DIS Flooding Attack in IOT Networks Running RPL

Software Recommended: NetSim Standard v14.1, Visual Studio 2022

Reference: Y. Mai, F. M. Rodriguez and N. Wang, "CC-ADOV: An effective multiple paths congestion control AODV," 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, 2018, pp. 1000-1004.

Project Download Link:

https://github.com/NetSim-TETCOS/DIS-Flooding-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 RPL, DIS messages are used by nodes to join the network. A node sends a DIS message to its neighbor nodes 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 neighbor 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 intervals for periodically transmitting DIS messages to its neighboring nodes; this is called a DIS flooding attack. This leads to an increase in the network's control packet overhead and power consumption.

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.

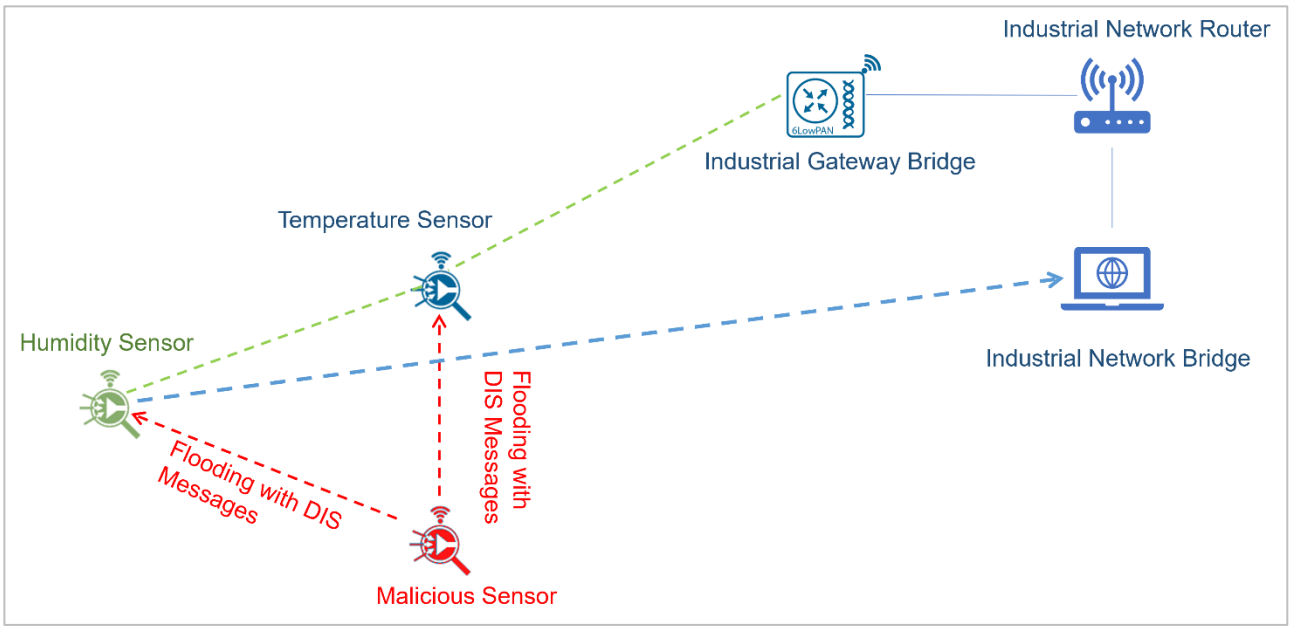

Figure 1: Real World scenario for DIS Flooding in IOT Network

DIS Flooding Overview:

- Humidity and temperature sensors periodically generate DIS messages containing their sensor readings and broadcast them to their nearby neighbors. These DIS messages include the sensor's identification information and the DODAG prefix, allowing potential parent nodes to identify the sensor and determine its location within the network.
- Malicious sensors exploit the DIS message mechanism to join the RPL network. They continuously flood the network with DIS messages, targeting humidity and temperature sensors, until they receive a DIO message from a parent node. This malicious behavior significantly increases power consumption, introduces latency, and imposes a high computational load on the network.

