evaluate the latter with the same experiments. Further, we observe that none of the proposals to cope with interference has found its way into the mainstream. Therefore, we compare against RPL [105] and ORPL [33], described previously in this thesis, as they are readily available and have been used as baselines in analogous works [74, 69, 108].

Results and main contributions. We show in 5.3 that when only natural interference is present, all protocols perform satisfactorily, but only Glossy and CRYSTAL achieve near-perfect reliability—and with a much lower duty cycle that the others. On the other hand, with JamLab interference, RPL performance degrades to unacceptable levels of *PDR* =85% when a single node acts as a WiFi interferer, while ORPL and synchronous transmissions protocols still achieve perfect reliability. Nevertheless, when WiFi interference covers the entire testbed, ORPL reliability also degrades. Interestingly, however, the roles are reversed when a JamLab-emulated microwave oven is placed 1m from the sink. Here, ORPL achieves nearly perfect reliability, while synchronous transmissions fall below 80%.

These results motivated us to explore two techniques to improve interference resilience in synchronous transmissions, described in 5.4. The first allows nodes to *escape* from interference by performing each transmission-acknowledgement pair—a core CRYSTAL constituent—on different channels, following a network-wide hopping sequence. Second, noise detection at all nodes enables them to schedule additional opportunities for transmission in a decentralized way, increasing the chances that a packet is delivered. This technique effectively *fights* interference; however, it may be detrimental when traffic is sparse—a scenario targeted by CRYSTAL—by keeping nodes more active than needed.

The experimental results in 5.5 show that the combination of these two techniques achieves near-perfect reliability extremely challenging scenarios with both microwave oven and WiFi interference are simultaneously present. Overall, we confirm that synchronous transmissions in general, and the original CRYSTAL in particular, can tolerate the moderate levels of interference commonly found in office environments. However, they can also be modified with relative ease to sustain much stronger interference patterns with extremely low energy consumption.

Finally, 5.6 concisely surveys related work, and 5.7 ends the chapter with brief concluding remarks.

5.1 Radio Interference in the Testbed

The experiments we report were performed in our local testbed, composed of 49 TMote Sky nodes deployed as shown in Figure 5.1 in a 60×40 m^2 office area, subject to WiFi interference. Similar to other reports [69] the latter *i)* is more intense during the day and less at night and during the weekends, and *ii)* varies depending on the channel considered.

Moreover, in addition to this *natural* interference, we also leverage controlled JamLab *generated* interference, enabling repeatable experiments. A brief description of JamLab is presented next.

5.1.1 Generating Interference: JamLab

JamLab [9] uses the same mote-class nodes available in a testbed, to faithfully emulate various types of interference relevant to IEEE 802.15.4, including Bluetooth, WiFi, and microwave ovens. These have very different characteristics. Bluetooth interferes with all IEEE 802.15.4 channels, as it uses a channel hopping scheme. WiFi spans 4 IEEE 802.15.4 channels with interference that is significantly stronger than Bluetooth, but also based on the type of data traffic. Microwave ovens interfere with 7 channels, and induce very strong, continuous interference for 10ms, alternated with inactive periods of similar duration. In the rest of the chapter we focus only on WiFi and microwave ovens, as they yield the strongest interference. Similarly, among the WiFi patterns offered by JamLab, we consider JL_WIFI4, which emulates the combination of a file transfer and radio streaming. Finally, to put ourselves in the worst-case scenario, we set the TX power of the interferers to the maximum, 0dBm, and consider only modulated carrier interference as it has a much stronger effect on the radio communication.

One JamLab limitation comes by noting that real interference sources typically interfere with many contiguous IEEE 802.15.4 channels at the same time, e.g., 4 for WiFi. In contrast, a JamLab node generates noise on a single channel. The majority of the proposed protocols, including the synchronous transmissions ones and the mainstream ones considered in this chapter, operate on a single channel; therefore this limitation does not affect the experiments in 5.3. However, in 5.5 we explore channel hopping and address this JamLab limitation with a channel mapping strategy.

Another limitation is that the maximum output 0dBm power of motes is much smaller than the typical one of other interference sources (e.g., 25 and 60dBm for WiFi and microwave ovens, respectively). As suggested in JamLab [9], we therefore use multiple motes, strategically placed at different points in our testbed (Figure 5.1).

5.1.2 Testbed Interference Scenarios

Overall, we define four types of interference described in Table 5.1. The choice of channels in the natural interference types derives from an extensive, cross-channel measurement campaign, which identified the best (26) and worst (18) channels during night and day, respectively. The generated interference is created at night on channel 26 (i.e., under natural T-LOW interference) and uses interfering JamLab nodes configured with the maximum TX power of 0 dBm. Our evaluation uses varying numbers of JamLab interferers for each type as well as combines different types in the same experiments, to obtain challenging, realistic setups. Node 1 acts as the sink in all experiments.

Figure 5.2 quantitatively compares the various types of interference. The natural T-LOW in Figure 5.2a exhibits an average noise of −92 dBm, rather stable and uniform across the network; a notable exception is node 45, physically located near complex cabling, which is subject to an average noise of −78 dBm affecting all channels. The interference environment in natural T-HIGH is drastically different (Figure 5.2b). The average noise is −88 dBm, but several nodes

Table 5.1 – Types of interference.

Type of interference		**Description**
Natural	T-LOW	testbed at night/weekends, channel 26
	T-HIGH	testbed during the day, channel 18
Generated	J-WIFI	JamLab WiFi interference (JL_WIFI4)
	J-MWO	JamLab microwave oven interference

are exposed to significantly higher noise levels, which occasionally reach values as high as −40 dBm. Table 5.2 shows that this higher noise also affects the network topology, e.g., causing a 10% increase in the average hopcount for the case with 49 nodes.

The interference generated via JamLab yields stronger noise than naturally present in the testbed. Figure 5.2c shows the J-WIFI interference generated by node 7 alone, the closest (1m) to the sink. Figure 5.2d shows instead the case with 6 J-WIFI interferers (including node 7) whose placement (Figure 5.1) is chosen to cover the entire testbed. Compared with Figure 5.2b, whose interference is mostly due to WiFi, the network in Figure 5.2d with J-WIFI is subjected to a slightly higher average noise of −85 dBm, and many more nodes are exposed to noise with significantly higher variance. This noise significantly affects also the network topology, increasing the average hopcount by 20% w.r.t. T-HIGH (Table 5.2).

Figure 5.2e shows the interference generated by J-MWO when the JamLab interferer is placed on node 7. About one quarter of the network (obviously including the sink) is severely affected by interference, with an average noise from −80 to −65 dBm, far higher than the previous scenarios. In the rest of the chapter, we experiment with alternate placements of the J-MWO

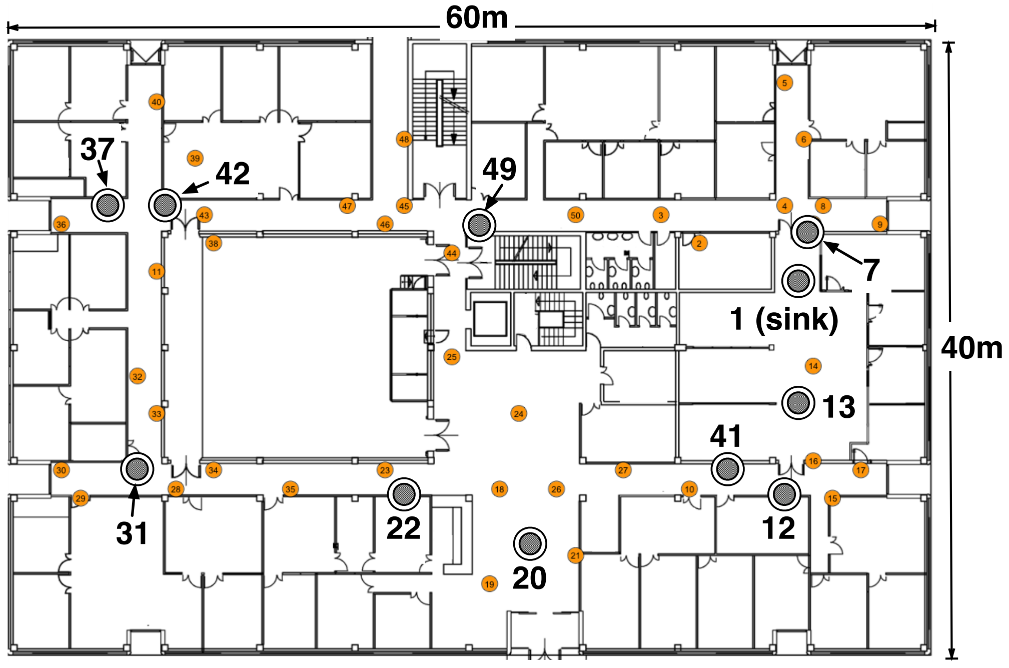
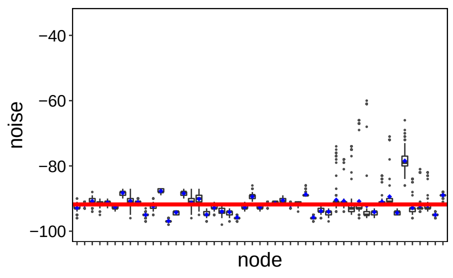
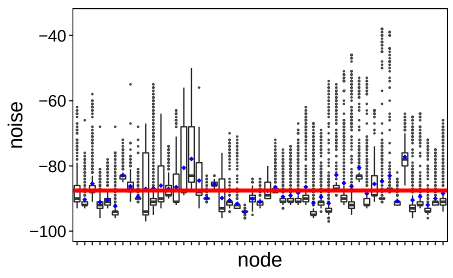


Figure 5.1 – Position of the JamLab interferers in the testbed.

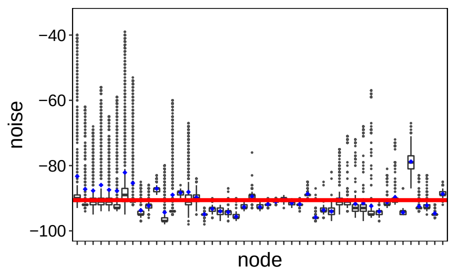
interferer at various distances from the sink. This clearly affects differently the sink, but also has different global effects on the network topology (Table 5.2). Moreover, we experiment with 2 J-MWO interferers (node 7 and 42) also in combination with J-WIFI. Figure 5.2f shows the noise generated by a configuration with 2 J-MWO and 4 J-WIFI interferers, which is significantly higher than all the previous scenarios and affects significantly the network (Table 5.2). This scenario is the most challenging we consider in this chapter, and can be considered akin to extreme settings such as industrial ones.



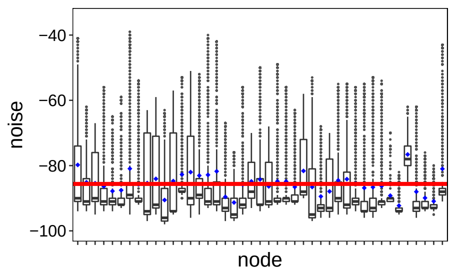
(a) T-LOW: channel 26, night.



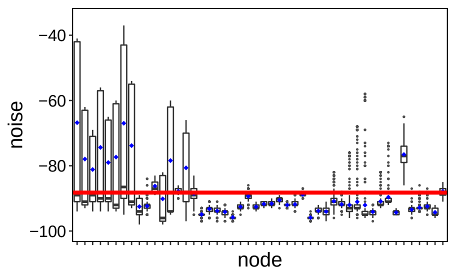
(b) T-HIGH: channel 18, day.



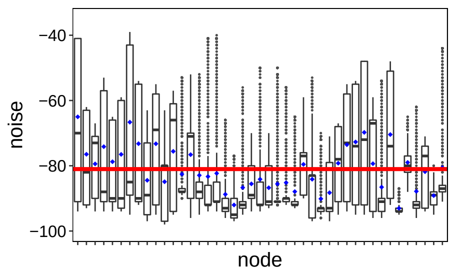
(c) J-WIFI: node 7.



