Improving Source Privacy in Routing Protocols for Wireless Sensor Networks

Owen Connors

Department of Computer Science

University of Warwick

Supervised by Arshad Jhumka

Year of Study: 3rd

3 May 2022

Abstract

Source Location Privacy (SLP) is an important property to maintain for Wireless Sensor Networks. Many successful approaches have been proposed, but these all incur significant penalties of message count and complexity. The existing routing protocol RPL has some properties that aid SLP, which have not been investigates thus far. We investigate these properties, then propose and test SLP improvements to RPL, without a significant increase in complexity.

Keywords: Wireless Sensor Networks, Source Location Privacy, RPL, Routing Protocols, Contiki, Networks

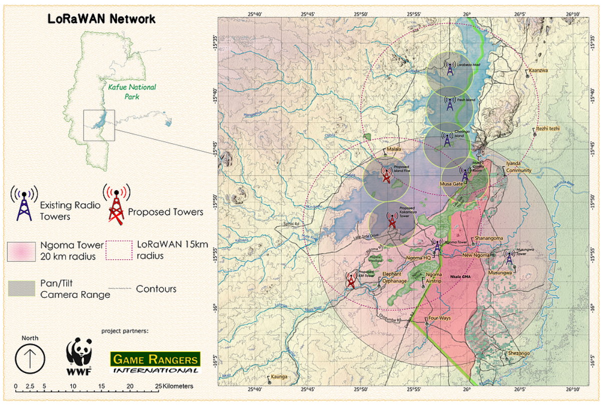
Contents

Chapter 1

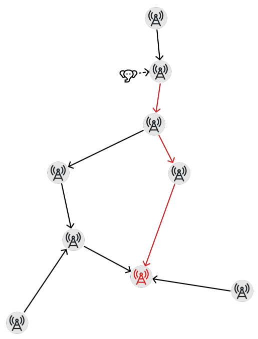
Introduction

The WWF recently installed a GPS tracking system for elephants and other endangered animals in Kafue National Park, Zambia (WWF, 2020a). This system installed trackers on each animal with short-range radio communication, used to send location information to a nearby radio tower. These radio towers then relay this information to a central headquarters.

If a poacher was hunting one of these animals, they could intercept a radio transmission between two of these towers with a low-cost radio receiver. Assuming these messages are sufficiently encrypted, the poacher cannot read the contents of the message and extract the exact position contained within. However, they could determine the direction towards the origin of this packet's source – with a directional antenna for instance – and follow this signal towards this hop's origin. Over repeated messages from the same animal, and by repeating this process, the poacher would be able to trace a path to the endangered animal. Thereby, making the system meant to track and ensure the safety of these animals, in fact, jeopardize them further.



(a) Map of Kafue (WWF, 2020a).



(b) Graph of radio towers

Figure 1.1: Map (a) and graph (b) of the radio tower network in Kafue National Park. The graph shows an elephant's tracker sending a message to the tower, which relays that message on to their headquarters.

Systems such as these exist at many scales and for many applications: conservation, such as above in WWF (2020a,b), target tracking (Dyo et al., 2012) and habitat monitoring (Mainwaring et al., 2002; Badescu and Cotofana, 2015); environmental control; logistical and warehouse support; manufacturing monitoring (Holmes, 2004) medical services, both in hospital (Bauer et al., 2000; Hakim et al., 2006) and out (Milenković et al., 2006); military use (Arampatzis et al., 2005); and many more (Akyildiz et al., 2002; Arampatzis et al., 2005). All of these systems are characterized as Wireless Sensor Networks (WSNs), which form a vital part of the Internet of Things.

Wireless Sensor Networks consist of a set of small, low-powered devices, which can sense attributes of the environment, and communicate wirelessly using radio signals amongst themselves. Sensor nodes are usually heavily constrained: processing capacity, processing power, memory capacity, and total power usage (especially when battery-powered) (Bormann et al., 2014). These constraints stem from a desire to keep size and cost per node low, due to the large quantities and minimal presence often required from these nodes.

The wireless nature of WSNs allow for the network to be much more flexible and discrete, since there is no reliance on fixed wired links – which can be expensive and difficult to fit in a constrained or pre-existing environment. This wireless nature comes with a few drawbacks however: communication between nodes can be unreliable or drop entirely, routing and conditions can change at any point, and messages are much more easily eavesdropped upon. As such, security should be upheld especially in safety- and security-critical systems.

Protecting the exact content of the message transmitted by sensor nodes is referred to as *content-based privacy*, such as reading or modifying an eavesdropped encrypted message. Content privacy has been researched greatly, with methods such as encryption schemes specific to sensor networks (Perrig et al., 2002) and other cryptographic techniques. *Context-based privacy* is the term given to protecting information that can be observed or inferred from the situation or context the message is sent in, such as the origin location of a message or the frequency of the monitored event. Context-privacy is a multi-faceted issue that can be very difficult to address. Protecting the origin location of the source, or source location, is *source location privacy (SLP)* (Gu et al., 2019).

SLP is an important concern in many applications, especially security-critical systems. Hiding the location of the asset being monitored is vital, whether it's an endangered animal from a poacher, or a squad of soldiers on a battlefield from the enemy. This situation is often formalized as an *asset monitoring* problem (Kamat et al., 2005), where *source nodes* collect and send information about the asset to a central collecting *sink node*, through a network of relaying nodes transmitting messages in multiple hops. Kamat et al. (2005) further formalizes this into the *Panda-Hunter Game*, with a single source that periodically reports data to the sink, and a hunter that tries to back-trace the messages' path to the source. The aim is to make it more difficult for the hunter to find the panda.

Many techniques for SLP exist, mainly grouped into those derived from phantom routing and from fake sources. Phantom routing was originally proposed in the seminal paper Kamat et al. (2005) where the message carries out a short directed random walk away from the source before being routed towards the source. Many variations exist to improve the SLP quality of this random walk: GROW (Xi et al., 2006), PRLA (Wang et al., 2008), or phantom walkabouts (Gu et al., 2019) for instance. Fake source techniques use spoofed messages sent by fake sources to disguise actual traffic and lead the hunter away from the actual source: the seminal SLFSR (Ozturk et al., 2004) used disproportionate amounts of energy. The changes to fake source selection in Jhumka et al. (2015) improves upon this, with Bradbury et al. (2018) making this technique dynamically update to the network condition.

These techniques are sometimes combined (Roy et al., 2015), and many others exist, including: ILP routing, where messages are bunched together to reduce attacker information gain (Bradbury and Jhumka, 2017); Raj et al. (2014) uses data mules to collect packets from sources and deliver them elsewhere; Cyclic Entrapment lures the attacker into a cycle of messages (Ouyang et al., 2006).

All these techniques can be effective, with some reducing source capture to below 1% (Bradbury and Jhumka, 2017). The downside to many of these methods is their power usage, especially for fake source techniques (Gu et al., 2019), and typically complicated implementations. In WSNs, it is very desirable to minimize complexity, so it may be worth considering minor variants to existing standard routing protocols. Many of these routing techniques work on a standard baseline technique called *flooding* (Kamat et al.,

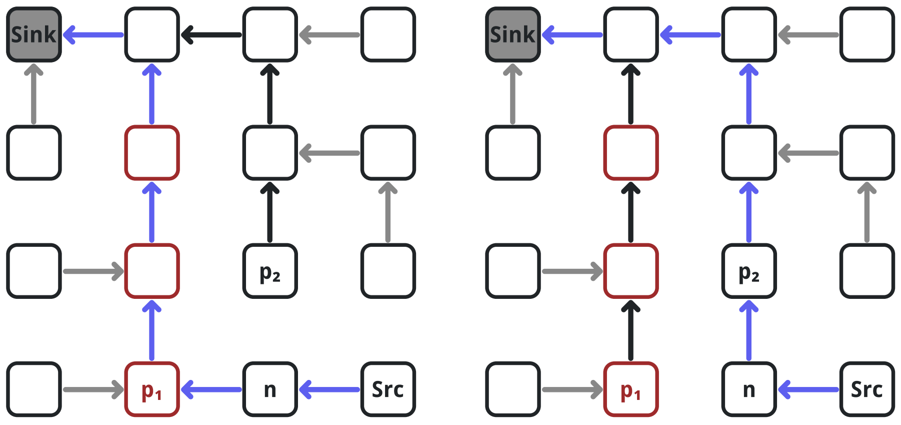


Figure 1.2: Before and after diagram of RPL stranding. Node n, which has the source in it's subtree, switches parents from p_1 to p_2, which would strand the attacker if they were in ancestors of p_1 but not p_2 (those with a red outline).

2005). In flooding, starting at the source, each node broadcasts the message to its neighbours, who broadcast it in turn, if they have not seen the message before. This routing protocol is utilized due to it's incredible simplicity, however it is very inefficient as every node has to receive and broadcast each message.

An alternative routing protocol that has gained much use is the IETF standard *RPL*, (pronounced 'Ripple') which constructs and maintains a routing tree or DAG, such that routing up the tree provides the most efficient route (Alexander et al., 2012; Gaddour and Koubaa, 2012).

In the effort to maintain optimal routing, the routing tree sometimes updates when a node finds an alternative neighbour that provides a superior route (Alexander et al., 2012). In this situation where a subtree changes parent from p_1 to p_2, traffic that previously flowed through the parents of p_1 now flows through the parents of p_2. If a hunter was following source-sink traffic towards p_1 and was in this region when the parent changed, the hunter would be *stranded*. In these situations, these strandings are rare and entirely at the whim of the RPL algorithm. In this paper, we study the SLP level provided by RPL, and we aim to improve the SLP characteristics of RPL by increasing the frequency of these changes through a variety of minor modifications to RPL.

Chapter 2

Background

In this section, we will describe the required background to this report. This includes Wireless Sensor Networks, routing protocols, the Panda-Hunter game, existing techniques, the simulator used, and details of the RPL protocol.

2.1 Wireless Sensor Networks

Wireless Sensor Networks are a vital part of the Internet of Things. They serve as a low-intrusiveness method to monitor conditions over a large area. Each sensor node is a low-power computing device fitted with a sensor for whichever property the network needs to monitor. These sensors have heavily restricted capabilities for processing, storage, communication reliability, and energy supply – due to size and price constraints (Bormann et al., 2014). The sensors are equipped with a wireless interface, and are capable of communicating with all other sensor nodes with the same certain communication range. This network can be represented as a graph with sensor nodes as graph nodes, and links between nodes within range of each other as edges (Bradbury et al., 2018). One of the nodes in this network is the *Sink*, which collects information from all the sensors in the network. This is analogous to the sink of a graph, there is no outgoing traffic, only incoming. The sink is much more powerful than the sensor nodes, and collects all the data of the WSN, either sending this immediately on to a server, or storing it for later physical retrieval (Conti et al., 2013).

Network Topology: Throughout most of this paper, we will only consider grid networks of sensor nodes, as it is a standard and idealized model of a sensor node layout that makes automated generation of test cases more feasible. This should make comparisons to other schemes more reliable and direct.

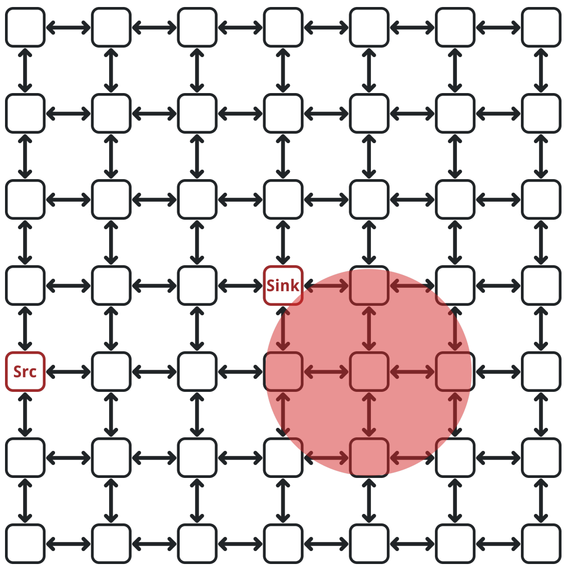


Figure 2.1: Example grid network. The Sink and Source (Src) are labelled, and nodes within communication range of each other are connected by an edge. The red circle gives an example of transmission range.

2.2 Routing Protocols

The routing protocol defines how messages are routed through the network when multi-hop routing is required. Sensor nodes send messages to the sink either on changes to the measured value (e.g. presence of object) or periodically (e.g. reporting air temperature every minute). These messages are transported through multi-hop routing through other sensor nodes to the sink. Many papers use flooding as a base routing protocol, which we also do here. We compare RPL and our modifications in a base safety period defined relative to flooding. Examples of both protocols are shown in Figure 2.2 and 2.3.

2.2.1 Flooding

Many SLP methods are routing protocol agnostic; they can be implemented on top of any routing protocol. The usual choice is flooding because of its simplicity and consistency, and because it is an easily standardized baseline. Furthermore, any technique that can improve SLP for flooding will provide equal or better performance with another routing protocol: at worst, the alternative will provide the same shortest-hop-count path as flooding, meaning the SLP performance cannot be worse.

Flooding is an appropriate name for this protocol. When a node receives a message from another, it broadcasts it to all neighbouring links apart from the source of the message. To stop the message being infinitely passed in cycles, nodes track previously received packet sequence numbers, and do not forward messages they have previously received (Kamat et al., 2005; Sanchez, 2020).

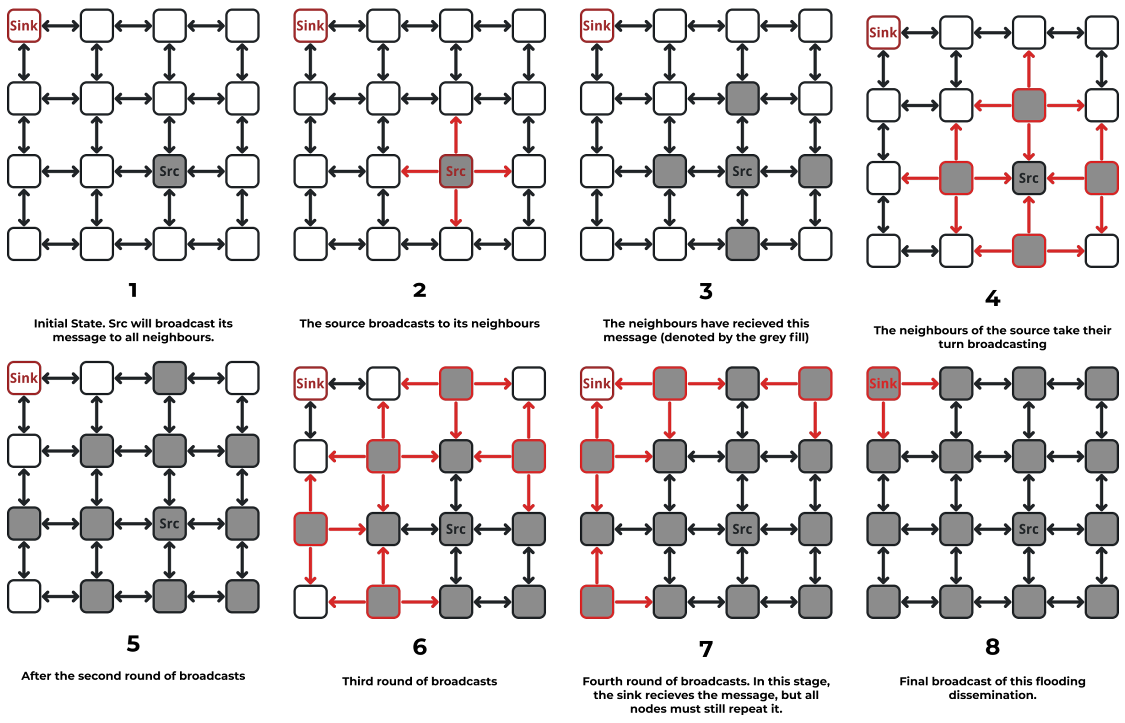

Sink
Src
1
Initial State. Src will broadcast its message to all neighbours.
2
The source broadcasts to its neighbours
3
The neighbours have recieved this message (denoted by the grey fill)
4
The neighbours of the source take their turn broadcasting
5
After the second round of broadcasts
6
Third round of broadcasts
7
Fourth round of broadcasts. In this stage, the sink recieves the message, but all nodes must still repeat it.
8
Final broadcast of this flooding dissemination.

