Dos Attack in Internetworks

Software Used: NetSim Standard v12.1 (32/64 bit), Visual Studio 2015/2017/2019

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.

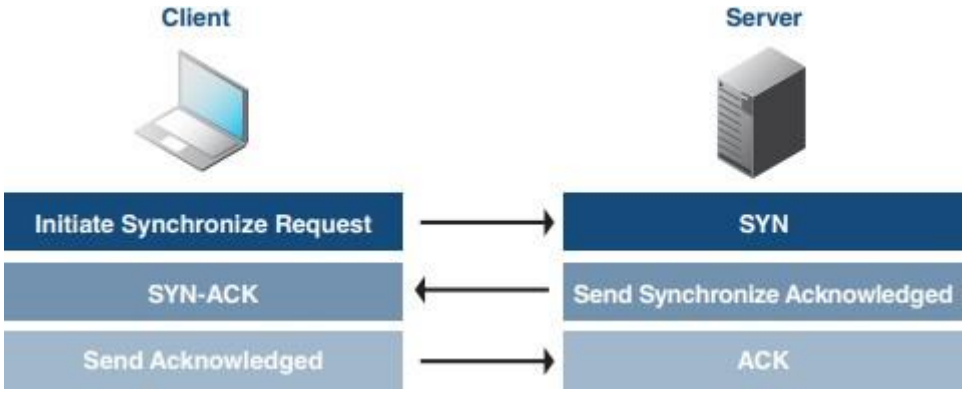
Standard DDoS Attack types:

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);

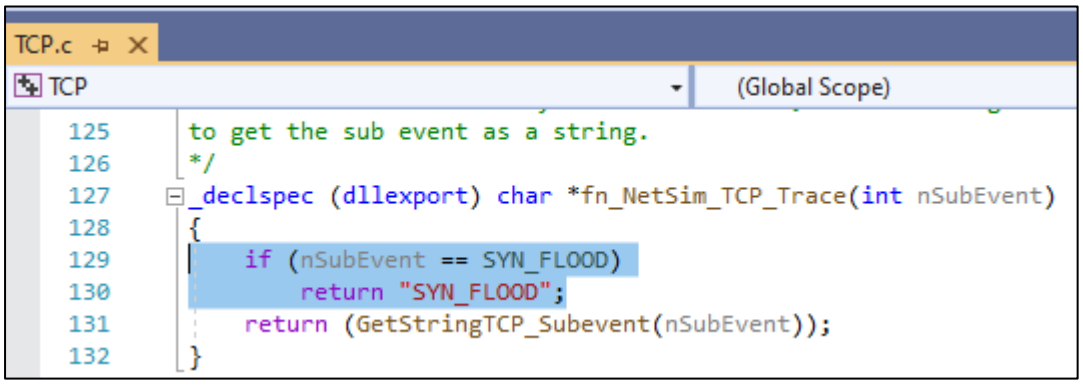
  This function is used to create and send SYN packet to the network layer

- 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



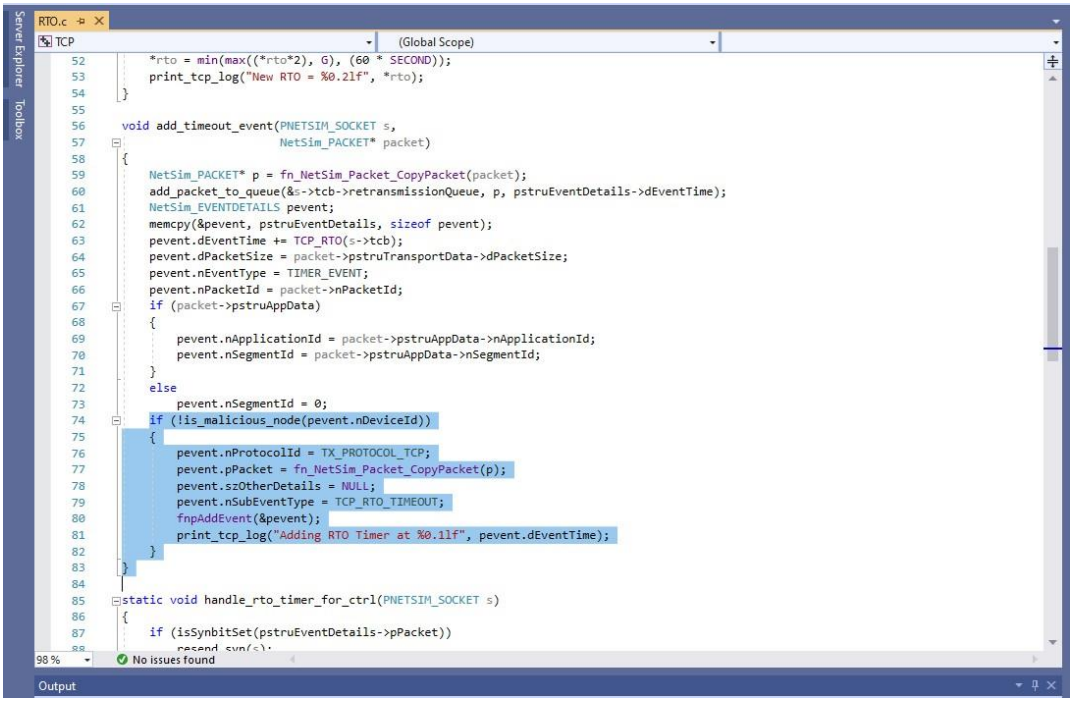
2. We have added the following lines of code in fn_NetSim_TCP_HandleTimer() function present in TCP.c file inside TCP project. Used to add a TCP sub_event called SYN_FLOOD

```
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  }
```