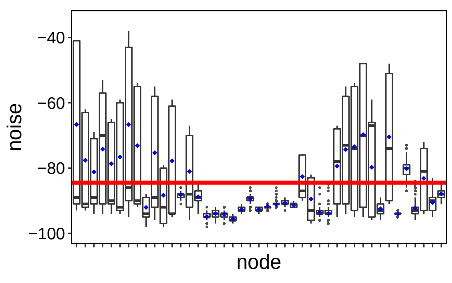
(d) J-WIFI: node 7,12,20,31,37,49.



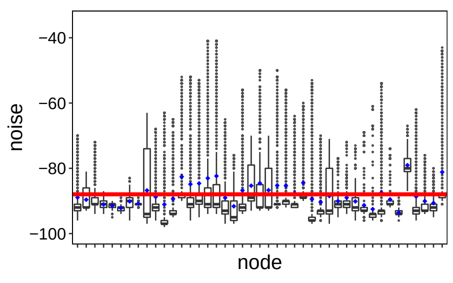
(e) J-MWO: node 7.



(f) J-WIFI (node 12,20,31,49) and J-MWO (node 7,42) combined.



(g) J-MWO: node 7 and 42.



(h) J-WIFI: node 12,20,31,49

Figure 5.2 – Noise levels for the interference types in Table 5.1.

Table 5.2 – Impact of interference on network topology.

TX power	ch.	type	removed or interfering nodes	nodes in network	# Glossy hops avg	max
0 dBm	26	T-LOW	—	49	2.6	5.1
	18	T-HIGH	—	49	2.9	5.6
	26	J-WIFI	7	48	2.9	5.3
	26	J-WIFI	7,12,20,31,37,49	43	3.5	6.3
	26	J-MWO	7	48	3.3	5.3
	26	J-MWO	13	48	3.0	5.0
	26	J-MWO	42	48	2.5	5.5
	26	J-MWO	7,42	43	3.6	7.0
	26	J-WIFI	12,20,31,49	43	3.3	5.9
	26	J-MWO J-WIFI	7,42 12,20,31,49	43	4.2	7.6
−7dBm	26	T-LOW	—	49	3.0	5.8
	18	T-HIGH	—	49	3.4	6.7
	26	T-LOW	7,12,20,31,42,49	43	3.1	6.3
	18	T-HIGH	7,12,20,31,42,49	43	3.9	7.8
	26	J-MWO J-WIFI	7,42 12,20,31,49	43	6.4	11.6

5.2 Protocols and Configurations

We briefly characterise the mainstream protocols we use as a baseline for comparison against the synchronous transmissions protocols we consider, Glossy and CRYSTAL, along with the configuration for all protocols used in the experimental campaign. All protocols considered in this chapter run atop Contiki [24]. At the application layer we use a payload of two bytes which translates into eight-byte packets for CRYSTAL, but much longer packets for RPL and ORPL because of their protocol overhead.

5.2.1 Mainstream Protocol Descriptions

RPL [105], the Routing Protocol for Low-power Lossy Networks, is an IETF standard. RPL can be seen as an evolution of CTP [45], as instead of a tree it maintains a Destination-Oriented Directed Acyclic Graph (DODAG), i.e., a directed graph without cycles, rooted at the sink. Therefore, each node maintains multiple parents towards the root; a preferred one is used for actual packet forwarding towards the root, while the others are kept as backup routes. This feature is beneficial for noise resilience since the forwarders eventually switch to a backup parent if the connectivity to the previous one was hampered by interference, however this process is slow and does not help in avoiding short-term noise variations.

ORPL [33] is an opportunistic routing protocol inheriting many design choices from RPL but replacing unicast forwarding with an anycast technique. Instead of relaying a packet to the parent, the forwarder broadcasts it; any neighbor closer to the sink is free to catch the packet, acknowledge it, and forward it further following the same technique. This improves interference-resistance as the packets may follow various paths, dynamically avoiding noisy areas. This mechanism reacts to interference much faster than the parent switching of RPL, however, the forwarding is still based on the routing gradient, possibly causing some data packets to get trapped. If all the potential forwarders are hampered by high noise, the packet will not be forwarded around the interferer, even if a noise-free path exists.

5.2.2 Mainstream Protocol Configurations

MAC wake-up interval. Both protocols rely on ContikiMAC [31] for medium access control and duty cycling; the value of the wake-up interval is therefore a key parameter affecting performance. We initially chose a value of 8Hz, as this is the default in ContikiMAC and therefore commonly used in the literature. Although our goal in this chapter is *not* to systematically explore the best configuration of these mainstream protocols, we experimented also with values of 1, 2, 4 Hz, as they may provide better performance under interference. We observed this to be the case for ORPL, which performs best at 2Hz. Therefore, hereafter we report only about experiments with a wake-up interval of 2Hz and 8Hz; in general, these also strike a different balance between reliability and duty cycle, and are therefore interesting to compare. The other configurations always perform worse, and are omitted due to space limitations.

Choosing the right CCA. Clear Channel Assessment (CCA) is a mechanism used in CSMA-based link layers to deter a packet transmission if the medium is observed busy. Its configuration significantly affects the interference resilience of the stack.

The CC2420 radio chip offers three CCA modes [96], in which the CCA reports a busy medium upon detecting either *1)* energy above the energy detection threshold; *2)* valid IEEE 802.15.4 data, regardless of the energy detection threshold; *3)* valid IEEE 802.15.4 data *or* energy above threshold as in mode 1.

We verified empirically that energy threshold of -90dBm used as default in ContikiMAC yields unacceptable performance; even with natural T-HIGH interference, the baseline protocols achieve $PDR < 30\%$. We are not aware of established guidelines about how to set the energy threshold in the presence of interference. Therefore, we tested the mainstream protocols with several values ranging from −60 to −90 dBm under natural T-HIGH and generated interference. The value of −77 dBm yielded the best performance and became our choice. In fact, this value is default for CC2420.

As for the CCA mode, the protocols we considered use the default mode 3. However, in the case of interference generated by JamLab nodes, the question arises whether the noise patterns these nodes emit can be possibly detected by other nodes as legitimate IEEE 802.15.4 data, instead of interference. We performed dedicated experiments comparing the performance obtained with CCA modes 1 and 3, observing essentially the same performance. Therefore, hereafter we used the default CCA mode 3.

Retransmissions. The protocols we consider employ different strategies concerning layer 2 transmission attempts. When an acknowledgment is not received, a maximum of 7 retransmissions is allowed by RPL, and 4 by ORPL. However, a retransmission can be triggered also by a CCA check detecting a busy channel, in which case a few subtleties of the Contiki operating system come into play. Contiki v.3.0, used by RPL, considers 5 busy CCAs as equivalent to a failed transmission attempt, while the two events are completely unrelated in Contiki v.2.7. The latter is used by ORPL, which allows unlimited number of CCA checks till the channel is free.

We did not modify these settings, as changing these *default* parameters may have unexpected and undesired effects on the protocols, whose analysis is outside the scope of this chapter. However, we mention them here because they are useful in interpreting the results we present in the next section, e.g., the superior performance of ORPL in the presence of strong interference next to the sink.

5.2.3 Crystal configurations

In essence, CRYSTAL builds a reliability layer atop Glossy, which strikes different tradeoffs w.r.t. energy consumption by exploiting the interplay between the two layers. As in Glossy, the number N of retransmissions in each flood is key, however, in CRYSTAL this value can be set independently for each of the **S**, **T**, **A** phases. This also holds for another key Glossy parameter, the maximum slot duration W.

Table 5.3 shows the configurations we use throughout this chapter, adapted from the originals along two dimensions. First, our testbed has a larger diameter than Indriya used in the previous chapter. This forced us to use larger values for the intervals W_T and W_A to allow Glossy floods to complete; we determined the optimal value using the methodology introduced in the previous chapter. Second, we experiment with combinations of N_T and N_A values to explore the impact of the **T** phase w.r.t. interference. The values of the remaining parameters W_S, G, R, Z and Y are unchanged.

Finally, we use two power settings, high (0dBm) and low (-7dBm), the former serving as the default throughout the chapter.

Table 5.3 – CRYSTAL configurations used in this chapter. The values of W_x and G are in milliseconds.

Power	N_S	N_T	N_A	W_S	W_T	W_A	G	R	Z	Y
High	3	2	3	10	6	8	0.15	2	4	2
	3	3	3	10	8	8				
Low	3	3	3	12	10	10				
	4	4	4	14	12	12				

5.3 Comparing Against the Mainstream

We compare the mainstream protocols from 5.2 against CRYSTAL, and indirectly Glossy, when all are exposed to the same natural or generated interference. Aside from the intrinsic and novel value of this experimental comparison, this section also serves a stepping stone toward tolerating stronger interference, discussed in 5.4.

5.3.1 Experimental Setup

We configure and analyze CRYSTAL and the baseline protocols in the interference scenarios described in 5.1. We experiment with a number U of concurrent senders between 0 and 48, aligning with the aperiodic, sparse data described in the previous chapter. $U = 0$ represents the absence of traffic while $U = 48$ offers a stress case when all nodes except the sink are senders. Further, note that the reliability of the underlying Glossy layer can be derived from the same CRYSTAL experiments, as the reliability of the **T** phase when $U = 1$ (Table 5.7).

In CRYSTAL, the U senders all attempt transmission of their packet at exactly the same time, i.e., in the first **T** phase of the epoch, for which we choose a duration $E = 2$s. Thanks to the very low latency of synchronous transmissions, this value is enough for dissemination to complete in all our tests except for $U > 43$ under high interference, for which we use a larger value and rescale DC accordingly. On the other hand, baseline protocols have much higher latency, especially under interference. Therefore, we chose a larger epoch $E = 10$s for them, which represents solely the period according to which packets are generated. In reporting duty cycle, we re-scale the values measured for CRYSTAL to 10s, enabling direct comparison between the two protocol classes. As shown in the previous chapter, such rescaling is legitimate because the inner structure of the active part of an epoch does not depend on the epoch duration, and the guard times need not be larger in longer epochs because of the clock skew compensation implemented in CRYSTAL.

Finally, experimental results of both synchronous transmissions and baseline protocols are based on several one-hour runs. For baseline protocols, this interval is preceded by a 30-minute period since bootstrap, allowing the network topology to stabilize. For CRYSTAL, the number of packets sent *per configuration* varied from 1800 to 500k, but typically was around 5k–50k.

5.3.2 Natural Interference: T-LOW

We first consider the most favorable conditions, T-LOW, described in 5.1 as the lowest possible interference. T-LOW is similar to most experiments throughout the literature, and thus offers the baseline for comparison when we increase interference.

The performance of the mainstream protocols in the T-LOW scenario (left-hand side of Table 5.4) is in line with several experiments in the literature [33, 44]. As expected, the MAC wake-up interval bears a significant effect: RPL performs best at 8Hz, while ORPL achieves

Table 5.4 – Natural interference: Mainstream protocols, $U = 1$.

		T-LOW		T-HIGH	
protocol	wake-up (Hz)	*PDR* (%)	*DC* (%)	*PDR* (%)	*DC* (%)
RPL	2	92.0	1.36	65.2	1.62
RPL	8	93.7	1.2	90.3	1.52
ORPL	2	99.8	0.380	98.2	0.71
ORPL	8	98.7	0.737	98.6	1.45

Table 5.5 – Natural interference: CRYSTAL.

			T-LOW		T-HIGH		
N_T	W_T	U	*PDR* (%)	*DC* (%)	*PDR* (%)	lost 1 pkt in	*DC* (%)
2	6	0	—	0.293	—	—	0.297
2	6	1	100	0.387	100	∞	0.396
2	6	2	100	0.479	100	∞	0.491
2	6	5	100	0.751	100	∞	0.773
2	6	10	100	1.205	99.997	33242	1.233
2	6	20	100	2.107	99.999	134077	2.162
2	6	48	100	4.883	100	∞	4.982
3	8	0	—	0.332	—	—	0.334
3	8	1	100	0.442	100	∞	0.451
3	8	2	100	0.551	100	∞	0.564
3	8	5	100	0.868	99.996	27667	0.890
3	8	10	100	1.391	100	∞	1.421
3	8	20	100	2.448	99.999	209201	2.475
3	8	48	100	5.596	100	∞	5.719

near-perfect reliability at 2Hz. Further, its *DC* is much lower than RPL thanks to opportunistic behavior.