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 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 behavior.
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 Node, 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_MaliciosNode() function in malicious.c file.

Example:

- The DIS-Flooding-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 DIS_FLOOD_Case1_Example from the list of experiments.
- The saved network scenario consists of
  - 3 Sensors
  - 1 6 _LOWPAN Gateway
  - 1 Router
  - 1 wired node

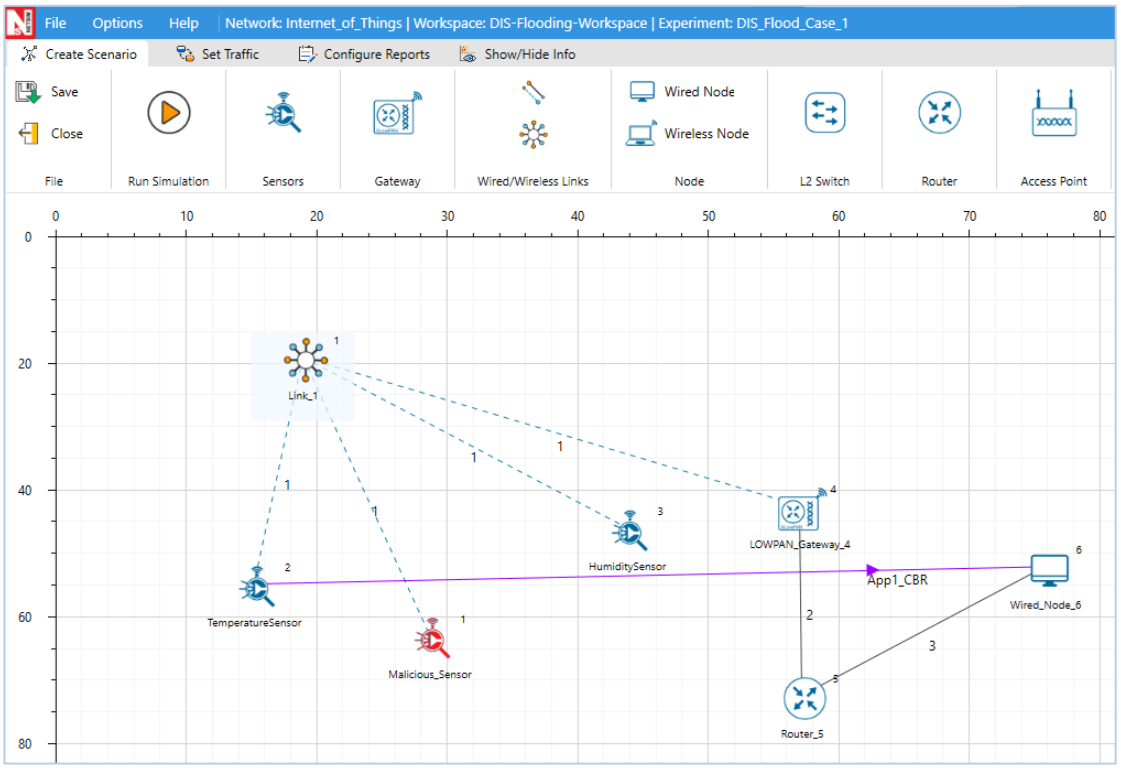

Figure 2: Network Setup for RPL DIS flooding in IOT

- Set the Application Properties

Application Properties	
Source ID	2
Destination ID	6
Transport Protocol	UDP
Other Properties	Set Default

Table 1:Application properties.

- Link Properties (Link1):
  - Channel Characteristics – Pathloss Only
  - Pathloss Model – Log Distance
  - Pathloss Exponent – 3.5
- Set Network Layer Routing Protocol to RPL in both sensor and 6_LOWPAN_Gateway
- Device Properties: Go to Sensor Properties -> Network Layer -> DIS_Interval -> 10ms.

- Run the Simulation for 100 seconds.

Results and Discussion:

1. In packet trace, you will find that the malicious node (Device id 1) even after receiving DIO from neighbor nodes it just Drops DIO and the malicious node frequently transmits DIS messages to the neighbor nodes.
2. This will have a direct impact on the Application Throughput and Delay which can be observed in the Application Metrics table present in the NetSim Simulation Results window.

