Dos Attack in Internet of Things

Software Used: NetSim Standard v13.0 (32/64 bit), Visual Studio 2019

Project Download Link:

https://github.com/NetSim-TETCOS/DOS_Attack_in_IoT_v13.0/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:

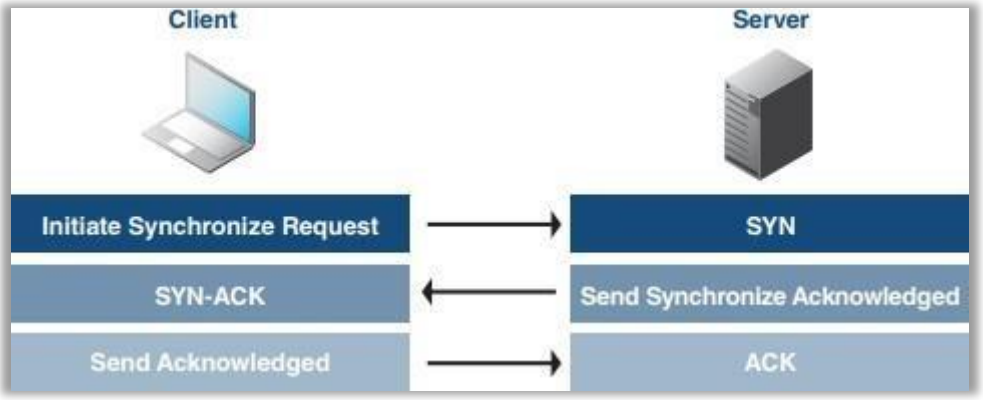
A Denial of Service (DoS) attack is an attempt to make a system unavailable to the intended user(s), such as preventing access to a website. A successful DoS attack consumes all available network or system resources, usually resulting in a slowdown or server crash. Whenever multiple sources are coordinating in the DoS attack, it becomes known as a DDoS (Distributed Denial of Service) attack. **Standard DDoS Attack types:**

1. SYN Flood
2. UDP Flood
3. SMBLoris
4. ICMP Flood
5. HTTP GET Flood

SYN Flood:

TCP SYN floods are DoS attacks that attempt to flood the DNS server with new TCP connection requests. Normally, a client initiates a TCP connection through a three-way handshake of messages:

- The client requests a connection by sending a SYN (synchronize) message to the server.
- The server acknowledges the request by sending SYN-ACK back to the client.
- The client answers with a responding ACK, establishing the connection.



This triple exchange is the foundation for every connection established using the Transmission Control Protocol (TCP). A SYN Flood is one of the most common forms of DDoS attacks. It occurs when an attacker sends a succession of TCP Synchronize (SYN) requests to the target in an attempt to consume enough resources to make the server unavailable for legitimate users. This works because a SYN request opens network communication between a prospective client and the target server. When the server receives a SYN request, it responds acknowledging the request and holds the communication open while it waits for the client to acknowledge the open connection. However, in a successful SYN Flood, the client acknowledgment never arrives, thus consuming the server's resources until the connection times out. A large number of incoming SYN requests to the target server exhausts all available server resources and results in a successful DoS attack. Before implementing this project in NetSim, users have to understand the steps given below:

1. TCP Log file

- User need to understand the TCP log file which will get created in the temp path of NetSim <Windows Temp Folder>/NetSim>

- The TCP Log file is usually a very large file and hence is disabled by default in NetSim.

- To enable logging, go to TCP.c inside the TCP project and change the function bool isTCPlog() to return true instead of false.

2. At malicious node:

Create a new timer event called SYN_FLOOD in TCP for sending TCP_SYN packets that should be triggered for every 1000 micro seconds. This will create and send the

TCP_SYN packet for every 1000 micro seconds. SYN request opens network communication between a client and the target **3. At Target node:**

When the target receives a SYN request, it responds acknowledging the request and holds the communication open while it waits for the client to acknowledge the open connection. If a SYN packet arrives at Receiver, it should reply with a SYN_ACK packet. For this SYN_ACK packet, add a processing time of 2000 micro seconds in Ethernet Physical Out. This delays the arrival of SYN_ACK at source node. During this delay, another SYN packet will get created at the malicious node. A large number of incoming SYN requests to the target exhausts all available server resources and results in a successful DoS attack **SYN_FLOOD in NetSim:**

To implement this project in NetSim, we have created SYN_FLOOD.c file inside TCP project. The file contains the following functions:

- int is_malicious_node();

  This function is used to check the node is malicious node or not

- int socket_creation();

  This function is used to create a new socket and update the socket parameters

- static void send_syn_packet(PNETSIM_SOCKET s);

  This function is used to create and send SYN packet to the network layer

- void syn_flood();

  This function is used to check whether the socket is present or not and also adds a timer event called SYN_FLOOD (triggers for every 1000µs)

Code modifications done in NetSim:

1. We have added the following lines of code in fn_NetSim_TCP_Trace() function present in TCP.c file inside TCP project. This is used to add the SYN_FLOOD sub-events in Event Trace file

```
TCP.c
TCP                                                    (Global Scope)
125     to get the sub event as a string.
126     */
127     _declspec (dllexport) char *fn_NetSim_TCP_Trace(int nSubEvent)
128     {
129         if (nSubEvent == SYN_FLOOD)
130             return "SYN_FLOOD";
131         return (GetStringTCP_Subevent(nSubEvent));
132     }
```

2. We have added the following lines of code in fn_NetSim_TCP_HandleTimer() function present in

   TCP.c file inside TCP project. Used to add a TCP sub_event called SYN_FLOOD

```
TCP.c
TCP                                                    (Global Scope)
206     static int fn_NetSim_TCP_HandleTimer()
207     {
208         switch (pstruEventDetails->nSubEventType)
209         {
210         case SYN_FLOOD:
211             syn_flood();
212             break;
213         case TCP_RTO_TIMEOUT:
214             handle_rto_timer();
215             break;
216         case TCP_TIME_WAIT_TIMEOUT:
217             handle_time_wait_timeout();
218             break;
219         default:
220             fnNetSimError("Unknown subevent %d in %s\n",
221                           pstruEventDetails->nSubEventType,
222                           __FUNCTION__);
223             break;
224         }
225         return 0;
226     }
```

3. And modified the following lines of code in fn_NetSim_TCP_Init() function resent in TCP.c inside TCP project

```
*/
_declspec (dllexport) int fn_NetSim_TCP_Init(struct stru_NetSim_Network* NETWORK_Formal,
    NetSim_EVENTDETAILS* pstruEventDetails_Formal,
    char* pszAppPath_Formal,
    char* pszWritePath_Formal,
    int nVersion_Type,
    void** fnPointer)
{
    fn_NetSim_TCP_Init_F(NETWORK_Formal,
        pstruEventDetails_Formal,
        pszAppPath_Formal,
        pszWritePath_Formal,
        nVersion_Type,
        fnPointer);
    NetSim_EVENTDETAILS pevent;
    memcpy(&pevent, pstruEventDetails, sizeof pevent);

    for (int i = 0; i < NETWORK->nDeviceCount; i++)
    {
        if (is_malicious_node(i + 1))
        {
            pevent.nDeviceId = i + 1;
            pevent.dEventTime += 1000;
            pevent.nEventType = TIMER_EVENT;
            pevent.nSubEventType = SYN_FLOOD;
            pevent.nProtocolId = TX_PROTOCOL_TCP;
            fnpAddEvent(&pevent);

        }
    }
    return 0;
}

/**
This function is called by NetworkStack.dll, once simulation end to free the
allocated memory for the network.
*/
```

4. And modified the following lines of code in add_timeout_event() present in RTO.c file inside TCP project which avoids RTO timer for malicious nodes

```
    *rto = min(max((*rto*2), G), (60 * SECOND));
    print_tcp_log("New RTO = %0.2lf", *rto);
}

void add_timeout_event(PNETSIM_SOCKET s,
                       NetSim_PACKET* packet)
{
    NetSim_PACKET* p = fn_NetSim_Packet_CopyPacket(packet);
    add_packet_to_queue(&s->tcb->retransmissionQueue, p, pstruEventDetails->dEventTime);
    NetSim_EVENTDETAILS pevent;
    memcpy(&pevent, pstruEventDetails, sizeof pevent);
    pevent.dEventTime += TCP_RTO(s->tcb);
    pevent.dPacketSize = packet->pstruTransportData->dPacketSize;
    pevent.nEventType = TIMER_EVENT;
    pevent.nPacketId = packet->nPacketId;
    if (packet->pstruAppData)
    {
        pevent.nApplicationId = packet->pstruAppData->nApplicationId;
        pevent.nSegmentId = packet->pstruAppData->nSegmentId;
    }
    else
        pevent.nSegmentId = 0;
    if (!is_malicious_node(pevent.nDeviceId))
    {
        pevent.nProtocolId = TX_PROTOCOL_TCP;
        pevent.pPacket = fn_NetSim_Packet_CopyPacket(p);
        pevent.szOtherDetails = NULL;
        pevent.nSubEventType = TCP_RTO_TIMEOUT;
        fnpAddEvent(&pevent);
        print_tcp_log("Adding RTO Timer at %0.1lf", pevent.dEventTime);
    }
}

static void handle_rto_timer_for_ctrl(PNETSIM_SOCKET s)
{
    if (isSynbitSet(pstruEventDetails->pPacket))
```