Table 5.6 – Natural interference: ORPL (2Hz) vs. U.

	T-LOW		T-HIGH	
U	*PDR* (%)	*DC* (%)	*PDR* (%)	*DC* (%)
0	—	0.295	—	0.571
1	99.8	0.380	98.2	0.710
5	98.9	0.859	97.4	1.312
10	98.9	1.497	98.4	2.140
20	97.8	2.977	86.3	4.718
48	73.0	6.845	65.5	7.402

These results were derived with a single sender per epoch, $U = 1$. Table 5.6 shows the results for various values of U; we consider only ORPL as the performance of RPL is significantly lower. The reliability of ORPL decreases with the increase in traffic; ORPL still achieves a good $PDR = 97.8\%$ with $U = 20$, but the *PDR* drops to 73% when all $U = 48$ nodes transmit in every 10 s epoch. Duty cycle similarly increases sharply with U.

These trends are of course expected; however, Table 5.5 shows that, under the same conditions, the performance of CRYSTAL is significantly better, in line with what we reported in the previous chapter. Regardless of the $\langle N_T, W_T \rangle$ combination used, CRYSTAL *always* achieves perfect *PDR*, even in the extreme case of $U = 48$. This is largely to be ascribed to the excellent performance of the underlying Glossy layer (Table 5.7). Further, CRYSTAL achieves a *DC* lower than ORPL, itself the best among the mainstream protocols considered. For instance, for $U = 48$ the improvement is 18% with $N_T = 3$, and 29% with $N_T = 2$. With no data sent ($U = 0$), the *DC* of ORPL is, however, comparable with $N_T = 2$, and even lower than $N_T = 3$.

Note how the CRYSTAL sink is duty cycled just like any other node; this may be an asset in deployments where powering the sink is complicated. In contrast, the results shown throughout this chapter for mainstream protocols are derived with an always-on sink, which we verified to provide higher reliability and lower duty cycle; the latter, poorer results are omitted for brevity.

5.3.3 Natural Interference: T-HIGH

Next we discuss experiments assessing the same protocols during daytime, which presents higher levels of interference mostly arising from WiFi traffic, as discussed in 5.1.

Concerning the mainstream baseline protocols, a comparison of the left- and right-hand sides of Table 5.4 shows a generalized decrease in *PDR* accompanied by significant increases in duty cycle. As in T-LOW, ORPL remains the protocol with the best performance. The price to pay, however, is that the duty cycle increases nearly twofold for both 2 and 8Hz, as a result of longer idle listening and retransmissions induced by interference. Varying the number U of senders shows a similar trend of decreasing *PDR* and increasing duty cycle, as can be observed in Table 5.6.

Instead, CRYSTAL performs quite well (Table 5.5). *PDR* is perfect or near-perfect regardless of the value of U; the occasional packet loss seen for some values of U is likely due to the unpredictable nature of natural T-HIGH interference. Further, the duty cycle is nearly identical to the T-LOW case. For instance, in the worst-case scenario of $N_T = 3$ and $U = 48$, the increase in T-HIGH w.r.t. T-LOW is a negligible 0.22%. This is partly ascribed to the inherent reliability of the Glossy protocol CRYSTAL builds upon. However, our experiments also show that Glossy *by itself* does not achieve perfect reliability. The superior reliability of CRYSTAL is due to its redundancy mechanisms built *atop* Glossy, overcoming daytime noise with little additional overhead. Another way to look at this is to observe that even the configuration with $N_T = 2$, i.e., less reliability in the Glossy layer, still achieves the same reliability as $N_T = 3$, while of course enjoying better duty cycle.

Table 5.7 – *PDR* of Glossy (**T** phase of CRYSTAL with $U = 1$).

scenario	N_T, W_T	*PDR* (%)
T-LOW	2, 6	100
	3, 8	100
T-HIGH	2, 6	99.971
	3, 8	99.985
J-WIFI 1 interferer	3, 8	100
J-WIFI 6 interferers	3, 8	99.32
J-MWO 42	3, 8	99.88
J-MWO 13	3, 8	100
J-MWO 7	3, 8	67.90
J-MWO 7	6, 12	83.86
J-MWO 7	10, 17	99.76

5.3.4 Generated Interference: J-WIFI

We turn our attention to interference scenarios that we can control with JamLab, outlined in 5.1. This section focuses on WiFi interference, first observing the impact of a single interferer next to the sink, then of 6 interferers covering the entire network. We focus on $U = 1$ as this is sufficient to draw the observations motivating the further investigation described in the rest of this chapter.

Single interferer next to the sink. We designate a single node as JamLab interferer, specifically node 7 in Figure 5.1; its placement is challenging, as it is only 1 m from the sink. Table 5.8 shows that RPL achieves a reasonable *PDR* = 84%, while ORPL achieves near-perfect *PDR* with both 2Hz and 8Hz, and a duty cycle comparable to natural T-HIGH interference. In the same conditions, CRYSTAL achieves perfect reliability and a duty cycle lower than ORPL, as shown in

Table 5.9. This remarkable performance is mainly a consequence of the perfect performance of Glossy, as shown in Table 5.7.

Table 5.8 – Generated interference: Mainstream protocols, $U = 1$.

	JamLab node id	protocol	wake-up (Hz)	*PDR* (%)	*DC* (%)
J-WIFI	7	RPL	8	84	1.50
			2	89	1.30
	7	ORPL	8	99.7	1.31
			2	99.9	0.59
	7,12,20,31,37,49	ORPL	8	60	3.91
			2	64	1.70
J-MWO	42	ORPL	8	98.3	2.13
			2	98.6	0.844
	13	ORPL	8	99.7	1.84
			2	98.0	0.67
	7	ORPL	8	99.1	2.23
			2	99.8	0.67

Table 5.9 – Generated interference: CRYSTAL, $U = 1$.

	JamLab node id	N_T, W_T	R	*PDR* (%)	*DC* (%)
J-WIFI	7	3, 8	2	100	0.457
		2, 6	2	100	0.403
	7,12,20,31,37,49	3, 8	2	100	0.497
		2, 6	2	100	0.443
J-MWO	42	3, 8	2	100	0.507
		2, 6	2	99.52	0.430
	13	3, 8	2	100	0.459
		2, 6	2	100	0.405
	7	3, 8	2	78.5	0.453
		2, 6	2	78.6	0.425
	7	3, 8	6	100	1.11
	7	6, 12	2	100	0.839

Six WiFi interferers covering the entire network. We next consider a scenario with 6 JamLab nodes generating WiFi interference across the entire network like T-HIGH but, as outlined in 5.1 with significantly higher noise. We focus only on ORPL, as RPL showed low performance even with a single interferer. As seen in Table 5.8, ORPL has significant difficulty overcoming this noise level, regardless of the wake-up interval; in the best case, 8Hz achieves $PDR = 60\%$.

Glossy, instead, achieves near-perfect *PDR* (Table 5.7) and together with the CRYSTAL mechanisms built atop it achieves perfect reliability (Table 5.9). Interestingly, this is achieved with a duty cycle that is only 12% higher than in the natural T-LOW interference.

5.3.5 Generated Interference: J-MWO

We now investigate the impact of a JamLab-emulated microwave oven that, as illustrated in 5.1, induces a type of interference that is much stronger than WiFi and also characterized by different temporal patterns. In our experiments, we move the source of interference increasingly closer to the sink, therefore evaluating its effect in increasingly challenging scenarios. Given the results in the previous section, our comparison against mainstream protocols considers only ORPL, as the others yield unacceptable performance.

Interferer *far* from the sink, node 42. Our first set of experiments use a JamLab interferer on node 42, which is far from the sink, in a corner of the network and in the middle of a dense neighborhood. This means that interference from node 42 affects nodes that initiate traffic in its neighborhood, but likely bears limited influence on packets flowing in the rest of the network.

Table 5.8 shows that ORPL performs well in this scenario, although with an increased duty cycle compared to lower noise scenarios. This is due to ORPL buffering and continuously attempting to re-transmit packets until it finds the channel free, as we discussed in 5.2.2. Recall from 5.1.1 that the J-MWO scenario induces periods of strong interference alternated to periods with no interference. Therefore, the buffering and infinite CCA retries in ORPL effectively delay packets when the microwave oven interference is active, enabling their transmission during the no-interference periods. Nevertheless, these retransmissions do increase the duty cycle.

Crystal instead achieves perfect reliability, as shown in Table 5.9. Nevertheless, the underlying Glossy layer is somewhat affected by interference, as seen in Table 5.7; therefore, reliability in CRYSTAL comes at the cost of a higher duty cycle. This cost is even higher than in the case with 6 WiFi interferers, despite the fact that in the latter scenario the reliability of Glossy is worse. The reason lies in the position of node 42; being in a corner of the network, its strong interference causes the loss of acknowledgments in that neighborhood, triggering retransmissions from the corresponding senders and unnecessarily keeping the *entire* network awake to help forwarding. In the scenario with 6 WiFi interferers that cover the entire network, packet losses are more spatially and temporally distributed; in this situation, the redundancy achieved by the combination of Glossy and CRYSTAL mechanisms enables packets to more easily find routes "around" the interference.

Interferer *close* to the sink, node 13. We now move the interference source to node 13, about 4m from the sink, based on the intuition that this is likely to be more disruptive than the far away node 42, but less than an even closer placement, discussed next.

However, our results tell a different story. The reliability of ORPL is nearly perfect, as shown in Table 5.8, and is achieved with a duty cycle that is about 20% lower than in the previous case with node 42. The same holds for CRYSTAL (Table 5.9), which achieves perfect reliability with a duty cycle that is about 9% less than in the case of node 42, thanks to the perfect reliability of Glossy (Table 5.7).

For all protocols, the reason for the better performance with the interferer on node 13 w.r.t. 42 is due to the position of the nodes. Node 13 is closer to the sink, and therefore in principle induces a stronger interference on it. Nevertheless, it also means that it has a more "central" position, which allows packets to follow routes "around" it. In contrast, node 42 is in the corner of the network, as already mentioned, and the disruption caused to that portion of the network is much harder to compensate via alternative routes.

Interferer *next* to the sink, node 7. When moving the emulated microwave oven on node 7, 1m from the sink, the reliability of CRYSTAL significantly degrades for the first time, causing a 21.5% packet loss (Table 5.9), mainly due to the fact that, unlike the previous scenarios, the underlying Glossy layer fails to achieve acceptable reliability, losing 32.1% of the packets. The reason is that the interference on node 7 is so strong and so close that Glossy cannot overcome it. Receiving packets via alternate routes, as in the case of node 13, is no longer an option because *all* routes are jammed by interference, given that the sink is basically at the center of it.

In contrast, ORPL achieves near-perfect reliability also in this case (Table 5.8), and with a duty cycle only marginally different w.r.t. the interference source on node 13. From the point of view of ORPL, the two situations are virtually the same: *i)* both node 7 and 13 are in the center of the network, and therefore do not suffer from the more challenging corner placement of node 42, and *ii)* in both cases, the buffering and retransmission guarantees that, if a packet is not lucky enough to be received by the sink despite interference, it eventually is received in the periods without interference.

On the other hand, CRYSTAL dissemination is designed to be as fast as possible, even with all the amount of redundancy built atop the even shorter one-shot Glossy floods. Consequently, CRYSTAL and Glossy cannot exploit a "wait-and-see" strategy as in ORPL.

5.3.6 Is There a Better Configuration?

We explore whether we can find a configuration that yields perfect reliability in the only scenario synchronous transmissions cannot cope with, i.e., node 7 acting as a J-MWO interferer. We explore two options: one in CRYSTAL, and one in the underlying Glossy layer.

CRYSTAL: Keeping the network awake. We already observed that an asset of ORPL in the scenario at stake is its ability to keep retransmitting until interference is over. The analogous CRYSTAL configuration comes from increasing R. As discussed in the previous chapter, this is the number of consecutive empty **T** slots that must be detected by the sink to determine that communication is over, and it is safe to enter sleep mode until the beginning of the next epoch. Increasing R keeps the network awake longer. This gives senders more opportunities to attempt retransmission under interference. Indeed, Table 5.9 shows that $R = 6$ enables perfect reliability. However, keeping the network awake for three times longer than before causes a nearly three-fold increase in duty cycle.

