Artifact Evaluation of the Paper a Reference Implementation for RPL Attacks Using Contiki-NG and COOJA

Alexander Yen
University of Adelaide
Adelaide, Australia
alexander.yen@student.adelaide.edu.au

ABSTRACT

The widespread adoption of Low power and Lossy Networks is facilitated by the growing interest in the Internet of Things (LLN), RPL-based IoT networks are vulnerable to some variety of attacks. RPL provides down and upward transmission that can lead to assaults, low compute and message costs, and multi-topology routing. The majority of known RPL attacks are implemented using Contiki-NG in this paper. Furthermore, we created a framework in COOJA to make it easier to simulate hybrid RPL assaults with various options for length and intensity.

KEYWORDS

RPL, Security, Internet of Things, Contiki-NG, COOJA

1 INTRODUCTION

Internet of Things (IoT) is a very popular technologies nowadays which allows all the technologies to operate together as a single system. The "things" connected in an IoT ecosystem could be anything, including goods, physical or virtual things, and people. It is important to secure the communication between these interconnected things. If not, others can abuse the information gathered from the IoT environment. Therefore, better routing mechanisms are required for IoT to ensure secure communication against various IoT attacks. Some protocols are used for routing in IoT.

One of the protocols used in IoT-based systems is the IPv6 Routing Protocol for Low Power and Lossy Networks (RPL). Because ubiquitous objects require a lot of address space, IPv6 is becoming more popular. The header of IPv6 packets is compressed and large ones are broken up using the 6LoWPAN adaption layer to fit them within the IEEE 802.15.4 frame. Time Slotted Channel Hoping (TSCH) is chosen as a channel access mechanism to lessen radio duty-cycling. The objective function is used by the RPL routeing protocol for low power and lossy networks to construct a Destination Oriented Directed Acyclic Graph (DODAG) from a collection of constraints and metrics. Attack simulation, which is primarily focused on dataset generation, should consider concurrent assaults instances where the number of changes throughout the simulation or where it may be necessary to configure the length of an attack.

Most of the research in RPL attacks is concentrated on performance analysis under attacks, with little focus on how operating system modules should be altered to effectively launch such attacks. Due to the RPL's characteristics, which include a lack of infrastructure, unreliable links, resource limitations, a lack of adequate physical protection, and changing topology, they are particularly vulnerable and challenging to defend against attacks.

RPL implementation for well-known RPL attacks developed by Contiki-NG, all of which are made publicly accessible via a Github repository. Additionally, the framework which is provided for researchers using COOJA, a concurrent simulation of a range of RPL attacks with variable durations running on various nodes using an event-driven simulator for Contiki-NG devices. The use of simulations is particularly helpful for carrying out virtual runs on huge sensor networks made up of many motes.

2 RPL and Security Concern

RPL is system where it uses distance-vector routing in accordance with IPv6. According to DODAG RPL devices are interconnected. Data sink of the graphs create DODAG graphs. Single network can run more than one RPL at the same time which uses multiple DODAG graphs. RPL is calculated by finding the best paths based on the given metrics and constraints. In detailed, this method is able to minimize energy consumption as it is run by the most efficient method with the shortest paths. Hence, RPL node is able to combine several instances, but it has limitation and only can include one DODAG graph. Different stimulation allows RPL to perform different optimization. The RPL packets can be elaborated according to 3 difference pattern of traffic from leaves to the root with an upward direction to the routes, point to multipoint from root to leaves with downward direction routes and point to point traffic as shows in the diagram red doted arrows represents both upwards and downward direction routes. (Mayzaud, Badonnel & Chrisment 2016, p. 460)

2.1 DODAG Graph

A step-by-step process has been used to assemble the DODAG graph. Initially, the root was broadcasted as a DIO message (DODAG Information Object). All messages contain the necessary information for RPL nodes to generate a RPL instance. For instance, abstracting the configuration parameters, choosing a parent set and maintaining the DODAG Graph. Once node receiving a DIO message, it will automatically add the details of the sender to its parents listing and self-determine its ranking by considering the objective as reported in the DIO message. The ranking is based on the value of the DIO message, with the considerations of the graphs, and roots must be greater in value that its parent's ranking for it to assure the nature of acyclic pattern of the graph. Then it sends its other neighboring nodes the updates DIO. Based on the listing, the nodes will then choose the preferent parent which will automatically become the default gateway. In the final part, all nodes which has participated in the DODAG graph will have an upward default pathway in the DODAG root. (Mayzaud, Badonnel & Chrisment 2016, p. 460)

DIO messages are sent in a said repetitive pattern in accordance to the timer set with the trickle algorithm in which is transmit frequency in trolling most of the messages with the considerations of the network state. By broadcasting a DIS message and asking its neighbours for their DIO messages, a new node may be expected to join an existing network. DAOs (Destination Advertisement Objects) are used to create paths that lead downward. Taking into account the mode of operation that the root indicated in the DIO message. Where it is claimed that router nodes maintain consistent routeing tables. The kid unicast will send a DAO message to the designated parent, which keeps note of it, in order to maintain the mode. The parents modify the paths taken to the DODAG root by the other DAO messages that had been unicasted. Despite the fact that intermediary notes have the ability to reverse the path, they do keep track of the address when receiving DAO even though they do not store their own routeing information (Mayzaud, Badonnel & Chrisment 2016, p. 461).

