Mobile and Embedded Computing

LINGI2146

Report

"RPL Attacks Framework"

Authors: Alexandre D'Hondt
Hussein Bahmad
Jérémy Vanhee
Professor: Ramin Sadre
Academic year: 2015-2016

1. Introduction

In the scope of the project for *LINGI2146* course, we are required to design a Wireless Sensor Network (WSN) on top of Contiki solving a common problem such as recording temperature measures. The main goal of this project is to become familiar with technologies and concept requisite in the field of mobile and embedded computing.

Instead of solving a common problem, we have decided to build a framework for testing attacks on the RPL protocol, i.e. its implementation in Contiki. The interest is that, as shown in [1], this routing protocol is insecure and can then be tested for multiple vulnerabilities. So, instead of building an application for a WSN at the data plane, we thus refactor the underlying goal and make it a security-oriented project, acting at the control plane in the network layer, as depicted in the following stack.

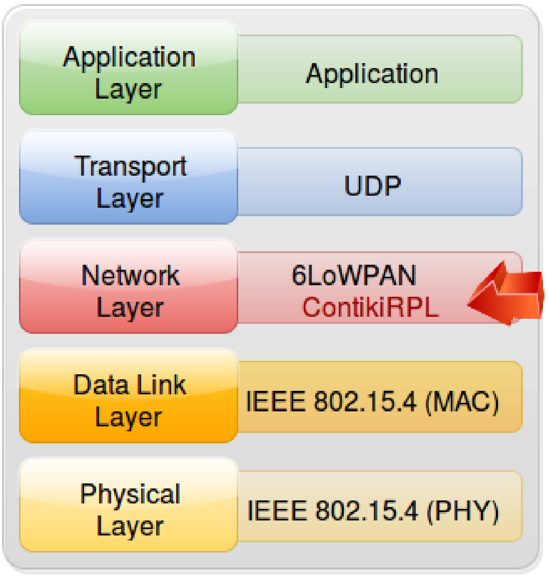


Figure 1: Contiki network stack (with the attacked layer highlighted)

Formally, the objectives are two-fold :

1. Build a convenient framework for testing a malicious node into Cooja simulations
2. Test and show the effects of some chosen attacks

The remainder of this document is structured as follows :

- RPL Attacks : we provide a bit of theory about attacks we want to test
- Framework : we explain how the framework is designed and implemented in order to achieve testing of RPL attacks
- Experiments : we provide results on tested RPL attacks
- Conclusion

The following conventions apply in the remainder of this document :

- A node in green represents DODAG's root
- A node in red with a label 'A' represents the attacker
- Nodes in yellow with a number as a label are normal nodes

2. RPL Attacks

This section addresses the taxonomy of RPL attacks as presented in [2] and [3]. It also mentions the attack we want to test.

2.1. Taxonomy

The taxonomy as explained in details in [3] is shown in the picture hereafter. Some particular attacks are surrounded by blue frames, indicating the attacks we have chosen to test.

The first category concerns the **exhaustion of network resources**, meaning that malicious node’s purpose is to overload the consumption of energy, memory or/and power. This can be done by forcing the legitimate nodes to perform unnecessary actions to increase the use of their resources. This may impact on the availability of the network by congesting available links or by incapacitating nodes and may therefore impact on the lifetime of the network.

