DIS Flooding Attack in IOT Networks Running RPL

Software Recommended: NetSim Standard v12.1/v12.2 (64 bit), Visual Studio 2019

Follow the instructions specified in the following link to clone/download the project folder from GitHub using Visual Studio:
https://tetcos.freshdesk.com/support/solutions/articles/14000099351-how-to-clone-netsim-file- exchange-project-repositories-from-github-

Other tools such as GitHub Desktop, SVN Client, Sourcetree, Git from the command line, or any client you like to clone the Git repository.

Note: It is recommended not to download the project as an archive (compressed zip) to avoid incompatibility while importing workspaces into NetSim.

Secure URL for the GitHub repository:

v12.1: https://github.com/NetSim-TETCOS/DIS_Flooding_Attack_in_IoT_v12.1.git

v12.2: https://github.com/NetSim-TETCOS/DIS_Flooding_Attack_in_IoT_v12.2.git

Note: The cloned project directory will contain the documentation specific to the NetSim version (v12.1/v12.2).

In RPL, DIS messages are used by nodes to join the network. A node sends a DIS message to its neighbour nodes in order to request the routing information so that it may join the existing DODAG. Thus, a new node continuously transmits DIS messages with a fixed interval until it receives a DIO message from any neighbour node. Once a node receives a DIO message, it stops transmitting DIS messages and joins the network by sending DAO to the solicited-node.

A malicious node can utilize this feature to degrade the network performance by choosing different DIS transmission interval for periodically transmitting DIS messages to its neighbouring nodes; this is called a DIS flooding attack. This leads to an increase in the network's control packet overhead and power consumption.

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 instead of joining existing DODAG it just Drops DIO and frequently transmits DIS messages. Which forces normal nodes to reset their trickle timers and flood the network with DIO messages.

A file Malicious.c is added to the RPL project.

The file contains the following functions

1. fn_NetSim_RPL_MaliciousNode( )

This function is used to identify whether a current device is malicious or not in-order to establish malicious behaviour.

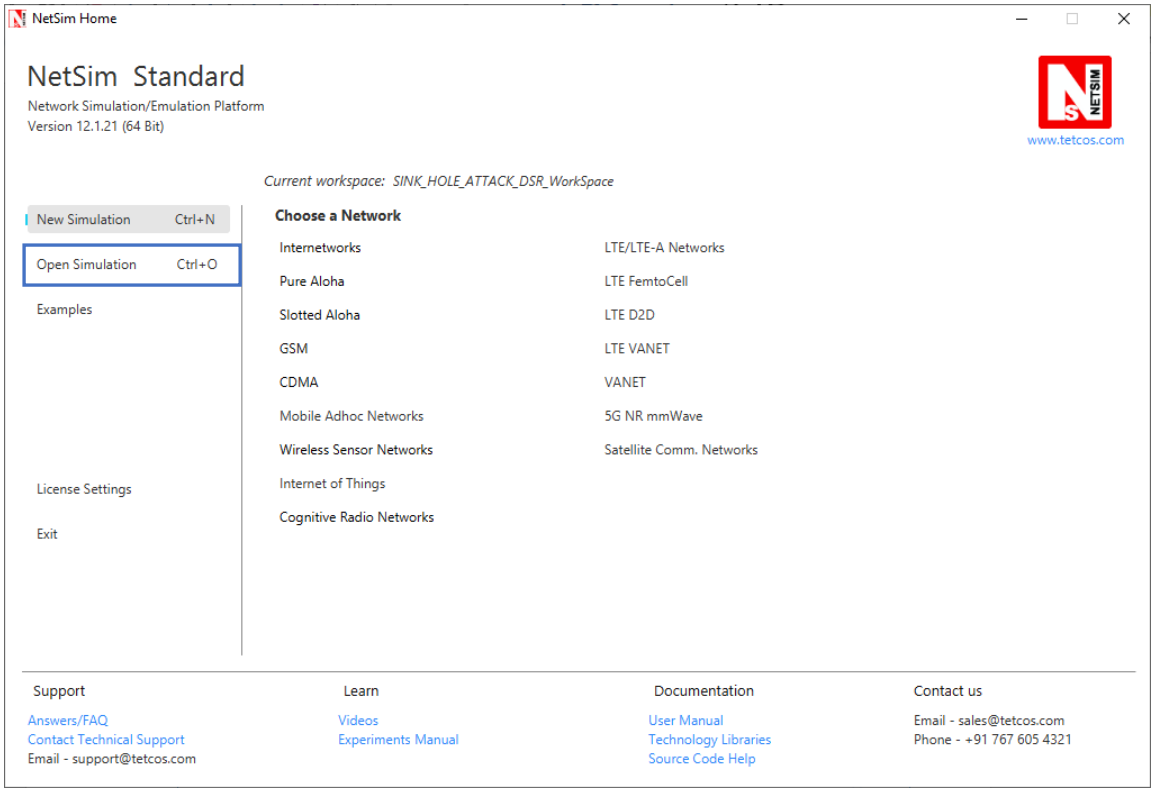
2. rpl_drop_msg( )