Figure 2.2: Example of flooding in a small network. This shows how all nodes must receive and broadcast the same signal, an expensive operation. Red nodes are currently broadcasting along red connections; grey nodes have previously transmitted the message.

2.2.2 RPL

RPL constructs a routing graph pointing towards the sink. This is called a Destination-Oriented Directed Acyclic Graph (DODAG) (Alexander et al., 2012). Figure 2.4 shows an example network and corresponding DODAG. In many cases, this DODAG is a tree, with optional secondary choices for parents recorded, for reliability and redundancy. As such, most language describing the DODAG will treat it like a tree. When a node wishes to send a message, it simply sends it upwards to its parent(s). The parent then forwards this message upwards. This is repeated until the sink is reached. This ensures messages are only sent where needed: towards the destination.

This is significantly more efficient than flooding, since messages are only sent to one or two neighbours at a time instead of to all neighbours, making the sending operation approximately $O(\sqrt{n})$ instead of $O(n)$, where n is the number of nodes in a square graph. RPL is specifically designed for low-power and lossy networks making optimizing power usage and reliability of the utmost importance. (Gaddour and Koubaa, 2012)

More details on RPL follow in Section 2.7.

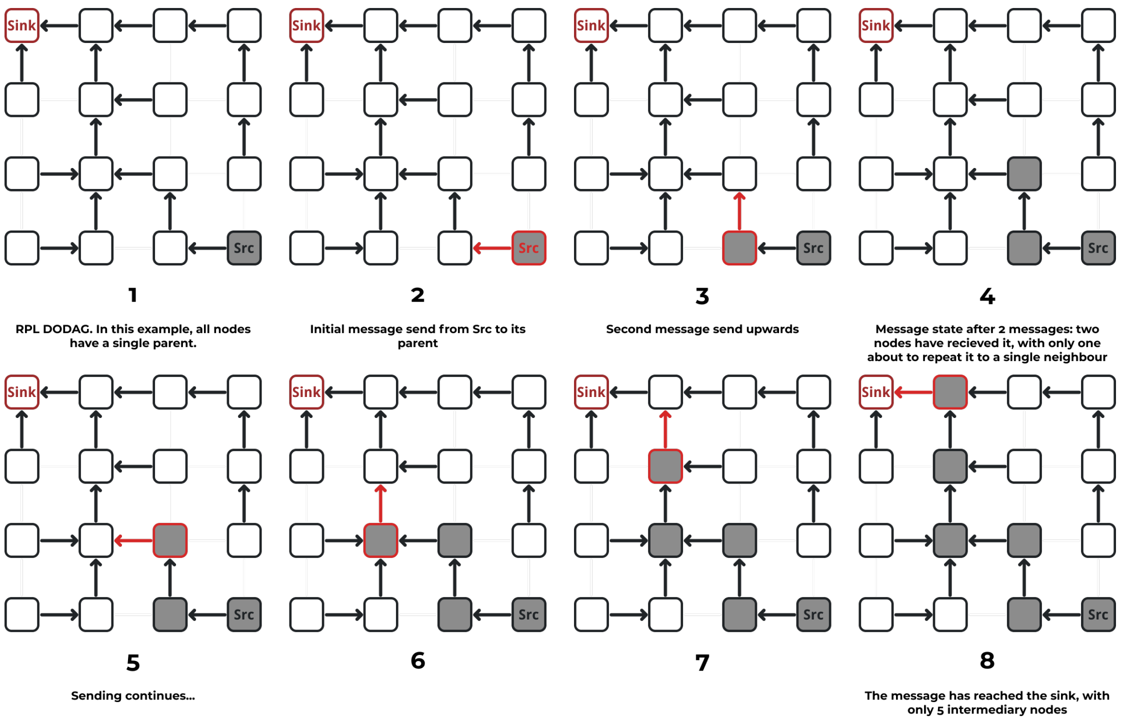
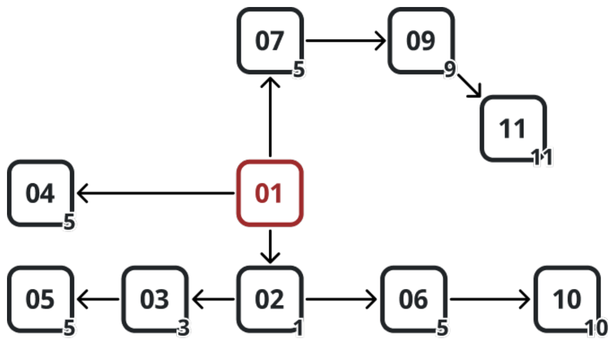
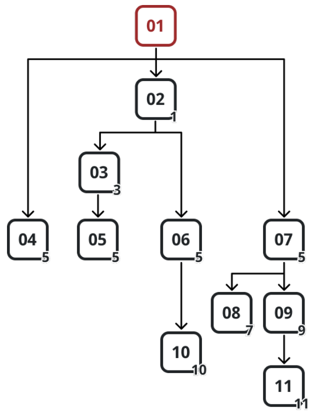

Sink
Src
1
RPL DODAG. In this example, all nodes have a single parent.
2
Initial message send from Src to its parent
3
Second message send upwards
4
Message state after 2 messages: two nodes have recieved it, with only one about to repeat it to a single neighbour
5
Sending continues...
6
7
8
The message has reached the sink, with only 5 intermediary nodes

Figure 2.3: Example of RPL in a small network. This shows the relative reduction in number of nodes required to transmit a message compared to flooding. RPL routes through a much reduced set of nodes.



(a) Example Network.



(b) Example Routing tree (DODAG).

Figure 2.4: Example of network and DODAG. Nodes are numbered with their IDs, their rank is the adjacent number. Arrows show parents in the DODAG.

2.3 Panda-Hunter Game

The *Panda-Hunter Game* is a formalization of the SLP problem first described in Ozturk et al. (2004) and formalized in Kamat et al. (2005). The Panda-Hunter game is very similar to the Elephant-Poacher example given in the Introduction. In a WSN with a Sink, a single static Source is randomly chosen. This Source periodically sends updates on the panda at that location to the Sink, following a defined multi-hop routing protocol.

The hunter starts at the Sink (as it is the only guaranteed recipient of messages from anywhere), and when the hunter eavesdrops a message being received by the Sink, they follow this message to the intermediate sender of the message. This is repeated until the hunter reaches the Source, or the safety period elapses.

2.3.1 Safety Period

Ideally, any SLP scheme would ensure the panda is never captured. We consider two cases to model: a static and mobile panda (source). If the source is static, then the attacker has the trivial tactic of performing an exhaustive search of the network, guaranteeing a capture at some point. Therefore, some limitation must be provided in these cases, typically a time-bound. In the mobile case, the source can move between nodes, and therefore would only be at a particular node for a certain period of time. This can be modelled with a time-bound before the source moves elsewhere. This time-bound is introduced in the form of a *safety period* (Bradbury et al., 2018).

The safety period has two dual definitions. The first being the most intuitive but least practical: The *safety period* is the time it takes for the hunter to capture the target. However, this means simulation runtime is unbounded, which can be highly impractical. An analogous definition is considering the *capture ratio* within a set period of time – the safety period. The capture ratio is simply the proportion of the time that the hunter captures the source. The safety period is established per network topology and source-sink location, as a set multiple of the time it takes to capture with a baseline protocol (Gu et al., 2019). Basic flooding is usually used as a baseline, as it gives the lowest SLP level (Kamat et al., 2005); the hunter always has the minimum hop-length route to the source to follow.

2.4 Attacker Model

The exact attacker model implemented varies between techniques, although most base their attacker implementation on (Kamat et al., 2005), where the attacker is termed a *distributed eavesdropper*. Attacker behaviour is often classified along two dimensions: presence and actions (Jhumka et al., 2015). Presence represents the scope of the attacker's knowledge and movement around the network, and actions represent the attacker's capability to interfere with the network.

Presence A global attacker has knowledge of the entire network, and can view every packet sent anywhere, whereas a local attacker only has knowledge of a subset of the network (Conti et al., 2013) – often just a single node's traffic, although this could range to multiple colluding bodies like in Jhumka et al. (2011). If the attacker is local, it is desirable to differentiate between static and distributed (mobile) attackers. While some techniques exist to counter global attackers, these are prohibitively computationally expensive, as the attacker must be completely fooled as to where the actual source is. Completely hiding the sources requires many or all nodes to behave as fake sources, massively increasing transmission count and computation required (Mehta et al., 2007; Yang et al., 2013). Furthermore, establishing global visibility requires significant resources (such as towers with sensitive long-range antennae or a large number of colluding attackers) (Gu et al., 2019), so is rarely encountered. Therefore, in this paper we will consider a local attacker. As a static attacker is a subset of distributed attackers, we choose a local distributed attacker, as the most powerful feasible form of attacker.

Actions Attacker actions exist on a wide scale, from just eavesdropping on messages to disrupting and modifying communication to reprogramming nodes entirely (Jhumka et al., 2011). A simple eavesdropping attacker is often chosen, due to any disruption/disturbing attack (such as a Denial of Service attack) moving and interrupting traffic flow, which the attacker requires to back-trace to the source. As the attacker's goal is to find this source, any attacks around crashing or disturbing the network are counter-productive. If an attacker attempts to impersonate or modify nodes, the WSN could potentially detect the intrusion and respond by stopping transmission in the affected area (Bradbury et al., 2018; Kamat et al., 2005), similar to the methods in Nassiri et al. (2016). As we are dealing with context-privacy here, it is a standard assumption that the contents of the message are suitably encrypted. Certain passive attacks, like attempting to steal encryption keys from a node, can take over half an hour (Benenson et al., 2008). This is prohibitively long in an even slightly time-limited SLP situation. Consequently, all attack types stronger than eavesdropping risk the attacker's ability to trace to the source, so should be discounted as attacker strategies (specifically for the SLP problem).

The attacker is also considered device- and resource-rich (Kamat et al., 2005). That is, the attacker has sufficient equipment (antenna and analyzers) to measure the direction (angle) of the message, and the ability to move to this intermediate source arbitrarily fast.

For a distributed eavesdropper, the most reliable technique (certainly with no SLP-enhancing measures) is to start at the sink and when the attacker eavesdrops on a data packet sent to the sink, they immediately move to the immediate source of that transmission. This is repeated until the attacker finds the source.

Previous research disagrees on whether the attacker should be allowed to have knowledge of the protocols and methods used in the system. All assume basic knowledge to be able to read and interpret some basic header information of the packets, such that the attacker can detect duplicate messages (Kamat et al., 2005) (but not faked decoy messages (Ozturk et al., 2004)). However, some assume the attacker could know protocols used, like Kamat et al. (2005), and some do not (Gu et al., 2019). For this paper, we assume that the attacker knows the network is operating under RPL, but not necessarily under the proposed modifications – although this does not change the modelled attacker's behaviour.

Originally, some papers assumed the hunter would backtrack if they had not intercepted any messages for a period of time (Ozturk et al., 2004), however this has since been dropped from standard descriptions of the problem. This is most likely due to the additional complexity of modelling such an attacker, for example Gu et al. (2019); Bradbury et al. (2018).

2.5 Existing Source Location Privacy Techniques

Conti et al. (2013) splits SLP techniques into eleven categories, however, these categories are somewhat outdated and many include only a single technique or two. As such, we will focus on two main families: random walk and fake sources. These techniques (along with the SLP problem) were initially proposed in the seminal papers Kamat et al. (2005) and Ozturk et al. (2004), which also formalized the SLP problem.

Random Walk techniques are based on phantom routing, proposed in Kamat et al. (2005). These techniques involve the packets following some form of random walk through the network before routing to the sink. The aim of these techniques is to randomize the apparent direction the packet came from, such that the hunter cannot reliably trace the source.

- *Pure Random Walk:* Kamat et al. (2005) originally proposes that the messages follow a short fixed length fully random walk. However, this still reveals significant directional information, both as the walk is short, and by the Central Limit Theorem, the packets will not reliably distance themselves enough from the original source.
- *Phantom Routing:* Kamat et al. (2005) then proposes this walk is directed in a simple manner: a landmark node is picked (e.g. west-most), and nodes record whether neighbours are closer or not to this landmark. When a packet is sent, it is randomly sent towards or away from the landmark for the random walk. This can be further improved by using a single-path routing scheme instead of flooding (PSRS) (Kamat et al., 2005; Gu et al., 2019). However, these methods have poor SLP performance, as

the random walk reuses paths too often (Gu et al., 2015) (Shaikh et al., 2010).

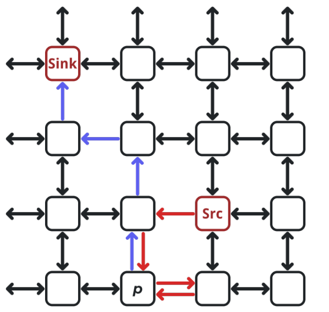
- *Self-Adjusting Directed Random Walks:* Zhang (2006) modifies Phantom Routing to avoid regions that would trap the packet in a position where no nodes exist in the desired direction.

- *Phantom Routing with Locational Angle (PRLA):* Wang et al. (2008) constructs the random walks based on the inclination angle from the node to the sink and source in order to construct a more varied set of random walks.

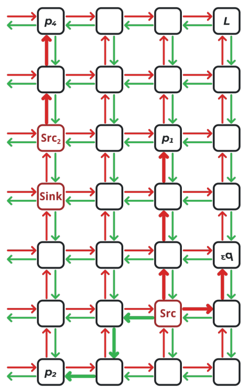
- *Greedy Random Walk (GROW):* (Xi et al., 2006) This is based on Rumour Routing (Braginsky and Estrin, 2002), which constructs random routes from the source and sink simultaneously, and reduces repetition in the walk by avoiding previously visited nodes.

- *Phantom Walkabouts:* Gu et al. (2017, 2019) proposes Phantom Walkabouts, an improvement on Phantom Routing, where two landmark nodes are used, creating four directional options, and more variety in path length leading to much improved SLP.

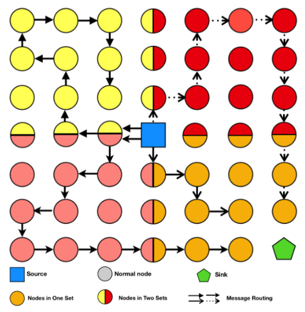
- *Multirings:* Yao et al. (2015) routes traffic radially around the sink before routing inwards. This ensures fully random approach directions to the sink, ensuring maximum variation.



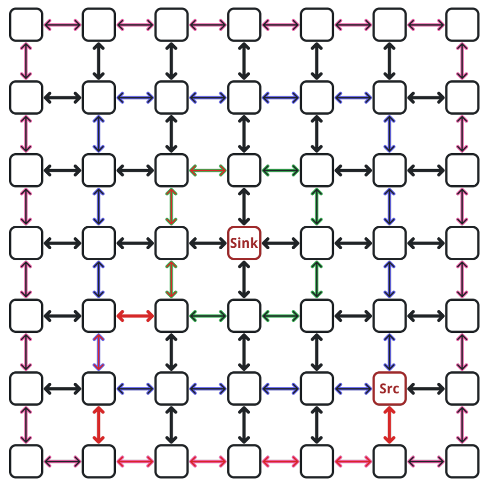
(a) Random Walk: Fully random walk (shown in red), before routing to the sink. Here the routing is shown as a single path for clarity, though flooding is often used instead.



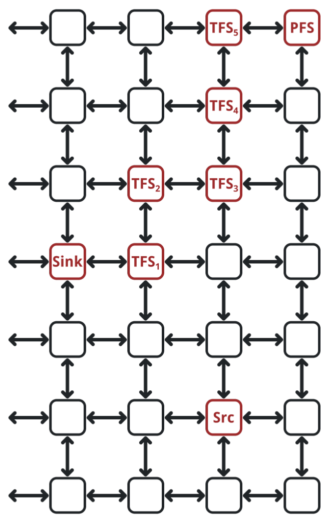
(b) Phantom Routing: At each step, nodes route towards (red) or away (green) from the landmark (L). The routing to the source from each p_i is omitted for clarity here. Examples p_1, p_2 shows routing towards and away from L. p_4 shows a non-beneficial route.



(c) Phantom Walkabouts: Two landmarks are used to create the routing directions. This diagram is from Gu et al. (2019) and shows example routing directions from the source (in blue).



(d) Multirings: Rings around the sink are established (green, blue, pink). When routing, the message gets taken to a random ring (ring 3 here) and routed around to a set angle. This process is repeated for a second ring before being routed to the sink.