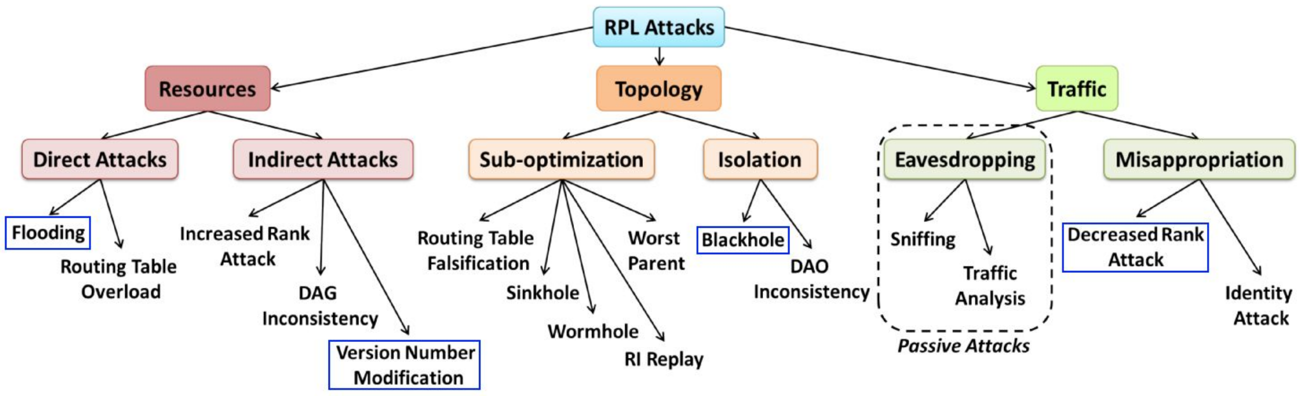


Figure 2: Taxonomy of attacks against RPL networks [2]

This category can be further subdivided in two sub-categories ; **direct attacks**, in which the malicious node directly generates the overload disturbing the network, and **indirect attacks**, in which the malicious node provokes the other nodes to make them generate the overload.

The second category holds the **attacks targeting the RPL network topology**. The goal of these attacks is to disturb the normal operation of the network. These could then cause the isolation of one or more nodes. This category can also be subdivided in two sub-categories ; **sub-optimisation**, meaning that the network will converge to a non-optimal form, inducing poor performance, and **isolation** of a node or a subset of nodes, cutting them from the rest of the network and hence the root node.

The third category covers **attacks against the network traffic**. These attacks are aimed to make a malicious node introduce itself inside the network, not disturbing its working. This leads to information leakage by eavesdropping the traffic or impersonating legitimate nodes. This category is again subdivided in two sub-categories ; **eavesdropping** (passively) the information that is forwarded through the network or **misappropriation** of a node or a set of nodes, namely for tampering the legitimate exchanged information.

2.2. Tested attacks

From the three discussed categories, our goal is to choose a sample of attacks and to test them. Amongst these, the followings are considered :

- **Flooding** *[Resources | Direct attack]* : consists of generating a large amount of traffic through DIS messages, causing nodes within range to send DIO messages (used to advertise information about DODAG's to new nodes) and reset their trickle timers (supposed to increase as the network stabilizes). Note that, if secure DIS are used, this attack can still be performed using a compromised node.

  Mitigation : A simple solution to this attack is to check the link to be bidirectional for each HELLO message. If no link-layer acknowledgements are received, the path is assumed to be bad, and a different route is chosen.

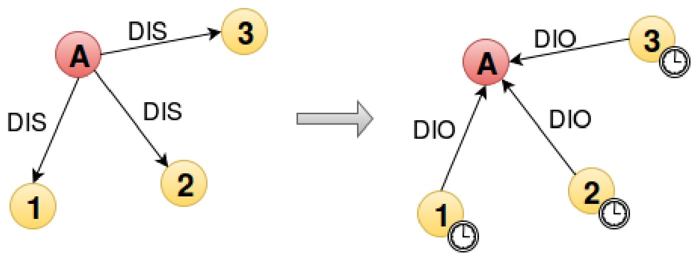


Figure 3: Flooding attack (**A** means Attacker ; **1**… are legitimate nodes)

- **Version Number Modification** *[Resources | Indirect attack]* [4] : consists of increasing the version number (which is normally a responsibility of the DODAG's root when a global repair is to be performed), hence causing unnecessary graph rebuildings. Indeed, as the root receives the DIO with an invalid version number, it updates it and resets its trickle timer (as depicted by the timer in the figure below) for resending a new DIO. By contrast, normal nodes initiate a global repair (as depicted by the wrench), that is, they remove their parents and use the received DIO to update their new parent.

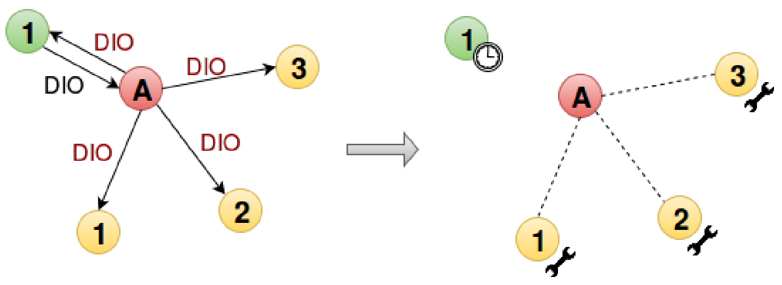


Figure 4: Versioning attack (DIO in black is legitimate; red one has version increased by 1)

  Mitigation : A mechanism called VeRa [5] (for Version Number and Rank Authentication) has been designed by Amit Dvir to prevent a malicious node from illegitimately increasing the version number, using authentication mechanisms based on hash operations, but does not seem to be implemented in ContikiRPL.