This function is used to drop the DIO messages received by the malicious nodes instead of replying with a DAO message.

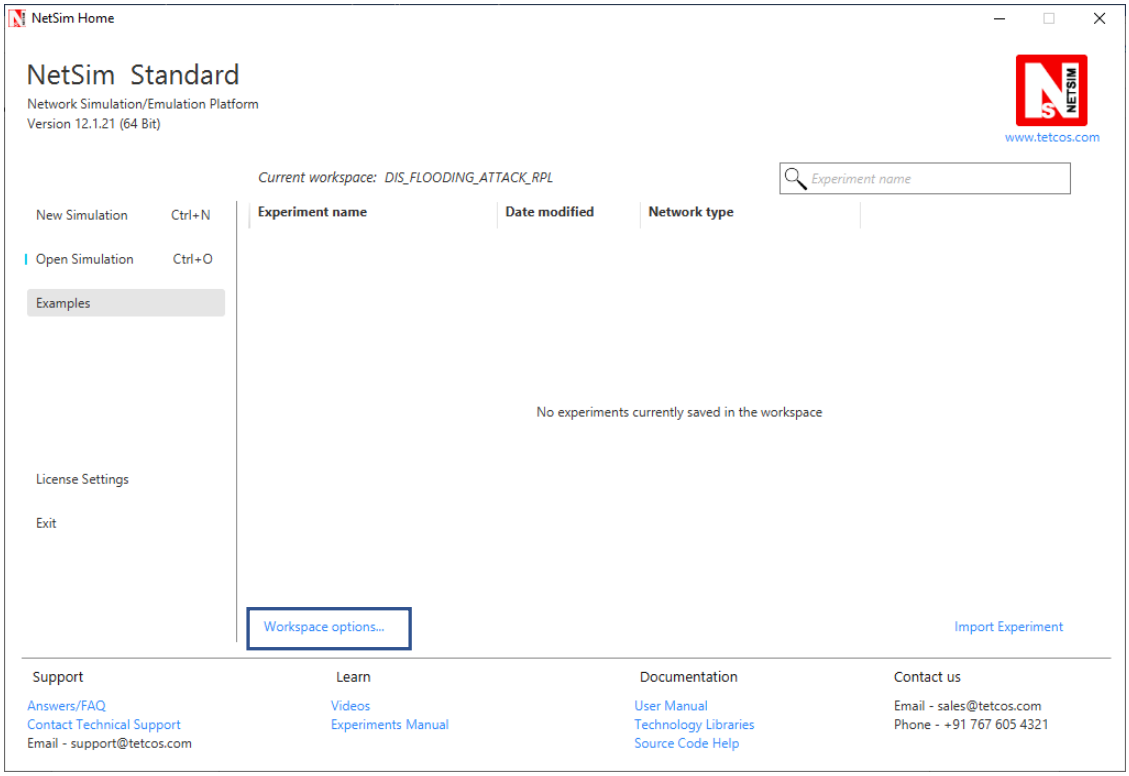
You can set any sensor as malicious and you can have more than one malicious node in a scenario. Device id's of malicious nodes can be set inside the fn_NetSim_RPL_MaliciosNode() function.

