

## Sink Hole Attack using RPL in IOT

**Software:** NetSim Standard v13.2, Visual Studio 2022

### Project Download Link:

[https://github.com/NetSim-TETCOS/Sink\\_Hole\\_Attack\\_in\\_IoT\\_RPL\\_v13.2/archive/refs/heads/main.zip](https://github.com/NetSim-TETCOS/Sink_Hole_Attack_in_IoT_RPL_v13.2/archive/refs/heads/main.zip)

Follow the instructions specified in the following link to download and setup the Project in NetSim:

<https://support.tetcos.com/en/support/solutions/articles/14000128666-downloading-and-setting-up-netsim-file-exchange-projects>

### Introduction

In sinkhole Attack, a compromised node or malicious node advertises fake rank information to form the fake routes, After receiving the message packet, it drops the packet information.

Sinkhole attacks affect the performance of IoT networks protocols such as RPL protocol.

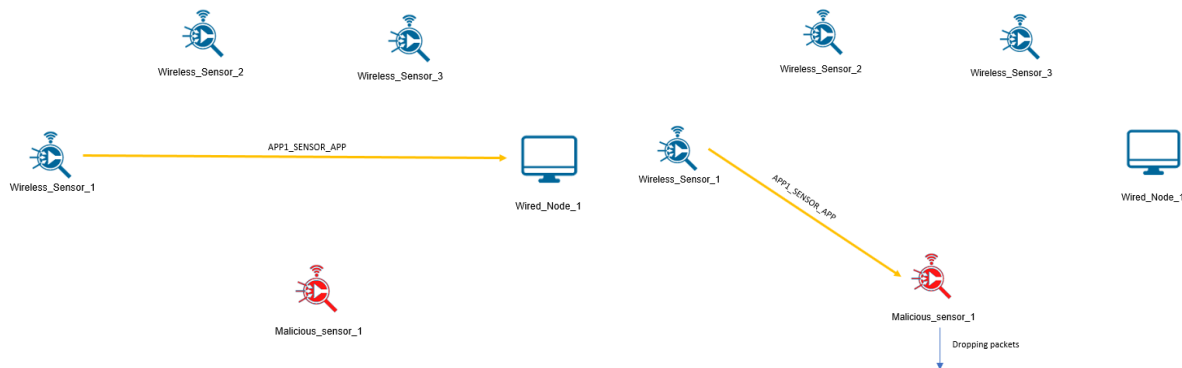


Figure 1: network configuration of how the traffic flow is configured

Figure 2 Network configuration of actual traffic flow along with the working of malicious node

### Implementation in RPL (for 1 sink)

- In RPL the transmitter broadcasts the DIO during DODAG formation.
- The receiver on receiving the DIO from the transmitter updates its parent list, sibling list, rank and sends a DAO message with route information.
- Malicious node upon receiving the DIO message it does not update the rank instead it always advertises a fake rank.
- The other node on listening to the malicious node DIO message the update their rank according to the fake rank.
- After the formation of DODAG, if the node that is transmitting the packet has malicious node as the preferred parent, transmits the packet to it but the malicious node instead of transmitting the packet to its parent, it simply drops the packet resulting in zero throughput.

A file **Malicious.c** is added to the RPL project. The file contains the following functions.

- **fn\_NetSim\_RPL\_MaliciousNode();** //This function is used to identify whether a current device is malicious or not in-order to establish malicious behavior.
- **fn\_NetSim\_RPL\_MaliciousRank();** //This function is used to give a fake rank to the malicious node.
- **rpl\_drop\_msg();** //This function is used to drop the packet by the malicious node if it enters into its network layer.
- **Fn\_NetSim\_RPL\_FreePacket();** // This function is used inside **rpl\_drop\_msg()** for dropping the packets.
- **Sink Hole Attack** -The malicious node advertises the fake rank **fn\_NetSim\_RPL\_MaliciousRank();** is the sink hole attack function.
- **Black Hole Attack:** The malicious node drops the packet, **rpl\_drop\_msg()** is the black hole attack function

You can set any device as malicious, and you can have more than one malicious node in a scenario. Device id's of malicious nodes can be set inside the **fn\_NetSim\_RPL\_MaliciousNode()** function.