5. Users can give their own number of malicious node in **TCP.h** file inside TCP project

```
TCP.h
TCP                                    (Global Scope)
49      //USEFUL MACRO
50  #define isTCPConfigured(d) (DEVICE_TRXLayer(d) && DEVICE_TRXLayer(d)->isTCP)
51  #define isTCPControl(p) (p->nControlDataType/100 == TX_PROTOCOL_TCP)
52
53      //Constant
54  #define TCP_DupThresh   3
55  #define NUMBEROFMALICIOUSNODE 2
56      int is_malicious_node(NETSIM_ID devid);
57      //Typedef
58      typedef struct stru_TCP_Socket  NETSIM_SOCKET, *PNETSIM_SOCKET;
59
60      typedef enum enum_tcpstate
61      {
62          TCPCONNECTION_CLOSED,
63          TCPCONNECTION_LISTEN,
64          TCPCONNECTION_SYN_SENT,
65          TCPCONNECTION_SYN_RECEIVED,
66          TCPCONNECTION_ESTABLISHED,
67          TCPCONNECTION_FIN_WAIT_1,
68          TCPCONNECTION_FIN_WAIT_2,
69          TCPCONNECTION_CLOSE_WAIT,
70          TCPCONNECTION_CLOSING,
71          TCPCONNECTION_LAST_ACK,
72          TCPCONNECTION_TIME_WAIT,
73      }TCP_CONNECTION_STATE;
74
75      typedef enum enum_tcp_variant
76      {
77          TCPVariant_OLDTAHOE, //Slow Start and Congestion Avoidance
78          TCPVariant_TAHOE,    //Fast Retransmit/Fast Recovery
79          TCPVariant_RENO,
80          TCPVariant_NEWRENO,
81          TCPVariant_BIC,
82          TCPVariant_CUBIC,
83      }TCPVARIANT;
```

6. Users can give their own target ID and malicious ID in **SYN_FLOOD.c** file inside TCP project

```
SYN_flood.c
TCP                                    (Global Scope)
13  * -----------------------------------------------------------------------------*/
14
15  #include "main.h"
16  #include "TCP.h"
17  #include "List.h"
18  #include "TCP_Header.h"
19  #include "TCP_Enum.h"
20
21  int malicious_node[NUMBEROFMALICIOUSNODE] = { 2, 6 };
22  static void send_syn_packet(PNETSIM_SOCKET s);
23  //static PNETSIM_SOCKET socket_creation();
24  int target_node = 4;
25  PNETSIM_SOCKET get_Remotesocket(NETSIM_ID d, PSOCKETADDRESS addr);
26  static PSOCKETADDRESS sockAddr = NULL;
27
28  int is_malicious_node(NETSIM_ID devid)
29  {
30      for (int i = 0; i < NUMBEROFMALICIOUSNODE; i++)
31          if (devid == malicious_node[i]) return 1;
32
33      return 0;
34  }
35
36  void syn_flood()
37  {
38      /*
39          if (!sockAddr)
40          {
41              sockAddr = calloc(1, sizeof * sockAddr);
42              sockAddr->ip = DEVICE_NWADDRESS(target_node, 1);
43
44          }
45
46          PNETSIM_SOCKET s = get_Remotesocket(malicious_node, sockAddr);
47  */
```

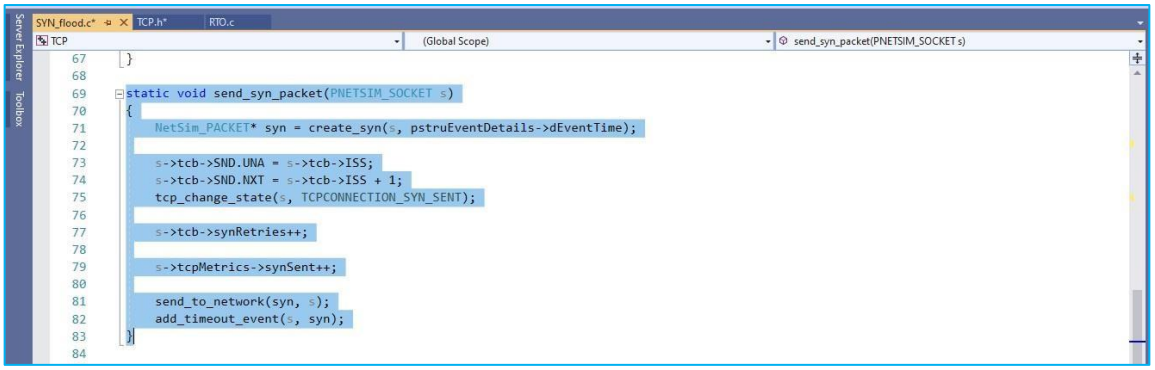
7. Added the following line in TCP_Enum.h file inside TCP project to add a new TCP_subevent called SYN_FLOOD

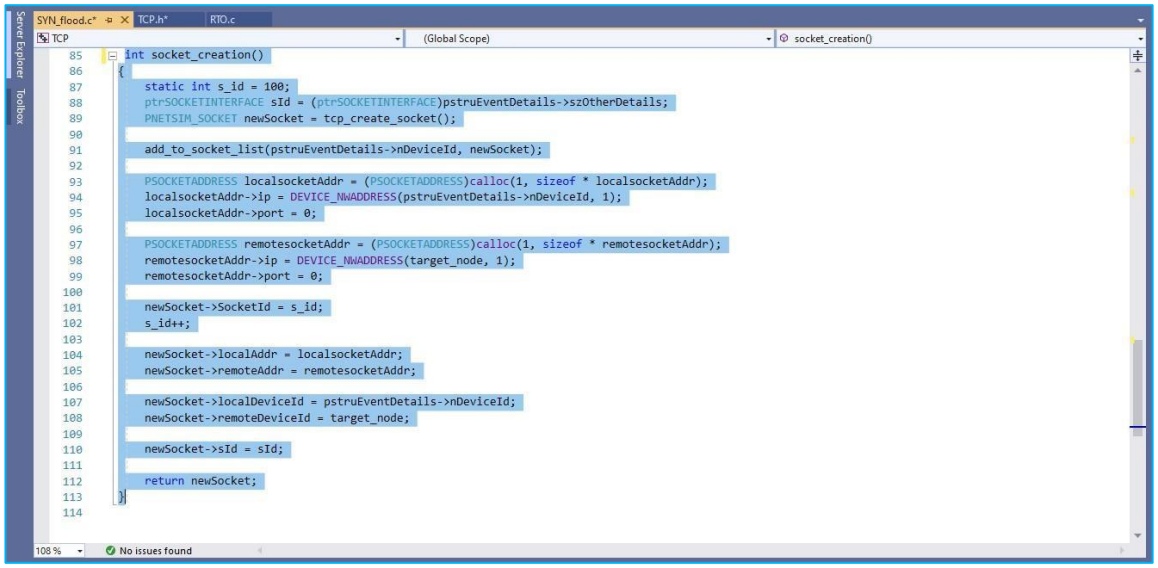
```
#include "EnumString.h"

BEGIN_ENUM(TCP_Subevent)
{
    DECL_ENUM_ELEMENT_WITH_VAL(TCP_RTO_TIMEOUT, TX_PROTOCOL_TCP * 100),
    DECL_ENUM_ELEMENT(TCP_TIME_WAIT_TIMEOUT),
    DECL_ENUM_ELEMENT(SYN_FLOOD),
}
#pragma warning(disable:4028)
END_ENUM(TCP_Subevent);
#pragma warning(default:4028)
```