Steps:

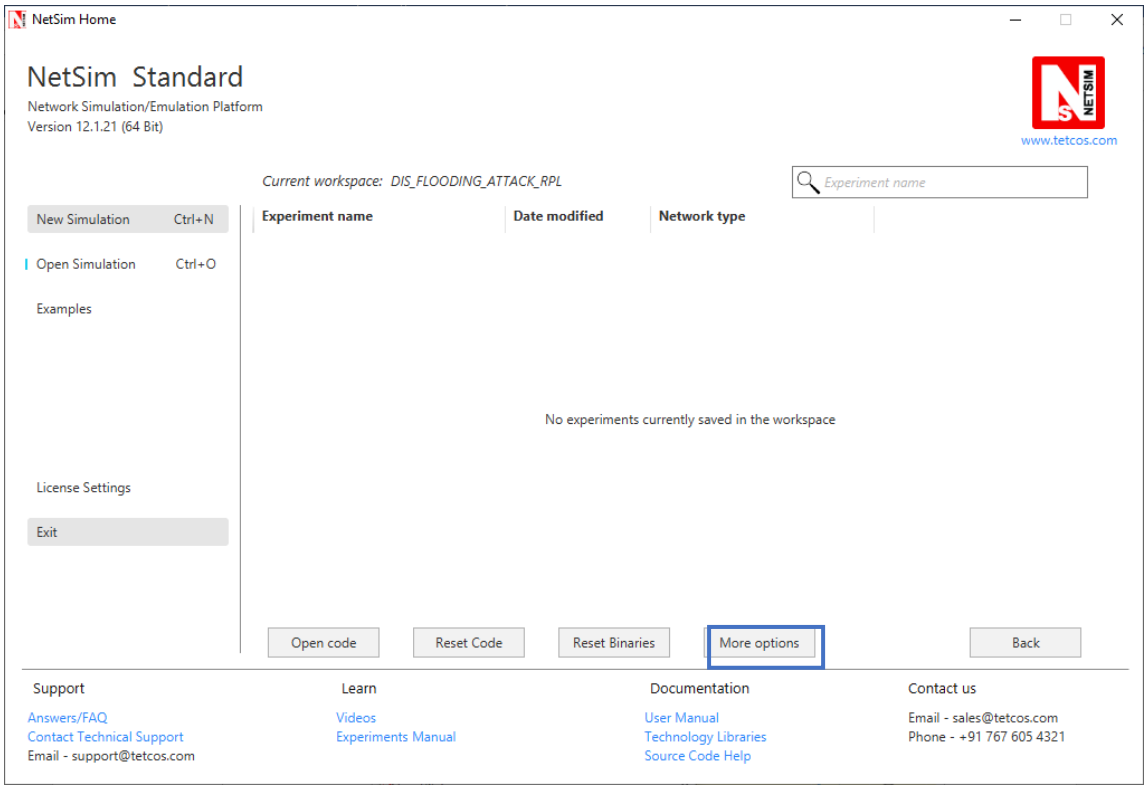
1. After you unzip the downloaded project folder, Open NetSim Home Page click on **Open Simulation** option,