### Example

1. The **Workspace\_SinkHole\_Attack\_RPL** comes with a sample network configuration that are already saved. To open this example, go to Your work in the home screen of NetSim and click on the **SinkHole\_Attack\_in\_RPL\_Example** from the list of experiments.
2. The saved network scenario consists of
  - a. **5** Wireless Sensors
  - b. **1** 6\_LOWPAN Gateway
  - c. **1** Router
  - d. **1** Wired Node

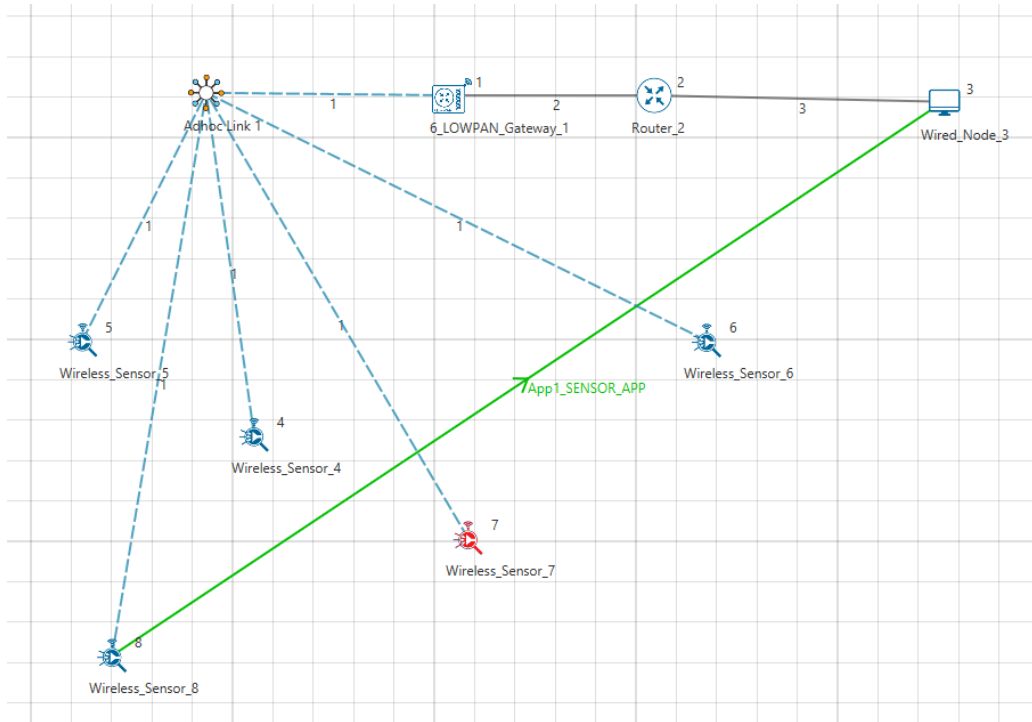


Figure 3: IoT Network Topology

3. Channel Characteristics: **Pathloss Only**, Pathloss Model: **Log Distance**, Pathloss Exponent: **2**
4. Run the simulation for 100 Seconds.

### Results and discussion

Open **rplog.txt** file from the results dashboard window, then you will find the information about DODAG formation. For every DODAG, 6LoWPAN Gateway is the root of the DODAG.