8. SYN_FLOOD.c file contains the following functions

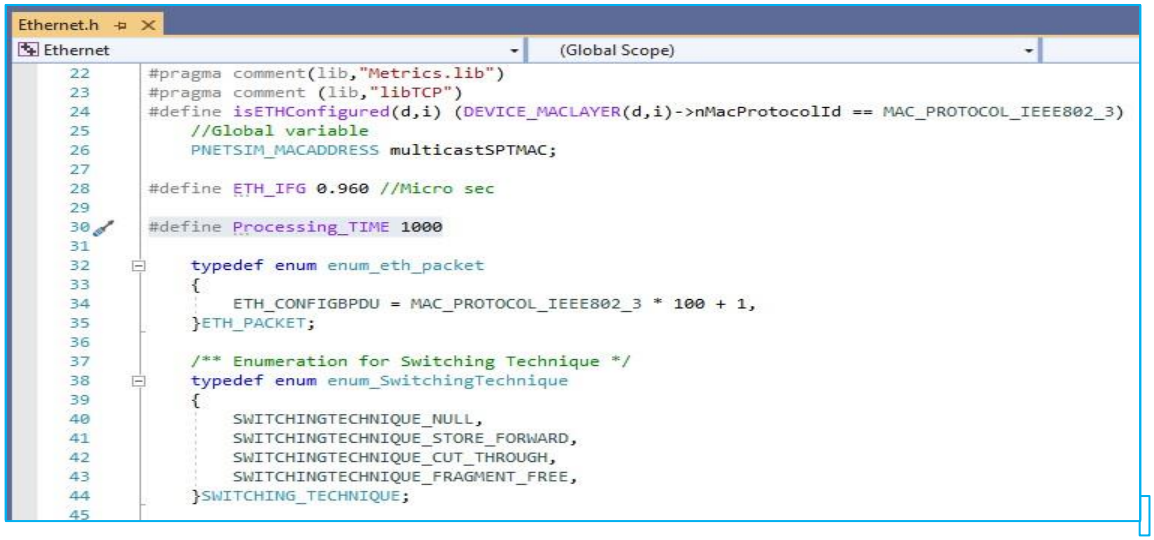
```
#include "TCP.h"
#include "List.h"
#include "TCP_Header.h"
#include "TCP_Enum.h"

int malicious_node[NUMBEROFMALICIOUSNODE] = { 2, 6 };
static void send_syn_packet(PNETSIM_SOCKET s);
//static PNETSIM_SOCKET socket_creation();
int target_node = 4;
PNETSIM_SOCKET get_Remotesocket(NETSIM_ID d, PSOCKETADDRESS addr);
static PSOCKETADDRESS sockAddr = NULL;

int is_malicious_node(NETSIM_ID devid)
{
    for (int i = 0; i < NUMBEROFMALICIOUSNODE; i++)
        if (devid == malicious_node[i]) return 1;

    return 0;
}
```

```
}

void syn_flood()
{
    extern PSOCKETADDRESS anySocketAddr;
    anySocketAddr->ip = DEVICE_NWADDRESS(target_node, 1);
    PNETSIM_SOCKET s = get_Remotesocket(pstruEventDetails->nDeviceId, anySocketAddr);
    ptrSOCKETINTERFACE sId = (ptrSOCKETINTERFACE)pstruEventDetails->szOtherDetails;
    NetSim_EVENTDETAILS pevent;
    if (!s)
    {
        s = socket_creation();
        tcp_connect(s, s->localAddr, s->remoteAddr);
    }

    else
    {
        s->localDeviceId = pstruEventDetails->nDeviceId;
        s->remoteDeviceId = target_node;
        s->sId = sId;
        send_syn_packet(s);
        memcpy(&pevent, pstruEventDetails, sizeof pevent);
        pevent.dEventTime = pstruEventDetails->dEventTime + 1000;
        pevent.nDeviceId = pstruEventDetails->nDeviceId;
        pevent.nPacketId = 0;
        pevent.nEventType = TIMER_EVENT;
        pevent.nProtocolId = TX_PROTOCOL_TCP;
        pevent.nSubEventType = SYN_FLOOD;
        fnpAddEvent(&pevent);
    }


}
```





9. Added PROCESSING_TIME macro in Ethernet.h file inside ETHERNET project

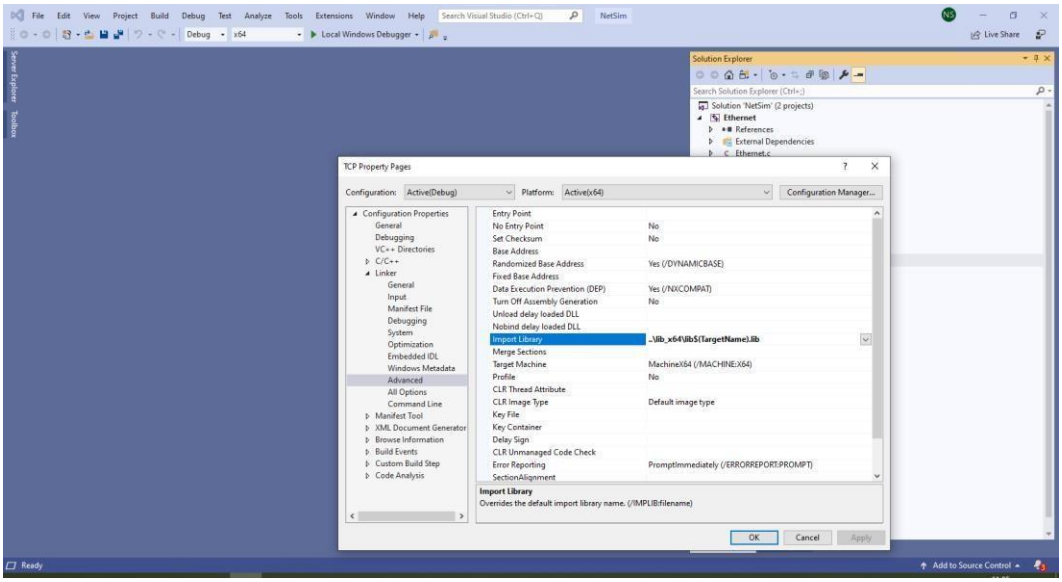


10. Modified the following lines of code in fn_NetSim_Ethernet_HandlePhyOut() function present in Ethernet_Phy.c file inside Ethernet project.

```
Ethernet.h | TCP.h | SYN_flood.c | Ethernet_Phy.c*
Ethernet | (Global Scope) | fn_NetSim_Ethernet_HandlePhyOut()

if (!packet)
    return 2; // No packet is there for transmission

double start;

if (pstruEventDetails->nDeviceId == target_node && (packet->nControlDataType == 40102 || packet->nControlDataType == 40105))
{
    if (phy->lastPacketEndTime + phy->IFG <= pstruEventDetails->dEventTime)
        start = pstruEventDetails->dEventTime + Processing_TIME;
    else
        start = phy->lastPacketEndTime + phy->IFG + Processing_TIME;
}
else
{
    if (phy->lastPacketEndTime + phy->IFG <= pstruEventDetails->dEventTime)
        start = pstruEventDetails->dEventTime;
    else
        start = phy->lastPacketEndTime + phy->IFG;
}
```

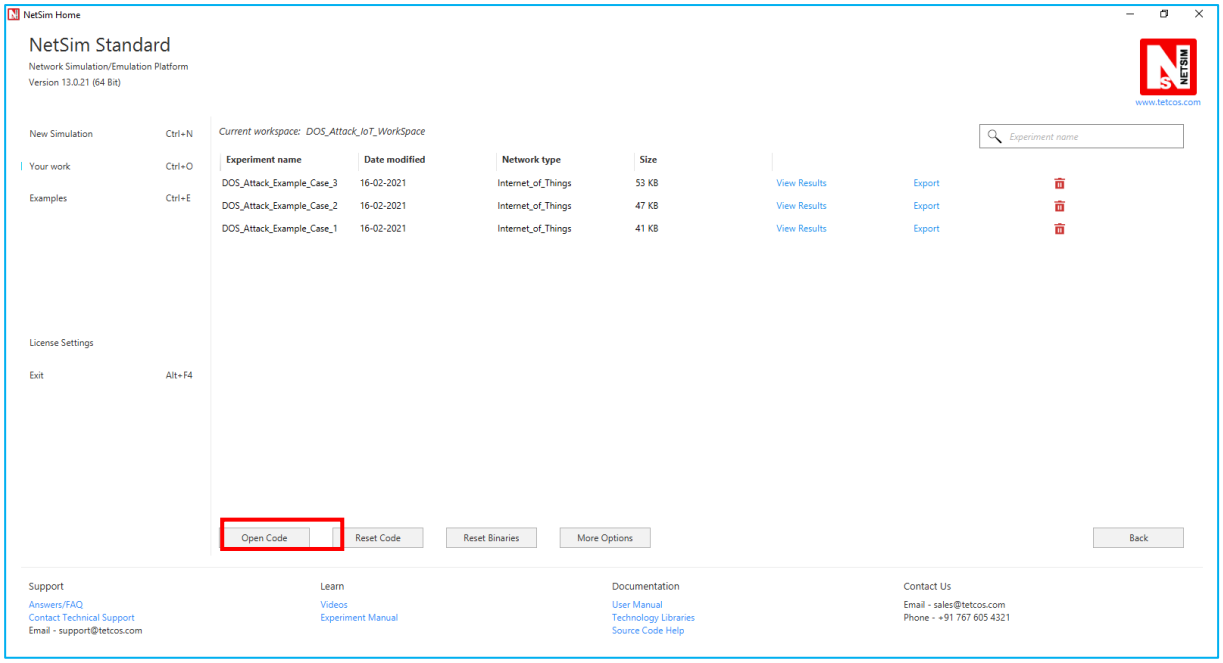
11. Right click on TCP project Properties Linker Advanced import library 32-bit and 64-bit

