Routing in Wireless Sensor Networks: An Experimental Evaluation of RPL

Master’s Thesis

Veronika Maria Bauer
Routing in Wireless Sensor Networks: An Experimental Evaluation of RPL

Master’s Thesis

Submitted by: Veronika Maria Bauer
Submitted on: 30th December, 2010

Supervisors: Prof. Dr. Bernhard Bauer, Universität Augsburg
            Prof. Dr. Alexander Knapp, Universität Augsburg

External Advisors: Dr. Emmanuel Baccelli, INRIA
                  Prof. Dr. Thomas Clausen, Ecole Polytechnique
Acknowledgements

This work was supported by INRIA, the Institut National de Recherche en Informatique et en Automatique.

First of all, I’d like to thank my advisors Prof. Dr. Bernhard Bauer and Prof. Dr. Alexander Knapp from the University of Augsburg for accepting to supervise my masters thesis abroad and the freedom and support they gave me for this project. This freedom and support were also granted by Dr. Emmanuel Baccelli and Prof. Dr. Thomas Clausen who accepted me in their research group and advised me during my internship at the Laboratoire d’Informatique (LIX) of the Ecole Polytechnique, Paris. I’d like to thank them, as well as the whole research group of Hipercom@LIX for immediately adopting me in the team and integrating me into the group’s research activities. Many thanks also to Marie-Jeanne Gaffard for her infallible advice on bureaucratic matters which was a great help in organizing the details of my stay at LIX.

Special thanks go to my colleagues Juan Antonio Cordero Fuertes, Ulrich Herberg, Matthias Phillip, and Dr. Georg Wittenburg. It was a privilege to work with them in a highly motivating atmosphere and to profit from their competent feedback. The considerable effort imposed by the experimental setups was mitigated by the support of Matthias Phillip who was managing the testbed access. Thanks also to Gidon Ernst for sharing his experience with Contiki.

Last, but not least, I want to thank my family and my friends who have motivated and encouraged me during the work on my masters thesis. I’m especially grateful to everyone being supportive during the difficult times around my accident, and to all of you who made my stay in France such an enjoyable and enriching experience. This goes in particular for Emmanuel Baccelli, Victor Chahuneau, Juan Antonio Cordero Fuertes, Marie Doutre-Roussel, Marie-Jeanne Gaffard, Ulrich Herberg, Ombline Lafont, Olivier Letellier, Cécile Ozanan, Matthias Philipp, Sonja Steiner, Céline de Vallée, Jaques Verrue, and Georg Wittenburg.
Abstract

Enabled by advances in communication technologies and electronics Wireless Sensor Networks (WSNs) have opened up a new field of application and research in the area of ad-hoc networks. Being composed of large numbers of small nodes, which are relatively restricted in terms of energy, memory, as well as computational and transmission power, WSNs are seen as a potentially versatile, inexpensive tool for measurements or monitoring of diverse phenomena. Visions for their application range from energy distribution in so-called 'Smart Grids' over environmental monitoring of glaciers, habitats, water and air pollution as well as warning systems to detect indicators for forest fires or volcano activity before the disaster occurs. Further applications are being developed for surveillance, home automation, medical, agricultural and industrial applications.

Key features which make WSNs applicable in these domains – and even preferable to conservative deployments in which clients report data directly to a centralized access point – are the low cost of the single sensor devices, which make a deployment feasible, as well as their un-intrusiveness in terms of size and radiation. WSNs are especially attractive to the scientific community, as they allow to constantly gain measurements over the lifetime of the deployment, at a resolution and scale which could not be achieved before.

This thesis evaluates RPL, the IPv6 Routing Protocol for Low Power and Lossy Networks, which in 2010 is under specification by the ROLL working group of the Internet Engineering Task Force (IETF). RPL has matured to a stage to be implemented and tested, thus allowing to gain experience with the system which verifies the soundness of the specification. Implementations for network simulation already exist for WSNs with up to thousand nodes. However, there is a ‘reality gap’ between simulated and real world environments due to the many assumptions and approximations stated in the simulation models, concerning, e.g., the internal state of hardware or the properties of the radio transmission medium. Simulations do provide valuable indications about a system's behavior, but can not replace experiments of the system on real target devices. To gain real world experience by testing the protocol's behavior under more realistic circumstances, the aim of this thesis was to create one of the first implementations of RPL on typical
sensor node hardware and use it to evaluate the behavior of RPL, as well as to verify the outcomes of the simulations against the results of experiments.
## Contents

1 Introduction .......................... 3
   1.1 Motivation .......................... 4
   1.2 Problem Statement ................. 5
   1.3 Structure of the Document ....... 5

2 Wireless Sensor Networks ............ 7
   2.1 Applications of WSNs .............. 8
   2.2 Architecture of WSNs .............. 10
   2.3 Wireless Networking .............. 13
      2.3.1 Wireless Networks ............. 13
      2.3.2 Routing in WSNs ............... 16
   2.4 Protocol Standardisation Process 19
      2.4.1 The Internet Engineering Task Force .... 20
      2.4.2 The ROLL Working Group ...... 20

3 Description of RPL .................... 23
   3.1 Topology Formation ................. 23
   3.2 Traffic Patterns .................... 28
   3.3 Data Dissemination ................. 29
      3.3.1 The Trickle Algorithm ............ 30
      3.3.2 Application in RPL .............. 34
   3.4 Data Collection .................... 34

4 Implementation ........................ 39
   4.1 Hardware Platforms ................. 39
      4.1.1 The MSB430 Scatterweb Mote .... 39
      4.1.2 The WSN430 Sensor Board ..... 41
   4.2 Contiki OS .......................... 41
      4.2.1 System Overview ................ 41
      4.2.2 The Process Model .............. 42
4.2.3 Networking .................................................. 44
4.3 Implementation of RPL ...................................... 44
  4.3.1 Parameters .............................................. 45
  4.3.2 Development Cycle .................................... 47

5 Experimental Evaluation ...................................... 49
  5.1 Observations on RPL ........................................ 49
  5.2 Setup of the Experiments .................................. 53
    5.2.1 The MSB430 Setup ................................... 53
    5.2.2 The SensLAB Setup ................................... 54
  5.3 Measurements ............................................... 55
    5.3.1 Control Plane Metrics ............................... 55
    5.3.2 Data Plane Metrics ................................. 60
  5.4 Results ..................................................... 62
    5.4.1 Control Plane Metrics ............................... 62
    5.4.2 Data Plane Metrics ................................. 65

6 Conclusion .................................................... 71
List of Figures

2.1 Multi-hop Communication over a WSN ......................... 8
2.2 The PermaSense Deployment .................................... 9
2.3 Point-to-Point Traffic Pattern .................................. 12
2.4 Multipoint-to-Point Traffic Pattern ........................... 13
2.5 Point-to-Multipoint Traffic Pattern ............................ 14
2.6 Ethernet ......................................................... 15
2.7 Multi-hop Wireless Network ...................................... 15
2.8 Asymmetric Communication between Wireless Nodes .......... 15

3.1 Example Network Topology ........................................ 24
3.2 Topology Formation, Step 1 ........................................ 25
3.3 Topology Formation, Step 2 ........................................ 26
3.4 Topology Formation, Step 3 ........................................ 27
3.5 Topology Formation, Step 4 ........................................ 28
3.6 Topology Formation, Resulting DODAG ......................... 29
3.7 Implementation of the Trickle Algorithm ......................... 33
3.8 The Interaction between RPL and Trickle ......................... 35
3.9 The Internal Trickle Mechanism .................................. 36

4.1 MSB430 Scatterweb Mote .......................................... 40
4.2 Trickle Implementation using Contiki ............................ 43

5.1 Example Network Topology ........................................ 51
5.2 Example Network Topology with DODAG ......................... 52
5.3 MSB430 Experimental Topology .................................. 55
5.4 SENSLAB1 Experimental Topology .............................. 56
5.5 SENSLAB1 Chain and Cluster Experimental Topologies ........ 58
5.6 SENSLAB2 Experimental Topology .............................. 59
5.7 Data Flow on the MSB430 Experimental Topology ............. 61
5.8 SENSLAB Evaluation of RPL’s Signal Traffic on a Cluster Topology .... 63
List of Figures

5.9 SENS LAB Snapshot of Node States ........................................ 64
5.10 SENS LAB Node Rank vs. Signal Traffic ............................. 65
5.11 MSB430 PDR vs. Data Rate .............................................. 66
5.12 MSB430 PDR vs. Hop Count ............................................. 68
5.13 SENS LAB Evaluation of PDR vs. Load ............................... 69
List of Tables

3.1 Parameters of the Trickle Algorithm .................................. 32
4.1 Parameters of the Implementation .................................. 45
5.1 SENSLAB: Parameters of the First Series ......................... 57
5.2 SENSLAB: Parameters of the Second Series ...................... 57
5.3 MSB430: Parameters of the Experiments .......................... 60
1 Introduction

Recently, Wireless Sensor Networks (WSNs) have become a subject of interest for many different applications. Enabled by advances in communication technologies and electronics, which allow for developing small devices at a cost which is much lower than the one of traditional sensors [56], WSNs have opened up a new field of application and research in the area of ad-hoc networks. Being composed of large numbers of small nodes, which are relatively restricted in terms of energy, memory, as well as computational and transmission power, WSNs are seen as a potentially versatile, inexpensive tool for measurements or monitoring of diverse phenomena [56]. Visions for their application range from energy distribution in so-called ‘Smart Grids’ over environmental monitoring of glaciers [20, 32], habitats [31], water [27] and air pollution as well as warning systems to detect indicators for forest fires or volcano activity before the disaster occurs [41, 53]. Further applications are being developed for military and civil surveillance [16], home automation [24], medical [39], agricultural [26] and industrial applications [3].

Key features which make WSNs applicable in these domains – and even preferable to conservative deployments in which clients report data directly to a centralized access point – are the low cost of the single sensor devices, which make a deployment feasible, as well as their un-intrusiveness in terms of size and radiation. WSNs are especially attractive to the scientific community, as they allow to constantly gain measurements over the lifetime of the deployment, at a resolution and scale which could not be achieved before [31]. Two sorts of deployments are distinguished in WSNs [56]: nodes can be distributed in a structured way, by putting them in pre-planned positions. However, in some cases this might not be possible, as, e.g., the terrain which is supposed to be monitored is intoxicated and would endanger the engineers deploying the network. In such circumstances, WSNs allow for an unstructured deployment where nodes are randomly distributed over the desired area by, e.g., dropping them out of an helicopter as in the volcano monitoring system presented in [41]. Unstructured deployments tend

\footnote{Inexpensive at the time of writing in 2010 means around 100 Euros for a Scatterweb mote - the type of device used for this work. However, the large-scale deployments envisioned by [5] assume a still much lower price of around 1USD.}
to need a higher density of nodes than structured ones, as their placement can not be optimized to achieve a good coverage of the environment [56].

1.1 Motivation

The currently deployed WSNs are small-scale, with the number of nodes ranging from around a few nodes in home automation [24] and the water monitoring system in [27] to a scale of tens of nodes in the PermaSense alpine project [20] and 32 in the habitat monitoring system presented in [31]. However, the envisioned applications demand a far larger scale of network to be deployed, with the number of nodes reaching several thousands [56]. One challenge raised by this vision is efficient communication in WSNs. Nodes usually run on batteries and are therefore limited in their power resources. This imposes a hard limit on the lifespan of a WSN. Since radio communication is the most expensive action a node performs [28], communication protocols for WSNs need to minimize the number of times a node needs to communicate. Another challenge is the scalability of a communication protocol in the face of dense WSNs with thousands of nodes, which all should be configured and managed properly. The envisioned dimensions of WSNs call for them to be self-organizing to the degree that interaction with them can be limited to a few nodes only.

To make the information from the WSN accessible remotely, they need to be integrated with a network of larger scale, possibly the Internet. Furthermore, an application could require different WSNs to communicate with each other. Currently, this is not easily possible as there exist numerous proprietary hardware platforms and protocols for WSNs which raise compatibility issues, preventing WSNs to be interoperable to each other or to the Internet [6].

Research Questions

The following research questions are thus imposed by the characteristics of WSNs: How can communication in WSNs be designed to be efficient and scalable to huge network sizes whilst taking into account the limitations of the hardware? Is it possible to develop standard communication protocols which allow for interoperability between WSNs and other networks? Do the proposed protocols work well in realistic environments?

