Informatics

Simulation des IPv6 Routing Protocols RPL

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Alexander Baranyai, BSc
Matrikelnummer 01525251

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner

Wien, 1. Mai 2024

Alexander Baranyai	Wolfgang Kastner

Informatics

Simulation of the IPv6 Routing Protocol RPL

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Alexander Baranyai, BSc
Registration Number 01525251

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner

Vienna, 1st May, 2024

Alexander Baranyai   Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Alexander Baranyai, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Mai 2024

Alexander Baranyai

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Anfertigung dieser Masterarbeit unterstützt haben. I möchte mich bei meinem Betreuer Wolfgang Kastner bedanken, der mir Feedback zu meiner Arbeit gegeben hat und mich nicht aufgegeben hat. Auch möchte ich mich bei Tommaso Pecorella bedanken, welcher mir bei Fragen zu RPL und ns-3 zur Seite gestanden hat, wenn ich nicht mehr weiter wusste. Abschließend möchte ich mich bei meinen Freunden und meiner Familie bedanken, die mich unterstützt haben dieses Ziel zu erreichen.

Acknowledgements

In this section, I want to thank everyone who supported me, in creating this master thesis. I want to thank my advisor Wolfgang Kastner for giving feedback to my work and not giving me up. I would also like to thank Tommaso Pecorella, who helped me with questions about RPL and ns-3, when I didn't know what to do anymore. Finally, I want to thank my friends and family wo supported me to reach this goal.

Kurzfassung

Drahtlose Sensornetzwerke spielen in heutigen Anwendungen eine tragende Rolle und erfordern effiziente und zuverlässige Routing-Protokolle, um die bestmögliche Datenübertragung sicherzustellen. Neben anderen Routing-Protokollen hat sich RPL als vielversprechende Lösung für drahtlose Sensornetzwerke herausgestellt, in denen Geräte nur über limitierte Energieversorgung verfügen und schlechte Funkverbindung haben, herausgestellt. In dieser Arbeit wird darauf eingegangen, RPL in einem performanten und benutzerfreundlichen Netzwerksimulator durch die Benutzung von ns-3 zu simulieren. In diesem Netzwerksimulator werden Beispiel-Topologien erstellt, um das Verhalten von RPL zu testen. Durch die Implementierung von RPL in ns-3 ist es möglich, RPL auf verschiedene Parameter zu überprüfen. Die Ergebnisse der Simulation zeigen, dass RPL linear mit der Anzahl an Hops zum Root Knoten skaliert, wenn die Objective Function Zero verwendet wird. Auch wurde beobachtet, dass der Doublings Wert des DIO Trickle Timers stark die Reaktionsfähigkeit des RPL Netzwerks beeinflusst. Ebenso wird verdeutlicht, wie RPL reagiert, wenn sich ein Leaf-Knoten durch ein statisches Raster aus RPL Knoten bewegt. Insgesamt kann der Schluss gezogen werden, dass RPL eine gute Reaktionsfähigkeit und Skalierbarkeit in drahtlosen Sensornetzwerken aufweist und dadurch sein Potenzial für den realen Einsatz in verschiedenen Umgebungen und Anwendungen unter Beweis stellen kann.

Abstract

Wireless Sensor Networks take a crucial role in today's applications, requiring efficient and reliable routing protocols to ensure the best possible data transmission. Among other routing protocols, RPL has emerged as a promising solution for WSNs with lossy links and low-power devices. In this thesis, the need to simulate RPL in a performant and user-friendly network simulator, by leveraging ns-3, a widely used simulation tool, is addressed. In this simulator, example topologies are defined to investigate the behavior of RPL. Implementing RPL in ns-3 then gives the opportunity to conduct extensive tests on these defined topologies. The results in this thesis demonstrate that RPL scales linearly with the hop count to the root node when utilizing the objective function zero. Furthermore, it is observed that the doublings value of the DIO trickle timer heavily influences the responsiveness of the RPL network. Additionally, it is revealed that RPL maintains functionality when a leaf node traverses through a grid of static RPL nodes. Overall, this thesis concludes that RPL demonstrates a good responsiveness and scalability in WSNs, showcasing its potential for real-world deployment in diverse environments and applications.

Contents

CHAPTER 1

Introduction

1.1 Motivation

The interest of the Internet of Things (IoT) is still growing and with the increasing number of devices and applications, scalable and efficient routing protocols are needed. Routing in a network is used to send messages in the network along the best possible path. Routing in a Wireless Sensor Network (WSN) faces the restrictions that there is limited energy supply, limited bandwidth and limited computing power. Thus these networks are commonly called low-power and lossy networkss (LLNs). An LLN consists of multiple nodes which are interconnected by a diversity of links, which are characterized by their limited resources and unreliable links. The importance of standardization in LLNs shall also be emphasized and different protocols and standards such as IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN), IPv6 Routing Protocol for low-power and lossy networks (RPL), and Constrained Application Protocol (CoAP) are often discussed [1]. These enable interoperability and communication between IoT devices. Also, LLNs are typically constrained by a limited transmission range and a sparse deployment of devices. One of the most common used standards in LLNs is the Internet Protocol version 6 (IPv6) routing protocol RPL. Accommodating for these spatial constraints, RPL is essential for making the best routing decisions possible with the use of an Objective Function (OF). RPL provides multipoint-to-point, point-to-multipoint and point-to-point routing for devices in the same LLN in a proactive manner. RPL is designed for networks with lossy links, which are prone to a high packet error rate and link outages [2]. As a distance-vector protocol, RPL has a routing topology, which is a Destination-Oriented Directed Acyclic Graph (DODAG) that is typically rooted at a network border router. Each node calculates its rank, representing its distance to the root using some cost function [3].

Furthermore, network simulation is an effective method for evaluating the performance of routing protocols in different environments and scenarios. One of the predominantly used network simulator is Network Simulator 3 (ns-3), which provides a comprehensive framework for modeling and simulating various types of networks, including LLNs. The integration of RPL in ns-3 shall enable the an easier and better way to evaluate the routing protocol in different controlled and reproduceable scenarios, which is important for assessing it for effectiveness and highlighting problem areas.

Therefore, the motivation for implementing RPL in ns-3 is to provide a basis to evaluate the performance of RPL in different scenarios and conditions, which can lead to the development of more efficient and scalable routing protocols for LLNs. Moreover, the integration of RPL in ns-3 can help the design and testing of new applications and services that rely on LLNs, leading to further advancements in the IoT.

1.2 Problem Statement

Since RPL is widely used in WSNs, especially in conjunction with 6LoWPAN, it is currently lacks comprehensive performance analysis. Therefore, the need for a performance analysis of RPL is inevitable in terms of onboarding, scalability, flexibility and energy consumption are crucial for understanding its capabilities and limitations.

Conducting performance analysis on real hardware can be cumbersome due to scalability limitations and lack of flexibility in modifying network parameters. A viable solution would be a robust network simulator that can simulate RPL networks. Therefore, a network simulator that can accurately simulate RPL networks is highly desirable. However, existing implementations of RPL in other simulators may have limitations such as being abandoned, requiring maintenance to work with updated frameworks, being proprietary or not available, missing important components, or being too obscure to use.

To achieve sustainable and rigorous simulation results, a simulator is needed which is actively maintained, has a decent amount of features already available and has a rigorous foundation, ensuring accuracy of simulation results and ongoing support by the community. From the multitude of network simulators, ns-3 is the preferred choice, as it is an already well established network simulator and is predominantly used in this research area.

As of writing, there exist other implementations of RPL in various simulators, which unfortunately all have at least one or more drawbacks, rendering them unusable for evaluation or further use. These faults include that they are abandoned and require further maintenance to work with updated frameworks, are proprietary or not available at all, have missing components and/or are simply too obscure to use.

There is no RPL implementation available in the preferred simulator ns-3 and this thesis aims at filling this gap. As ns-3 has many modules already built in, RPL can be coupled with 6LoWPAN and Low-Rate Wireless Personal Area Network (LR-WPAN). This allows for numerous realistic scenarios to be simulated and evaluated, using the various tools and modules readily available (e.g. energy estimations).

1.3 Aim of the Work

The aim of this work is to develop a suitable RPL implementation in ns-3, to provide the research community with a rigorous, flexible and sustainable RPL implementation for a network simulator with an active community. Further, RPL shall be tested regarding onboarding, latency, scalability and energy consumption, offering much needed insights, in particular in WSN research. This should allow to answer if RPL usable enough in LLNs.

Hypothesis 1 *Simulation of RPL using ns-3 offers significant advantages over existing simulations/implementations.*

The thesis shall answer the following research questions:

1. Which network simulator is suitable to evaluate RPL in LLNs?
2. Which experiment topologies shall be conducted to research the behavior of RPL in LLNs?
3. How fast does RPL react to changes in the network?
4. How well does RPL perform in WSNs, regarding onboarding, latency, scalability and energy consumption?

1.4 Approach

First, different routing protocols shall be researched and compared to each other, with special attention on RPL. Then, existing network simulators shall be research, and the best suitable simulator for RPL shall be chosen. Next, information from already published scientific publications about RPL shall be gathered, to gain knowledge about the state of the art of existing RPL implementations. This shall give deeper insight on how to implement the RPL protocol, as well as highlight improvement potential in comparison to the existing implementations.

Then, RPL shall be implemented as an IPv6 routing module in ns-3, in conformance with the RFC standard [4]. For this implementation, part of RPL shall be abstracted where possible and necessary, ommiting details that are not relevant for the simulation, while functionality for generic use-cases shall be prioritized.

Examples shall then be designed to to evaluate the created RPL implementation in terms of different metrics.

Next, the ns-3 RPL implementation shall be checked for conformity with the RFC standard regarding its behavior. This shall be done by simulating smaller use cases that encompass common functionality, aka performing smoke tests. Using this created examples, an evaluation of the implementation of RPL shall be conducted, focusing on how performant RPL is, regarding onboarding, latency, scalability, energy consumption and flexibility. Finally, the evaluations shall be analyzed and interpreted.

1.5 Structure of the Work

The rest of this thesis is structured as follows: Chapter 2 describes what routing protocols are, gives an overview of different routing protocols and compares them. Chapter 3 describes what network simulators are, gives an overview of different network simulators and present the best suited simulator for RPL. Chapter 4 showcases the different examples, which are used for the evaluation of RPL in this thesis. Chapter 5 outlines how RPL is implemented in ns-3. Chapter 6 includes the evaluation and interpretation of RPL regarding onboarding, flexibility, scalability, energy consumption and simulation time. Chapter 7 concludes the thesis and states future work.

CHAPTER 2

Routing Protocols

2.1 Concepts of Routing Protocols

A routing protocol specifies how nodes[1] know where to find other nodes in a network. Additionally the routing protocol selects which routes shall be taken to communicate with another node. Usually all information a node currently has over a network it is in, is stored in a routing table. The information in these tables can vary from protocol to protocol but usually contains the other nodes address and interface as well as the next hop address. The next hop address is the address of a node that is directly reachable from the sender node and is in the senders routing table for the specified receiver. Therefore, if a node wants to send a message to a node it can not reach directly, it can send the message to a next hop node. The next hop node will then forward the message to the destination or, if it also cannot reach the destination directly, to it's own next hop node for the receiver of the message.

To select the best route (if there are multiple) to another node, metrics are often introduced in routing protocols. These metrics can vary, such as in a wireless setting the signal strength can be used.

Routing protocols can be either proactive or reactive. The difference is that in proactive protocols nodes know the route before sending messages and in reactive protocols the route is calculated when the message is sent. Both approaches have their advantages and drawbacks and it depends on the situation which to use. A proactive protocol has the benefit that it has a fast route discovery and reliability, but can have a large routing overhead due to periodic updates. A reactive protocol has a low routing overhead and consumes less resources, but can have a high latency in route finding, which can also clog the network.

[1]The terms node, host and router are interchangeable used in this document.

Furthermore routing protocols can be classified into three major categories. There are Interior Gateway Protocols type 1, Interior Gateway Protocols type 2 and Exterior Gateway Protocols. Interior Gateway Protocol type 1 are link state routing protocols. In link state routing, each node has its own understanding of the network in form of a graph, which shows which nodes are connected to which. Interior Gateway Protocol type 2 are distance vector protocols. Nodes that use distance vector routing protocol determine the best route to another node by exchanging routing information and use the best possible path to the node. This is normally based on the number of hops to the destination, but can also be any other metric. Finally, Exterior Gateway Protocols are routing protocols that exchange routing information between autonomous systems, which normally themselves run Interior Gateway Protocols of either type.

2.2 RPL

RPL is a proactive distance-vector routing protocol. RPL is a routing protocol designed for LLNs, which are characterized for having devices with limited processing power, memory and energy resources. RPL is based on the concept of a DODAG, which is a directed acyclic graph that models the network topology and defines the routes between nodes. The goal of RPL is to provide efficient and reliable routing in LLNs, while also minimizing network resources and energy consumption on nodes. RPL also tries to to optimize routing performance, by adding features, such as an OF to select the best path, a trickle timer to reduce control traffic and the support for source routing and traffic engineering.

Other research was already done on the development of new OFs, the integration of RPL with other routing protocols, the implementation of RPL on different platforms, and the evaluation of RPL in various application scenarios [5][2][6][7][8][9][10][3][11]. As RPL can be confusing to understand due to the DODAG structure of the protocol and having a very extensive RFC with many optional configurations.

The basic working principle of RPL is as follows: RPL nodes have first to be configured to be either root nodes or not. This can be done by configuring the device directly or through other means. Root nodes send periodically (with a trickle timer) generalized DODAG Information Object (DIO) messages to all nodes in the area. Normal nodes either wait for a DIO message or request one with a DODAG Information Solicitation (DIS) message from anyone, that can hear the message. When a node sees a DIO message it sends a DIS message to the node, to request more information. The node should then respond with a DIO message directed to the sender of the DIS message, containing information about the current RPL instance it is in. The node then adds itself to the RPL instance and also advertises periodically the RPL network with DIO messages. Also the node gives routing information to its parent by sending Destination Advertisement Object (DAO) messages whenever changes occur or when itself receives a DAO message (from its children). When a node now wants to send a message, it is passed upwards through the parents, until a parent (or the root) know whose child the destination is and then passed downwards to this node.

In summary, RPL is a routing protocol for low-power and lossy networks, which provides efficient and reliable routing while minimizing the use of network resources and preserves energy on nodes [4].

2.3 OSPF

Open Shortest Path First (OSPF) is a proactive link-state routing protocol, which means that routing information is distributed in a single autonomous system before any message data is sent [12] [13]. Due to being a link-state protocol, routing information is always sent to one-hop neighbors. In a wireless setting, this means all nodes that are in transmission range. As each node also propagates the information it received further, eventually all nodes in the network know the whole structure of the network. Then all nodes can calculate the best/shortest path between any two nodes. This is normally done by using the Dijkstra algorithm [14].

The basic working principle of OSPF is as follows: Nodes broadcast OSPF HELLO messages to all one-hop nodes in range at regular intervals to discover neighbor nodes. These HELLO messages are not propagated further. Furthermore, these messages contain the senders identification number as well as all neighbors the sender has already discovered. With these messages, each node knows eventually who his neighbors are. Furthermore these messages are also used to determine if a neighbor has become inactive or lost connectivity, when no HELLO messages are received anymore. To exchange routing information Link State Advertisements (LSA) messages are used, which distributes the state of the nodes connections. The messages are broadcasted whenever the network updates for a node. Upon reception of an LSA message, a node updates its Link State Database (LSDB), in which the node stores its current view of the network topology. Eventually, every node in the network will have the same view of the network topology.

The problem with OSPF is that it scales poorly. With an increasing number of nodes the size and frequency of the topology updates, as well as the length of time it takes to calculate end-to-end routes increases drastically. This is the reason OSPF is only used within an autonomous system.

With the OSPF version 3, OSPF is updated to support IPv6 by adding compatibility with the 128-bit address space. This version makes some slight changes to the protocol behavior. These changes are that the protocol processing is now per-link and not per subnet, the Flooding scope for LSAs has been generalized and authentication has been removed from the OSPF protocol, which now shall handle the IPv6's Authentication Header and Encapsulating Security Payload.

2.4 AODV

Ad hoc On-Demand Distance Vector Routing (AODV) is a reactive routing protocol, which means routes are only created when they are needed [15]. Each node of the network keeps track of the network with a routing table and uses sequence counters in its messages to prevent routing loops.

The basic working principle of AODV is as follows: Whenever a node wants to send a message to a new node, that is not already in it's routing table, it needs to discover a route to said node. The source node sends a Route Request (RREQ) message as broadcast. When a RREQ message is received by a node: Either the node is or has a valid route to the requested target (a route with a sequence number greater or equal to the one in the RREQ message), then it responds with a unicast Route Reply (RREP) message back to the source. Otherwise the RREQ message is forwarded (by broadcast) with an incremented hop count. As RREQ messages can be uniquely identified by the source address and request ID pair, receiving multiple of the same RREQ message, additional messages are dropped by the receiver. RREQ messages are sent periodically if no RREP message is received. When a node the receives a RREP message, it can update its routing table and can begin to send messages to the target [16] [15].

AODV can be further optimized by using an expanded ring technique, by adding a Time To Live (TTL) to the RREQ message to keep track on how often the message is re-broadcasted. This reduces the number of hops a RREQ message traverses through the network. When the number of hops is greater than the TTL, the requesting node has to send another RREQ message with a higher TTL. Although the cost can sometimes be higher than full flooding, the expected overall cost is less with a set of optimal chosen TTL values [17] [15].

Also to keep track of Neighbors AODV can be configured to use HELLO messages, as already mentioned in Section 2.3. These HELLO messages are periodically broadcasted locally and are not forwarded further. When a node receives a HELLO message from another node, it updates the neighbor in its routing table to be reachable and (still) alive. As this behavior is more expected from a proactive protocol, rather than an reactive protocol, this feature is optional and separate to the general topology management [15].

2.5 OLSR

Optimized Link State Routing (OLSR) is a proactive routing protocol for mobile ad hoc networks with focus to satisfy the needs of mobile Wireless Local Area Network (WLAN) [18]. As the protocol is proactive, the routes are always available when needed. The main gimmick of OLSR is that it chooses a subset nodes in the network to be Multipoint Relays, which are the only nodes that can forward data through the network. This is done to reduce the flooding in a network.

The basic working principle of OLSR is as follows: As in Section 2.3, nodes send periodic HELLO messages to inform potential one hop neighbors of their existence. Furthermore the HELLO messages contains also a list of all already found neighbors of the sender. This means, that eventually all nodes then know their one and two hop neighbors and know if the connection to another host is symmetrical or asymmetrical, which will be stored in each nodes routing table. Each node then chooses a small set of MPR nodes, which is the Multipoint Relay Selector Set (MPR Selector Set). To reduce the amount of flooding, the minimum number of one hop neighbors which cover all two hop neighbors are chosen with the MPR selection algorithm. Each node then broadcasts periodically Topology Control (TC) messages, which contains the MPR Selector Set and a set of its own advertised links. Only MPR nodes are allowed to forward TC messages. Furthermore TC messages contain a sequence number to drop older messages and prevent loops. Upon receiving a TC message, a node updates it routing table [16] [18].

2.6 IS-IS

Intermediate System to Intermediate System (IS-IS) is a link state routing protocol like OSPF. IS-IS is based on Connectionless Network Service (CLNS), which means that it not necessarily needs IP Addresses to work, but can use them. IS-IS designates areas (level 1 and 2) where routers can only exchange information in the level they are. Additionally routers (level 1/2 routers) that are on the border to both areas can exchange information with all level routers. To which level a node belongs is set on each node by the network administrator.