- **Decreased Rank** *[Traffic | Misappropriation]* : consists of advertising a lower rank to make the legitimate nodes connect to the DODAG via the attacker ; this can of course be a basis for sinkhole, blackhole or also eavesdropping attacks.

  Mitigation : Several mechanisms can be used together, such as VeRa and TRAIL [7].

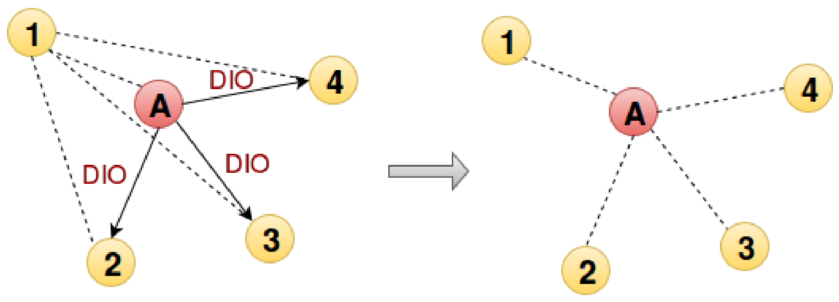


Figure 6: Decreased rank attack (DIO sent contains a better rank).

- **Blackhole** *[Topology | Isolation]* [6] : aims to drop all the packets that the malicious node is supposed to forward ; combined with a sinkhole attack, it can be very damaging as it causes the loss of the whole deflected traffic. This attack can be seen as a denial-of-service attack. If the position of the node is well chosen, it can isolate several nodes from the network. The selective forwarding attack (gray hole) is a variant of this type of attack. With this variant, it is possible to do DoS attacks but the malicious node selects the packets to forward. This attack has as consequence to disturb routing paths, it can be used to filter any protocol.

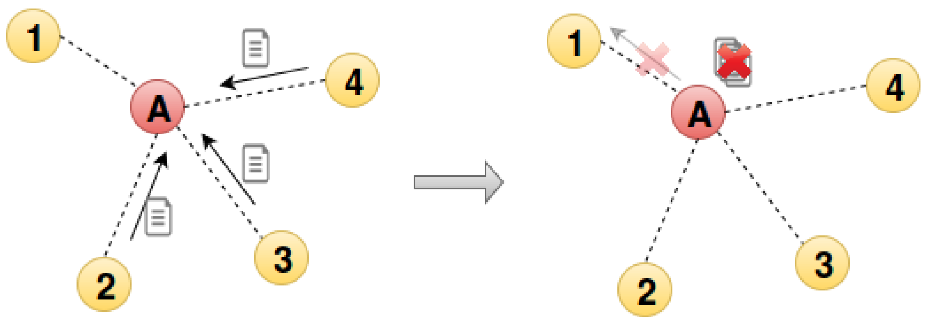


Figure 5: Blackhole attack (data received from legitimate nodes is dropped)

  Mitigation : A solution to counter this attack is to create disjoint paths between the source and the destination nodes but it is difficult to create this for the entire network. Another solution is to dynamically select the path to parents/children. There are different indicators to detect these attacks, such as rate and frequency of DIO messages, packet delivery ratio, loss percentage and delay. It is generally difficult to defend against all selective-forwarding attacks. But the use of encryption and analysis of application level traffic can be a good solution. Another solution is to make sure that the attacker will not distinguish the different types of traffic and then to force him to drop all the traffic or none.

3. Framework

This section addresses the design, implementation and usage of the framework used to test some RPL attacks, (so cutely) called *RPL Attacks Framework*.

3.1. Design Principles

The framework is built in Python upon Contiki and the Cooja simulation tool [8]. It basically automates simulation creation through a templating API provided by Jinja2. In order to make the analysis as complete and easy as possible, a **simulation** is structured as follows :

- **Simulation without the malicious node**, holding a topology with a root and a user-defined number of sensors built on a same platform (in our experiments, we choose the Zolertia Z1 by default).
- **Simulation with the malicious node**, as it tells, holding the same topology but with the malicious node that can be built with a different platform (e.g. a Skymote).

