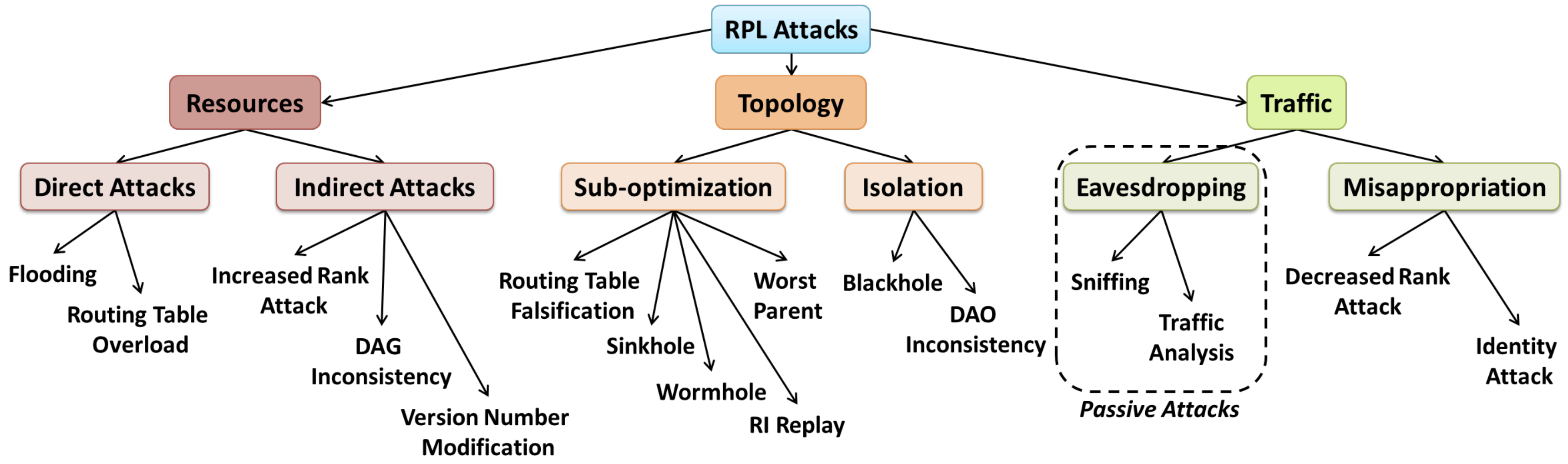
Flooding Attack Tutorial

Resource Attacks

- Resource attacks are one category of security attacks on the RPL protocol, as shown in the taxonomy below
  - Their purpose is the exhaustion of node or network resources, e.g., via an overload on power consumption, memory, etc.



Source: https://hal.inria.fr/hal-01207859/document

Resource Attacks (cont.)

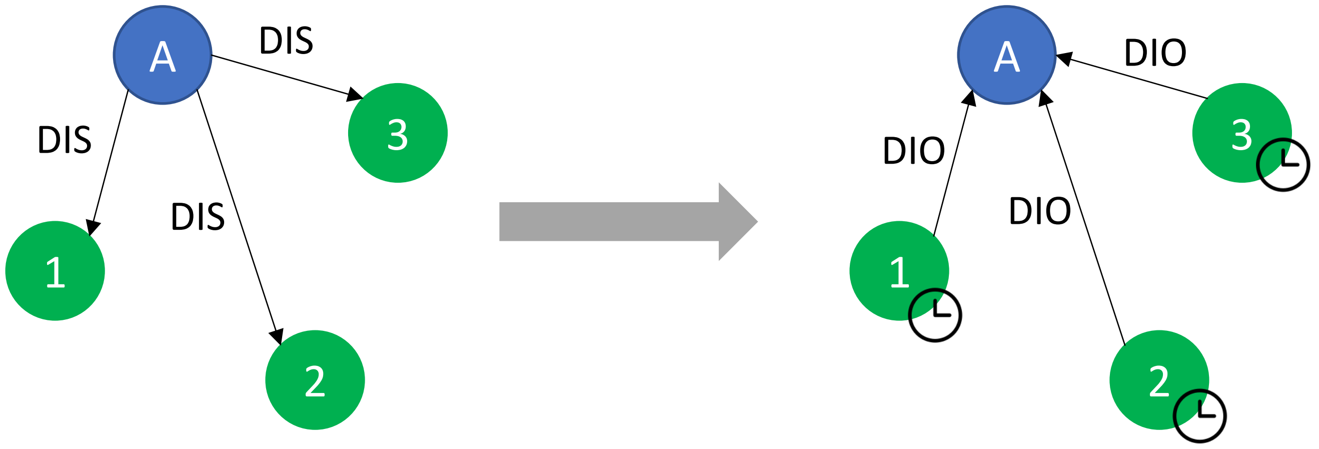
- Resource attacks can be done by forcing legitimate nodes to perform unnecessary actions to increase their use of resources
  - Have impact on the availability of the network by congesting available links or by incapacitating nodes, thus may also influence the lifetime of the network
- Direct attacks are one category of resource attacks in which the malicious node is directly responsible for the resource overload that disturbs the network
  - Typical attack mechanisms are network flooding and execution overloading (e.g., regarding routing tables)