The basic working principle in each area works as OSPF described in Section 2.3, such that HELLO messages are periodically sent to know all adjacent neighbors. Furthermore, it uses Link State Protocol Data Unit (LSP) messages, which are periodically flooded in the same area, to share network information. Level 1/2 nodes flood them to all adjacent nodes. As in OSPF, the best/shortest path between any two nodes is normally calculated by using the Dijkstra algorithm [14] [19].

2.7 RIP and RIPng

Routing Information Protocol (RIP) is an Interior Gateway Protocol (IGP) that uses a distance-vector algorithm to determine the best route to a destination. RIP uses the number of hops (hop count) as the metric [20].

RIP broadcasts routing information to all directly connected RIP nodes every 30 seconds, which will update their routing table. A routing update contains the entire routing table of the sending node. When a node receives such an update it will update its own routing table. If a RIP node has not received a routing update from another router for 180 seconds, it assumes the RIP node is down, sets the metrics of all routes involving this router to infinity and stops routing over this node. Infinity in RIP equals 16 or more hops to the destination. Therefore only routes with maximum 15 hops to a destination are possible. If no update was received for further 120 seconds the router will completely remove all routes involving the outer router. To calculate the best route to another node, the Bellman-Ford algorithm is used [21] [22].

Routing Information Protocol Next Generation (RIPng) is different to RIP as it uses link-local addresses to keep track of the next-hop addresses of the neighboring routers and uses the FF02::9 address to exchange routing information [23].

RIPng routers only share network information with each other and utilize the IPv6 link-local addresses to keep track of the next-hop. This is problematic in a wireless environment as only networks are forwarded and hosts are only reachable if they are link-local so the sender itself. If a message shall be delivered to a node in the same network, but not in its direct sending range, the message will not be forwarded to the node, as no node has the information to forward the message [23].

2.8 EIGRP

The Enhanced Interior Gateway Routing Protocol (EIGRP) is a distance-vector routing protocol.

Unlike RIP, EIGRP does not send periodic routing updates to other nodes, except for HELLO messages. This prevents loops from happening and reduces the network load.

The basic working principle of EIGRP is as follows: Each EIGRP node sends periodic HELLO messages as broadcast/multicast, as in Section 2.3, to all one hop neighbors. Each node then eventually knows all adjacent neighbors, as well as if they are still alive. When ever the cost or the status of a link changes for a node, it sends an UPDATE messages to all one hop neighbors, which contains the distance vector of the changed link. Upon receiving such message, a node updates its routing tables, as well as it's topology table, in which it keeps track of all the distances to each known destination [24]. To calculate the best route to another node, the Diffusing Update Algorithm (DUAL) is used [25]. EIGRP nodes have the option to send EIGRP packets reliably (e.g. for UPDATE messages), by indicating in HELLO message, that the next multicast message requires an acknowledgment from every one hop neighbor [26].

2.9 Routing Protocol Comparison

To give a quick overview of the Routing Protocols and for an easier comparison, the properties for each routing protocol is given in table 2.1. This table showcases how the routing protocols work and one can select the best fitting protocol for a specific use case.

Properties	**RPL**	**OSPF**	**AODV**	**OLSR**
Type	Distance Vector	Link State	Distance Vector	Link State
Activity	Proactive	Proactive	Reactive	Proactive
Topology	DODAG	SPF Tree	Mesh	MPR (Multi-Point Relays)
Topology Discovery	Utilizes DAG	Hello Messages and LSAs	Route Request	Hello Messages and Topology Control
Use Case	IoT, Sensor Networks	Enterprise, Internet	MANETs	MANETs
Algorithm	DODAG	Dijkstra	Distributed Bellman-Ford	Dijkstra
Routing Metric	OF	Cost (Bandwidth, Delay)	Hop Count	Link Quality (ETX)
Routing Updates	Periodic, Event-Driven	Periodic, Event-Driven	On-Demand	Periodic, Event-Driven
Loop Prevention	DAG Rank, Parent Set	Split Horizon, Poison Reverse	Sequence Numbers	MPR selectors
Concurrency	Multipath	Multipath	Single Path	Multipath

Properties	**IS-IS**	**RIP**	**EIGRP**
Type	Link State	Distance Vector	Hybrid
Activity	Proactive	Proactive	Proactive and Reactive
Topology	SPF Tree	Distributed	DUAL
Topology Discovery	Hello Messages and PDUs	Periodic exchanges	Hello Messages
Use Case	Enterprise	small networks	Enterprise, Internet
Algorithm	Dijkstra	Bellmann-Ford	DUAL
Routing Metric	Cost (Bandwidth, Delay)	Hop Count	Cost (Bandwidth, Delay,...)
Routing Updates	Periodic, Event-Driven	Periodic, Event-Driven	Incremental
Loop Prevention	Split Horizon	Split Horizon	DUAL
Concurrency	Multipath	Multipath	Multipath

Table 2.1: Comparison of Network Protocols

CHAPTER 3

Network Simulators

3.1 What is a Network Simulator

As networks grow larger and more complex in the last few years, the effort to test and experiment on such networks is not feasible anymore. Therefore, network simulation has become increasingly important to analyse networks and predict their behaviour [27]. Network simulation is often used to predict the performance of networks and their used protocols, to test certain behaviour of protocols for correctness and to explore protocol designs by rapid evaluation and iteration. In operation, network simulators let the users design, modify and test networking protocols in a simulated environment, which is modeled with devices, links, application, etc. to test on [28].

In contrast, developed protocols can be implemented and deployed on real hardware, which is called testbed implementation. While such an implementation may yield more accurate results than a network simulator, the drawbacks are that hardware can become quite costly, monitoring options can be limited, the test environment can be difficult to set up and the system can be prone to measurement errors [29].Therefore, testbed implementation is more likely used with a smaller number of nodes.

Network simulators can be classified into two types: software-based and hardware-based. Software-based network simulators are the more commonly used type of network simulators. These simulators are software programs in which networks can be modelled and simulated and show the behaviour and traffic. Hardware-based network simulators are designed to simulate specific hardware and are designed to test specific hardware such as routers and switches under different network conditions. Unfortunately it can be difficult to use network simulators, as most users struggle with installing most used network simulators, as well with configuring the simulation model properly. This may deter users from using network simulators and suggests that there is an importance to improve documentation [28].

As most simulation models make use of abstractions and approximations from reality, a simulation of these model is often trimmed to answer a specific set of questions. Therefore, simulation models are inherently not designed to accurately depict reality, but rather are geared towards a specific use/test case. In contrast, laboratory experiments can also prove to be invalid, e.g. when measurement artefacts appear or the inevitable errors when measured results are extrapolated from small networks to large ones [27].

Additionally prior works, conducted on network simulators, show that it is hard to select the correct network simulator for the desired use case, because of wide variations in operating systems, hardware requirements, programming software requirements, output features and scalability [30] [31] [32]. To ease the choice which simulator to use, the properties of the later described network simulator is given in Table 3.1, which is slight altered from [30]

As wireless networks have other demands it can be even more challenging to pick a suitable network simulator, as nodes can be resource constrained and link can heavily fluctuate [29].

The general work flow to use a network simulator is the following:

1. Define the network topology: Use the network simulators interface to create nodes, specify their connections, and configure their network interfaces to form the desired network topology.
2. Define application behaviour: Implement the application logic, specifying the behaviour of each node, including its applications. Also set the amount of data sent, the timing of transmissions, and the protocol used.
3. Configure simulation parameters: Set simulation parameters such as the start and end times, the random start seed, and any specific simulatior options.
4. Build the simulation model: Use the tools from the simulator to compile the simulation model into an executable file or script.
5. Run the simulation: Use the simulators interface or command line interface to run the simulation and generate trace files.
6. Analyze the results: Use the simulators tools or external tools to analyse the trace files and generate statistics, such as throughput, delay, and packet loss.
7. Visualize the results: Use the simulators tools or external tools, such as wireshark or RapidMiner, to create visualizations, such as packet flow diagrams or timeline charts, to help understand the simulation results.

In conclusion, network simulators are an essential tool for anyone researching or developing networks. Network simulators provide an effective way to test and evaluate network behaviour under various conditions, helping to improve network performance and reliability.

Simulator	License	Language	GUI	Ease of Use	Documentation Quality	Parallelism
ns-3	GNU GPLv2	C++	Yes	Moderate	Good	Yes
OMNeT++	Academic Public License	C++	Yes	Moderate	Good	Yes
J-Sim	Proprietary	Java	Yes	Easy	Fair	No
COOJA	GNU GPLv2	Java	Yes	Easy	Fair	No

Table 3.1: Comparison of Network Simulators

3.2 ns-3

ns-3 is an open-source discrete-event network simulator that is widely used in the field of network research and development. It is a simulation framework that allows to simulate the behaviour of complex network systems, including wired and wireless networks, cellular networks, and satellite networks. ns-3 is mainly used for research and educational use [30].

It is written in its entirety in C++ in which it's modules and use cases are also made. Additionally optional Python bindings are also available. Therefore it is a prerequisite to know how to write Python or C++ code to use this network simulator. Other than the language barrier, it provides a user-friendly simulation environment, making it easy to create and modify simulation scenarios. ns-3 provides a wide range of features, including an advanced event scheduler, a powerful and extensible modular architecture, and a comprehensive set of network models and protocols.

One of the advantages of ns-3 is its support for a wide range of network protocols and technologies, including IPv4 and IPv6, TCP and UDP, Wi-Fi, LTE, and 5G. It also provides support for packet tracing and visualization, allowing developers to visualize the flow of packets through the network and analyse network behaviour. ns-3 also supports a real-time scheduler, to be able to run a simulation-in-the-loop to interact with real hardware systems. This can be done by emitting and receiving ns-3-generated packets on real network devices, where ns-3 serves as an interconnection framework [33].

Overall ns-3 is a powerful and versatile network simulator that provides an effective means of evaluating and optimizing network performance in a virtual environment. Its advanced features and support for a wide range of network protocols and technologies make it an essential tool for anyone working in the field of network research and development [34] [33].

3.3 OMNeT++

OMNeT++ is an open-source discrete-event simulator that is widely used in the field of network research and development. OMNeT++ itself is not directly a network simulator. It just includes basic machinery and tools to to create simulations, but it does not include components specifically for computer networks. These applications are added by frameworks, such as the extensible InterNETworking framework (INET) framework or Castalia. Through this framework approach, this simulator provides users with a more generic component architecture, as these models can be freely designed and used in different environments as building blocks [33].

OMNeT++ is written in the C++ programming language and provides a wide range of features, such as an event scheduler, a graphical user interface, and a flexible and customizable simulation kernel. It also provides a comprehensive library of network models and protocols, making it an ideal tool for simulating a wide range of network scenarios. The components itself are connected and configured with the NEtwork Description (NED) language, the OMNeT++ own topology description language. One of the advantages of OMNeT++ is its ability to simulate both network protocols and application behaviour simultaneously, allowing to evaluate the performance of network applications under various conditions. OMNeT++ also provides support for parallel simulation, which enables simulations to be run more efficiently on multi-core processors or distributed computing clusters. Overall OMNeT++ is a powerful and versatile simulator that provides an effective means of evaluating and optimizing network performance in a virtual environment. Its advanced features and extensibility make it an essential tool for anyone working in the field of network research and development [35] [33].

3.4 J-Sim

J-Sim is an open-source network simulator designed for simulating communication networks. It is a tool that allows to test and evaluate the performance of network protocols and algorithms in a virtual environment, before deploying them in the real world.

J-Sim is written in the Java programming language and provides a wide range of features, such as network visualization, packet tracing, and link and node failures modelling. J-Sim is built upon Autonomous Component Architecture (ACA) and INET. By having an ACA, components can be individually designed, implemented and tested in J-Sim. When data arrives at an input of a component an individual routine of the component takes care of it in an independent execution context. This makes it that components, i.e. nodes, behave like real Integrated Circuits. INET lets J-Sim have access to create and used packet-switched networks. Also J-Sim provides a scripting framework and a graphical user interface.

One of the advantages of J-Sim is its flexibility, as it allows to customize and extend the simulation environment to meet their specific needs. J-Sim can simulate different types of networks, such as Local Area Networks and Wide Area Networks. Furthermore, it can be used to evaluate the performance of different network protocols and algorithms. Overall J-Sim is a tool for anyone working in the field of network protocol development, as it provides an effective means of evaluating and optimizing network performance in a virtual environment [36] [33].

3.5 Contiki COOJA

COOJA is a network simulator designed specifically for simulating WSN and IoT systems. Furthermore, it is a hardware-based open-source simulator that allows to test and evaluate the performance of WSN applications and protocols in a virtual environment before deploying them in the real world. The Contiki project also developed the Contiki OS, which is an embedded OS entirety written in C.

COOJA is a Java-based simulator, and provides a wide range of features such as network visualization, energy profiling, and radio interference modelling. It also supports a variety of network protocols and can simulate different types of sensor devices with different hardware capabilities. When using the Contiki OS in COOJA, the simulator can give additional information on energy consumption and running applications.

One of the advantages of COOJA is that it provides a user-friendly graphical interface that allows to set up and configure their simulation, such that the node can be freely placed, sensor input can be modified and communication noise can be set up. Additionally, COOJA can be extended and customized with plugins, making it a versatile and flexible tool for network simulation.

COOJA can simulate on different abstraction level: the network level, the code level and the instruction-set level. Simulation on the instruction-set level uses a compiled firmware binary by invoking an external microcontroller simulator.

Overall, COOJA is an essential tool for anyone working in the field of WSN and IoT development and it provides an effective means of evaluating and optimizing network performance in a virtual environment [37].

3.6 RPL in the Network Simulators

The performance of RPL was studied in the papers [7] [6] [5] [38] [39]. In [7] it was shown that RPL is a powerful technique, which gives networks a quick set-up with bounded communication delays. It is also shown that node distance to the sink node plays a large role. Also they concluded that the protocol overhead was quite high compared to the data traffic. In [5] it was shown that the performance of RPL is heavily dependent on the OF. This makes RPL harder to use, as of currently there is no clear guideline on how to specify this OF to optimize the protocol. Also [38] shows the how the general OF, which is based on the hop count, to a OF that tries to preserve more energy on nodes. In [39] it is stated that RPL performs better in most cases than Software Defined Networking (SDN) on a mesh topology.

As previously stated, there are already several existing network simulators available, in which RPL implementations can be used. These are OMNeT++,J-Sim, COOJA and NetSim. Each of these simulators has its strengths and weaknesses, and the choice of simulator may depend on the specific research goals and requirements. One important aspect of RPL simulation in network simulators is the modeling of the underlying network topology.

Implementations of RPL in OMNeT++ are presented in [10] and [6]. The implementation in both papers is evaluated through simulation experiments that measure the performance of the RPL protocol in terms of network lifetime, packet delivery ratio, and end-to-end delay. The paper [10] highlights the importance of testing and evaluating the importance of RPL before deploying it on hardware in the real world. The paper [6] suggest that the code that implements RPL in OMNeT++ cannot keep up with the updates the network simulator receives. OMNeT++ is probably the most used simulator to simulate RPL, as a lack of better options. While OMNeT++ can simulate RPL, it is very tedious to use, as seen in [6], as OMNeT++ does not provide built-in support for RPL. Furthermore, in [6] it is mentioned, that there are shortcomings related with the physical layer. With these deficiencies this simulator does not seem like an attractive option.

Then there is a RPL implementation in the hardware simulator COOJA, as these paper show [7][9][40][11]. These paper presents a thorough analysis of the performance of the RPL routing protocol in various scenarios, using the COOJA network simulator. These studies investigate the impact of multiple parameters, such as network size, traffic load and different topologies, on the performance of RPL. As it suggests in [6], nodes have memory limitations, which means that nodes cannot run implementations, that need a lot of space. Furthermore, since COOJA is a hardware simulator, the devices used, can become outdated and it takes significant effort, in order to add newer hardware to the simulator. While being able to more realistically simulate a RPL network, a hardware simulator has the drawback that it is much more computation intensive and is far less flexible [6]. Furthermore, COOJA has dramatically fewer features in comparison to other established simulators.

The paper [8] proposes a simulation model for the RPL protocol in the J-Sim network simulator. The simulation results show that the model could successfully simulate the RPL protocol and its behavior in a wireless sensor network. It also briefly describes open issues on the RPL protocol, such as the support on multiple instances and routing strategies based on different metrics. Unfortunatly the J-Sim RPL implementation code from [8] was not sourceable.

Lastly there is a RPL implemtation in NetSim. As this is a commercial product, which can simulate Cisco systems, it can not be freely used.

As shown, the drawbacks of these RPL implementations in the different simulators will necessarily lead to evaluations that will ultimately be flawed. Additionally most communicated RPL evaluations, and more importantly their implementations, are difficult if not impossible to come by. Therefore, this work shall give a proper RPL implementation in ns-3 that can be correctly evaluated.

CHAPTER 4

Design of Experiments

To evaluate RPL in ns-3, different scenarios are created to test its behavior. Those examples are described in the following sections. Here is some configuration listed, which is used in every example, unless the specific example says otherwise.

Nodes in every example are placed in a 2 dimensional space, i.e. every node is positioned on the same height on the z-axis to reduce complexity. Every example uses LR-WPAN devices, which are all associated to the same Personal Area Network (PAN)-ID. Furthermore every node uses 6LoWPAN with the same context configured. Every node is associated to the IPv6 Network "2001:2::", which is also set as context for 6LoWPAN, such that node 0 has "2001:2::ff:fe00:1", node 1 has "2001:2::ff:fe00:2" and so on. Furthermore, every node is set to be able to forward messages to another node. As described in Section 5.2.3, the classical Neighbor Discovery Protocol (NDP) is used normally for every example. On top of this, when not stated otherwise, RPL is used as routing protocol.

In all examples one or more nodes (clients) send User Datagram Protocol (UDP) packets to another node (server), which echos the packets back to the sender (clients). To ensure that the network is well established and stable, the clients start sending UDP packets every second, 100 seconds after the simulation is started. In the case of the line example and the many nodes example, the application starts after 200 seconds, as so many nodes need a longer setup time. As the size of the packets does not really matter, they are always set to 10 bytes per packet. Since UDP traffic requires a port, the port 6000 is chosen, to not be in the range of the well-known ports [41]. Each example that is evaluated with RPL, the default configuration of RPL, as stated in [4], is used. This also means that only Objective Function Zero (OF0) is used. Lifetime values are all set to 0xFFFFFFFF, as they need to be long enough to not interfere with the evaluations. Furthermore, the DAO delay is set to 1 s, as specified as default value in [4]. Also how often a DAO message is sent before a node gives up on it is set to 5 and a node will wait 10 s for a Destination Advertisement Object Acknowledgement (DAO-ACK) before it resends a DAO message. In case of RPL, node 0 is always taken as the root node of the network. The RPL configuration values are given in Table 4.1.