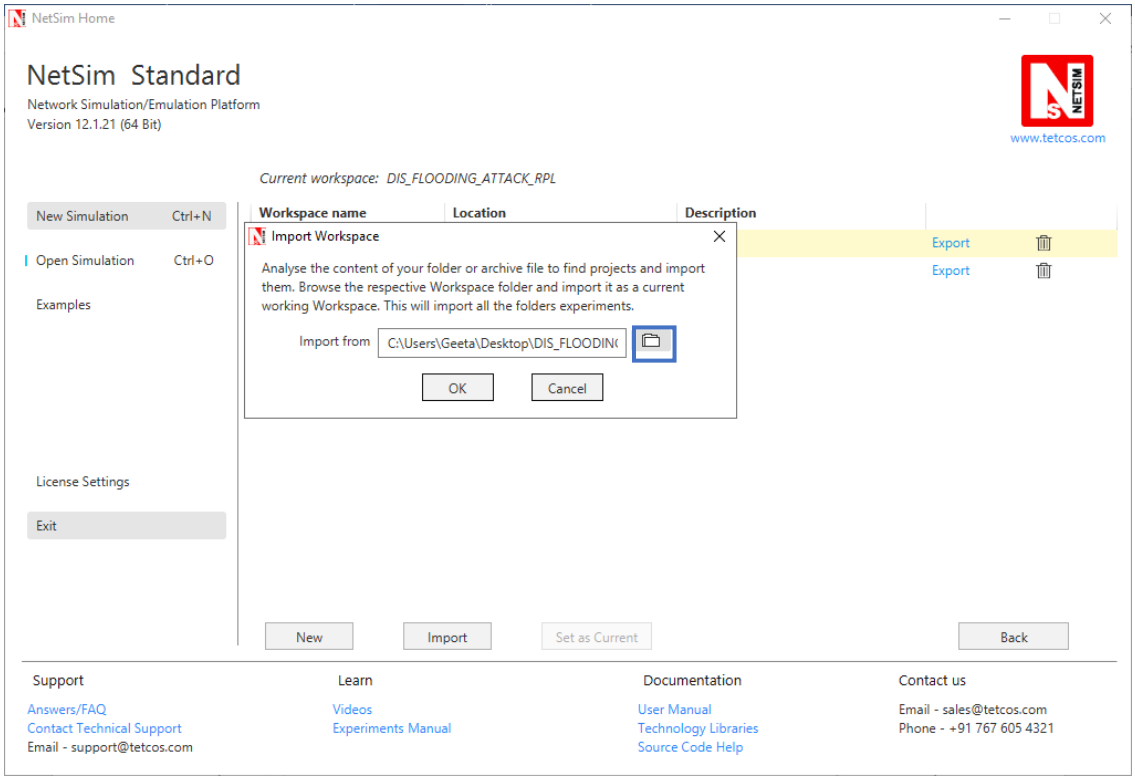
2. Click on Workspace options



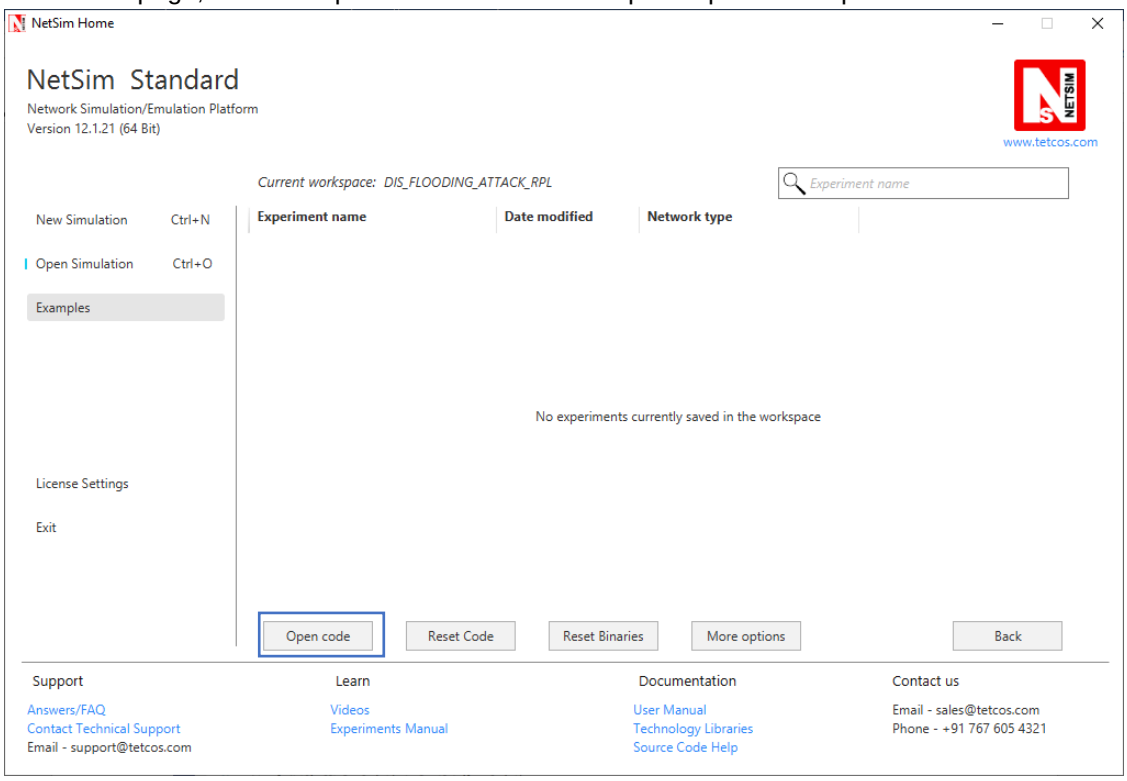
3. Click on More Options,



4. Click on Import, browse the extracted folder path and go to the DIS_FLOODING_ATTACK_RPL_WorkSpace directory. Click ok.



