Sink Hole Attack using RPL in IOT

Software: NetSim Standard v14.1, Visual Studio 2022

Project Download Link:

https://github.com/NetSim-TETCOS/Sinhole-attack-in-IoT-RPL-v14.1/archive/refs/heads/main.zip

Follow the instructions specified in the following link to download and set up the Project in NetSim:
https://support.tetcos.com/en/support/solutions/articles/14000128666-downloading-and-setting-up-netsim-file-exchange-projects

Introduction:

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

Real-World Context:

Consider an Industrial IoT (IIoT) environment in a manufacturing plant where various sensors are deployed to monitor and control critical equipment. These sensors communicate with a central gateway using the RPL routing protocol. When one node in this network is malicious, the impact can be significant

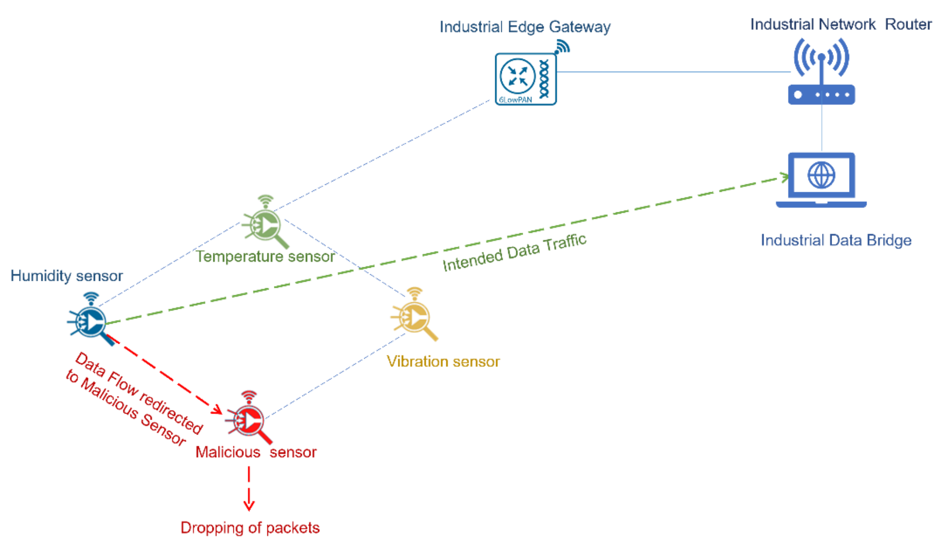


Figure 1:Real world setup for Sinkhole Attack in IOT

Sinkhole Attack Overview:

- Sensors broadcast DIO messages during DODAG formation, containing rank, parent list, and sibling list information.

- A malicious sensor broadcasts fake ranks, falsely presenting itself as a preferred parent to attract data traffic.
- As other sensors, such as the humidity, pressure, and vibration sensors, receive the malicious sensor's DIO messages, they update their routing information based on the fake rank advertised. This means they start considering the malicious sensor as a preferred parent for data transmission.
- Once DODAG is established, the malicious sensor intercepts and discards data packets instead of forwarding them, disrupting the normal routing process.

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

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 sinkhole attack function.
- **Black Hole Attack:** The malicious node drops the packet**, rpl_drp_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 IDs of malicious nodes can be set inside the **fn_NetSim_RPL_MaliciousNode()** function.

Example

1. The **Sinkhole-attack-in-IoT-RPL-Workspace** comes with a sample network configuration that is already saved. To open this example, go to Your work in the home

screen of NetSim and click on the Sink_Hole_Attack_one_Malicious from the list of experiments.