    ..\lib\lib$(TargetName).lib** or **..\lib_x64\lib$(TargetName).lib



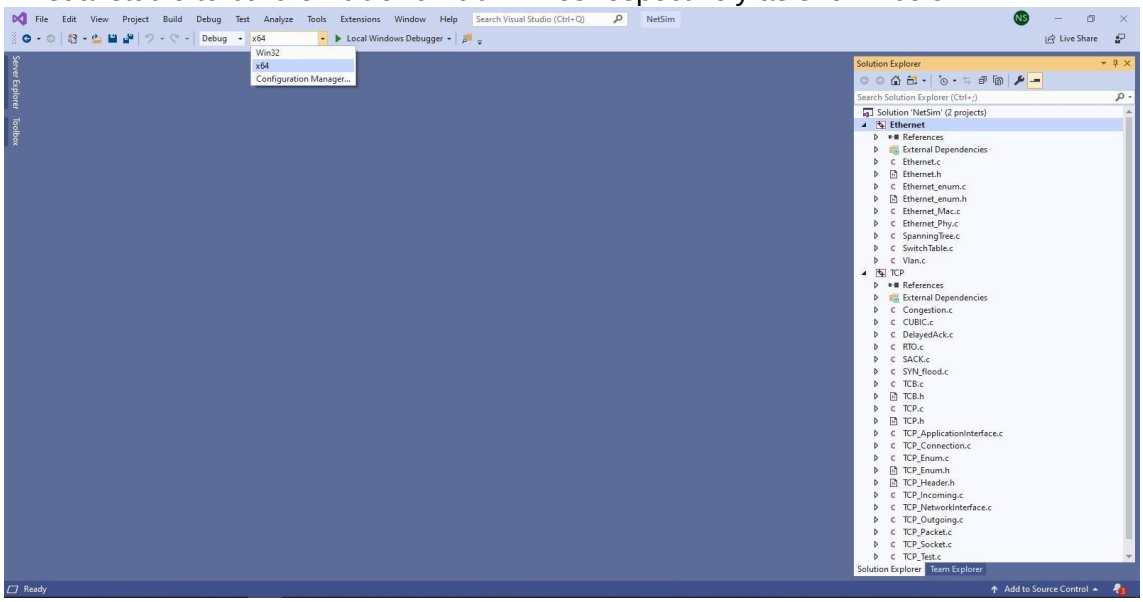
Steps:

1. Open the Source codes in Visual Studio by going to Your work-> Workspace Options and Clicking on Open code button as shown below:

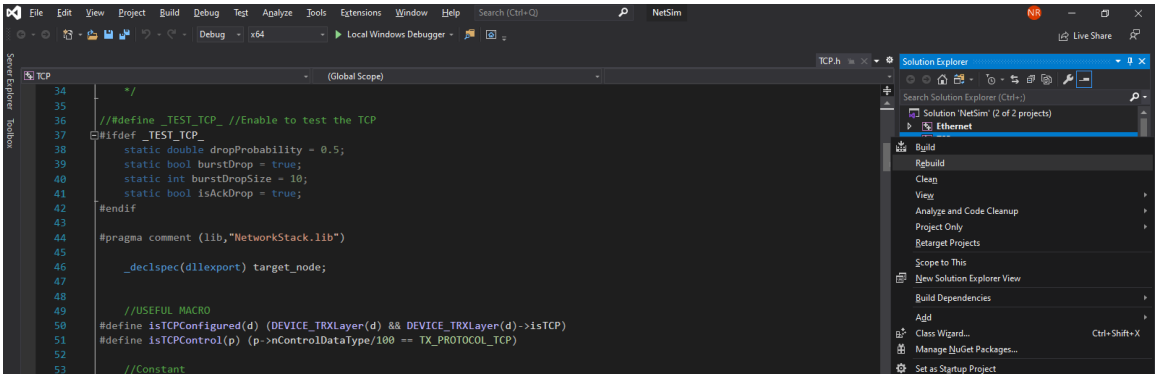


2. Under the **TCP** project in the solution explorer you will be able to see that **SYN_FLOOD.c** file.

3. 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:



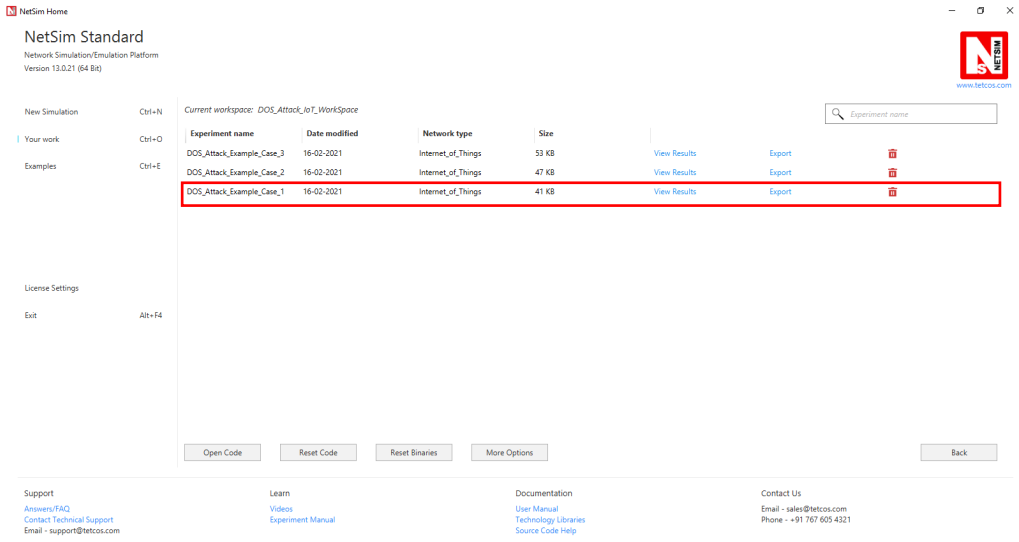
4. Right click on the solution in the solution explorer and select Rebuild (Note: first rebuild the TCP project and then rebuild the Ethernet project)



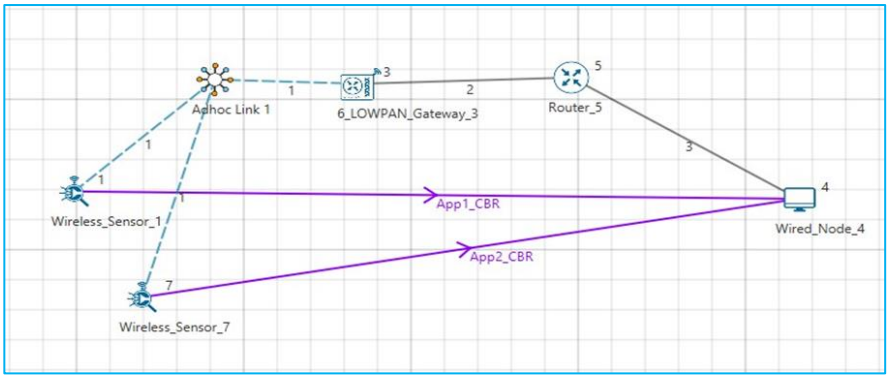
5. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries.

Case-1: Without Malicious Node

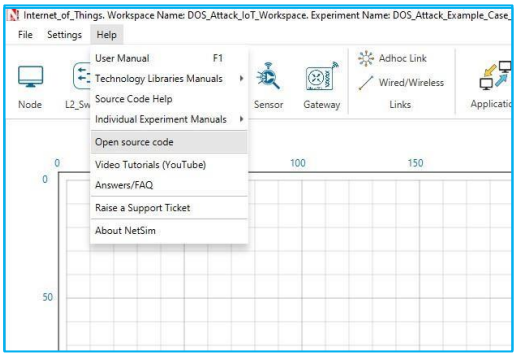
1. Then DOS_Attack_IoT_Workspace comes with a sample configuration that is already saved. To open this example, go to Your work and click on the DOS_Attack_Example_Case_1 that is present under the list of experiments as shown below:



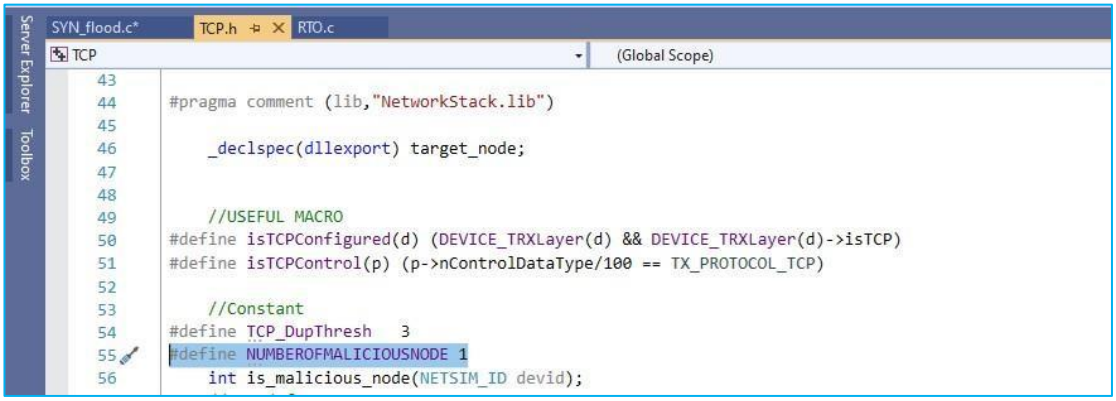
2. The saved network scenario consisting of 2 sensors, 1 6LOWPAN Gateway, 1 router, and 1 wired node in the grid environment forming a IoT Network. Traffic is configured from sensor node to the Wired Node.