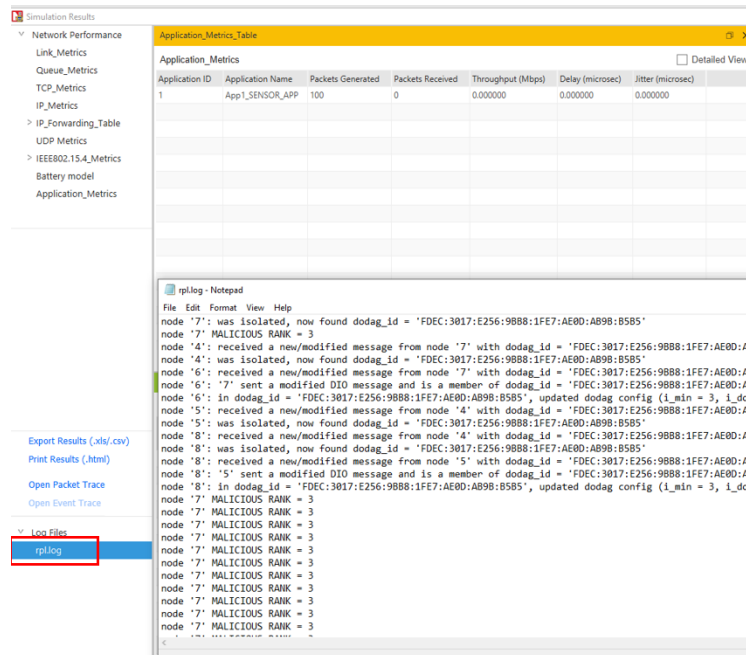


Figure 4: Result Dashboard Window

- Root is 1 with rank = 1 (Since the Node Id\_1 is always 6LoWPAN Gateway)
- Wireless\_Sensor\_Node\_7 (Malicious Node)
- Packet is 'transmitted' by **node 8(Sensor\_8)** is 'received' by **node 7(Sensor\_7)** since the node 7 is **malicious node** it drops the packet. So, the Throughput in this scenario is 0.
- Open packet trace file from simulation results window and filter the **control packet Type/App Name** to **App1\_Sensor\_App**.
- Check the data packets flow, the Transmitter\_Id and receiver\_Id column. Since the node 7 is malicious node, it drops the packet without forwarding it further.

	A	B	C	D	E	F	G	H
1	PACKET_ID	SEGMENT_ID	PACKET_TYPE	CONTROL_PACKET_TYPE/APP_NAME	SOURCE_ID	DESTINATION_ID	TRANSMITTER_ID	RECEIVER_ID
92	2	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-4
93	2	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-4	SENSOR-7
113	3	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-4
114	3	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-4	SENSOR-7
124	4	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
125	4	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-4	SENSOR-7
143	5	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-4
144	5	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-4	SENSOR-7
154	6	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-4
155	6	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-4	SENSOR-7
163	7	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-4
164	7	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-4	SENSOR-7
172	8	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-4
173	8	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-4	SENSOR-7
190	9	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-4
191	9	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-4	SENSOR-7
199	10	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-4
200	10	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-4	SENSOR-7
218	11	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-4
220	11	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-4	SENSOR-7
230	12	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-4
231	12	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-4	SENSOR-7
239	13	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-4

Figure 5: NetSim Packet Trace

### Introducing multiple malicious nodes:

To introduce the multiple malicious nodes in the network, consider a larger network consisting of more of sensors and with multiple sensor devices generating traffic. Malicious nodes can be distributed in different locations of the network and their impact on the network can be analyzed.