5. Go to home page, Click on Open Simulation → Workspace options → Open code



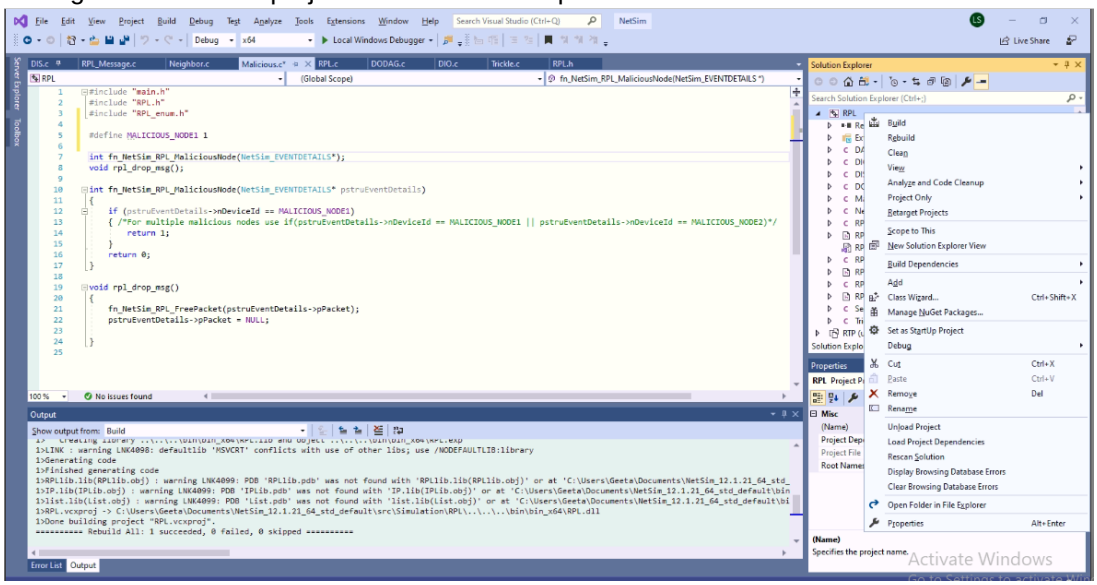
a. Expand RPL project and open Malicious.c file.
b. Set malicious node id.

```c
#include "main.h"
#include "RPL.h"
#include "RPL_enum.h"

#define MALICIOUS_NODE1 1

int fn_NetSim_RPL_MaliciousNode(NetSim_EVENTDETAILS*);
void rpl_drop_msg();

int fn_NetSim_RPL_MaliciousNode(NetSim_EVENTDETAILS* pstruEventDetails)
{
    if (pstruEventDetails->nDeviceId == MALICIOUS_NODE1)
    { /*For multiple malicious nodes use if(pstruEventDetails->nDeviceId == MALICIOUS_NODE1 || pstruEventDetails->nDeviceId == MALICIOUS_NODE2)*/
        return 1;
    }
    return 0;
}

void rpl_drop_msg()
{
    fn_NetSim_RPL_FreePacket(pstruEventDetails->pPacket);
    pstruEventDetails->pPacket = NULL;

}
```

6. Add the code that is highlighted in RPL_Message.c file

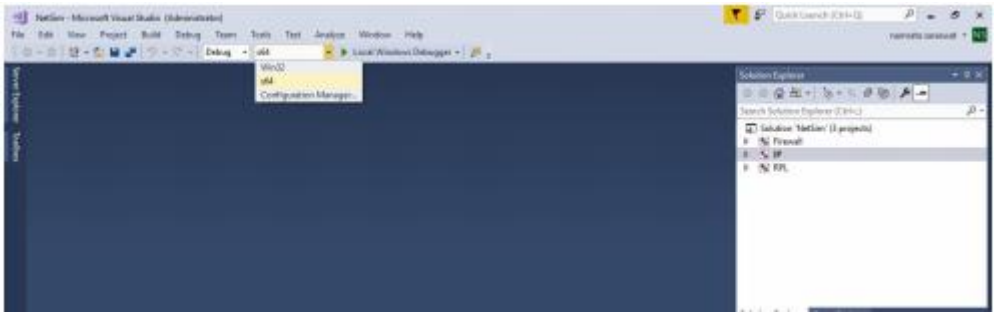
```c
    pevent.nDeviceType = DEVICE_TYPE(ndevid);
    pevent.nEventType = NETWORK_OUT_EVENT;
    pevent.nPacketId = packet->nPacketId;
    pevent.nProtocolId = NW_PROTOCOL_IPV6;
    pevent.pPacket = packet;
    fnpAddEvent(&pevent);
}

void rpl_process_ctrl_msg()
{
    switch (pstruEventDetails->pPacket->nControlDataType % 100)
    {
    case DODAG_Information_Object:
        if (fn_NetSim_RPL_MaliciousNode(pstruEventDetails))
            rpl_drop_msg();
        else
            rpl_process_dio_msg();
        break;
    case Destination_Advertisement_Object:
        rpl_process_dao_msg();
        break;
    case DODAG_Information_Solicitation:
        rpl_process_dis_msg();
        break;
    default:
        fnNetSimError("Unknown rpl ctrl msg %d in %s",
                      pstruEventDetails->pPacket->nControlDataType,
```