Flooding Attacks

- Flooding attacks consist in generating a large amount of traffic in the network with the purpose of making nodes and links unavailable
  - In the worst case, such attacks can exhaust the resources of all the network nodes
- Flooding attacks can be performed both by external or internal attackers
- When using solicitation messages to perform the flooding, the attack is called a “HELLO flood attack”

Flooding Attacks (cont.)

- To perform flooding attacks in RPL networks, an attacker can
  - Broadcast DIS (DODAG Information Solicitation) messages to neighboring nodes, which must reset their trickle timer
  - Or send unicast DIS messages to each node, which must reply with a DIO (DODAG Information Object) message



Flooding attack example: node A is malicious; nodes 1, 2, 3 are legitimate

Flooding Attack Simulation

Flooding Attack Simulation

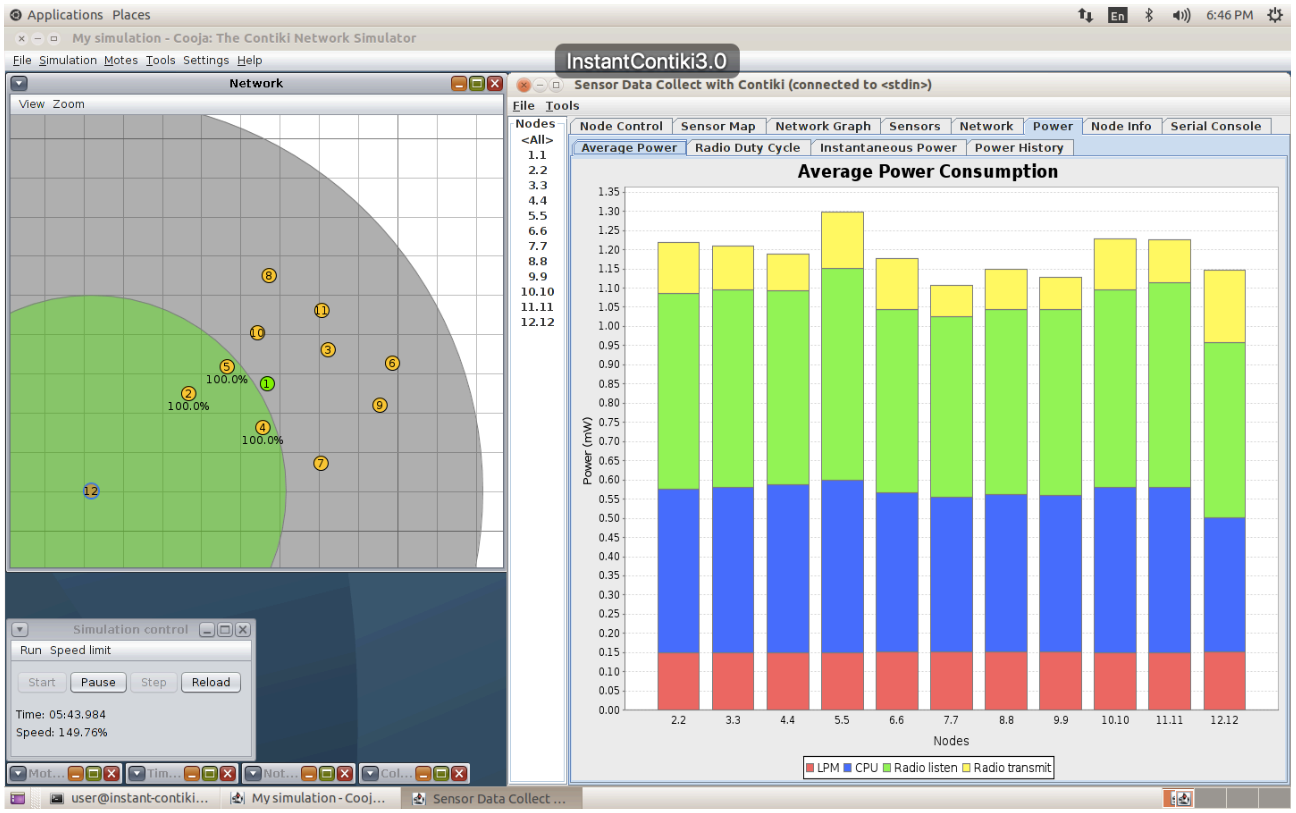
- Open the desired simulation in Cooja by selecting the corresponding scenario via the IoTrain-Sim interface
    - We recommend that you first select the “Reference Scenario Simulation” entry to view the reference scenario
- Alternatively, the simulations can be opened manually as follows
    - In Cooja, select the menu File > Open simulation > Browse…
    - Go to the folder “iotrain-sim/database/security_training/ flooding_attack/simulation/”
    - Select “flooding_attack-reference.csc” for the reference scenario, and click “Open”