(e) Fake Sources: When the sink receives a message, a trail of temporary sources is created, which each broadcast for a short amount of time, before picking the next temporary source. When backed into a corner, the temporary source becomes a permanent source.

Figure 2.5: Examples of some of the SLP techniques described here.

Fake Source techniques were also initially explored in Kamat et al. (2005); Ozturk et al. (2004), however these were discounted due to the disproportionate increase in energy-use required to disguise traffic. Fake source techniques involve designating some nodes *fake sources*, which repeatedly broadcast messages to imitate a source. This can pull the attacker away from the actual path. If fake source frequency and fake broadcast frequency is too high, much energy is wasted, but too low and it does not pull enough. If all nodes are considered fake sources, full SLP can be achieved, but at a prohibitively high cost. This technique is actually used in Mehta et al. (2007) to protect against an attacker with a global view. Yang et al. (2013) provides a statistics-based technique that still delivers high SLP levels against a global attacker with a significantly reduced energy cost.

Further research has shown that with a good selection of nodes, SLP levels can be reduced to very low levels, with reasonable efficiency against a local attacker. Jhumka et al. (2015) significantly improves the selection of permanent and temporary fake sources to form a trail drawing the hunter away from the actual source. One limitation of this technique is its requirement to fix parameters at compile time, making it impractical for variable and changing networks. Dynamically adjusting parameters is included in Bradbury et al. (2018) which builds on this technique.

Other Techniques include Cyclic Entrapment (Ouyang et al., 2006) which aims to trap the attacker in a cycle; ILP routing (Bradbury and Jhumka, 2017) strategically delays messages to group them, reducing attacker progress; and Data Mules (Raj et al., 2014) which collect source packets, and drop them all at once elsewhere. Onion Routing is a common technique for source anonymization in the wider internet, however it was discounted for many years for WSNs as it had too much overhead. Palmieri (2016) presents a modified onion routing solution suitable for source location privacy in WSNs.

Overall, all these given techniques can significantly improve SLP levels, but most have significant cost overheads, especially compared to single-path routing (instead of flooding). In this paper, we look at what minor changes could be made to RPL to improve SLP levels in a minor capacity.

2.6 OS and Simulator

The low-power and cost constraints of sensor devices severely restrict the capabilities of Internet of Things Operating Systems. IoT devices can typically be split into low- and high-end devices: high-end devices can support a full OS such as Linux or Windows, examples include a Raspberry Pi or smartphone. Low-end devices are usually much more constrained, and often provide the absolute minimum (Hahm et al., 2016). Musaddiq et al. (2018) gives the example of Crossbow's Telos B, which provides only 10kB of RAM and 48kB of flash memory, or the Zolertia Z1 with 8kB RAM and a 92kB Flash memory (Zolertia, 2010). These devices are focused on WSN applications, and so have the minimum capabilities to read and process the sensor information, then transmit and relay that information throughout the network. Furthermore, these devices should be flexible as IoT and WSN applications significantly vary over a large number of characteristics.

These constraints only get more pressing as denser networks of cheaper devices become more prevalent (Musaddiq et al., 2018). Low-end devices are not expected to get significantly more powerful due to a desire to keep cost down and that this class of device is sufficient for processing and communicating sensor signals.

Contiki (Oikonomou et al., 2022) and TinyOS are considered two of the most prevalent open-source IoT OSs, and feature in most comparisons. There are many alternatives, such as FreeRTOS (in Musaddiq et al., 2018) and RIOT (in Zikria et al., 2019), amongst others compared in Javed et al. (2018). TinyOS is often preferred in WSN applications, while Contiki is more popular for other IoT tasks (Javed et al., 2018).

Both operating systems provide a high-quality simulator: TOSSIM (Levis et al., 2003) and Cooja (Osterlind et al., 2006) for TinyOS and Contiki respectively. Both effectively model sensor node behaviour, with TOSSIM being used especially commonly for SLP problems: Gu et al. (2019) or Bradbury et al. (2018) for instance. However, TOSSIM only simulates down to the OS level (Stehlık, 2011) – not truly providing accurate simulations. This is a common trade-off for ease of computation, however it does lead to some inaccuracy. Cooja provides "cross-level simulation" – simulation of both low-level hardware and the high-level application simultaneously (Osterlind et al., 2006). Furthermore, Cooja and Contiki have a modular architecture that facilitates development and extension.

2.6.1 Protocols

Both TinyOS and Contiki provide implementations of RPL (Parasuram (2016) compares these and more: SimpleRPL and RIOT-RPL). ContikiRPL and RIOT-RPL are the most fully featured as compared in Parasuram (2016), however TinyRPL was applicable in many cases. This paper also introduced the RPL-Lite standard – a simplified version of the RPL specification that should allow better standardization; the RPL standard has many optional components, so implementations are often not fully interoperable (Parasuram, 2016). Indeed, Korte et al. (2012) show the ContikiRPL and TinyRPL implementations do not combine well. Both TinyOS and Contiki now implement RPL-Lite in addition to their more complex implementations.

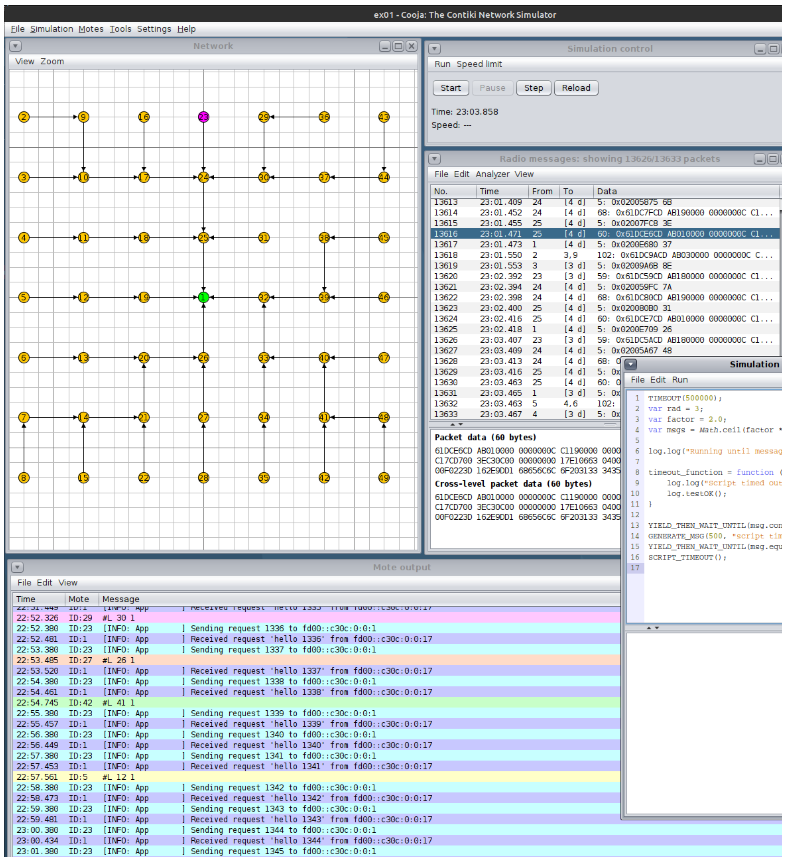
ContikiRPL provides features such as allowing multiple DODAGs over the same network, however these features are rarely used in many applications, and certainly are not required for the simple cases used to test SLP problems. As such, a RPL-Lite implementation is sufficient, and due to the accuracy of the Cooja simulator, Contiki and Cooja were chosen.

2.6.2 Cooja

The Cooja simulator is written in Java, and has an extensible framework, allowing for plug-ins to enhance functionality, such as recording radio logs as `.pcap` files [1] – a common format, especially used with the pcap library and the well-known network analyser tool Wireshark (Munz and Carle, 2008). Cooja is flexible in other ways. It allows modelling of different hardware, as well as software, within the same simulation. Node hardware may be compiled to the native host processor, or through a micro-processor simulator such as for the Texas Instruments MSP430[2] platform. Furthermore, these nodes could be driven directly by Java code or another IoT OS. Implementation in Java runs faster, but the code is not actually deployable – it makes direct use of Cooja's API instead of a low-end device emulator. This is useful for creating test cases for nodes. For example, the edge nodes could be directed by an external pattern, while the response of actual nodes is tested. Cooja also allows for use of standard debugging tools such as `gdb` (Osterlind et al., 2006).

[1] `https://github.com/iPAS/Cooja-RadioLogger` created by Cetic, modified by iPAS.
[2] Texas Instruments MSP430: `https://www.ti.com/microcontrollers-mcus-processors/microcontrollers/msp430-microcontrollers/overview.html`

As stated before, cross-level simulation is one of the major benefits of Cooja: the OS level is modelled with Cooja, down to modelling each exact clock cycle at the machine code instruction level (Osterlind et al., 2006).



```
Time: 23:03.858
Speed: ---
```

```
No.     Time       From  To     Data
13613   23:01.409  24    [4 d]  5: 0x02005875 6B
13614   23:01.452  24    [4 d]  68: 0x61DC7FCD AB190000 0000000C C1...
13615   23:01.455  25    [4 d]  5: 0x02007FC8 3E
13616   23:01.471  25    [4 d]  60: 0x61DCE6CD AB010000 0000000C C1...
13617   23:01.473  1     [4 d]  5: 0x0200E680 37
13618   23:01.550  2     3,9    102: 0x61DC9ACD AB030000 0000000C C...
13619   23:01.553  3     [3 d]  5: 0x02009A6B 8E
13620   23:02.392  23    [3 d]  59: 0x61DC59CD AB180000 0000000C C1...
13621   23:02.394  24    [4 d]  5: 0x020059FC 7A
13622   23:02.398  24    [4 d]  68: 0x61DC80CD AB190000 0000000C C1...
13623   23:02.400  25    [4 d]  5: 0x020080B0 31
13624   23:02.416  25    [4 d]  60: 0x61DCE7CD AB010000 0000000C C1...
13625   23:02.418  1     [4 d]  5: 0x0200E709 26
13626   23:03.407  23    [3 d]  59: 0x61DC5ACD AB180000 0000000C C1...
13627   23:03.409  24    [4 d]  5: 0x02005A67 48
13628   23:03.413  24    [4 d]  68: 0
13629   23:03.416  25    [4 d]  5: 0x
13630   23:03.463  25    [4 d]  60: 0
13631   23:03.465  1     [3 d]  5: 0x
13632   23:03.463  5     4,6    102:
13633   23:03.467  4     [3 d]  5: 0x
```

```
Packet data (60 bytes)
61DCE6CD AB010000 0000000C C1190000 0000
C17CD700 3EC30C00 00000000 17E10663 0400
00F0223D 162E9DD1 68656C6C 6F203133 3435
Cross-level packet data (60 bytes)
61DCE6CD AB010000 0000000C C1190000 0000
C17CD700 3EC30C00 00000000 17E10663 0400
00F0223D 162E9DD1 68656C6C 6F203133 3435
```

```
TIMEOUT(500000);
var rad = 3;
var factor = 2.0;
var msgs = Math.ceil(factor *

log.log("Running until messag

timeout_function = function (
    log.log("Script timed out
    log.testOK();
}

YIELD_THEN_WAIT_UNTIL(msg.con
GENERATE_MSG(500, "script tim
YIELD_THEN_WAIT_UNTIL(msg.equ
SCRIPT_TIMEOUT();
```

```
Time       Mote   Message
22:52.326  ID:29  #L 30 1
22:52.380  ID:23  [INFO: App       ] Sending request 1336 to fd00::c30c:0:0:1
22:52.481  ID:1   [INFO: App       ] Received request 'hello 1336' from fd00::c30c:0:0:17
22:53.380  ID:23  [INFO: App       ] Sending request 1337 to fd00::c30c:0:0:1
22:53.485  ID:27  #L 26 1
22:53.520  ID:1   [INFO: App       ] Received request 'hello 1337' from fd00::c30c:0:0:17
22:54.380  ID:23  [INFO: App       ] Sending request 1338 to fd00::c30c:0:0:1
22:54.461  ID:1   [INFO: App       ] Received request 'hello 1338' from fd00::c30c:0:0:17
22:54.745  ID:42  #L 41 1
22:55.380  ID:23  [INFO: App       ] Sending request 1339 to fd00::c30c:0:0:1
22:55.457  ID:1   [INFO: App       ] Received request 'hello 1339' from fd00::c30c:0:0:17
22:56.380  ID:23  [INFO: App       ] Sending request 1340 to fd00::c30c:0:0:1
22:56.449  ID:1   [INFO: App       ] Received request 'hello 1340' from fd00::c30c:0:0:17
22:57.380  ID:23  [INFO: App       ] Sending request 1341 to fd00::c30c:0:0:1
22:57.453  ID:1   [INFO: App       ] Received request 'hello 1341' from fd00::c30c:0:0:17
22:57.561  ID:5   #L 12 1
22:58.380  ID:23  [INFO: App       ] Sending request 1342 to fd00::c30c:0:0:1
22:58.473  ID:1   [INFO: App       ] Received request 'hello 1342' from fd00::c30c:0:0:17
22:59.380  ID:23  [INFO: App       ] Sending request 1343 to fd00::c30c:0:0:1
22:59.481  ID:1   [INFO: App       ] Received request 'hello 1343' from fd00::c30c:0:0:17
23:00.380  ID:23  [INFO: App       ] Sending request 1344 to fd00::c30c:0:0:1
23:00.434  ID:1   [INFO: App       ] Received request 'hello 1344' from fd00::c30c:0:0:17
23:01.380  ID:23  [INFO: App       ] Sending request 1345 to fd00::c30c:0:0:1
```

Figure 2.6: Example of the Cooja simulator. The top-left panel shows the network layout, with the RPL routing DODAG displayed. Below is the message log, showing source broadcast and sink receives, as well as signalling messages interpreted to display the DODAG. Down the right hand side is the simulation control panel, the radio message log (which can parse messages, but is just showing in raw form here), and the simulation code controller.

2.7 RPL

This section offers an overview of the RPL protocol, we would like to particularly emphasise the repair section (Section 2.7.3, pg. 33). RPL is pronounced 'Ripple' and stands for Routing Protocol for LLNs (low-power and lossy networks).

RPL is a type of Distance Vector routing. Distance vector routing schemes are generally based on the Bellman-Ford algorithm for routing, which can be adapted to work in a distributed system (Perkins and Bhagwat, 1996). In a distance vector algorithm, each node repeatedly sends routing information to its neighbours, who may connect or switch to routing through that node, if it offers an improvement. The oldest distance-vector protocol is RIPv1 (Hedrick, 1988), which simply worked off the hop count. Metrics have progressed since, with RPL offering a very flexible approach to choosing the connection quality metric.

RPL is also designed to operate using existing LLN transmission standards, using the low-power IEEE 802.15.4 standards, such as 6LoWPAN and IPv6 (Algahtani et al., 2021). This ensures inter-operability between different types of sensor node.

According to RFC 6550 (Alexander et al. (2012), summarized in Gaddour and Koubaa (2012)), RPL provides five key features:

1. *Auto-configuration:* Because RPL is compliant with IPv6, it can use IPv6's dynamic discovery of neighbours to construct routes.
2. *Self-healing:* The network can adapt to topology changes and failures. It also provides redundant routes to mitigate failures and link degradations.

3. *Loop avoidance:* Maintains the acyclic property of the network: it can detect and recover from their occurrence, which may occur during topology shifts.

4. *Independence and Transparency:* This is independent of the chosen data-link layer protocol, meaning RPL can be implemented over many link types.

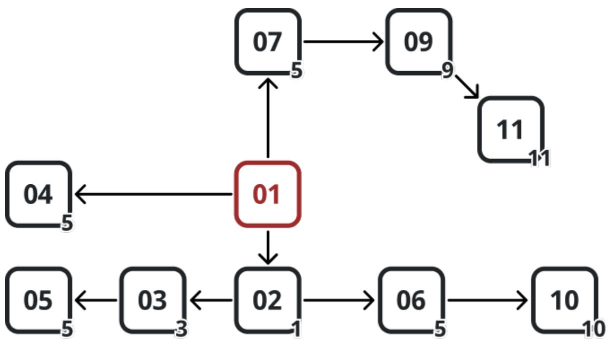
5. *Multiple edge routers:* A RPL network can maintain multiple (intersecting) DODAGs, where each node can use multiple simultaneous networks for different purposes. This can provide load balancing and redundancy.