Representatives of industry and academia have started to tackle these questions by working in the framework of the IETF [23] and the IPSO Alliance [6] on specifying a standardized protocol for routing in WSNs: RPL, the IPv6 Routing Protocol for Low Power and Lossy Networks. The goal is to design an IPv6-based routing protocol which
would allow a full integration of WSNs with IP-based networks and applications [22]. At the same time, the issue of resource preservation in WSNs should be addressed. The basic mechanisms of RPL have matured enough to be tested in simulations. Now missing is practical evidence concerning RPL’s performance on typical hardware devices in realistic settings.

1.2 Problem Statement

This manuscript evaluates RPL which in 2010 is under specification by the ROLL working group of the IETF. RPL has matured to a stage to be implemented and tested, thus allowing to gain experience with the system which verifies the soundness of the specification. Implementations for network simulation of the basic mechanisms already exist [11, 54] for WSNs with up to thousand nodes. However, there is a ‘reality gap’ between simulated and real world environments due to the many assumptions and approximations used in the simulation models, concerning, e.g., the internal state of hardware or the properties of the radio transmission medium. Simulations do provide valuable indications about a system’s behavior, but can not replace experiments of the system on real target devices. To gain real world experience by testing the protocol’s behavior under more realistic circumstances, the aim of this manuscript was to create one of the first implementations of RPL on typical sensor node hardware and use it to evaluate the behavior of RPL as well as to verify the outcomes of the simulations against the results of experiments.

1.3 Structure of the Document

Chapter 2 sets out with an overview of the Wireless Sensor Network application space, describes existing deployments thereof, and introduces their particularities in networking and routing. It further briefly presents the IETF and the standardization process. Chapter 3 introduces RPL, and the algorithms used by it. Subsequently, Chapter 4 describes the hardware and software platforms used for the reference implementation, as well as the protocol configuration chosen for the experiments. Chapter 5 describes the hardware setup of the experiments and the metrics which were measured before presenting the outcomes of the measurements. Chapter 6 presents future work and concludes the manuscript.
2 Wireless Sensor Networks

Wireless Sensor Networks (WSNs) open up a vast space of application and research [38]. Due to this variety, the characteristics of WSNs may differ significantly from one instance to another\(^1\). However, early research projects have established the following de-facto definition of WSNs which is assumed in this work.

WSNs are composed of a large number of small sensor nodes, which can be randomly distributed in any environment. These devices are communicating with each other over a low power wireless communication medium and are severely restricted in their resources. These restrictions are in terms of energy, which usually is furnished by batteries, transmission and computing power, as well as the memory available, which is available in the order of magnitude of tens of KBytes. Due to these restrictions, nodes in WSNs need to collaborate to establish a multi-hop network in which data is transmitted over several nodes to one or more sink nodes. Figure 2.1 illustrates this mechanism. Direct communication between all nodes and the sink node is avoided to save the nodes’ energy resources, which are most strained when emitting maximum strength radio signals. The collaborative aspect, as well as the lack of predefined and planned infrastructure, allows for WSNs to be classified as ad-hoc networks, which are defined in the following way [36]:

*An ad-hoc network is the cooperative engagement of a collection of mobile nodes without the required intervention of any centralized access point or existing infrastructure.*

However, WSNs are a special case, in terms of their limitation in resources which tends to be very severe in WSNs, as well as the large number of nodes deployed in a high density. At the time of writing, WSNs are deployed in relatively static environments, where nodes do not usually change their positions. Therefore, the mobile component of the definition is not yet taken into account in WSN applications.

The following Sections introduce the application space of WSNs and discuss their characteristics in more detail before exploring the entailed particularities of networking

\(^1\)These differences, which, e.g., include the way of WSN deployment, the hardware which is suitable, or the presence or absence of network infrastructure, build up a design space which is further elaborated in [38].
Figure 2.1: The Figure shows multi-hop communication over a WSN which is connected with an external network over a sink node. The circles symbolize sensor nodes. The sending node is denoted with a filled circle, and the sink node with a double circle. The path of the data is represented by the plain lines, whilst the dashed lines indicate the connectivity between the different sensor nodes.

in WSNs. The Chapter ends with a brief overview of the Internet Engineering Task Force (IETF), the standards body which currently specifies the RPL routing protocol.

2.1 Applications of WSNs

WSNs are designed to be the infrastructure for a broad spectrum of sensing applications [56]. Their use cases are highly diverse and range from structural and environmental monitoring, home automation, over health care over to vehicle tracking [5, 28]. The following paragraphs present different application scenarios of WSNs.

Environmental Monitoring

An example of a WSN deployment for environmental monitoring is PermaSense\(^2\), a joint project of ETH Zurich, and the Universities of Zurich and Basel, Switzerland [20]. To continuously observe the phenomenon of permafrost in its harsh and inaccessible alpine environment, two WSNs have been deployed. After the first deployment, researchers

\(^2\)PermaSense http://www.permasense.ch/, accessed Jul 21, 2010
2.1. Applications of WSNs

drew the conclusions that robust synchronisation as well as low-power routing algorithms are crucial for achieving a stable network topology as well as prolonging the network’s lifetime. Both is even more desirable under the aspect that replacement of failing nodes is hard due to their environment and therefore undesirable. Figure 2.2 shows the topology of the deployment at Matterhorn, Switzerland.\(^3\)

![Logical and physical topology of the PermaSense deployment at Matterhorn, Switzerland](image)

Figure 2.2: Logical and physical topology of the PermaSense deployment at Matterhorn, Switzerland\(^4\)

Around 25 nodes are deployed at different positions at the mountain. At the highest point, a base station with solar panels is installed. It serves as sink node, which means that all sensor nodes report their results to it. Furthermore, the sink is able to relay the measurements to a remote server. However, in order to save power at the single sensor nodes, only the ones with a reliable connection to the sink send their results directly to it. The other nodes try to send their messages to their neighbors, which might have a good connection to the sink. To summarize, the architecture of the PermaSense deployment consists of three tiers: sensor nodes which are deployed as a WSN, with a base station

---

\(^3\)Live data to be found at http://data.permasense.ch/topology.html, accessed July 26, 2010
that collects the measurements and sends them to a remote server. Integration between different network types therefore is required by this deployment.

**Home Automation**

[24] proposes a remote controlled home automation system implemented with a WSN in combination with mobile phone interaction via GPRS. Several sensors are deployed in bathroom, living room, bedroom, and study of a model house. They are used to observe the status of doors and windows, as well as to detect gas and smoke. Weather conditions are also monitored and the results are subsequently used to control air conditioning and heating. The owner of the house receives information and can interact with the system remotely by using their mobile phone.

**Volcano Monitoring**

[27] reports experience from an unstructured WSN deployment for volcano hazard monitoring, where the sensor nodes were dropped on the area from a helicopter. The nodes were installed at Mount St. Helen in distances up to 2 km and collect and report real-time seismic, infrasonic, lightning, GPS raw data to a gateway. The environmental influences on the system were harsh, as it was exposed to rain, snow, and ice.

### 2.2 Architecture of WSNs

Despite of the application domains being very diverse, the software and hardware architectures used in these deployments tend to be very similar. WSNs are envisioned to feature a large number of nodes, deployed in high density. A WSN can feature one or more sink nodes, which collect the data acquired by the sensor nodes and might function as gateways to external networks, as for example the Internet. The typical hardware used are mote platforms with a micro-controller featuring memory in the order of magnitude of tens of KB which is split into ROM and RAM [40], a low power radio device [9], and circuitry supporting low power standby modes to prolong the battery’s lifetime [42].

As WSNs are designed to operate unattendedly [56] they have to address the issues imposed by:

1. limited power supply,
2. severely constrained memory,
3. self organization, and
2.2. Architecture of WSNs

4. Lossy wireless communication.

These key factors form the fundamental constraints which have to be taken into account when designing wireless embedded systems and networks. The following paragraphs detail the issues connected to them.

1) **Limited power supply**: usually sensor nodes are battery powered. Especially in networks with high density or deployed in inaccessible locations it is undesirable or impossible to replace a node’s battery once it is exhausted. An alternative for powering sensor nodes would be to scavenge energy from the given environment, e.g., with solar panels in open spaces, or nano-generators transforming movement into electricity. Independently from the power source, energy saving is important in WSNs as the network lifetime is directly influenced by the power resources of the nodes.

2) **Severely constrained memory**: to reduce costs and energy consumption, memory on the nodes is kept small, in the order of tens of KBytes. [28] describes this restriction in more detail.

3) **Self organization**: many factors in WSN deployments make self organization desirable. Unstructured deployments with up to thousands of nodes would not be feasible if the network would not be able to configure itself and to react to changes in its structure which might be caused by node failures or environmental influences. Limiting the interaction with a WSN to one or a few sink nodes thus brings a significant gain in managing them.

4) **Lossy wireless communication**: wireless communication can be disturbed by interferences from the environment. This factor is elaborated in detail in Section 2.3.

**Traffic Patterns**

The tasks of WSNs usually involve sending information to the nodes in the network and extracting the data they have acquired. Due to their special application domain as measurement tools, WSNs have additional traffic patterns different compared to conventional point-to-point, unicast, data traffic [48]. Point-to-point describes the a pattern of communication between a designated sender and receiver. This type of traffic can occur in WSNs in two ways: firstly, one sensor node might be requesting measurements from another one somewhere in the network. In this case, which is shown in Figure 2.3, it is likely that due to the size of WSNs this request and the response have to pass via intermediate sensor nodes. Secondly, the point-to-point traffic pattern could be used from outside the WSN to prompt measurements from a specific node. Harvesting the data measured within the WSN requires a collection protocol which draws information from many nodes and forwards it to one or more sinks in a multipoint-to-point fashion.
Figure 2.3: Point-to-point traffic pattern. The node represented with a bold circle is requesting information from the node represented by the filled circle. As these nodes are distant from each other, the communication has to pass via intermediate nodes.

Figure 2.4 shows this traffic pattern. Data collection protocols need not necessarily be reliable [28]. This is due to the fact that a lot of the measurements tend do be threshold based, so a single node’s results being lost do not severely impact the results. This is particularly true in settings where measurements are aggregated in the forwarding process to save resources in terms of transmissions. Which level of reliability is required of data collection protocol differs depending on the application space.

WSNs need to be reprogrammable, allowing e.g. for thresholds to be changed, or different sensors to be sampled. Therefore, a data dissemination protocol must be provided as well, allowing information to be injected and distributed in the network, starting from one or more sink nodes. This traffic pattern is point-to-multipoint, and, as shown in

---

5For example in a WSN monitoring forests to prevent fires it is important to know if the temperature in a forest has suddenly raised significantly, but not so important to know exactly by how many degrees.

6Intermediate nodes might collect data from their neighbors over a certain time span to process and aggregate it before transmitting it further. In this way, the intermediate node sends only one message with the aggregated data instead of forwarding all messages which it has received from its neighbors.
2.3. Wireless Networking

The following Sections introduce principles of wireless networking. This is necessary, as the wireless transmission media introduce characteristics which invalidate several assumptions taking for granted in wired networking and therefore influence the design of networking protocols. The issues raised by properties of wireless transmission are illustrated in the following Section, before introducing the concept of routing and its particularities in wireless networks.

2.3.1 Wireless Networks

Many assumptions valid in wired networking can not be relied on in wireless networks. The most significant differences are sketched out in the following Section, according to [7] where the interested reader can find further explanations for the observed phenomena.
In networking, the notion of neighborhood is significant as it answers the question from which node another one can be reached directly. This is important information as soon as communication in the network is to take place. In an Ethernet [17], each node has knowledge about the link it is connected to. Nodes have a unique and complete view of the link local network, which means that all nodes have the same information about their neighborhood. Due to this fact, communication in an ethernet network is symmetric, as well as transitive. Figure 2.6 illustrates the transitive communication setting: if node 1 can hear\(^7\) node 4, this means that also node 2 and 3 can hear it sending its packets.

In a wireless network, the assumptions of symmetry and transitivity do not longer hold. Instead, as depicted by Figure 2.7, communication has become intransitive. This means, that even though node 2 can hear both, node 1 and 3, these two can not hear each other. Therefore, in this case each node may have different information about its neighborhood.

As mentioned above, communication in wired networks can be assumed to be symmetrical. This means that if Machine 1 can receive packets from Machine 2, this is true also vice versa. Conversely, in wireless networks, this assumption is no longer valid as

---

\( ^7 \) ‘Hearing’ in this context means being able to send packets to and receive packets from the other node. This relationship is visualized by the overlapping circles in the diagram.
communication can be observed to be asymmetrical. Figure 2.8 illustrates this: Nodes 1 and 2 have radio transmission ranges which overlap only partially. Node 1 is within the transmission range of Node 2, which means that it can directly receive packets sent from 2. However, the transmission range of 1 is not reaching 2, so 2 is unable to receive packets from 1, and might well be unaware of 1’s existence.