Glossy: Increasing redundancy. An alternative is to make the underlying Glossy layer more reliable. As discussed in 5.2.3, the main knob to achieve this is to increase the number N of retransmissions during a flood, and increase the slot duration W to ensure the flood has enough time to complete. We verified experimentally that, when pure Glossy is used in isolation, a setting $N = 10$, $W = 17$ yields $PDR = 99.76\%$. However, the reliability provided by CRYSTAL atop Glossy enables the use of a smaller N, considerably reducing duty cycle. Indeed, Table 5.9 shows that with $N_T = 6$, $W_T = 12$, CRYSTAL achieves perfect delivery (despite Glossy yielding only $PDR = 83.86\%$, see Table 5.7) but nearly doubles the duty cycle, as each packet is transmitted twice as many times w.r.t. $N_T = 3$.

In summary, a proper *static* configuration of either CRYSTAL or Glossy parameters enables perfect reliability, but with unacceptable energy consumption if compared to what ORPL achieves, albeit with only nearly-perfect reliability. Ideally, perfect reliability should come without a significant increase w.r.t. the duty cycle observed in the other scenarios in Table 5.9, i.e., at most 0.50%. Further, over-provisioning for the worst case, as these static configurations achieve, is undesirable. Ideally, CRYSTAL should *dynamically* adapt to interference, bearing extra duty cycle costs only when needed. These observations motivate the techniques we illustrate next.

5.4 Taming Strong Interference

In this section we illustrate a technique to *escape* interference and a complementary one to *fight* it after detecting its presence.

Escaping Interference: Channel Hopping. We first turn to a well-known technique for interference resilience: exploiting frequency diversity. As discussed in 5.1.1, interference typically affects only a subset of the 16 channels available. Therefore, a network-wide channel-hopping sequence can be exploited to enable subsequent **TA** pairs to move to different channels, reducing the probability that two consecutive ones both execute on noisy channels. This simple modification does not affect any CRYSTAL parameters.

Figure 5.3 illustrates this, highlighting that channel hopping is performed on the **S** phase, following a predefined sequence. The channels of the subsequent **TA** pairs in the epoch depend on the channel of the **S** phase. This mechanism keeps all nodes on the same, rotating channel at the beginning of each epoch, independent of the number of **TA** phases they executed in the previous epoch.

A key design decision is how to determine the next channel to be used. As discussed in 5.1.1, WiFi and microwave ovens are common sources of interference that occur on 4 and 7 consecutive channels, respectively. Spacing the current and next channel apart by 4 channels would be sufficient to escape a WiFi source, but not a MWO source. Therefore, our implementation uses a hopping sequence with 7 channel spacing. Different hopping sequences could be designed, e.g., based on knowledge of industrial interference.

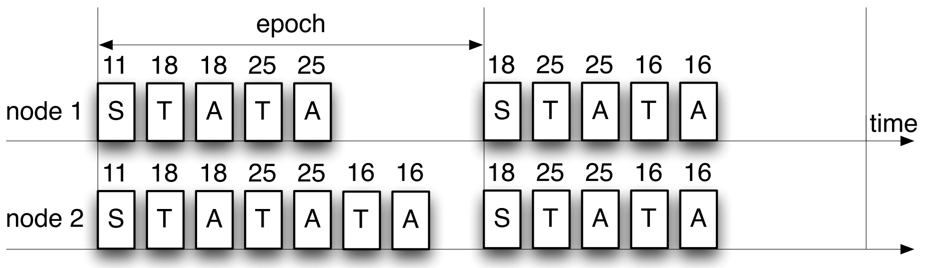


Figure 5.3 – Channel hopping in CRYSTAL. The number on each CRYSTAL phase denotes the channel used.

Fighting Interference: Noise Detection. Our next technique relies on the ability to detect abnormally high noise levels. Recall from the previous chapter that, in CRYSTAL, the distributed termination condition relies on counting **T** slots without data and **A** slots without acknowledgements. Under high noise, these *missing-packet* conditions often occur even when a packet was transmitted, but encountered interference. If this happens during the **T** phase and in the neighborhood of the sink, the sender will re-transmit the packet in the next **T** slot but the sink still may not receive the packet in R consecutive **T** slots, therefore mistakenly detecting termination and putting the whole network to sleep. Instead, noise in the network periphery may cause a node to similarly miss Z acknowledgements, going to sleep, likely before the sink. In both cases, data may remain undelivered because termination was falsely detected.

Adding noise detection and modifying distributed termination conditions fights these cases. Intuitively, in the presence of noise, missing packets are not counted toward the termination condition, keeping the network awake and allowing more opportunities for data and acknowledgments to escape the interference. Recall that receiving any packet keeps a node awake to serve as a forwarder.

Noise detection can be easily achieved by periodically checking the CCA pin of the CC2420 radio chip; in our current implementation, all nodes perform the CCA check every 64μs while listening during **T** or **A** phases, and define high noise when an $RSSI > -60$dBm is detected at least 80 times. Note that this threshold is designed to detect only very high noise, e.g., a microwave oven; lower thresholds would unnecessarily trigger the scheduling of extra **TA** pairs, e.g., in the WiFi scenarios of 5.3, where the unmodified CRYSTAL achieves perfect reliability.

We make the following modifications to CRYSTAL:

- define R_{noise} as the maximum number of *consecutive* slots that *i)* do not contain a packet *and ii)* with high noise.
- change the termination rule at the sink to put the network to sleep when *either i)* it detects R non-noisy empty **T** slots since the last received data packet, *or ii)* $max(R, R_{noise})$ consecutive noisy **T** slots (with no data) occur.
- change the termination rule at network nodes to go to sleep when *either i)* it receives a sleep command from the sink (piggybacked on an acknowledgement), *ii)* it has data to send and it detects Z non-noisy **A** slots with no acknowledgements since the last time

it received an acknowledgement, *or iii)* it does not have data to send and it detects Y non-noisy **TA** pairs without any packets since the last time it received a packet in either **T** or **A** slot, *or iv)* $max(Z, R_{noise})$ consecutive noisy **A** slots (without acknowledgements) occur.

The new termination strategy of CRYSTAL allows to overcome interference that lasts less than R_{noise} **TA** pairs. A potential drawback of this technique is that it schedules extra **TA** pairs upon detecting high noise regardless of whether a packet is currently being disseminated. If no packet is being disseminated (and there is no way to ascertain it in CRYSTAL due to its aperiodic and sparse traffic) our noise detection technique may yield unnecessary energy consumption.

We empirically determined that $R_{noise} = 6$ strikes a good balance between reliability and energy consumption, and use this value. If $R_{noise} = 0$, the noise detection mechanism is disabled.

Fighting *and* Escaping Interference. Although each of these techniques improves performance along some dimension, it is only through their combination that very strong sources of interference can be effectively overcome with very low energy consumption. Indeed, the adoption of frequency diversity reduces the probability of the sink to be exposed, in consecutive **TA** pairs, to high noise levels from the same interference source, and therefore mitigates the aforementioned drawback of noise detection. On the other hand, the ability to detect and react to noise is helpful in reducing packet loss when hopping from one bad channel to another one. The next section illustrates experimental results supporting these arguments.

5.5 Under Strong Interference

We now evaluate the techniques in 5.4 and show that they not only overcome the interference scenarios considered in 5.3, but also sustain much higher noise levels, whose details are described next.

5.5.1 Experimental Setup

We extend our experimental setup along three dimensions.

Channel mapping. Testing our channel hopping mechanism in principle requires reproducing interference across multiple channels, something JamLab cannot do, as discussed in 5.1.1. We overcome this limitation by defining a *mapping* between the 16 channels provided by IEEE 802.15.4 and those in the testbed. Whenever our channel hopping mechanism decides to switch to a channel c, a corresponding channel c_{real} is instead used for communication, based on a predefined mapping $c \rightarrow c_{real}$ established based on the interference types and channels affected we want to reproduce. For instance, when emulating a microwave oven, we map channels 20–26 to the real channel used by J-MWO interferers.

More challenging interference scenarios. As described later, extending CRYSTAL with the techniques in 5.4 allows it to sustain much stronger interference than considered in 5.3, which considered the *separate* effect of generated J-WIFI and J-MWO interference. Therefore, we now focus on the *combined* effect of these two interference types. We combine them in two ways, yielding the scenarios in Table 5.10. The first, COMBINED$_{split}$, combines the two types of interferences by placing each on different real channels. This significantly reduces the chances that channel hopping finds a good channel, and increases the likelihood to hop from one type of interference to the other. The second scenario is even more challenging, placing J-MWO and J-WIFI interferers on the *same* real channel, generating noise that is the *sum* of the two. Increasingly challenging scenarios can be generated by determining the number n of channels this strong interference is mapped to. Table 5.10 shows we experiment with values ranging from 7 (i.e., the maximum of channels affected when J-MWO and J-WIFI fully overlap in reality) to 16 (i.e., *all* channels jammed by the same combined interference).

Besides combining the types of interference, we also strengthen J-MWO, the most disruptive one, by using 2 interferers simultaneously, selected to be the worst based on 5.3: node 7 next to the sink, and node 42 in the corner. As for J-WIFI, using the scenario with 6 interferers used in 5.3 would force us to remove a total of 8 nodes, therefore reducing further the size of the network. Therefore, we used 4 J-WIFI interferers that, as we verified experimentally, yields a noise pattern close to natural T-HIGH interference.

Table 5.10 – Scenarios with combined interference generated by 2 J-MWO and 4 J-WIFI.

scenario	#channels jammed		description
	2 J-MWO	4 J-WIFI	
COMBINED$_{split}$	7	6	interferers on different *real* channels based on type, mapped on different sets of channels
COMBINED$_n$	$n \in \{7, 10, 13, 16\}$		all interferers on one *real* channel, itself mapped on n channels

43-node topology. Therefore, overall we use 6 out of the 49 nodes in the testbed as JamLab interferers. In the rest of this section, we *always* consider a network of the same 43 nodes, even when reporting results where some or none of the interferers are used as in T-HIGH. This avoids biasing the results with different connectivity properties, which is more important in this section given that this contains our final and most impressive results, derived under significantly higher interference levels than in 5.3. Note that this 43-node topology is itself a *more* challenging environment, given that nodes are intrinsically less connected.

To re-establish our baseline against ORPL, the left-hand side of Table 5.11 shows results from experiments in this 43-node topology, by considering the individual performance with 4 J-WIFI and 2 J-MWO, and their combined one over a single channel. Note that ORPL still performs quite well even with 2 J-MWO (one of which includes node 7), but its performance degrades significantly even with only 4 J-WIFI interferers, instead of the 6 used in 5.3.4.

Table 5.11 – ORPL (2Hz) in a 43-node network, $U = 1$.

scenario	TX Power 0 dBm		TX Power −7 dBm	
	PDR (%)	*DC* (%)	*PDR* (%)	*DC* (%)
T-LOW	99.6	0.497	97.0	0.454
T-HIGH	98.5	0.776	—	—
4 J-WIFI	61.0	1.35	39.5	6.192
2 J-MWO	97.8	1.19	94.8	1.503
2 J-MWO 4 J-WIFI	65.0	2.14	39.6	5.375

5.5.2 Channel Hopping

We are now ready to investigate CRYSTAL extended with channel hopping as discussed in 5.4. We call this variant CRYSTAL^{CH}, to distinguish it from the original single-channel one, and similarly call CRYSTAL$^{CH}_{ND}$ the variant that also adds noise detection.

Table 5.12 – CRYSTAL^{CH}, T-HIGH.

N_T	U	*PDR* (%)	*DC* (%)
2	0	—	0.294
2	1	100	0.392
2	2	100	0.486
2	5	100	0.766
2	10	100	1.221
2	20	100	2.122
2	48	100	4.768

A first glimpse at the impact of channel hopping is shown in Table 5.12, reporting experiments under natural T-HIGH interference, without channel mapping and with CRYSTAL^{CH} hopping across all 16 channels. A comparison with Table 5.5 shows that CRYSTAL achieves perfect reliability regardless of U, and does so with the setting $N_T = 2$, which generally yields worse reliability w.r.t. $N_T = 3$. Further, the duty cycle is comparable to (actually 1–2% lower than) the single-channel version. This confirms that our incorporation of channel hopping into CRYSTAL is already useful with natural interference.