7. Now right click on RPL project in the solution explorer and select Rebuild

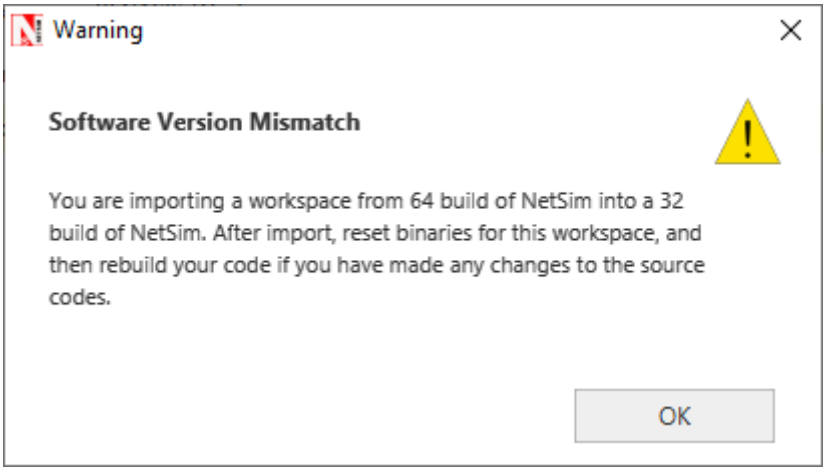


Note:

1. Based on whether you are using NetSim 32 bit or 64 bit setup you can configure Visual studio to build 32 bit or 64 bit Dll files respectively as shown below:



2. While importing the workspace, if the following warning message indicating Software Version Mismatch is displayed, you can ignore it and proceed.

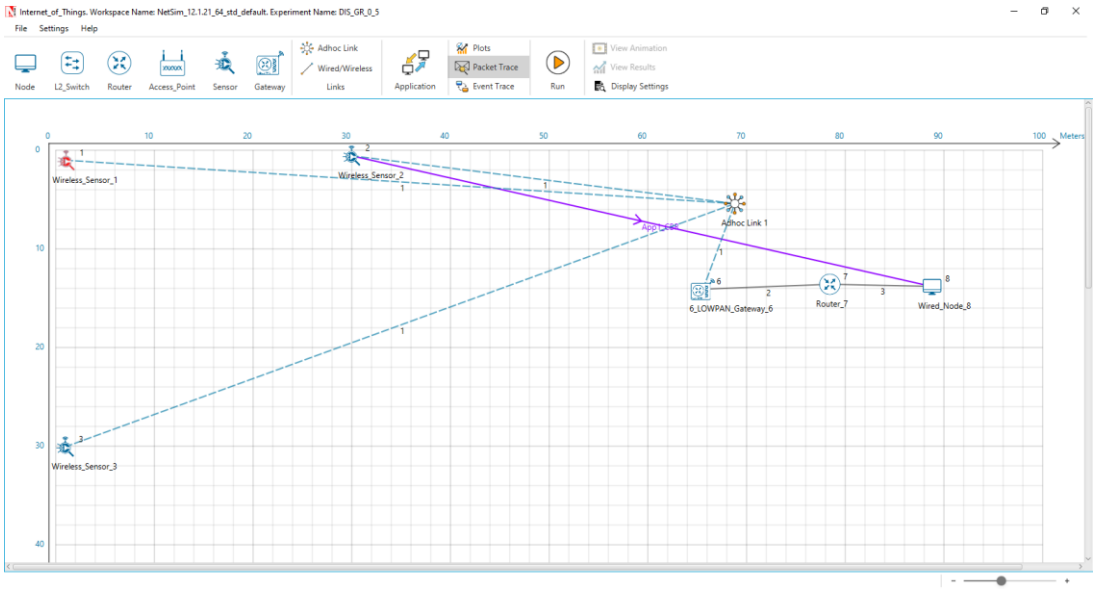


8. Upon rebuilding, libRPL.dll will automatically get updated in the respective bin folder of the current workspace.

Settings that were done to create the network scenario for DIS_Flooding Attack.