1. Add one more sensor i.e., Sensor\_9 for the similar scenario and create traffic as shown below.

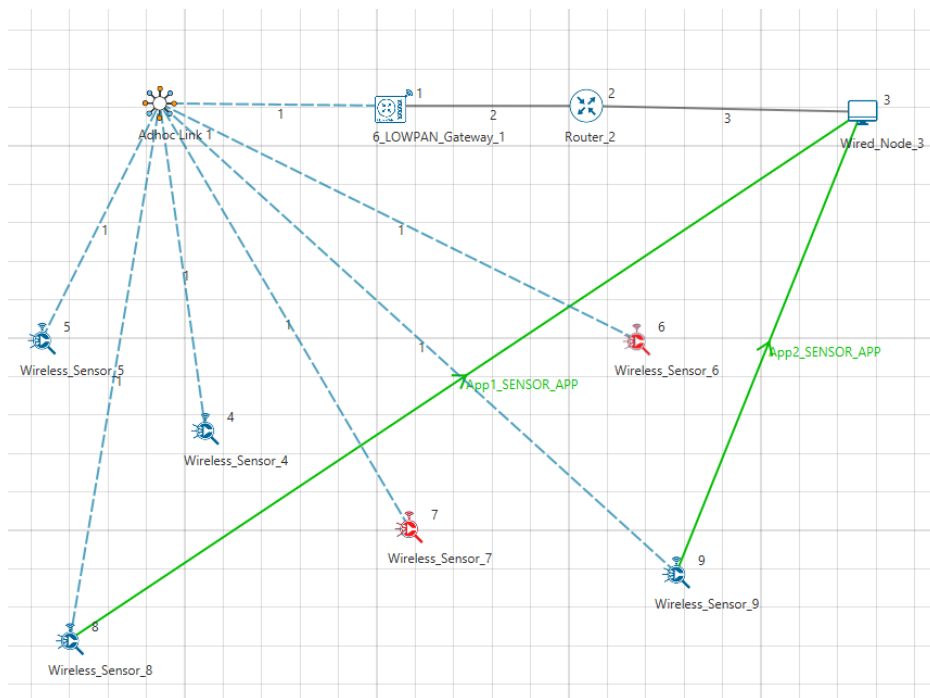
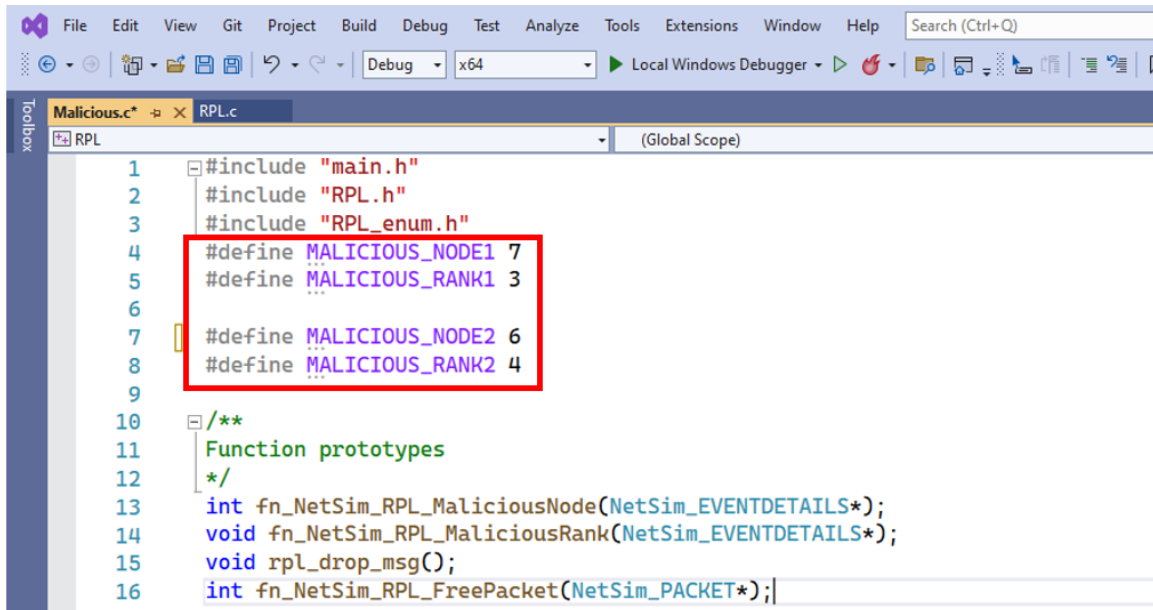


Figure 6: IoT Network Topology for multiple malicious nodes

2. Make sure that the Routing protocol in the added sensor is same as the network configured.

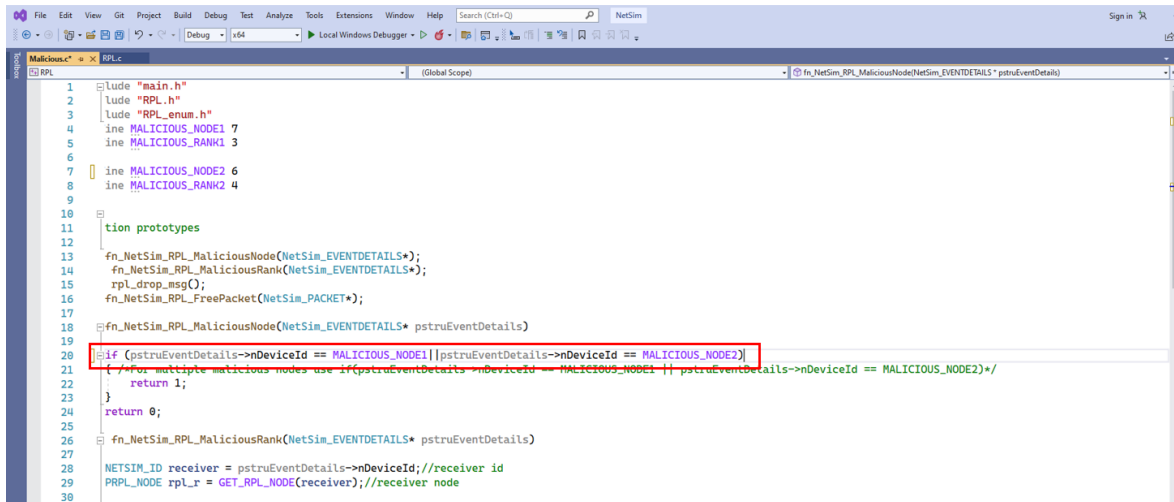
3. Consider sensor 6 and 7 as malicious nodes with fake rank by defining it in the Malicious.c file as shown below.



```
1 #include "main.h"
2 #include "RPL.h"
3 #include "RPL_enum.h"
4 #define MALICIOUS_NODE1 7
5 #define MALICIOUS_RANK1 3
6
7 #define MALICIOUS_NODE2 6
8 #define MALICIOUS_RANK2 4
9
10 /**
11  * Function prototypes
12  */
13 int fn_NetSim_RPL_MaliciousNode(NetSim_EVENTDETAILS*);
14 void fn_NetSim_RPL_MaliciousRank(NetSim_EVENTDETAILS*);
15 void rpl_drop_msg();
16 int fn_NetSim_RPL_FreePacket(NetSim_PACKET*);
```