The bigger question lingering from 5.3, however, is whether CRYSTAL^{CH} is able to overcome J-MWO interference next to the sink. To this end, we first analyze the performance in the COMBINED$_{split}$ scenario, shown in Table 5.13 (left). Recall that this scenario subsumes the J-MWO on node 7 we discussed at the end of 5.3, by adding a second interferer on node 42, and overall defines a much more challenging scenario. Indeed, when hopping out of J-MWO interference, which is found with a $\frac{7}{16} = 43.75\%$ probability, there is still a 37.5% chance to stumble on J-WIFI interference, and only a 18.75% chance to enjoy T-LOW interference. Nevertheless, in this case CRYSTAL^{CH} achieves perfect reliability for $U = 1$ and three-nines reliability for $U > 1$, $N_T = 3$. Further, this is achieved with only a slight increase in duty

Table 5.13 – CRYSTAL^{CH} vs. CRYSTAL$^{CH}_{ND}$.

		CRYSTAL^{CH} COMBINED$_{split}$			CRYSTAL$^{CH}_{ND}$ COMBINED$_{16}$		
N_T	U	*PDR* (%)	lost 1 pkt in	*DC* (%)	*PDR* (%)	lost 1 pkt in	*DC* (%)
2	0	—	—	0.335	—	—	0.543
2	1	100	∞	0.454	99.526	211	0.763
2	2	100	∞	0.583	99.360	156	0.988
2	5	99.949	1980	0.954	99.027	103	1.712
2	10	99.982	5620	1.568	98.643	74	3.020
2	20	99.979	4822	2.792	98.582	71	5.372
2	42	97.627	42	5.646	90.056	10	9.308
3	0	—	—	0.374	—	—	0.739
3	1	100	∞	0.514	99.705	339	0.955
3	2	100	∞	0.661	99.402	167	1.218
3	5	99.949	1978	1.069	97.665	43	1.792
3	10	99.988	8374	1.750	96.224	26	2.964
3	20	99.941	1681	3.138	96.640	30	5.206
3	42	100	∞	6.434	94.397	18	9.952

cycle w.r.t. our lowest-interference scenario, T-LOW: 14.3% and 12.6% for $N_T = 2$ and $N_T = 3$, respectively.

A natural next step is to identify the limit at which CRYSTAL^{CH} breaks, which clearly depends on the type of interference applied and on the number of channels affected. Table 5.14 explores this limit by using the COMBINED$_n$ scenario of Table 5.10. The interference is stronger, as it is the *sum* of 2 J-MWO *and* 4 J-WIFI, which in the left part of Table 5.13 are instead split on separate sets of channels. Moreover, we apply this strong interference to an increasing number n of channels, which allows us to leave fewer and fewer good options to hop away from interference. The left-hand side of Table 5.14 shows that when $n = 7$, CRYSTAL^{CH} achieves perfect reliability regardless of the configuration N_T and number U of senders, with a duty cycle marginally smaller than in COMBINED$_{split}$. On the other hand, when only 6 channels are free and the others subjected to COMBINED$_{10}$ interference, performance drastically drops. When only $U = 1$ sender is active, $PDR = 95\%$ is achieved at best, with $N_T = 3$; as U increases, *PDR* plummets. Finally, with only 3 channels free reliability becomes unacceptable, reaching $PDR < 85\%$ with a single sender, and degrading to at best 13.9% with all 42 senders.

5.5.3 Channel Hopping *and* Noise Detection

These scenarios are very challenging, both in absolute terms and relative to the literature, making the performance of CRYSTAL^{CH} already remarkable. Nevertheless, here we show that we can push reliability even further. When interference affects so many channels that it becomes difficult to *escape* it, the only other choice to improve reliability is to *fight* it, with the

Table 5.14 – CRYSTAL$^{\text{CH}}$ vs. CRYSTAL$^{\text{CH}}_{\text{ND}}$, COMBINED$_n$.

			CRYSTAL$^{\text{CH}}$			CRYSTAL$^{\text{CH}}_{\text{ND}}$		
n	N_T	U	*PDR* (%)	lost 1 pkt in	*DC* (%)	*PDR* (%)	lost 1 pkt in	*DC* (%)
7	2	0	—	—	0.328	—	—	0.367
7	2	1	100	∞	0.456	100	∞	0.497
7	2	10	100	∞	1.574	100	∞	1.624
7	2	20	100	∞	2.746	100	∞	2.890
7	2	42	100	∞	5.848	100	∞	5.936
7	3	0	—	—	0.370	—	—	0.444
7	3	1	100	∞	0.513	100	∞	0.587
7	3	10	100	∞	1.749	100	∞	1.826
7	3	20	100	∞	3.142	100	∞	3.256
7	3	42	100	∞	6.494	100	∞	6.566
10	2	0	—	—	0.347	—	—	0.430
10	2	1	93.947	17	0.479	100	∞	0.562
10	2	10	71.201	3	1.289	99.962	2637	1.911
10	2	20	46.900	2	1.511	99.788	471	3.414
10	2	42	22.459	1	1.618	99.557	226	7.008
10	3	0	—	—	0.386	—	—	0.521
10	3	1	95.094	20	0.536	100	∞	0.690
10	3	10	74.722	4	1.469	99.646	282	2.130
10	3	20	54.497	2	1.878	99.252	134	3.730
10	3	42	30.353	1	2.271	98.402	63	7.464
13	2	0	—	—	0.362	—	—	0.483
13	2	1	84.363	6	0.489	99.748	397	0.656
13	2	10	36.581	2	0.844	99.719	356	2.322
13	2	20	19.497	1	0.817	99.409	169	4.132
13	2	42	9.277	1	0.882	97.577	41	8.456
13	3	0	—	—	0.410	—	—	0.628
13	3	1	85.938	7	0.558	99.919	1237	0.696
13	3	10	47.427	2	1.073	98.608	72	2.484
13	3	20	26.522	1	1.121	96.901	32	4.268
13	3	42	13.936	1	1.230	97.360	38	8.436

noise detection technique of 5.4.

The right-hand side of Table 5.14 shows results. With $U = 1$, CRYSTAL$^{\text{CH}}_{\text{ND}}$ achieves perfect reliability with $n = 10$ channels jammed, two- to three-nines reliability with $n = 13$. However, the biggest performance gap between CRYSTAL$^{\text{CH}}$ and CRYSTAL$^{\text{CH}}_{\text{ND}}$ can be observed with $U > 1$, noting that the *PDR* remains always above 99%, except for $U \geq 10$ and $N_T = 3$; the worst-case $U = 20$, however, still achieves a respectable *PDR* = 96.9%.

Noise detection becomes more and more important as the number n of jammed channels increases. The extreme case is when *all* channels are jammed by the same strong interference (Table 5.13, right), in which channel hopping becomes pointless and reliability is provided entirely by noise detection, which performs quite well. Indeed, the *PDR* achieved here is only marginally lower than in COMBINED$_{13}$, with the worst-case for $U = 42$ achieving *PDR* = 90%.

To put this value in context, we observe that it is *i)* comparable with what RPL achieves in the T-LOW scenario with $U = 1$ (Table 5.4), and *ii) more* than what ORPL achieves in the natural T-HIGH scenario (no microwave ovens) with $U = 20$ (Table 5.6).

The price to pay for this remarkable reliability, however, is energy consumption. As mentioned (5.4), a drawback of noise detection is that high noise keeps the network awake even without packet transmissions. This is reflected in the increase of *DC* as the number n of jammed channels increases; as more channels are jammed by strong interference, the likelihood of remaining unnecessarily awake increases. This is clearly undesirable for $U = 0$; yet, it is key to reliability as U increases, as seen by comparing the two sides of Table 5.14. However, the actual impact of this increased *DC* on the overall energy consumption depends on the characteristics of the aperiodic traffic at hand, as we analyze in 5.5.5.

5.5.4 A Different Topology: Low Power

We now present results with lower output transmission power, −7 dBm. This reduces the number of neighbors and increases network diameter (Table 5.2), providing a more challenging topology.

To re-establish the ORPL baseline, we repeated experiments in the new topology (Table 5.11, right-hand side). ORPL performs close to the high-power setting with only minimal (T-LOW) or J-MWO interference, but shows drastic performance degradation in the presence of J-WIFI, with an almost halved *PDR*.

We ran multiple experiments with CRYSTAL, confirming the trends we observed throughout this chapter. Nevertheless, duty cycle increases slightly in all cases, as we necessarily increase the size of Glossy slots to handle the larger network diameter (Table 5.3). Due to space limitations, we focus only on the final and most challenging scenarios, with CRYSTAL$^{\text{CH}}_{\text{ND}}$ incorporating both techniques from 5.4.

The left-hand part of Table 5.15 shows the performance in the COMBINED$_{split}$ scenario. Com-

Table 5.15 – Low power: CRYSTAL$^{\text{CH}}_{\text{ND}}$.

	COMBINED$_{split}$				COMBINED$_{16}$			
U	N_T	*PDR* (%)	lost 1 pkt in	*DC* (%)	N_T	*PDR* (%)	lost 1 pkt in	*DC* (%)
0	3	—	—	0.541	4	—	—	1.072
1	3	99.992	11825	0.709	4	99.031	103	1.427
2	3	100	∞	0.865	4	97.936	48	1.830
5	3	100	∞	1.335	4	97.969	49	2.871
10	3	99.997	28889	2.125	4	96.376	28	4.515
20	3	99.987	7840	3.652	4	95.082	20	7.784
42	3	100	Inf	7.612	4	93.146	15	14.982

Table 5.16 – An aperiodic, sparse data traffic profile, showing number and fraction of epochs with U concurrent senders.

	U	0	1	2	5	10	20
epochs	#	84.3K	15.5K	2.2K	606	46	1
	%	82.1	15.1	2.2	0.14	0.038	0.005

paring against the left part of Table 5.13 we see that $N_T = 3$ is enough to obtain a reliability similar to the high-power case. On the other hand, to sustain the most challenging scenario COMBINED$_{16}$ where all channels are jammed, the redundancy of the underlying Glossy layer must be increased to $N_T = 4$ (Table 5.15, right-hand side). With this setting, CRYSTAL$^{\text{CH}}_{\text{ND}}$ achieves a *PDR* within 0.5–3% of the high-power setting, remaining above 93% even with all 42 concurrent senders.

These results confirm that our techniques bring significant benefits also in the larger-diameter, lower-power setting we considered.

5.5.5 Back to Aperiodic, Sparse Data Collection

We now reconcile the experimental results we reported with the original goal of supporting aperiodic, sparse data collection.

To this end, we use one of the data traffic profiles presented in the previous chapter, the one resulting from applying the data prediction technique to temperature samples of the 36-day Intel dataset [58]. The sampling rate (epoch) is equal to 30 seconds.

We slightly adapted the traffic profile to cater for missing values of U in our experiments (Table 5.16). A missing value is always replaced by the next higher value of U available; for instance, the value 606 corresponding to $U = 5$ is actually the sum of the epochs in which 3, 4, or 5 concurrent senders were present. This achieves a worst-case estimate of *PDR* and *DC*, as both increase with U. These are aggregated over the entire dataset as

$$C = \frac{\sum_{u=0}^{N} c(u) \times e(u)}{\sum_{u=0}^{N} e(u)}$$

where $c(u)$ is the value of *PDR* or *DC* for a given number u of concurrent senders (reported in previous sections) and $e(u)$ is the number of epochs in which u concurrent senders are present (from Table 5.16). We re-scaled *DC* to the epoch $E = 30$s (i.e., $\frac{1}{3}$ of those hitherto shown) used the original dataset, therefore enabling a comparison with the performance reported in the previous chapter, albeit in a different testbed. The results are shown in Table 5.17 where, due to space limitations, we consider only the extremes of the interference scenarios we analyzed in this chapter, viz. natural interference and generated interference in the COMBINED$_{16}$ scenario. These are however sufficient to draw a few interesting observations.