1. Create a network scenario in Internet of Things with the Transport Protocol as UDP.

2. For example, you can create a scenario as shown in the following screenshot:



Source – Device id 2
Destination – Device id 8
Sinkhole (malicious node) – Device id 1

Link Properties (Adhoc link1)

Channel characteristics – Path Loss only
Path Loss model – LOG DISTANCE
Path Loss Exponent: 3.5

Device Properties: Go to Sensor Properties -> Network_Layer -> DIS_Interval -> 10ms

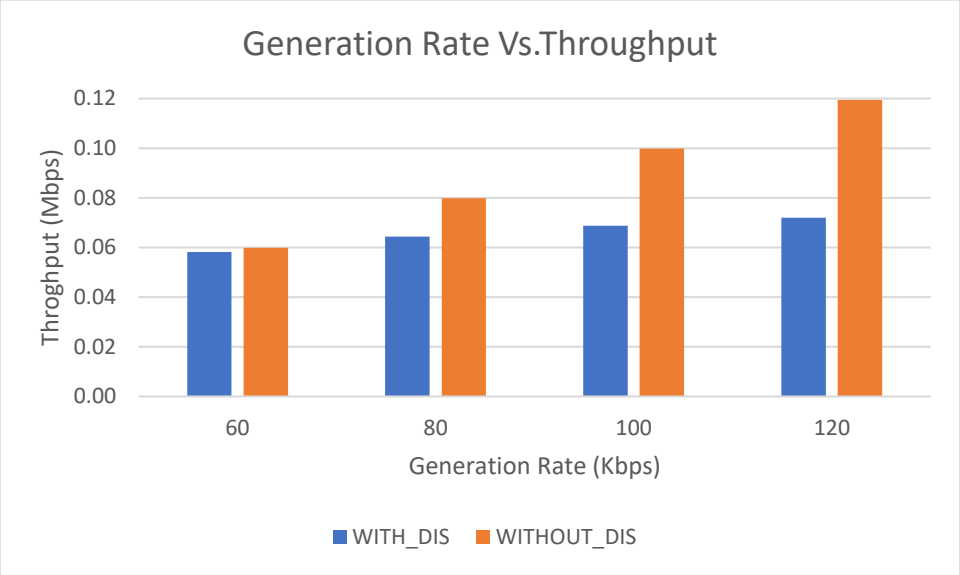
3. Run the Simulation for 100 seconds.
4. View the packet animation. You will find that the malicious node (Device id 1) even after receiving DIO from neighbour nodes instead of joining existing DODAG it just Drops DIO and frequently transmits DIS messages.
5. This will have a direct impact on the Application Throughput and Delay which can be observed in the Application Metrics table present in NetSim Simulation Results window.

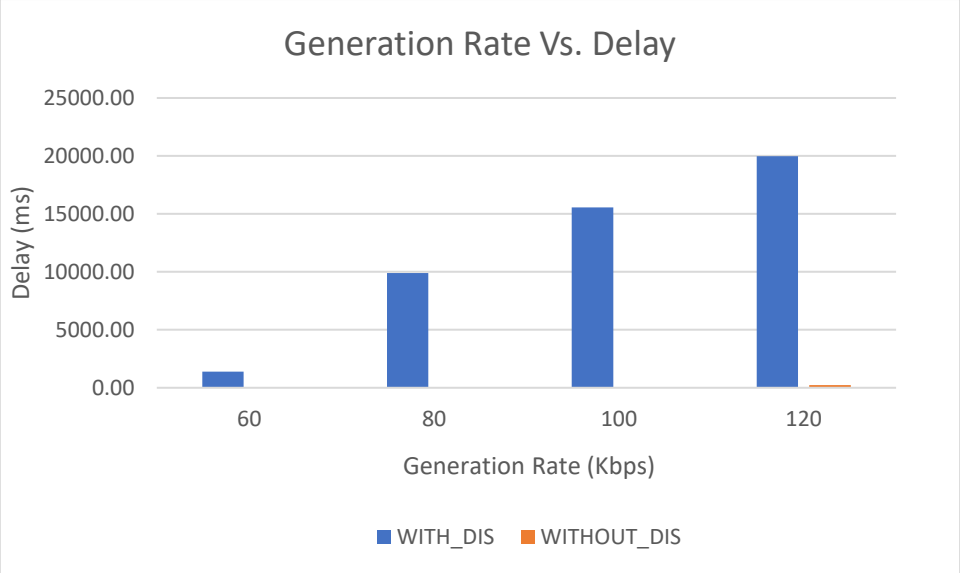
Case 1: Application throughput Vs. Application generation rate

We fix the DIS interval to 10 milli seconds and vary the application generation rate to see the impact of DIS flooding on the network performance.