2.2 Security Concern

There are various types of security threats that could target the RPL protocol. The limitations on resources, inadequate physical security, unpredictable topology, infrastructure, and faulty connectivity that characterize LLN networks making the particularly weak and challenging to defend against assaults. It is the uniqueness of RPL protocol, but it can also be used with wired or wireless sensor networks. The RPL protocol specifies a number of security-related techniques. It specifies two encryption techniques for data packets. However, link layer and transport/application layer security are the foundation for most implementations of these networks. We make the assumption that an attacker can defeat security at the link layer by either taking advantage of a flaw or obtaining access to a shared key. A broken or incorrectly configured node with characteristics that can impair network performance can also act as an attacker (Mayzaud, Badonnel & Chrisment 2016, p. 461).

Three categories of security assaults are taken into account by the taxonomy, which is shown in Figure 1. Attacks intended to exhaust network resources (power, energy, and memory) fall under the first category. Because they significantly reduce the device lifetimes and RPL network lifetime, these attacks are especially harmful for such restricted networks. Attacks aimed at the RPL network topology are included in the second category. They interfere with the network's regular operation in two ways: either a group of RPL nodes may be cut off from the network or the topology may be less optimal than it would be during a typical convergence of the network. The third category includes assaults that target network traffic, such as eavesdropping and misappropriation attacks (Mayzaud, Badonnel & Chrisment 2016, p. 461).

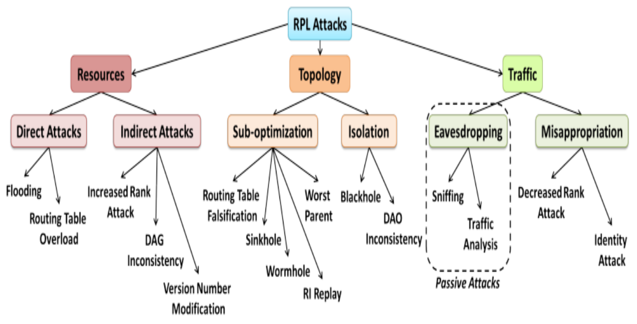


Figure 1 : Taxonomy of attacks against RPL networks (Mayzaud, Badonnel & Chrisment 2016, p. 460)

3 Technologies for Implementation

3.1 Virtual Box

Virtual Box is open-source virtualization software program that was created by Sun Microsystems and innotek. A wide range of guest operating systems, including Linux, Solaris, OpenBSD, and the majority of Windows desktop or server operating systems, can be supported by VirtualBox, which can be installed on host systems running Linux, Windows, Open Solaris, and Mac OS X. Since its introduction on January 15th, 2007, VirtualBox has gained the support of a sizable user base and has established itself as a dominant force in the virtualization industry. VirtualBox was selected as the main virtualization solution due to its widespread use and open source nature. Two more important components that were crucial to the operation of these tests were VRDP and VBox Manage (Griffin & Doyle 2009, p. 3)

3.2 Contiki-NG

For wireless embedded devices with severe resource constraints, there is Contiki-NG (Next Generation), an open source, cross-platform operating system. It focuses on established low-power protocols including 6LoWPAN, IPv6, 6TiSCH, RPL, and CoAP as well as dependable (reliable and secure) low-power communications. Its main goals are to I speed up the development and evaluation of Internet of Things research concepts, shorten the time needed for Internet of Things applications to reach the market, and offer an intuitive platform for teaching embedded systems-related courses in higher education. The Contiki OS was the source of Contiki-NG, which still has a lot of its original functionality. This article discusses the inspiration for Contiki-NG, provides the most recent version (v4.7), and uses concrete examples to demonstrate the impact of Contiki-NG. Contiki-NG occupies the same space as other embedded operating systems, including RIOT, Zephyr, Arm Mbed, Apache Mynewt, TinyOS, and FreeRTOS. The authors of this research believe that Contiki-NG fits the need for low-power, IEEE 801.15.4 wireless mesh networks with its implementation of TSCH, RPL-Classic, and RPL-Lite. (Oikonomou et al. 2022, p. 2)

3.3 COOJA

The Cooja network simulator, made available by the Contiki OS project, has grown to be a popular tool in the Wireless Sensor Networks field (WSN). It is heavily utilized by the research community to simulate small to reasonably large wireless networks of linked devices embedded with sensors and/or actuators, also known as "motes," and to create, debug, and evaluate projects based on the WSN technology. The Java-based Cooja application itself offers three key features such as an extensible framework that enables the integration of external tools to add new features to the Cooja application, graphical user interface (GUI) based on Java's standard Swing toolkit for designing, running, and analysing WSN simulations, and a simulation of the radio medium underlying the wireless communications of sensor networks. Cooja can truly imitate the numerous motes that make up the WSNs thanks to the last functionality. In fact, it incorporates and employs specialized emulator programs that carry out cycle-accurate emulation of the chips used to construct motes, such as radio transceivers and microcontroller units (MCUs). This emulation mechanism is one of Cooja's strongest points because it enables the execution of extremely fine-grained, accurate, and low-level simulations. As a result, Cooja has emerged as a top tool for debugging and assessing WSN-related software (Roussel, Song & Zendra 2016, p. 3).