2. The saved network scenario consists of
    a. **5** Wireless Sensors
    b. **1** LOWPAN Gateway
    c. **1** Router
    d. **1** Wired Node

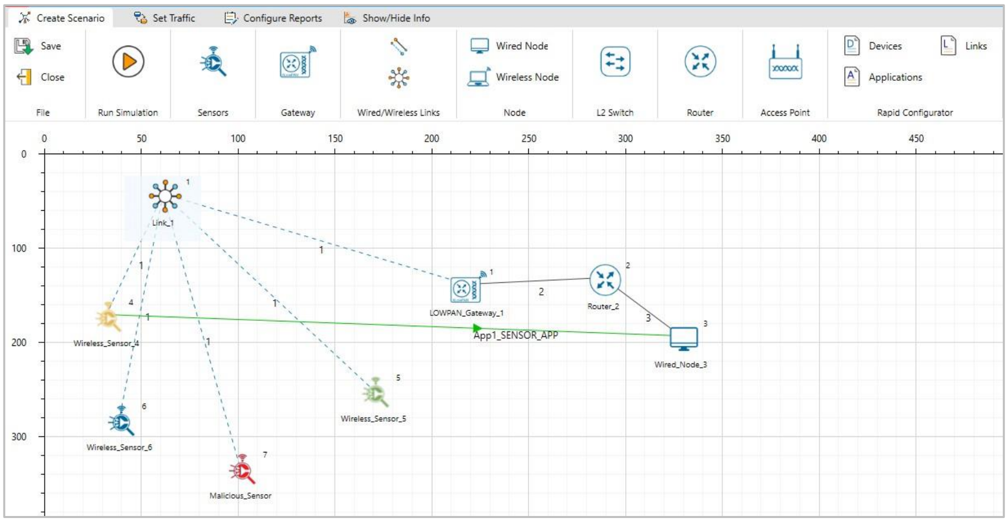


Figure 2:Network Setup of Sinkhole Attack in IOT RPL

3. In Ad-hoc link set the Channel Characteristics: **Pathloss Only,** Pathloss Model: **Log Distance,** Pathloss Exponent: **2**
4. Enable the packet trace on the top ribbon and enable the Wireshark on all the devices
5. Run the simulation for 100 Seconds.

Results and discussion:

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

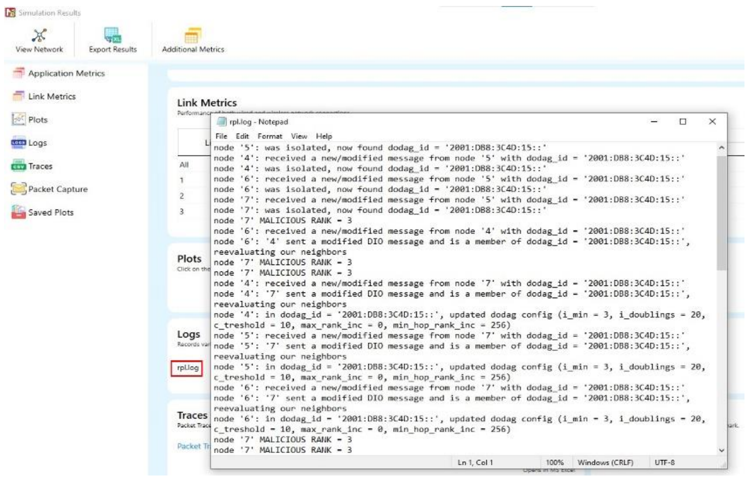


Figure 3: Result Dashboard Window

- Root is 1 with rank = 1 (Since the Node Id_1 is always LoWPAN Gateway)
- Wireless_Sensor_Node_7 (Malicious Node)
- Packet is 'transmitted' by **node 8(Sensor_8)** and is 'received' by **node 7(Sensor_7)** since node 7 is a **malicious node** it drops the packet. So, the Throughput in this scenario is 0.
- Open the packet trace file from the 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 node 7 is a malicious node, it drops the packet without forwarding it further.