Flooding Attack Simulation (cont.)

- Simulation and data collection procedure
  1. In the CollectView window, click on the "Start Collect" button, then click on the "Send command to nodes" button
  2. In the Simulation control window of Cooja, click on the "Start" button to begin the simulation
  3. Wait for at least two minutes of simulation time
  4. Back in the CollectView window, go to the Power tab and see the Average Power plot for the scenario
- Follow the same procedure to perform the attack simulation and compare the results
  - The attack scenario can be opened via the menu "Flooding Attack Simulation" in IoTrain-Sim, or directly in Cooja via the file "flooding_attack-simulation.csc"
  - You may need to wait for more than five minutes of logical simulation time to get statistics for all the nodes

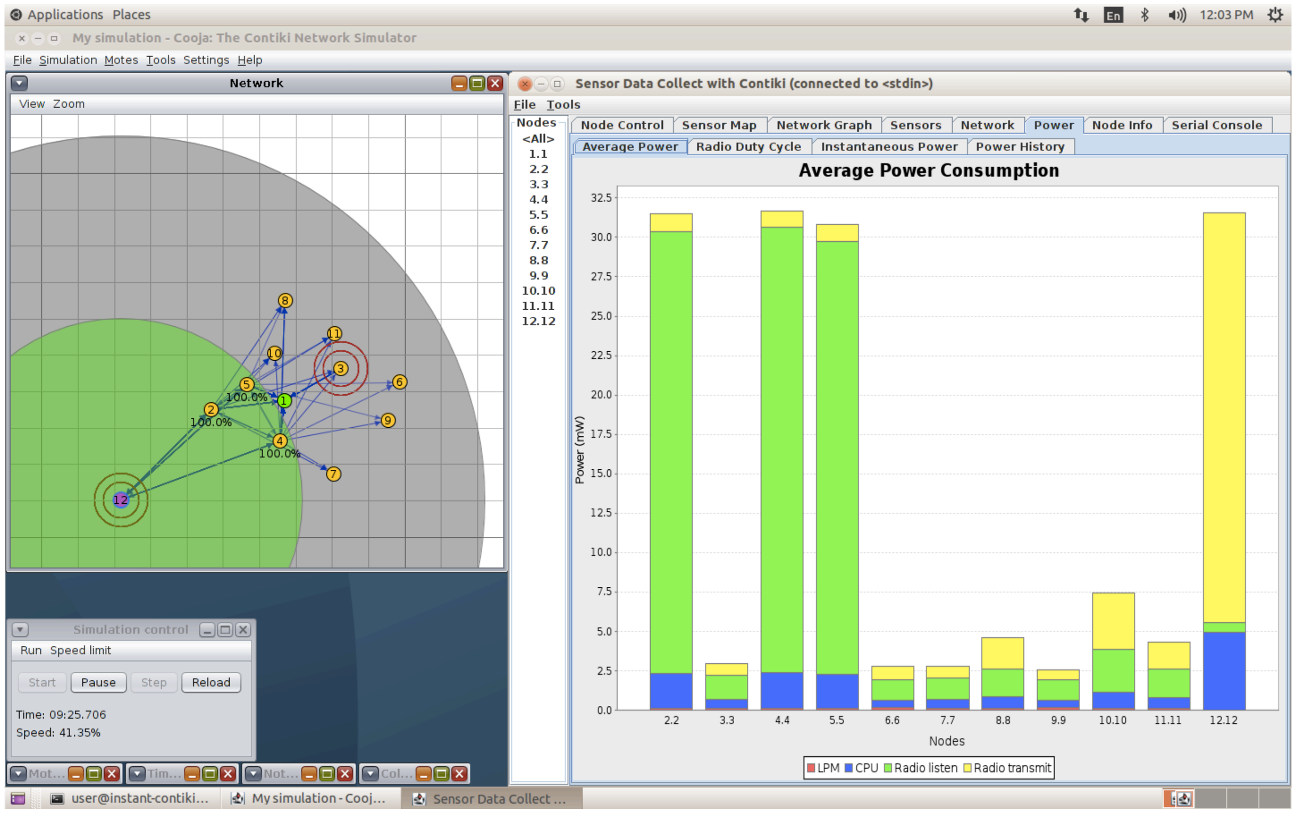
Reference Scenario and Results

- Node 1 (green color) is a sink node that acts as a border router; the other nodes are sender nodes that act as normal sensors
- Notice that nodes 2, 4 and 5 are all in the range of node 12



Attack Simulation and Results

- Node 1 and the nodes in yellow color have the same roles as before
- Node 12 became a malicious node performing a flooding attack, with direct effects on nodes 2, 4 and 5