	Throughput (Mbps)		Delay (ms)	
Generation Rate (Kbps)	**With_DIS**	**Without_DIS**	**With_DIS**	**Without_DIS**
60	0.06	0.06	1394.49	51.80
80	0.06	0.08	9896.58	51.75
100	0.07	0.10	15553.54	51.76
120	0.07	0.12	19988.49	239.62

This can be further understood with the help of following plots:


Generation Rate Vs.Throughput
Throghput (Mbps)
0.12
0.10
0.08
0.06
0.04
0.02
0.00
60
80
100
120
Generation Rate (Kbps)
WITH_DIS
WITHOUT_DIS


Generation Rate Vs. Delay
Delay (ms)
25000.00
20000.00
15000.00
10000.00
5000.00
0.00
60
80
100
120
Generation Rate (Kbps)
WITH_DIS
WITHOUT_DIS

We can observe that the application throughput decreases in case of DIS flooding when compared with the usual simulations for various application traffic generation rates.

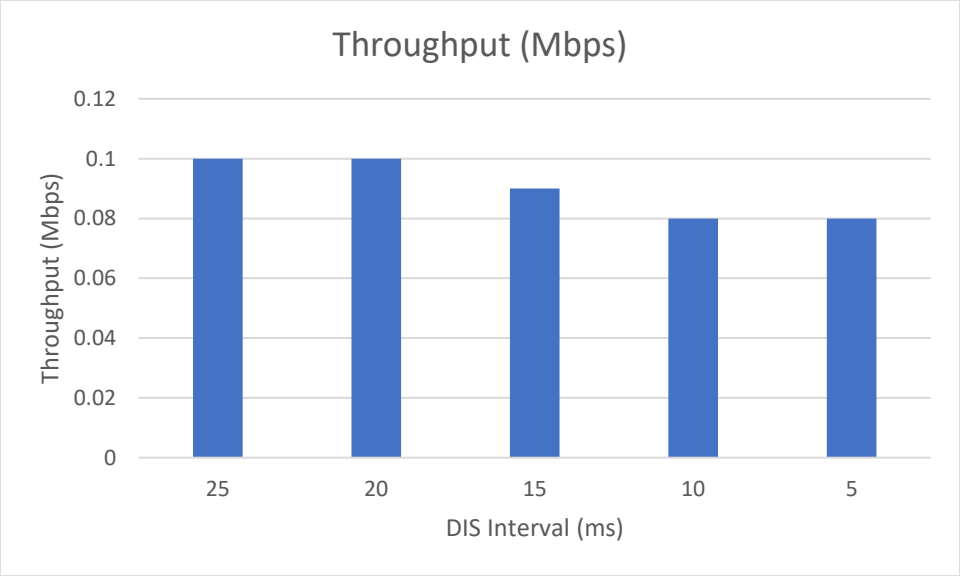
Delay is comparatively high in case of DIS flooding and increases with the increase in generation rate. This is because the nodes are busy receiving and responding to DIS messages from malicious node frequently. The nodes that receive DIS messages are forced to reset their trickle timers and flood the network with DIO messages.

Case 2: Application throughput Vs. DIS Interval Time

We fix the application generation rate to 250 Kbps and vary the DIS interval to see the impact of DIS flooding on the network performance.

DIS Interval (ms)	Throughput (Mbps)
25	0.10
20	0.10
15	0.09
10	0.08
5	0.08

This can be further understood with the help of following plots:



We can observe that the application throughput decreases as we decrease DIS Interval time. Upon decreasing the DIS interval, more DIS messages will be sent by the malicious nodes more frequently. Legitimate sensors spend more time in processing and responding to DIS messages than sending the data packets.

DIS flooding severely degrades the performance of Low Power and Lossy Networks (LLNs) because of the increase in control packet overhead.