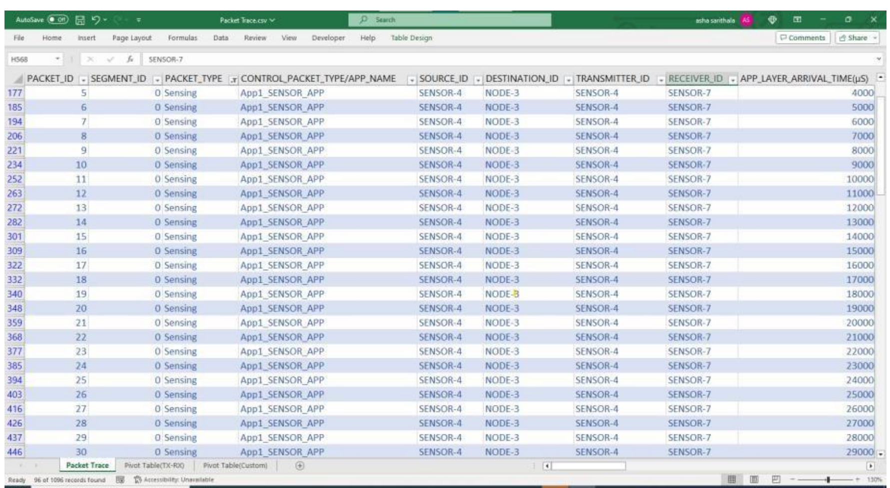


Figure 4: NetSim Packet Trace Window

Introducing multiple malicious nodes:

To introduce the multiple malicious nodes in the network, consider a larger network consisting of more 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 analysed.

1. Add one more sensor i.e., Sensor_9 for a similar scenario and create traffic as shown below.

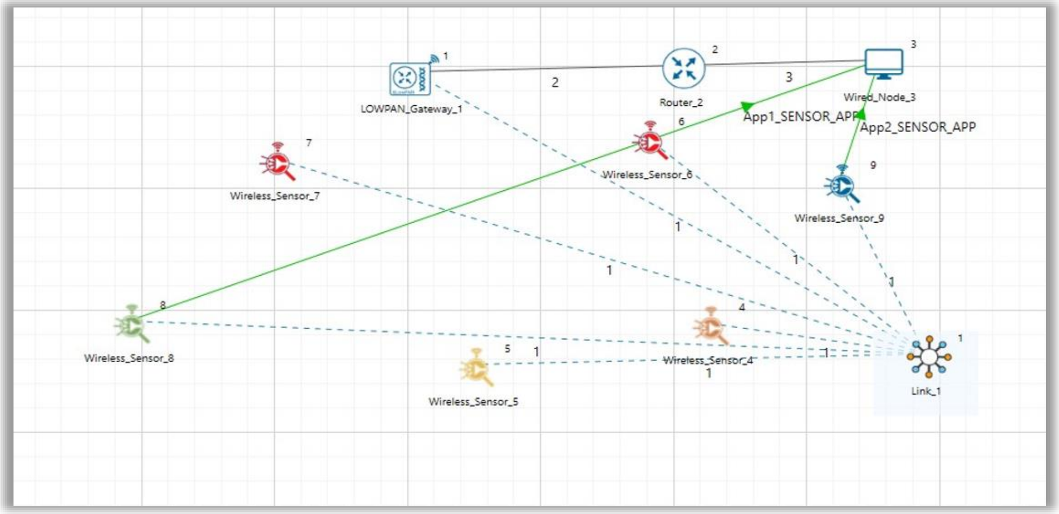


Figure 5:IoT Network Topology for multiple malicious nodes

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

2. Consider sensors 6 and 7 as malicious nodes with fake rank by defining them in the Malicious.c at RPL Project file as shown below.