In this project, for simplicity, only single DODAG networks will be considered, since they only contribute complexity, and are unnecessary to this analysis. Furthermore, the RPL-Lite implementation used in Cooja makes this same restriction, since it is excessively complex for the vast amount of cases (Parasuram, 2016). In fact, RPL's over-complication is a common criticism of the standard – that it has too many options and variants in its standard implementation to ensure interoperability across implementations. Even two of the most common implementations do not operate well when combined (Ko et al., 2011). Despite this over-complication, RPL is being adopted in many IoT OS's, due to its efficiency, reliability and low-overhead.

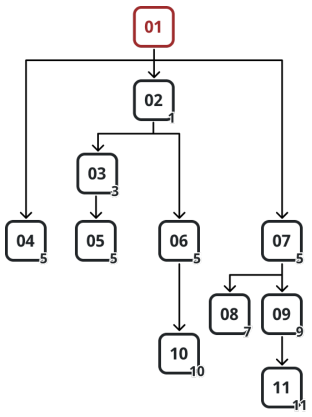
Content-privacy provisions are built into RPL, with the ability to communicate securely with key-based encryption (Alexander et al., 2012), however it does not explicitly provide context-privacy or SLP provisions.

2.7.1 DODAG Establishment

To establish a DODAG, nodes in the network periodically broadcast a message containing basic information on the network (DIO, DODAG Information Object). If this is received by a node not in the network which wants to join this network, it responds with a DAO (Destination Advertisement Object) which registers this node with the network. Optionally, a DAO-ACK packet can be sent back to verify joining. This process is visualized in the sequence diagram in appendix A.1. If an existing node receives a DIO from another node in the network with a better route, it may choose to reroute through a new primary parent. Nodes maintain a list of eligible parents and siblings constructed from their DIOs, so that the nodes have options to route through if the preferred route is broken or degrades.



(a) Example network.



(b) Example routing tree (DODAG).

Figure 2.7: Example of network and DODAG. Nodes are numbered with their IDs, their rank is the adjacent number. Arrows show parents in the DODAG.

The reason discussion of route/link quality so far has been vague is that it entirely depends on the Objective Function and metric chosen. The parent(s) chosen from the candidates are selected by the Objective Function. Each node maintains a *Rank* value to track the quality of the connection to the sink. This is analogous to the path cost in other standard graph routing algorithms. The *Rank* is used to detect cycles and identify parents, children and siblings, as it must strictly increase further from the sink.

The metric could be based on latency or link quality (ETX) (Couto et al., 2005), and can be implemented with OF0 or MRHOF. Objective Function Zero (OF0) is a simple objective function that switches to an improvement no matter how small or temporary the improvement is. It also requires no extension to RPL, and is the simplest to implement (Thubert, 2012). MRHOF is like OF0, but requires a threshold of improvement before switching parents to avoid churn (Gnawali and Levis, 2012) (Richardson and Robles, 2015). As such, MRHOF is considered a more stable and accurate approximation of link quality (Pradeska et al., 2016). Both protocols select a primary parent and a backup parent in case the primary route becomes unavailable.

If the network only requires upward (node-to-sink) communication, it uses RPL in a *non-storing mode*. Each node does not need to store routing tables for nodes below them in the DODAG if no messages are ever sent down the graph, however, the sink must store the routing table of the whole network. This is unlikely to be problematic since the sink is usually a much more capable device. If node-to-node or sink-to-node communication is required, then each node must store routing information for nodes below them. This can have a significant memory footprint per node and graph maintenance can be difficult, since any routing changes have to be fully propagated.

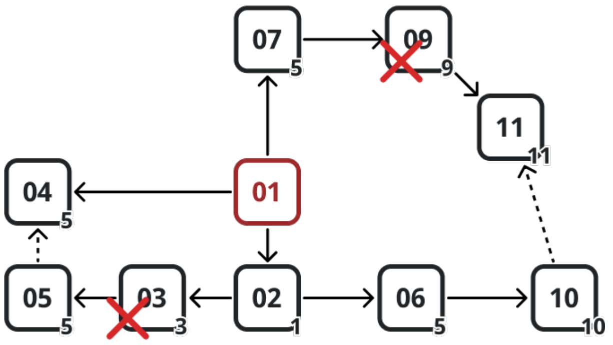
2.7.2 DODAG Maintenance

To ensure pathing knowledge is up to date, DIOs are periodically broadcast according to a Trickle Timer. Essentially, this exponentially decreases the broadcast frequency when no network changes occur. This ensures the network uses minimal messages to maintain itself when it is in a static stable state (Levis et al., 2011). The timer is reset when the local network reaches an "inconsistent" state. This is a situation where a node's routing information needs to be updated, for example a parent becomes unreachable or updates their route. Resetting the timer ensures changes are rapidly disseminated when information changes. The DIO includes a *VersionNumber* field to track configuration versions, used to stop nodes unnecessarily updating when receiving consistent information.

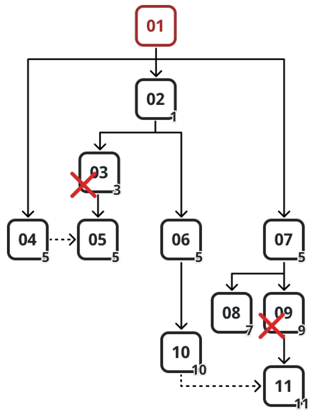
Nodes that wish to join a network broadcast DIS (DODAG Information Solicitation) messages. These simply reset recipient nodes' timers, so a DIO is sent to the new node. This process is also shown in the diagram in appendix A.1.

2.7.3 DODAG Repair

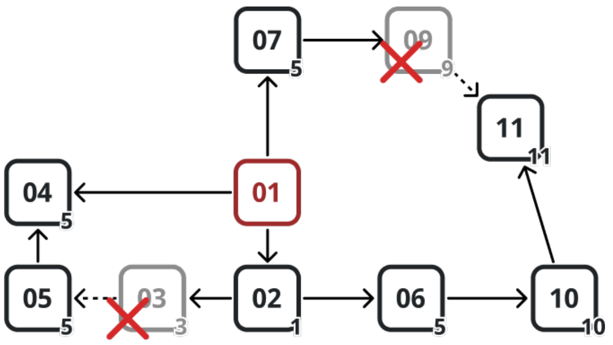
This is an essential part of the proposed SLP approach. A repair may be necessary if a node disappears (e.g. battery failure) or a link degrades or disconnects. The simplest repair method is the global repair. This instructs the sink to increment the *VersionNumber*, forcing a rebuild of the entire DODAG, as if it was newly constructed. This guarantees optimal routing according to the objective function used, however it is an expensive process that is impractical every time a connection temporarily drops or degrades (Korte et al., 2012).



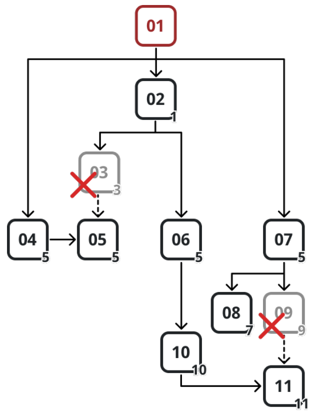
(a) Network before disconnection.



(b) Routing tree (DODAG) before disconnection.



(c) Network after disconnection.



(d) Routing tree (DODAG) after disconnection.

Figure 2.8: Example of DODAG repair, structure from Korte et al. (2012). Nodes are numbered with their IDs, their rank is adjacent. Nodes 03 and 09 are failing. Node 05 has a recorded sibling of Node 04, and Node 11 has a recorded alternative parent of Node 10. When these nodes fail, Node 05 routes through it's existing sibling 04, and Node 11 routes through it's alternative parent Node 10.

The alternative is a form of local repair. Alexander et al. (2012) does not detail the methods of local repair, however Korte et al. (2012) describes the implementation in Contiki. If a node has an alternative parent or sibling in range, it may route through the parent or sibling instead. The sibling list is constructed from received DIOs, just as the eligible parents list is. When constructing the DODAG, each node can record a secondary back-up parent which can be switched to if available. An example is given in Figure 2.8, where Node 05 undergoes sibling repair, and Node 11 undergoes parent repair.

When a node gets disconnected or changes route, it poisons the routing of all children with a DIO, which updates the routing information of nodes below in the DODAG – triggering the whole disconnected subtree to search for new connections. If the parent is still the best option the sub-tree has, this search stops quickly after verifying that fact. Otherwise, if any part of this sub-graph finds (or has) a connection, it broadcasts DIOs outwards until the whole disconnected section is now directed towards the new connection point (Richardson and Robles, 2015).

Chapter 3

Design

In this section, we discuss the implementation of the firmware and SLP methods. Test cases shall be limited to perfect transmission and grid networks (see Section 2.1). The following chapter will consider the implementation of the test and analysis system around these policies.

3.1 Firmware

We shall use the Zolertia Z1 mote in Cooja for testing. Cooja provides simulation of a range of node (mote) architectures, many based on the MSP430 architecture from Texas Instruments[1]. Well-used, well-maintained and well-supported tools exist for the MSP430 range, ensuring good support. Furthermore, due to their popularity, their use increases the relatability of this work. The Zolertia Z1 is chosen over the Wismote and Sky motes (also MSP430 motes). The Z1 provides the largest storage of the alternatives – in particular

[1]Texas Instruments MSP430: `https://www.ti.com/microcontrollers-mcus-processors/microcontrollers/msp430-microcontrollers/overview.html`

the Wismote provides 16kB ROM and Sky mote provides 48kB, whereas the Z1 provides 96kB. Detrimentally to testing, the Sky mote does not provide enough ROM space for the full netstack[2] – notable omissions include downward RPL routing and minimal logging.

Basic driver code for each firmware node is very simple, only being a minor modification of an example[3] using RPL routing over UDP is needed. This example needed modifying to provide firmware for non-sink, non-source nodes. This example provides a `server.c` node, which is the sink in our case, and simply registers a UDP handler that accepts the message, and logs it's receipt. The client periodically attempts to send a message to the RPL network route (the sink) if connected. These 'middle' or relay nodes simply exist to relay messages, with no particular driver code.

Varying properties of the network and routing can be done in the `project-conf.h` file, which is shared by all firmware types, so is perfect for setting protocol properties.

3.2 Disable Policies

To exploit the SLP properties of RPL, we need the network to reconfigure more frequently, but not reconfiguring so frequently as to significantly disrupt network connectivity or amount of communication. Our proposed way of doing this is to disable the radio of nodes, forcing a reconfiguration around that node, an example of a reconfiguration is shown in Figure 3.1.

[2]Contiki-NG Documentation: `https://github.com/contiki-ng/contiki-ng/wiki/Platform-sky`

[3]`https://github.com/contiki-ng/contiki-ng/wiki/Tutorial:-running-a-RPL-network-in-Cooja`

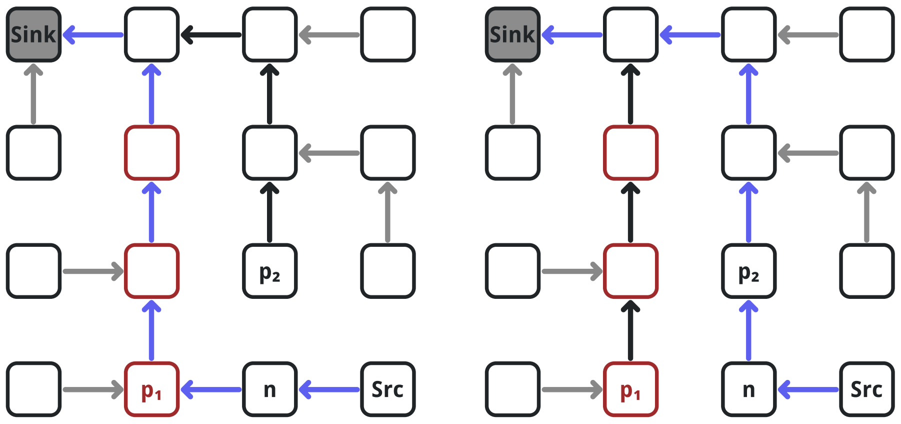
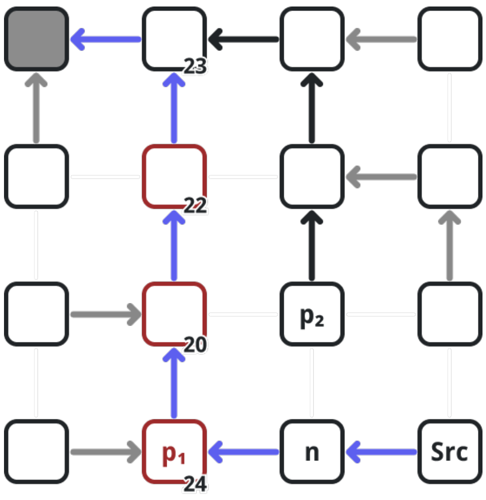


Figure 3.1: Before and after diagram of RPL stranding. If node p_1 switches off, node n switches parents from p_1 to p_2, which would strand the attacker if they were in ancestors of p_1 but not p_2 (those with a red outline).

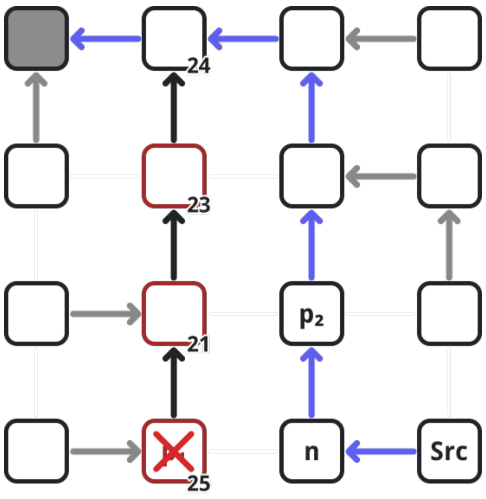
We propose a sequence of *'disable policies'* for these nodes. This policy is called once per data packet that passes through a node. Limiting disabling to only nodes on the route saves power and bandwidth, while ensuring that there is a stable network to reconfigure to.

- **Random:** each relay node has a fixed probability to disable for each message that passes through.
- **Increasing Random:** if random is too erratic, limit this behaviour by slowly increasing the probability of disabling.
- **Counter:** The most regular. Count through-messages up to a certain threshold, then disable the node.
- **Random-Initialization Counter:** If multiple counters' nodes start at the same point, their disabling may cascade, causing a large blocking wall to appear. Therefore, randomize the initial value of the counter so disablings are distributed more evenly.

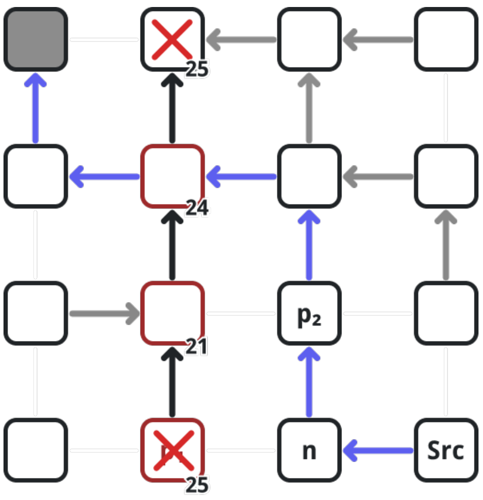
The random-initialization counter is also more accurate to a long-term real-life deployment. When the network has been running for a long time with many routes, the counters are suitably randomized, even if they started at the same value. It is also unreasonable to assume the hunter may start their hunt the moment the first message arrives – a hunter would most likely act against a pre-existing deployment, which would have been running for an indeterminate amount of time.



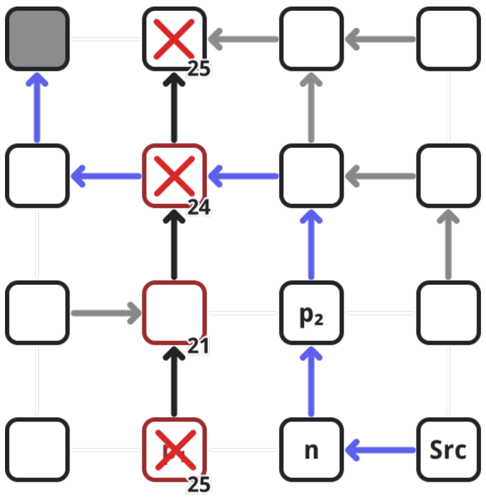
(a) Example as previously, except most of the nodes on this route are near the threshold of 25.