Each simulation uses a Javascript to automate data collection through Cooja's plugins. This basically collects serial messages, RPL logging, power tracking and nodes relationships. Root and sensors are compiled such that they execute a dummy application sending Echo messages marked with the source node's identifier towards the root. Malicious node's compilation can be performed using either a root or a sensor C program. Simulation execution ends with a parsing of the collected logs in order to retrieve an animated GIF of DODAG's formation.

In order to easily implement attacks, a **malicious** node can be tuned either by using an **external modified RPL library** (thus redefining code sections in the RPL library sources), by setting RPL configuration constants (only acting on the heading section of node's source code) or by modifying single lines in the standard ContikiRPL library. These last two features are gathered in what we call **building blocks**. Indeed, some attacks as shown in section 2 can be defined as parts for complex attacks, that is, the famous building blocks. So, a malicious node can hold several blocks in order to easier implement such an attack.

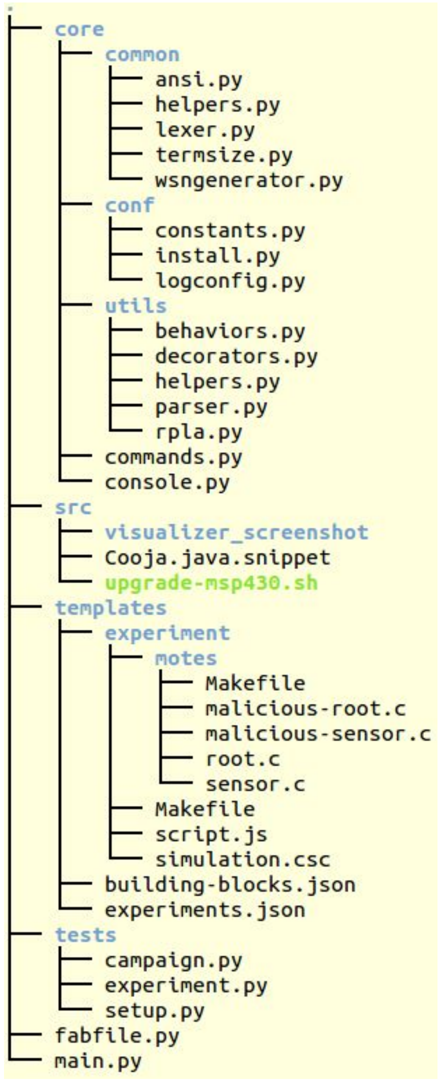
For easier mass generation of simulations, campaigns can be defined from a set of parameters gathered in a single file. This set can be partly defined in a "base" simulation allowing to reuse same parameters for all the tested attacks. This provides a very convenient way to perform our experiments.

3.2. Implementation

The framework can be used either through Fabric (a library designed to automate installations through SSH) or through the integrated console (using Python's built-in cmd module). This last interface allows to create and run multiple simulations at the same time using multi-processing.

It uses a homemade straightforward algorithm to automatically generate a WSN topology spreading sensor nodes randomly in specific areas around the root and then placing the malicious node in the surrounding of the root such that we are sure it can have a significant impact on the network. Roughly, the algorithm separates a range from the root into quadrants (that are the areas where sensor nodes' positions can be randomized) and this range can be stepped if there are too many nodes in order to generate a scattered-enough topology.

Basically, RPL Attacks Framework contains a `core` package holding a few sub-packages and the `templates` folder, structured as follows :



```
├── core
│   ├── common
│   │   ├── ansi.py
│   │   ├── helpers.py
│   │   ├── lexer.py
│   │   ├── termsize.py
│   │   └── wsngenerator.py
│   ├── conf
│   │   ├── constants.py
│   │   ├── install.py
│   │   └── logconfig.py
│   ├── utils
│   │   ├── behaviors.py
│   │   ├── decorators.py
│   │   ├── helpers.py
│   │   ├── parser.py
│   │   └── rpla.py
│   ├── commands.py
│   └── console.py
├── src
│   ├── visualizer_screenshot
│   ├── Cooja.java.snippet
│   └── upgrade-msp430.sh
├── templates
│   ├── experiment
│   │   ├── motes
│   │   │   ├── Makefile
│   │   │   ├── malicious-root.c
│   │   │   ├── malicious-sensor.c
│   │   │   ├── root.c
│   │   │   └── sensor.c
│   │   ├── Makefile
│   │   ├── script.js
│   │   └── simulation.csc
│   ├── building-blocks.json
│   └── experiments.json
├── tests
│   ├── campaign.py
│   ├── experiment.py
│   └── setup.py
├── fabfile.py
└── main.py
```