Root	node 0
RPL Instance ID	0
RPL Objective Code Point	0
RPL mode of operation	2 (equals Storing Mode without Multicast)
K-Flag (decides if DAO-ACK messages are sent)	true
DIO Interval Doublings	20 (equals 2.3 hours)
DIO Interval Min	3 (equals 8 ms)
DIO Redundancy Constant	10
Minimum Hop Rank Increase	256
DAO Delay	1 s
DAO Retries	25
DAO-ACK Wait Time	2 s

Table 4.1: RPL Configuration

4.1 Line Example

The line example features nodes positioned in a straight line, as shown in Figure 4.1. The nodes in this example are set up in a way, that each node only sees at most 2 neighbor nodes, by setting the distance between nodes accordingly. In this case, each node is 70 meters apart from each other. The in- and outgoing traffic of each node can then be easily analyzed, as there is only one in- and outgoing path. To generate some traffic, node 0 is configured as a client and node n, the node furthest away from node 0, is configured to be the server. Therefore after the initial 100 seconds, node 0 will send a UDP packet every second to node n, which node n will echo back to node 0. When using RPL in this example, the root node (node 0) is positioned on one end.

For testing basic functionality of message passing this example well suited. Furthermore, in this example we can observe how long it takes for nodes to be added to the network with a varying number of nodes in the network. In most routing protocols, this can be observed by looking at the routing table and see when the last node has been added. For RPL, the time is expected to linearly increase with the number of nodes, as with each new hop, the configured DAO delay adds up. The easiest case to see packet forwarding is with 3 nodes, such that the middle node can be seen to forward the packets.

Therefore, the line example is tested for:

- Network Setup Time: The median time when node 0 has a route to the nth node.

- Success Rate: The number of unique UDP messages that are successfully echoed back to node 0 divided by the number of unique UDP messages sent by node 0.
- Latency: The median round trip time for the UDP messages sent from node 0 until the echo is received again on the node.

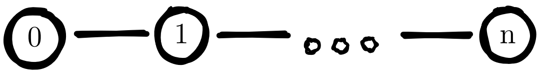


Figure 4.1: Line Example

4.2 Tree Example

The example shall showcase the path generation of the routing protocols and how quickly paths are found. In this example, the node are positioned in a tree like topology. The nodes are placed in such a way, that each node can only see the next hop node to the root. This is done to force the network to ensure the paths are created as shown in Figure 4.2. The positions of the nodes used in this example can be seen in Table 4.2.

In this example to simulate two real world applications, traffic is either going from a gateway to its sensors or inter-sensor-communication shall be shown by having two leaf node send packets to each other. Explicitly one setup is that each leaf node in the network sends the UDP packets to node 0, which will echo the packets back to the respective leaf node. The second setup is that node 5 sends UDP packets to node 7, which will echo them back to node 5. Finally, as a control test, node 7 is configured as client and node 0 is configured as server, which will echo the packets back to node 7. The setups can be seen in Table 4.3. In all setups, packets are sent every 1 second.

Furthermore, to better understand the scalability of the routing protocol the quantitative metrics latency and success rate are measured [42]. These quantitative metrics are latency and success rate [42]. The latency shall be the average amount of time a message takes from the sender to the receiver. The success rate shall be the total number of unique packets delivered divided by the total number of packets sent.

Therefore, the tree example is evaluated for:

- Network Setup Time: The median time when node 0 has a route to every other node in the network.
- Success Rate: The number of unique UDP messages that are successfully echoed back to the sender node(s) divided by the number of unique UDP messages sent by the sender node(s) for all setups.
- Latency: The median round trip time for the UDP messages sent from the clients until the echo is received again on the clients in all setups.

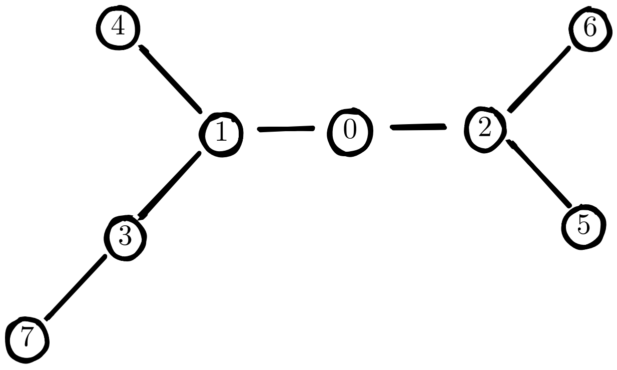


Figure 4.2: Tree Example

Node	X Position / m	Y Position / m
0	1000	1000
1	940	1000
2	1060	1000
3	880	1050
4	880	950
5	1120	1050
6	1120	950
7	820	1100

Table 4.2: Tree Example Node Positions

Setup	Client	Server
1	4, 5, 6, 7	0
2	5	7
3	7	0

Table 4.3: Tree Example Setups

4.3 Alternate Path Example

This example shall showcase how responsive a routing protocol is when a link fails, e.g. by a battery failure or a node moving out of the network. After some time, e.g. by replacing the battery of the node or the node moving back into the network, the broken link then reconnects with the network.

The network is built as shown in Figure 4.3, such that node 4 has potentially two paths to reach node 0. To be able to show the broken link and the re-connection to the network, node 1, shown in Figure 4.3, fails after 130 seconds and reactivates after 300 seconds. For RPL, by using OF0, the preferred path for node 4 is over node 1, as this path has fewer hops than the path over node 3. The exact node positions are shown in Table 4.4. Node 4 is configured to be the client that sends a UDP packet every second to the server node 0, which will echo back the packet to the client. In RPL, node 0 is chosen as the root node.

The delay between the initial path becoming unavailable and the network being ready to send messages over the alternative path shall be measured. Furthermore, if the routing protocol is able to, the time it takes restore the initial path again shall also be measured. This shall show how flexible the routing protocol is and how well it handles changes in the network. For RPL, it shall show that the alternative path is found and later, the better path over node 1 shall be found again. It shall be kept in mind, that in this RPL implementation only OF0 is used, which can give highly different results than with another OF. How long RPL takes to find the new paths is depending on when specific DIOs are received. This may introduce wildly varying delays over different simulation runs. To further investigate this, the trickle timer parameter will be changed to see how much impact they actually have.

Therefore, the alternate path example is evaluated for:

- Alternate Path Time: The median time it takes node 4 to have a route to node 0 over node 3 and 2 after node 1 fails with different Trickle Timer configurations as shown in Table 4.5.
- Better Path Time: The median time it takes node 4 to have a route to node 0 over node 1 after it reactivates with different Trickle Timer configurations as shown in Table 4.5.
- Success Rate: The number of unique UDP messages that are successfully echoed back to the sender nodes divided by the number of unique UDP messages sent by the sender nodes.

Node	**X Position / m**	**Y Position / m**
0	100	0
1	50	70
2	160	60
3	160	140
4	80	150

Table 4.4: Alternate Path Example Node Positions

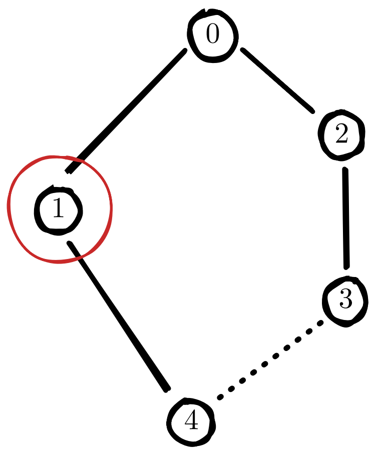


Figure 4.3: Alternate Path Example

DIO Interval Doublings	DIO Interval Min	resulting DIO Interval Max
20	3	2.33 h
13	3	1 min
11	3	16.38 s
10	3	8.192 s

Table 4.5: Alternate Path Example Trickle Timer Configurations

4.4 Many Nodes Example

To test how scale-able and flexible routing protocols are, an example with a significantly higher number of nodes compared to the previous examples is used. This is sketched in Figure 4.4. The number of nodes will be varied over multiple runs. The nodes in this example are either uniformly or normal distributed.

The area the nodes are dispersed when uniformly distributing is dependent on the number of nodes. This is done by using the next perfect square of the number of nodes as grid width and then scaling it with a factor α. The grid width G in both x and y direction is calculated by the following formula by using the number of nodes n as argument.

$$G = (\lfloor\sqrt{n}\rfloor + 1) \cdot (\lfloor\sqrt{n}\rfloor + 1)$$

Then the Box for the uniform distribution is set to be $\alpha G \times \alpha G$, where $\alpha = 50$. For the normal distribution the same grid width G is used, such that the mean is $\mu = \frac{G}{2} \cdot \alpha$ and the variance is $\sigma^2 = \frac{5}{3} \cdot \alpha$, where $\alpha = 50$.

Those distributions are chosen to represent real world applications. Uniformly distributed nodes could represent an outdoors sensor network, whereas normal distributed nodes around a root in the middle could be found in an office building, where nodes may be clustered together with only a few outliers.

As in the tree example, described in Section 4.2, one time, traffic is setup to go from one or more node(s) to another node, where the nodes are placed by the distribution function.

Explicitly this means that one node is used as server node and other distributed node(s) are used as clients, which send the UDP packets every second to the server, which will echo them back. The number of clients used in each distribution setting are 1, 2 and 4. It shall be noted, that client and server cannot be on the same node in this example, as well as the root node in RPL is never chosen as client or server. Also as in Section 4.2, the data packets are sent every 1 second. When using RPL, node 0, which is also randomly distributed, is configured as the root node. The used setups in conjunction with the distributions are also shown in Table 4.6.

It shall be measured how fast the complete network can be reached from the server node. In RPL, this provides a good metric, as when the root can reach a node, every other node in the network can also reach it. It is expected that the time to reach every node in the network would increase logarithmically with increasing number of nodes regardless of the distribution, as the hop count is also expected to increase nearly logarithmically. This should also hold true for the uniformly distributed case, as the area the nodes are placed in is limited, as shown by the formula above.

As this example is the deterministic large-scale example to the tree example in Section 4.2, the same scalability metrics shall also be measured.

Therefore, the many nodes example is evaluated for:

- Network Setup Time: The median time when node 0 has a route to every other node in the network in both distributions over varying node count.
- Success Rate: The number of unique UDP messages that are successfully echoed back to the sender nodes divided by the number of unique UDP messages sent by the sender nodes in both distributions.
- Latency: The median round trip time for the UDP messages sent from the clients until the echo is received again on the client nodes in both distributions.

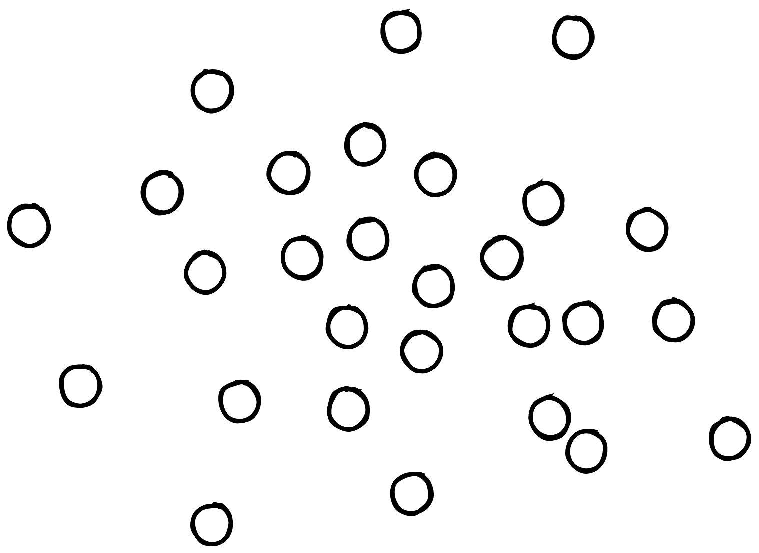


Figure 4.4: Many Nodes Example

Setup	Distribution	Client	Server
1	uniform	random node	random node
2	uniform	2 random nodes	random node
3	uniform	4 random nodes	random node
4	normal	random node	random node
5	normal	2 random nodes	random node
6	normal	4 random nodes	random node

Table 4.6: Many Nodes Example Setups

4.5 Moving Node Example

This example shall represent a mobile device which moves through a (sensor) network which are fixed at their location. Arguably a normal or uniform distribution would reflect the real world better, but would also introduce more uncertainties to the simulation due to the randomness of the positioning. Furthermore, placing the nodes in a grid should not differ much from a closely normal distributed network of nodes.

In this example, the nodes are arranged statically in a 5×5 grid, where each node is 100 m apart to the next one in x and y direction. Furthermore, one node (node 25) moves through this grid. The moving node starts on the same position as the center node 12. Then node 25 moves to node 14, then node 4, then 200 m to the right and finally to node 0. The node moves with a speed of 2 m/s and pauses at each destination for 50 s. The topology and the movement of the node is shown in Figure 4.5. After the pause of 50 s at the last destination, the simulation shall be terminated. The moving node 25 takes the role of the client and node 12 of the server. This means that node 25 sends packets to node 12 every 1 second which are echoed back. Using RPL node 12 is chosen as the root node and node 25 is declared as a leaf node, to prevent it from having a sub-DODAG itself.

To track how good the connection from the network to the moving node is the latency of the messages from and to the moving node shall be measured, as well as the overall success rate of packages. To test the limits of the routing protocol, the success rate is evaluated over varying speed of the moving node. It is expected, that the success rate will drop quite fast, as the moving node will move out of range before it even has the chance to connect to a new node.

In case of RPL, the parent list of the moving node shall be evaluated, to see when and with whom the node connects. This of course does not necessarily correspond with the real world, as in this simulation each node can only have one parent and only OF0 is used.

Therefore, the moving node example is evaluated for:

- Success Rate: The number of unique UDP messages that are successfully echoed back to the sender nodes divided by the number of unique UDP messages sent by the sender nodes. The success rate is also evaluated over varying speeds and trickle timer parameters as shown in Table 4.7.
- Latency: The median round trip time for the UDP messages sent from node 25 until the echo is received again on the node.
- Parent List: The parent list of the moving node until the final stop at node 0.

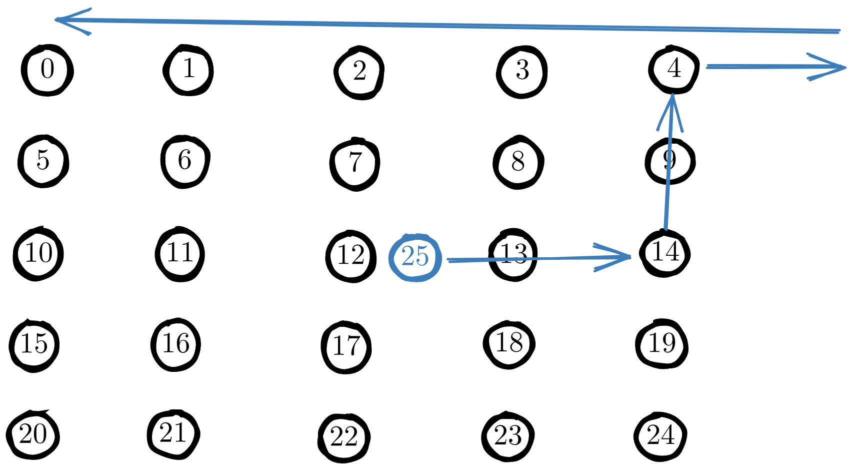


Figure 4.5: Moving Node Example

Speed[m/s]	DIO Interval Doublings	DIO Interval Min	resulting DIO Interval Max
2	20	3	2.33 h
2	10	3	8.192 s
3	20	3	2.33 h
3	10	3	8.192 s

Table 4.7: Moving Node Example Configurations

4.6 Tree with Strict Subtrees

This example is modeled after the example shown in [10], Chapter 7.3. Here the network is separated into three “sub-networks“, as seen in Figure 4.6. The left and the right “sub-networks“ can only be reached over nodes 4 and 5, respectively. The middle “sub-network“ can be reached over all nodes 4 to 8. The exact positions of the nodes are not given in [10]. Therefore, positions for the nodes, as shown in Table 4.8, are chosen, which shall emulate the structure shown in Figure 4.6. Node 0 is configured to be the server node, while the leaf nodes (nodes 34 to 42) are configured to be client nodes. As described in Chapter 4, the leaf nodes will send a packet every second to the server node, which will echo back the packet to the leaf nodes. Furthermore, node 0 shall be the root node in RPL.

The energy consumption on all nodes with a hop count distance to the root node of 2 (nodes 4 to 8) shall be measured. The energy consumption of LR-WPAN packets in ns-3 cannot be measured, since there is no built in energy model for LR-WPAN as of writing this, as the model discussed in [43] never made it to ns-3. It is expected that the nodes 4 and 5 will be drained faster, i.e. received more packets, than the ones in the middle, as they have to pass the messages to their respective “sub-network“. This can then be compared to the evaluation done in [10]. This comparison can only be done with the hop count metric used in [10], as this is basically OF0.

Furthermore, the amount of DIO messages sent by each node can be reduced, by increasing the DIOIntervalMax[1]. By having less DIO messages sent, the nodes should preserve more energy over time. This can then also be compared to the results in [10] by setting DIOIntervalMax to an equivalent of 10 seconds and 20 seconds.

Therefore, the special network example is evaluated for:

- Energy Consumption: The number of sent and received LR-WPAN packets (i.e. the approximated energy consumption) as well as all outgoing UDP packets on all nodes with a hop count distance to the root node of 2 over the time with different Trickle Timer configurations as shown in Table 4.9. This can then be compared to the energy consumption in [10].

[1]The terms DIOIntervalMax and doublings value are interchangeable used in this document.

- Network Setup Time: The median time when node 0 has a route to every other node in the network in both distributions over varying node count.
- Success Rate: The number of unique UDP messages that are successfully echoed back to the sender nodes divided by the number of unique UDP messages sent by the sender nodes.
- Latency: The median round trip time for the UDP messages sent from the leaf nodes (clients) until the echo is received again on the client nodes.

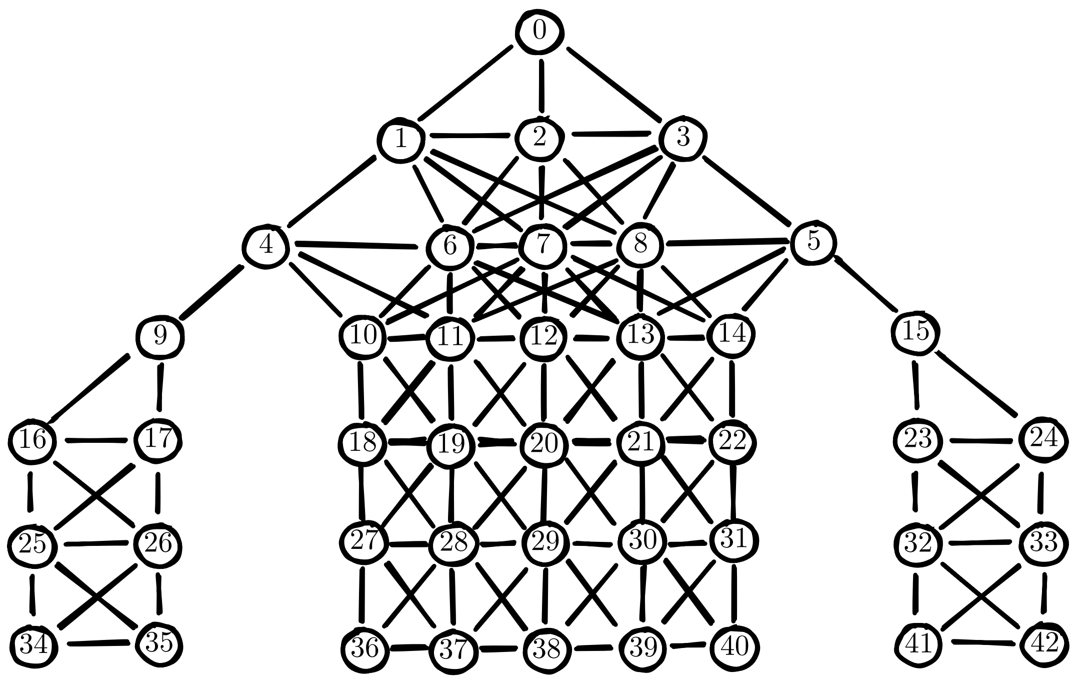


Figure 4.6: Special Network with Sub-Networks Example

Node	X Position / m	Y Position / m	Node	X Position / m	Y Position / m
0	500	0	21	550	280
1	450	70	22	570	280
2	500	70	23	690	280
3	550	70	24	760	280
4	380	140	25	240	350
5	620	140	26	310	350
6	450	140	27	430	350
7	500	140	28	450	350
8	550	140	29	500	350
9	310	210	30	550	350
10	430	210	31	570	350
11	450	210	32	690	350
12	500	210	33	760	350
13	550	210	34	240	420
14	570	210	35	310	420
15	690	210	36	430	420
16	240	280	37	450	420
17	310	280	38	500	420
18	430	280	39	550	420
19	450	280	40	570	420
20	500	280	41	690	420
			42	760	420

Table 4.8: Energy Example Node Positions

DIO Interval Doublings	DIO Interval Min	resulting DIO Interval Max
20	3	2.33 h
11	3	16.38 s
10	3	8.192 s

Table 4.9: Energy Example Trickle Timer Configurations

CHAPTER 5

Design and Implementation of RPL in ns-3

5.1 ns-3 Design

ns-3 consists of several modules that collectively provide a comprehensive simulation environment for network research and development. Some of the key modules are shown in Figure 5.1. Modules that are highlighted in Figure 5.1 are specifically used in this RPL implementation. These modules, along with others not mentioned here, collectively form the foundation of ns-3.