(b) When node p_1 disables (due to passing 25), the network reconfigures through p_2.



(c) However, the last node in the route passes it's threshold and disables.



(d) The network attempts to reconfigure, however this node also reconfigures. This continues until an impenetrable wall is built across the network.

Figure 3.2: Example of the naive Counter policy causing problems, with a threshold of 25. When a single route is well used, all nodes along it will reach the threshold around the same time. This pattern requires breaking up with random initialization.

Chapter 4

Implementation

In this chapter, we design the test harness and analysis methods for this problem. Sections are organized approximately chronologically, generally starting with the basics of testing, followed by initial analysis, then expanding the test harness to work with the designed variations and the Department of Computer Science's Batch Compute system.

Previous works have created similar test frameworks, as necessitated by their projects, but all available versions the author could find were either now unavailable or worked with TOSSIM only. This required the building of this framework from scratch.

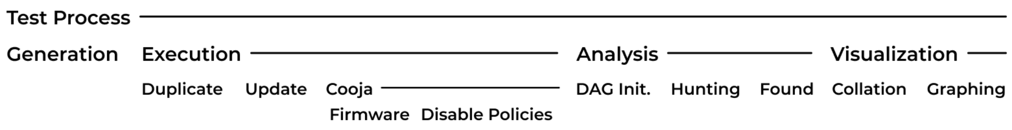


Figure 4.1: A breakdown of the testing and analysis process.

4.0.1 RPL Classic vs RPL-Lite

One early problem that was reached was the decision between RPL Classic and RPL-Lite. RPL Classic is what Contiki-NG (improved fork of the original ContikiOS) called ContikiRPL. As said in Section 2.6.1, RPL Classic is more fully featured, although more complicated. These extras features were judged as unnecessary for a test such as this. As such, RPL-Lite was originally chosen. However, RPL-Lite displayed some strange behaviour in not being able to grow a network with a routing tree of depth greater than three. This behaviour has not been described nor discussed anywhere. Multiple computers, operating systems, and installations displayed the same behaviour. In response, we continued with RPL Classic as an alternative, which worked until it came to implementing the disable policies.

When the disable policy decides that the node needs to disconnect from the network, the node must stop passing traffic. For the RPL routing of a subtree to update in a timely manner, the parent node should poison it's routing (Korte et al., 2012). This involves notifying child nodes that this route is now unavailable, and they should carry out local repair and check for other routes. Without this process, it can take a number of dropped (and unacknowledged) packets before the child node decides to re-route. This would make the network catastrophically fail.

Unfortunately, one oversight in the RPL Classic implementation is the ability to voluntarily leave a network. This resulted in an investigation into both issues, which took a considerable amount of time. While investigating the RPL Classic leaving problem, we inspected the possibility of creating our own implementation, however, this is unrealistic due to the magnitude of changes required. Presumably, this is also why this feature is not implemented in

```
#if RADIO_OFF_SLP == RADOFF_SLP_RAND
    // Random float in [0, 1]
    double r = (double) random_rand() / (double) RAND_MAX;
    LOG_INFO("Random %lf threshold %lf\n", r, OFF_TIMER_PROB);
    if (r < OFF_TIMER_PROB) {
#elif RADIO_OFF_SLP == RADOFF_SLP_CUMUL_RAND
    ...
```

Listing 1: Example implementation of an SLP policy. The section ends with the condition that triggers leaving the network. Contiki's random provides a value between 0 and `RAND_MAX`, which is an implementation specific value.

the first place. Eventually, we discovered that disabling RPL-Lite's DAO-ACK messages and link-quality probing fixed this lack of growth. DAO-ACK messages are sent in response to a node sending a DAO (a request to join the network), more information in Section 2.7.1. Probing is RPL checking link quality before accepting a parent. Especially in such an ideal network (no interference present), probing should not present a problem, and DAO-ACK's should just be an optional final confirmation – Classic RPL does not implement them, and RPL-Lite has the option to disable them. Even now, both these parts do not seem like they should cause such a change. Possibly this is due to a particular bug that only affects a small range of versions of Contiki-NG.

4.0.2 SLP

The choice of SLP technique is dictated by a C pre-processor macro and condition. This is expanded at compile-time, and ensures only one method is compiled into the firmware. Various policies also require different properties and variables to be stored, which each have their own condition. The policies are implemented in the relay (non-source, non-sink) nodes.

There were a couple of considerations during implementation. Firstly, where to implement this condition. Contiki-NG provides various possible callbacks and functions to trigger on packet receipt. However, some (like the UDP receive callback) only trigger for the end of the route, and others have problems with scope, for instance RPL's `link_callback`. This function is part of the standard routing_driver for Contiki. While this seemed like a good candidate, ensuring consistent scope between this section and the driver code was difficult, as both operate in different scopes, and many sections are linked purely at compile time through macros. This is a good decision for Contiki in general, as it ensures irrelevant sections are not compiled and included – essential given the storage constraints of sensor nodes. This design makes implementation here difficult.

Contiki's network stack provides a `netstack_sniffer` and a `netstack_process_ip_callback` interface. Both of these can be registered with the network stack from the main driver code, making them a good choice. Furthermore, the process IP callback provides an easy way to discard packets if needed – allowing more flexibility than a binary radio on-off. This allows us to instruct disabled nodes to cancel only a subset of packets. Due to Contiki's event-based architecture, we can easily implement dropping packets for a period of time, using callback timers – timers that call a passed function on finishing.

A further problem that needed considering was that disabling the radio or dropping all packets means there is no time for the poisoning messages to proliferate. To combat this, the node should not drop RPL control messages for a short period before fully disabling. This is implemented with a second timer.

```
static enum netstack_ip_action ip_input(void) {
    uint8_t proto = 0;
    uipbuf_get_last_header(uip_buf, uip_len, &proto); // Get protocol (is UDP)

    if (!ctimer_expired(&off_timer)
        || (!ctimer_expired(&to_off_timer) && proto != UIP_PROTO_UDP)) {
        return NETSTACK_IP_DROP;  // Cancel packet if timer(s) still going
    }

    if (proto == UIP_PROTO_UDP) {
#if RADIO_OFF_SLP == RADOFF_SLP_RAND
        ...
#elif RADIO_OFF_SLP == RADOFF_SLP_CUMUL_RAND
        ...
#elif RADIO_OFF_SLP == RADOFF_SLP_COUNTER
        ...
#elif RADIO_OFF_SLP == RADOFF_SLP_RANDINIT_COUNTER
        // Initialize to random
        if (count == -1) count = random_rand() % OFF_TIMER_THRESHOLD;
        count++;
        if (count >= OFF_TIMER_THRESHOLD) {
            count = 0;
#else
    #error "ERROR: Unknown RADIO_OFF_SLP"
#endif
            // Condition matched, leaving network
            NETSTACK_ROUTING.leave_network();

            ctimer_set(&to_off_timer, SEND_INTERVAL/2, switch_off, NULL);
        }
    }
    return NETSTACK_IP_PROCESS;
}
```

Listing 2: The incoming packet processor. Most disable policies are not shown for brevity. The first step is to find the protocol of the packet, then decide whether to cancel. If the main off timer is running, cancel everything, if the pre-timer (`to_off_timer`) is running, don't cancel RPL packets. If not cancelling, check the disable policy. For example the Random-Initialized Counter is shown here: as calls to random have to be made inside a function, the `count` is initialized to -1 to indicate it needs to be randomized. For each message, increment the counter, and trigger leave if over the threshold.

4.0.3 Disable Policies

The implementation of each disable policy is simple, random is shown in Listing 1. This simply generates a random number and compares it against the probability given. The policies are implemented with the condition within each policy to allow them to execute actions on disable, for example, reset the count. This can confuse some IDEs, however this is not a problem with sufficient consideration.

The Increasing Random policy is implemented by initializing the probability to a small amount, and multiplicatively increasing it for each message. This probability is reset on disabling. Counter and Random-Initialization Counter are implemented in a similar way, except count is incremented for each message, and for Random-Initialization, the count is randomized. However, since C does not allow calling functions to instantiate variables in the global scope, this is set to a placeholder (`-1`), and randomized on the first call of the policy.

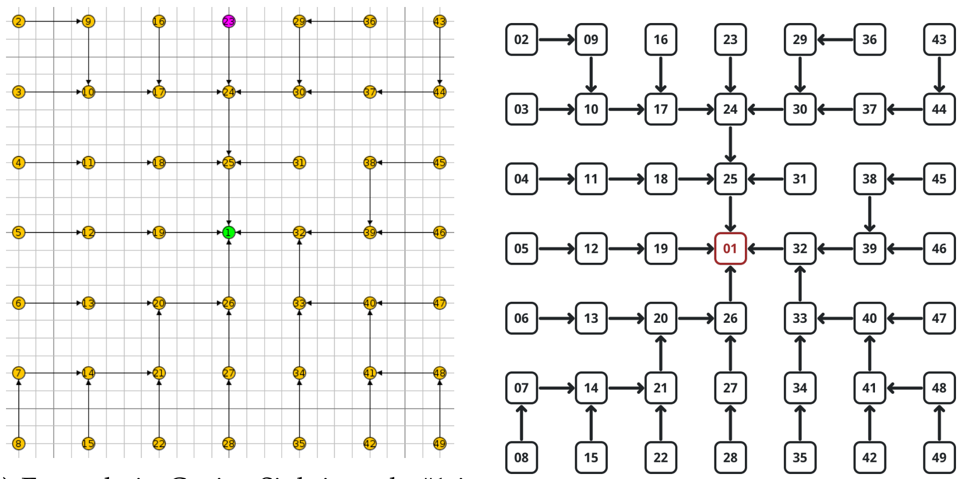
4.1 Test Harness

While implementing the firmware on it's own can be useful, manually testing hundreds of examples is not feasible, therefore we created an automated testing system.

4.1.1 Test Case Generation

The initial consideration is the format of test cases. Cooja allows loading simulations from `.csc` files. These are XML formatted files, which include the node firmware source, node locations and types, control script, and Cooja plug-ins configuration, among other simulation properties: transmission and interference probabilities and the random seed for instance. To generate these test cases, there are three main sections needed: the header, including node firmware files and simulation properties; the footer, including plug-in configuration; and the node list itself. While there is not much documentation on this format, it is fairly self-evident in its construction.

Test cases consist of a square grid, with the sink located at the centre. A random node in this grid is chosen as the source. Grid sizes are chosen to be odd, so the sink can be perfectly centrally located, such that there is no bias. For example, a 15x15 grid has 7 nodes above, below, left, and right of the sink. We will refer to this as 15x15 grid, or a grid of radius 7.



(a) Example in Cooja. Sink is node #1 in green, and the source is #23 in magenta.

(b) Equivalent Graph

Figure 4.2: A generated test case, graphed in Cooja, and outside.

4.1.2 Headless Cooja

For large-scale testing to be practical, the simulator must be able to be run without a GUI – not only is having that many windows open awkward to manage, but GUIs are not supported on remote servers. While Cooja provides a non-GUI mode, the requirements for it are not clearly enumerated anywhere. This supports a simulation as a `.csc` file, but this file must include a test runner script.

A test runner script is a Javascript file that controls the execution of the script. Various functions, macros and variables are provided by Cooja, however these are only explained on the original Contiki wiki[1], not the Contiki-NG wiki[2], which was the primary source of information. These scripts read nodes logging messages, and can end the simulation at a certain point.

This script is intended to run the simulation for a generous amount of time, then the analysis can cut this off earlier if required. Many SLP tests run the simulation for a set amount of time based on the time flooding takes. In our case, this is problematic, as the routing network must be constructed before the hunter can reasonably start. As such, the script waits for the network to be constructed – when the source sends it's first message destined for the sink. The generous amount of time is until a large threshold of messages pass. A standard safety factor is 1.5x. To allow some further flexibility, we run the simulation for twice the maximum time flooding could take on that network size. Again, this can be shortened in analysis as required.

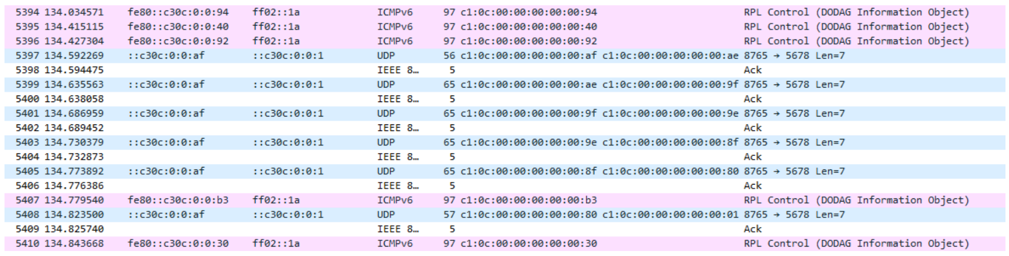
[1] `https://github.com/contiki-os/contiki/wiki/Using-Cooja-Test-Scripts-to-Automate-Simulations`

[2] `https://github.com/contiki-ng/contiki-ng/wiki/`

Radio Logging

A fundamental problem with Cooja's logging is that it does not provide radio logs when headless. In the GUI, the radio logs panel allows exporting to a text file, but this is not ideal for both testing and analysis. Furthermore, the location of Cooja's simulation log is shared between all simulations, and is not changeable with changes to the simulator itself. There exists a plug-in[3] that provides this functionality. It logs to `.pcap` files, which is a common format for network traffic recording. This plug-in needs minor updates to work with Cooja, and removal of separate logs per node – as only the combined log was required. The significant benefit with logging to `.pcap` files is the ability to use existing libraries to aid analysis.

Initially, this plug-in did not load in non-GUI simulations, but loaded in GUI simulations. Ultimately, this was due to the GUI and non-GUI modes using a different method to load plug-ins. This was not clearly discussed in any Contiki resource we could find.



```
5394 134.034571  fe80::c30c:0:0:94  ff02::1a         ICMPv6    97 c1:0c:00:00:00:00:00:94                         RPL Control (DODAG Information Object)
5395 134.415115  fe80::c30c:0:0:40  ff02::1a         ICMPv6    97 c1:0c:00:00:00:00:00:40                         RPL Control (DODAG Information Object)
5396 134.427304  fe80::c30c:0:0:92  ff02::1a         ICMPv6    97 c1:0c:00:00:00:00:00:92                         RPL Control (DODAG Information Object)
5397 134.592269  ::c30c:0:0:af      ::c30c:0:0:1     UDP       56 c1:0c:00:00:00:00:00:af c1:0c:00:00:00:00:00:ae 8765 → 5678 Len=7
5398 134.594475                                      IEEE 8…    5                                                 Ack
5399 134.635563  ::c30c:0:0:af      ::c30c:0:0:1     UDP       65 c1:0c:00:00:00:00:00:ae c1:0c:00:00:00:00:00:9f 8765 → 5678 Len=7
5400 134.638058                                      IEEE 8…    5                                                 Ack
5401 134.686959  ::c30c:0:0:af      ::c30c:0:0:1     UDP       65 c1:0c:00:00:00:00:00:9f c1:0c:00:00:00:00:00:9e 8765 → 5678 Len=7
5402 134.689452                                      IEEE 8…    5                                                 Ack
5403 134.730379  ::c30c:0:0:af      ::c30c:0:0:1     UDP       65 c1:0c:00:00:00:00:00:9e c1:0c:00:00:00:00:00:8f 8765 → 5678 Len=7
5404 134.732873                                      IEEE 8…    5                                                 Ack
5405 134.773892  ::c30c:0:0:af      ::c30c:0:0:1     UDP       65 c1:0c:00:00:00:00:00:8f c1:0c:00:00:00:00:00:80 8765 → 5678 Len=7
5406 134.776386                                      IEEE 8…    5                                                 Ack
5407 134.779540  fe80::c30c:0:0:b3  ff02::1a         ICMPv6    97 c1:0c:00:00:00:00:00:b3                         RPL Control (DODAG Information Object)
5408 134.823500  ::c30c:0:0:af      ::c30c:0:0:1     UDP       57 c1:0c:00:00:00:00:00:80 c1:0c:00:00:00:00:00:01 8765 → 5678 Len=7
5409 134.825740                                      IEEE 8…    5                                                 Ack
5410 134.843668  fe80::c30c:0:0:30  ff02::1a         ICMPv6    97 c1:0c:00:00:00:00:00:30                         RPL Control (DODAG Information Object)
```

Figure 4.3: An example section of a test case capture, shown in Wireshark. This shows some DIOs being periodically broadcast, while a message is sent from the source (`...:af` or Node 175) to the sink (Node 1). Each hop's source and destination is shown in the seventh and eighth columns.