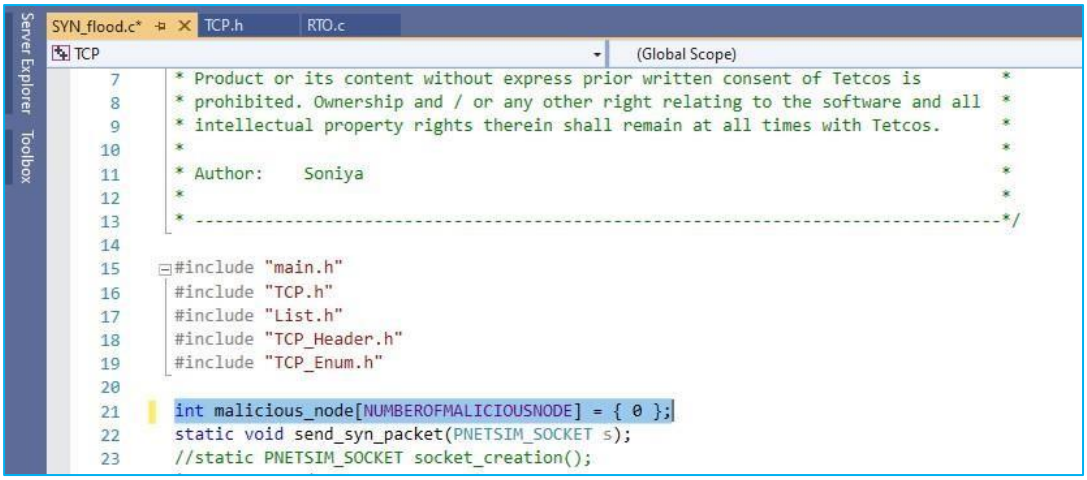
3. Help Open Source code



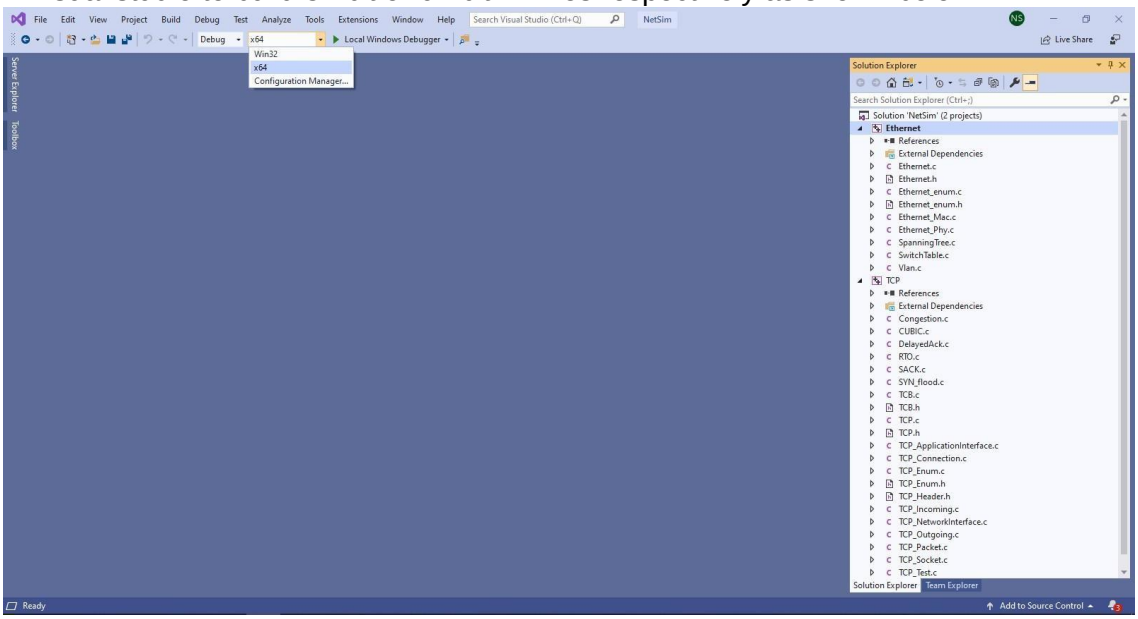
4. In TCP.h set **NUMBEROFMALICIOUSNODE** as 1.



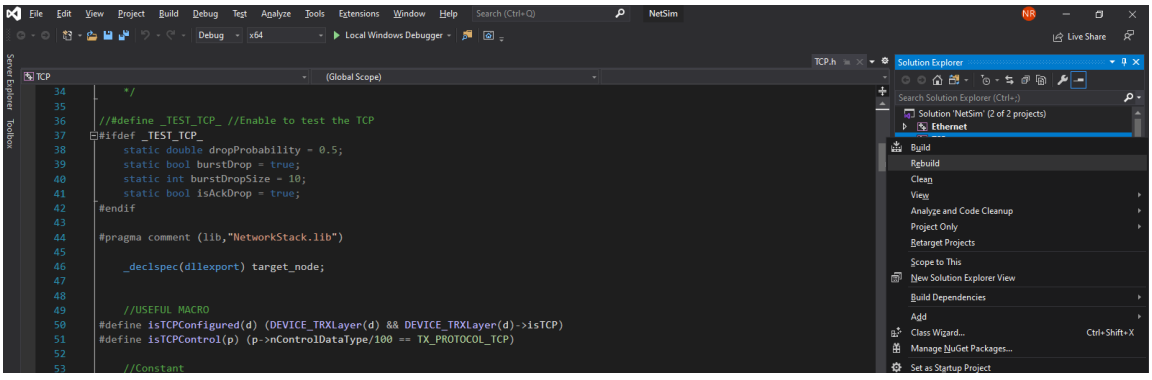
5. In SYN_FLOOD.c set **malicious node** as 0**.**



6. 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:



7. Right click on the solution in the solution explorer and select Rebuild. (Note: first rebuild the TCP project and then rebuild the Ethernet project).

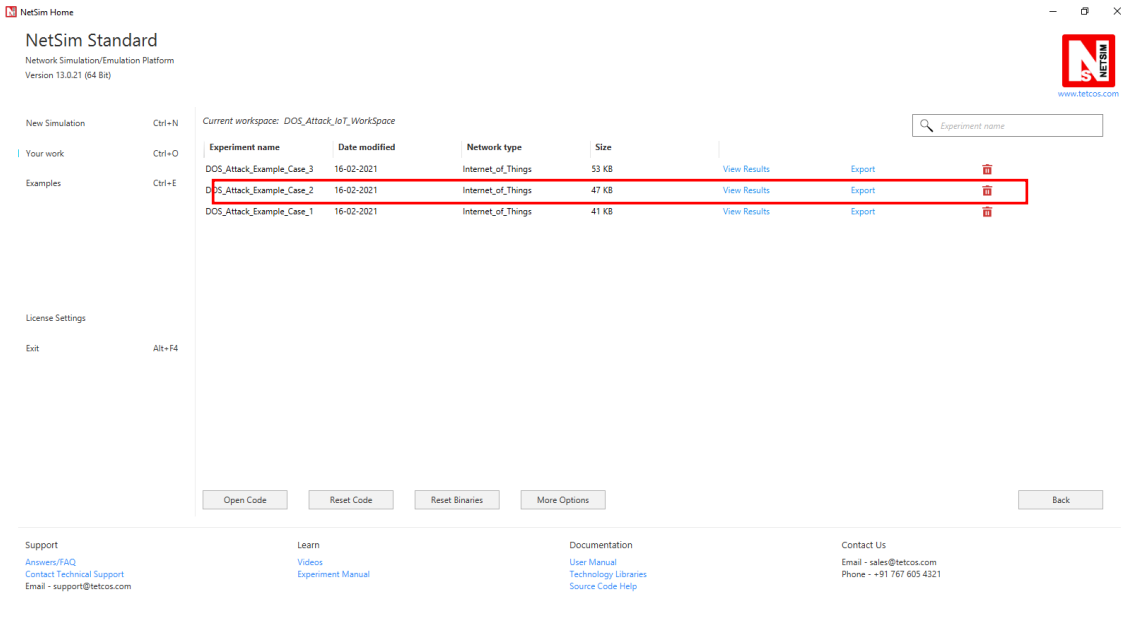


8. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries.

9. Run the simulation for 100 seconds.

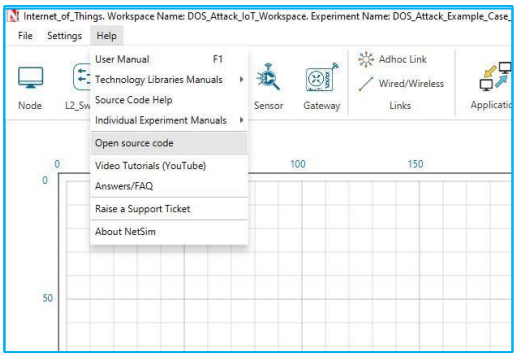
Case-2: With one Malicious Node

1. Then DOS_Attack_IoT_Workspace comes with a sample configuration that is already saved. To open this example, go to Your work and click on the DOS_Attack_Example_Case_2 that is present under the list of experiments as shown below:

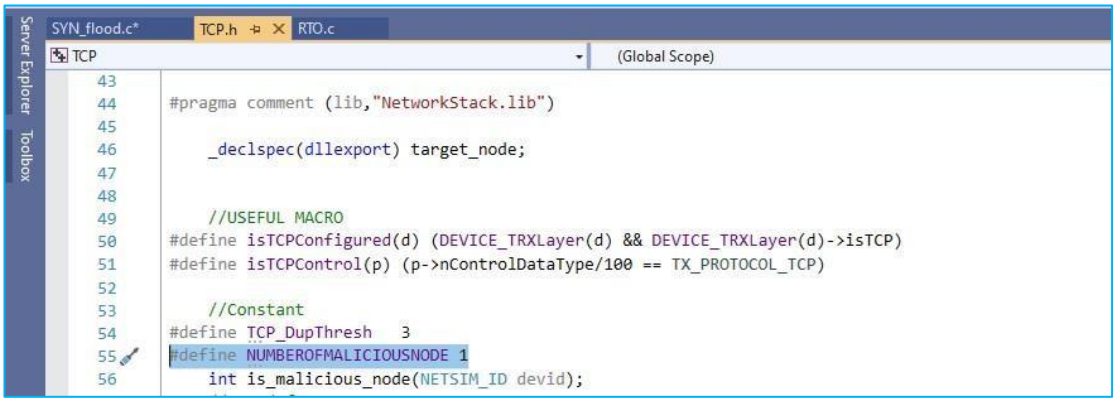


2. The saved network scenario consisting of 3 sensors, 1 6LOWPAN Gateway, 1 router, and 1 wired node in the grid environment forming a IoT Network. Traffic is configured from sensor node to the Wired Node.