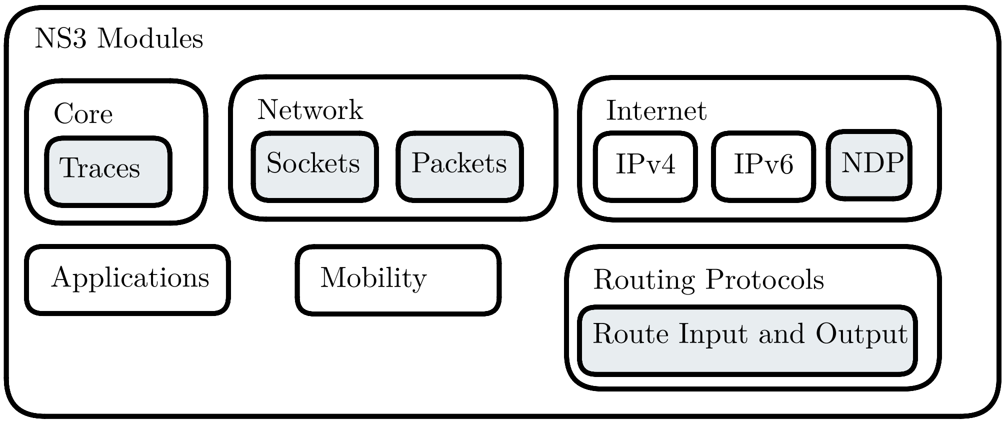


Figure 5.1: ns-3 Key Modules

5.1.1 Core

The core module serves as the foundation of ns-3, providing essential functionalities for ns-3. The module provides time management, the simulator, a scheduler, attributes, traces and other basic functionalities of ns-3. The ns-3 time class can hold and convert between different time units. With the scheduler, simulation events can be created. The simulator is the heart of the network simulator, as it is used to create, schedule and cancel events. Attributes are an easy way to modify modules, without changing the internal structure of a module. [44]

Tracing

The whole point of running simulations in ns-3 is to gather data to analyze for a study. The obtain results, there are two main ways to access it. First is by using pre-defined bulk output mechanisms and parsing their content. This does not require any changes to ns-3. Output messages come in the form of PCAP or `NS_LOG` log messages and have to be parsed/filtered afterwards to be manageable. Furthermore, `NS_LOG` output is only accessable in debug build and is not part of the ns-3 API, which means it can change from one release to the next.

The other mechanism to get output originates from traces. In ns-3, traces are mechanisms used to record and analyze various events or behaviors occurring within simulated network scenarios. It allows selective tracing of events, reducing data management burden, and enables direct output formatting, eliminating the need for postprocessing. By adding hooks in the core, users can access information as needed without excess output. [44]

5.1.2 Network

The network module incooperates how data is exchanged between devices. This is foremost done with sockets and packets.

Sockets

Sockets provide an interface for sending and receiving data between processes, abstracting away the underlying networking details. In ns-3, they facilitate the exchange of data between simulated entities in a network model. In this RPL implementation, the ns-3 `Ipv6RawSocket` is used to send the RPL Internet Control Message Protocol (ICMP) messages.

Packets

The Packets framework in ns-3 was designed with several key objectives in mind, which are:

- Avoid altering the simulator's core for introducing new packet headers or trailers.

- Simplify integration with real-world systems.
- Support fragmentation, defragmentation, and concatenation efficiently, crucial for wireless systems.
- Ensure efficient memory management.
- Allow storage of application data or dummy bytes for emulated applications.

Each packet includes a byte buffer, byte tags, packet tags, and metadata.

The byte buffer of a packet stores the serialized content of the headers and trailers added. This serialized representation is expected to match that of a real packet. Fragmentation and defragmentation are facilitated by the byte buffer, enabling easy integration with real-world code. A challenge of this design is pretty-printing packet headers without context, which metadata addresses. The packet metadata illustrates the type of the headers and trailers, which are written in the byte buffer. By default the metadata is disabled to save resources, but can be easily enabled.

The Packets class allows to add additional data not found in real packets through tags to the packet. Tags can be used for example to give a timestamp to packets or add a hop-by-hop header, such as discussed in Section 5.3.5.

Packet memory management is automatic and efficient, with virtual buffers for application-level payload, minimizing memory allocation. [44]

5.1.3 Internet

The network module adopts the functionalities of the networking protocols of the OSI model. In particular, the module features functionalities for Internet Protocol version 4 (IPv4) and IPv6. Furthermore, the module allows access to protocols like UDP, Transmission Control Protocol (TCP) and ICMP.

Neighbor Discovery Protocol

NDP is a core component of IPv6 networks and facilitates essential functions, such as neighbor discovery, address autoconfiguration, router discovery, and address resolution. It enables IPv6 nodes to manage neighbor relationships, discover routers, configure addresses automatically, and resolve IPv6 addresses to link-layer addresses. NDP replaces Address Resolution Protocol (ARP) of IPv4 and incorporates mechanisms like Duplicate Address Detection (DAD) to ensure the smooth operation and reliability of IPv6 networks. [45]

5.1.4 Applications

While applications in ns-3 can be easily written by oneself, this module offers some basic apps that can be installed on nodes. Furthermore, the module also functions as a base class, to use for own applications. The main purpose of the module is to provide a uniform way to start and stop applications. In this thesis for evaluation, data packages are sent with the `UdpEcho` client and server.

5.1.5 Mobility

The mobility module is used to take care of the placement and movement of nodes in a simulated space. With this module, a node can be placed using cartesian coordinates (x, y, z). If the third axis is not needed, the z parameter can be omitted, and the z position will be set to 0 as default. Additionally, nodes can be placed through different distributions. Furthermore, nodes can be moved in different ways over simulation time, such as moving them to waypoints or doing a random walk. [44]

5.1.6 Routing Protocols

There is no single module in ns-3, in fact there multiple. As each implemented routing protocol in ns-3 would be too much to cover, but there are similarities between them. Therefore, rather than to describe how each routing protocol was implemented in ns-3, it has first to be explained how routing protocols work in ns-3.

Basically routing protocols have two main functions: `RouteOutput` and `RouteInput`. The `RouteOutput` function determines whether and how to send packets generated by a node. The `RouteInput` function determines the appropriate action for incoming packets at the node, including forwarding, dropping, local delivery, and so forth.

In addition to those functions, since both can fail, there is also the option to implement `ListRouting` on a node. `ListRouting` enables a node to be able to use multiple routing protocols at once. So if one routing protocol fails to deliver a package, a second routing protocol can try to deliver it. It can be seen as sort of backup routing. In detail, `ListRouting` checks all routing protocols, it has registered, in order until it finds one with a valid route, in both route input and output functions. In all examples of this thesis, only RPL is used as a routing protocol. In real examples, static routing could be configured as a backup routing protocol to have pre-configured neighbors and to take care of deliveries not considered in the RPL implementation.

5.2 ns-3 Adaptions

As discussed above, for a routing protocol to work in ns-3 it has to fulfill the minimum requirements, such as providing a route in- and output function. Also, it is described below how ns-3 was used to create the routing protocol and which changes were necessary.

5.2.1 Route In- and Output

Like all routing protocols in ns-3, this RPL implementation can be added to an example by creating a `RplHelper` instance and applying it to the `InternetStackHelper`, which itself is installed on the nodes. The RPL protocol can also be configured through the helper, like setting a node to be a root node.

The RPL model itself sets all parameters to the default RPL parameters, unless set differently through the helper. Also the routing protocol sets up a trickle timer, which is started when the node is attached to a Directed Acyclic Graph (DAG) or is a root node. The `RouteOutput` function looks up if it has a route in the routing table. If found it returns the route and if not it will be forwarded to the preferred parent. In case of the root node if an address is not found, it should normally give the packet to a gateway into another network. Since no other networks are tested in conjunction with RPL in this thesis, this case terminates the program with an error. If for any other reason a route is not found, the routing protocol signals to the lower layers that no route is found.

The routing table itself consists of a local DAG for each node, which is updated through the DIOs and DAOs. This means that each node only knows its sub-tree of the network.

In the `RouteInput` function, all link-local packets are dropped, as they shall not be forwarded. Furthermore, it checks if an interface is disabled to forward packets, in which case the packet is also dropped. Since multicast addresses are yet to be implemented, the `RouteInput` function returns false to let another routing protocol take care of this packet. This is done via the aforementioned `ListRouting`. A unicast destination address will be looked up in the node routing table, like in the route output function.

Furthermore, the `RouteOutput` function adds a RPL hop-by-hop header tag to the packets and the `RouteInput` function checks and updates the tag. This is further described in Section 5.3.5.

5.2.2 Sockets

As is usual in ns-3, packets are sent via sockets. For each interface on the node a socket is created, as well as a multicast socket to receive all messages, that are sent to the RPL all nodes address (`ff02::1a`). Interfaces that are excluded by parameterizing the node through the RPL helper, will not create a socket. When a RPL unicast control message is sent, it is sent on all interfaces that are not excluded.

5.2.3 Neighbor Discovery Protocol

RPL and 6LoWPAN are independent of Layer 2 technologies and make no assumptions on them. They were created with IEEE 802.15.4 in mind but were not bound to it. Therefore, any Layer 2 technologies can be used with RPL and 6LoWPAN, but also at least one form of a Link-Layer technology is needed for the routing to work properly.

In ns-3, this leaves the options of a classical NDP [45], an experimental version of 6LoWPAN-ND, developing a new NDP, simply sending all packets as IEEE 802.15.4 broadcasts or having a know-all object for the NDP lookup [46].

Sending every packet as an IEEE 802.15.4 broadcast leads to unwanted packet duplication as every node that receives a packet, tries to forward it, which causes broadcast storms.

Developing a new NDP would not find any use case, as would not be found in any real deployment. Furthermore, this is out of scope for this thesis.

The 6LoWPAN-ND is as of writing this in a very early development stage, such that is currently nor really usable in ns-3.

Classical NDP gives unwanted and unnecessary overhead to finding the IEEE 802.15.4 short addresses, which are used by the Link-Layer. Furthermore, the NDP adds noise to the simulation, e.g. packet drops, unwanted delays, etc. On the other hand with NDP activated, it gives RPL the functionality to drop routes, due to destination unreachable messages which are issued.

Lastly it is possible to adapt the NDP lookup in ns-3 to return the corresponding MAC address for each IPv6 address by doing the stateless address autoconfiguration the other way around, as described in [47] chapter 6. To do this, the lookup function in the `ndisc-cache.cc` has to be changed to return an 48-bit pseudo address to prevent packet duplication as described by sending only IEEE 802.15.4 broadcasts. Also the `icmpv6-l4-protocol.cc` should be prevented to send messages to disable the periodic Neighbor Solicitations and Advertisements. This also disables the functionality to send Internet Control Message Protocol for the Internet Protocol Version 6 (ICMPv6) ping messages.

For this thesis, the classical NDP is preferred over the adapted version due to its broader applicability, ease of use, and better comparability with other examples.

5.2.4 Packet Traces

In ns-3, packets can be tracked via traces. This helps to analyse a great sum of packets. In this thesis, this feature is used to help with the evaluation. In particular, the `rpl-example-helper.cc` file includes all functions that interact with the traces. These functions write the traces in text-files, which can than be later statistically analyzed.

Furthermore, the UDP echo client and server were adapted to include a timestamp tag on each packet, to see when the packet was sent. This make calculating the round trip time of a packet easier.

5.2.5 Applications

The `UdpEcho` client and server, used to send the data in the network, were also adapted, by adding tags to the packets sent from these applications. Firstly, a timestamp tag was added to packets, with the timestamp added to the packet being the time of generation of the packet, to make tracing and latency calculation easier. Also a counter tag, which increments its number by one for each packet sent was added to the packets. This is done as workaround, as LR-WPAN can resend packets until an acknowledgment is received.

The LR-WPAN layer knows that these are the same packet, due to the sequence counter, and will only forward it once. The `UdpEcho` client and server does not and receives each resent packet as an individual packet. By using the counter tag, multiple received packets with the same counter number are just discarded and in case of the server is only echoed back once.

5.2.6 Ipv6RawSocket

A determential flaw in ns-3 at the time of creating the routing protocol was that Ipv6RawSockets did not add the multicast address they're bound to to the Ipv6Interface. This lead to packets with multicast addresses not being received by other nodes. [48]

5.2.7 Wrong ckecksums were not discarded

Due to creating RPL ICMPv6 packets, an own version of an ICMPv6-Header was created. By creating such packets it became imminent, that checksums in packets were randomly generated. Later it was revealed that, since the checksum generation was wrong, checksum generation had to be turned off. Furthermore, it became clear, that packets with wrong checksums were not discarded. [49]

5.3 RPL Modules

Since RPL can be a convoluted routing protocol, the most basic features of RPL are extracted in Figure 5.2. Furthermore, it is described what these modules do and how they are implemented in ns-3.

5.3.1 Topology

LLNs like radio networks, typically lack predefined topologies. Consequently, RPL must first discover links and scarcely select peers. RPL optimizes routes for traffic to or from one or more roots acting as sinks within the topology. This results in RPL structuring the topology as a DAG, partitioned into one or more DODAGs, each corresponding to a sink. In cases where the DAG has multiple roots, it's expected that these roots are interconnected via a common backbone, such as a transit link. [4]

A RPL node has a set of values, which uniquely classifies it. These are:

- A `RPLInstanceID`, which identifies a set of one or more DODAGs, allowing for multiple independent sets optimized for different OFs and applications within a network.

- A `DODAGID`, which in conjunction with the `RPLInstanceID` uniquely identifies a DODAG

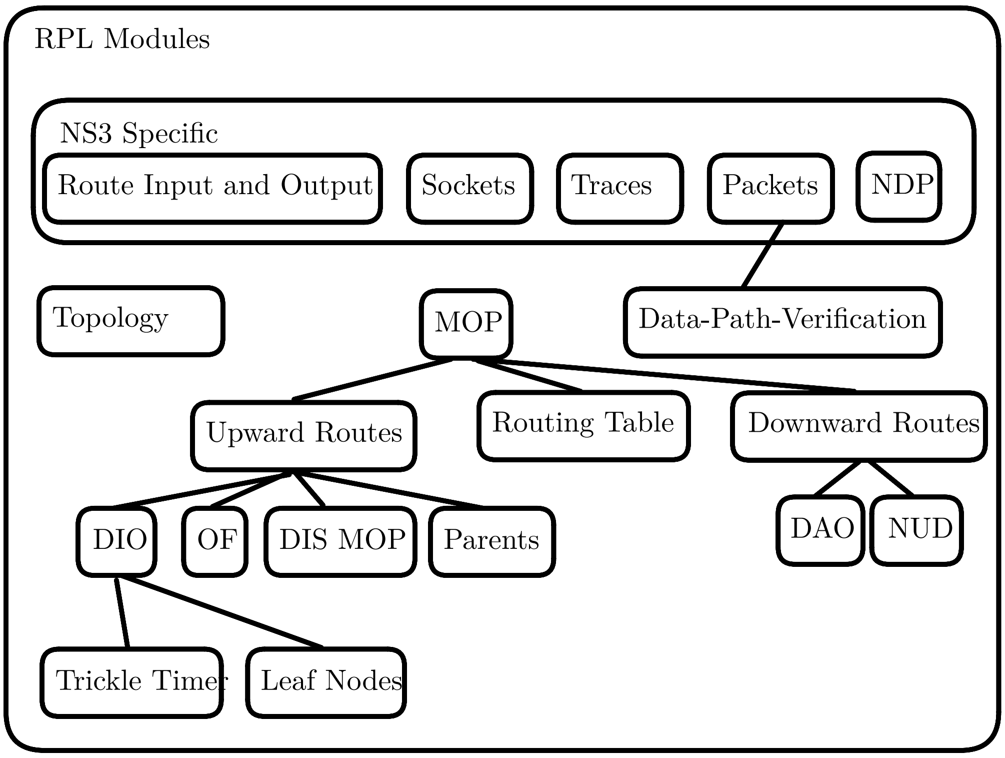


Figure 5.2: RPL Design in ns-3

- A `DODAGVersionNumber`, which indicates the DODAG version, is used when the DODAG is reconstructed from the root, leading to an increment in the version number.

- The rank, which is used to establish a partial order in the DODAG in respect to the root node.

Inside a DODAG, routes are formed, as described in Section 2.2, via DIO and DAO messages according to the given OF. The OF is how RPL nodes select their parent and child nodes.

The standard RPL [4] differentiates between local and global `RPLInstanceIDs`, where local DODAGs with a local `RPLInstanceID` can only have one root. In this implementation, local and global `RPLInstanceIDs` are treated equally and are left to the user to differentiate correctly when creating an ns-3 example, by creating more RPL instances and adding them to a node via list routing. Since the RPL is currently only tested in storing mode, no DODAG version increment is implemented. This means that there is also no global repair.

5.3.2 Mode of Operation

The mode of operation gives the basic working principle on how a DODAG operates. The mode of operation is propagated through a DODAG by the DODAG root through the MOP field in the DIO messages. The mode can be set to only maintain upward routes, be in non-storing mode, in storing mode with no multicast support or with multicast support. In non-storing mode, no node holds a routing table, except for the root node. Therefore, all nodes only know their parent set and send all traffic upward until it reaches the root node. For downward traffic RPL uses IP source routing. In storing mode, each node has its own sub-DAG stored, which it constructs by DIO and DAO messages. Therefore, when sending a packet, each node either knows where to send it or giving the packet upward to its preferred parent as default route. This implementation mostly only supports storing mode, but with a few changes to the routing table, it can easily be changed to also support non-storing mode.