4 Sink Hole Attack

The routing attacks with a high destruction in the 6LOWPAN and RPL is the sinkhole attacks. It draws nodes by displaying various routing matrices fake link quality, the quickest way, and sending false information to a node that is vulnerable. In the beginning, the malicious node is successful in drawing a lot of traffic by promoting fabricated information data. It then alters or drops the traffic after having illegally received it. With the manipulation of the rank value, the attack is easily to carry out in RPL networks. Due to this fabricated advertisement, malicious node is commonly selected by the other nodes as their preferred parent even though it does not offer higher performance. The paths are therefore not network optimized. Attackers change the topology, which lowers network performance**.**

4.1 Mechanism of Sink Hole Attack

The closer a node is to the root and the more traffic it must handle in a DODAG graph, the lower its rank. A malicious node overstates its performance when it falsely advertises a lower rank value. As a result, the attacker is used by numerous trustworthy nodes to join to the DODAG graph. As seen in Figure 2, this attracts a significant portion of the traffic. The malicious node can now carry out sinkhole attacks because of this action. An attacker can modify the rank value of the RPL protocol by forging DIO messages (Jamil et al. 2021, p. 96).

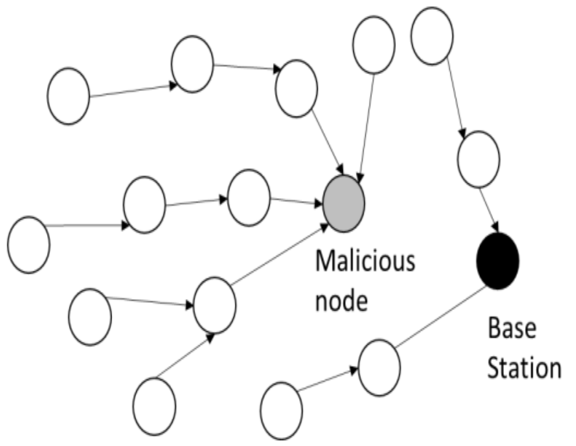


Figure 2 : Illustration of sink hole attacks (Jamil et al. 2021, p. 96)

4.2 Implementation Sink Hole Attack in Cooja

A compromised node or malicious node promotes false rank information to create the fakes routes in a sinkhole attack. The packet information is lost after the message packet has been received. Sinkhole attacks have an impact on the RPL protocol's performance in IoT networks. During DODAG creation, the transmitter in RPL broadcasts the DIO. The receiver updates its sibling list, parent list, rank, and route information after receiving a DIO from the transmitter. Malicious node always promotes a fake rank after receiving the DIO message rather than updating the rank. The other nodes adjust their rank in accordance with the bogus rank while listening to the malicious node's DIO communication. If the malicious node is the preferred parent after the establishment of the DODAG, instead of sending the packet to its parent, the malicious node will receive it. The difference between normal node and malicious node is there is additional code to activate the attack as shown in Listing 1. We need click the mote when activated the attack. If we want to deactivated we can click again the mote**.**

```
if((ev==sensors_event) && (data == &button_sensor)) {
            if(attack_flag_sinkhole)
            {
                    printf("\nAttack deactivated");
                    attack_flag_sinkhole = 0;
            }else {
                    printf("\nAttack activated\n");
                    attack_flag_sinkhole = 1;
            }
    }
```

Listing 1: home/Contiki/examples/ipv6/rpl-udp/malicious-udp-client.c

```
if(INFINITE_RANK - base_rank < rank_increase) {
  /* Reached the maximum rank. */
  new_rank = INFINITE_RANK;
} else {
 /* Calculate the rank based on the new rank information from DIO or
    stored otherwise. */
  if(attack_flag_sinkhole) {
                 new_rank = (int) (p->rank * 7  / 20);
                 if(new_rank<256)
        new_rank=256+20;

      } else {
                PRINTF("Ute av SINKHOLEattack_flag\n");
                new_rank = base_rank + rank_increase;
      }
  PRINTF("Rank value : %d \n ",new_rank);
  PRINTF("Parent is : %u\n",(rpl_get_parent_ipaddr(p))->u8[15]);
}
```

Listing 2: home/Contiki/core/net/rpl/rpl-mhrof.c

As shown in Listing 2, if the attack is activated the rank of malicious will be calculated and should be more than 256, which is higher than root rank. If the calculation result is lower than 256, the new rank will be set to 276. The difference sink hole attack compare to other attacks is malicious node in sink hole attacks use higher rank that root rank to do the attack.

5 Reproduce the simulation in Cooja

This project involves the simulation of an RPL routing protocol in Ad-hoc networks which is done using Cooja.