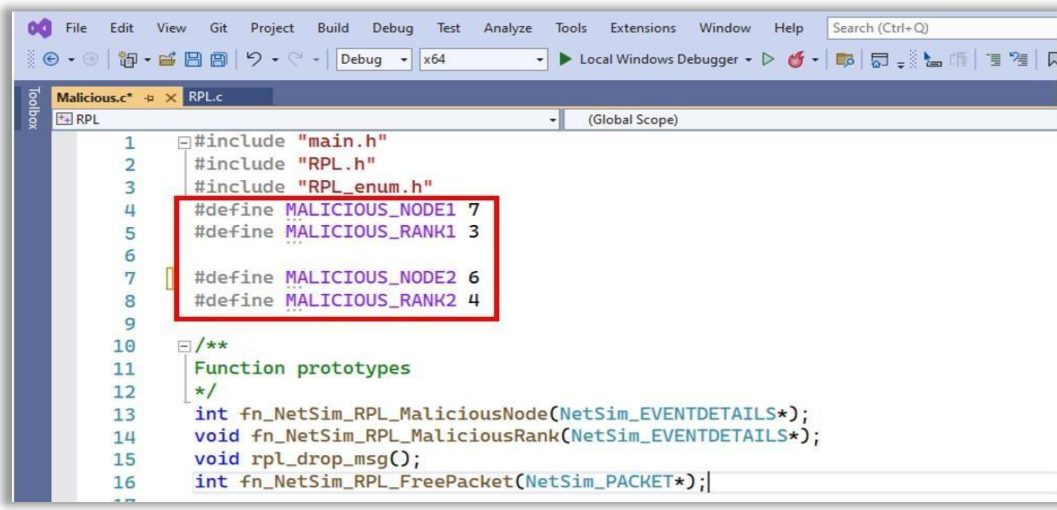


Figure 6: Defining malicious nodes in Malicious.c file

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

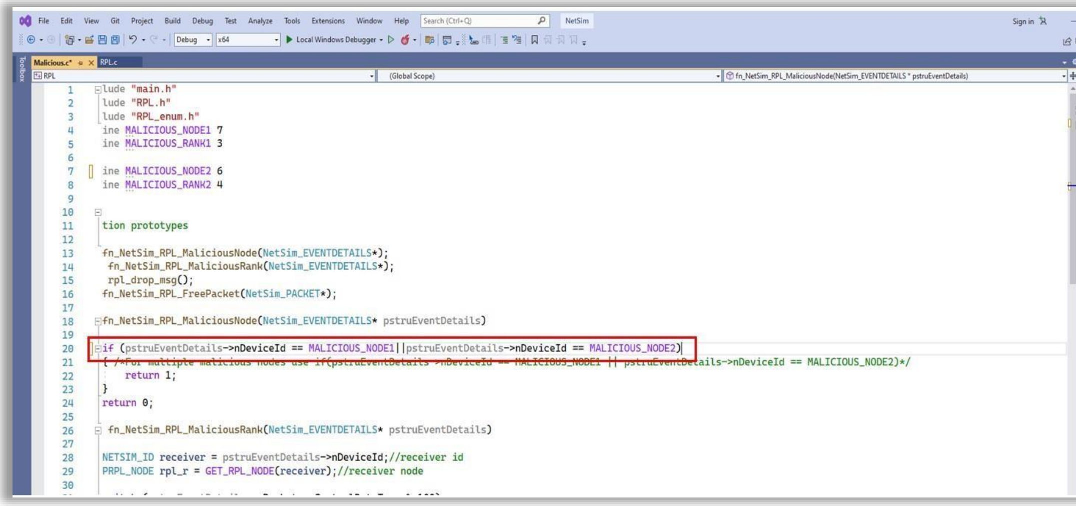


Figure 7:If condition for checking multiple malicious nodes 5.

Now right click on Solution explorer and select Rebuild.