A third assumption generally holds in wired networks: stability of a node’s neighborhood. Changes are rare, as cables are usually not continuously plugged in and unplugged.
Furthermore, the communication link has a high fidelity. In wireless networks, the communication medium does not follow the binary characteristic of its wired counterpart. Instead, signal strength might vary due to energy levels, changes in the environment might interfere with a node’s signal, or node mobility might cause changes in the network topology. As a result, a node’s neighborhood in wireless ad-hoc networks might be constantly changing, causing communication to be time-variant.

2.3.2 Routing in WSNs

Routing is the process of enabling data transfer over a network from a source to a destination machine. This is achieved in a two-step process: First, paths have to be established in the network along which data traffic can then be forwarded if the sender and receiver are not on the same link. Once the paths have been selected, data traffic is forwarded from one endpoint of the transmission via intermediate nodes to the other endpoint. Routing algorithms are used to determine the paths the data will take and should fulfill the following properties: the routes should be chosen such that data reaches its destination in the ‘best’ way possible. ‘Best’ is defined by one or more metrics, depending on the application requirements. For example, one widely used metric is using the route with the lowest end-to-end delay, or the highest throughput, whilst other ones could be to use the route with the least hop distance, the best link quality, or least energy consumption. More on routing metrics for WSNs can be found in [25]. The restrictions imposed by WSNs, which were presented earlier in this Chapter, add further requirements to suitable routing algorithms: they have to deal efficiently with an ever changing topology, whilst imposing as little control traffic overhead as necessary on the network, as the transmission of messages is very costly in terms of energy [15].

In order to enable the nodes in the network to actually send data to each other, the routing protocol needs to provide the route it computed between them. Some protocols periodically precompute these routes, whereas others do so only on demand, i.e. when a data packet needs to be transmitted. This behavior is used to distinguish between different types of routing protocols.

Routing protocols are classified into two main categories, proactive and reactive, based on how and when they acquire routes in the network. Research also is conducted in developing hybrid protocols, attempting to incorporate the benefits of proactivity and reactivity. The following Sections describe the different kinds of routing protocols and give an example of routing protocols for each category.
2.3. Wireless Networking

Reactive Protocols

In reactive protocols, routes are acquired by nodes on demand when a packet needs to be forwarded and no path to the destination is currently known. The node triggers a route discovery process, e.g. by diffusing a route request packet through the network and then waiting for a response from the destination node. This response might take time to arrive, causing the packet delivery to be delayed. In reactive protocols, the overhead of control traffic is depending on the data traffic in the network. By acquiring routes on demand, a node has only a partial knowledge about the network, as routes are computed only for destinations to which data traffic has to be forwarded. This might be advantageous in terms of state, as reactive protocols do not require each node to store routes for the entire network. The Ad-hoc On-Demand Distance Vector (AODV) [36] is an example of a reactive protocol.

Proactive Protocols

Proactive routing protocols take a different stance: nodes regularly compute routing tables of the complete network, thus pre-provisioning all possible paths for the entire network topology. In this way, data traffic can be sent out to its destination immediately, without the delay imposed by route acquisition in reactive protocols. However, a certain amount of control traffic is needed to keep routing tables up to date and consistent over the whole network. This control traffic is always present, independently of data traffic on the network. Amongst proactive routing protocols, Optimized Link State Routing (OLSR) [12] is a prominent example as it is used in real world deployments.

WSN Routing Protocols

An array of routing protocols for WSNs exist [4], which use different strategies to address one or more of the restrictions introduced in Section 2.2. The most straightforward way to diffuse information in a WSN is to flood it throughout the network [4]. Flooding means that every node which receives new information will forward it to all its neighbors until it reaches its destination. To prevent broadcast storms, several mechanisms are available: nodes check for duplicates, i.e. messages they have already received, and packets may contain information on how many times they are allowed to be retransmitted. Whilst being easy to implement, flooding causes several drawbacks: amongst others, nodes

\footnote{OLSR is used in the Freifunk and Funkfeuer projects, where private wireless nodes form a mesh-network. The network can exist on its own, but also allows to share Internet access. The interested reader is referred to http://www.funkfeuer.at/ or http://start.freifunk.net/ respectively for further information.}
may receive duplicated messages and a large amount of energy is wasted as there is no mechanism to include energy constraints.

Protocols like, e.g., Sensor Protocols for Information via Negotiation, (SPIN) [21] were developed to allow querying the WSN for data without being able to address particular nodes and to implement energy savings at the same time. SPIN follows an “interest-advertisement-request strategy” in which information is described by meta-data which initially is exchanged between the nodes. Nodes which acquired new data advertise it via its meta-data classification. Neighboring nodes which have an interest in that kind of data reply with a request, on which the advertising node transmits the data to the requesting node. After receiving the new data, the requesting node advertises it to its neighbors. SPIN achieves a high energy efficiency compared to flooding, as only requested information is transported in the network [4]. However, there is no standard meta-data format, as this is supposed to be application specific. Also, the delivery of data is not guaranteed by SPIN’s advertisement mechanism, as the nodes interested in a specific class of data might be distant from the node acquiring this data. If intermediate nodes are not interested in the given class of data, the interested node will never receive it.

To address the issues of scalability and energy preservation in a different way, the notion of hierarchy was introduced in several WSN routing protocols with the goal of avoiding an overload of sink nodes by too many received messages, as well as reducing the amount of overall message transmissions. To achieve this, nodes are grouped into clusters, which feature a node designated as cluster head. Information is relayed to this cluster head which aggregates data to bundle the information and reduce the number of messages which are sent to the sink nodes. With this strategy, communication is forced into a multi-hop manner, relaying information over neighboring nodes which in turn preserves energy as the energy cost of radio communication increases with the distance. Low-energy adaptive clustering hierarchy (LEACH) [19] is one of the first routing protocols applying this strategy.

A third strategy for energy efficient routing found in WSN routing protocols is to use knowledge about the spatial position of sensor nodes to query the WSN in a localized way. If the deployment of sensor nodes is known, queries for data can be directed to the area of interest, thus reducing the overhead in transmissions on the entire network. Some of the protocols falling into this category, e.g., Minimum Energy Communication Network (MECN) [37] or Geographic adaptive fidelity (GAF) [55] were originally designed for mobile ad-hoc networks, but according to [4] might be applicable for WSNs as they are energy aware.
2.4 Protocol Standardisation Process

Another widely used mechanism for energy saving in WSNs is the aggregation of data [4]. Aggregation in this context means that intermediate nodes collect information from their neighbors and then apply aggregation functions like, e.g., computing the maximum, minimum, or average values of the data. However, which aggregation is used, and when, is application dependent and thus ties these protocols to a specific application domain, limiting their scope, and preventing interoperability between different protocols.

As WSNs continue to gain more attention for research and commercial applications, integration of WSNs with other networks becomes desirable to exploit the possibilities offered by the design space. This demands for compatible and interoperable protocols, which provide abstractions for, e.g., heterogeneous hardware configurations and different wireless communication mediums. The absence of such was recognized in academia and industry alike as a considerable issue and consequently initiated efforts to specify a standard routing protocol for WSNs.

2.4 Protocol Standardisation Process

The broad range of application possibilities of WSNs caused a high industry interest in the issue. However, the gain of these systems is currently limited by compatibility problems due to many proprietary protocols being in use. The importance of interoperability has become clear to industry and academia and created lot of interest to moving WSNs to IPv6\(^9\), as this allows for abstracting from heterogenous wireless communication media and thus enables routing the be provided as a service within the IP stack. One indicator, mirroring the industry interest, is the founding of the IP for Smart Objects (IPSO) alliance [6], a group of industry engineers, working on enabling WSNs for IP. Another indicator is the founding of the ROLL [45] and 6lowpan [2] Working Groups of the IETF [23], an important standardization body for IP-based protocols. Also in these Working Groups, industry participation is high. This is even more important to note, as participation in an IETF working group requires from the participants to disclose and private intellectual property rights on the issues discussed and produced. Therefore, all discussions and decisions of the IETF groups are publicly accessible online.

\(^9\)Moving WSNs to IPv4 is not feasible due to the limited address space provided by it. To make IPv6 feasible on such resource constrained networks as WSNs, address compression mechanisms are being researched on.
2.4.1 The Internet Engineering Task Force

"The Internet Engineering Task Force (IETF) is an organized activity of the Internet Society (ISOC). ISOC is a not-for-profit organization founded in 1992 to provide leadership in Internet related standards, education, and policy." [23] Anyone interested in Internet-related topics, as architecture or operation, is free to participate in the IETF. Most work is done via mailing lists, to which everyone can sign up. This allows cooperation independently from the physical location of the participants. Three times per year meetings are held around the world\textsuperscript{10}. In this large\textsuperscript{11} and open international community, network designers, operators, vendors, and researchers form Working Groups in which they specify and standardize protocols as RFCs to solve specific problems. The Working Groups are bundled into several areas, like routing, transport, etc..

The following are the principles of the IETF’s mission: open process, technical competence, volunteer core, rough consensus and running code, and protocol ownership. For further details, the reader is invited to refer to [18].

The typical process of a document becoming an RFC involves it to undergo an intensive period of development and several iterations of review in the shepherding Working Group. Revisions are made based, e.g., upon proofs, experience gained from implementations, simulations, and real world experiments. After the document matured, the IESG adopts it as a Standard and publishes it. The details of the document lifecycle can be found in [8].

2.4.2 The ROLL Working Group

Specification drafts are usually shepherded by a Working Group. The group which was chartered to work on an IPv6 routing protocol for Low power and Lossy Networks (LLNs) is the Routing Over Low power and Lossy networks (ROLL) Working Group [45]. As WSNs are LLNs, they fall into the responsibility of ROLL, where RPL is specified as IPv6 routing protocol for WSNs. Particularly, the group aims at specifying an entire IPv6 routing architectural framework for the application space of LLNs, which should incorporate mechanisms to provide routing security as well as dealing with manageability issues. To this end, the ROLL group has specified several documents including routing requirements for different WSN application scenarios like home and building automation [44, 46], urban LLN deployments [49], and industrial applications [47]. Furthermore,

\textsuperscript{10}In 2010 the meetings were held in the USA, the Netherlands, and China.

\textsuperscript{11}The number of participants of the mailing lists is difficult to estimate. However, the number of attendees at each of the three meetings in 2010 was approximately 1200 according to http://www.ietf.org/meeting/past.html.
it specifies extensions to RPL such as a security framework [43] as well as a point-to-point module [48].
3 Description of RPL

RPL [50], is a distance-vector routing protocol entirely defined for IPv6-based WSNs. It is currently under specification by the ROLL Working Group [45] of the IETF [23]. RPL is designed to be a very flexible protocol in the sense of providing a standard of few base mechanisms, which form the smallest common denominator of functionality on which different WSN applications can agree. Various extensions are provided to tailor RPL to the specific requirements of a particular application. As the RPL specification was subject to frequent changes during the time of writing\(^1\), the scope of this thesis is limited to cover the already well-defined mechanisms of the protocol.

RPL is a proactive routing protocol, constructing its routes in periodic intervals. In one physical network it is possible to run several RPL Instances. Nodes in the network are allowed to be part of more than one Instance. Starting from one or more root nodes, each Instance builds up a tree-like routing structure in the network, resulting in a Destination-Oriented Directed Acyclic Graph (DODAG). Per RPL Instance, multiple DODAGs may exist. However, a node must not join more than one DODAG within one Instance. For the rest of the Chapter, the protocol is explained for one RPL Instance with one DODAG rooted at one designated node.

3.1 Topology Formation

Figure 3.1 displays an example network which will serve for illustration purposes.

Topology formation in RPL starts with designating one node as root node. The root node determines the configuration parameters for the network. The configuration is packed into a DODAG Information Object (DIO) message, which is then used to disseminate the information in the network. There are many options which can be configured in a DIO (see [50]) to tailor the network configuration to the application’s requirements. The compulsory information contained in a DIO comprises amongst others:

- RPLInstanceID for which the DIO is sent,

\(^1\)The specification progressed from version 07 to version 15.
Figure 3.1: Example Network: WSN nodes are depicted by circles, the links between them by dashed lines.