Discussion

- Reference scenario
  - All the sender nodes (2 to 12) have nearly the same power consumption, which is at a low level of around 1.2 mW
- Attack simulation
  - Compared with other sender nodes, node 2, 4, 5 and 12 have significantly higher power consumption, of around 30 mW; the other sender’ power consumption is also higher than before, at around 2.5 mW or even more
  - For node 12, radio transmit represents a large proportion of the power consumption, as it continuously sends messages to nodes 2, 4, and 5
  - For nodes 2, 4 and 5, radio listen represents a large proportion of the power consumption, as they continuously receive messages from node 12

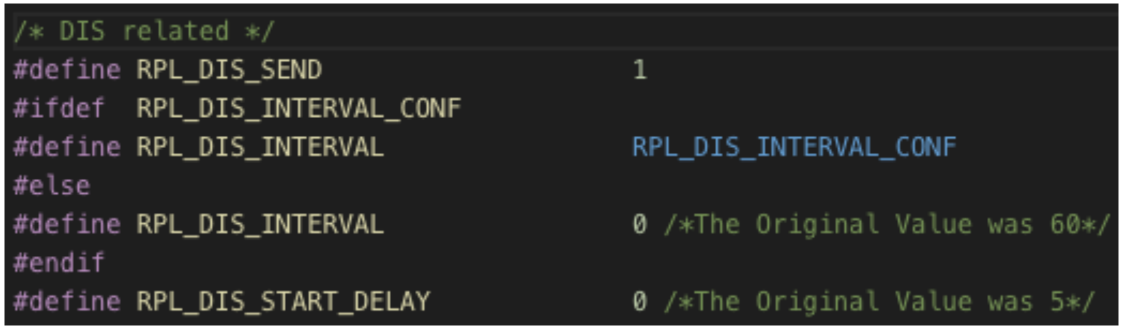
Flooding Attack Implementation

Implementation Overview

- To implement the flooding attack, some changes are necessary to the normal source code for the RPL implementation in Contiki
- The files to be modified are located in the directory "contiki/core/net/rpl/"
  - rpl-private.h, which contains private declarations for the Contiki RPL implementation, such as the default values for the ICMP control messages and timers associated to them, modes of operation, DAG routing tables, etc.
  - rpl-timers.c, which is the RPL timer management implementation in Contiki