`core` holds the computation logics for creating the simulations in a modularized fashion :

- `common` contains independent modules
- `conf` contains the configuration-related code and constants specific to the framework
- `utils` contains the logics of the framework

`src` contains a bit of code for enhancing Cooja :

- a modification to add a `-hidden` option to Cooja's `jar`
- a plugin for taking screenshots of the built-in Visualizer, named *VisualizerScreenshot*
- a script for upgrading `msp430-gcc` in order to extend the supported memory of the MSP430 emulated hardware

`templates` contains :

- the famous building blocks in a JSON file
- a sample experiments campaign as a JSON file
- an `experiment` folder holding simulation (with its Javascript), nodes' templates and a Makefile for achieving compilation using Contiki

Like in every self-respecting project, `tests` are available for :

- the creation of a campaign of simulation
- the creation of a single simulation (experiment)
- the setup procedure

3.3. Usage

Details about the usage can be found at : https://github.com/dhondta/rpl-attacks

The main use case is to make and run a simulation campaign, like the following (assuming that the interactive console is used :

```
user@instant-contiki:rpl-attacks>> prepare sample-attacks
user@instant-contiki:rpl-attacks>> make_all sample-attacks
user@instant-contiki:rpl-attacks>> run_all sample-attacks
```

Note : The campaign can be customized in the corresponding JSON file in the configured experiments folder after the `prepare` command.

4. Experiments

This section presents some experiments for the attacks chosen in section 2 with their relevant results using *RPL Attacks Framework*. First, we give the implemented building blocks and the base parameters used to generate the simulations.

4.1. Building blocks

Several building blocks were implemented using the features detailed in subsection 3.1. The available ones are :

```
"hello-flood": {
  "RPL_CONF_DIS_INTERVAL": 0,
  "RPL_CONF_DIS_START_DELAY": 0,
  "rpl-timers.c": ["next_dis++;",
  "next_dis++; int i=0; while (i<10) {i++; dis_output(NULL);}"]
}
```

This uses two ContikiRPL configuration constants in order to make the malicious node immediately start sending DIS at a sustained rate.

```
"increased-version": {
   "rpl-icmp6.c": ["dag->version;", "dag->version++;"]
}
```

This modifies the related ContikiRPL's file for increasing the version number on the fly in order to trigger the global repair.

```
"decreased-rank": {
   "RPL_CONF_MIN_HOPRANKINC": 0,
   "rpl-private.h": [
    ["#define RPL_MAX_RANKINC (7 * RPL_MIN_HOPRANKINC)",
        "#define RPL_MAX_RANKINC 0"],
    ["#define INFINITE_RANK 0xffff",
        "#define INFINITE_RANK 256"]
   ],
   "rpl-timers.c": ["rpl_recalculate_ranks();", null]
}
```

This modifies :

- a ContikiRPL configuration constant to set the minimum rank hop to 0 (default is 256)
- 2 lines in `rpl-private.h`
- 1 line in `rpl-timers.c` (removal)

4.2. Simulation base parameters

Each of the simulations presented in the following subsection is based on the following parameters :

```
"BASE": {
  "simulation": {
    "number_motes": 10,
    "target": "z1",
    "Duration": 120 (seconds)
  }
}
```

Default parameters :

```
  "area-square-side": 200.0 (meters)
  "transmission-range": 50.0 (meters)
  "interference-range": 100.0 (meters)
```

As this suggests, such a configuration builds a topology of 10 sensor nodes (so, root node not included) spread around the root node in a radius of 100 meters (the are square side divided by 2) and the simulation is run during 2 (virtual) minutes. Note that, e.g. for the Version Number Attack, the simulation can take a while to finish as the attack generates a large amount of messages in the collected PCAP and consumes lots of I/O operations). Nodes are all configured to work as servers and no sensor with `RPL_LEAF_ONLY` set is then present (this has namely an important influence on the control flow of the ContikiRPL library).

Note that, thanks to *RPL Attacks Framework*'s features, it is possible to tune the WSN topology by opening a simulation in Cooja. When the simulation file is overwritten and the user exits Cooja, the framework automatically updates all simulations in the related campaign.