- the DODAGID of the RPLInstance of which the sending node is part,
- the current DODAG version number, and
- the node’s rank within the DODAG.

The **RPLInstanceID** is a unique identifier of an RPL Instance in a network. The **DODAGID** serves the same purpose: to uniquely identify a DODAG in an RPL Instance. A node’s rank describes its logical distance from the root node within the DODAG. When traversing the DODAG from the root node towards the leaf nodes, the rank of nodes is monotonically increasing. When forming the DODAG, each node is required to select parent nodes from its neighbors. Afterwards, when the node is calculating its rank, this has to be larger than the rank of all its parents. In this way, the formation of loops in the routing structure is prevented. Note that rank is not necessarily related to the physical distance, nor to the distance in hops between a node and the root node, but a metric determining a node’s desirability (in terms of application goals, which might, e.g., be load balancing for energy preservation) as a next hop on a route to the root node. A node’s rank is calculated based on the **Objective Function (OF)**, which
3.1. Topology Formation

Figure 3.2: DIO message is broadcast by the root node to its neighbors (as indicated by the bold dashed arrows). The numbers indicate the node’s respective rank, i.e., their logical distance to the root node.

is specified according to the DODAG’s application goals. The OF is therefore one of the hooks which can be used to tailor RPL closely to serve a specific application. Not only does the OF contain the parameters for calculating a node’s rank, it is also responsible for selecting a node’s parents by describing the desirability of a neighboring node to be chosen as parent and therefore be part of a route towards the root node. To give an example, a node’s energy level or its type of power resource could be used in the OF to calculate its rank. For the sake of simplicity, the hop count distance between a node and the root node are chosen as determining parameter for the OF in the following example.

The root node triggers the DODAG formation by broadcasting a DIO message to its neighbors (Figure 3.2). Note that only the root node of a DODAG is allowed to initiate the diffusion of DIOs. Whilst the RPLInstanceID and the DODAGID remain unchanged throughout the whole topology formation, the rank field is updated, as the DIO messages are traversing the network. Since the root node has a distance of 0 to itself, its rank is

\footnote{Further metrics can be found in [25].}
Figure 3.3: Neighboring nodes of root have chosen it as best parent (indicated by the bold plain arrows) and rebroadcast the updated DIO messages (dashed bold arrows).

set to 0. Each neighbor receiving the DIO calculates its rank according to the OF by computing its hop count distance to the root node and sets its rank to 1.

After calculating its rank, each node updates the DIO and broadcasts it to its neighbors (Figure 3.3). Each node retains a candidate neighbor set, in which it keeps track of the neighbors with lower or equal rank it has heard of (i.e., from which it has received a DIO message). Out of this candidate neighbor set, each node selects parent nodes, which have to have a lower rank than the node itself. From the parent set, the node picks a so-called preferred parent, which serves as the node’s next hop when routing a data packet towards the root. This choice is determined by the OF. In the example in (Figure 3.3), the neighbors of the root node only know of one node fulfilling this condition, so they pick the root as their preferred parent. In the Figure, this relationship is represented by the bold plain arrows.

Figure 3.4 displays the last step of the topology formation: all nodes of the network have received DIO messages and joined the DODAG by calculating their rank, whilst the nodes with hop count distance 2 have picked their preferred parents. As the Figure
shows, several nodes in a node’s parent set might fulfill the conditions imposed by the OF, making them eligible as preferred parent. In this case, the node is free to pick any of the suitable parent nodes as preferred parent. Note that the preferred parent must be unique.

Figure 3.5 shows the tree structure formed by each node picking one preferred parent. Even after selecting its best parent, each node keeps its set of parent nodes, to which it can resort as next hops in case its preferred parent becomes unreachable. The DODAG structure including these parent-child relations (displayed with fine dashed arrows) is shown in Figure 3.6.

With all nodes having joined the DODAG, the topology formation is complete - for this iteration which was initiated by the root node. It can happen that node failures or changing environmental conditions create the need to rebuild the routing topology. To help the nodes keep track of which DODAG iteration they are in, and to determine whether it is the newest one, a version number is written in the DIO message. Note that only the root node is allowed to increment the version number in order to trigger a rebuild of the DODAG. So whenever a node receives a DIO message containing a newer version number than the one it recorded, it can add the sender of this DIO to its candidate neighbor set and might even select it as parent. However, a node can only become part
of the new DODAG iteration - and advertise the appropriate version number - once all its parents are part of the new iteration as well. This switch again is governed by the OF which could for example define that a certain percentage of a node’s parents need to be a member of the new DODAG iteration before the node is allowed to switch and to discard all of its outdated parents. However, the mere fact that a node detects a clash in version numbers indicates changes in the network which have to be consolidated by ensuring that all nodes are updated to the current DODAG version. This task is crucial, yet challenging in that it requires reliability when disseminating new information whilst still aiming to be performed at a minimal overhead on the protocol. For meeting these goals, RPL is employing the Trickle algorithm. Both, the problem of data dissemination and the algorithm are described in detail in the next Section.

3.2 Traffic Patterns

RPL by default provides a mechanism for multipoint-to-point (MP2P) data traffic from nodes within the network to the root node. This traffic flow is called `upward’ and is enabled by the DIO mechanism, which is already well understood and rather unchanging over the latest versions of the specification. The DIO mechanism is explained in
3.3. Data Dissemination

As mentioned in Section 2.2, data dissemination is a crucial task in WSNs. RPL builds strongly on this traffic pattern: the DODAG formation relies on it as it is dependent
on DIO messages being disseminated in the entire network, starting at the root node. To guarantee reliability in disseminating the desired data, different attempts have been proposed [28]. Flooding, a mechanism where all nodes rebroadcast the received packets, is straightforward in its idea and its simplicity in terms of implementation. However, it leads to several problems: reliability in transmission can not be guaranteed, and in too dense networks the simultaneous transmissions of packets at different nodes causes message collisions and network collapses (also known as ‘broadcast storms’ [33]). Mending these issues was attempted by adding a request mechanism for missing packets [30]; however, this turned out to be difficult to implement in a reliable way. Furthermore, the huge amount of retransmissions required by packet floods has a very negative impact on the energy resources of a WSN, which are strained noticeably by wireless communication (see [15]). As a solution to this problem, [28] proposes Trickle, an algorithm which tackles the data dissemination problem by reducing the requirement of reliable message transmission to the one of eventual consistency of state between nodes. It does use broadcasting to disseminate information, however retransmissions are bounded by a polite gossiping policy. Since this algorithm forms a core part in RPL, it is introduced in more detail in the following Sections.

3.3.1 The Trickle Algorithm

To build up and maintain its DODAG structure, RPL is using the Trickle algorithm [28, 35] which governs the dissemination of DIOs. Trickle is a polite gossip algorithm, guaranteeing eventual consistency of state between the nodes of a WSN.

**Polite Gossiping** describes the following principle: When listening to a conversation, it is ‘rude’ to repeat what someone else has already said, so one remains silent. However, if one does have different information on the matter, one is encouraged to speak up. Trickle uses this policy on the one hand to restrict packet retransmissions via broadcast. On the other hand, it is used to guarantee eventual consistency of state throughout the WSN, given that a minimum amount of stability is provided by the topology\(^3\). **Eventual consistency** of state is given when all nodes in the network are configured with the same state parameters. Aiming for this goal is Trickle’s way of handling reliability problems such as packet loss or corrupted transmissions. As mentioned above, ensuring that every node in a WSN receives a certain control packet which is sent via broadcast is a costly

\(^3\)Trickle can guarantee eventual consistency only on the premisses of the network topology being sufficiently stable. If, contrarily, nodes continuously keep failing and rejoining the network, the algorithm will always push the network towards eventual consistency — however, this goal would never be achieved due to continuously occurring local inconsistencies.
task. However, ensuring that a WSN eventually converges to a consistent state comes hand in hand with the polite gossiping approach. By listening to its neighbors, a node can always learn the current network configuration (given that all nodes know the rule on how to distinguish current from outdated configurations). In the same way, a node can identify neighbors with outdated configurations and retransmit the current version. Inconsistencies are thus resolved locally by interaction of neighboring nodes, which update each other to the newest version of state they know of. In this way, it is sufficient to inject a new configuration of parameters on only one node in the network. This node will update all its neighbors, which in turn will disseminate the new information deeper in the network. Trickle does not guarantee reliability in terms of distributing each message to every node. However, it does guarantee that all nodes will eventually converge on the same, newest, version of state. Therefore, should a node miss a few packets due to problems in transmissions or failure, it will still be updated to the current version as soon as it re-joins the network. Under the aspect of saving transmissions, this behavior is even advantageous, as retransmitting all missed outdated configurations would not bring any benefit to the network whilst wasting the nodes’ energy resources.

By itself, the concept of eventual consistency, where divergences are resolved quickly, does not yet completely solve the problem of packet floods. There are additional requirements which an efficient algorithm needs to fulfill: whenever the network is in a consistent state, only very few messages should be sent. This prevents the broadcast storm problem from occurring and leaves a considerable amount of air time to other network applications. Furthermore, the state kept at nodes must be small, as the devices are severely constrained in terms of memory. The Trickle algorithm was designed to fulfill these requirements.

General Idea

The idea of Trickle is as follows: nodes advertise their state only within their local neighbor range. Whenever they find inconsistencies, they communicate frequently with each other in order to find and agree on a consistent state. When this is reached, the communication frequency is reduced exponentially to a predefined maximum timespan which, depending on the requirements of the application, might be no more than a few packets per hour. This causes each node to hear a sufficient number of packets against which it checks the consistency of its state, whilst imposing a minimum overhead of control traffic on the network. Due to the use of local communication, Trickle resolves smaller inconsistencies and changes in network topologies without impacting the rest
of the network. Implementations of Trickle show its fulfilling also the requirement of requiring little state with around 10 bytes [28].

**Variables**

Trickle is configured with the following parameters: $T_{\text{min}}$ and $T_{\text{max}}$, the minimum and maximum communication intervals, bounding the communication frequency of the nodes. $k$, the redundancy constant, determines a threshold of how many coinciding messages a node needs to have heard in order to assume its neighborhood to be consistent. Further variables used in the algorithm are: $c$, the communication counter, counting how many times within the current communication interval a node has heard a message advertising the identical state. $I$, the current communication interval. $t$, a timer value with lower bound inclusive of $I/2$ and upper bound less than $I$.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{\text{min}}$</td>
<td>the minimum communication interval</td>
</tr>
<tr>
<td>$T_{\text{max}}$</td>
<td>the maximum communication interval</td>
</tr>
<tr>
<td>$k$</td>
<td>the redundancy constant</td>
</tr>
<tr>
<td>$c$</td>
<td>the communication counter</td>
</tr>
<tr>
<td>$I$</td>
<td>the current communication interval, s.t. $T_{\text{min}} \leq I \leq T_{\text{max}}$</td>
</tr>
<tr>
<td>$t$</td>
<td>the timer value, s.t. $I/2 &lt; t &lt; I$</td>
</tr>
</tbody>
</table>

Table 3.1: Parameters of the Trickle Algorithm.

**Algorithm**

[35] describes the Trickle algorithm with the following rules: When an interval begins, Trickle resets $c$ to 0 and sets $t$ to a random point in the interval, taken from the range $[I/2, I)$, that is, values greater than or equal to $I/2$ and less than $I$. The interval ends at $I$. Whenever Trickle hears a transmission that is ‘consistent’, it increments the counter $c$. At time $t$, Trickle transmits if and only if the counter $c$ is less than the redundancy constant $k$. When the interval $I$ expires, Trickle doubles the interval length. If this new interval length would be longer than $I_{\text{max}}$, Trickle sets the interval length $I$ to be $I_{\text{max}}$. If Trickle hears a transmission that is ‘inconsistent’, the Trickle timer resets. Trickle can also reset the timer in response to external ‘events’. ‘Consistent’, ‘inconsistent’ and ‘event’ need to be defined by the protocol which uses Trickle. Figure 3.7 shows an implementation of the algorithm.
### Figure 3.7: Implementation of the Trickle Algorithm

```c
int select_new_t(I){
    t = I / 2 + (random % (I / 2));
    return t;
}

void set_timers(trickle_t* trickle){
    etimer_set(&trickle->send_timer, select_new_t(trickle->I));
    etimer_set(&trickle->interval_timer, trickle->I);
}

void reset_trickle(trickle_t* trickle){
    trickle->c = 0;
    trickle->I = I_MIN;
    set_timers(trickle);
}

void incrementCounter(trickle_t* trickle){
    trickle->counter++;
}

void initTrickle(trickle_t* trickle, trickle_callbacks_t* callback){
    trickle->callback = callback;
    trickle->isStopped = 0;
    process_start(&trickle_process, (char*) trickle);
    reset_trickle(trickle);
}

PROCESS_THREAD(trickle_process, ev, data)
{
    static trickle_t* trickle;
    PROCESS_BEGIN();
    trickle = (trickle_t*)data;

    while(!trickle->isStopped){
        PROCESS_WAIT_EVENT_UNTIL(ev==PROCESS_EVENT_TIMER);
        if(trickle->c < k){
            trickle->callback->sendDio();
        }
        PROCESS_WAIT_EVENT_UNTIL(ev==PROCESS_EVENT_TIMER);
        trickle->I = (trickle->I * 2 > I_MAX) ? I_MAX : trickle->I * 2;
        set_timers(trickle);
        trickle->c = 0;
    }
    PROCESS_END();
}
```
3.3.2 Application in RPL

RPL is using the Trickle algorithm to support the DODAG formation as well as local and global repair. Figure 3.8 shows the sequence of events which follow the reception of a DIO message in a node in the network.