[3]`https://github.com/iPAS/Cooja-RadioLogger` created by Cetic, modified by iPAS.

4.1.3 Test Analysis and Hunter Implementation

Due to the timescales involved in running tests, we implemented the hunter as a separate analysis step. This meant the analysis of the hunter could be quickly updated without requiring re-running all the tests (1 hour instead of around 24 hours). Furthermore, the analyser could be implemented using whichever language or libraries are best, instead being constrained to Cooja scripting or plug-ins.

To analyse test data, Python 3.8 and the package `scapy`[4] was used. `scapy` is a general purpose packet manipulation package. It primarily focuses around forging and sending packets, but it provides an easy framework to parse and decode packets too. Various alternatives exist, such as `pcap-ct`, however `scapy` has a much easier to use interface, and is more widely used and documented – although it's documentation on IEEE 802.15.4 and RPL protocol layers is not comprehensive.

Once the `pcap` file is loaded, the analyser starts it's first phase: waiting for the first data (UDP) packet. This allows RPL to build its network before the hunter starts. Once this packet is sent, the analyser can infer the source – the source is the only node sending UDP packets, so the first sent UDP packet must come from the source. The analysis also provides the ability to make the hunter wait for a delay past this point, although this was not used.

When hunting, the hunter is assumed to be able to decode packets to the extent of identifying basic information like the protocol used. If the analyser detects a frame containing a UDP packet arriving at the hunter location, the hunter back-traces to this hop's frame source. A frame is the equivalent of a packet, but for the link layer in particular. The link/MAC layer handles sending a

[4]`scapy` can be found at `https://scapy.net`

packet on a single hop along it's multi-hop route. This step also counts the number of messages, hunter moves, source sends, and sink receives, such that we can calculate the delivery and capture ratios.

After the hunter reaches the source, the analyser stops processing packets after a small delay. This delay ensures the last packet can finish its journey and be counted as a received message for the delivery ratio, otherwise the delivery ratio would erroneously be less than expected. Analysis is also stopped if the hunter hasn't reached the source in the allotted time or once the capture recording is exhausted.

This recorded data is then saved to a `JSON` file for aggregation by the visualization script. Also included is the length of time it would take the hunter to find the source. This is determined by the position of the source in the network. With flooding, the hunter can take a step towards the source from anywhere on the network with each source message. Therefore, flooding would take the Manhattan/taxi-cab distance between the source and sink, on this grid. This is the sum of the horizontal and vertical distances. However, this cannot be deduced from the capture file alone, but can be found by knowing the network layout and source node ID. One complicating factor is that the node ID numbering is not in a neat order for this. Node 1 is the sink in the centre, node 2 is the top-left corner, then nodes are numbered top-to-bottom left-to-right, skipping over the sink. A better choice would be to number in a spiral out from the sink, therefore the distance to a particular node ID would be the same irrespective of network size.

```
{
    'target': '::c10c:0:0:af',
    'found': True,
    'messages': 3755,
    'moves': 6,
    'broadcasts': 6,
    'recieved': 6,
    'flood_bcs': 6,
    'init_time': 1651083113.357734,
    'start_time': 64.568847,
    'found_time': 114.615707,
    'end_time': 115.615707
}
```

Listing 3: Example JSON file output by the scapy analysis script. It records whether the source was captured/found, how long that took (in various metrics), estimated number of source broadcasts it would take for flooding, and the times the simulation covered. The initialization time records the start time in the capture file. All other times are relative to it.

Test Visualization

Once each packet capture file has been analysed, and the hunter has been simulated, these results need to be collated and visualized. This script is also written in Python 3.8 and uses `matplotlib`[5].

This program finds the output JSON files within a directory, and categorizes them based on their parent directory – so different parameters could be easily be graphed against each other. The data is read out of these files and collated, then the capture and delivery ratios are calculated and graphed. This graphing uses a bar or line chart for plotting the ratios, and a box plot with a violin plot underlaid to plot other properties, such as number of moves to capture. This gives a better visualisation of the distribution of this data, especially in regards to outliers.

[5]`matplotlib` can be found at `https://matplotlib.org`

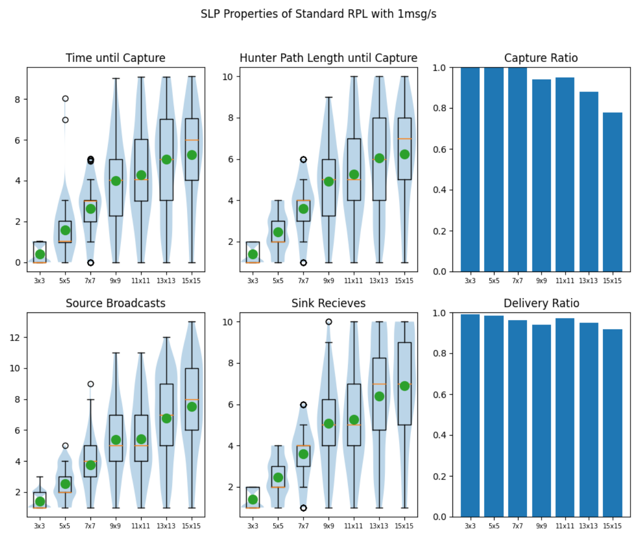


Figure 4.4: Example of graphic produced after analysis and graphing.

4.1.4 Test Running

Now the pipeline to run and analyse individual tests is created, the tests need to be automatically run in parallel. Originally, this was a simple bash script[6], using `find` to discover test cases, then using `parallel`[7] both to call the test function on the test case (like `xargs` could do), and run it in parallel. `parallel` has some extra uses, such as limiting the number of jobs when memory or processor usage is too high, and saving a job log it can resume from. The test function simply created the folder for the job and then called Cooja. To improve run time of each case, the firmware was compiled before running tests, so each test case did not unnecessarily independently compile the firmware.

[6]Bash is a standard choice of Linux terminal, which provides scripting functionality
[7]https://www.gnu.org/software/parallel

Batch Compute Adaptations

This simple system works for cases running on a single computer, where only a single set would be running at a time, but once we migrated to the department's batch compute systems, some further adaptations were required. When only one test set was running, the firmware, the tests and the output could be shared. With this batch compute system, the file storage is shared between nodes, so this means that each set could be overwriting each other.

To address this, the test is given it's own directory, into which the tests and firmware are copied. References to the Contiki installation are updated in the firmware; set properties are updated to match; references to the firmware and script runner are also updated to match; the output directory is updated; and finally the new firmware is built.

This was conceptually fairly straight forward, however, ensuring all references were replaced correctly was a time-consuming process, and allowing different parameters to be changed depending on what was being tested was somewhat awkward at times. Some of these problems stemmed from the use of bash instead of a more user-friendly scripting language, such as Python. However, this script developed slowly as features were needed, and at no particular point was it especially beneficial to rewrite the previous sections in Python. Furthermore, significant parts of this script would have been more complicated in Python – instead of just calling the programs `make` or Cooja, Python would need to use a few libraries to call these from the command line anyway. By default bash does not allow passing variables to other scripts, and `parallel` or `sbatch` (batch compute job dispatch) does not allow input by the standard input pipe. Both can access environment variables from the outer script, so these can be set instead, although this is more involved.

Chapter 5

Results

In this section, we will analyse the quality of source location privacy results provided by standard RPL and our minor modifications.

The capture ratio is the fraction of the time that the hunter successfully reaches/captures the source within the safety period. The safety period is set to be 1.5x a baseline of flooding in the given network. This scales the safety period with the distance between the Sink and Source. The delivery ratio is the fraction of messages sent from the Source that successfully reach the Sink. The aim is to decrease the capture ratio, while keeping the network operable with a high delivery ratio.

5.1 Standard RPL

Experiments in this section are on the RPL-Lite implementation through Cooja. As previously explained, test cases are square grids, with the sink positioned at the centre. Each network size was tested with 100 examples. For network sizes larger than 15x15, we have simulated an equivalent off-centre grid (see Figure

5.1). This is due to the impractical size of the networks, and the direct nature of RPL pathing makes it unlikely these nodes see much use. These results are likely to be less accurate than their full-size counterparts, but they still provide insight into possible results. To create a point of comparison, 15x15 tests are carried out on both a full-size grid and an off-centre grid. The results are presented in Figure 5.2.

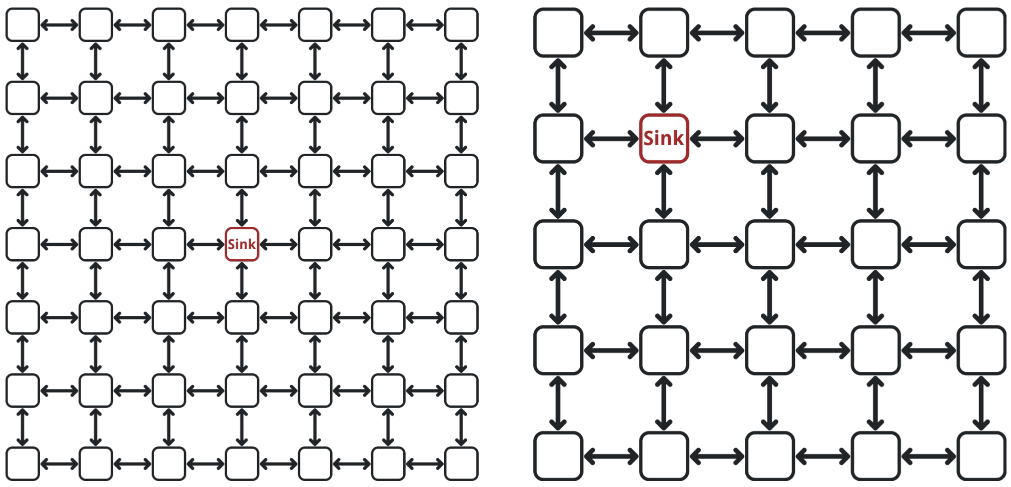
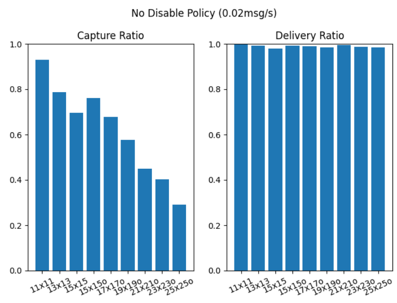
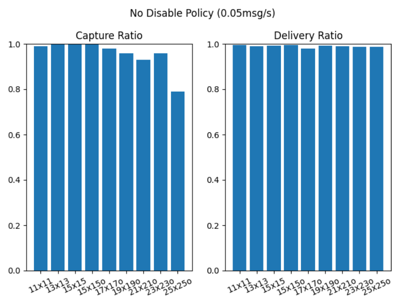


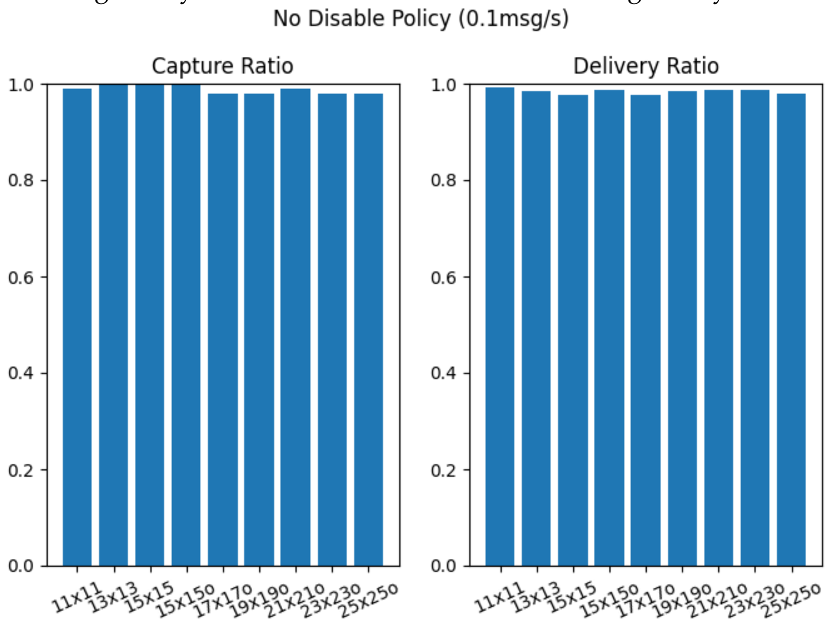
Figure 5.1: Example of a 7x7 full network, and the equivalent 7x7 quarter offset network.



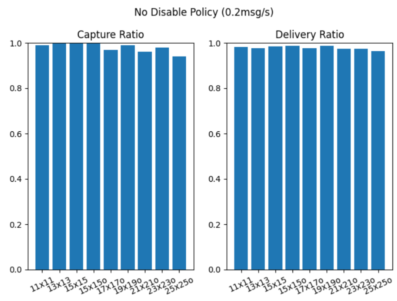
(a) Message every 50 seconds



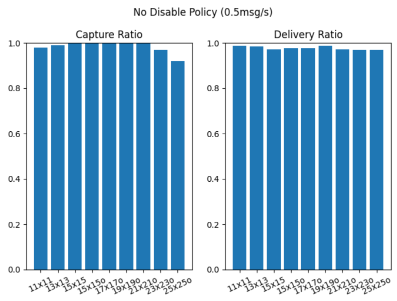
(b) Message every 20 seconds



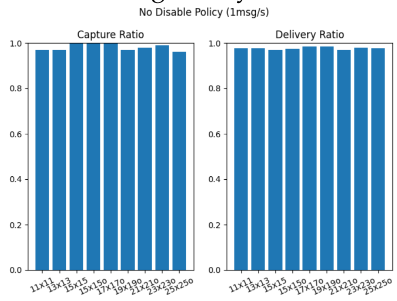
(c) Message every 10 seconds



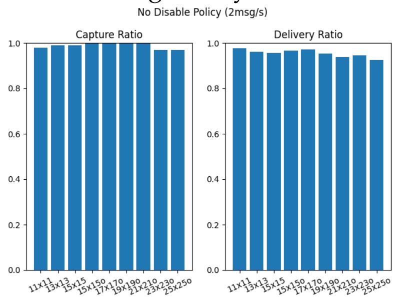
(d) Message every 5 seconds



(e) Message every 2 seconds



(f) Message every second



(g) Message every half a second

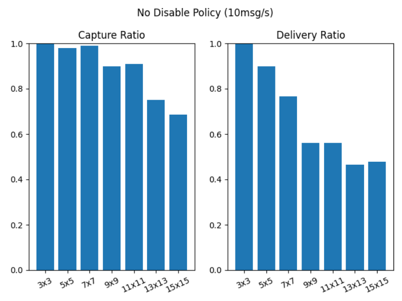
Figure 5.2: Capture Ratio and delivery ratio of a variety of each broadcast speed. The o suffix denotes the simulation was only carried out with a quarter of the grid. The delivery ratio of all is very high. Capture ratio is very high in general across most speeds, with only the longest broadcast frequencies to compare against. These figures are broken down further in the following section.

With standard RPL, some source location privacy was provided as expected, though this is only evident with slow broadcast speeds (see Figures 5.2a and 5.2b). In both these examples, larger networks provided more SLP. Larger networks and longer gaps between broadcasts give the network more chance to reconfigure during and between transmission. This does provide some SLP levels on all network sizes, but significant SLP levels on the largest networks (25x25). These largest sizes are offset networks, so the topology is not perfectly comparable, however the 15x15 to 15x15 offset comparison in Figure 5.2a suggests the actual capture ratio may even be lower for non-offset larger networks.

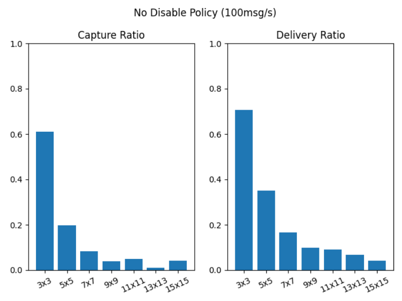
On networks with a broadcast period of 10 seconds down to a message every half a second shows RPL is consistently delivering, however it is not reconfiguring enough to occlude the source, even for larger networks. There is a slight decrease in capture ratio of a couple of percent (up to five percent in Figure 5.2e), although this is fairly insignificant and most likely within the margin of error.