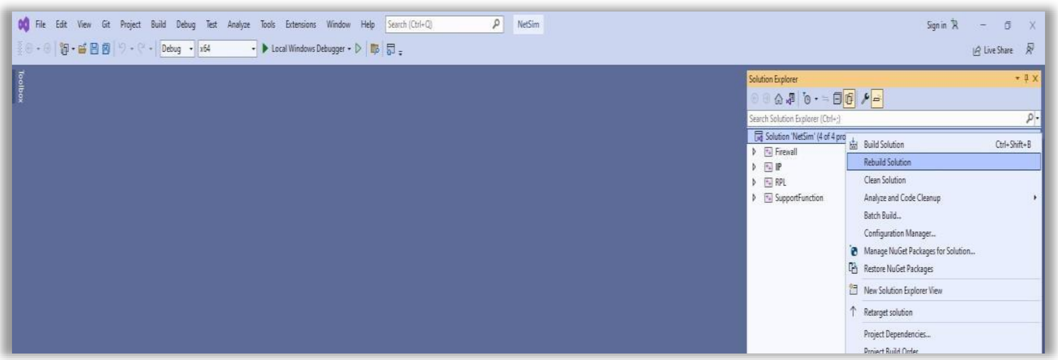
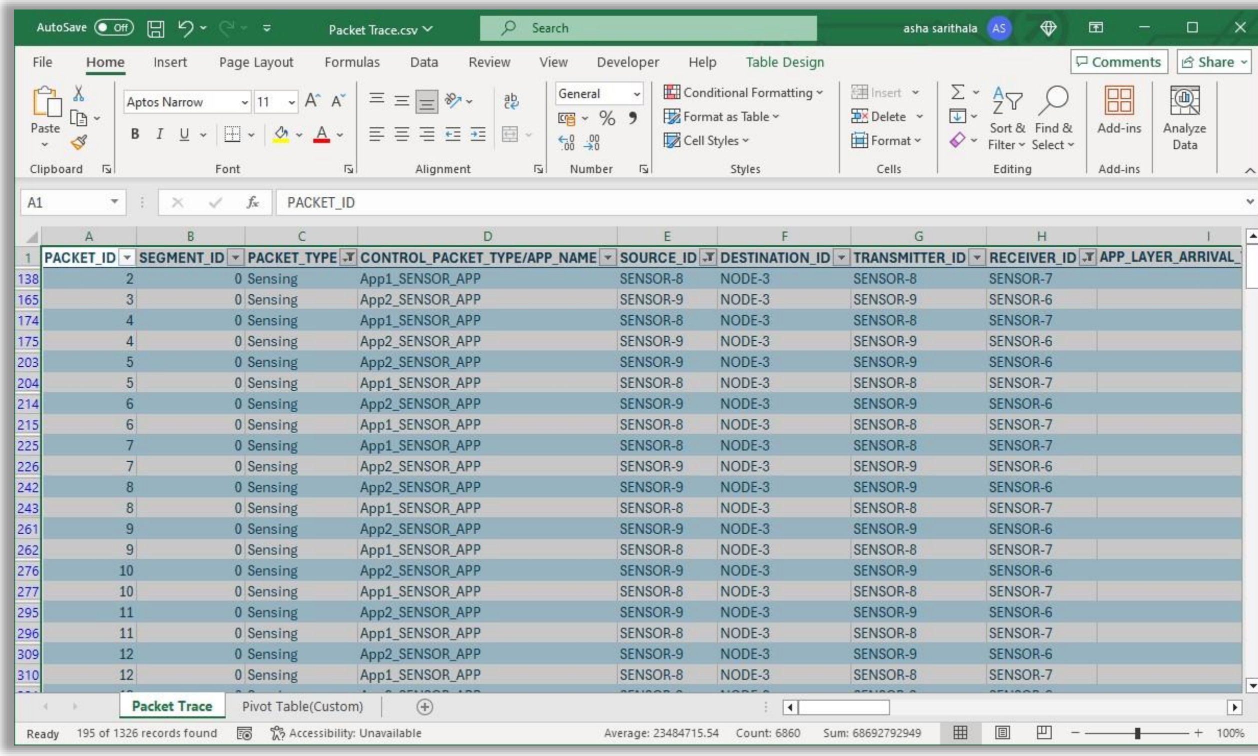


Figure 8: solution Explorer rebuild.

Results and discussion:

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



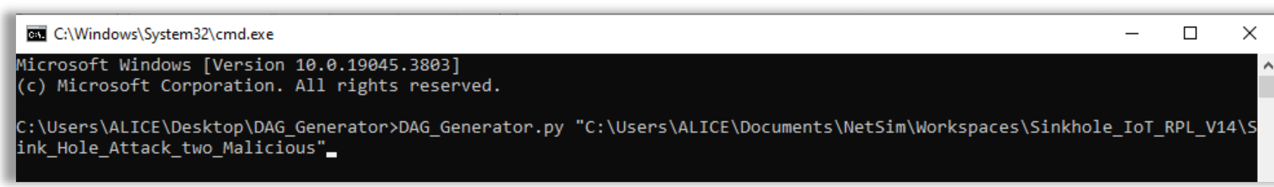
	PACKET_ID	SEGMENT_ID	PACKET_TYPE	CONTROL_PACKET_TYPE/APP_NAME	SOURCE_ID	DESTINATION_ID	TRANSMITTER_ID	RECEIVER_ID	APP_LAYER_ARRIVAL_
138	2	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7	
165	3	0	Sensing	App2_SENSOR_APP	SENSOR-9	NODE-3	SENSOR-9	SENSOR-6	
174	4	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7	
175	4	0	Sensing	App2_SENSOR_APP	SENSOR-9	NODE-3	SENSOR-9	SENSOR-6	
203	5	0	Sensing	App2_SENSOR_APP	SENSOR-9	NODE-3	SENSOR-9	SENSOR-6	
204	5	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7	
214	6	0	Sensing	App2_SENSOR_APP	SENSOR-9	NODE-3	SENSOR-9	SENSOR-6	
215	6	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7	
225	7	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7	
226	7	0	Sensing	App2_SENSOR_APP	SENSOR-9	NODE-3	SENSOR-9	SENSOR-6	
242	8	0	Sensing	App2_SENSOR_APP	SENSOR-9	NODE-3	SENSOR-9	SENSOR-6	
243	8	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7	
261	9	0	Sensing	App2_SENSOR_APP	SENSOR-9	NODE-3	SENSOR-9	SENSOR-6	
262	9	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7	
276	10	0	Sensing	App2_SENSOR_APP	SENSOR-9	NODE-3	SENSOR-9	SENSOR-6	
277	10	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7	
295	11	0	Sensing	App2_SENSOR_APP	SENSOR-9	NODE-3	SENSOR-9	SENSOR-6	
296	11	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7	
309	12	0	Sensing	App2_SENSOR_APP	SENSOR-9	NODE-3	SENSOR-9	SENSOR-6	
310	12	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7	

Figure 9: NetSim packet trace window.

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

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

Downloaded workspace contains “DAG_Generator.py” file ,Go to the “DAG_Generator.py” file path, open the command line interface ,Run the “py” file by providing the path of saved RPL experiment with packet trace enabled .



Note: please ensure that wireshark and packet trace are enabled, and that the pandas, network, and matplotlib packages are installed before running the script.

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 at the bottom left in the plot window.

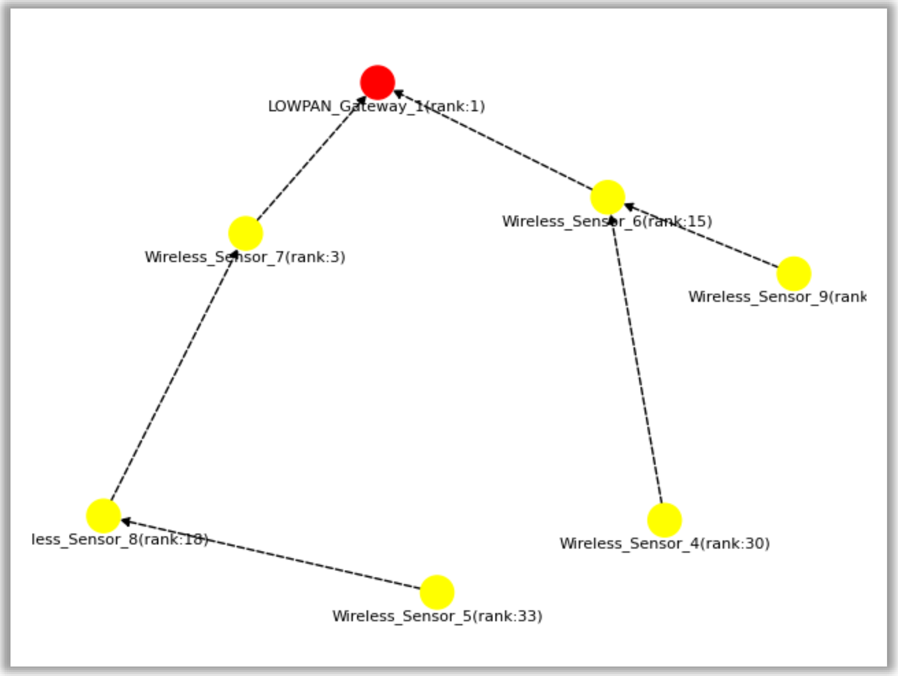


Figure 10: RPL DODAG Visualizer

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;
```