On reception of a DIO message (state ‘Received DIO’), the node first checks whether its configuration information is equal to the one in the DIO message (Test ‘DIO version == own’), i.e. when their version numbers match. If this is the case, Trickle’s $c$ counter is increased. If the node has a lower rank than the node, the DIO sender is added to its candidate neighbor set. Conversely, if the version numbers are different, the Trickle component is reset (refer to state ‘Reset Trickle’ in Figure 3.8). As an inconsistent state has been detected, there is need for frequent communication amongst the nodes to resolve the issue. Therefore, $I$ is reset to $T_{\text{min}}$, and counter $c$ is set to 0. If the node’s version number is lower than the one advertised in the DIO, the node is adding the DIO sender to its candidate neighbor set. The selection of the node’s parents, its rank calculation, as well as designating its best parent (Figure 3.8) are out of scope of Trickle. They are completely governed by the Objective Function and only displayed in the Figure for the sake of completeness.

Internally, Trickle is keeping track of two time intervals (represented by the send timer, which is set to $t$ and the interval timer (set to $I$)) in Figure 3.9). Whenever the send timer expires, Trickle is evaluating the consistency condition $k \leq c$, determining if the network is still in a consistent state. If the condition is fulfilled, Trickle suppresses the DIO transmission. Otherwise, the node is broadcasting its current DIO configuration. When $I$ expires, Trickle doubles the communication interval. If the doubling would exceed the value configured for $T_{\text{max}}$, $I$ is set to $T_{\text{max}}$. After choosing a new $I$, a new value for $t$ is randomly chosen with respect to the rules mentioned in 3.3.1. $c$ is reset to 0 and the timers are restarted for the next iteration.

3.4 Data Collection

RPL does not employ any specific data collection mechanism, but relies on the routes that have been established by selecting a preferred parent at each node in the DODAG. Whenever a node needs to send data to the root, it is forwarding the packet to its best parent who will then know the best direction to forward it to. In this way, RPL is rather failure tolerant, as nodes do not have to change their routes due to a remote node failing. These events are taken care of by the local repair mechanism managed by Trickle. Additionally, data collection in RPL does not need to maintain much space at each node.
3.4. Data Collection

![Diagram of Trickle Idle process]

- **Trickle Idle**
  - Received DIO
  - **Process DIO**
    - compare DIO version with own version
  - **Else**
    - DIO version == own
    - Increment Trickle
      - \( c++ \)
      - compare sender rank with own rank
    - Sender rank better or equal?
    - **Objective Funktion**
      - Insert neighbor
      - Select parents
      - Compute rank
      - Pick preferred parent
  - **Else**
    - DIO version > own
    - Reset Trickle
      - \( c = 0 \)
      - \( I = \text{Tmin} \)
      - \( t = \text{random_in}[I/2,I) \)
      - reset & restart timers

Figure 3.8: The interaction between RPL and Trickle
Increment Interval

\[ c = 0 \]
\[ I = I \times 2 > T_{\text{max}} ? T_{\text{max}} : I \times 2 \]
\[ t = \text{random}_\text{in}[I/2,I) \]

reset & restart timers

\[ \text{Wait Send Timer} \]
block until send timer fires

\[ \text{Send DIO} \]
send DIO message

\[ c \geq k \]

\[ \text{Wait Interval Timer} \]
block until interval timer fires

\[ \text{Increment Interval} \]
\[ c = 0 \]
\[ I = I' \times 2 > T_{\text{max}} ? T_{\text{max}} : I' \times 2 \]
\[ t = \text{random}_\text{in}[I/2,I) \]
reset & restart timers

\[ \text{Init Trickle} \]
\[ c = 0 \]
\[ I = \text{Imin} \]
\[ t = \text{random}_\text{in}[I/2,I) \]
start timers

Figure 3.9: The internal Trickle mechanism
The only entries needed for forwarding a packet are the node’s preferred parent and, in case of it being unreachable, the other selected parents. If, in the worst case, all parents of a node have become unreachable, it is detached from the DODAG.
4 Implementation

For this thesis, RPL was implemented in the C programming language on the Contiki OS platform and for typical sensor node hardware, in this case the MSB430 Scatterweb mote for the field experiments, as well as the WSN430 Sensor Board in the SensLAB testbed. The building blocks of the implementation can be structured in three ‘layers’: the hardware platforms used, the operating system, and the protocol. Following this bottom-up approach, these components are introduced in this Chapter.

4.1 Hardware Platforms

Two different hardware platforms were available for the experiments conducted for this thesis. One is the MSB430 Scatterweb Mote, which was used in manually setup experiments. The other platform is the WSN430 sensor board, which is installed in a SensLAB testbed.

4.1.1 The MSB430 Scatterweb Mote

The hardware used for the implementation are Scatterweb [40] motes. This sensor platform, also known as MSB430, was originally developed at the FU Berlin and is available for research purposes only. It is featuring the Texas Instrument microcontroller MSP430F1612IPM [42] with 55KByte flash memory and 5KByte RAM. As radio device, the Chipcon CC1020 chip is built in. It has a maximum of 8.6 dBm transmission power and an external LNA (low noise amplifier). Furthermore, the MSB430 ships with a humidity and temperature sensor sensirion SHT11 as well as an three-axis accelerometer MMA7260Q. Since this thesis focuses on routing, this Section does not further introduce these sensors. Additionally, the MSB430 features a red LED and an SD card socket. Figure 4.1 shows an MSB430.

MSP430 Microcontroller

The MSP430 Microcontroller [42] contains a 16-bit RISC CPU with an Instruction Cycle Time of 125-ns and 16-bit registers. It is optimized in terms of energy consumption
and code efficiency. The device supports five different low-power modes. These disable different components of the chip, however keeping it in a state that allows for a quick wake-up response (under 6 µs from stand-by). The power consumption in Active Mode is 330 µA at 1 MHz with 2.2 V. In Standby Mode the microcontroller needs 1.1 µA, whereas in Off Mode, with RAM retention, it consumes 0.2 µA. The supply voltage is low; it must be in the range of 1.8 V to 3.6 V.

Radio Chip

The radio chip used on the MSB430 sensor node is the CC1020 Single Chip Low Power RF Transceiver for Narrowband Systems, produced by Chipcon [9]. It is especially designed for very low power wireless applications as it has a low current consumption of 19.9 mA for receiving signals. Also the required supply voltage is low, ranging from 2.3 V to 3.6 V. The frequencies available are within the Industrial, Scientific and Medical (ISM) as well as the Short Range Device (SRD) frequency bands, which transmit in a range from 402 to 469 as well as at 868 and 915 MHz. The output power of the radio chip can be
programmatically changed. Depending on the channel spacing, the data rate available with the CC1020 lies between 2.4 and 153.6 kBaud.

4.1.2 The WSN430 Sensor Board

WSN430 sensor boards [51] are offering a low-power MSP430-based platform, with a set of standard sensors. Two versions have been developed for the SensLAB project, differing only in the radio interface. Version 1.3 presents an open 868MHz radio interface, while version 1.4 has an IEEE 802.15.4 radio interface at 2.4GHz.

As the Scatterweb MSB430 mote, the board is equipped with a micro-processor by Texas Instruments. However, the MSP430F1611 built into the boards offers a different memory configuration with 48KB ROM and 16KB RAM. The WSN430 boards feature three physical sensors to measure sound, temperature, and ambient light. The radio interface for the 868 MHz ISM frequency (board model wsn430v1.3) is the Chipcon 1101 device. By default, an omnidirectional antenna is connected to the interface. However, also an external antenna can be mounted. For the 2.4GHz radio interface (board model wsn430v1.4) the radio chip is the Chipcon CC2420, which complies to the IEEE 802.15.4 standard. It is connected to an omnidirectional antenna. The boards are identified by a unique EEPROM serial number which is provided by a Maxim DS2411 chip and is readable by the MPS430 firmware. Additionally to the internal memory of the microcontroller, 1 MB of external flash memory is provided with a ST M25P80 chip which is accessible through an SPI bus. A MCP73861 microchip is monitoring the power supply of the board, thus allowing to recharge the batteries whenever needed. The MSP430 firmware is informed about the board’s energy status over two digital outputs.

4.2 Contiki OS

Contiki is a lightweight operating system for wireless sensor nodes presented in [15]. It is written in standard C and has been ported to many different hardware platforms, including the ones used for the experiments of this thesis.

4.2.1 System Overview

As an operating system built for embedded devices such as sensor nodes, Contiki is adapted for straining the given resources as little as possible. The severe restriction in terms of memory and computing power are reflected in the architecture of Contiki in the following way: firstly, the system is highly modularized to keep the necessary core lightweight. Secondly, Contiki saves as much memory as possible by avoiding memory
4.2.2 The Process Model

To define a process, an event handler function is declared. Processes are assigned with a portion of private memory on the global shared stack to hold their state. However, Contiki does not provide memory segmentation or protection domains – instead, all processes share the same address space and the kernel just keeps a pointer to the state’s memory location. Inter-process communication always passes via the kernel. To communicate with each other, processes post events. The events are dispatched to the processes by a scheduler, which also regularly executes the polling handlers. All program execution in Contiki is triggered by these two mechanisms. Scheduled event handlers are never preempted by the system, allowing them to run to completion. They can however internally decide to yield and return control to the kernel. Interaction between applications, device drivers, and the hardware is direct, as Contiki does not implement a hardware abstraction layer. Both, asynchronous and synchronous, events are supported by the kernel. The
only difference in treatment is that asynchronous events are queued by the kernel and
dispatched at a later time, whilst synchronous events are passed on directly and cause
the target process to be scheduled immediately. The posting process is resumed only
after the completion of the invoked process. To gather information from the hardware,
a polling mechanism is used. The polls are scheduled between asynchronous events. On
execution, all processes with a registered handler are called. Preemption among events is
not possible, as Contiki uses only one scheduling level. However, interrupts are allowed
to preempt events.

Figure 4.2: An implementation of the Trickle algorithm showing the Contiki event-based
programming model.

Figure 4.2 shows a code example for the Contiki process and programming model:
the implementation of the Trickle algorithm. The macro in line 81 defines a process
with its name `trickle_process`, and the events `ev` and data `data` passed to it when
it is launched. The variable in line 83 is declared static, so it will ‘survive’ the process being suspended despite of the shared stack provided by Contiki. Lines 88 and 102 show statements which will suspend the process until a certain event is received by it, thus allowing for other processes to be executed in the meantime. For further details on the Contiki process model, see [15].

4.2.3 Networking

Contiki provides two networking stacks. On the one hand, it features $\mu$IP, a minimalistic, yet compliant implementation of IPv6 for sensor networks, which can be used to connect a WSN to any other IPv6-based network as well as to run an entire WSN with IPv6. On the other hand, Contiki comes with a link-layer-protocol independent networking module, called Chameleon. Chameleon consists of two parts: a header transformation module, and the Rime networking stack. The Rime stack provides a layered set of network communication primitives, ranging from anonymous broadcast to identified reliable unicast. Depending on the communication primitive used, different header attributes are specified, indicating parameters required by the primitive. For example, for anonymous broadcast, no header attribute will be set, as apart from the content of a packet, no specific information is needed by the receiver to process the message. Conversely, when using identified broadcast, a header attribute is set to contain the Rime address of the sender. When processing the outgoing message, Chameleon’s header transformation module inspects the header attributes. Depending on the requested transmission protocol, it packs the attributes into the appropriate header format before sending. In this way, Chameleon allows for communication between heterogeneous network types.