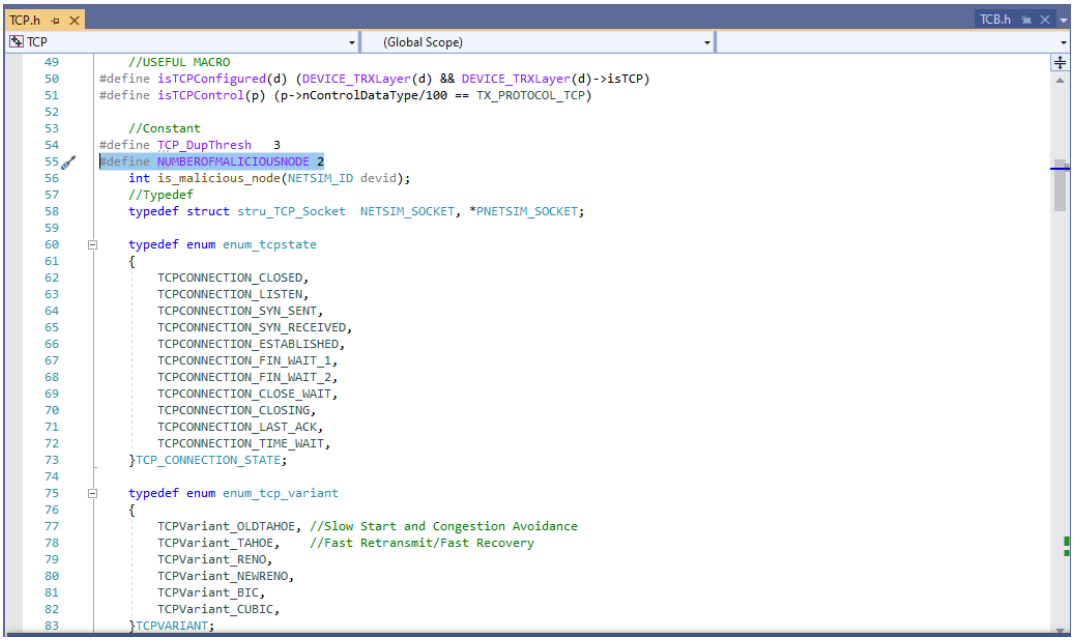
3. And modified the following lines of code in fn_NetSim_TCP_Init() function resent in TCP.c inside TCP project

```
TCP.c
TCP                    (Global Scope)                    fn_NetSim_TCP_Init(stru_NetSim_Network * NETWORK_For
79   */
80   _declspec (dllexport) int fn_NetSim_TCP_Init(struct stru_NetSim_Network* NETWORK_Formal,
81       NetSim_EVENTDETAILS* pstruEventDetails_Formal,
82       char* pszAppPath_Formal,
83       char* pszWritePath_Formal,
84       int nVersion_Type,
85       void** fnPointer)
86   {
87       fn_NetSim_TCP_Init_F(NETWORK_Formal,
88           pstruEventDetails_Formal,
89           pszAppPath_Formal,
90           pszWritePath_Formal,
91           nVersion_Type,
92           fnPointer);
93       NetSim_EVENTDETAILS pevent;
94       memcpy(&pevent, pstruEventDetails, sizeof pevent);
95
96       for (int i = 0; i < NETWORK->nDeviceCount; i++)
97       {
98           if (is_malicious_node(i + 1))
99           {
100              pevent.nDeviceId = i + 1;
101              pevent.dEventTime += 1000;
102              pevent.nEventType = TIMER_EVENT;
103              pevent.nSubEventType = SYN_FLOOD;
104              pevent.nProtocolId = TX_PROTOCOL_TCP;
105              fnpAddEvent(&pevent);
106
107          }
108      }
109      return 0;
110  }
111
112  /**
113  This function is called by NetworkStack.dll, once simulation end to free the
114  allocated memory for the network.
115  */
98 %   No issues found
Output
```