4.3. Attacks

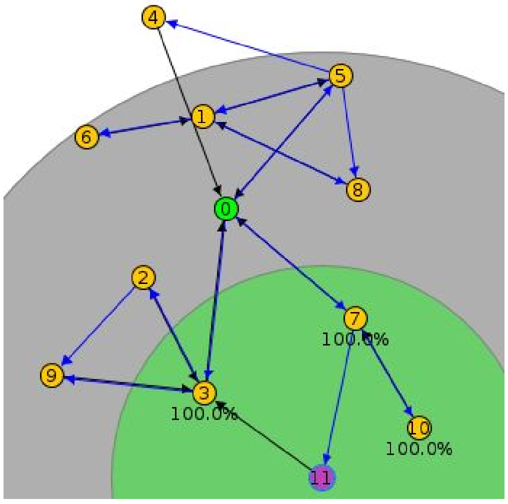
As a preliminary note, we can mention that the following attacks do not require any particular mode or settings on the attacked network, unlike various other attacks (more details in [1] in Tables 1 and 2). Also note that these attacks mainly affect the availability of all or part of the WSN. Moreover, the Version Number modification and Blackhole attacks affect the integrity.

Name	**Flooding**
Building Blocks	`hello-flood`
Working	While entering the WSN, thanks to the ContikiRPL configuration constants set with the building block, the malicious node immediately starts sending DIS messages to its neighbors, then

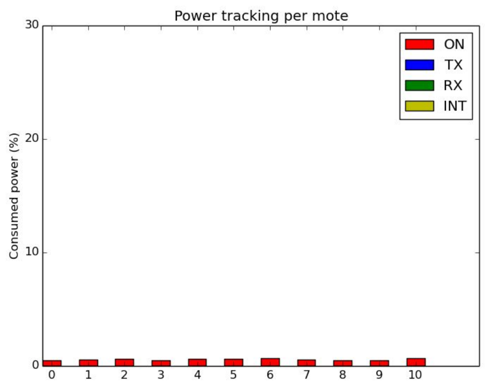
	triggering DIO messages and trickle timers reset.
Expected Impact	No change in DAG, important energy exhaustion.

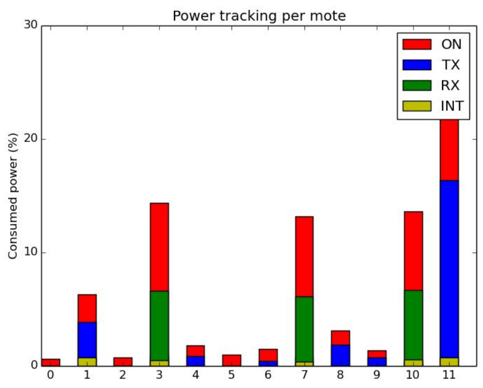
Results

After running the simulation a few (virtual) minutes, the graph looks like this :



As we can see, the malicious node (in violet) impacts nodes 3, 7 and 10. We can now illustrate the attack efficiency using this information to compare the power consumption in the simulation without (on the left) and with the malicious node (on the right).





As it can easily be observed, nodes 3, 7 and 10 are particularly impacted by the attack in terms of ON and RX times.

Important note : However, these nodes are not impacted in term of TX time. The reason is that upon the reception of a DIS, the nodes reset their trickle timers but do not immediately send a DIO, due to the multicast nature of the sent DIS.

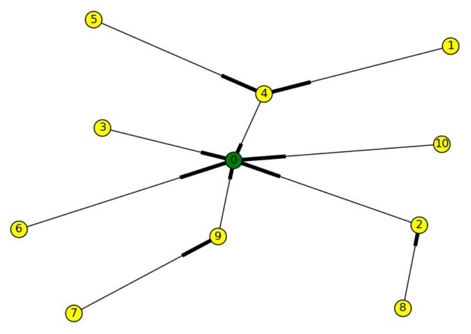
Variant of the attack : Another way of performing a flooding attack can be to unicast DIS to the neighbors, immediately triggering a DIO in response but not the trickle timer reset. This behavior can be verified in the ContikiRPL library, inside the file `rpl-icmp6.c`, in `dis_input(void)`.