5.1.1 High Broadcast Frequency

Initially, trials indicated a very low level of SLP by RPL, to try to force more reconfigurations, we attempted to send significantly more messages, causing more traffic collisions, therefore varying the link quality further. A couple of examples of this are shown in Figure 5.3. Both these frequencies show the nodes and RPL cannot maintain such a broadcast rate, and the delivery ratio is appropriately low. The capture ratio follows, but the delivery ratio is the cause of this. This is partially a case of the network getting overloaded, but also RPL switching far too fast to maintain a full network.



(a) 10 messages per second



(b) 100 messages per second

Figure 5.3: Graphs of information for very high broadcast frequencies. This shows the nodes and RPL cannot handle too many messages.

5.2 Modifications

While RPL provides some SLP on larger networks, this can be improved. In this section, each proposed disable policy will be evaluated for a 15x15 grid, with a broadcast period of ten seconds (frequency of 0.1msgs/s). We will then explore the best policy on a wider range of networks.

5.2.1 Random Disable Policy

The random disable policy gives each node a fixed chance to disable per message received. As can be seen in figure 5.4, this is far too unreliable, at least for a probability above one percent. The delivery ratio never surpassed 30%. We followed this with exploring lower probabilities, however the same behaviour was observed. From this, we can only conclude that this is an unsuitable disable policy.

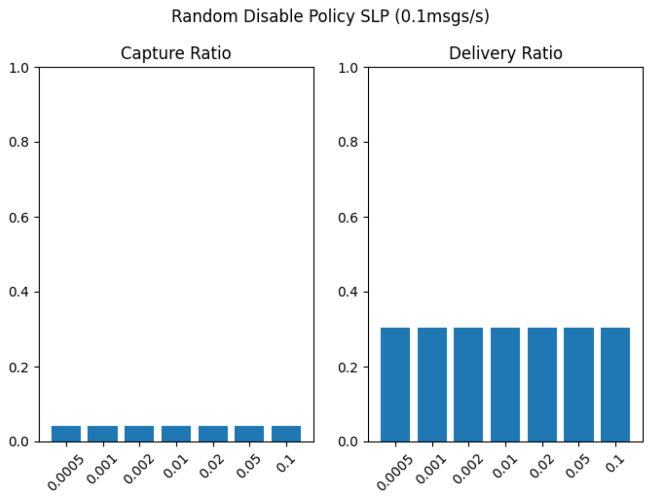


Figure 5.4: Graphs of Capture and Delivery Ratio for Random Disable Policy with different probabilities on 15x15 networks. Delivery ratio is consistently far too low for use. The ratios being consistent is an unsourced issue, although we can still assume random policy is insufficient.

5.2.2 Increasing Random

Increasing random multiplicatively increases the probability of disabling for each message. This should deliver more stability, however, this led to a very similar result as Random: insufficient delivery ratios. To ensure decent coverage, this was tested with each pairing of base probability 1%, 2%, 5% and 10%, and multiplicative factor 101%, 102%, 105% and 110%. None of these cases surpassed the 30% delivery ratio of Random. Just as Random displayed no improvements with even substantially lower probabilities, we expect the same of this policy.

5.2.3 Counter

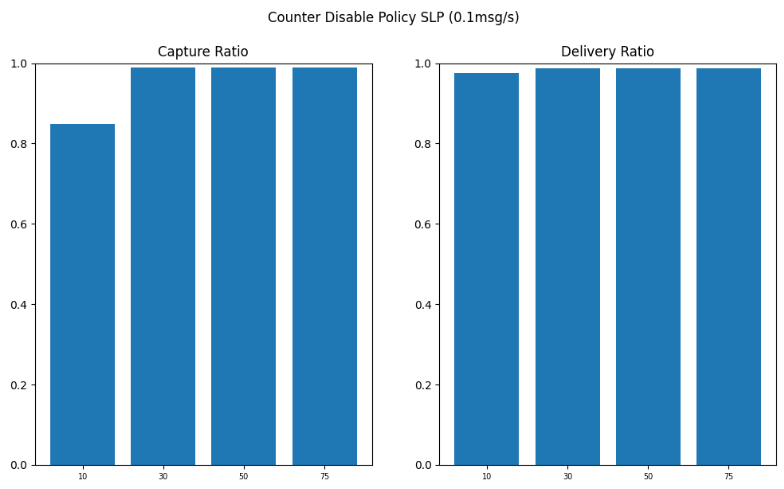


Figure 5.5: Graphs of Capture and Delivery Ratio for Counter Disable Policy with different counter thresholds on 15x15 networks. Shows no effect apart from with a small counter value.

Counter simply disables the node once a set number of messages have passed through. As previously explained, with random initialization, it is rare that a node receives enough messages to pass the counter threshold (Figure 5.5). In around 17% of cases with a counter of 10, the hunter could not capture and the delivery ratio was maintained. This demonstrates that this technique could have value, if this threshold is reached more often and reliably. For a smaller threshold, this may happen too frequently: if every node on the path switches every 3 or 5 messages, the routing cannot keep up, and it may shortly become like the example in Figure 3.2. The safety period was not a long enough period of time for the described behaviour to happen, and if the threshold was low enough, there would be hardly any time between disables.

5.2.4 Randomly-Initialized Counter

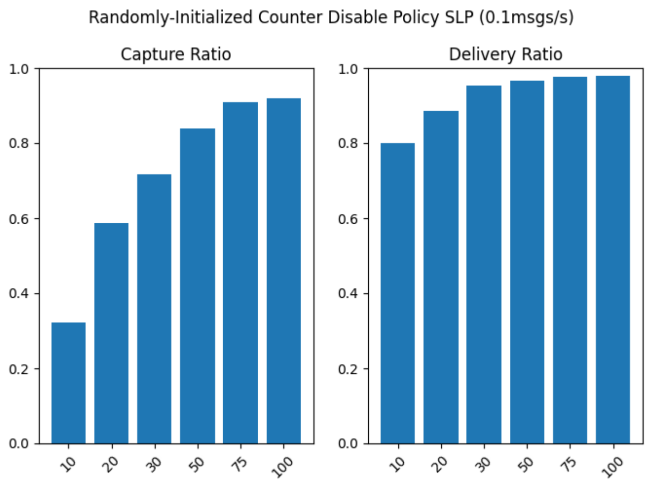


Figure 5.6: Graphs of Capture and Delivery Ratio for Randomly Initialized Counter Disable Policy with different counter thresholds on 15x15 networks.

The counter is randomly initialized to between 0 and the threshold. The stability of this method, and the fact that it could have an immediate effect is beneficial. On this 15x15 grid, a threshold of 30 reduces the capture ratio from nearly 100% to 71%, while only lowering the delivery ratio to 95% (Figure 5.6).

This technique displays a scale of capture ratio to delivery ratio trade-offs, such as a threshold of 100 providing 98% delivery ratio and a 92% capture ratio.

5.3 Expanding Results

Given the success of the randomly-initialized counter policy, we further tested this policy with a range of networks and broadcast speeds. For sizes larger than 15x15, we have resorted to simulating an equivalent off-centre grid. This is due to the impractical size of the networks. These results are likely to be less accurate than their full-size counterparts, but they still provide insight into possible results.

These tests were executed for each broadcast period 50, 10, 5, and 1 seconds, threshold 30, 50, 70 and 90 messages, and for each network size up to (offset) 25x25. Unfortunately, due to the considerable length of time these tests took (over a week) and storage space being exhausted causing results to not be stored, results for these tests are partly missing and the remainder is unreliable. As such, these final results cannot be included in this report.

5.4 Lack of Testbed Experiments

Our test results are from the COOJA simulator, not real hardware. Testing on real testbeds was investigated, with testbeds such as FIT/IoT-Lab[1] or Flocklab[2]. However these systems both had poor topology for SLP testing(Gu et al., 2019), and were not suitably equipped to record network traffic as required by the hunter simulation.

[1]`https://iot-lab.github.io/`

[2]`https://tec.ee.ethz.ch/research/networked-embedded-systems/flocklab.html`

Chapter 6

Evaluation

6.1 Project Evaluation and Timeline

The initial objectives of this project were primarily met, in spite of issues faced during development, especially towards the start of the project.

While there were some issues tackled during the development of this project, the objectives were primarily met. We analysed the SLP properties of RPL and improved upon them with minor changes. A significant reduction of the capture ratio is achieved for RPL (down to 73% as previously stated). While these changes did not have an effect as great as many of the other techniques previously described, it is a much simpler method and is meant to augment RPL, not replace it.

This analysis and the test system required were successfully built and functioned reliably during testing. This project initially aimed to investigate further parameters of RPL that could improve SLP, but these were only partially explored due to time constraints.

The primary issue with this project was the very long duration testing with COOJA takes. Running a set of several hundred examples took take around 24 hours, and in addition data analysis took another few hours. This led to a slow development cycle, and a reduced pace of available tests/experiments. This perhaps could have been mitigated by designing the full test architecture before implementation. However, this waterfall style approach would have led to similar problems and oversights during development.

The informal iterative agile approach that was taken meant that the system was highly modular and could be adapted to requirements of the project, as they developed. This ensured each module could be inspected for reliability individually rather than all at once. However the interface of some systems is unnecessarily involved, for instance the analysis script must be called separately on each example (using bash utilities) instead of automatically doing that itself. This decision was meant to improve total analysis speed through parallelism, although this was a primarily memory-constrained problem, leading to reduced actual gains.

One of the author's personal goals of this project was to explore a new knowledge domain, which was certainly achieved by exploring the SLP problem and Contiki.

This project was intentionally chosen to be a departure from the author's previous knowledge and experience, resulting in an increased breadth of knowledge by the end of the project. This was partly due to a change in supervisor shortly before the year began, and a desire to fit the project subject with the supervisor, ensuring that the supervisor would be able to provide insightful feedback throughout the project's duration.

Due to a late change of supervisor, this decision stretched into the start of term one. Combined with the amount of required research and the term-one-heavy module choice of the author, progress in term one was slower than expected. While these factors were understood in the original timeline, their effects were underestimated. Furthermore, as knowledge and requirements developed, the project's focus shifted from being about RPL and SLP to becoming more focused on developing the required test suite.

Over term two, progress was rapid – as a consequence of having less other commitments and being equipped with a rounded understanding of the project area. Two of the issues described previously drew significant time away from development, although otherwise development continued rapidly and at a steady pace. As previously stated, the amount of time testing took limited the results that could be obtained after development had progressed sufficiently for results to be produced. This was exacerbated by additional issues faced when testing on the department's batch compute system, meaning that a considerable number of tests were originally ran on my desktop computer, further limiting the available resources.

6.2 Project Management

Despite the initial issues faced, a successful result was achieved in no small part due to the project's management, despite the initial issues facing this project. A fairly loose iterative agile approach was chosen. This ensured that the project had the flexibility to grow and be steered as the author gained more knowledge and experience. Furthermore, as this was a primarily researching and investigation problem, progress had to be adaptable in the face of unexpected results.

Throughout this project, code was managed with git, and stored on GitHub.

A Notion[1] Kanban board was originally used to organize project progress, especially when keeping track of the multiple ongoing test experiments at any time. Sources and notes on these sources were managed in the citation manager Zotero[2], ensuring easy organization of many sources.

While this project produced a large amount of data, it was carefully managed, ensuring clear directory organization at all times, so test data would not be confused.

Approximately weekly meetings were held with the project supervisor. During term one these were more concentrated earlier, as this was when supervisor involvement was most required. For the full duration of term two, these regular supervisor meetings were maintained, which were helpful to ensuring progress was made consistently in the right areas.

[1]`https://www.notion.so`
[2]`https://www.zotero.org`

Chapter 7

Conclusion

Analysis of the source location privacy properties of RPL has not been previously carried out. In this project, we analysed the SLP properties of RPL and improved upon them by making changes. These changes had a substantial effect on the source privacy of RPL (down to 73%). Other techniques have seen further reductions, however they incurred further costs for complexity and energy-use. The scope of this project was to investigate RPL's suitability for SLP, with smaller modifications. Furthermore, a new SLP test suite has been created for COOJA, allowing higher quality simulations than previous test suites that used the lower-accuracy TOSSIM. This project has also demonstrated the resilience of the RPL protocol – even with a large number of disruptions, it can still efficiently deliver the vast majority of packets.

7.1 Future Work

Due to this work, we now have a much greater understanding of source location privacy properties of the routing protocol RPL. As a result of this, future work could involve implementing and studying more significant SLP methods to further lower the capture ratio of RPL. We would suggest a phantom routing based technique, such as phantom walkabouts, since these techniques complement RPL's single-path routing to a greater extent than fake sources methods. Further work could also involve testing these properties in real-life situations. This was initially investigated, however all current IoT testbeds are unsuitable for SLP testing. This analysis could also include further consideration of the energy costs of these methods.

Bibliography

Akyildiz, I. F. & Su, W. & Sankarasubramaniam, Y. & Cayirci, E. Wireless sensor networks: A survey. *Computer Networks*, 38(4):393–422, March 2002. ISSN 1389-1286. doi:10.1016/S1389-1286(01)00302-4.

Alexander, Roger & Brandt, Anders & Vasseur, J. P. & Hui, Jonathan & Pister, Kris & Thubert, Pascal & Levis, P. & Struik, Rene & Kelsey, Richard & Winter, Tim. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. Technical Report RFC 6550, Internet Engineering Task Force, March 2012.

Algahtani, Faya & Tryfonas, Theo & Oikonomou, George. A Reference Implemenation for RPL Attacks Using Contiki-NG and COOJA. In *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 280–286, Pafos, Cyprus, July 2021. IEEE. ISBN 978-1-66543-929-9. doi:10.1109/DCOSS52077.2021.00053.

Arampatzis, Th. & Lygeros, J. & Manesis, S. A Survey of Applications of Wireless Sensors and Wireless Sensor Networks. In *Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation Intelligent Control, 2005.*, pages 719–724, June 2005. doi:10.1109/.2005.1467103.

Badescu, Alina-Mihaela & Cotofana, Lucian. A wireless sensor network to monitor and protect tigers in the wild. *Ecological Indicators*, 57:447–451, October 2015. ISSN 1470-160X. doi:10.1016/j.ecolind.2015.05.022.

Bauer, P. & Sichitiu, M. & Istepanian, R. & Premaratne, K. The mobile patient: Wireless distributed sensor networks for patient monitoring and care. In *Proceedings 2000 IEEE EMBS International Conference on Information Technology Applications in Biomedicine.*, pages 17–21, November 2000. doi:10.1109/ITAB.2000.892341.

Benenson, Z. & Cholewinski, P. M. & Freiling, F. C. Vulnerabilities and Attacks in Wireless Sensor Networks. *Wireless Sensors Networks Security*, 1:22–43, 2008.

Bormann, Carsten & Ersue, Mehmet & Keränen, Ari. Terminology for Constrained-Node Networks. Technical Report RFC 7228, Internet Engineering Task Force, May 2014.

Bradbury, Matthew & Jhumka, Arshad. A Near-Optimal Source Location Privacy Scheme for Wireless Sensor Networks. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 409–416, August 2017. doi:10.1109/Trustcom/BigDataSE/ICESS.2017.265.

Bradbury, Matthew & Jhumka, Arshad & Leeke, Matthew. Hybrid online protocols for source location privacy in wireless sensor networks. *Journal of Parallel and Distributed Computing*, 115:67–81, May 2018. ISSN 0743-7315. doi:10.1016/j.jpdc.2018.01.006.

Braginsky, David & Estrin, Deborah. Rumor routing algorthim for sensor networks. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, WSNA '02, pages 22–31, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 1-58113-589-0. doi:10.1145/570738.570742.

Conti, Mauro & Willemsen, Jeroen & Crispo, Bruno. Providing Source Location Privacy in Wireless Sensor Networks: A Survey. *IEEE Communications Surveys Tutorials*, 15(3):1238–1280, 2013. ISSN 1553-877X. doi:10.1109/SURV.2013.011413.00118.

Couto, Douglas S. J. De & Aguayo, Daniel & Bicket, John & Morris, Robert. A High-Throughput Path Metric for Multi-Hop Wireless Routing. *Wireless Networks*, 11(4): 419–434, July 2005. ISSN 1572-8196. doi:10.1007/s11276-005-1766-z.