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



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



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

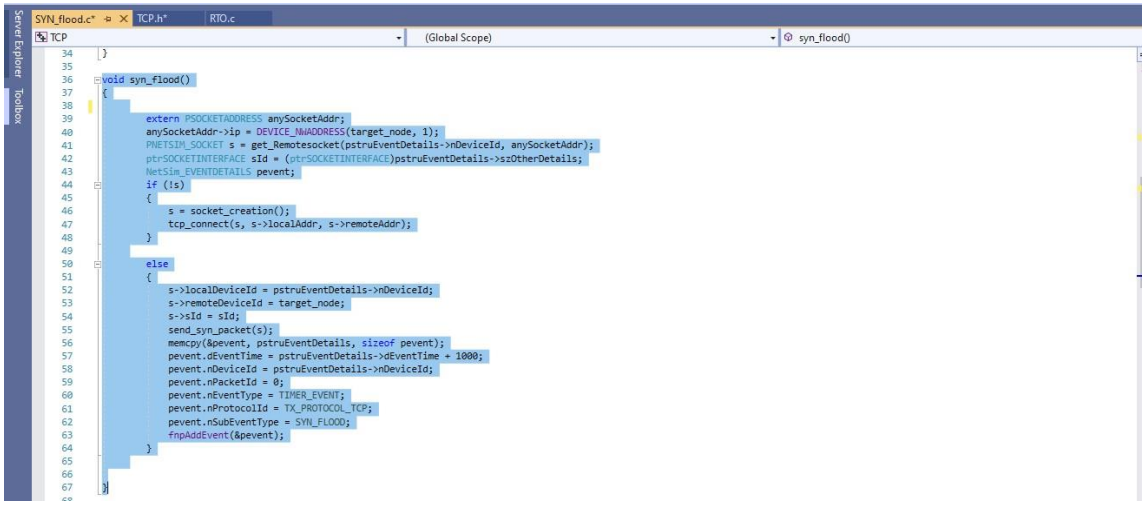
```
SYN_flood.c                                                    TCB.h
TCP                                          (Global Scope)
 13   * ----------------------------------------------------------------------------*/
 14
 15   #include "main.h"
 16   #include "TCP.h"
 17   #include "List.h"
 18   #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

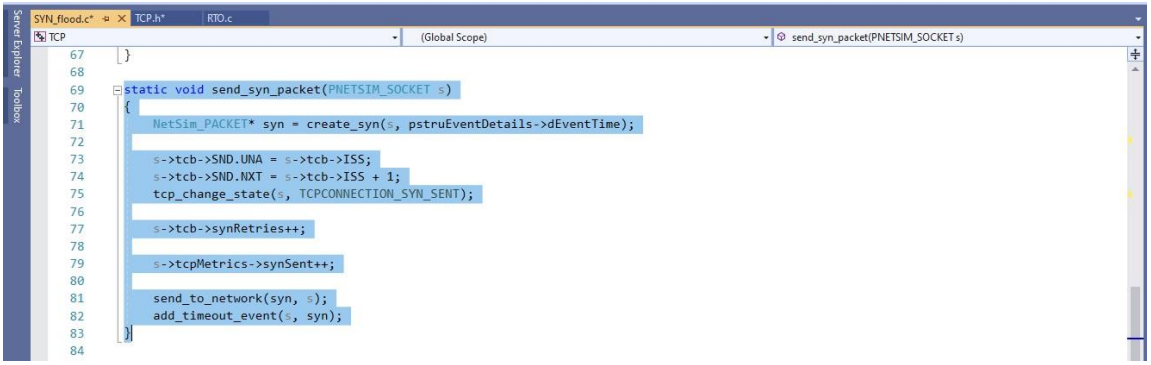
```
Server Explorer  Toolbox
TCP_Enum.h | TCP.h | RTO.c | SYN_flood.c | TCP_Connection.c | TCP.c
TCP                                          (Global Scope)
    #include "EnumString.h"