Since the primary scope of this thesis is to understand, implement, and experiment with the mechanisms of the RPL protocol, the Rime stack was used for the implementation. This was preferable for several reasons: firstly, the assignment of addresses to sensor nodes was much easier with the Rime stack, as the address format was easier to generate, as well as easier to enter manually on the nodes, compared to full IPv6 addresses. Secondly, the Rime stack is already part of the Contiki system core and therefore there is no overhead in configuring the system as well as the development platform.

4.3 Implementation of RPL

To evaluate RPL on sensor node hardware, we limited the scope to the basic mechanisms of the protocol which were chosen for simulation in [11]: formation of the DODAG, using the DIO mechanism and Trickle.
4.3. Implementation of RPL

4.3.1 Parameters

When implementing the protocol, several parameters had to be chosen. As shown in table 4.1, the decisions for parameters influenced three parts of the protocol: Firstly, the number of entries in the candidate neighbor set had to be fixed, as memory for dynamic allocation had to be reserved at compile time. Given the space limitations as well as the limited size of the networks (between 11 and 200 nodes), each node is keeping track of 5 candidate neighbors, which is more than double the number of parents which nodes have on average in a random DODAG [11].

Secondly, the Trickle component had to be configured. As proposed in the original Trickle paper [29], the redundancy constant $k$ was set to 1, which means that a node overhearing one consistent DIO transmission within its current interval $t$, will increment its counter $c$ and suppress its transmission. The $T_{\text{min}}$ and $T_{\text{max}}$ values were chosen in different orders of magnitude than proposed in [35] to make the DIO transmission mechanism more observable in the experiments. Especially the $T_{\text{max}}$ value had to be kept relatively small in order to shorten the experimental cycles. However, this divergence from the propositions of the specification [35] do not tamper with the Trickle mechanism, as depending on an application’s requirements, the values chosen could be a reasonable configuration for a scenario, where a large burst of control traffic is undesired on protocol initialization and frequent changes to the network topology are expected, thus justifying a more chatty Trickle configuration.

<table>
<thead>
<tr>
<th>Candidate Neighbor Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of entries</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trickle</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{\text{min}}$</td>
</tr>
<tr>
<td>$T_{\text{max}}$</td>
</tr>
<tr>
<td>$k$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Objective Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
</tr>
<tr>
<td>ETX window size</td>
</tr>
<tr>
<td>ETX beacon interval</td>
</tr>
<tr>
<td>ETX neighbor table</td>
</tr>
</tbody>
</table>

Table 4.1: Parameters of the Implementation
Furthermore, the metrics for the Objective Function had to be chosen. [11] used the hop count in their simulations. Therefore the metric of the implemented objective function was chosen accordingly: each node calculates and advertises its rank in terms of hop count distance from the root node. However, this metric on its own causes the OF to select the neighbors which are still within the reach of a node and closest to the root node. Usually, these neighbors are the ones most distant from the node. This produced the following problem during the experiments: during the network initialization phase, there is no traffic on the network except for the DIO messages which are transmitted. For this reason, even nodes with a relatively weak signal from the root node were able to receive the DIO issued by it and therefore picked it as their preferred parent and advertised their rank as 1 to their neighbors. As soon as load was put on the network, interferences caused the direct link between these nodes and the root node to become worse, which lead to a huge packet loss.

To mend this issue, the metrics of the simulated OF were extended by a link level ETX [13]. The same metric was re-used for neighborhood detection. ETX stands for Expected Transmission Count and is a probabilistic measure for path quality between nodes in a wireless network. Since the path quality is determined from the quality of the links composing it, ETX can also be used to evaluate the link quality on a local basis (i.e. the quality of the links a certain node has with its neighbors). This was done to extend the OF. To determine the quality of a link between two nodes, ETX is estimating the number of retransmissions at the sending node needed for a packet to reach the receiving node. To enable this mechanism, beacon messages are broadcasted in regular intervals by all nodes in the network. These beacon messages contain a list reporting the number of beacons the node has received from its neighbors. The reported values enable a node receiving a beacon to determine how many of its sent beacons were received by each neighbor. Based on this, the ETX value is computed with a ratio, resulting in a value between one and infinity. One indicates a perfect, lossless link, whilst infinity indicates that a formally existing link is no longer there. ETX uses a sliding time window for recording the received beacons. In this way, the metric also reflects variances in link quality, which is important as environmental conditions of the network might change, influencing the transmission quality in the network.

Data Traffic

To test RPL under realistic circumstances, data traffic needed to be transmitted over the network. The mechanism chosen to enable this is a shell command, which is can be entered manually or via scripting at the sending nodes. The advantage of this approach
4.3. Implementation of RPL

is flexibility in the experiments as setups can be changed by scripting instead of reprogramming the sensor nodes. Furthermore, full control is given over which and how much data traffic is caused when, where, and at which data rate. The packets sent contain 42 Bytes, of which 32 Bytes were payload and 10 Bytes the Rime header.

**Space Requirements**

The size of the complete implementation consisting of the Contiki core, the basic RPL mechanisms, and logging tools – as flashed on the sensor nodes – sums up to 39.0 KB. 13.0 KB are accounted for by the Contiki core and 26.0 KB by the protocol implementation, logging mechanisms and the additional Contiki modules used in the code.

4.3.2 Development Cycle

To develop the implementation of RPL, an iterative and incremental approach has been used. The components have been developed in a modularized way. Software development processes for embedded software development have not yet been established. This is due to the restrictions in testing and debugging which are a consequence the scarce resources available on the devices, as well as the close interaction with the hardware, which is not abstracted by an operating system. Initially, familiarity with the hardware and the Contiki system was gained by implementing and testing single functions which later would be necessary for the system as a whole. This involved programming simple data packet protocols with the Rime communication primitives and testing the data throughput over single hops. Also, a possibility to manually interact with the sensor nodes turned out to be needed. Therefore, the Contiki shell was explored, extended, and turned into the main instrument of testing, debugging, and logging the system behavior. This rather archaic and limited mechanism proved to be the only one successful, after experiments with external flash memory failed on the MSB430 mote.

Since the code was transferred on the MSB430 nodes by manually flashing one node after the other, testing and debugging on the real hardware imposed a considerable overhead in terms of time on the development cycle. Furthermore, the limited insight into the deployed embedded distributed system posed additional challenges for testing. To mend this situation, the protocol was implemented and tested in the COOJA simulator, which is provided with Contiki OS [34]. Initially, this proved to be efficient - however, the MSB430 mote was not amongst the simulated platforms, so the Tmote Sky configuration had to be used. The different configurations caused several problems when switching between the real and simulated hardware, as despite Contiki’s high level of portability, some parts of the source code had to be changed. The portability issue kept recurring
over the entire development phase. It was crucial in two phases: at first, the need to change the preconfigured MAC layer on the MSB430 motes for measuring the Packet Delivery Ratio made the simulator unusable. Later, when deploying the system onto the SensLAB testbed, the differences of the micro-processors caused multiple problems. Some of these were easy to recognize, as for example the overrun in code size which needed several parts of the system to be changed to match the memory constraints of the WSN430 boards. Others were more difficult to trace, as for example a difference in bit alignments, which occurred only under certain circumstances.
5 Experimental Evaluation

As stated in Chapter 4, several observations in the implementation phase provided indications as to the behavior and the performance of RPL in a real world setting. Furthermore, simulated experiments by [10, 11] shaped certain expectations. This Chapter is exploring these indications further by explicitly measuring the control plane and data plane performance of RPL. Since the experiments were run on real hardware platforms in different scenarios, the Chapter describes the experimental setups and then presents the results of the tests and the evaluation of RPL.

5.1 Observations on RPL

When implementing RPL, observations for improvements of the specifications could be made. However, since several implementations of RPL have provided feedback and experience to the development process (to name a prominent one, ContikiRPL [52]), many issues have been raised and corrected over the ROLL mailing list. This caused significant changes to the specification. Nevertheless, some points concerning the Trickle component, as well as the Neighborhood Detection Mechanisms are introduced in the following Sections. Additionally, this Section presents some remarks about the current state of the specification.

Expectations from Simulations

Following expectations were shaped by the findings in [10, 11], which focussed on the same core mechanisms of RPL as this manuscript: Firstly, the amount of control traffic is low, increasing linearly with the size of the network. Secondly, the convergence time was expected to grow logarithmically with the number of nodes in the network. Thirdly, unicast data traffic from random hop distances in the DODAG was expected to remain continuously around 90%. Forthly, a significant decrease in traffic delivery ratio was observed when the hop distance between sending nodes and the root node increased. Since the simulations were run on networks with 100 up to 1000 nodes, their setup could
not be used for the experiments in this work. However, the outcomes of the simulations influenced the experimental setups.

**Observations on Trickle**

The main assumption of Trickle is that by using a polite gossiping mechanism paired with randomized sending intervals the overhead caused by sending configuration messages (DIOs in the case of RPL) is evenly distributed in the network. Furthermore, by using the redundancy constant $k$ and a very small initial interval of 10ms for the timer, it is expected that DIOs proliferate quickly, causing the network to converge rapidly. Consequently, after this convergence phase the DIO traffic is expected to die down to a few messages per hour.

Take as example the mesh topology in Figure 5.1. The network formation was triggered by setting the Trickle redundancy constant to 1, as proposed in the original Trickle paper. Since all node pairs are connected by bidirectional links, it is reasonable to expect all nodes to receive the first DIO transmitted by root node 4. One node would respond, causing all other nodes to double their intervals and suppress their transmissions. This process would repeat itself, with different nodes sending one message per interval, causing the Trickle to reach the maximum interval as fast as possible. However, it could happen that a message is lost in this scenario, so node 8 would not receive the first DIO sent by root 4. If Node 6 were next to reply, it would cause node 8 to select it as best parent as well as all the other nodes to double their timer interval and suppress their transmission. Figure 5.2 shows the resulting DODAG.

The DODAG formed in this example is suboptimal, as node 8 should have selected root 4 as its best parent but as long as the root node does not resend its DIO, node 8 will not move to a better position. To address the problem of suboptimal DODAG formation, the following tweak is proposed in the Trickle draft: disabling the Trickle suppression mechanism, causing every node to send a DIO each time its sending timer fires, independent of the state of the redundancy constant. This strategy trades resilience against lost packets for a higher control traffic overhead. Depending on the value of $T_{\text{max}}$ and the amount of fluctuation in the DODAG, these additional messages have a more or less significant impact on the network lifetime, as message transmission draws the lion's share of energy from the nodes. The RPL draft on the other hand proposes to ignore DIOs received from nodes with lower rank than the own, thus only increasing the own redundancy constant when a DIO is received from a potential parent node. This would solve the problem of the root node remaining silent after broadcasting its first DIO.
message and consequently nodes missing a message get the chance to hear it another time and insert themselves into better position of the DODAG.

Convergence problems with Trickle seemed to aggravate with physical topologies being less favorable than the one in the example. For example, in preliminary tests it could be observed that with a multi-hop chain topology the network occasionally did not converge as rapidly as expected. Furthermore, despite the claim of Trickle, DIO transmissions seemed to not be equally distributed over the nodes. On the contrary, an alternating pattern could be found, with several nodes sending many DIOs, whereas others rarely do so.

Figure 5.1: 5-node topology with dashed lines representing a bidirectional link between node pairs.