Figure 7: Defining malicious nodes in Malicious.c file

4. In `fn_NetSim_RPL_MaliciousNode()` function, the if condition for checking malicious nodes needs to be updated.



```
1 #include "main.h"
2 #include "RPL.h"
3 #include "RPL_enum.h"
4 #define MALICIOUS_NODE1 7
5 #define MALICIOUS_RANK1 3
6
7 #define MALICIOUS_NODE2 6
8 #define MALICIOUS_RANK2 4
9
10 /**
11  * Function prototypes
12  */
13 fn_NetSim_RPL_MaliciousNode(NetSim_EVENTDETAILS*);
14 fn_NetSim_RPL_MaliciousRank(NetSim_EVENTDETAILS*);
15 rpl_drop_msg();
16 fn_NetSim_RPL_FreePacket(NetSim_PACKET*);
17
18 fn_NetSim_RPL_MaliciousNode(NetSim_EVENTDETAILS* pstruEventDetails)
19 {
20     if (pstruEventDetails->nDeviceId == MALICIOUS_NODE1 || pstruEventDetails->nDeviceId == MALICIOUS_NODE2)
21     {
22         /* For multiple malicious nodes use if(pstruEventDetails->nDeviceId == MALICIOUS_NODE1 || pstruEventDetails->nDeviceId == MALICIOUS_NODE2) */
23         return 1;
24     }
25     return 0;
26 }
27
28 fn_NetSim_RPL_MaliciousRank(NetSim_EVENTDETAILS* pstruEventDetails)
29 {
30     NETSIM_ID receiver = pstruEventDetails->nDeviceId; // receiver id
31     PRPL_NODE rpl_r = GET_RPL_NODE(receiver); // receiver node
32 }
```

Figure 8: If condition for checking multiple malicious nodes

5. Now right click on Solution explorer and select Rebuild.

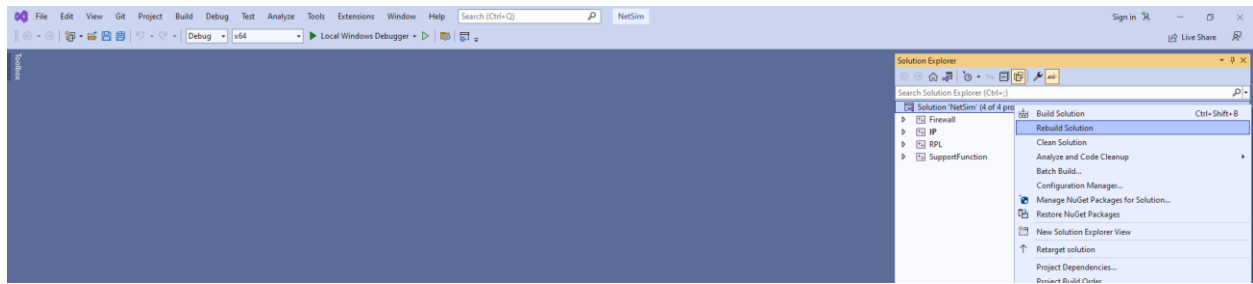


Figure 9: solution Explorer rebuild

## Results and discussion

Sensor 8 will consider sensor 7 as a parent and sensor 9 will consider sensor 6 as parent instead of sensor 4 since sensor 6 advertises lower rank compared to sensor 4. Packets reach sensors 7 and 6 get dropped. Results can be visualized in the rpllog.txt and packet trace.