5.1 Methodology

We download the file in Github directory that contains the necessary files to replicate the project. The first step is installed Virtual Box. After that Install InstantContiki3.0 which can be found in https://sourceforge.net/projects/contiki/files/Instant%20Contiki/Instant%20Contiki%203.0/ and run it on a Virtual Box. The password to access Contiki is 'user'. After that Navigate to contiki/examples/ipv6/rpl-udp and replace the Makefile with the modified version in this Github. Many of the required code to simulate an RPL network is built into Contiki. To open the simulator, navigate to the home/user/contiki/tools/cooja then type 'ant run' and select UDGM as the radio medium.

Next Step is Click [file > new simulation] A new box opens, asking for the simulation's title and basic simulation parameters, including the radio medium to be utilized and the random start-up delay period of the nodes. Pressing the generate button creates the simulation after that. It then displays the primary simulation window along with various plugins for interacting with the simulation, such as the network window and simulation control window.

The "Motes" menu is where the node type is created. Keep in mind that COOJA refers to a node as a "mote." This menu lists the several types of motes that can be utilised in a simulation, such as java, operating system-level, and emulated motes. A variety of particular hardware platforms are also included, including Z1 Mote, Sky Mote, and Mica Mote. The Contiki application is chosen for the chosen mote type after the mote type has been chosen. The COOJA built-in compiler is used to compile the process for this particular mote type. In out simulation we use Z1 mote. Click [motes > add motes > create new mote types > Z1 mote > browse] as shown in Figure 3. Select the mote from the contiki/examples/ipv6/rpl-udp which is any of the c program files. For example, the default files udp-client.c is a normal node and udp-server.c is a root node.

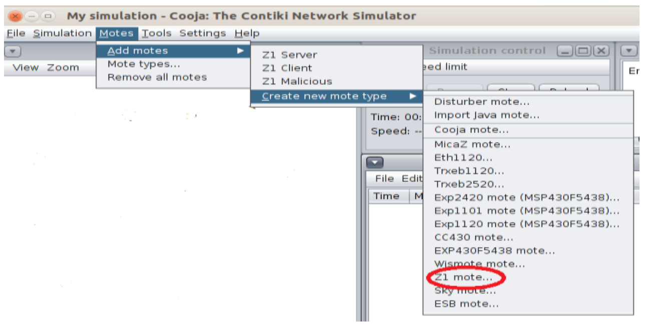
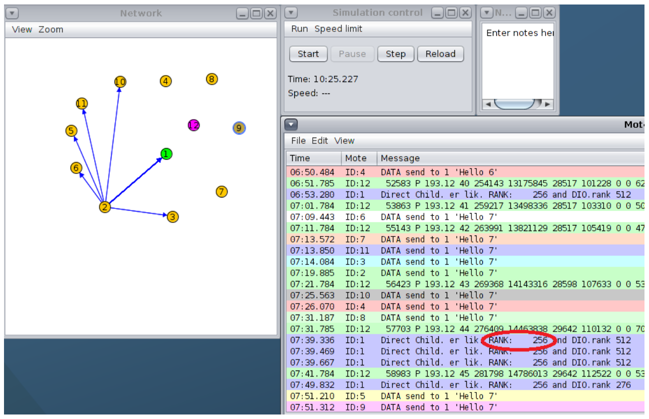


Figure 3 : Create new mote Z1

Once the mote is selected, select clean then compile. You will be given the option to choose the position and quantity of motes to include. For project, the positioning was random. And the number of motes. We use 1 root mote, 10 normal mote, and 1 malicious mote in this simulation. Once the motes are in position, click start on the simulation and allow it to run as shown in Figure 4. Additionally, to provide a better observation of the network simulation, the 'view' menu can be opened to enable the display of radio traffic, node ID, etc. To include a malcious mote which will act as a udp-client, add the malicious-udp-client-sinkhole.c and malicious-udp-client-sinkhole.h files into home/contiki/examples/ipv6/rpl-udp folder. Then replace the rpl-mhrof.c in the home/contiki/core/net/rpl folder with the modified version in this github. The malicious will default as a normal node once the simulation is running.