**Observations on Neighborhood Detection**

With RPL being designed as to be IPv6-ready\(^1\), the specification assumes the presence of bidirectional links between pairs of nodes to be ensured by the IPv6 Neighbor Un-reachability Detection (NUD), elaborated in the IPv6 specification. NUD is drawing on several sources of information to determine whether a neighbor is still reachable - and in doing so, it is relying on multiple protocols in the IP stack. As the DODAG formation implicitly only verifies one direction of the links between node pairs, an additional mechanism needs to be provided to ensure a bidirectional connectivity. This is especially crucial when hop count is used as the main metric for calculating a node’s rank, as min-

\(^1\)This assumption has changed over the progress of the specification [14]. By now (version 15 of the draft), RPL is assumed to be completely IPv6-based.
imizing hop count generally tends to favor bad links to distant nodes over good links to physically closer nodes, thus degrading the overall performance of the network [13]. In the worst case, the link between a node and its best parent might be unidirectional, or broken down completely without the child node noticing. Consequently, all data sent via this best parent would be lost. To prevent this situation from occurring, the link metric ETX [13] was implemented and included in the objective function so only a candidate with sufficiently good connection is allowed as best parent. Furthermore, each node verifies once per trickle interval whether its best parent is still alive. ETX is a discrete number describing the estimated number of retransmissions which is needed for a packet to reach its destination. The calculation is based on the number of beaconing messages which nodes periodically send and receive. In each beacon message the sender reports the number of beacons it has received from its neighbors. Due to the periodic beaconing, each node can compute the number of messages it has sent. The ratio of reported versus sent beacons determines the ETX value for each link and can range from 0, which indicates a perfect link, to infinity for links which have become unavailable. Due to the characteristics of wireless communication, this number may vary significantly over time. ETX was found suitable for solving the issues mentioned above. However, employing this mechanism comes with a cost in the following terms: Firstly, code space is needed to implement ETX on the sensor nodes. Secondly, ETX requires beacon messages to be sent regularly, and therefore imposes a constant overhead in terms of radio transmissions, which can not be reduced by a mechanism like Trickle. Thirdly, ETX needs to keep track
5.2. Setup of the Experiments

The experiments were run on two kinds of platforms: one consisted in eleven MSB430 sensor nodes, being manually placed in different indoor environments\(^2\) as well as in different multi-hop topologies. This setup is referred to as MSB430. The other experimental platform was the sensor node testbed of INRIA-Lille which is deployed by the SensLAB project [1]. Experiments done on this platform are labeled with SENSLAB. The following paragraphs introduce the setups on both platforms in more detail.

5.2.1 The MSB430 Setup

For the experiments on the MSB430 platform, the network was deployed in a building of the university, exposing the experiments to realistic environmental conditions like people

---

\(^2\)Originally, also outdoor experiments were planned. First test runs have been conducted, however, the experiment series had to be abandoned due to the weather conditions.
walking across the room, etc. Out of the eleven nodes, four could be connected to and surveilled with a personal computer. In this setup, only a limited insight into the state of the network is possible, as data can be retrieved only from the command shell of nodes connected to a personal computer\(^3\). The topology chosen for the experiments reflects these restrictions in combination with the limitations given by the material at disposal\(^4\).

**Topology**

The nodes in the MSB430 setup were distributed manually over an area of 190 \(m^2\) into the topology shown in Figure 5.3. The topology consists of three clusters of nodes, which are connected over links between nodes 3, 7, and 11. In this way, each cluster could be surveilled by one notebook, whilst leaving one further connection to extract information from the current root node\(^5\) of the network. To generate a multi-hop topology in a manageable area, the antennas were truncated on all but three sensor nodes. The nodes with lower sending capacities formed the local cluster. The remaining nodes were used to overcome the larger physical distances and to connect the local components.

**5.2.2 The SensLAB Setup**

Experiments requiring a complete overview of the system, like measuring the convergence time of the network and the distribution of control traffic over time, were moved on a SensLAB testbed. In this way, the range of possible experiments has been expanded noticeably, as they can be run under changing physical conditions with setup MSB430, whilst within the SensLAB setup, physical conditions remain rather stable, but experiments can be monitored precisely and be scaled up to a larger number of nodes of one order of magnitude higher than in the MSB430 setup\(^6\). In this way, a bigger spectrum of the simulation results of [11] could be used for comparison with the results of this thesis.

It is worth noting that the different runs were executed on different times of the day, however always on the same physical devices. The testbed is shared with other users.

\(^3\)In theory, Contiki OS would support running an IP stack in parallel to the Rime Protocol stack used in our implementation. This would have allowed for extracting node data directly via an internet browser. However, this setup would have inflicted a noticeable traffic overhead on the network, thus interfering strongly with our measurements. For this reason, this approach of surveilling the state of the network was discarded.

\(^4\)For this experimental setup, four connecting cables and three notebooks were at disposal, thus limiting the nodes that could be monitored simultaneously.

\(^5\)The root node was changed over the different runs of the experiment.

\(^6\)The SensLAB testbed contains approximately 250 sensor nodes. However, some of them were down during the period of the experiments.
which run simultaneous experiments, interfering with the measurements. Additionally, the testbed is installed in an office building, so environmental conditions are bound to change as e.g. people move around the building, or turn on or off electrical devices. These factors might have influenced the results, however, these sorts of environmental conditions are difficult to avoid in the WSN application space.

5.3 Measurements

To evaluate RPL, different viewpoints towards the routing protocol have been taken. On the one hand, RPL’s control plane performance needs to be evaluated in order to validate the hypothesis of it being an eligible routing protocol for WSNs. On the other hand, the data plane performance has been evaluated to get an impression of the behavior of a network running RPL.

5.3.1 Control Plane Metrics

To explore and evaluate the control plane of RPL, it is of interest to measure the overhead of signaling traffic which is imposed on the network. This is done by counting the DIO messages which are sent over a certain timespan, as well as per node. Since the dissemination of the DIO messages is managed by the Trickle algorithm, the following behavior would be expected: a large number of DIO messages in the beginning, to
quickly disseminate the new configuration through the network, followed by a continuous small number of DIO messages which ensure the network is still in a consistent state. Additionally, the distribution of DIO messages over the nodes in the network is expected to be balanced. Since the information about the number of DIO messages sent is needed on a time synchronized as well as on a per node basis, the SensLAB setup was chosen for these experiments.

**SensLAB Platform**

On the SensLAB platform, the experiments were run on with network sizes between 15 and 215 nodes. As their physical placement is unknown and can not be influenced, the logical topology of the network in terms of connections is shown in Figure 5.4.

![SensLAB Logical Topology](image)

**Figure 5.4: SensLAB: Logical topology of the testbed tray used for the experiments with the SensLAB1 setup.** The sensor nodes are represented with ellipses and the numbers contained in them correspond to their local IDs and Rime addresses. Bidirectional connectivity between a pair of nodes is symbolized by plain lines, unidirectional connectivity by dashed arrows. The link quality is represented with the line weight, with bold for high reliability and fine for low reliability. The maximum node degree is 8.0, the minimum 1.0, and the average 4.3 with a standard deviation of 1.7.
In a first series, one tray with 34 nodes of the testbed was used. Out of these nodes, five senders were picked at random for each iteration. These senders simultaneously transmitted 40 sets of 100 data packets to the root node. These runs were executed for all data rates. For a summary of the parameters, see table 5.1.

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>12 to 34</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data rates [packets/second]</td>
<td>1, 2, 4, 8, 16</td>
</tr>
<tr>
<td>Hop count [distance to root]</td>
<td>random between 1 and 8</td>
</tr>
<tr>
<td>Number of packets [#/run]</td>
<td>100</td>
</tr>
<tr>
<td>Packet size [Bytes]</td>
<td>32</td>
</tr>
<tr>
<td>Number of simultaneous senders</td>
<td>5</td>
</tr>
<tr>
<td>Number of iterations</td>
<td>40</td>
</tr>
</tbody>
</table>

Table 5.1: SENSLAB: Parameters of the First Series

The experimental setup of the first series was varied by restricting the topologies in the SENSLAB1 configuration to form a chain and a cluster (see Figure 5.5). This was done because the preliminary experiments on the MSB430 setup gave clues that the performance of Trickle could be varying according to the topology in the network. To further investigate this trace, the SENSLAB1 configuration was changed by restricting the communication between certain nodes, thus forcing them into a chain or a cluster topology. In a second series, all available nodes on the testbed were activated and the experiments were run with the same parameters as in the first series (table 5.2). The logical topology of this series is shown in Figure 5.6.

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>between 215 and 183</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data rates [packets/second]</td>
<td>1, 2, 4, 8, 16</td>
</tr>
<tr>
<td>Hop count [distance to root]</td>
<td>random between 1 and 8</td>
</tr>
<tr>
<td>Number of packets [#/run]</td>
<td>100</td>
</tr>
<tr>
<td>Packet size [bytes]</td>
<td>32</td>
</tr>
<tr>
<td>Number of simultaneous senders</td>
<td>5</td>
</tr>
<tr>
<td>Number of iterations</td>
<td>40</td>
</tr>
</tbody>
</table>

Table 5.2: SENSLAB: Parameters of the Second Series

\(^7\)On the outer chain-like parts of the MSB430 topology (Figure 5.3), it could be observed that the network occasionally would not converge properly, as some nodes did not receive any DIO message at all.
Figure 5.5: SENSLAB: Topologies for forming a chain (top) and a cluster (bottom) on the SENSLAB1 configuration. The participating nodes are visualized with bold ellipses and the root node is highlighted with a double ellipse. The bold arrows indicate the established best parent relationships between the nodes.
Figure 5.6: SENSLAB: Logical topology of the network for the experiments with the SENSLAB2 setup. The maximum node degree is 64.0, the minimum 0.0, and the average 39.0 with a standard deviation of 14.7.
5.3.2 Data Plane Metrics

The metrics chosen for evaluating the data plane performance of RPL are: **Packet Delivery Ratio (PDR)** versus **Load** or **Hop-count Distance**. Experiments to observe this behavior were conducted on both platforms.

**MSB430 Platform**

On the MSB430 platform one node in the network was designated as root node, and amongst the other nodes a sender was picked\(^8\). Figure 5.7 shows the data flows of several experimental runs. The data packets travel from the sending node to the root node on the path with the lowest hop count, as computed by the Objective Funktion.

To measure the PDR with respect to the hop count distance between sending node and root node, nodes with a distance between one and five hops were selected from the different clusters and the PDR recorded at the root node. In each run, the sending node sent five sets of 100 data packets to the root node. To observe the PDR with respect to a changing data rate, each sending node executed a data run for the data rates of 1, 2, 4, 8, and 16 packets per second. Table 5.3 summarizes the parameters of the experiments.

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data rates [packets/second]</td>
<td>1, 2, 4, 8, 16</td>
</tr>
<tr>
<td>Hop count [distance to root]</td>
<td>1, 2, 3, 4, 5</td>
</tr>
<tr>
<td>Number of packets [#/run]</td>
<td>100</td>
</tr>
<tr>
<td>Packet size [Bytes]</td>
<td>32</td>
</tr>
<tr>
<td>Number of simultaneous senders</td>
<td>1</td>
</tr>
<tr>
<td>Number of iterations</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 5.3: MSB430: Parameters of the Experiments

**SensLAB Platform**

Three series of experiments focusing on the data plane performance of RPL were done on the SensLAB platform with the SENSLAB1 configuration. It differs from the setup of the MSB430 experiments in several aspects: firstly, the hop distance of the nodes range from 1 to 9. Secondly, for each series the senders were randomly chosen for each data rate. The data rates remained the same as for the MSB430 experiments.

\(^8\)This experimental setup did not allow for multiple senders at a time, as the sending of the data traffic on the nodes could not be synchronised externally.
Figure 5.7: MSB430: Data flow over 3, 3, 4, and 5 hops on the MSB430 topology. Data packets are forwarded along the minimum hop count path calculated by the Objective Function (represented by the bold arrows) to the root node, which is highlighted by a double circle.
5.4 Results

The following Sections present the results of the experiments conducted during this thesis. Firstly, it focuses on the protocol specific metrics of RPL, observing the behavior of the control traffic and the overhead thus generated. Secondly, it provides an overall view on the performance of a network running RPL as routing protocol.

5.4.1 Control Plane Metrics

To evaluate RPL, the following Section analyses the overhead generated by the protocol to establish and maintain the routing topology. A full overview of the WSN is needed to draw sufficient information about this control plane behavior. Therefore, the SensLAB platform was chosen for the experiments.

Trickle Convergence

Plot SENSLAB1 configuration in Figure 5.8 displays how the amount of DIO messages sent in the network varies over time in the SENSLAB1 experimental setup. As expected from the Trickle component, initially a large number of DIO messages are sent through the network, thus rapidly updating all nodes to the current state. Then the signaling traffic dies down as Trickle converges to its maximum interval. This interval visibly governs the further dissemination of DIO messages: in the first half of the interval all nodes are listening to their neighbors, whilst in the second half the nodes with insufficient information broadcast their DIO message. The periodicity of this mechanism is reflected in the periodic small spikes in the Figure.