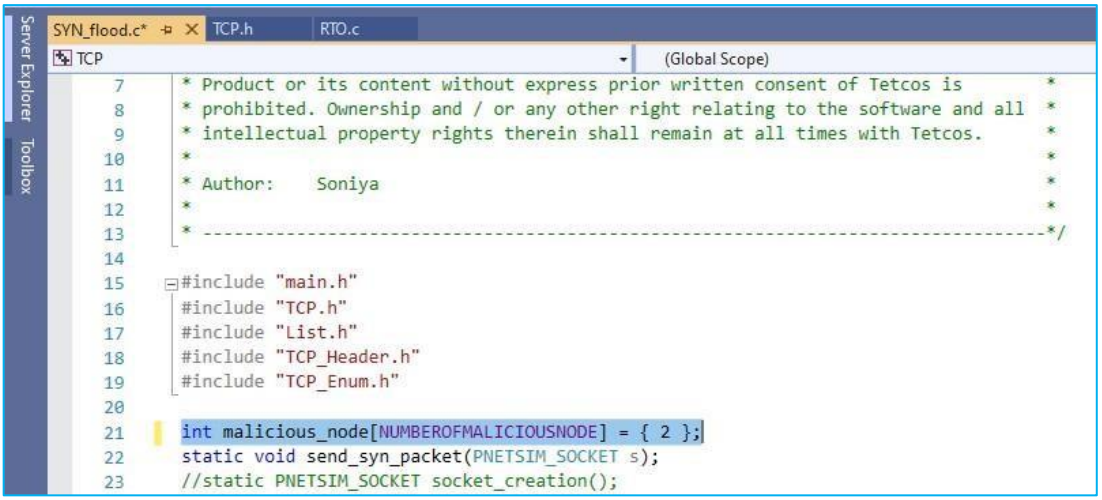
3. Help Open Source code



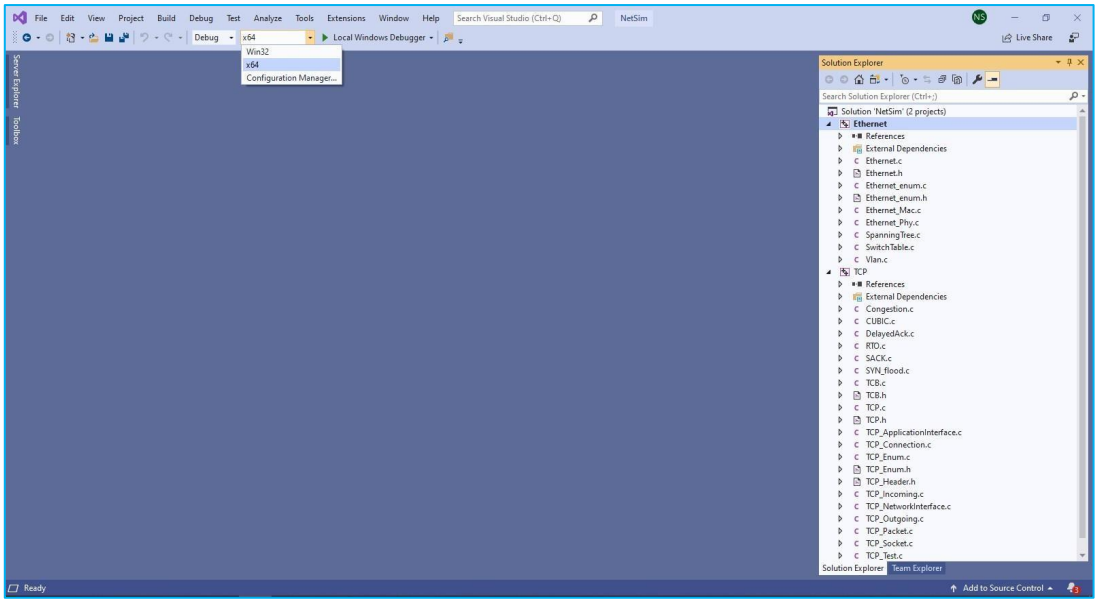
4. In TCP.h set **NUMBEROFMALICIOUSNODE** as 1.



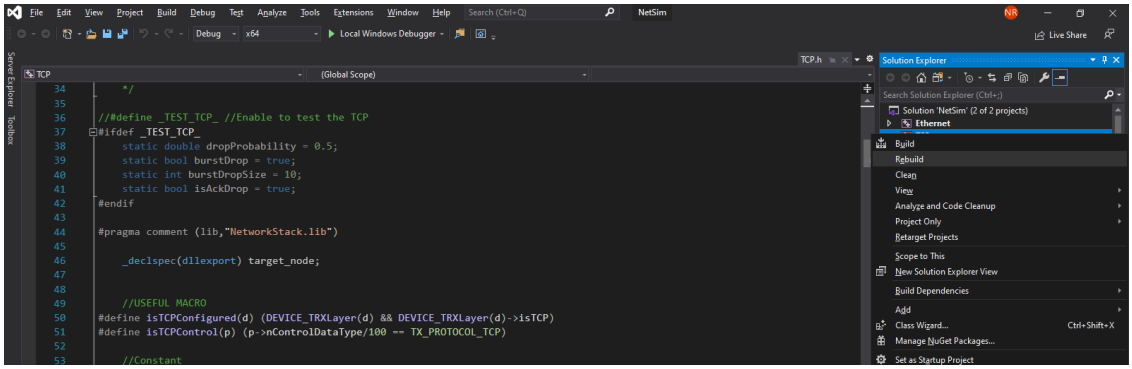
5. In SYN_FLOOD.c set **malicious node** as 2**.**



6. 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:



7. Right click on the solution in the solution explorer and select Rebuild.(Note: first rebuild the TCP project and then rebuild the Ethernet project)

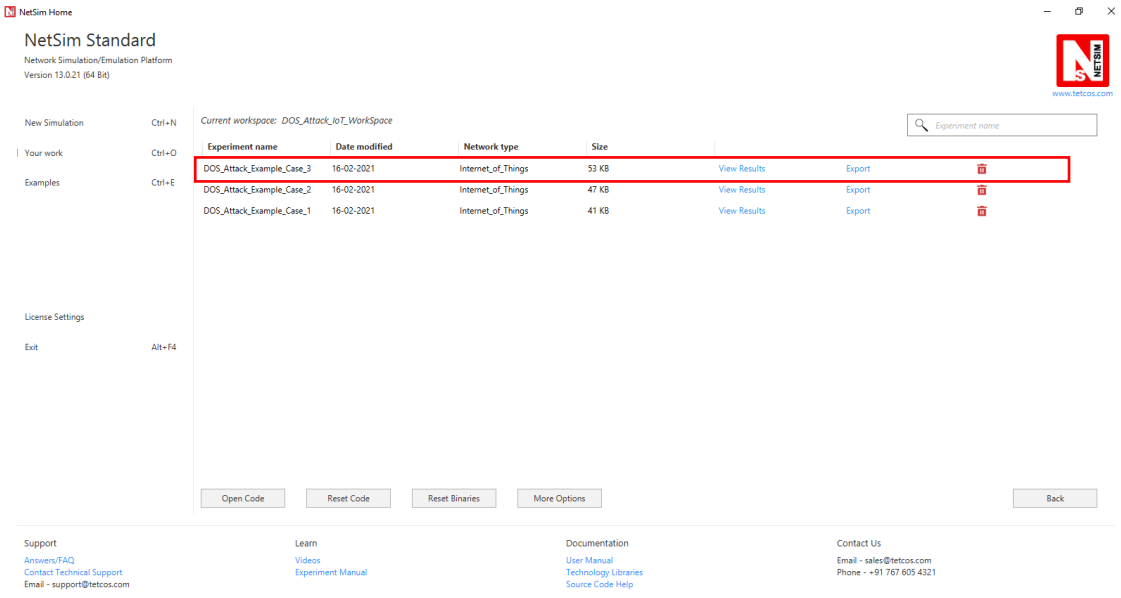


8. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries.