Table 5.17 – Performance of CRYSTAL$^{\text{CH}}_{\text{ND}}$ with the aperiodic, sparse, real-world data traffic profile shown in Table 5.16.

interference scenario	$N_T = 2$		$N_T = 3$	
	PDR	*DC*	*PDR*	*DC*
T-LOW	100	0.105	100	0.119
T-HIGH	100	0.105	—	—
COMBINED$_{16}$	99.487	0.198	99.592	0.263

First, *DC* in the T-LOW scenario is around 0.1%, i.e., achieving the per-mille duty cycle targeted and reported in the previous chapter of this thesis, confirming that the results we report for T-LOW are in line with the original ones.

Indeed, in our previous study, we ran experiments on channel 20 and 26 which *"showed very similar performance [...] during the night runs; however, the daytime results were inconsistent and difficult to assess"* and therefore *"the results only from night runs on channel 26"* were included. It is therefore interesting to look at the performance of CRYSTAL$^{\text{CH}}_{\text{ND}}$ in (daytime) T-HIGH; Table 5.17 shows that it is identical to the one in T-LOW, a result that can be ascribed to the higher performance under interference of our techniques.

Finally, Table 5.17 shows that the *PDR* accrued over the 36-day dataset remains near to 99.5% in COMBINED$_{16}$, which is remarkable given the very challenging nature of this interference scenario. Further, this is achieved with a *DC* slightly above or below 0.2% (2 per mille) depending on N_T; in relative terms this is twice the baseline established by natural interference, but in absolute terms is very small w.r.t. what commonly reported in the state of the art.

5.6 Related Work

Our goal was to offer an in depth evaluation of synchronous transmission, specifically CRYSTAL, in the presence of interference, motivating our study CRYSTAL and CRYSTAL$^{\text{CH}}_{\text{ND}}$ in a range of conditions and parameters. We offered detailed comparison to ORPL as it offers good performance and also serves as the baseline for related work.

CSMA + Channel Hopping. Adding channel hopping to combat interference is well accepted in the literature, with recent works modifying standard, CSMA protocol stacks. MiCMAC [74] extends ContikiMAC with channel hopping, resulting in a synchronization-free MAC with high *PDR* in the presence of WiFi interference. MiCMAC mechanisms require transmitting and receiving nodes to synchronize in time as well as across channels, increasing latency. Oppcast [69] and MOR [108] offer full-stack alternatives to RPL and MicMAC, combining channel hopping and opportunistic routing to combat high latencies while also escaping high interference.

To offer an informal, numerical comparison, we consider the evaluation in [108] for FlockLab with duty cycling WiFi on one channel and $U = 2.1$. MOR, using this single jammed channel plus two free channels, showed the best results, namely $PDR = 99.35\%$ and $DC = 1.56\%$. We compare to a more challenging scenario with constant, generated WiFi traffic on all channels in our testbed where CRYSTAL shows $PDR = 100\%$ and $DC = 0.559$ for $N = 2$. This *DC* is nearly *three times smaller* than that of MOR, and is achieved without any interference avoidance mechanisms. Naturally, with more concurrent packets, the CRYSTAL *DC* increases, however the same is true for other protocols. Further, in the absence of traffic, a common case in 5.5.5, CRYSTAL maintains $DC < 0.4\%$, levels that duty cycling protocols cannot achieve due to the required periodic channel probing. Finally, to manage latency, these protocols hop among few channels, selected during pre-deployment evaluations. By contrast, CRYSTAL^{CH} can use all channels without affecting its performance, allowing it to adapt to changing interference conditions. Finally, unlike our investigation, these protocols have been stress tested only under WiFi interference, and only for constant periodic traffic.

TDMA + Channel Hopping. TSCH [2] with Orchestra [34] scheduling offers a protocol in which all nodes follow a repeating, slotted schedule, with local and independent slot allocation. The number and type of slots is statically determined, according to expected traffic. Results from Indriya show Orchestra with 47 slots maintains $PDR = 99.99\%$ with an average $DC = 0.4\%$, without interference. While this is better than CRYSTAL, which in an analogous setting offers $DC = 0.8\%$. However, Orchestra node duty cycles vary significantly throughout the network, with nodes closer to the sink reporting much higher values. Further, Orchestra is designed for periodic data, which is critical to statically configure slot parameters. In dynamic scenarios such as 5.5.5, Orchestra would over-dimension for the worst case, unable to reduce the *DC* under low traffic. Finally, recall that Orchestra has not been evaluated under interference.

Synchronous Transmissions + Channel Hopping. Applying channel hopping to synchronous transmissions has been explored in the context of the EWSN dependability competition [91]. The three winning 2017 approaches [64, 36, 73] perform channel hopping directly in Glossy. This has significant drawbacks: *i)* the implementation is more complex, as changing channel between Glossy transmissions interferes with timing; *ii)* it requires a high value of the Glossy parameter N, to attempt retransmission on several (potentially all 16) channels. In contrast, our application of channel hopping is *atop* Glossy. This *i)* yields a simple implementation, and *ii)* relies on the intrinsic reliability of Glossy even with low N.

5.7 Conclusions

This chapter set out to verify the extent to which the common wisdom that synchronous transmissions are highly resilient to interference holds. We selected the recent CRYSTAL protocol to study, as its heavy reliance on the capture effect and focus on aperiodic, sparse traffic define very challenging requirements for interference resilience. Along the way, we also evaluated the underlying Glossy protocol, as well as two representative mainstream protocols, RPL and ORPL.

Unlike many existing works limited to study under WiFi interference, we subjected our protocols to the stronger noise generated by microwave ovens, emulated via JamLab. In our reproducible and controlled setting we showed, for the first time, that ORPL is very resilient to this type of interference, while synchronous transmissions are not. This motivated us to push them further, exploiting a combination of channel hopping and noise detection. We showed that our enhanced $\text{CRYSTAL}^{\text{CH}}_{\text{ND}}$ protocol achieves unprecedented, near-perfect reliability even against the combination of emulated WiFi and microwave ovens, along with a per-mille duty cycle in the aperiodic, sparse traffic targeted by CRYSTAL.

Conclusion

In this thesis we proposed and evaluated novel techniques and protocols that harmonise support for *data collection* and *actuation* traffic in protocol stacks for multi-hop wireless low-power and lossy networks (LLNs). We started by analysing shortcomings of the standard IPv6 protocol stack for LLNs on memory-constrained devices and, motivated by a smart city application scenario, improved the scalability of its downward forwarding mechanisms. This improvement brought the downward forwarding aspect of the stack on par with the upward forwarding in networks of several hundreds of nodes and with a very moderate increase in energy consumption. This achievement extends the applicability of the stack while preserving its main benefit of being IPv6-compliant.

Nevertheless, in our second application scenario that comprises a control loop and requires real-time guarantees, the reliability of our standard-compliant stack is not enough. Moreover, the overhead of the routing and MAC layers prevents proportional energy savings when the application-level data prediction technique suppresses most of the traffic.

To take full advantage of data prediction, we created CRYSTAL, a novel data collection protocol, based on synchronous transmissions. We showed that it has unprecedented reliability, and that it dynamically adapts to the immediate traffic demands, delivers all data fast, when they appear, and is ultra-low power when the traffic is sparse or not present.

CRYSTAL is applicable as a general-purpose data collection protocol with properties similar or better than those of the state of the art, however, its full potential is unleashed in applications that are periodic in their nature but are able to suppress the great majority of their traffic. We showed in this thesis that control applications like the adaptive tunnel lighting achieve a sub-permille radio duty cycle with CRYSTAL. We also proved experimentally that CRYSTAL sustains very strong external interference, which is an asset for wireless control systems in industrial environments.

Being a data collection protocol, CRYSTAL builds atop Glossy floods, that themselves provide a similarly efficient, fast and reliable actuation service. Our protocol already uses sink-initiated Glossy floods to deliver acknowledgements, the same point-to-multipoint primitive can be utilised for sending actuation commands, resulting in a combined actuation-collection protocol stack.

Through these contributions, this thesis pushes forward the applicability of LLNs, by improving their scalability, reliability, latency, energy efficiency and interference resilience. At the same time, this thesis opens new possibilities for future research, briefly characterised in the following.

CRYSTAL is a clean-slate design, not constrained by Internet standards. Therefore, a new IPv6-enabled stack based on the principles we used in CRYSTAL might be created in the future. Related to this, it is in our research plan to thoroughly compare CRYSTAL with recent IPv6-enabled techniques built on top of IEEE 802.15.4 TSCH [34].

The current version of CRYSTAL was implemented for a relatively old TMote Sky platform with an MSP430 microcontroller and a separate CC2420 radio chip. Newer platforms built around a system-on-chip and/or new physical layers like Ultra-Wide Band PHY of IEEE 802.15.4 may provide even better properties thanks to more precise timing of events and operations they provide.

A promising area where CRYSTAL or techniques based on similar principles can make a difference is *event-based control* [5] that requires a combination of scheduled and non-predictable aperiodic traffic with exactly the reliability and adaptability CRYSTAL provides. It is in our immediate plans to try CRYSTAL in combination with various event-based approaches to control.

Bibliography

[1] http://wirelesstrident.sourceforge.net.

[2] 802.15.4e Task Group. “802.15.4e-2012: IEEE Standard for Local and metropolitan area networks - Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer”. In: (2012).

[3] G. Anastasi et al. “Energy conservation in wireless sensor networks: A survey”. In: *Ad Hoc Networks* 7.3 (2009), pp. 537–568.

[4] E. Ancillotti, R. Bruno, and M. Conti. “On the interplay between RPL and address autoconfiguration protocols in LLNs”. In: *Proc. of the Int. Wireless Communications and Mobile Computing Conf. (IWCMC)*. July 2013, pp. 1275–1282. DOI: 10.1109/IWCMC.2013.6583740.

[5] Jose Araujo et al. “System Architectures, Protocols and Algorithms for Aperiodic Wireless Control Systems”. In: *IEEE Transactions on Industrial Informatics* 10.1 (Feb. 2014), pp. 175–184. ISSN: 1551-3203. DOI: 10.1109/TII.2013.2262281. URL: http://ieeexplore.ieee.org/document/6514939/.

[6] Nouha Baccour et al. “A testbed for the evaluation of link quality estimators in wireless sensor networks”. In: *Proc. of the* 8^{th} *Int. Conf. on Computer Systems and Applications (AICCSA)*. 2010.

[7] Kenneth Bannister, Gianni Giorgetti, and Eep K. S. Gupta. “Wireless Sensor Networking for “Hot” Applications: Effects of Temperature on Signal Strength, Data Collection and Localization”. In: *Proc. of the* 5^{th} *Workshop on Embedded Networked Sensors (HotEmNets)*. 2008.

[8] J. Berry, W.E. Hart, and C.A. Phillips. “Sensor Placement in Municipal Water Networks”. In: *J. Water* 131 (2003).

[9] Carlo Alberto Boano et al. “JamLab : Augmenting Sensornet Testbeds with Realistic and Controlled Interference Generation”. In: (2011), pp. 175–186.

[10] Carlo Alberto Boano et al. “The Impact of Temperature on Outdoor Industrial Sensornet Applications”. In: *IEEE Trans. on Industrial Informatics* 6.3 (2010).

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

[12] A. Brandt et al. *Applicability Statement: The use of the RPL protocol suite in Home Automation and Building Control draft-ietf-roll-applicability-home-building-12*. Internet-Draft. IETF, 2014.

[13] N. Burri, P. von Rickenbach, and R. Wattenhofer. "Dozer: Ultra-Low Power Data Gathering in Sensor Networks". In: *Proc. of the Int. Conf. on Information Processing in Sensor Networks (IPSN)*. 2007. DOI: 10.1109/IPSN.2007.4379705.

[14] D. Carlson et al. "Forwarder Selection in Multi-transmitter Networks". In: *Proc. of the Int. Conf. on Distributed Computing in Sensor Systems (DCOSS*. 2013. DOI: 10.1109/DCOSS.2013.73.

[15] Matteo Ceriotti et al. "Motes in the jungle: lessons learned from a short-term WSN deployment in the ecuador cloud forest". In: *Proc. of the* 4^{th} *Int. Workshop on Real-World Wireless Sensor Networks Applications (REALWSN)*. 2010.