Plot SENSLAB1 chain configuration in Figure 5.8 shows the same experiments run on the chain topology in Figure 5.5, whilst Plot SENSLAB1 cluster configuration shows the results for the cluster topology in the same Figure. These results are very similar to each other and to the ones for the unconstrained topology. Therefore, the suspicion of Trickle being less suitable for chain topologies is invalidated. However, a significant performance difference of Trickle shows when comparing the results from the SENSLAB1 experiments to the SENSLAB2 experiments, which are shown in Plot SENSLAB2 configuration in Figure 5.8. Since the number of DIOs sent is displayed relative to the number of nodes in the network, it can be observed that Trickle - and therefore RPL - is performing significantly better in terms of control traffic in the SENSLAB2 setup. When comparing the physical topologies chosen for the SENSLAB1 experiments (Figure 5.4 and Figure 5.5) to the one used in the SENSLAB2 experiments (Figure 5.6), one notes a significant difference in density. The better performance is to be found on the denser network, as
Figure 5.8: SENSLAB: RPL’s signal traffic overhead over time, evaluated with the SENSLAB1 and SENSLAB2 configurations (Tables 5.1 and 5.2) on the unrestricted topologies in shown in Figures 5.4 and 5.6, as well as with the SENSLAB1 configuration on the cluster and chain topologies depicted in Figure 5.5. Each Plot is computed over 10 to 13 repetitions of the experiment. The data points display the maximum, minimum, upper and lower quartile values, and the mean.

due to Trickle’s local broadcasting mechanism one DIO transmission can be overheard by proportionally more nodes. This observation backs up a remark in [29], which claims a better performance in denser networks, and attests RPL a good scalability.

**DIO Message Distribution**

Another hypothesis about the behavior of Trickle is the assumption of its evenly distributing the DIO message sending amongst all nodes in the network. To investigate this aspect, several iterations of the experiments were analyzed. Figure 5.9 shows a snapshot of all nodes in the SENSLAB1 setup after one iteration of one experimental run.

Each node is displayed with its current rank, as well as with the number of DIO messages it has sent during this iteration. The distribution of DIO messages over the nodes is not as uniform as expected. To find out whether a relation exists between the node’s rank and the number of DIO messages it has sent, several iterations were summarized into Figure 5.10. The Figure shows that there is no clear proportional
Figure 5.9: SENSLAB: Snapshot of node states during one iteration of a SensLAB experiment. Node 31 is the root node, with rank = 0. Nodes 1, 14, 15, 21, and 30 have been chosen randomly as sender for this iteration. Each sender transmitted 100 packets at a rate of 2 packets per second.

The relationship between a node’s rank within the DODAG and the number of DIOs it has sent. The Figure shows that there is no clear proportional relationship between a node’s rank within the DODAG and the number of DIOs it has sent. Instead, the nodes in whose neighborhood there is a steady trickle of DIO messages can be observed to not send any whilst nodes in whose neighborhood there is much fluctuation in observed DIO messages are seen to send more frequently. Overall, the number of DIO messages sent varies significantly between none at all up to 7.

It would be interesting to compare these findings with the physical topology of the SensLAB setups. This would allow to set these results in relation to the placement of the nodes. [28] mentions a skew of the number of messages sent in scarcer parts of the network, as nodes tend to have less neighbors and therefore are bound to observe less DIO messages in their neighborhood, which in turn prompts them to send them more often. In our result, this mechanism could explain why nodes with higher rank send more DIO messages. At the same time, it could be interesting to investigate the node degree of the middle nodes (with rank 3 and 4). Their sending so few DIO messages indicates

\[\text{Unfortunately, at the time of writing, a map with the physical placements of the nodes could not yet be acquired.}\]
5.4. Results

![Graph showing # of DIOs sent vs rank for SENS LAB2 configuration]

Figure 5.10: SENS LAB: Rank of nodes in the SENS LAB2 setup with the number of DIOs they sent. The Plot displays the maximum, minimum, upper and lower quartile, and the mean values of DIO messages sent at each rank. The data was normalized over the number of nodes present at each rank. Each data point was computed from 73 experimental runs.

that they are observing many of them in their immediate neighborhood, causing them to remain silent. This in turn would cause the nodes with rank 1 and 2 to observe very few DIO messages, again leading these nodes to broadcasting more DIOs.

5.4.2 Data Plane Metrics

After evaluating RPL from the control plane aspect, it is of interest to investigate the performance on the data plane, i.e., the behavior of a network running it as its routing protocol. The following Section present an analysis on the MSB430 setup as well as on the SensLAB one.

Figure 5.11 shows the PDR achieved with RPL with the network load increasing from 1 packet per second up to 16 packets per second. The curves display different experimental settings, with the load being put on the network at varying hop distances from the root node.

The observed behavior is as expected, coinciding with the trend in the simulations in [10]. With a low data rate of 1 or 2 packets per second, a high PDR can be achieved for all hop count distances. In all curves, but for the one of one hop distance, the PDR
drops drastically as the data rate is increased. Due to the higher sending frequency, more packets are lost due to collisions.

With increasing the hop count distance between sending nodes and the root node, a decrease in PDR occurs. This increase was achieved by selecting different data paths in the MSB430 topology. Figure 5.7 shows the data paths for the experiments. For a sender with a hop count of 2, the PDR remains relatively stable with slight decrease up to 4 packets per second, and drops drastically at a rate of 8 packets per second. This drop is similar for nodes with a hop count of three. When increasing the hop count distance to 4 or 5 hops, the PDR decreases significantly at two points. The first drop occurs when increasing the data rate to 2 packets per second, the other one when increasing it to 8 packets per second.

Two points concerning these results are elaborated further in the following paragraphs: Significant drop in PDR between 4 and 8 packets per second: most curves show a clear drop in PDR at a data rate of between 4 and 8 packets per second. Supposedly, this drop is larger than for lower data rates due to the higher amount of interferences caused by the higher data rate. However, as collisions occur more frequently now, packets are regularly retransmitted. In this way, a stable quota of packets is transmitted. This behavior occurs especially at a higher hop distance.
Convergence of curves at a data rate of 8 packets per second and subsequent low decrease of PDR when doubling the load to 16 packets per second: This behavior might seem unintuitive at a first glance. However, the convergence of values, as well as the comparatively minor decrease in PDR in this sending interval, are due to two factors: Firstly, with 8 packets per second, the boundary of the system's data transmission capacity is approached. When calibrating the transmission capacity on the MSB430 sensor nodes with the minimum Contiki core and single hop message sending, a time lag in sending could be observed around the frequency of 10 packets per second at a single hop distance. The PDR is therefore expected to significantly decrease, and also converge, under this load in a multi-hop environment. When doubling the load to 16 packets per second, the measurements are taken slightly outside the border of the transmission capacity causing packets to be sent at a lower rate than the 16 packets per ‘real time’ seconds. The PDR over the single hop, however, remained high between 90 and 100%. When further increasing the load, packet transmission became impossible with the PDR dropping to 0% as the received packets could not be processed fast enough and therefore got dropped.

Secondly, at a data rate of 8 packets per second, the system’s MAC layer, using CSMA, was observed to attempt retransmissions of packets on collision detection. As described above, this effect contributed to mitigate the packet loss caused by the higher load.

Figure 5.12 shows the PDR achieved with RPL over varying hop count distances. The PDR stays relatively stable up to three hops in the experiments with 1, 2, and 4 packets per second and drop significantly at a hop distance of four. For the data rates of 8 and 16 packets per second, the PDR already drops significantly at two hops distance from the root. When approaching a hop distance larger than 3, the PDR drops significantly even for lower data rates of 2 and 4 packets per second. Over 5 hops, a PDR of over 50% could only be achieved with the lowest measured data rate of 1 packet per second.

Figure 5.13 shows results of the PDR versus load experiments on the SENSLAB2 setup. In this experiment, all nodes available on the testbed were used. Out of these, in each run of the experiments, with run designating an iteration of the experiment with a fixed data rate, five random nodes were selected as senders. Each data point thus was measured over 40 runs. The first significant drop of the PDR occurs at an offered load of 2 packets per second and is larger than expected. This could be due to sending nodes being in close physical proximity, thus interfering with each others messages. Given the networks high density, this hypothesis is probable. Also, external interferences to the testbed can not be excluded. To interpret this drop correctly, additional information would be needed. The the further data points of this Plot are as expected, with a PDR of around 50% for a load of 4 packets per second, and continuously decreasing with higher load. It is to be
Figure 5.12: MSB430: Evaluation of PDR vs Hop-count in the MSB430 setup. Each Plot consists of the results of five runs per data point.

noted that the given Plot resembles the Plot of PDR vs. Load with a distance of 4 hops, displayed in Figure 5.11.
Figure 5.13: SENSLAB: Evaluation of PDR vs. Load on the complete testbed with the SENSLAB2 setup. For each run of the experiment, five random nodes were selected as senders. For each data point, 40 runs of the experiments were executed.
6 Conclusion

In this thesis, RPL was to be evaluated under realistic deployment circumstances on typical WSN hardware. To this end, the core mechanisms of the protocol were implemented in C on the Contiki OS software platform. This implementation was ported to two typical hardware platforms, namely the MSB430 Scatterweb mote and the WSN430 Sensor Board. The MSB430 Scatterweb motes were used for manually deployed small-scale experiments focusing on the data plane performance of a network running RPL as its routing protocol. Conversely, the WSN430 Sensor Boards were installed in a testbed structure. The experiments on the testbed were focusing on the control plane aspects of RPL, such as control traffic overhead and distribution. Experiments were done for networks of various size, with the number of nodes ranging from 11 up to 215.

The results of the experiments partially correspond to the expectations of the specification. In terms of control traffic overhead, RPL performed equally well independently from the different network topologies it was running on. The control traffic peaked as expected during the period of topology formation with 0.8 up to 2.7 packets per node per Trickle interval and subsequently decreased to 0.2 up to 0.5 packets per node per Trickle interval. However, the expectation of a regular distribution of control traffic over all nodes in the network was not fulfilled. A pattern could be observed in which nodes at a certain rank were found to send control messages more frequently than others, which could cause the network to lose connectivity by prematurely depleting the energy of intermediate nodes. However, depending on the settings of the Trickle component, this behavior could potentially be mitigated. Considering the different densities of the experimental networks, RPL performed by factor three better on the setup with the largest number of nodes which were also deployed in the highest density. This attests RPL's core mechanisms a good scalability and makes them seem suitable for routing in large scale, dense WSNs.

In terms of data plane behavior, the results in this manuscript fall behind the expectations of the simulations. The simulated PDR constantly was larger than 90% for a load of 3.2Kb per second in a multi-hop network with up to 100 nodes. In the corresponding experiment on real hardware, the maximum PDR achieved on a network with 215 nodes was 80% for a load of 42Bytes per second which decreased constantly to 10% for a load
of 5.2Kb per second. When focussing on PDR vs. hop count, the PDR decreased continuously from 90% at a distance of one hop to 60% for a load of 42Bytes per second up to 20% for a load of 5.2Kb per second.

This thesis provides results of one of the first implementations of RPL\(^1\) tested on real WSN hardware. However, there are some limitations to them. Firstly, the implementation was restricted to the core mechanisms, thus not dealing with all the possible configurations and options. Therefore, no prediction concerning the performance of more elaborated mechanisms can be made. The focus on the default configuration also kept the space overhead caused by control messages at a minimum. When installing RPL as routing protocol for a specific deployment, experiments with the planned configuration should be done to ensure all application specific requirements are fulfilled. Secondly, the network sizes which could be achieved with the testbed structure are still far behind the dimensions which are envisioned for future WSN deployments. Thirdly, feasibility in terms of energy consumption and its influence on network lifetime were not yet considered, as it is closely tied to application requirements. RPL does impose little overhead in terms of control traffic, however, neighborhood detection mechanisms also use up radio time. It still needs to be found out how well RPL would perform when being combined with with power saving mechanisms, such as e.g. duty cycling. One hypothesis is that nodes would miss control messages due to their radio sampling interval. As a result, the amount of control traffic would increase and the network would converge more slowly as nodes need to wait longer until they find out about their neighbors.

There are still many steps along the process of examining whether RPL fulfilling the goal of becoming the standard routing protocol for WSNs. Firstly, a thorough testing of more elaborated concepts of RPL is under way and needs to be done extensively. Secondly, it remains to be seen whether the concept of providing a standardized core protocol with extensions is able to fulfill its promise as a flexible architecture, making RPL suitable for a broad range of WSN applications. Thirdly, the design space of Objective Functions is still rather unexplored. Therefore, research is needed to investigate how and how well the requirements of typical WSN applications can be translated into Objective Functions, as well as on how much impact the chosen Objective Function has on the performance of the WSN in which it is deployed.

\(^1\)In the 2010, also the developers of Contiki OS have implemented a version of RPL and set it as Contiki’s default routing protocol.
Bibliography