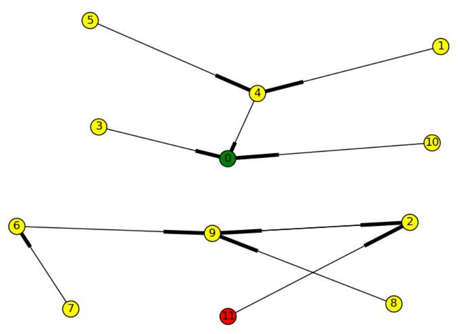
Efficiency	Local, important exhaustion of resources.

Name	**Version Number modification**
Building Blocks	`increased-version`
Working	With its modified RPL file, the malicious node increases the version number before forwarding received DIO messages, thus triggering unnecessary global repairs.
Expected Impact	No change in DAG, important energy exhaustion.

Results

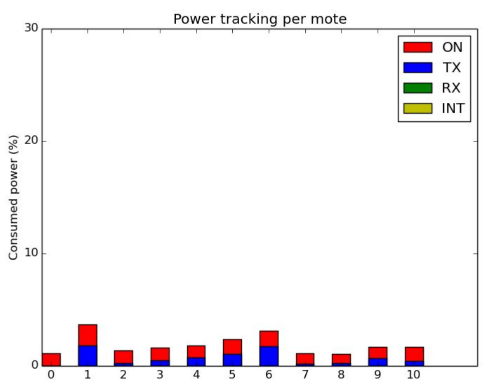
After running the simulation a few (virtual) minutes, an instant screenshot of the graph without (on the left) and with (on the right) the malicious node gives :

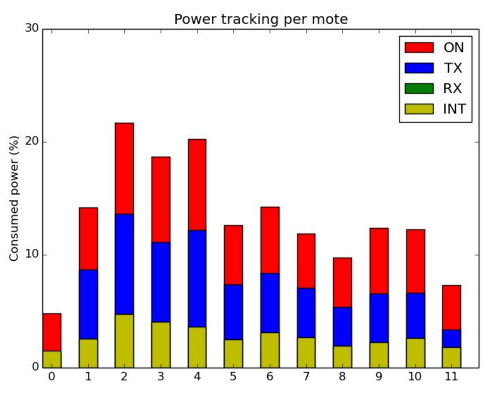




On the graph for the simulation with the malicious node, we can observe that the DAG is cut in two parts as the screenshot was taken at a time when the DAG was in global repair state. Note that the malicious node is at an end of the topology.

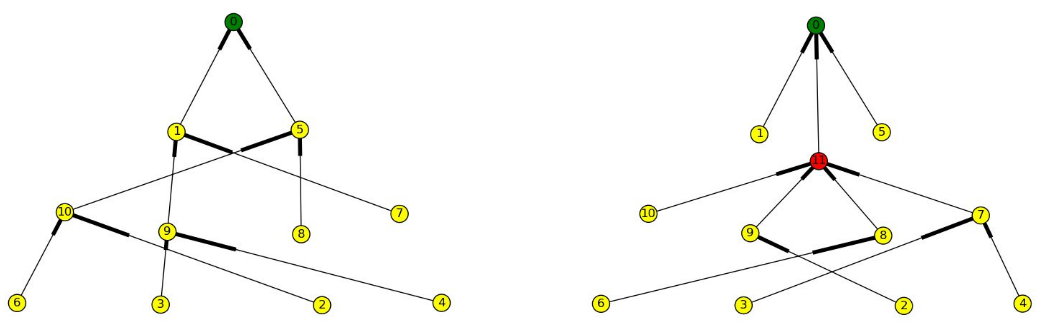
We can now illustrate the attack efficiency using this information to compare the power consumption in the simulation without (on the left) and with the malicious node (on the right).





By looking at this, we can immediately ascertain that this attack enjoys a strong efficiency on the whole network as it triggers lots of messages because of the global repair mechanism.

Efficiency	Global, dramatic exhaustion of resources and congestion in the whole network.

Name	**Decreased Rank**
Building Blocks	`decreased-rank`
Working	With the modified RPL configuration constant, the malicious node will advertise a better rank than neighbors, causing the DAG to be modified. This attack does not damage a network, however, combining with other building blocks could be very effective because it allows the attacker to suck all the traffics to him.
Expected Impact	DAG changed, legitimate nodes in the neighborhood of the malicious node have now set it as their parent.
Results	
After running the simulation a few (virtual) minutes, an instant screenshot of the graph without (on the left) and with (on the right) the malicious node gives :  On the left, we can see a typical construction of DODAG (at a time when this is maybe not stable yet and with potentially non-optimal links such as for 9-4 or 5-10). On the right side, we easily observe that the traffic is channeled through the attacker node.	
Efficiency	Potentially dramatic (i.e. for integrity), depending on the malicious node location, especially when combined with other building blocks (such as, for example, the attacks addressed in the last test hereafter).