[16] M. Ceriotti et al. "Is There Light at the Ends of the Tunnel? Wireless Sensor Networks for Adaptive Lighting in Road Tunnels". In: *Proc. of the Int. Conf. on Information Processing in Sensor Networks (IPSN)*. 2011.

[17] M. Ceriotti et al. "Monitoring Heritage Buildings with Wireless Sensor Networks: The Torre Aquila Deployment". In: *Proc. of the* 8^{th} *Int. Conf. on Information Processing in Sensor Networks (IPSN)*. 2009.

[18] A. Cerpa, N. Busek, and D. Estrin. *SCALE: A tool for simple connectivity assessment in lossy environments*. Tech. rep. UCLA, 2003.

[19] T. Clausen and P. Jacquet. *Optimized Link State Routing Protocol (OLSR)*. RFC 3626. IETF, 2003.

[20] T. Clausen, A. C. de Verdiere, and J. Yi. "Performance analysis of Trickle as a flooding mechanism". In: *Proc. of the IEEE Int. Conf. on Communication Technology (ICCT)*. 2013. DOI: 10.1109/ICCT.2013.6820439.

[21] Thomas Clausen, Ulrich Herberg, and Matthias Philipp. "A critical evaluation of the IPv6 Routing Protocol for Low Power and Lossy Networks (RPL)". In: *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob'11)*. Wuhan, China: IEEE, 2011, pp. 365–372. DOI: 10.1109/WiMOB.2011.6085374.

[22] C. Cobarzan, J. Montavont, and T. Noel. "Analysis and performance evaluation of RPL under mobility". In: *Proc. of the IEEE Int. Symp. on Computers and Communications (ISCC)*. 2014. DOI: 10.1109/ISCC.2014.6912471.

[23] U. M. Colesanti, S. Santini, and A. Vitaletti. "DISSense: An adaptive ultralow-power communication protocol for wireless sensor networks". In: *Proc. of the Int. Conf. on Distributed Computing in Sensor Systems (DCOSS)—Workshops*. 2011.

[24] *Contiki: The open source OS for the Internet of Things*. 2015. URL: http://contiki-os.org.

[25] Sebastien Dawans, Simon Duquennoy, and Olivier Bonaventure. “On link estimation in dense RPL deployments”. In: *37th Annual IEEE Conference on Local Computer Networks – Workshops* (Oct. 2012), pp. 952–955. DOI: 10.1109/LCNW.2012.6424087. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6424087.

[26] M. Doddavenkatappa, M. Chan, and B. Leong. “Splash: Fast Data Dissemination with Constructive Interference in Wireless Sensor Networks”. In: *USENIX Symp. on Networked Systems Design and Implementation (NSDI)*. USENIX, 2013. ISBN: 978-1-931971-00-3. URL: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/doddavenkatappa.

[27] M. Doddavenkatappa, M.C. Chan, and A.L. Ananda. “Indriya: A low-cost, 3D wireless sensor network testbed”. In: *Proc. of the Int. Conf. on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom)*. 2011.

[28] M. Doddavenkatappa and M. Choon. “P3: A Practical Packet Pipeline using synchronous transmissions for wireless sensor networks”. In: *Proc. of the Int. Conf. on Information Processing in Sensor Networks (IPSN)*. 2014. DOI: 10.1109/IPSN.2014.6846753.

[29] M. Dohler et al. *Routing Requirements for Urban Low-Power and Lossy Networks*. RFC 5548. IETF, May 2009.

[30] W. Du et al. “When Pipelines Meet Fountain: Fast Data Dissemination in Wireless Sensor Networks”. In: *Proc. of the Int. Conf. on Embedded Networked Sensor Systems (SenSys)*. Seoul, South Korea, 2015. ISBN: 978-1-4503-3631-4. DOI: 10.1145/2809695.2809721. URL: http://doi.acm.org/10.1145/2809695.2809721.

[31] A. Dunkels. *The ContikiMAC Radio Duty Cycling Protocol*. Technical Report T2011:13. Swedish Institute of Computer Science, Dec. 2011.

[32] Adam Dunkels et al. “Software-based on-line energy estimation for sensor nodes”. In: *Proceedings of the 4th workshop on Embedded networked sensors - EmNets ’07*. New York, New York, USA: ACM Press, 2007, p. 28. ISBN: 9781595936943. DOI: 10.1145/1278972.1278979. URL: http://portal.acm.org/citation.cfm?doid=1278972.1278979.

[33] Simon Duquennoy, Olaf Landsiedel, and Thiemo Voigt. “Let the Tree Bloom: Scalable Opportunistic Routing with ORPL”. In: *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys 2013)*. Rome, Italy, Nov. 2013.

[34] Simon Duquennoy et al. “Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH”. In: *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems - SenSys ’15*. New York, New York, USA: ACM Press, 2015, pp. 337–350. ISBN: 9781450336314. DOI: 10.1145/2809695.2809714. URL: http://dl.acm.org/citation.cfm?doid=2809695.2809714.

[35] EPANET. www.epa.gov/nrmrl/wswrd/dw/epanet.html.

[36] Antonio Escobar et al. “Competition: RedFixHop with Channel Hopping”. In: *Proceedings of the* 14^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Uppsala, Sweden, Feb. 2017.

[37] Dave Evans. *The Internet of Things: How the next evolution of the Internet is changing everything.* White Paper. Cisco, 2011.

[38] F. Ferrari et al. “Efficient Network Flooding and Time Synchronization with Glossy”. In: *Proc. of the ACM/IEEE Int. Conf. on Information Processing in Sensor Networks (IPSN).* 2011.

[39] F. Ferrari et al. “Low-power Wireless Bus”. In: *Proc. of the Int. Conf. on Embedded Networked Sensor Systems (SenSys).* 2012. ISBN: 978-1-4503-1169-4. DOI: 10.1145/2426656.2426658. URL: http://doi.acm.org/10.1145/2426656.2426658.

[40] Olfa Gaddour and Anis Koubâa. “RPL in a nutshell: A survey”. In: *Computer Networks, Elsevier* 56.14 (2012), pp. 3163–3178. DOI: 10.1016/j.comnet.2012.06.016. URL: http://linkinghub.elsevier.com/retrieve/pii/S1389128612002423.

[41] Wei Gan et al. “MERPL: A more memory-efficient storing mode in RPL”. In: *IEEE International Conference on Networks (ICON’13).* Singapore, Malaysia: IEEE, 2013, pp. 1–5. DOI: 10.1109/ICON.2013.6781985.

[42] Guillermo Gastón Lorente et al. “BMRF: Bidirectional Multicast RPL Forwarding”. In: *Ad Hoc Networks* 54 (2017), pp. 69–84. ISSN: 15708705. DOI: 10.1016/j.adhoc.2016.10.004.

[43] O. Gnawali and P. Levis. *The Minimum Rank with Hysteresis Objective Function.* RFC 6719. IETF, 2012.

[44] Omprakash Gnawali. “Performance of RPL under wireless interference”. In: *IEEE Communications Magazine* 51.12 (Dec. 2013), pp. 137–143. ISSN: 0163-6804. DOI: 10.1109/MCOM.2013.6685769. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6685769.

[45] Omprakash Gnawali et al. “Collection Tree Protocol”. In: *Proc. of the 7th ACM Conference on Embedded Networked Sensor Systems.* SenSys ’09. Berkeley, California: ACM, 2009, pp. 1–14. ISBN: 978-1-60558-519-2. DOI: 10.1145/1644038.1644040. URL: http://doi.acm.org/10.1145/1644038.1644040.

[46] Shuo Guo et al. “Opportunistic Flooding in Low-duty-cycle Wireless Sensor Networks with Unreliable Links”. In: *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking.* MobiCom ’09. Beijing, China: ACM, 2009, pp. 133–144. ISBN: 978-1-60558-702-8. DOI: 10.1145/1614320.1614336. URL: http://doi.acm.org/10.1145/1614320.1614336.

[47] Jonathan Hui and Richard Kelsey. *Multicast forwarding using trickle draft-ietf-roll-trickle-mcast-01.* Internet-Draft. IETF, 2012.

[48] Jonathan Hui and Richard Kelsey. *Multicast Protocol for Low power and Lossy Networks (MPL).* Internet-Draft. IETF, 2014.

[49] N.Q.V. Hung, H. Jeung, and K. Aberer. “An Evaluation of Model-Based Approaches to Sensor Data Compression”. In: *IEEE Trans. on Knowledge and Data Engineering* 25.11 (2013), pp. 2434–2447. ISSN: 1041-4347. DOI: 10.1109/TKDE.2012.237.

[50] Oana Iova, Fabrice Theoleyre, and Thomas Noel. “Stability and efficiency of RPL under realistic conditions in Wireless Sensor Networks”. In: *IEEE International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC’13)*. London, UK: IEEE, 2013, pp. 2098–2102. DOI: 10.1109/PIMRC.2013.6666490.

[51] O. Iova et al. “RPL: The Routing Standard for the Internet of Things... Or Is It?” In: *IEEE Communications Magazine* 54.12 (Dec. 2016), pp. 16–22. ISSN: 0163-6804. DOI: 10.1109/MCOM.2016.1600397CM.

[52] Timofei Istomin, Csaba Kiraly, and Gian Pietro Picco. “Is RPL ready for actuation? A comparative evaluation in a smart city scenario”. In: *Proceedings of the European Conference in Wireless Sensor Networks (EWSN’15)*. Porto, Portugal: Springer International Publishing, 2015, pp. 291–299. DOI: 10.1007/978-3-319-15582-1_22.

[53] Timofei Istomin et al. “Data Prediction + Synchronous Transmissions = Ultra-low Power Wireless Sensor Networks”. In: *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*. SenSys ’16. Stanford, CA, USA: ACM, 2016, pp. 83–95. ISBN: 978-1-4503-4263-6. DOI: 10.1145/2994551.2994558. URL: http://doi.acm.org/10.1145/2994551.2994558.

[54] C. Kiraly et al. “D-RPL: Overcoming memory limitations in RPL point-to-multipoint routing”. In: *2015 IEEE 40th Conference on Local Computer Networks (LCN)*. Oct. 2015, pp. 157–160. DOI: 10.1109/LCN.2015.7366295.

[55] JeongGil Ko et al. “ContikiRPL and TinyRPL: Happy together”. In: *Proceedings of the Workshop on Extending the Internet to Low Power and Lossy Networks (IPSN’11)*. Chicago, Illinois: ACM, 2011.

[56] JeongGil Ko et al. “DualMOP-RPL: Supporting multiple modes of downward routing in a single RPL network”. In: *Trans. on Sensor Networks, ACM* 11, 2.Article 39 (2015), 20 pages. DOI: 10.1145/2700261.

[57] JeongGil Ko et al. “Evaluating the performance of RPL and 6LoWPAN in TinyOS”. In: *Proceedings of the Workshop on Extending the Internet to Low Power and Lossy Networks (IPSN’11)*. Chicago, Illinois: ACM, 2011.

[58] Intel Research Lab. http://db.csail.mit.edu/labdata/labdata.html.

[59] O. Landsiedel, F. Ferrari, and M. Zimmerling. “Chaos: Versatile and Efficient All-to-all Data Sharing and In-network Processing at Scale”. In: *Proc. of the Int. Conf. on Embedded Networked Sensor Systems (SenSys)*. 2013. ISBN: 978-1-4503-2027-6. DOI: 10.1145/2517351.2517358. URL: http://doi.acm.org/10.1145/2517351.2517358.

[60] O. Landsiedel et al. “Low Power, Low Delay: Opportunistic Routing Meets Duty Cycling”. In: *Proc. of the Int. Conf. on Information Processing in Sensor Networks (IPSN)*. 2012. ISBN: 978-1-4503-1227-1. DOI: 10.1145/2185677.2185731. URL: http://doi.acm.org/10.1145/2185677.2185731.

[61] K. Leentvaar and J. Flint. “The Capture Effect in FM Receivers”. In: *IEEE Trans. on Communications* 24.5 (May 1976), pp. 531–539. ISSN: 0090-6778. DOI: 10.1109/TCOM.1976.1093327.