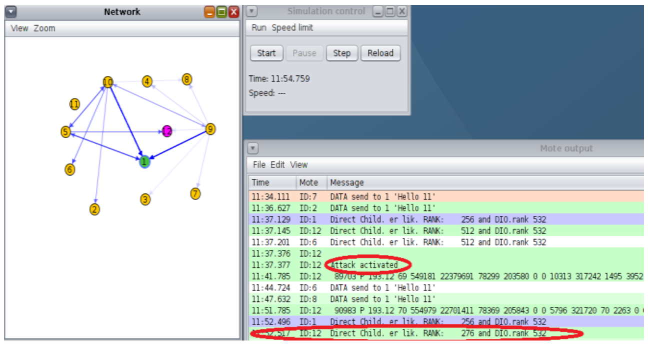


```
Time: 10:25.227
Speed: ---
```

```
Time       Mote   Message
06:50.484  ID:4   DATA send to 1 'Hello 6'
06:51.785  ID:12   52583 P 193.12 40 254143 13175845 28517 101228 0 0 62
06:53.280  ID:1   Direct Child. er lik. RANK:    256 and DIO.rank 512
07:01.784  ID:12   53863 P 193.12 41 259217 13498336 28517 103310 0 0 50
07:09.443  ID:6   DATA send to 1 'Hello 7'
07:11.784  ID:12   55143 P 193.12 42 263991 13821129 28517 105419 0 0 47
07:13.572  ID:7   DATA send to 1 'Hello 7'
07:13.850  ID:11  DATA send to 1 'Hello 7'
07:14.084  ID:3   DATA send to 1 'Hello 7'
07:19.885  ID:2   DATA send to 1 'Hello 7'
07:21.784  ID:12   56423 P 193.12 43 269368 14143316 28598 107633 0 0 53
07:25.563  ID:10  DATA send to 1 'Hello 7'
07:26.070  ID:4   DATA send to 1 'Hello 7'
07:31.187  ID:8   DATA send to 1 'Hello 7'
07:31.785  ID:12   57703 P 193.12 44 276409 14463838 29642 110132 0 0 70
07:39.336  ID:1   Direct Child. er lik. RANK:    256 and DIO.rank 512
07:39.469  ID:1   Direct Child. er lik. RANK:    256 and DIO.rank 512
07:39.667  ID:1   Direct Child. er lik. RANK:    256 and DIO.rank 512
07:41.784  ID:12   58983 P 193.12 45 281798 14786013 29642 112522 0 0 53
07:49.832  ID:1   Direct Child. er lik. RANK:    256 and DIO.rank 276
07:51.210  ID:5   DATA send to 1 'Hello 7'
07:51.312  ID:9   DATA send to 1 'Hello 7'
```

Figure 4 : Simulation before attack is activated

Simulation is run by pressing start button on simulation control window. We should wait for a while to wait the mote start communicating to each other in this RPL Network. In this simulation all the mote sending package to the root mote Number 1. In mote output, you can see the rank is 256 and all the traffic will be directed to root mote.To activate the malicious node, right click the mote and click 'Click button on Z1' to initiate attack. You will see in the mote output there is message that Attack activated as shown in Figure 5. The attacker mote Number 12 is advertising the rank which is 276. All the network traffic will be directed to attacker mote, but sometimes directed to root mote too. In this simulation we have generated sink hole attack in RPL network using Cooja Simulator successfully by create malicious node to promote false rank information to create the fake routes.



```
Time: 11:54.759
Speed: ---
```

```
Time       Mote   Message
11:34.111  ID:7   DATA send to 1 'Hello 11'
11:36.627  ID:2   DATA send to 1 'Hello 11'
11:37.129  ID:1   Direct Child. er lik. RANK:    256 and DIO.rank 532
11:37.145  ID:12  Direct Child. er lik. RANK:    512 and DIO.rank 532
11:37.201  ID:6   Direct Child. er lik. RANK:    512 and DIO.rank 532
11:37.376  ID:12
11:37.377  ID:12  Attack activated
11:41.785  ID:12   89703 P 193.12 69 549181 22379691 78299 203580 0 0 10313 317242 1495 3952
11:44.724  ID:6   DATA send to 1 'Hello 11'
11:47.632  ID:8   DATA send to 1 'Hello 11'
11:51.785  ID:12   90983 P 193.12 70 554979 22701411 78369 205843 0 0 5796 321720 70 2263 0
11:52.496  ID:1   Direct Child. er lik. RANK:    256 and DIO.rank 532
11:52.517  ID:12  Direct Child. er lik. RANK:    276 and DIO.rank 532
```

Figure 5 : Simulation after attack is activated

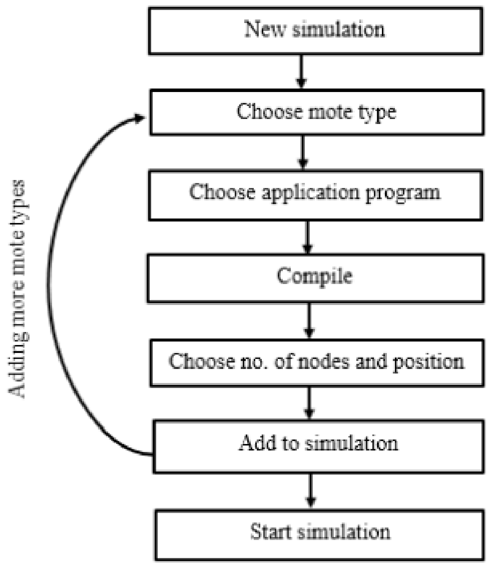


Figure 6 : Simulation development cycle (Jamil et al. 2021, p. 97)

In Figure 6 we can see how the simulation development cycle from new simulation is created until start simulation. Additionally, the simulation can be closed and reopened. COOJA records the number, type, and placement of nodes during a simulation. Every time the simulation is opened and restarted, the running state is not preserved, and the simulation starts over from scratch.