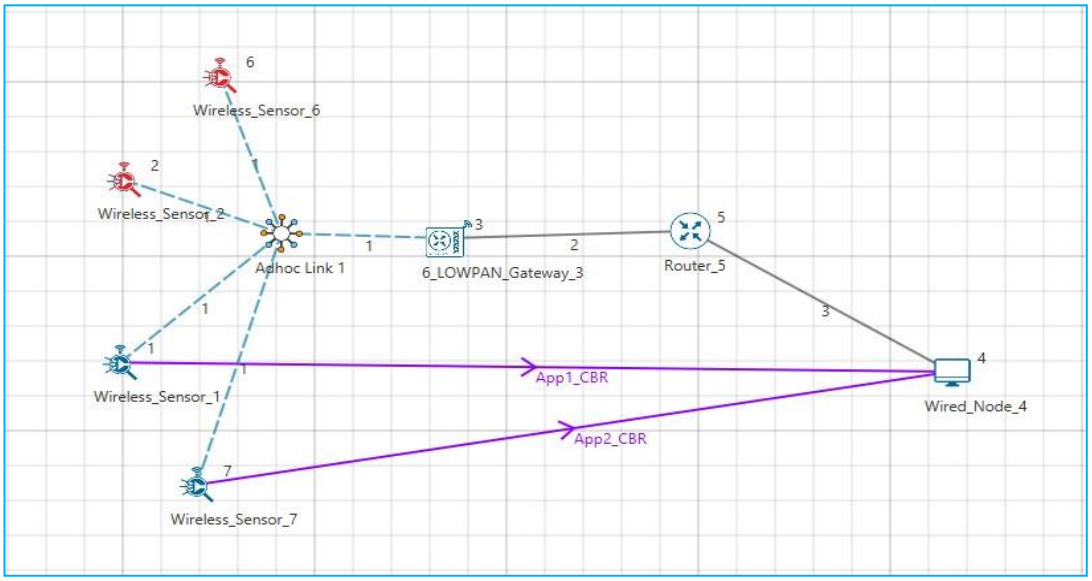
9. Run the simulation for 100 seconds.

Case-3: With two Malicious Node

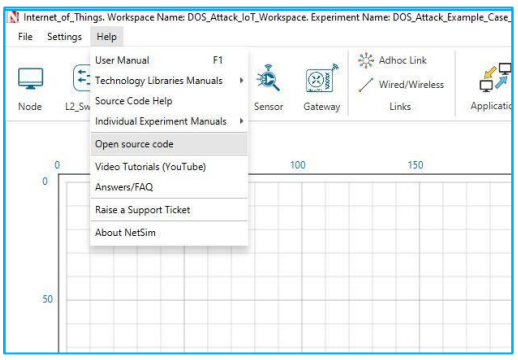
1. Then DOS_Attack_IoT_Workspace comes with a sample configuration that is already saved. To open this example, go to Your work and click on the DOS_Attack_Example_Case_3 that is present under the list of experiments as shown below:



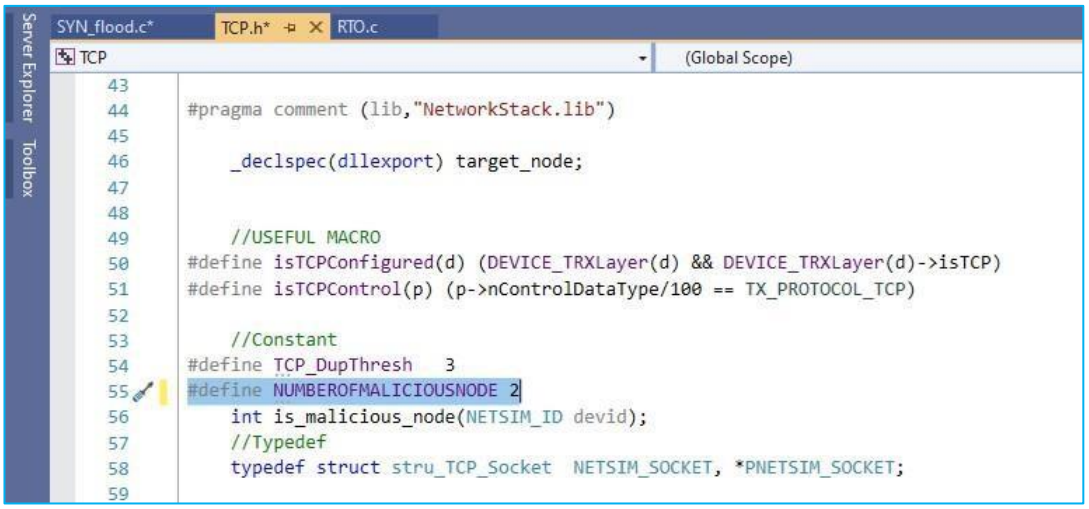
2. The saved network scenario consisting of 4 sensors, 1 6LOWPAN Gateway, 1 router, and 1 wired node in the grid environment forming a IoT Network. Traffic is configured from sensor node to the Wired Node.



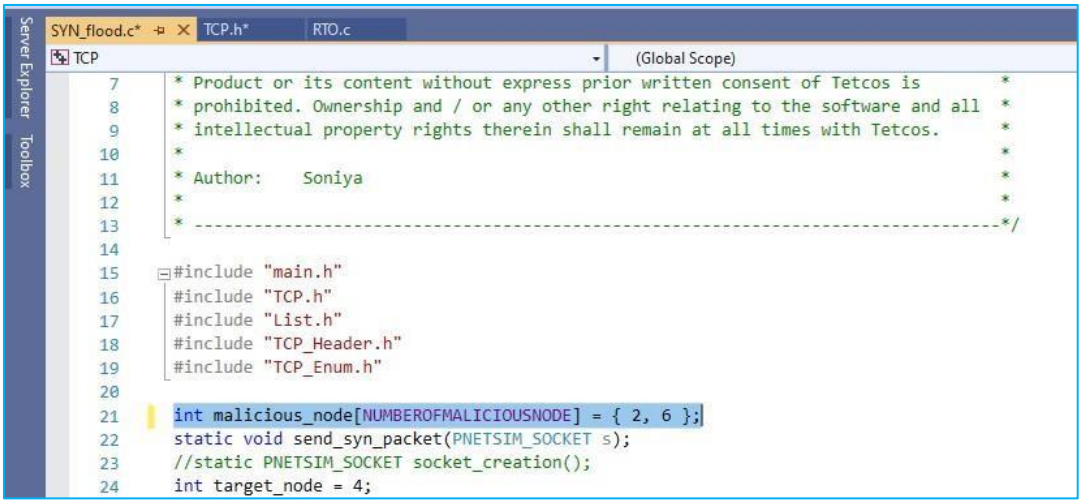
3. Help Open Source code



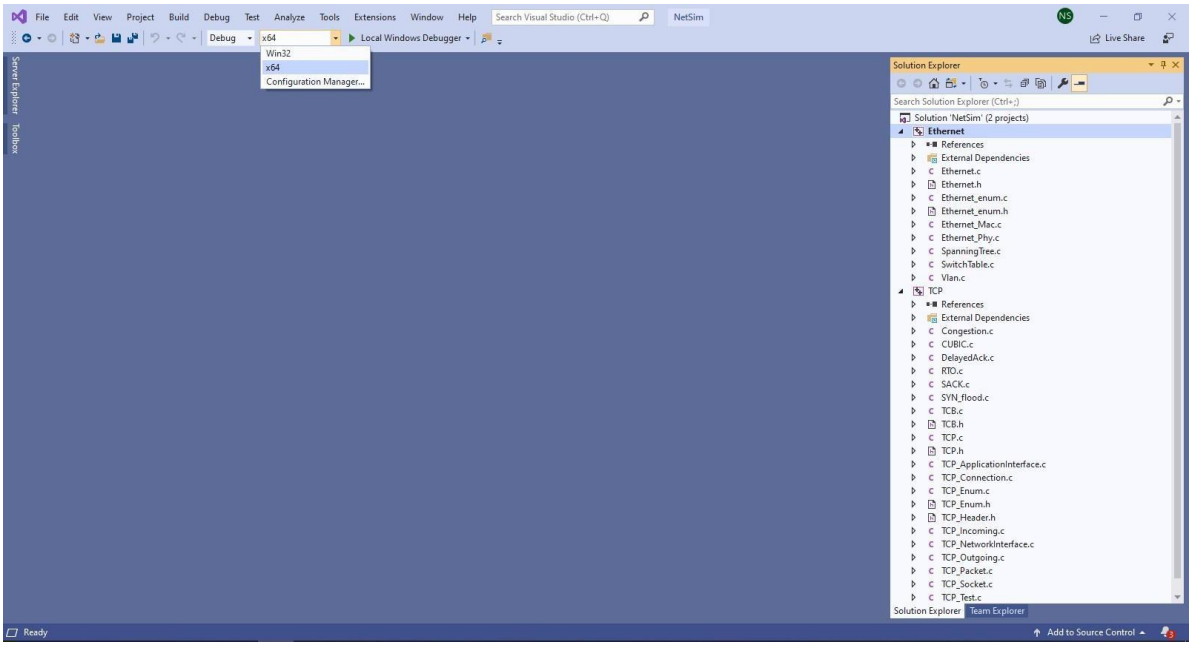
4. In TCP.h set **NUMBEROFMALICIOUSNODE** as 2.