Routing Table

Since each node has its own DAG in storing mode, it has to save it somewhere. This is where a routing table comes into play. When a node is added to a DODAG or it gets a new parent, the route to the parent is saved in the table. Also whenever a upward or downward route is discovered it is stored.

The routing table therefore consists of many entries, where each entry extends from the ns-3 `Ipv6RoutingTableEntry` class. In addition to the parameters given from `Ipv6RoutingTableEntry`, such as `destiantion adddress` and `next hop`, the parameters `dodagid`, `instanceid`, `metric`, `path sequence`, `lifetime` and `downward` are added. The IDs are given to clearly specify the DODAG. The `metric` parameter is currently unused as it would matter when a different OF then OF0 would be used and can be set to whatever metric the OF should use. Also the `path sequence` is currently unused, as the DAO parent set is equal to the preferred parent. The `lifetime` is used to set how long the route shall be used and is refreshed a life signal is received. Finally, the `downward` parameter, which is set as a boolean, is used to quickly check if a route is added as downward or upward route. This is used for the data path validation, described in Section 5.3.5, to check if a DAO loop has been formed.

5.3.3 Upward Routes

In RPL, upward routes refer to routes from leaf nodes (nodes at the bottom of the network hierarchy) towards the root of the DODAG, which typically is a sink or a border router. DIO messages are a type of RPL control messages which propagate routing information and establish upward routes within the network. DIOs are periodically transmitted by nodes in the network to announce their presence and advertise the network. DIO messages include information of the advertised DODAG, propagated from the root, and the sender rank. With this information, a node can join the DODAG and calculate its own rank from the parent rank.

Trickle Timer

In RPL, DIO messages are periodically sent with a trickle timer when a node joins a DODAG or is a root node. A trickle timer is a periodic timer that does not have a fixed delay after which it fires. Rather the interval between each timer trigger gets increased. In RPL, this has the advantage that when a node first joins a DODAG, it will send out DIO rather fast to onboard many nodes quickly and later on more spaced out DIOs to preserve energy and prevent excessive network overhead. This adaptive behavior helps strike a balance between responsiveness and efficiency. [50]

Leaf Nodes

A node can be configured to be a leaf node. A leaf node itself does not participate in the RPL routing itself. Therefore, a leaf node generally never sends DIO messages and only maintains its upward routes and should not have downward routes. This can be useful when a node is moving to prevent it from creating unnecessary child nodes which will be dead routes anyway when moving out of reach after onboarding.

Objective Function

An OF is how a RPL node selects its routes. The OF a node uses is defined in the Objective Code Point (OCP), given in the DIO configuration, which itself is received from the parent. Therefore, the root of a network has to be configured with a valid OCP. The OF defines how the rank inside a RPL DODAG is calculated, where the rank approximates the node's distance from a DODAG root. Therefore, the OF is also used to calculate the list parents of the node. The rank of a node is always a positive number and is calculated with the OF from the rank of the parent. The OCP selection of each node is implemented very rudimentary, as it will just use the first OCP in its set.

The OF0 is the default OF. OF0 is designed to facilitate interoperability across implementations in various use cases, as it does not specify link properties into a rank increase. [51] In this implementation, only OF0 is used which utilizes the hop count from each node to the root. Also to only select feasible parents for the OF, the Link Quality Indicator (LQI) of LR-WPAN is used to omit all DIO messages with a poor quality. This prevents to select a node as preferred parent which will not be reachable and only reached a random DIO message.

DIS Mode of Operation

In RPL, nodes, that are not configured to be root nodes, normally stay silent until they receive a DIO message of interest. Optionally, a node can be configured to proactively send periodically a number of DIS messages to probe for nearby DODAGs. A node that receives a DIS should responde with a DIO message to the DIS sender. If no DIO message is received after a configured number of DIS messages, the node itself can decide to become a root node and can start sending DIO messages.

Parent Design

In RPL, a node can have, when onboarded in a DODAG, a DODAG parent set. In this set a preferred parent is chosen to which the upward traffic is sent. This allows a node when its preferred parent becomes unreachable, to send DAO messages to another parent in its parent set. This has the benefit, that the node does not need to completely detach from the DODAG when the preferred parent is lost.

Furthermore, in RPL there is an option to have a set of DAO parents, which is a subset of the DODAG parent set. These parents can be seen as backup parents with an already established route. Also a node can set the bits in the `path control` field of DAO parents in order to show a preference among these parents. That preference can influence the decision of the DODAG root when selecting among the alternate parents/paths for constructing Downward routes [4]. This multipath feature (multiple DAO parents) is not supported in this implementation, but can be easily added as the logic is already handling the execution as if multiple parents would be possible. Only the `path control` should be added. This is because it adds too much unnecessary complexity and is mosty also missing in other implementations as well.

Candidate Neighbor Implementation

As stated in [4], the candidate neighbor set is a subset of the nodes that can be reached via link-local multicast. Furthermore, the selection of this set is implementation and OF dependent. In this implementation the candidate neighbor set is equal to the DODAG parent set, which itself are nodes reached via link-local multicast where it's rank is smaller than the rank of the node. This is done such that RPL does not have to store each neighbor and reduce the memory load of the protocol. The drawback to this is that whenever a node does not have nodes in the DODAG parent set, it has to detach from the DODAG and wait again for DIO messages.

5.3.4 Downward Routes

In RPL, downward routes refer to routes from the root of the DODAG towards leaf nodes or other nodes within the network hierarchy. DAO messages are a type of RPL control messages which are used by RPL to create and maintain downward routes within the network. DAOs are generated and sent upwards whenever a nodes adds or updates its children. This also means that DAO messages are sooner or later propagated upwards until they reach the root node. Also, DAOs have the option to be hold back for short period of time, to be bundled together if more DAOs shall be sent in the same timeframe, to reduce the message load on the network. Furthermore, a DODAG can be configured, through the root node, to expect an acknowledgment for each sent DAO, to ensure that the DAOs have been well-received from their DAO parents. In RPL, as stated in Section 5.3.2, there is a difference between non-storing and storing mode. In both cases, packets travel up and then down the DODAG. In non-storing mode, the packets will go all the ways up to the root node, whereas in storing mode the packets are only given upwards to a common ancestor of both the source and destination node.

In addition to the preferred parent, a RPL node can have multiple DAO parents. With the RPL path control feature, a node can manage multiple downward routes, which allows the node to distribute traffic across multiple paths, enhancing network reliability. In this implementation, the DAO parent set is restricted to the preferred parent, which leads to the path control being unused.

DAO Resend

Whenever a DAO is sent and an acknowledgment for the DAO is expected, a resend is immediately scheduled with a specified DAG-DAO-ACK-timeout. If in the meantime until the resend triggers, a DAO-ACK is received with the sent DAO sequence number, the resend is canceled. If no DAO-ACK is received until the resend is triggered, the DAO is resent and like before rescheduled again. The DAO is resent up to the number of DAO retries. If by then still no DAO-ACK is received, the node sends a no-path to the parent that did not send the DAO-ACK back and removes it as a parent.

Neighbor Unreachable Detection (NUD)

As stated in [4], no specified route adjacency maintenance for neither upward and downward paths are specified. As in other routing protocols, as in OSPF or IS-IS, keep-alive messages are sent to ensure the routes are up to date. In LLNs, such a proactive approach is often not desirable and could lead to excessive control traffic in light of the data traffic with a negative impact on both link loads and nodes resources. Therefore, RPL needs an external mechanism, that a neighbor is no longer reachable. The RFC 6550 suggests that reactive methods shall be used such as the Neighbor Unreachability Detection of RFC 4861 or Layer 2 acknowledgments. In ns-3, since the Neighbor Unreachability Detection is already implemented, it shall be used to determine which routes are not available anymore.

5.3.5 Data-Path-Validation

RPL uses on-demand loop detection with data packets in LLNs to conserve energy. LLNs often experience changes in physical connectivity that aren't critical for routing until data needs to be sent, saving energy by avoiding constant updates to the routing topology. Routing loops, also called DAO loops, may be formed when a parent has a route to a child by receiving and processing a DAO from it. When the parent now drops out of the network and its former child is not a leaf node, it might happen when the former parent now rejoins the network to receive a DIO from its former child. The node then may join the network by setting its former child as a parent. Then, a loop has formed. This would normally be prevented by a no-path message that would be sent from the parent node to the child upon leaving the network, but it is in the nature of LLNs that messages might disappear. This is where the data-path-validation comes in. Each data packet in RPL includes the transmitter's rank into an IPv6 hop-by-hop header, enabling nodes to detect potential loops. If a packet's routing direction conflicts with the rank relationship between sender and receiver, indicating a possible loop, the receiving node initiates a local repair operation.

In ns-3, it is not possible for a routing protocol to add such a header to an IPv6 packet. While it is possible to add an header to an IPv6 packet, the problem here is that it is not doable to change the header on each hop due to how `ipv6-l3-protocol` works in ns-3. The rank and the direction the packet is going need to be updated on each hop.

The workaround in ns-3 is the usage of tags. By creating a custom tag with all the information needed, it can be easily added to a packet, as well as updated on each hop[1].

[1]This workaround has been made possible by the contribution from Tommaso Pecorella (University of Florence)

CHAPTER 6

Evaluation and Discussion

All evaluations are done over 10 runs, to achieve a higher confidence in the results. If not stated otherwise, the default configuration, shown in Table 4.1, is used.

The basic procedure for a node to be onboarded to a RPL instance is the following. A node in the RPL network (at the beginning only the root node(s)) sends DIO messages periodically. Upon receiving a DIO, a node, not already joined a RPL instance, will immediately issue a unicast DIS to the sender. The DIO sender will then send a unicast DIO with a DODAG configuration option. The node will then join the RPL network as a child of the DIO sender. After a specified delay the node will then send a DAO message to its new parent with routing information to it. The node has then successfully joined the network.

The examples are set up, that the node at one or multiple nodes are configured to be a client, which send UDP packets to a node configured to be a server, which will echo the packet back to the client(s). The clients will send the UDP packets after 100 seconds, to give the RPL network time to be set up.

In ns-3, wireless devices have a good connection range up to about 100 meters. After that the connectivity falls off drastically to about 120 meters. Beyond that, nodes cannot really reach other nodes, due to signal strength.

As stated in [44], for random variables to change in a statistically uncompromising way, the seed in all examples is set to 1 and the run number is advanced in each run. This makes sure that the Random Number Generator (RNG) is different for each run, but is the same when other parameters are changed. In addition to RPL, to mirror the real world more accurately, 6LoWPAN and LR-WPAN is used in all examples. Since in ns-3 only the ad-hoc mode for LR-WPAN is implemented, this mode is used. Also each node gets assigned the IPv6 stack with ”2001:2::” as base address.

6.1 Conformity check with the RFC

To check conformity with the RFC [4], smoke tests are deployed.

6.1.1 Onboarding and Message Passing

As mentioned in Section 4.1, a small basic line example is the easiest to test RPL in ns-3. In this example, one can see how each node is added to the RPL instance. The root node is chosen to be at one end of the line. As each node has at most 2 neighbors, each node (except for the root and the leaf node) has exactly one parent and one child node.

By stepping through the line example with 3 nodes, the onboarding process can be observed by checking the PCAP file. This is shown in Figure 6.1. It can be seen that in the beginning node 0 broadcasts DIO to the all-RPL-nodes multicast address periodically with the trickle timer, as node 0 is configured to be the root. As soon as node 1 receives the DIO it sends node 0 a unicast DIS message. Upon reception, node 0 sends a unicast DIO to node 1 with the DODAG configuration. Node 1 then immediately also begins to send periodic DIO messages to the all-RPL-nodes multicast address. It does not matter that node 0 has not yet received a DAO message from node 1, as node 1 has already all necessary information. The same then happens between node 1 and node 2. After the specified DAO delay, node 1 sends a DAO to node 0 with routing information to itself. Node 2 then also sends after the specified DAO delay a DAO to node 1 with its routing information. Node 1 then waits again the DAO delay and forwards the route it got from node 2 to node 1 with a DAO. Hypothetically speaking, if node 1 would have received the DAO from node 2 before sending its DAO to node 0, node 1 would have bundled both DAO messages together and only one DAO message to node 0 would have been sent.

After this exchange, node 0 now knows the whole DAG of the network. After 100 seconds, when the client (node 0) then decides to send UDP packets to node 2, it knows the IP address of node 2, as well as the next hop node 1 from the received DAO messages. Node 1 also knows to forwards the packet to node 2 from the DAO that node 2 sent. Upon echoing the packet back, node 2 and node 1 will not know where to send the packet and just hand it upward to its parent, which they know from the DIO message, when the packet then finally is received back at node 0.

6.1.2 Root Node Failure

In a typical RPL network, multiple root nodes are often deployed to enhance network reliability and resilience. However, if the network has only one root node and it fails, the behavior can vary. In some cases, the failure of the only root node can lead to network partitioning. The network may split into smaller segments or become disconnected, causing communication disruptions between nodes. The partitioned segments may operate independently until a new root node is manually introduced or an automated recovery mechanism is triggered. In other cases, RPL may not include built-in mechanisms for automatic recovery from the failure of the only root node. In this implementation,

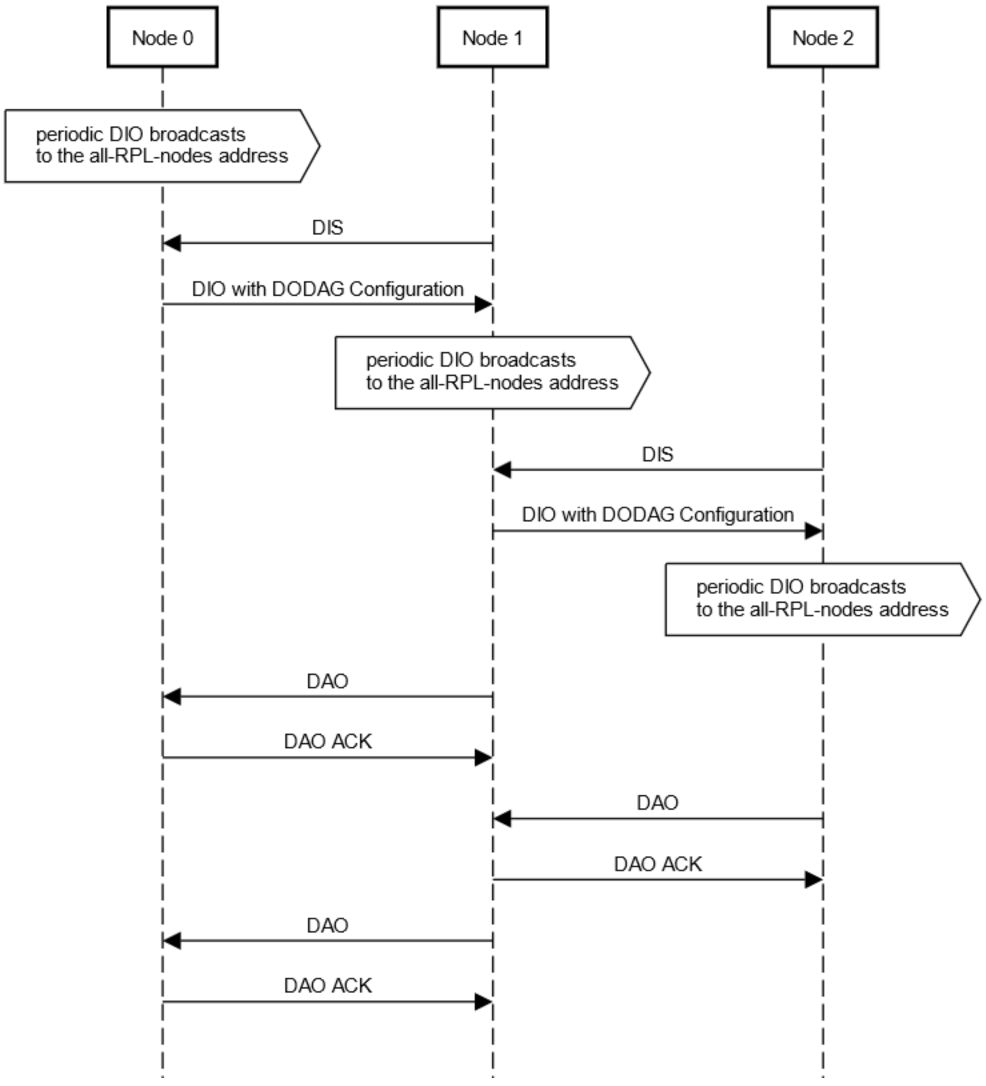

Node 0
Node 1
Node 2
periodic DIO broadcasts to the all-RPL-nodes address
DIS
DIO with DODAG Configuration
periodic DIO broadcasts to the all-RPL-nodes address
DIS
DIO with DODAG Configuration
periodic DIO broadcasts to the all-RPL-nodes address
DAO
DAO ACK
DAO
DAO ACK
DAO
DAO ACK

Figure 6.1: Basic Line Example with 3 Nodes

the DIS Mode of Operation is slightly changed, such that a node configured with it, also sends DIS messages (and becomes root when it does not find another one), when it is detached from a RPL network and not only on boot-up.

In the line example, in Section 4.1, node 0 can be set up to fail after 10 seconds and node 1 to be configured with the DIS Mode of Operation. When node 0 fails the RPL network will normally resume until either node 1 deems node 0 as unreachable or after the maximum tries of DAO retries still no DAO-ACK is received (in this case after 60 s). After this, node 1 will delete node 0 as its parent and now broadcasts DIS messages again. Since no other root is configured in this network, node 1 will itself become root and add all other nodes in its network. After this, the network is once again operational with node 1 as root.

6.2 Onboarding

In RPL, the path generation of the routing protocol can be measured by looking in the routing table of the root node. When all other nodes are added to the routing table of the root, every node can be reached in the network. How the paths are chosen, i.e. which parents are selected, in this network is objective function dependent, in this case OF0.

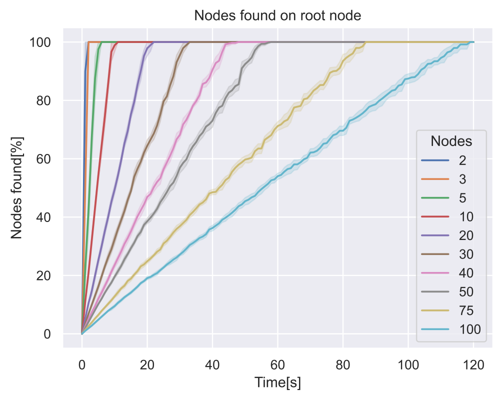
In the examples, a network setup time of 100 seconds is given, where no data packets are sent, as a margin, to ensure that the DODAG is already formed. Also this ensures that the trickle timers are at a high rate, and the advertisement rate is about the same for each node.

To get a grasp on how well RPL does onboarding, it is shown in Figure 6.2 how the root nodes learns new nodes over time. The number of newly added nodes to the roots routing table is plotted over the time in seconds. The data for all plots was resampled to 1 second precision and shown with errorbars over 10 different runs.

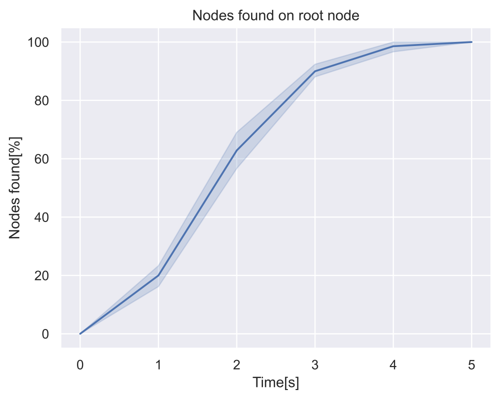
In all examples it can be seen that the DAO delay has a dominant role. This is especially noticeable in Figure 6.2b. On average, every hop a node is farther away from the root node, it adds the DAO delay, which in this case is 1 second. Also due to the circumstance that DAO messages can be bundled together, the root node can register multiple routes at once, which leads to the vertical lines in Figure 6.2a.