5.2 Difficulties

When the installation Contiki-NG and Cooja start from the scratch, we found some issues. The newest version of Ubuntu 22.04.01 cannot run the script as shown in Listing 3. The error is because of python serial. The solution of this error is change python-serial to python3-serial. We can use Ubuntu 18.04.06 which is better and do not have a lot of error like the newest version. After I successfully installed Cooja I found that open the UI of cooja simulator is different. If I run ant run in cooja directories, there is error that build.xml didn't exist. I try to find the information and find to open the UI of cooja simulator, we need to use gradlew run command.

```
$ sudo apt update
$ sudo apt install build-essential doxygen git git-lfs curl wireshark python-serial srecord rlwrap
```

Listing 3: script installing Contiki-NG

After I can open the simulation, when I want to add the new mote and try to clean and compile. There is error MSPsim emulator should be updated to the newest version. I try to update MSPsim emulator but still error. We are advised to import the OS that was given by the lecture assistant so we can run the simulation quickly because we have limited time. When we want to create malicious mote there is an error again because of the path as shown in Listing 4 After we change the path correctly, the problem was solved.

```
#include "/home/user/Documents/IDS/malicious-udp-client.h"
#include "/home/user/Documents/IDS/malicious-udp-client-sinkhole.h"
```

Listing 4: home/contiki/core/net/rpl/rpl-mhrof.c

It was very hard reproducing the simulation because minimal resources about Contiki-NG and Cooja Simulator. We are not familiar with the command in Linux too because we always use Windows that makes the project takes time to be reproduced.

6 Result and Analysis

Based on the paper, all simulations ran for 10 minutes with real-time speed (i.e. 1.0x) and COOJA's Unit Disk Graph Medium (UDGM) as a radio medium. There are five types of attack which is Selective Forwarding Attack (SFA), Sinkhole Attack (SHA), Version Number Attack (VNA), DIS Flooding Attack (DFA), Sybil Attack (SYA).

An attacker node engages in a Selective Forwarding Attack (SFA), dropping all packets other than control packets (i.e. DAO messages in RPL non-storing mode). Attack X in the template is replaced with SFA by the SFA implementation, which is also true for project-conf.h and contiki-default-conf.h. The implementation is simple; upon attack activation, all forwarded packets are discarded with the exception of ICMPv6 packets carrying the code 155, which is the code for RPL packets. The exclusion is meant to handle the scenario when DAO messages are routed back to the root of the DODAG when RPL is in nonstoring mode (Algahtani, Tryfonas & Oikonomou 2021, p. 282).

In a Sinkhole attack as described in Section 3, To draw more traffic for the purpose of listening in on conversations, a node informs its neighbours of a better routeing path. The current version of a DODAG is denoted by a number in Version Number Attack (VNA) for RPL DIO messages. When a DODAG's root notices inconsistencies, the version number is increased to start a worldwide repair. This causes all Trickle timers to be reset, routeing tables to be cleared, and the DODAG joining process to be carried out once more. Since other nodes have no way of confirming the validity of such an increase, a hijacked node can increment the version number (Algahtani, Tryfonas & Oikonomou 2021, p. 283).

In DIS The goal of flooding attacks in RPL-based IoT is to exhaust resources. A hacked node can conduct a flooding attack using DIS messages, which are meant to elicit DODAG information from neighbours. This causes the neighbours to respond with DIO messages after resetting their DIO Trickle timers. Whether or whether a compromised node has already joined the DODAG, it should always be possible for it to initiate the attack at any time. A hacked node shortens its DIS broadcast duration to start the flooding assault. We allow the DFA's attack object to receive a random number for its target node in order to demonstrate that a target node in our framework can be random. The random target cannot be the root, zero, or the number seven (Algahtani, Tryfonas & Oikonomou 2021, p. 284).

Malicious node claiming numerous false identities in Sybil attack. These false identities can be used by the malicious node to register fraudulent routes, pose as other genuine nodes, or get beyond identity inspection-based countermeasures. As a result, SYA renders the majority of security countermeasures ineffective and is extremely challenging to stop or even detect. A node's link-layer address and link-local address must both be changed in order to implement SYA (Algahtani, Tryfonas & Oikonomou 2021, p. 284).

In the paper, the author run five types of simulation, but we only focus reproduce sink hole attack because of time limit of this project. In Listing 5, the author include SHA into Contiki-NG RPL-lite's implementation by using the set16 function to set the 2 bytes of the rank field in DIO messages to be ROOT_RANK.

```
#if SHA == 1 // after line 80
uint8_t SHA_on;
#endif
#if SHA == 1 // after line 371
if(SHA_on)
  set16(buffer, pos, ROOT_RANK);
#endif
```

Listing 5: os/net/routing/rpl-lite/rpl-icmp6.c (Algahtani, Tryfonas & Oikonomou 2021, p. 283)

In Listing 6, SHA into Contiki-NG RPL-lite's implementation by using the set16 function to set the 2 bytes of the rank field in DIO messages to be INFINITE_RANK. In Listing 2, we set the INFINITE_RANK by the calculation if SHA is activated. We Calculate the rank based on the new rank information from DIO or stored otherwise.

```
#if RPL_LEAF_ONLY
  PRINTF("RPL: LEAF ONLY DIO rank set to INFINITE_RANK\n");
  set16(buffer, pos, INFINITE_RANK);
#else /* RPL_LEAF_ONLY */
  set16(buffer, pos, dag->rank);
#endif /* RPL_LEAF_ONLY */
```