You can also check the distribution of ranks with the help of DODAG visualizer-

<https://support.tetcos.com/support/solutions/articles/14000134056-how-to-visualize-the-rpl-dodag-in-netsim-iot-simulations->

The DoDAG plots appear vertically flipped when compared to the network topology in NetSim since the origin (0,0) is at the top left in NetSim whereas it is in the bottom left in the plot window.

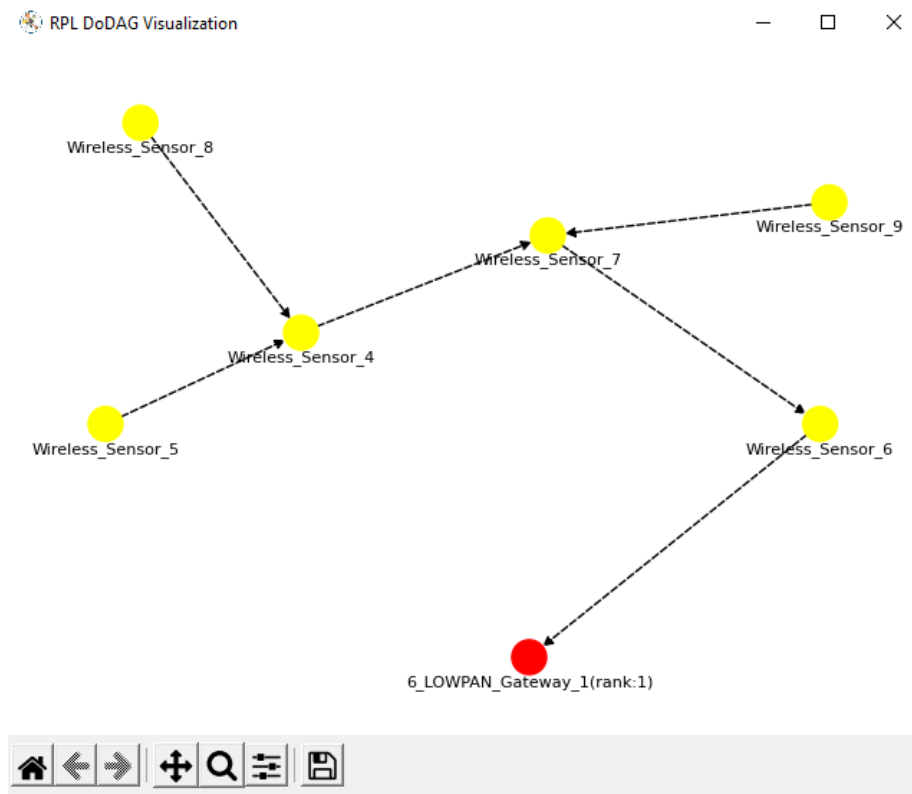


Figure 10: RPL DODAG Visualizer

**Note: Conditions for Malicious node to be able to attract other legitimate nodes:**

- The malicious node should be within the range of other nodes.
- The malicious nodes' DIO broadcast should be received by other nodes with a rank lower than other DIO messages received.

## Appendix: NetSim source code modifications

---

Set malicious node id and the fake Rank in **Malicious.c** file which is present under **RPL** project

---

```
#include "main.h"
#include "RPL.h"
#include "RPL_enum.h"
#define MALICIOUS_NODE1 7
#define MALICIOUS_RANK1 3

#define MALICIOUS_NODE2 6
#define MALICIOUS_RANK2 4
```

---

Code changes done in **fn\_NetSim\_RPL\_Run()**, in **RPL.c** file, within RPL project

---

```
_declspec (dllexport) int fn_NetSim_RPL_Run()
{
    switch (pstruEventDetails->nEventType)
    {
        case NETWORK_OUT_EVENT:
        {
        }
        break;
        case NETWORK_IN_EVENT:
        {
            rpl_add_to_neighbor_list();
            if (is_rpl_control_packet(pstruEventDetails->pPacket))
            {
                if (fn_NetSim_RPL_MaliciousNode(pstruEventDetails))
                    fn_NetSim_RPL_MaliciousRank(pstruEventDetails);
                else
                {
                    rpl_process_ctrl_msg();
                    fn_NetSim_Packet_FreePacket(pstruEventDetails->pPacket);
                    pstruEventDetails->pPacket = NULL;
                }
            }
            else if (pstruEventDetails->nPacketId &&
fn_NetSim_RPL_MaliciousNode(pstruEventDetails))
            {
                rpl_drop_msg();
            }
        }
        break;
    }
}
```