By varying the number of nodes in the line example, i.e. increasing the hops to the root node and the leaf node, it can be seen that it takes increasingly longer for the root node to know the whole DODAG, due to the DAO delay. This effect over multiple number of nodes can be seen in Figure 6.2a. As expected, the time it takes for the last node to be added to the routing table in the root is pretty much linear.

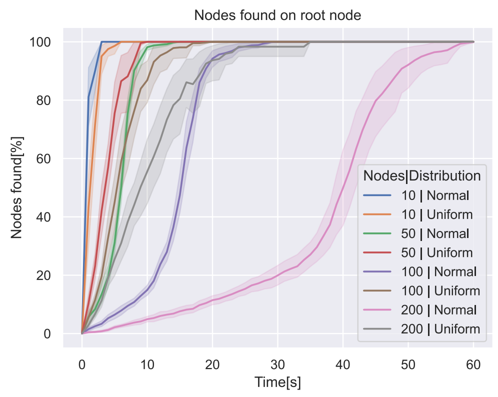
In case of the many nodes example, we have to differentiate between the two distributions and the number of nodes, as shown in Figure 6.2c. It can be seen that having fewer nodes, the nodes are quicker found, as expected, but the shape of the line keeps the same. The main differences here come from the different distributions. The uniform distribution resembles an logarithmic function. This can be explained that the root node has already many nodes in its immediate vicinity, but has to select few nodes that are much farther away. In case of the normal distribution, the root node is not fixed in the center, but is also placed randomly. Therefore, having the root node a bit farther away from the center of the normal distribution makes an noticeable difference. The root node slowly discovers more nodes until nodes in the center of the normal distribution are reached which they in turn find then the most nodes. Also having many nodes in the center causes collisions of packets, which delays the onboarding. This explains the low nodes found percentage at the beginning and the rather steep increase at a certain point. If the root node would be placed dead in the center of the normal distribution, the line would be more similar to Figure 6.2a. Furthermore, as seen in Figure 6.2c by the errorbars, not in every run



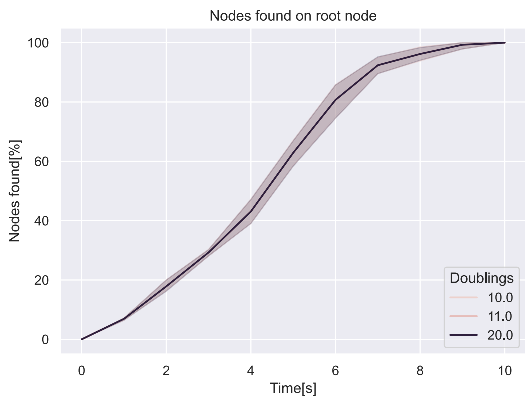
(a) Line Example



(b) Tree Example



(c) Many Nodes Example



(d) Tree with Strict Subtrees Example

Figure 6.2: Network Setup Time

the root node can find all other nodes in the network, as through the random placement, it is not ensured that nodes are even reachable from anywhere.

Also it can be seen in Figure 6.2d, that in all different doublings configurations the lines pretty much overlap. This means that the trickle parameters do not matter shortly after booting the device, which is expected as the frequency of sent DIOs will only slow down over time.

In conclusion, the more hops nodes are away from the root node, the longer it takes RPL to build the whole DODAG. This corresponds to the conclusion that was drawn in [40]. Obviously the hop count is heavily influenced by the OF. Furthermore the trickle timer parameters effects only nodes that are already a longer time in a DODAG.

6.3 Protocol Flexibility

6.3.1 Broken Link

To test how efficiently RPL recovers from a lost link, the example described in Section 4.3 can be used.

In this example, node 4 tries to send UDP packages to the root node 0 every second after the 100 seconds network setup time. The root node then will echo the packets back. As OF0 is used, the preferred parent is the one with the least hops to the root node. Therefore, node 4 will send the packets over node 1 to node 0. Node 1 will operate as normal for 130 seconds and then turn off its interface, to simulate a battery outage on the device. Therefore, node 4 will eventually receive a neighbor unreachable destination message, since NDP is used. As recommended in [4], RPL will now remove node 1 as parent and tries to discover a new parent by listening for a DIO message. Node 4 will then discover node 3 as a parent and build a path to node 0 over node 3 and 2 and send DAO messages. The root will update the new path in the routing table due to a newer path sequence number.

After 300 seconds, node 4 will turn on its interface again, with the same IP address as before. The node will listen again for DIO messages and eventually taking node 0 as parent. Node 4 will then recognize the shorter path over node 1 and use node 1 again as preferred parent.

It shall be kept in mind, that the current limitation that each node can have only one DODAG parent, as stated in Section 5.3.3, makes it that the alternative path has to be found upon removing the routes over node 1. If more than one DODAG parent would be possible, the the alternative path would instantly be used, when the routes over node 1 would be removed. This is because node 3 would already be a secondary DODAG parent.

Using node 0 as sender does not work when node 1 is disabled, since RPL responds reactive to traffic. This is because node 0 will receive a neighbor unreachable destination message and remove node 1 from its routing table, but node 4 will never set node 2 as a parent as long as it not sends packets upward and will therefore never send a DAO to the root. In case mulitpath is enabled, this would also work, due to having multiple DAO parents, which makes the RPL routing protocol much more robust. Also if a global repair would be set to trigger periodically, this issue would also resolve itself eventually.

In Figure 6.3 the received UDP packets of the alternate path example are shown over time. The first 100 seconds nothing happens, as described in Section 6.2, as the client waits for the network to be established. Afterwards, the network operates normally for another 30 seconds. Then node 1 is configured to fail and no more packets are received back on node 4. The time it takes node 4 from node 1 failing to finding node 3 as new parent is given in Table 6.1. On second 300, node 1 starts up again. In Table 6.1, it is also shown the time node 4 takes, after node 1 is running again until it selects it as a parent. When node 1 joins the DODAG it has a fresh trickle timer and sends its DIO messages in very short intervals. The reason it takes so long for node 4 to select node 1

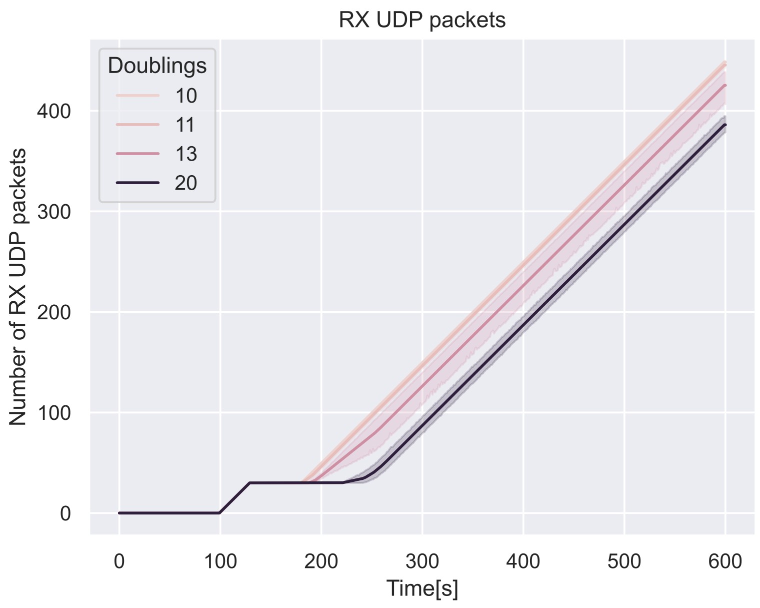


Figure 6.3: UDP Packets Received Alternate Path Example

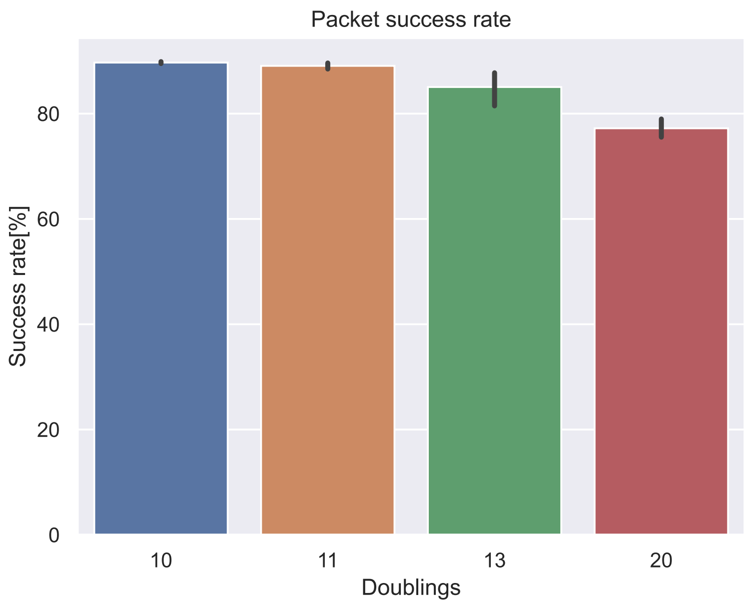


Figure 6.4: Success Rate Alternate Path Example

as parent, is that node 1 takes long to reconnect to the RPL network due to the slow interval of DIO messages from node 0. It can be seen, that with an increased doublings

Doublings	**Time[s] until path over node 3 is found**	**Time[s] until path over node 1 is found**
10	47.468	8.243
11	46.93	5.746
13	56.46	21.732
20	111.252	142.977

Table 6.1: Times when alternative and better paths are found

parameter of the trickle timer, it takes longer to find a new parent, as expected. An inconsistency arises in Table 6.1, since the times of doublings 11 are shorter than those with doublings 10. This happens because trickle timer do not increase their intervals at random, but deterministic. Since node 1 is always turned off and on at the same time in each run, it just happens that the DIOs are also received earlier in each run.

Furthermore, Figure 6.3 shows that no more packets are received back on the sender after 130 seconds, which is expected, as node 1 turns off at that time. Then the reception of packet continues when the alternate path over node 3 is found for each configured doublings parameter. It shall also be noted, that there is no change in packet reception when the better path over node 1 is found again after 300 seconds, as the packets are just sent over another path, which does not hamper the success rate.

This is further showcased inFigure 6.4, by displaying the median success rate of the runs for each doublings value. The success rate can never reach 100% as after 130 seconds the link is lost. Due to the quicker recovery with lower doublings values, the success rate is also higher.

6.3.2 Moving Node

To see how well RPL performs when moving a test is conducted where a single moving node traverses through a grid of nodes, as described in Section 4.5.

Speed	**Doublings**	**RTT Latency[s]**
2	10	0.062
2	20	0.045
3	10	0.06
3	20	0.032

Table 6.2: Latency of the Moving Node Example

The behavior which parents of the moving node are chosen with OF0. To give a rough overview on which nodes are considered and chosen as parent by the moving node, the chosen preferred parent over time is shown over multiple runs and in different speed and

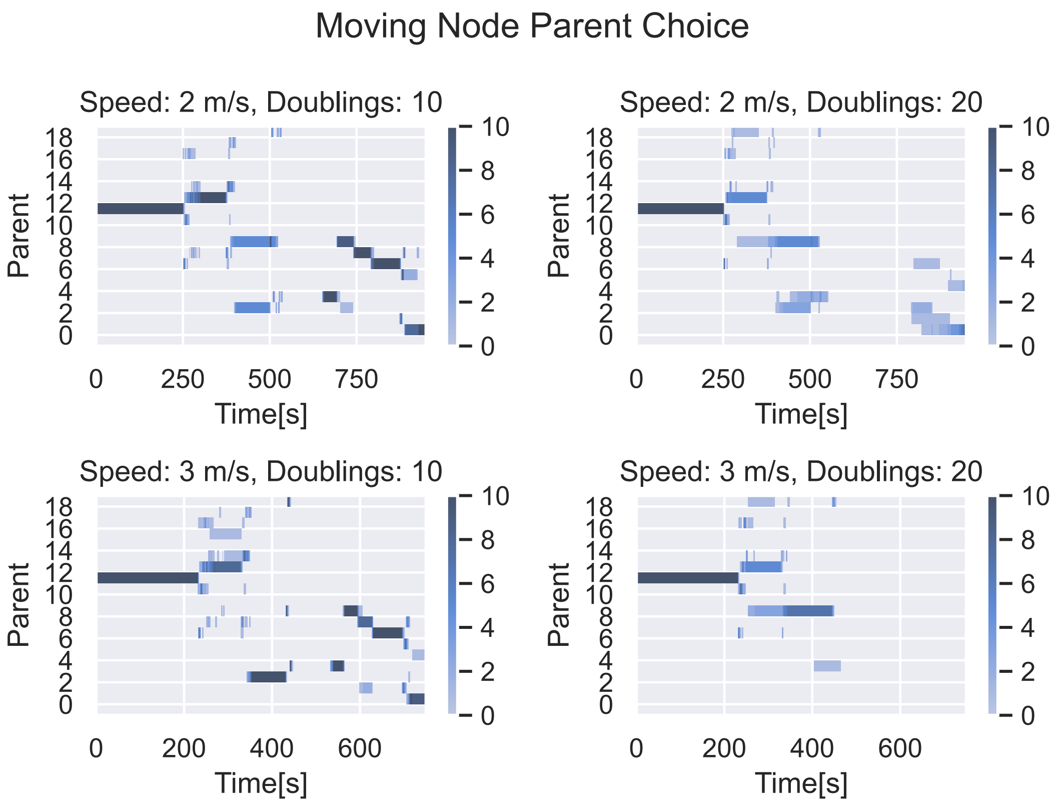


Figure 6.5: Moving Nodes Parent Choice

doublings configurations in Figure 6.5. The color of the plot indicates the confidence this node was picked as a parent, i.e. multiple runs with the same parent chosen.

It shall be noted, that the simulations of runs with 2 m/s are 950 seconds and with 3 m/s are 750 seconds long, since the simulation is stopped 50 seconds after the moving node 25 has reached its final destination.

In Figure 6.5, it can be seen that node 12 is always chosen as a parent node in the beginning, as it is the root node and the moving node starts right beside it. Also a gap with no preferred parent can be seen in all configurations, as the moving node runs 200 meters off from the grid, before returning to it again. Furthermore, nodes can become the parent of the moving node multiple times per run, as when is loses connection to it and later reconnects to it.

In this RPL implementation, as stated in Section 5.3.3, a node only adds other nodes with a lower rank than itself to its DODAG parent set. This means in this case, that normally, with OF0, node 25, which has the root node as preferred parent, node 25 would not add any other nodes to its DODAG parent set. The reason it still adds other nodes to its DODAG parent set is because it is configured to be a leaf node, which always

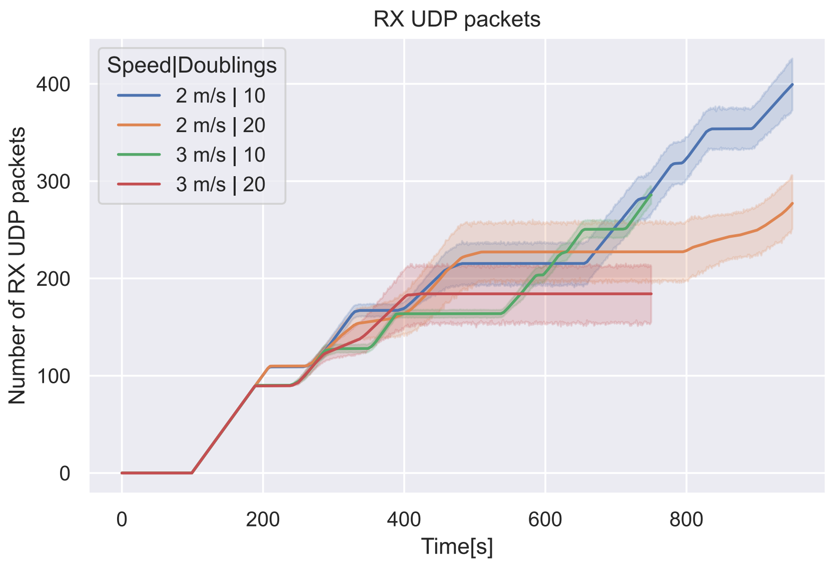


Figure 6.6: UDP Packets Received Moving Node Example

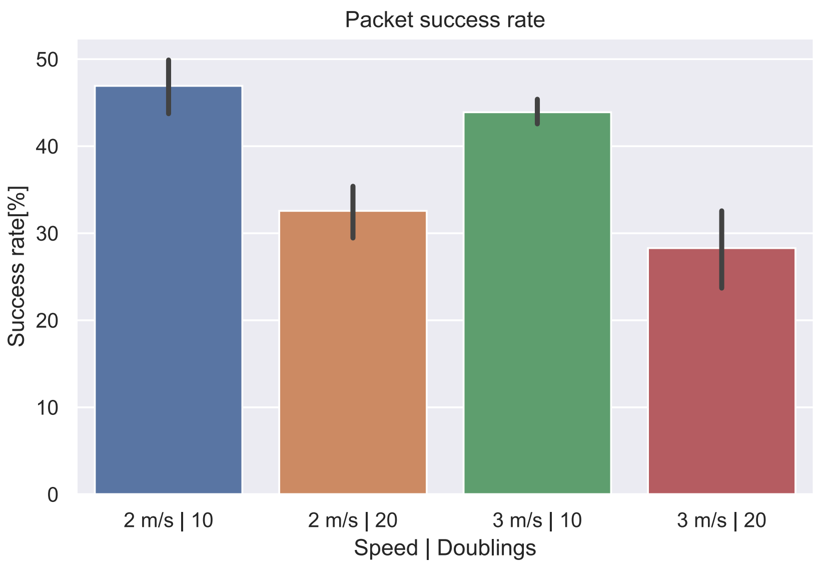


Figure 6.7: Success Rate Moving Node Example

has infinite rank. This enables the node to instantly change the preferred parent when its preferred parent is not reachable anymore. This has the advantage, that the node does not have to detach from the DODAG completely. A node knows if another node is unreachable via the neighbor unreachable detection, as stated in Section 5.3.4.

Also in Figure 6.5, it can be seen that whenever a node switches its preferred parent, it can happen that the node switches through multiple parents in quick succession. Whenever a node removes its preferred parent, it looks in its DODAG parent set for the next best candidate to be the new preferred parent via its OF. Since the node is moving, it can happen, that this one is also not reachable anymore and will also remove it and switch again. Furthermore, by moving into a new area, the node can then receive new DIO messages it has not seen before and can also select them to be the new preferred parent. This quick fluctuation of parents is different for each run and looks in the Figure 6.5 like noise.

While comparing the different speed and doubling configurations in Figure 6.5, some conclusions can be drawn. The most obvious is that having a lower speed and a lower doublings value, i.e. a faster DIO interval, leads to a more robust and up-to-date network. Also, in this grid structure RPL handles both, 2 and 3 m/s relatively well, which is above average walking speed [52]. Furthermore, the doublings parameter has a major impact, best seen with 3 m/s, as after detaching from the DODAG, the moving node does not reconnect to the RPL network in all 10 runs anymore, as the simulation ends before that.