Listing 6: home/Contiki/core/net/rpl/rpl-icmp6.c

In the paper, the author also turn off DIO Trickle timer to maximise SHA's impact, by restoring the original ticks value as in Listing 7.

```
#if SHA == 1 // after line 145
if(SHA_on)
  ticks = (time * CLOCK_SECOND) / 1000;
#endif
```

Listing 7: os/net/routing/rpl-lite/rpl-timers.c ((Algahtani, Tryfonas & Oikonomou 2021, p)

When we reproduce this project, we also do the same restoring the original ticks value as shown in Listing 8.

```
 /* Convert from milliseconds to CLOCK_TICKS. */
 ticks = (time * CLOCK_SECOND) / 1000;
```

Listing 8: home/Contiki/core/net/rpl/rpl-timers.c

In the paper the author shows the scenario as shown in Figure 7, For the duration of the experiment, node 7 will launch SFA and SHA. Node 7 is the sink for most nodes. The scenario using 1 root node which is node 1, 1 malicious node which is node 7, and 14 normal nodes.

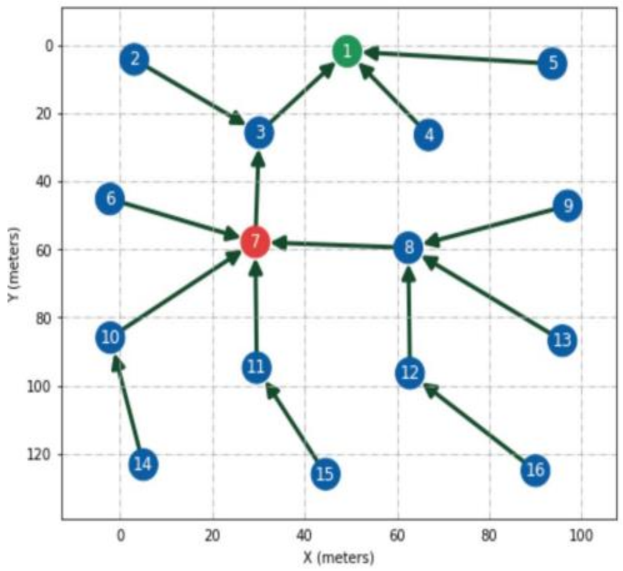


Figure 7: SFA and SHA DODAG (Algahtani, Tryfonas & Oikonomou 2021, p. 283)

When we reproduce this project, we use 1 root node which is node 1, 1 malicious node which is node 12, and 10 normal nodes as shown in Figure 7. Based on the paper, we have successfully reproduced the SFA and SHA simulation together. SFA basically drops all the incoming packets. On the other hand, in SHA, malicious nodes advertise themselves as a better routing path to attract more traffic. This is the reason why SFA used to drop all incoming packets after receiving it and we need to run both attacks together to simulate a sinkhole attack.

6 Strengths and Weaknesses of the Simulation

6.1 Strengths

The strength of the simulation is Cooja can perform cross-level simulations as opposed to conventional simulations limited to one level. For these cross-level simulations, COOJA is an appropriate simulator. A system simulator, which enables the development of algorithms, analysis of system behaviour, and observation of interactions in a controlled environment, can simplify software development for sensor networks. At a particular, fixed level, such as the application, operating system, or hardware level, current WSN simulators do simulation. The level of simulation impacts both the efficiency of the simulator's execution and the level at which software development is possible. A simulator that replicates a specific sensor node platform at the hardware level allows for the creation of low-level software, such as device drivers, but at the expense of longer simulation periods and more complex code. On the other hand, a high-level simulator that solely supports the creation of high-level algorithms may offer quick simulation speeds (Roussel, Song & Zendra 2016, p. 3).

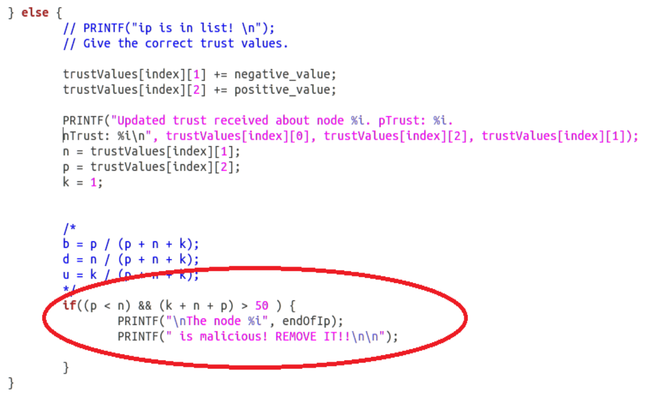
A novel simulator for the Contiki operating system that supports cross-level simulation, or concurrent simulation at numerous system levels. COOJA integrates high-level behaviour simulation and low-level simulation of sensor node hardware into a single simulation. All tiers of the system, including the operating system, radio transceivers, and radio transmission models, can be modified or replaced, making COOJA versatile and extendable. Since the simulator is implemented in Java, users can easily modify it, but using the Java Native Interface, sensor node software written in C is also possible. Additionally, the sensor node software can be launched in a sensor node emulator, which simulates an actual sensor node at the hardware level, or as compiled native code for the platform on which the simulator is run (Roussel, Song & Zendra 2016, p. 3).