Simulation instructions in Visual Studio:

- For **With_DIS,** Run the simulation of the imported workspace.
- For **Without_DIS,** Reset the binaries of the imported workspace and run the simulation.
- To reset the binaries, go to your Work -> Source Code -> Reset Binaries

To recheck the impact of the network performance **with_DIS,** Rebuild the **RPL** project in source code, Go to your Work -> Source Code -> Open Source Code.

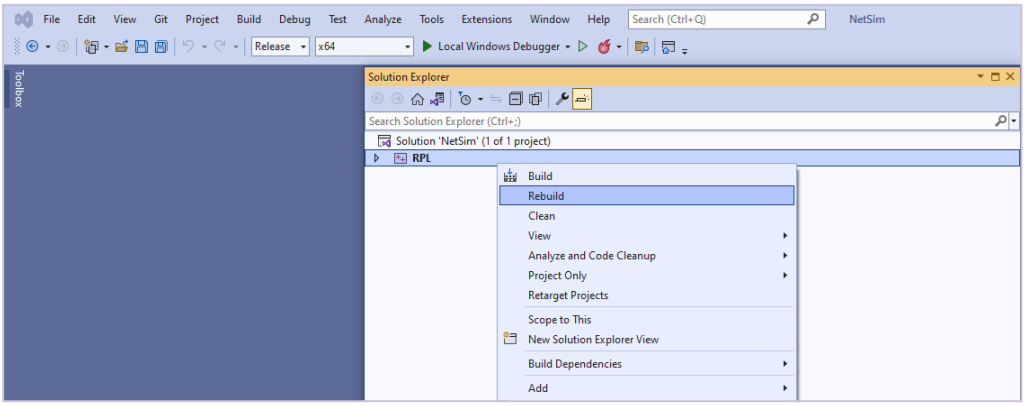

Figure 3: Source code of current Workspace

Case 1: Application throughput Vs. Application generation rate:

We fix the DIS interval to 10 milliseconds 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	0.0419	0.0597	7023.682	51.932
80	0.0460	0.0798	14518.141	52.009
100	0.0498	0.0997	19147.977	52.106
120	0.0515	0.1181	23208.916	726.934