Name	**Blackhole**
Building Blocks	`decreased-rank`, `drop-messages`
Working	The malicious node simply drops the collected application data plane messages instead of forwarding them.
Expected Impact	DAG changed, legitimate nodes in the neighborhood of the malicious node have now set it as their parent. The malicious node drops the received data plane messages.

Results	
The transformation of DODAG causes the same effect as the previous attack but also operates at the data plane by preventing hijacked messages to come to the malicious' parent node. **Variants of the attack** : - **Sinkhole** : by using `decreased-rank` and by eavesdropping traffic - **Greyhole** : by using `decreased-rank` and `selective-forwarding`	
Efficiency	Potentially dramatic (i.e. for integrity), depending on the malicious node location.

5. Conclusion

Our first goal was to *build a convenient framework for testing a malicious node into Cooja simulations*. As shown in the previous section, *RPL Attacks Framework* seems very promising as it already handles various interesting features for quickly designed and implementing malicious nodes.

Our second goal was to *test and show the effects of some chosen attacks*. Indeed, we have shown some relevant attacks, uniformly chosen amongst the presented taxonomy, and their expected results on some relevant WSN topologies.

Possible further improvements, regarding :

- Attack simulation testing :
  - (trivial) Test more attacks with new building blocks
  - Add more WSN topology generation algorithms
  - Test a malicious node with some application-level projects from other students
  - Make the simulation support multiple malicious nodes
- Framework user-friendliness :
  - Refine the console for displaying task progress
  - Add more guidance for managing experiments and campaigns
- Framework quality :
  - Refine the `.travis.yml` (for confirming that the build is passing)
  - Increase the code coverage of the tests (and get the Github badge for this)
  - Refine the documentation (and publish it on https://readthedocs.org)

References

[1] C. Karlof, D. Wagner, *Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures*, University of California at Berkeley, 2003.

[2] A. Mayzaud, R. Badonnel, I. Chrisment, *A Taxonomy of Attacks in RPL-based Internet of Things*, International Journal of Network Security, Vol.18, No.3, pp.459-473, May 2016.

[3] L. Wallgren, S. Raza, T. Voigt, *Routing Attacks and Countermeasures in the RPL-Based Internet of Things*, International Journal of Distributed Sensor Networks, Volume 2013, Article ID 794326, 11 pages, 5 June 2013.

[4] A. Mayzaud, A. Sehgal, R. Badonnel, I. Chrisment, J. Schönwälder, *A Study of RPL DODAG Version Attacks*, 8th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2014, Jun 2014, Brno, Czech Republic. pp.92-104, 2014, <10.1007/978-3-662-43862-612>. <hal-01090993>

[5] A. Dvir, T. Holczer, and L. Buttyan, *Vera - version number and rank authentication in RPL*, in Proceedings of Mobile Adhoc and Sensor Systems Conference (MASS'11), pp. 709-714, 2011.

[6] K. Chugh, A. Lasebae, J. Loo, *Case Study of a Black Hole Attack on 6LoWPAN-RPL*, SECURWARE 2012 : The Sixth International Conference on Emerging Security Information, Systems and Technologies, 2012.

[7] H. Perrey, M. Landsmann, O. Ugus, M. Wählish, T. C. Schmidt, *TRAIL: Topology Authentication in RPL*, Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks, 2016, pp.59-64.

[8] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, T. Voigt, *Cross-Level Sensor Network Simulation with COOJA*, Proceedings. 2006 31st IEEE Conference on Local Computer Networks, Tampa, FL, 2006, pp. 641-648.

[9] A. Sehgal, A. Mayzaud, R. Badonnel, I. Chrisment, J. Schönwälder, *Addressing DODAG inconsistency attacks in RPL networks*, Global Information Infrastructure and Networking Symposium (GIIS), 2014, Sep 2014, Montreal, QC, Canada. pp.1-8, 2014, <10.1109/GIIS.2014.6934253>. <hal-01090986>