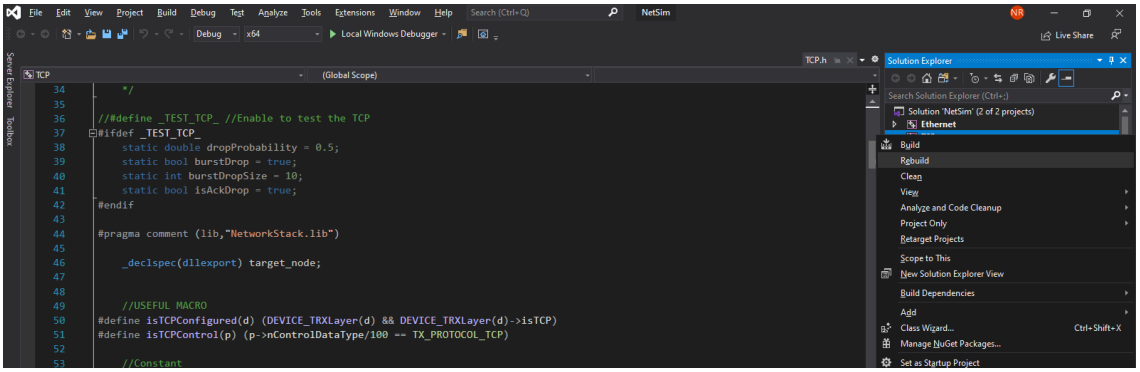
5. In SYN_FLOOD.c set **malicious node** as 2, 6**.**



6. 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:



7. Right click on the solution in the solution explorer and select Rebuild.(Note: first rebuild the TCP project and then rebuild the Ethernet project)

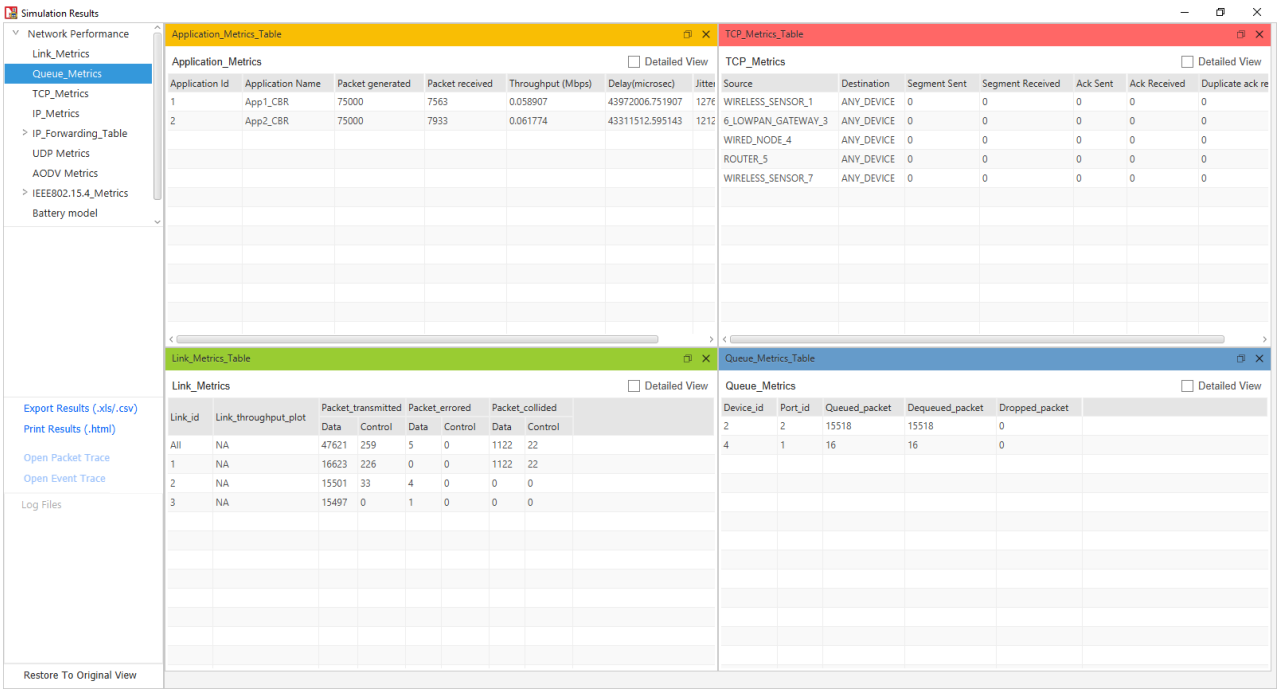


8. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries.

9. Run the simulation for 100 seconds.

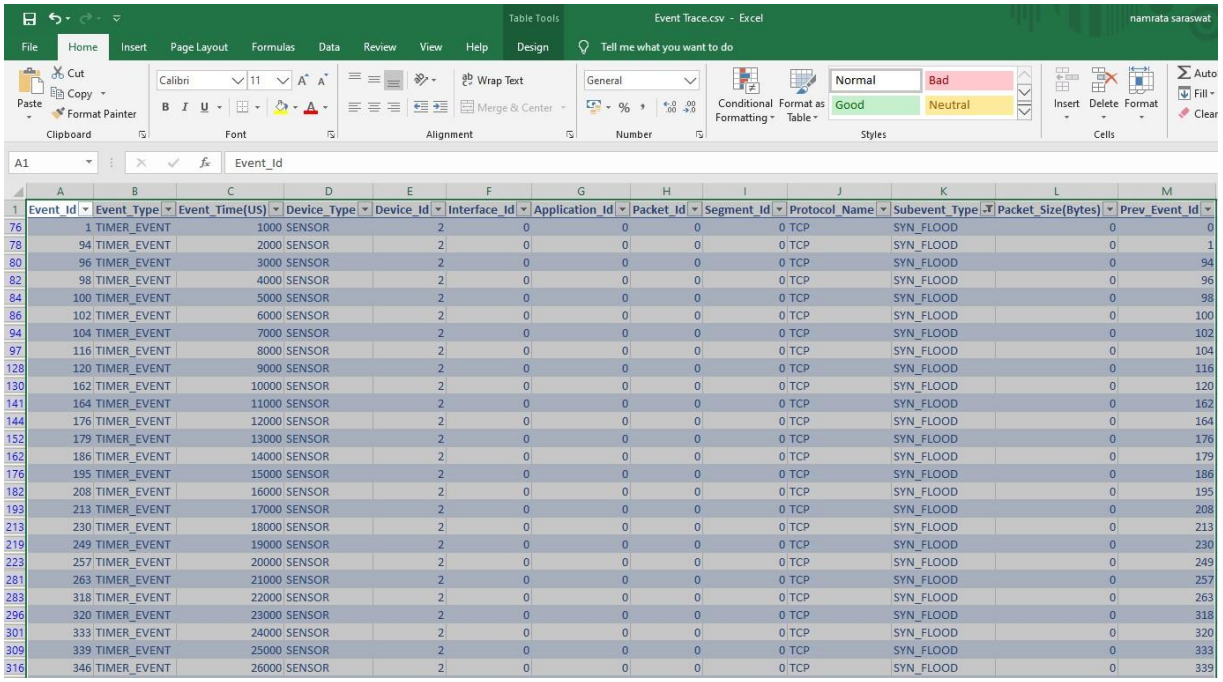
Result:

After simulation, open metrics window and observe the Application_Throughput is decreasing for both applications as we increase the malicious node because of the SYN flood sent from the malicious node. In case 1 there is no malicious node so there will be no SYN_FLOOD packets.



	Throughput_APP1 (Mbps)	Throughput_APP2(Mbps)
Case-1: Malicious Node =0	0.06	0.06
Case-2: Malicious Node =1	0.05	0.05
Case-3: Malicious Node =2	0.04	0.04

Go to the result window open Event trace, user can find out the SYN_FLOOD packets via filtering subevent type as SYN_FLOOD.



Event Trace.csv - Excel

Event_Id	Event_Type	Event_Time(US)	Device_Type	Device_Id	Interface_Id	Application_Id	Packet_Id	Segment_Id	Protocol_Name	Subevent_Type	Packet_Size(Bytes)	Prev_Event_Id
1	TIMER_EVENT	1000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	0
94	TIMER_EVENT	2000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	1
96	TIMER_EVENT	3000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	94
98	TIMER_EVENT	4000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	96
100	TIMER_EVENT	5000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	98
102	TIMER_EVENT	6000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	100
104	TIMER_EVENT	7000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	102
116	TIMER_EVENT	8000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	104
120	TIMER_EVENT	9000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	116
162	TIMER_EVENT	10000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	120
164	TIMER_EVENT	11000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	162
176	TIMER_EVENT	12000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	164
179	TIMER_EVENT	13000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	176
186	TIMER_EVENT	14000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	179
195	TIMER_EVENT	15000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	186
208	TIMER_EVENT	16000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	195
213	TIMER_EVENT	17000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	208
230	TIMER_EVENT	18000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	213
249	TIMER_EVENT	19000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	230
257	TIMER_EVENT	20000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	249
263	TIMER_EVENT	21000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	257
318	TIMER_EVENT	22000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	263
320	TIMER_EVENT	23000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	318
333	TIMER_EVENT	24000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	320
339	TIMER_EVENT	25000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	333
346	TIMER_EVENT	26000	SENSOR	2	0	0	0	0	TCP	SYN_FLOOD	0	339

Note: Users can also create their own network scenarios in Internet of Things and run simulation.