[62] P. Levis et al. *The Trickle Algorithm.* RFC 6206. IETF, 2011.

[63] Life Under Your Feet Project. lifeunderyourfeet.org/en/src.

[64] Roman Lim et al. “Competition: Robust Flooding using Back-to-Back Synchronous Transmissions with Channel-Hopping”. In: *Proceedings of the* 14^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN).* Uppsala, Sweden, Feb. 2017.

[65] Ramona Marfievici et al. “How Environmental Factors Impact Outdoor Wireless Sensor Networks: A Case Study”. In: *Proc. of the* 10^{th} *Int. Conf. on Mobile Ad-hoc and Sensor Systems (MASS).* 2013.

[66] Miklos Maroti and Janos Sallai. *Packet-level time synchronization (TEP 133).* http://www.tinyos.net/tinyos-2.1.0/doc/html/tep133.html. 2008.

[67] J. Martocci et al. *Building Automation Routing Requirements in Low-Power and Lossy Networks.* RFC 5867. IETF, June 2010.

[68] S. Moeller et al. “Routing Without Routes: The Backpressure Collection Protocol”. In: *Proc. of the Int. Conf. on Information Processing in Sensor Networks (IPSN).* Stockholm, Sweden, 2010. ISBN: 978-1-60558-988-6. DOI: 10.1145/1791212.1791246. URL: http://doi.acm.org/10.1145/1791212.1791246.

[69] Mobashir Mohammad, XiangFa Guo, and Mun Choon Chan. “Oppcast: Exploiting Spatial and Channel Diversity for Robust Data Collection in Urban Environments”. In: *Proceedings of the 15th International Conference on Information Processing in Sensor Networks.* IPSN ’16. Vienna, Austria: IEEE Press, 2016, 19:1–19:12. ISBN: 978-1-5090-0802-5. URL: http://dl.acm.org/citation.cfm?id=2959355.2959374.

[70] D. Moss and P. Levis. *BoX-MACs: Exploiting Physical and Link Layer Boundaries in Low-Power Networking.* Tech. rep. SING-08-00. Stanford Information Networks Group, 2008.

[71] Luca Mottola et al. “Not all wireless sensor networks are created equal: A comparative study on tunnels”. In: *ACM Trans. on Sensor Networks* 7.2 (2010).

[72] R. Musaloiu-E., C.-J.M. Liang, and A. Terzis. “Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks”. In: *Proc. of the Int. Conf. on Information Processing in Sensor Networks (IPSN).* 2008. ISBN: 978-0-7695-3157-1. DOI: 10.1109/IPSN.2008.10. URL: http://dx.doi.org/10.1109/IPSN.2008.10.

[73] Beshr Al Nahas and Olaf Landsiedel. “Competition: Towards Low-Power Wireless Networking that Survives Interference with Minimal Latency”. In: *Proceedings of the* 14^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN).* Uppsala, Sweden, Feb. 2017.

[74] Beshr Al Nahas et al. “Low-power listening goes multi-channel”. In: *Proceedings - IEEE International Conference on Distributed Computing in Sensor Systems, DCOSS 2014.* IEEE, May 2014, pp. 2–9. ISBN: 9781479946198. DOI: 10.1109/DCOSS.2014.33. URL: http://ieeexplore.ieee.org/document/6846139/.

[75] C. Noda et al. “On the Scalability of Constructive Interference in Low-Power Wireless Networks”. In: *Proc. of the European Conf. on Wireless Sensor Networks (EWSN).* 2015. ISBN: 978-3-319-15582-1. DOI: 10.1007/978-3-319-15582-1_17. URL: http://dx.doi.org/10.1007/978-3-319-15582-1_17.

[76] Tony O’donovan et al. “The GINSENG System for Wireless Monitoring and Control: Design and Deployment Experiences”. In: *ACM Trans. Sen. Netw.* 10.1 (Dec. 2013), 4:1–4:40. ISSN: 1550-4859. DOI: 10.1145/2529975. URL: http://doi.acm.org/10.1145/2529975.

[77] George Oikonomou and Iain Phillips. “Stateless multicast forwarding with RPL in 6LowPAN sensor networks”. In: *IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM’12).* Lugano, Switzerland: IEEE, 2012, pp. 272–277. DOI: 10.1109/PerComW.2012.6197494.

[78] Fredrik Osterlind et al. “Cross-level sensor network simulation with COOJA”. In: *IEEE Conference on Local Computer Networks (LCN’06).* Tampa, Florida: IEEE, 2006, pp. 641–648. DOI: 10.1109/LCN.2006.322172.

[79] Marko Paavola and Kauko Leiviskä. “Wireless Sensor Networks in Industrial Automation”. In: *Factory Automation.* Ed. by Javier Silvestre-Blanes. InTech, 2010, pp. 201–221. ISBN: 978-953-307-024-7. URL: http://www.intechopen.com/books/factory-automation/wireless-sensor-networks-in-industrial-automation.

[80] T. Palpanas et al. “Streaming Time Series Summarization Using User-Defined Amnesic Functions”. In: *IEEE Trans. on Knowledge and Data Engineering* 20.7 (2008).

[81] S. Papadimitriou, J. Sun, and C. Faloutsos. “Streaming Pattern Discovery in Multiple Time-Series.” In: *Proc. of the Int. Conf. on Very Large Data Bases (VLDB).* 2005.

[82] Charles Perkins, Elizabeth Belding-Royer, and Samir Das. *Ad hoc On-Demand Distance Vector (AODV) routing.* RFC 3561. IETF, 2003.

[83] Stig Petersen and Simon Carlsen. “WirelessHART versus ISA100.11a: the format war hits the factory floor”. In: *Industrial Electronics Magazine, IEEE* 5.4 (2011), pp. 23–34. ISSN: 1932-4529. DOI: 10.1109/MIE.2011.943023.

[84] T. Phinney, P. Thubert, and RA. Assimiti. *RPL applicability in industrial networks draft-ietf-roll-rpl-industrial-applicability-02.* Internet-Draft. IETF, 2013.

[85] K. Pister et al. *Industrial Routing Requirements in Low-Power and Lossy Networks.* RFC 5673. IETF, Oct. 2009.

[86] D. Popa et al. *Applicability Statement for the Routing Protocol for Low Power and Lossy Networks (RPL) in AMI Networks draft-ietf-roll-applicability-ami-11.* Internet-Draft. IETF, 2015.

[87] D. Puccinelli et al. “Broadcast-free Collection Protocol”. In: *Proc. of the Int. Conf. on Embedded Network Sensor Systems (SenSys)*. 2012. ISBN: 978-1-4503-1169-4. DOI: 10.1145/2426656.2426660. URL: http://doi.acm.org/10.1145/2426656.2426660.

[88] Ion Emilian Radoi, Aditi Shenoy, and D.K. Arvind. “Evaluation of Routing Protocols for Internet-Enabled Wireless Sensor Networks”. In: *The Eighth International Conference on Wireless and Mobile Communications (ICWMC’12)*. Venice, Italy: IARIA, 2012, pp. 56–61.

[89] U. Raza et al. “Practical Data Prediction for Real-World Wireless Sensor Networks”. In: *IEEE Trans. on Knowledge and Data Engineering* 27.8 (2015), pp. 2231–2244.

[90] M. A. Razzaque, C. Bleakley, and S. Dobson. “Compression in Wireless Sensor Networks: A Survey and Comparative Evaluation”. In: *ACM Trans. Sensor Networks* 10.1 (Dec. 2013), 5:1–5:44. ISSN: 1550-4859. DOI: 10.1145/2528948. URL: http://doi.acm.org/10.1145/2528948.

[91] Markus Schuß et al. “A Competition to Push the Dependability of Low-Power Wireless Protocols to the Edge”. In: *Proceedings of the* 14^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Uppsala, Sweden: Feb. 2017.

[92] Kannan Srinivasan et al. “An empirical study of low-power wireless”. In: *ACM Trans. on Sensor Networks* 6.2 (2010).

[93] Kannan Srinivasan et al. “SWAT: enabling wireless network measurements”. In: *Proc. of the* 6^{th} *Int. Conf. on Embedded Networked Sensor Systems (SenSys)*. 2008.

[94] Kannan Srinivasan et al. “The β-factor: Measuring Wireless Link Burstiness”. In: *Proc. of the* 6^{th} *Int. Conf. on Embedded Networked Sensor Systems (SenSys)*. 2008.

[95] M. Suzuki, Y. Yamashita, and H. Morikawa. “Low-Power, End-to-End Reliable Collection Using Glossy for Wireless Sensor Networks”. In: *Proc. of the Vehicular Technology Conference (VTC Spring)*. June 2013. DOI: 10.1109/VTCSpring.2013.6692624.

[96] Texas Instruments. “CC2420: 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver”. In: (2013). URL: http://www.ti.com/product/CC2420/technicaldocuments.

[97] John Thelen and Daan Goense. “Radio wave propagation in potato fields”. In: *Proc. of the* 1^{st} *Workshop on Wireless Network Measurements*. 2005.

[98] *TinyOS official website*. 2013. URL: http://tinyos.net.

[99] D. Tulone and S. Madden. “An energy-efficient querying framework in sensor networks for detecting node similarities”. In: *Proc. of the Int. Conf. on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*. 2006.

[100] Malisa Vucinic, Bernard Tourancheau, and Andrzej Duda. “Performance comparison of the RPL and LOADng routing protocols in a Home Automation scenario”. In: *IEEE Wireless Communications and Networking Conference (WCNC’13)*. Shanghai, China: IEEE, 2013, pp. 1974–1979. DOI: 10.1109/WCNC.2013.6554867.

[101] Y. Wang et al. “Exploiting Constructive Interference for Scalable Flooding in Wireless Networks”. In: *IEEE/ACM Trans. on Networking* 21.6 (Dec. 2013), pp. 1880–1889. ISSN: 1063-6692. DOI: 10.1109/TNET.2013.2238951.

[102] Y. Wang et al. “TriggerCas: Enabling wireless constructive collisions”. In: *Proc. of the Int. Conf. on Computer Communications (INFOCOM)*. 2013. DOI: 10.1109/INFCOM.2013.6566819.

[103] H. Wennerstrom et al. “A long-term study of correlations between meteorological conditions and 802.15.4 link performance”. In: *Proc. of* 10^{th} *Int. Conf. on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*. 2013.

[104] Geoffrey Werner-Allen et al. “Fidelity and yield in a volcano monitoring sensor network”. In: *Proc. of the* 7^{th} *Symp. on Operating Systems Design and Implementation (OSDI)*. 2006.

[105] Tim Winter et al. *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. RFC 6550. IETF, 2012.

[106] D. Yuan, M. Riecker, and M. Hollick. “Making ‘Glossy’ Networks Sparkle: Exploiting Concurrent Transmissions for Energy Efficient, Reliable, Ultra-Low Latency Communication in Wireless Control Networks”. In: *Proc. of the European Conf. on Wireless Sensor Networks (EWSN)*. 2014. ISBN: 978-3-319-04651-8. DOI: 10.1007/978-3-319-04651-8_9. URL: http://dx.doi.org/10.1007/978-3-319-04651-8_9.

[107] J. Zhang et al. “RFT: Identifying Suitable Neighbors for Concurrent Transmissions in Point-to-Point Communications”. In: *Proc. of the Int. Conf. on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*. 2015. ISBN: 978-1-4503-3762-5. DOI: 10.1145/2811587.2811595. URL: http://doi.acm.org/10.1145/2811587.2811595.

[108] Peilin Zhang, Olaf Landsiedel, and Oliver Theel. “MOR: Multichannel Opportunistic Routing for Wireless Sensor Networks”. In: *EWSN: Proceedings of the International Conference on Embedded Wireless Systems and Networks*. Feb. 2017.

[109] Zigbee Alliance. *ZigBee-2007 PICS and Stack Profiles*. 2008.

[110] Marco Zimmerling et al. “Adaptive Real-Time Communication for Wireless Cyber-Physical Systems”. In: *ACM Transactions on Cyber-Physical Systems* 1.2 (Feb. 2017), pp. 1–29. ISSN: 2378962X. DOI: 10.1145/3012005. URL: http://dl.acm.org/citation.cfm?doid=3015781.3012005.