Figure 6.6 depicts when exactly packets are echoes back to the moving node. After the initial 100 seconds, where time is given for the network to construct the DODAG, packets are received back on the moving node until the connection to the root node is lost. The moving node will then try to find the next preferred parent which is reachable. Here it also becomes obvious that a lower doublings value makes the RPL network much quicker to respond to changes. Also, the variance over multiple runs can be seen with the errorbars, as time goes on, as different preferred parents are selected.

The summary of the received packets can be clearer seen in Figure 6.7. This showcases how well a moving RPL node performs, since the success rate of the UDP packets should give a good estimate. This also shows, that in this example the doublings value has a much greater impact than the speed, for the success of received back packets.

Finally, the previous findings can be cross checked with the latency, given in Table 6.2. The configurations with the lowest latency are those with a doublings value of 20, since only packets which actually made it back to the client are used for latency calculation. Therefore, since the moving node starts near the server node the latency is in the beginning lower and gets greater over time when the moving node moves away from the server node.

In the moving node example, the case mentioned in [4] Section 3.7.2 can happen. At some point the moving node detaches from the DODAG due to moving out of the parents range. Upon detaching from the DODAG, the moving node sends a poison to all nodes in range, to notify them, that the node has detached. This poison can get lost in the LLN, such that not all children of the moving node may receive it. Then the moving node may later take a former child, as a new parent. Upon sending a DAO message to the former child, while not having received before mentioned poison message, the former child not detects a loop. In the RFC the solution to this problem is, by relying on an mechanism described in [53], that adds RPL information in the IPv6 Hop-by-Hop option header.

6.4 Scalability

In IoT networks, scalability refers to the ability of the network to efficiently handle increasing numbers of devices and data traffic while maintaining performance. Therefore, the RPL implementation is check for success rate and latency in networks with varying node counts and different application setups.

6.4.1 Success Rate

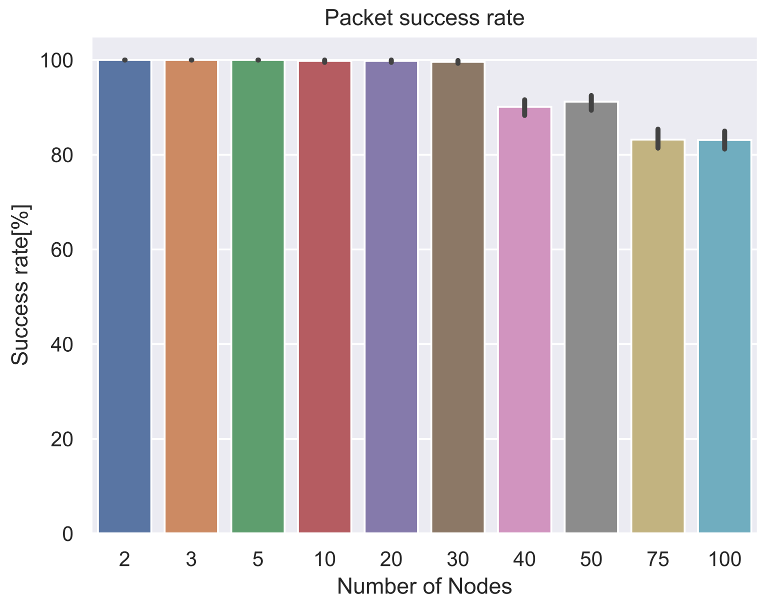


Figure 6.8: Success Rate Line Example

The success rate is expected to be relatively constant over multiple runs, as the topology in this example is constant. This is best seen in the line example as showcased in Figure 6.8. The success rate is for the most part 100% and gets worse the more nodes are in the network, as messages can get lost in LLNs, which is more likely the further packets travel. The reason the cases of 40 and 50, as well as 75 and 100, are so close in success rate is because simulation RNG does only change between different runs and not other between parameters. For example, run 1 and 2 with with 40 nodes use different RNG, but run 1 with 40 and 50 nodes use the same RNG, but have a different node count.

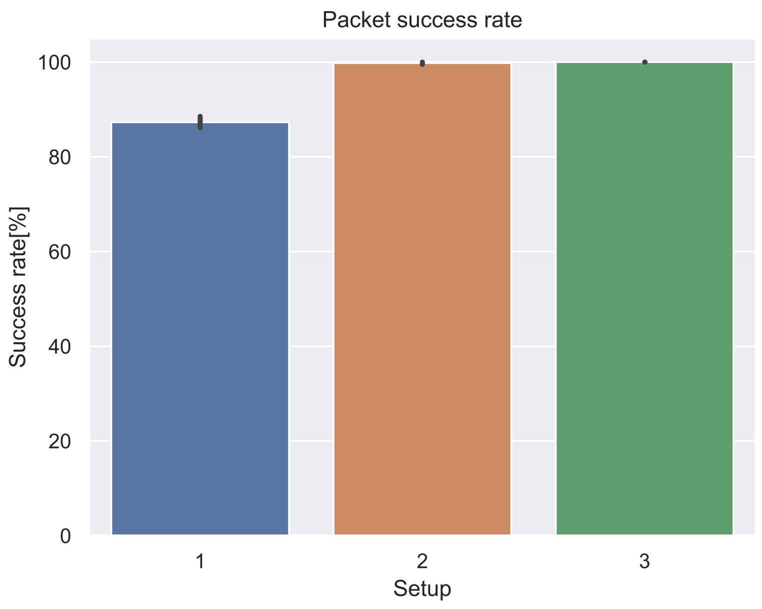


Figure 6.9: Success Rate Tree Example

In the tree example, as described in Section 4.2, the success rate is at 100%, except for setup 1, where the success rate can be seen to be around 90% as seen in Figure 6.9. This is because setup 1 does have multiple clients that send at the same time. This seems to be an issue of LR-WPAN collisions, due to the nature of Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA). In CSMA/CA the Hidden-Station-Problem can happen [54]. The Hidden-Station-Problem is when two nodes A and B want to send to a third node C, shown in Figure 6.10. Assuming A and B are not in the range of each other but both are in range to C. Further assuming A begins sending to C. If now B will also want to send to C, it will check the medium, to see if it is free. B will not see the traffic of A, as it is out of range. Therefore B proceeds to also send to C, causing a data collision at the destination C. As this is a simulator, which runs of a global clock, the data packets are nearly sent at the exact same time. Furthermore in the tree example, the nodes 1 and 2, 3 and 4, as well as 5 and 6 each send packets to the same parent and they all have the same distance to the their respective parent node. This all makes a data collision much more likely.

To see how well RPL handles a large network, an example, as described in Section 4.4, with up to 200 nodes is chosen. As stated in Section 4.4, the nodes are uniformly distributed in setup 1 to 3 and normal distributed in setup 4 to 6. Also in setup 1 and 4 there is one client, in setup 2 and 5 there are two clients and in setup 3 and 6 there are 4 clients. As in the tree example in Figure 6.9, Figure 6.11 shows a similar behavior, since having more clients that send at the same time, drops the success rate significantly. Having only a single client that sends, leads to a near 100% success rate in

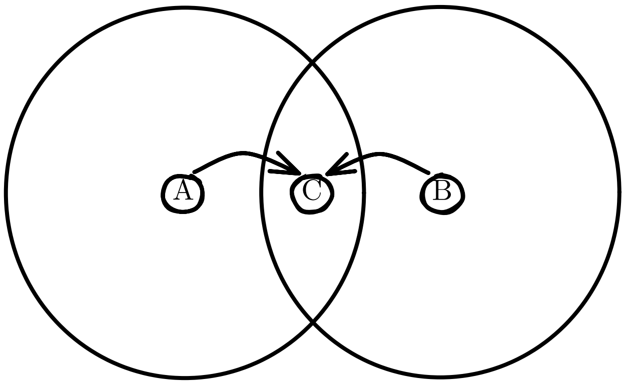


Figure 6.10: Hidden Station Problem

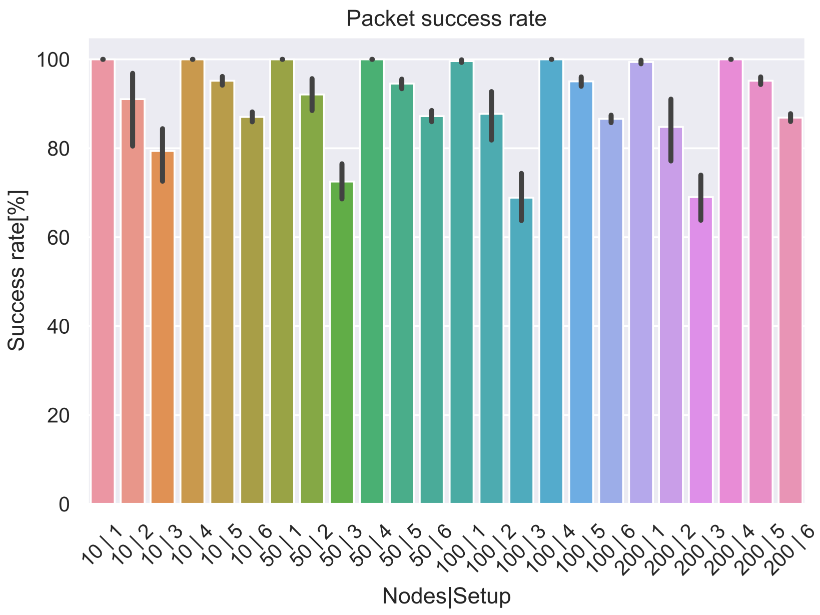


Figure 6.11: Success Rate Many Nodes Example

all distributions. Furthermore, the uniform distribution has a higher variance in success are the normal distributed nodes, as in the normal distribution most nodes are positioned close together.

6.4.2 Latency

The latency is expected to be relatively constant over multiple runs, as the topology in this example is constant.

In Table 6.3, the latency of the UDP packets of the line example, as described in

Number of Nodes	RTT Latency[s]
2	0.006
3	0.034
50	0.09
10	0.23
20	0.509
30	0.79
40	1.137
50	1.417
75	2.179
100	2.876

Table 6.3: Latency of the Line Example

Section 4.1, over the number of nodes is shown. With increasing node count, the latency also increases linear.

Setup	RTT Latency[s]
1	0.095
2	0.118
3	0.062

Table 6.4: Latency of the Tree Example

In Table 6.4, the latency of the UDP packets of the tree example, as described in Section 4.1, over the different setups is shown. The latency of the setup 2 is the longest, as the packets traverse from one leaf of the tree to another one. Also the latency of the setup 1 is longer than in setup 3. This is because with more than 1 sender, collisions happen more often, which leads to LR-WPAN to resend packets. This in turn makes the round-trip-time longer.

In Table 6.5, the latency of the many nodes example with different number of nodes and over the different client setups is stated. As stated in Section 4.4, the nodes are uniformly distributed in setup 1 to 3 and normal distributed in setup 4 to 6. Also in setup 1 and 4 there is one client, in setup 2 and 5 there are two clients and in setup 3 and 6 there are 4 clients. In setups with more clients, the latency is larger than with less clients, as more nodes send at the same time and due to collisions and LR-WPAN resends packets arrive at a later time. By comparing the different distributions, it can be seen that the latency in the normal distribution is ís always very close to each other for each independent setup as the variance of the distribution chosen is very low. Therefore, the most nodes are very close to each other, which in turn means the latency is also comparable low and is not really effected by the number of nodes. In contrast, the latency where the nodes are

Number of Nodes	Setup	RTT Latency[s]
10	1	0.034
10	2	0.053
10	3	0.104
10	4	0.034
10	5	0.063
10	6	0.11
50	1	0.084
50	2	0.131
50	3	0.197
50	4	0.034
50	5	0.063
50	6	0.109
100	1	0.15
100	2	0.168
100	3	0.232
100	4	0.034
100	5	0.063
100	6	0.109
200	1	0.193
200	2	0.236
200	3	0.314
200	4	0.034
200	5	0.063
200	6	0.11

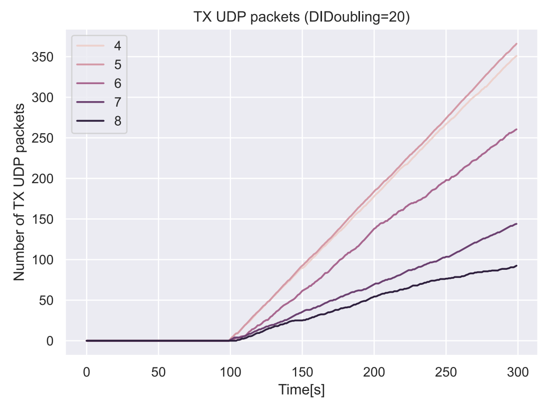
Table 6.5: Latency of the Many Nodes Example

uniformly distributed increases over the number of nodes.

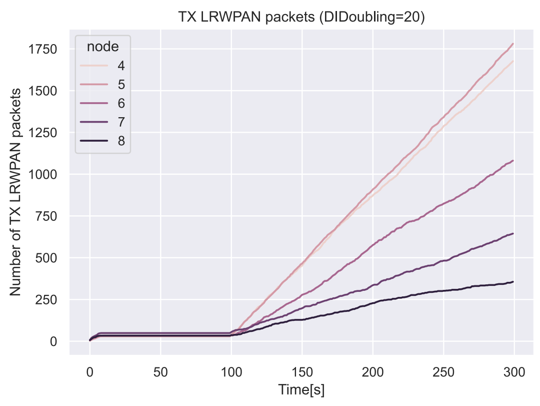
In summary having multiple nodes sending at the same time causes collisions and resends, which then increase the end-to-end latency. Other than that a higher number of nodes in most cases means an increased hop count, which also increases the latency, as packets have to be forwarded more often. In the end latency of the round trip time is below 1 second in most cases and only takes longer after 30 hops round trip.

6.5 Energy Consumption

Normally to test a network for energy consumption an energy model can be used. For LR-WPAN, there is no directly built in energy model in ns-3, but even if there was, using an energy model the energy estimate would still be questionably at least. Also the



(a) UDP Packets



(b) LRWPAN Packets

Figure 6.12: Sent Packets of Nodes with a Hop Count Distance to the Root Node of 2 Tree with Subtrees Example

current ns-3 MAC implementation does not use low power features. This means, that the radio is always on in the ns-3 LR-WPAN implementation. Therefore, the problem is that LR-WPAN always uses energy [43]. Accordingly, it was chosen to abstract the energy consumption to simply count the packets that are sent and received on a node.

The energy test are then conducted on the tree with subtrees example, described in Section 4.6. This example was chosen to mirror the example in [10] Section 7.3, to compare them. In [10] it is stated that node 4 and 5 are drained the fastest when using a hop count metric, which should be similar to OF0.

The first issue that arose when evaluating this example is, as there are so many clients that send at the same time, the success rate of this example is slightly above 40%, due to so many collisions happening.

As stated above, the energy consumption is approximated over the number of packets. When looking at the number of sent UDP packets, as shown in Figure 6.12a and the number of all sent LR-WPAN packets in Figure 6.12b, on all nodes with a hop count distance to the root node of 2 it can be seen that node 4 and 5 increase the fastest, as both have their own subtree which they need to forwards the packets to the server node 0 and their respective leaves.

This changes when adding the LR-WPAN packets that are received on these nodes to the sent packets, as seen in Figure 6.13. As these nodes are more in the middle of the network they receive more packets, even though they not necessarily are targeted to them, they still have to receive and check them. By using this as a metric, node 6 to 8 drain the fastest when using OF0.

In Table 6.6, the median number of packets sent of all nodes with a hop count distance to the root node of 2 of the energy example over the different doublings values. It can be

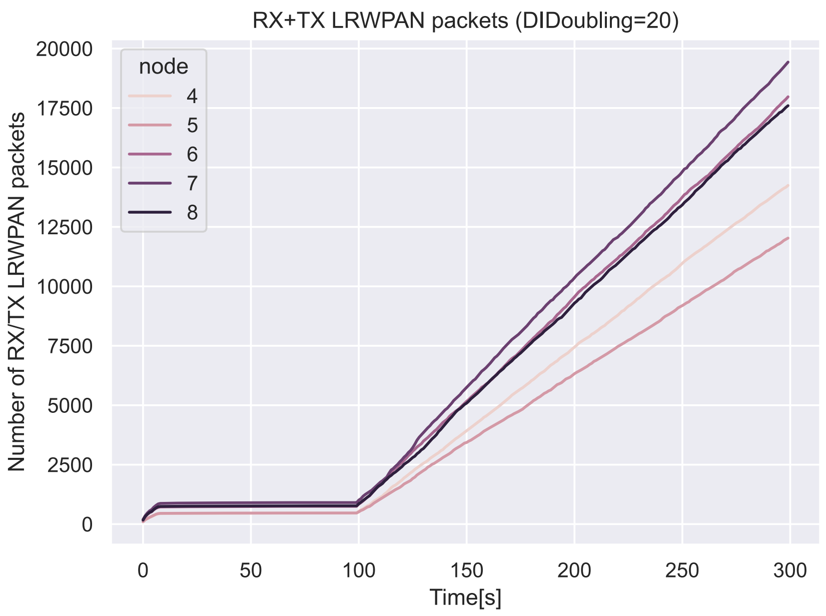


Figure 6.13: Sent and Received Packets of Nodes with a Hop Count Distance to the Root Node of 2 Tree with Subtrees Example

Doublings	Packets
10	1594
11	1593
20	1577

Table 6.6: Median Number of LRWPAN Packets Sent of a Node with a Hop Count Distance to the Root Node of 2 in the Energy Example

seen that there is only 1% difference in sent packets of a node between a doublings value 10 and 20. Therefore, if a network is not too constrained in resources a lower doublings value is the preferred choice.

Finally, for the sake of completeness, the median latency of this example is 0.375 seconds.

6.6 Simulation Time

To be useful simulations have to be done in a timely manner, as the most accurate simulation does nothing if it can never be evaluated. The many nodes example should display this well, since the number of nodes goes up to 200 and the nodes are dispersed with 2 distributions.

The machine used to simulate all the examples, runs ubuntu with 8 GB memory and an

Number of Nodes	Setup	Distribution	Simulation Time[s]
10	1 to 3	Uniform	3.174
10	4 to 6	Normal	3.672
50	1 to 3	Uniform	30.297
50	4 to 6	Normal	43.615
100	1 to 3	Uniform	99.254
100	4 to 6	Normal	165.098
200	1 to 3	Uniform	324.376
200	4 to 6	Normal	819.835

Table 6.7: Simulation Time of the Many Nodes Example

intel core i7-6700HQ CPU at 2.6 GHz. The simulation times, on this machine, of the many nodes example is shown in Table 6.7. Having a low number of nodes in the network the simulation barley takes any time, regardless on how the nodes are positioned, due to the good performance of ns-3. At higher node counts it takes the simulation increasingly longer, as each node has to be calculated. In this case over 10 minutes. The reason the normal distribution takes longer than the uniform distribution is simply because more nodes are closer to each other. Therefore, more nodes received more messages. Even if the message is not directed at them, the node still hat to decide what to do, which increases the simulation time. Sadly, there is not much that can be done about this, except for upgrading the simulation hardware.

CHAPTER 7

Conclusion

In this thesis, an introduction on routing protocols was given and different routing protocols were discussed and compared. Further, it was explained what network simulators are and what they are good for. Also, different networks simulators were discussed, in addition how RPL was implemented in them. Different examples were highlighted, on which performance checks were done. It was demonstrated how RPL was implemented in ns-3, while satisfying conformity with the RFC [4], mentioned in Section 6.1. Distinct metrics, which are onboarding, flexibility, scalability, energy consumption and simulation time, were calculated from the previously explained examples.