Table 2 : Throughput Vs Delay

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

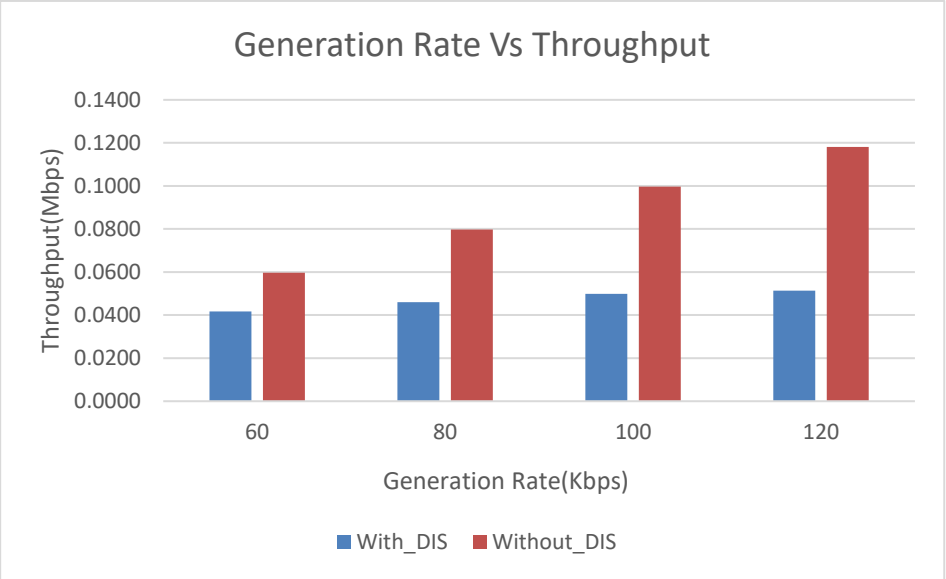

Figure 4:Generation Rate Vs Throughput

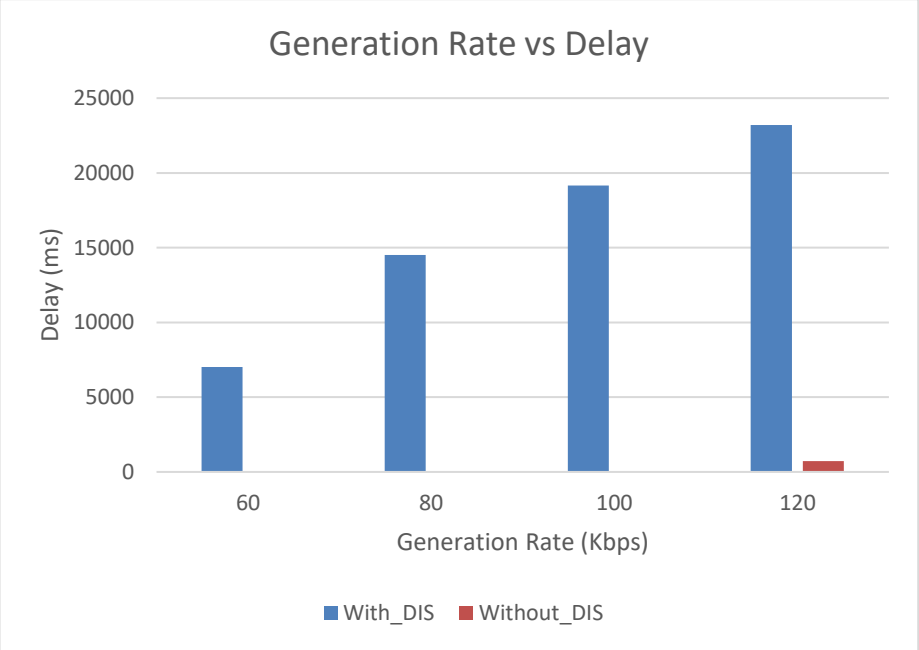

Figure 5:Generation Rate Vs Delay

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 the 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 nodes 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.

To change the DIS Interval parameter, go to Sensor Properties -> Network_Layer -> DIS_Interval -> 20ms.

DIS Interval (ms)	Throughput (Mbps)
25	0.091
20	0.085
15	0.075
10	0.058
5	0.056

Table 3: DIS Interval Vs Throughput (Mbps)

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

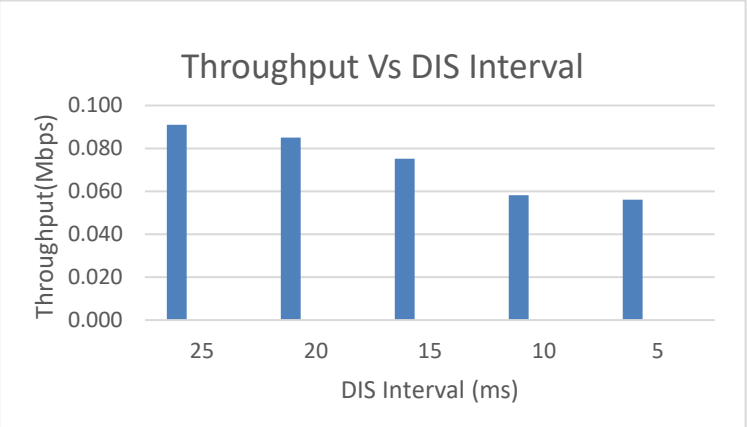

Figure 6: DIS Interval Vs Throughput (Mbps)

We can observe that the application throughput decreases as we decrease the 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 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.

Appendix: NetSim source code modifications

Set malicious node id, in malicious.c file, within RPL project

/* User can set the MALICIOUS node ID*/

```
#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;
}
```

Changes to rpl_process_ctrl_msg(), in RPL_Message.c file, within RPL project

```
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,
                                                                    __FUNCTION__);
break;
}  }
```