Changes to rpl-private.h

- The file “rpl-private.h” includes several constants related to DIS operation
- The flooding attack can be implemented by decreasing the value of the DIS message interval and the DIS start delay, to force the malicious mote to send DIS messages as quickly as possible
  - We set both values to 0 for a maximum effect

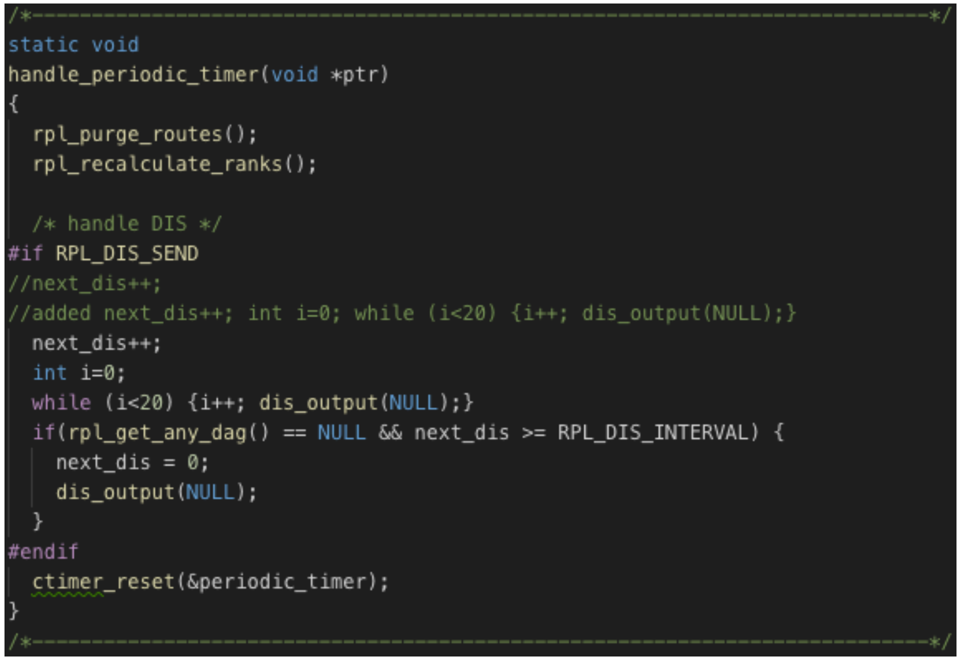


```
/* DIS related */
#define RPL_DIS_SEND                    1
#ifdef  RPL_DIS_INTERVAL_CONF
#define RPL_DIS_INTERVAL                RPL_DIS_INTERVAL_CONF
#else
#define RPL_DIS_INTERVAL                0 /*The Original Value was 60*/
#endif
#define RPL_DIS_START_DELAY             0 /*The Original Value was 5*/
```

Flooding attack implementation via the modification of RPL_DIS_INTERVAL and RPL_DIS_START_DELAY

Changes to rpl-timers.c

- The file “rpl-timers.c” includes functions related to the way in which DIS message timers are handled
- A loop is added in the periodic timer function to send a total of 20 DIS messages without any conditions



```
/*---------------------------------------------------------------------------*/
static void
handle_periodic_timer(void *ptr)
{
  rpl_purge_routes();
  rpl_recalculate_ranks();

  /* handle DIS */
#if RPL_DIS_SEND
//next_dis++;
//added next_dis++; int i=0; while (i<20) {i++; dis_output(NULL);}
  next_dis++;
  int i=0;
  while (i<20) {i++; dis_output(NULL);}
  if(rpl_get_any_dag() == NULL && next_dis >= RPL_DIS_INTERVAL) {
    next_dis = 0;
    dis_output(NULL);
  }
#endif
  ctimer_reset(&periodic_timer);
}
/*---------------------------------------------------------------------------*/
```

Flooding attack implementation via an artificial increase in the number of DIS messages

Exercises

- After making the suggested modifications in a copy of the Contiki source code, compile the files and assign the resulting malicious firmware to one of the motes in the reference scenario*

1. We suggest you first use node 12 as malicious node, as in our example, then change the malicious node to another one, to see how the results change
2. You can also use multiple malicious nodes and compare the simulation results
3. Another exercise is to change some of the attack code parameters, such as the loop count in rpl-timers.c, and see the effects of this change on the results

* See “Security Training Tutorial” for an explanation of the procedure