Dyo, Vladimir & Ellwood, Stephen A. & Macdonald, David W. & Markham, Andrew & Trigoni, Niki & Wohlers, Ricklef & Mascolo, Cecilia & Pásztor, Bence & Scellato, Salvatore & Yousef, Kharsim. WILDSENSING: Design and deployment of a sustainable sensor network for wildlife monitoring. *ACM Transactions on Sensor Networks*, 8(4):29:1–29:33, September 2012. ISSN 1550-4859. doi:10.1145/2240116.2240118.

Gaddour, Olfa & Koubaa, Anis. RPL in a nutshell: A survey. *Computer Networks*, 56 (14):3163–3178, September 2012. ISSN 1389-1286. doi:10.1016/j.comnet.2012.06.016.

Gnawali, Omprakash & Levis, P. The Minimum Rank with Hysteresis Objective Function. Technical Report RFC 6719, Internet Engineering Task Force, September 2012.

Gu, Chen & Bradbury, Matthew & Jhumka, Arshad & Leeke, Matthew. Assessing the Performance of Phantom Routing on Source Location Privacy in Wireless Sensor Networks. In *2015 IEEE 21st Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 99–108, November 2015. doi:10.1109/PRDC.2015.9.

Gu, Chen & Bradbury, Matthew & Jhumka, Arshad. Phantom walkabouts in wireless sensor networks. In *Proceedings of the Symposium on Applied Computing*, pages 609–616. Association for Computing Machinery, April 2017. ISBN 978-1-4503-4486-9. doi:10.1145/3019612.3019732.

Gu, Chen & Bradbury, Matthew S. & Jhumka, Arshad. Phantom walkabouts : A customisable source location privacy aware routing protocol for wireless sensor networks. *Concurrency and Computation: Practice and Experience*, 31(20):e5304, October 2019. ISSN 1532-0626. doi:10.1002/cpe.5304.

Hahm, Oliver & Baccelli, Emmanuel & Petersen, Hauke & Tsiftes, Nicolas. Operating Systems for Low-End Devices in the Internet of Things: A Survey. *IEEE Internet of Things Journal*, 3(5):720–734, October 2016. ISSN 2327-4662. doi:10.1109/JIOT.2015.2505901.

Hakim, H. & Renouf, R. & Enderle, J. Passive RFID Asset Monitoring System in Hospital Environments. In *Proceedings of the IEEE 32nd Annual Northeast Bioengineering Conference*, pages 217–218, April 2006. doi:10.1109/NEBC.2006.1629830.

Hedrick, C. Routing Information Protocol. Technical Report RFC 1058, Internet Engineering Task Force, June 1988.

Holmes, I. A. Making maintenance invisible. *AB Journal*, pages 49–53, September 2004.

Javed, Farhana & Afzal, Muhamamd Khalil & Sharif, Muhammad & Kim, Byung-Seo. Internet of Things (IoT) Operating Systems Support, Networking Technologies, Applications, and Challenges: A Comparative Review. *IEEE Communications Surveys Tutorials*, 20(3):2062–2100, 2018. ISSN 1553-877X. doi:10.1109/COMST.2018.2817685.

Jhumka, Arshad & Leeke, Matthew & Shrestha, Sambid. On the Use of Fake Sources for Source Location Privacy: Trade-Offs Between Energy and Privacy. *The Computer Journal*, 54(6):860–874, June 2011. ISSN 0010-4620. doi:10.1093/comjnl/bxr010.

Jhumka, Arshad & Bradbury, Matthew & Leeke, Matthew. Fake source-based source location privacy in wireless sensor networks. *Concurrency and Computation: Practice and Experience*, 27(12):2999–3020, 2015. ISSN 1532-0634. doi:10.1002/cpe.3242.

Kamat, Pandurang & Zhang, Yanyong & Trappe, Wade & Ozturk, Celal. Enhancing source-location privacy in sensor network routing. In *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 599–608, 2005. doi:10.1109/ICDCS.2005.31.

Ko, Jeonggil & Eriksson, Joakim & Tsiftes, Nicolas & Dawson-Haggerty, Stephen & Terzis, Andreas & Dunkels, Adam & Culler, David. ContikiRPL and TinyRPL: Happy together. In *Workshop on Extending the Internet to Low Power and Lossy Networks*, volume 570, 2011.

Korte, Kevin Dominik & Sehgal, Anuj & Schönwälder, Jürgen. A Study of the RPL Repair Process Using ContikiRPL. In *Dependable Networks and Services*, volume 7279, pages 50–61. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-30632-7 978-3-642-30633-4. doi:10.1007/978-3-642-30633-4_8.

Levis, P. & Clausen, Thomas H. & Gnawali, Omprakash & Hui, Jonathan & Ko, JeongGil. The Trickle Algorithm. Technical Report RFC 6206, Internet Engineering Task Force, March 2011.

Levis, Philip & Lee, Nelson & Welsh, Matt & Culler, David. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pages 126–137, New York, NY, USA, November 2003. Association for Computing Machinery. ISBN 978-1-58113-707-1. doi:10.1145/958491.958506.

Mainwaring, Alan & Culler, David & Polastre, Joseph & Szewczyk, Robert & Anderson, John. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, WSNA '02, pages 88–97, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 1-58113-589-0. doi:10.1145/570738.570751.

Mehta, Kiran & Liu, Donggang & Wright, Matthew. Location Privacy in Sensor Networks Against a Global Eavesdropper. In *2007 IEEE International Conference on Network Protocols*, pages 314–323, October 2007. doi:10.1109/ICNP.2007.4375862.

Milenković, Aleksandar & Otto, Chris & Jovanov, Emil. Wireless sensor networks for personal health monitoring: Issues and an implementation. *Computer Communications*, 29(13-14):2521–2533, August 2006. ISSN 0140-3664. doi:10.1016/j.comcom.2006.02.011.

Munz, Gerhard & Carle, Georg. Distributed Network Analysis Using TOPAS and Wireshark. In *NOMS Workshops 2008 - IEEE Network Operations and Management Symposium Workshops*, pages 161–164, April 2008. doi:10.1109/NOMSW.2007.27.

Musaddiq, Arslan & Zikria, Yousaf Bin & Hahm, Oliver & Yu, Heejung & Bashir, Ali Kashif & Kim, Sung Won. A Survey on Resource Management in IoT Operating Systems. *IEEE Access*, 6:8459–8482, 2018. ISSN 2169-3536. doi:10.1109/ACCESS.2018.2808324.

Nassiri, Ali & Razzaque, M. A. & Abdullah, Abdul Hanan. Isolated Adversary Zone for source location privacy in Wireless Sensor Networks. In *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 108–113, September 2016. doi:10.1109/IWCMC.2016.7577042.

Oikonomou, George & Duquennoy, Simon & Elsts, Atis & Eriksson, Joakim & Tanaka, Yasuyuki & Tsiftes, Nicolas. The Contiki-NG open source operating system for next generation IoT devices. *SoftwareX*, 18:101089, 2022. ISSN 2352-7110. doi:10.1016/j.softx.2022.101089.

Osterlind, Fredrik & Dunkels, Adam & Eriksson, Joakim & Finne, Niclas & Voigt, Thiemo. Cross-Level Sensor Network Simulation with COOJA. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, pages 641–648, November 2006. doi:10.1109/LCN.2006.322172.

Ouyang, Yi & Le, Xhengyi & Chen, Guanling & Ford, J. & Makedon, F. Entrapping adversaries for source protection in sensor networks. In *2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks(WoWMoM'06)*, pages 10 pp.–34, June 2006. doi:10.1109/WOWMOM.2006.40.

Ozturk, Celal & Zhang, Yanyong & Trappe, Wade. Source-location privacy in energy-constrained sensor network routing. In *Proceedings of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks - SASN '04*, page 88, Washington DC, USA, 2004. ACM Press. ISBN 978-1-58113-972-3. doi:10.1145/1029102.1029117.

Palmieri, Paolo. Preserving Context Privacy in Distributed Hash Table Wireless Sensor Networks. In Qing, Sihan & Okamoto, Eiji & Kim, Kwangjo & Liu, Dongmei, editors, *Information and Communications Security*, Lecture Notes in Computer Science, pages 436–444, Cham, 2016. Springer International Publishing. ISBN 978-3-319-29814-6. doi:10.1007/978-3-319-29814-6_37.

Parasuram, Aishwarya. *An Analysis of the RPL Routing Standard for Low Power and Lossy Networks*. PhD thesis, EECS Department, University of California, Berkeley, May 2016. URL `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-106.html`.

Perkins, Charles E. & Bhagwat, Pravin. Routing Over Multi-Hop Wireless Network of Mobile Computers. In Imielinski, Tomasz & Korth, Henry F., editors, *Mobile Computing*, The Kluwer International Series in Engineering and Computer Science, pages 183–205. Springer US, Boston, MA, 1996. ISBN 978-0-585-29603-6. doi:10.1007/978-0-585-29603-6_6.

Perrig, Adrian & Szewczyk, Robert & Tygar, J.D. & Wen, Victor & Culler, David E. SPINS: Security Protocols for Sensor Networks. *Wireless Networks*, 8(5):521–534, September 2002. ISSN 1572-8196. doi:10.1023/A:1016598314198.

Pradeska, Nurrahmat & Widyawan, & Najib, Warsun & Kusumawardani, Sri Suning. Performance analysis of objective function MRHOF and OF0 in routing protocol RPL IPV6 over low power wireless personal area networks (6LoWPAN). In *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 1–6, October 2016. doi:10.1109/ICITEED.2016.7863270.

Raj, Mayank & Li, Na & Liu, Donggang & Wright, Matthew & Das, Sajal K. Using data mules to preserve source location privacy in Wireless Sensor Networks. *Pervasive and Mobile Computing*, 11:244–260, April 2014. ISSN 1574-1192. doi:10.1016/j.pmcj.2012.10.002.

Richardson, Michael & Robles, Ines. RPL- Routing over Low Power and Lossy Networks, November 2015. URL `https://www.ietf.org/proceedings/94/slides/slides-94-rtgarea-2.pdf`.

Roy, Pradeep Kumar & Singh, Jyoti Prakash & Kumar, Prabhat & Singh, M. P. Source Location Privacy Using Fake Source and Phantom Routing (FSAPR) Technique in Wireless Sensor Networks. *Procedia Computer Science*, 57:936–941, January 2015. ISSN 1877-0509. doi:10.1016/j.procs.2015.07.486.

Sanchez, Victor. The Network Link Layer: Ad hoc networks. page 21, 2020. URL `https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs345/lecture_8_networklayer_adhocnet_part1.pdf`.

Shaikh, Riaz Ahmed & Jameel, Hassan & D'Auriol, Brian J. & Lee, Heejo & Lee, Sungyoung & Song, Young-Jae. Achieving Network Level Privacy in Wireless Sensor Networks. *Sensors*, 10(3):1447–1472, March 2010. ISSN 1424-8220. doi:10.3390/s100301447.

Stehlık, Martin. Comparison of Simulators for Wireless Sensor Networks. Master's thesis, Masaryk University, 2011. URL `https://is.muni.cz/th/u6wrk/master_thesis.pdf`.

Tange, Ole. GNU Parallel 2018. April 2018. doi:10.5281/zenodo.1146014.

Thubert, Pascal. Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL). Technical Report RFC 6552, Internet Engineering Task Force, March 2012.

Wang, W.-P. & Chen, L. & Wang, J.-X. A Source-Location Privacy Protocol in WSN Based on Locational Angle. In *2008 IEEE International Conference on Communications*, pages 1630–1634, May 2008. doi:10.1109/ICC.2008.315.

WWF, . The Connected Conservation Project in Kafue, August 2020a. URL `https://www.wwfzm.panda.org/the_connected_conservation_project_in_kafue/`.

WWF, . Wildlife Crime Technology Project, 2020b. URL `https://www.worldwildlife.org/projects/wildlife-crime-technology-project`.

Xi, Yong & Schwiebert, L. & Shi, Weisong. Preserving source location privacy in monitoring-based wireless sensor networks. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, pages 8 pp.–, April 2006. doi:10.1109/IPDPS.2006.1639682.

Yang, Yi & Shao, Min & Zhu, Sencun & Cao, Guohong. Towards statistically strong source anonymity for sensor networks. *ACM Transactions on Sensor Networks*, 9(3): 34:1–34:23, June 2013. ISSN 1550-4859. doi:10.1145/2480730.2480737.

Yao, Lin & Kang, Lin & Deng, Fangyu & Deng, Jing & Wu, Guowei. Protecting source–location privacy based on multirings in wireless sensor networks. *Concurrency and Computation: Practice and Experience*, 27(15):3863–3876, 2015. ISSN 1532-0634. doi:10.1002/cpe.3075.

Zhang, Liang. A self-adjusting directed random walk approach for enhancing source-location privacy in sensor network routing. In *Proceeding of the 2006 International Conference on Communications and Mobile Computing - IWCMC '06*, page 33, Vancouver, British Columbia, Canada, 2006. ACM Press. ISBN 978-1-59593-306-5. doi:10.1145/1143549.1143558.

Zikria, Yousaf Bin & Kim, Sung Won & Hahm, Oliver & Afzal, Muhammad Khalil & Aalsalem, Mohammed Y. Internet of Things (IoT) Operating Systems Management: Opportunities, Challenges, and Solution. *Sensors*, 19(8):1793, January 2019. ISSN 1424-8220. doi:10.3390/s19081793.

Zolertia, . Zolertia Z1 Datasheet, March 2010. URL `http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf`.

Appendix A

Diagrams

A.1 RPL Joining Sequence Diagram

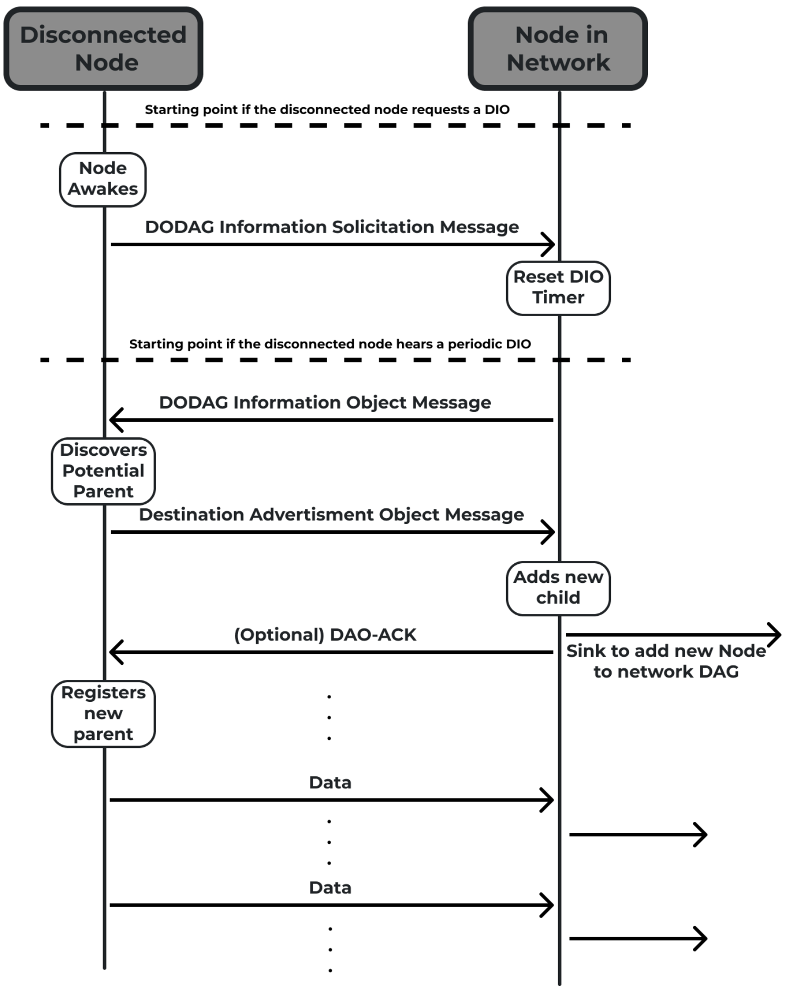

Disconnected Node
Node in Network
Starting point if the disconnected node requests a DIO
Node Awakes
DODAG Information Solicitation Message
Reset DIO Timer
Starting point if the disconnected node hears a periodic DIO
DODAG Information Object Message
Discovers Potential Parent
Destination Advertisment Object Message
Adds new child
(Optional) DAO-ACK
Sink to add new Node to network DAG
Registers new parent
Data
Data

Figure A.1: A slightly simplified Sequence Diagram of a node joining an RPL network.