6.2 Weaknesses

The weakness of the simulation is a simulated model which is a condensed representation of an actual system or object. Because simulated models only comprise modelling of events and factors that are presumptively pertinent to the operation of an application of a system or an object, their scope is likewise constrained. Real-world objects are not perfectly modelled in simulations. An infinite number of uncontrollable occurrences from an unidentifiable object in the vicinity of the thing under observation, may take place, some of which may or may not be relevant to how real-life objects function. Simulated findings become meaningless if the model is oversimplified since numerous important elements may be disregarded. To produce findings that can be trusted, the model must be founded on reliable hypotheses. The complexity and computing requirements of the simulator, however, rise as the model tends to become more realistic by adding more simulated components. Simulation always involves a trade-off between the accuracy and effectiveness of the model. When compared to experiences created on real hardware, there is timing errors. More specifically, the simulations predicted delays in packet transmission (TX) over the radio medium that weren't actually seen in the real world. The COOJA application development process can be made better by including support for more hardware platforms. Additionally, documentation for the collect-view and MSPsim emulator can improve COOJA's usefulness (Roussel, Song & Zendra 2016, p. 4).

7 Improvement of the Simulation

We can enhance the simulation in the intrusion detection system (IDS), which keeps track of hostile network traffic. The IDS serves as a second line of defence to keep attackers out of the network. IDS give defenders the opportunity to identify and mitigate assaults before they could result in significant losses, which makes them crucial in the protection of sensitive computer systems. A defence must incur excessively high operational costs as a result of an oversensitive IDS's high frequency of false alerts since alarms must be manually investigated. Defenders must therefore achieve the ideal balance between increasing security and lowering costs. When multiple interconnected computer systems must be protected from a strategic attacker who can target computers in order to maximise losses and reduce the likelihood of detection, optimising the sensitivity of intrusion detection systems is particularly difficult.



```c
} else {
        // PRINTF("ip is in list! \n");
        // Give the correct trust values.

        trustValues[index][1] += negative_value;
        trustValues[index][2] += positive_value;

        PRINTF("Updated trust received about node %i. pTrust: %i.
nTrust: %i\n", trustValues[index][0], trustValues[index][2], trustValues[index][1]);
        n = trustValues[index][1];
        p = trustValues[index][2];
        k = 1;


        /*
        b = p / (p + n + k);
        d = n / (p + n + k);
        u = k / (p + n + k);
        */
        if((p < n) && (k + n + p) > 50 ) {
                PRINTF("\nThe node %i", endOfIp);
                PRINTF(" is malicious! REMOVE IT!!\n\n");

        }
}
```

Listing 7: home/Contiki/examples/ipv6/rpl-udp/udp-server-time-trust.c

To set up the IDS system, replace the rpl-icmp6.c and rpl-private.h in the home/contiki/core/net/rpl folder then include the udp-client-time-trust.c udp-server-time-trust.c files in home/contiki/examples/ipv6/rpl-udp along with their respective header files. To change characteristics of the IDS, edit the line 140 of udp-server-time-trust.c file to modify the uncertainty threshold as shown in Listing 7. We may change the value 50 to any other value based on the network we have because it is all about the rank of the motes. The aim of this code is to remove malicious node when it is detected.

8 CONCLUSIONS

We have provided a framework for simulating sink hole attacks. The framework was created using C preprocessor directives, which allow attacks to be optionally compiled and activated via COOJA. By initiating sink hole attacks on various randomly selected nodes, we demonstrated the capability of our simulation. Network traffic attacks enable a malicious node to gather and analyze a significant portion of the traffic. We have implemented IDS system for improvement to prohibit malicious node for the solution of security issue.

REFERENCES

[1] Algahtani, F., Tryfonas, T. & Oikonomou, G 2021, 'A Reference Implemenation for RPL Attacks Using Contiki-NG and COOJA,' *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pp. 280-286.

[2] Jamil, A., Ali, M., and Alkhalec, M.E.A 2021, 'Sinkhole Attack Detection and Avoidance Mechanism for RPL in Wireless Sensor Networks', *Annals of Emerging Technologies in Computing*, vol. 5, pp. 94-101.

[3] Roussel,K., Song,Y., Zendra, O 2016. 'Using Cooja for WSN Simulations: Some New Uses and Limits', *EWSN 2016 - NextMote workshop*, pp. 319-324.

[4] Mayzaud, A., Badonnel, R. and Chrisment, I 2016. 'A Taxonomy of Attacks in RPL-based Internet of Things. International Journal of Network Security', 18(3), pp.459-473.

[5] Oikonomou, G., Duquennoy, S., Elsts, A., Eriksson, J., Tanaka, Y., & Tsiftes, N 2022. 'The Contiki-NG open source operating system for next generation IoT devices'. *SoftwareX*, 18, 101089.https://doi.org/10.1016/j.softx.2022.101089.