7.1 Summary

With the evaluations, some key conclusions can be drawn. Firstly, the doublings value of the DIO trickle timer upon booting up a device has very little impact on the network, but it has a major impact on the responsiveness of the network in the long run, in terms of creating new paths and onboarding nodes. For very quick reaction times, a low DAO delay has to be set. While this make the propagation of routes through the network quicker, it also pollutes the network more. Also, when using OF0, it is a good idea to have the most nodes near the root node, as the mean hop count between any two nodes is smaller, which lowers the latency and enables the network to finish onboarding quicker.

Also, RPL does work when a node is moving within a static RPL network. By having a relative low speed of 2 m/s or 3 m/s and a doublings value of 10, the results in Section 6.3.2, show that most of the time the moving node has a valid parent and data is received. The parent selection of the moving node operates smoothly only under the condition that the moving node is designated as a leaf node. By setting the moving node as a leaf node, it adds all other non-leaf nodes it receives a DIO from to its parent set, as it itself is set to have an infinite rank. This leads to the assumption, that RPL could potentially used in applications where mobility of nodes is needed.

Finally, RPL appears to scale quite well, as it can handle nodes in a straight line up to 100 nodes and potentially more, which was not explored, due to high simulation times. Also placing node through a uniform and a normal distribution yielded good results, although it has to be stated that there were runs where, not 100% of nodes were found by the root node. The latency was also way below 1 second, except for higher node counts in the line example, as packets have to be forwarded through all nodes. The latency in general could also potentially be improved with a better OF. The biggest limiting factor in the examples of the success rate of received packets were collisions of LR-WPAN packets, due to the use of ad-hoc mode.

7.2 Future Work

In this section, further implementation and evaluation ideas for RPL are discussed.

7.2.1 Other LRWPAN modes

Since only the ad-hoc mode for LR-WPAN is implemented in ns-3, it would be interesting to see if other LR-WPAN modes would yield a better success rate of packages sent. Also, it would be compelling to see if other modes, especially the time-slotted mode, would increase or decrease the median latency of packets.

7.2.2 Objective Function

Since RPL is still explored and is currently not much used, even less is known about different OF for it. It could be interesting to see how RPL behaves, when the network adopts an OF instead of OF0, as it is not always to best to take the lowest hop count to the root node every time, as it is the case in OF0. While evaluating the examples for this thesis, it was seen that a node sometimes connect to a parent node, to which it has a bad connection. Such a connection has two detriments. Firstly, this makes the success rate of packets sent worse. Secondly, when the node does not get any packets from the parent and would detach from the DODAG, it could happen that the parent now connects to its former child node. This phenomenon is then only stopped after some time, due to the data path validation.

7.2.3 Energy Model

Instead of measuring packet count, an energy model could have been used in order to simulate energy consumption on nodes. There was already an attempt made in creating an energy model for LR-WPAN in [43], but it has not made it to the ns-3 release, yet.

7.2.4 Multipath

As stated in Section 5.3.3, this RPL implementation in ns-3 still lacks the multipath feature with multiple DAO parents. This can be added to add additional functionality as suggested by the RFC [4].

7.2.5 Multicast

As of [4], RPL can support the IPv6 multicast. Whether a RPL network supports multicast, is defined in the MOP field in the DIO base object. The multicast is not implemented in this RPL implementation. A starting point to implement the multicast support, would be to look at the static IPv6 routing of ns-3, and model it after that.

Acronyms

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks. 1, 2, 21, 37, 47

ACA Autonomous Component Architecture. 16

AODV Ad hoc On-Demand Distance Vector Routing. 8

ARP Address Resolution Protocol. 35

CLNS Connectionless Network Service. 9

CoAP Constrained Application Protocol. 1

COOJA Contiki OS simulator for Java. 15, 17, 18

CSMA/CA Carrier Sense Multiple Access/Collision Avoidance. 59

DAD Duplicate Address Detection. 35

DAG Directed Acyclic Graph. 37, 39, 41, 44, 48

DAO Destination Advertisement Object. 6, 21, 22, 37, 40, 41, 43, 44, 47–50, 52, 57, 67, 68

DAO-ACK Destination Advertisement Object Acknowledgement. 21, 22, 44, 49

DIO DODAG Information Object. 6, 22, 25, 30, 37, 40–44, 47, 48, 51–54, 57, 67, 69

DIS DODAG Information Solicitation. 6, 42, 47–49

DODAG Destination-Oriented Directed Acyclic Graph. 1, 6, 11, 29, 39–43, 47, 48, 50–52, 55, 57, 68

DUAL Diffusing Update Algorithm. 10, 11

EIGRP Enhanced Interior Gateway Routing Protocol. 10

IC Integrated Circuit. 16

ICMP Internet Control Message Protocol. 34, 35

ICMPv6 Internet Control Message Protocol for the Internet Protocol Version 6. 38, 39

IGP Interior Gateway Protocol. 9

INET extensible InterNETworking framework. 16

IoT Internet of Things. 1, 2, 11, 17, 58

IPv4 Internet Protocol version 4. 35

IPv6 Internet Protocol version 6. 1, 3, 35, 45, 47, 69

IS-IS Intermediate System to Intermediate System. 9, 44

J-Sim Java Simulator. 15, 18

LAN Local Area Network. 16

LLN low-power and lossy networks. 1–3, 6, 39, 44, 57, 58

LQI Link Quality Indicator. 42

LR-WPAN Low-Rate Wireless Personal Area Network. 2, 21, 30, 38, 39, 42, 47, 59, 61–63, 68

LSA Link State Advertisements. 7

LSDB Link State Database. 7

LSP Link State Protocol Data Unit. 9

MPR Multipoint Relay. 8, 9

MPR Selector Set Multipoint Relay Selector Set. 9

NDP Neighbor Discovery Protocol. 21, 35, 37, 38, 52

NED NEtwork Description. 16

ns-3 Network Simulator 3. 2–4, 15, 19, 21, 30, 33–41, 44, 45, 47, 48, 62, 63, 65, 67–69

OCP Objective Code Point. 42

OF Objective Function. 1, 6, 11, 17, 25, 39–43, 51, 57, 68

OF0 Objective Function Zero. 21, 24, 25, 29, 30, 41, 42, 50, 52, 54, 55, 63, 67, 68

OLSR Optimized Link State Routing. 8

OMNeT++ Objective Modular Network Testbed in C++. 15, 16, 18

OSPF Open Shortest Path First. 7, 9, 44

PAN Personal Area Network. 21

RIP Routing Information Protocol. 9, 10

RIPng Routing Information Protocol Next Generation. 10

RNG Random Number Generator. 47, 58

RPL IPv6 Routing Protocol for low-power and lossy networks. 1–4, 6, 7, 17, 18, 21, 22, 24, 25, 27, 29, 30, 33, 34, 36–45, 47–55, 57–59, 67–69

RREP Route Reply. 8

RREQ Route Request. 8

SDN Software Defined Networking. 17

TC Topology Control. 9

TCP Transmission Control Protocol. 35

TTL Time To Live. 8

UDP User Datagram Protocol. 21–23, 25, 27, 29–31, 35, 38, 47, 48, 52, 57, 60, 61, 63

WAN Wide Area Network. 16

WLAN Wireless Local Area Network. 8

WSN Wireless Sensor Network. 1–3, 17

Bibliography

[1] Maria Rita Palattella, Nicola Accettura, Xavier Vilajosana, Thomas Watteyne, Luigi Alfredo Grieco, Gennaro Boggia, and Mischa Dohler. Standardized Protocol Stack for the Internet of (Important) Things. *IEEE Communications Surveys & Tutorials*, 15(3):1389–1406, 2013. Conference Name: IEEE Communications Surveys & Tutorials.

[2] Olfa Gaddour and Anis Koubâa. RPL in a nutshell: A survey. *Computer Networks*, 56(14):3163–3178, September 2012.

[3] Joakim Eriksson, Niclas Finne, Nicolas Tsiftes, Simon Duquennoy, and Thiemo Voigt. Scaling RPL to Dense and Large Networks with Constrained Memory. In *Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks*, EWSN '18, pages 126–134, USA, February 2018. Junction Publishing.

[4] Roger Alexander, Anders Brandt, J. P. Vasseur, Jonathan Hui, Kris Pister, Pascal Thubert, P. Levis, Rene Struik, Richard Kelsey, and Tim Winter. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. Request for Comments RFC 6550, Internet Engineering Task Force, March 2012. Num Pages: 157.

[5] Belghachi Mohamed and Feham Mohamed. Experimental evaluation of RPL protocol. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 455–458, December 2016.

[6] Hedayat Hosseini, Elisa Rojas, and David Carrascal. Implementation of RPL in OMNeT++, July 2021. arXiv:2107.02551 [cs].

[7] N. Accettura, L. A. Grieco, G. Boggia, and P. Camarda. Performance analysis of the RPL Routing Protocol. In *2011 IEEE International Conference on Mechatronics*, pages 767–772, April 2011.

[8] P. Karkazis, P. Trakadas, Th. Zahariadis, A. Hatziefremidis, and H. C. Leligou. RPL modeling in J-Sim platform. In *2012 Ninth International Conference on Networked Sensing (INSS)*, pages 1–2, June 2012.

[9] Leila Ben Saad, Cedric Chauvenet, and Bernard Tourancheau. Simulation of the RPL Routing Protocol for IPv6 Sensor Networks: two cases studies. *International Conference on Sensor Technologies and Applications SENSORCOMM 2011*, September 2011.

[10] Simen Kurtzhals Hammerseth. *Implementing RPL in a mobile and fixed wireless sensor network with OMNeT++*. PhD thesis, University of Oslo, 2011.

[11] Shaimaa Abdel Hakeem, Anar Ahady, and Hyungwon Kim. RPL Routing Protocol Performance in Smart Grid Applications Based Wireless Sensors: Experimental and Simulated Analysis. *Electronics*, 8:186, February 2019.

[12] John Moy. OSPF Version 2. Request for Comments RFC 2328, Internet Engineering Task Force, April 1998. Num Pages: 244.

[13] Jack Drury, Peter Höfner, and Weiyou Wang. Formal Models of the OSPF Routing Protocol. *Electronic Proceedings in Theoretical Computer Science*, 316:72–120, April 2020. arXiv:2004.13286 [cs].

[14] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959.

[15] Prashant Maurya, Gaurav Sharma, Vaishali Sahu, Ashish Roberts, Mahendra Srivastava, and Scholar ALLAHABAD. An Overview of AODV Routing Protocol. *International Journal of Modern Engineering Research (IJMER)*, 2(3):728–732, June 2012.

[16] Aleksandr Huhtonen. Comparing AODV and OLSR routing protocols. *Telecommunications Software and Multimedia*, June 2004.

[17] Simon Shamoun, David Sarne, and Steven Goldfeder. Elastic Ring Search for Ad Hoc Networks. In Ivan Stojmenovic, Zixue Cheng, and Song Guo, editors, *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pages 564–575, Cham, 2014. Springer International Publishing.

[18] Thomas H. Clausen and Philippe Jacquet. Optimized Link State Routing Protocol (OLSR). Request for Comments RFC 3626, Internet Engineering Task Force, October 2003. Num Pages: 75.

[19] Hannes Gredler and Walter Goralski. IS-IS Basics. In *The Complete IS-IS Routing Protocol*, pages 79–108. Springer, London, 2005.

[20] Gary S. Malkin. RIP Version 2. Request for Comments RFC 2453, Internet Engineering Task Force, November 1998. Num Pages: 39.

[21] Thomas H. Cormen and Thomas H. Cormen, editors. *Introduction to algorithms*. MIT Press, Cambridge, Mass, 2nd ed edition, 2001.

[22] Oris Krianto Sulaiman, Amir Mahmud Siregar, Khairuddin Nasution, and Tasliyah Haramaini. Bellman Ford algorithm - in Routing Information Protocol (RIP). *Journal of Physics: Conference Series*, 1007:012009, April 2018.

[23] Gary S. Malkin and Robert E. Minnear. RIPng for IPv6. Request for Comments RFC 2080, Internet Engineering Task Force, January 1997. Num Pages: 19.

[24] Bob Albrightson, J. J. Garcia-Luna-Aceves, and Joanne Boyle. EIGRP–A Fast Routing Protocol based on Distance Vectors. UC Santa Cruz. Retrieved from https://escholarship.org/uc/item/9h48b8x2, 1994.

[25] Gabriel Y. Handler and Israel Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–309, 1980. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230100403.

[26] Donnie Savage, James Ng, Steven Moore, Donald Slice, Peter Paluch, and Russ White. Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP). Request for Comments RFC 7868, Internet Engineering Task Force, May 2016. Num Pages: 80.

[27] J. Heidemann, K. Mills, and S. Kumar. Expanding confidence in network simulations. *IEEE Network*, 15(5):58–63, September 2001. Conference Name: IEEE Network.

[28] Syful Islam, Yusuf Sulistyo Nugroho, and Md Javed Hoss. What network simulator questions do users ask? a large-scale study of stack overflow posts. *Indonesian Journal of Electrical Engineering and Computer Science*, 21(3):1622–1633, March 2021. Number: 3.

[29] Johannes Lessmann, Peter Janacik, Lazar Lachev, and Dalimir Orfanus. Comparative Study of Wireless Network Simulators. In *Seventh International Conference on Networking (icn 2008)*, pages 517–523, April 2008.

[30] Mohammed Kabir, Syful Islam, Md Hossain, and Sazzad Hossain. Detail Comparison of Network Simulators. *International Journal of Scientific & Engineering Research*, 5(10):203–218, October 2014.

[31] Lelio Campanile, Marco Gribaudo, Mauro Iacono, Fiammetta Marulli, and Michele Mastroianni. Computer Network Simulation with ns-3: A Systematic Literature Review. *Electronics*, 9(2):272, February 2020. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.

[32] Kazim Ergun, Xiaofan Yu, Nitish Nagesh, Ludmila Cherkasova, Pietro Mercati, Raid Ayoub, and Tajana Rosing. RelIoT: Reliability Simulator for IoT Networks. In Wei Song, Kisung Lee, Zhisheng Yan, Liang-Jie Zhang, and Huan Chen, editors, *Internet of Things - ICIOT 2020*, Lecture Notes in Computer Science, pages 63–81, Cham, 2020. Springer International Publishing.

[33] Hossam Mahmoud Ahmad Fahmy. Simulators and Emulators for WSNs. In Hossam Mahmoud Ahmad Fahmy, editor, *Concepts, Applications, Experimentation and Analysis of Wireless Sensor Networks*, Signals and Communication Technology, pages 547–663. Springer Nature Switzerland, Cham, 2023.

[34] George F. Riley and Thomas R. Henderson. The ns-3 Network Simulator. In Klaus Wehrle, Mesut Güneş, and James Gross, editors, *Modeling and Tools for Network Simulation*, pages 15–34. Springer, Berlin, Heidelberg, 2010.

[35] Andras Varga. OMNeT++. In Klaus Wehrle, Mesut Güneş, and James Gross, editors, *Modeling and Tools for Network Simulation*, pages 35–59. Springer, Berlin, Heidelberg, 2010.

[36] Ahmed Sobeih, Wei-Peng Chen, J.C. Hou, Lu-Chuan Kung, Ning Li, Hyuk Lim, Hung-Ying Tyan, and Honghai Zhang. J-Sim: a simulation environment for wireless sensor networks. In *38th Annual Simulation Symposium*, pages 175–187, April 2005. ISSN: 2331-107X.

[37] Patrick Kugler, Philipp Nordhus, and Bjoern Eskofier. Shimmer, Cooja and Contiki: A new toolset for the simulation of on-node signal processing algorithms. In *2013 IEEE International Conference on Body Sensor Networks*, pages 1–6, May 2013. ISSN: 2376-8894.

[38] Craig Thomson, Isam Wadhaj, Imed Romdhani, and Ahmed Al-Dubai. Performance evaluation of RPL metrics in environments with strained transmission ranges. In *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pages 1–8, November 2016. ISSN: 2161-5330.

[39] Eleftherios Tsapardakis, Mike Ojo, Periklis Chatzimisios, and Stefano Giordano. Performance Evaluation of SDN and RPL in Wireless Sensor Networks. In *2018 Global Information Infrastructure and Networking Symposium (GIIS)*, pages 1–5, October 2018. ISSN: 2150-329X.

[40] Olfa Gaddour, Anis Koubâa, Shafique Chaudhry, Miled Tezeghdanti, Rihab Chaari, and Mohamed Abid. Simulation and performance evaluation of DAG construction with RPL. In *Third International Conference on Communications and Networking*, pages 1–8, March 2012. ISSN: 2163-663X.

[41] Chet Hosmer. Chapter 9 - Network Forensics: Part II. In Chet Hosmer, editor, *Python Forensics*, pages 237–263. Syngress, Boston, January 2014.

[42] Lubna Alazzawi and Ali Elkateeb. Performance Evaluation of the WSN Routing Protocols Scalability. *Journal of Computer Systems, Networks, and Communications*, 2008, January 2008.

[43] Vishwesh Rege and Tommaso Pecorella. A Realistic MAC and Energy Model for 802.15.4. In *Proceedings of the 2016 Workshop on ns-3*, WNS3 '16, pages 79–84, New York, NY, USA, June 2016. Association for Computing Machinery.

[44] ns-3, User Manual, https://www.nsnam.org/docs/manual/html/index.html, 2024.

[45] William A. Simpson, Thomas Narten, Erik Nordmark, and Hesham Soliman. Neighbor Discovery for IP version 6 (IPv6). Request for Comments RFC 4861, Internet Engineering Task Force, September 2007. Num Pages: 97.

[46] Neighbor Discovery issue, https://gitlab.com/nsnam/ns-3-dev/-/issues/691, June 2022.

[47] Gabriel Montenegro, Jonathan Hui, David Culler, and Nandakishore Kushalnagar. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. Request for Comments RFC 4944, Internet Engineering Task Force, September 2007. Num Pages: 30.

[48] Ipv6RawSocktets multicast address issue, https://gitlab.com/nsnam/ns-3-dev/-/issues/1039, February 2024.

[49] ICMPv6 checksum issue, https://gitlab.com/nsnam/ns-3-dev/-/issues/1036, February 2024.

[50] P. Levis, Thomas H. Clausen, Omprakash Gnawali, Jonathan Hui, and JeongGil Ko. The Trickle Algorithm. Request for Comments RFC 6206, Internet Engineering Task Force, March 2011. Num Pages: 13.

[51] Pascal Thubert. Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL). Request for Comments RFC 6552, Internet Engineering Task Force, March 2012. Num Pages: 14.

[52] Richard Bohannon and Addison Andrews. Normal walking speed: A descriptive meta-analysis. *Physiotherapy*, 97:182–9, September 2011.

[53] Jonathan Hui and J. P. Vasseur. The Routing Protocol for Low-Power and Lossy Networks (RPL) Option for Carrying RPL Information in Data-Plane Datagrams. Request for Comments RFC 6553, Internet Engineering Task Force, March 2012. Num Pages: 9.

[54] G. Anastasi, E. Borgia, M. Conti, and E. Gregori. Wi-fi in ad hoc mode: a measurement study. In *Second IEEE Annual Conference on Pervasive Computing and Communications, 2004. Proceedings of the*, pages 145–154, March 2004.