    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

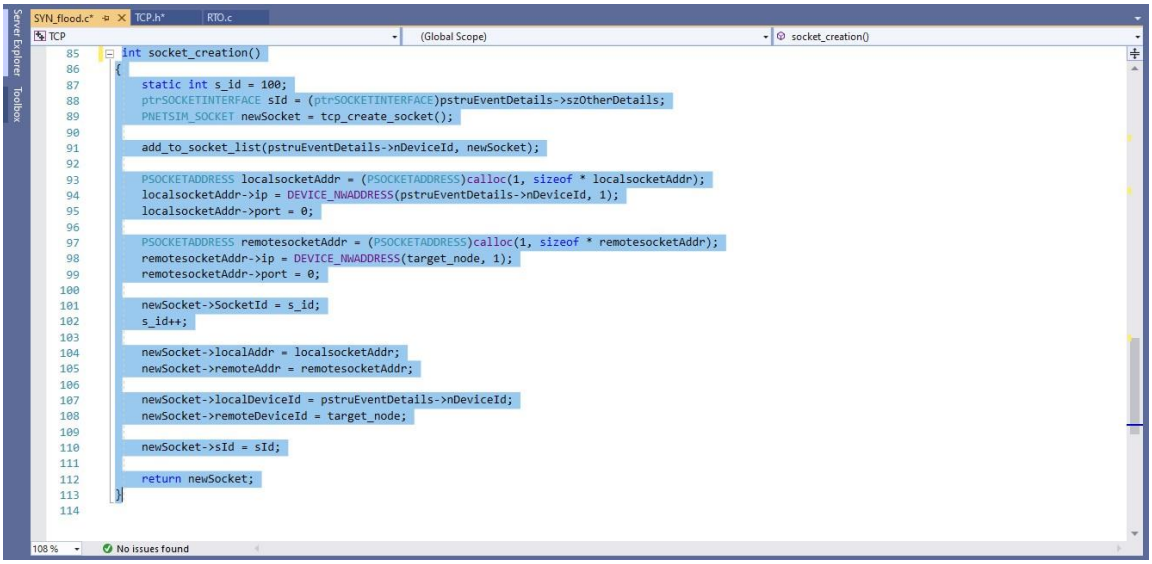
```
SYN_flood.c
TCP                                          (Global Scope)
 16   #include "TCP.h"
 17   #include "List.h"
 18   #include "TCP_Header.h"
 19   #include "TCP_Enum.h"
 20
 21   int malicious_node[NUMBEROFMALICIOUSNODE] = { 2, 6 };
 22   static void send_syn_packet(PNETSIM_SOCKET s);
 23   //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   }
```




```
}

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);
    }

}
```




```
}

static void send_syn_packet(PNETSIM_SOCKET s)
{
    NetSim_PACKET* syn = create_syn(s, pstruEventDetails->dEventTime);

    s->tcb->SND.UNA = s->tcb->ISS;
    s->tcb->SND.NXT = s->tcb->ISS + 1;
    tcp_change_state(s, TCPCONNECTION_SYN_SENT);

    s->tcb->synRetries++;

    s->tcpMetrics->synSent++;

    send_to_network(syn, s);
    add_timeout_event(s, syn);
}
```




```
int socket_creation()
{
    static int s_id = 100;
    ptrSOCKETINTERFACE sId = (ptrSOCKETINTERFACE)pstruEventDetails->szOtherDetails;
    PNETSIM_SOCKET newSocket = tcp_create_socket();

    add_to_socket_list(pstruEventDetails->nDeviceId, newSocket);

    PSOCKETADDRESS localsocketAddr = (PSOCKETADDRESS)calloc(1, sizeof * localsocketAddr);
    localsocketAddr->ip = DEVICE_NWADDRESS(pstruEventDetails->nDeviceId, 1);
    localsocketAddr->port = 0;

    PSOCKETADDRESS remotesocketAddr = (PSOCKETADDRESS)calloc(1, sizeof * remotesocketAddr);
    remotesocketAddr->ip = DEVICE_NWADDRESS(target_node, 1);
    remotesocketAddr->port = 0;

    newSocket->SocketId = s_id;
    s_id++;

    newSocket->localAddr = localsocketAddr;
    newSocket->remoteAddr = remotesocketAddr;

    newSocket->localDeviceId = pstruEventDetails->nDeviceId;
    newSocket->remoteDeviceId = target_node;

    newSocket->sId = sId;

    return newSocket;
}
```

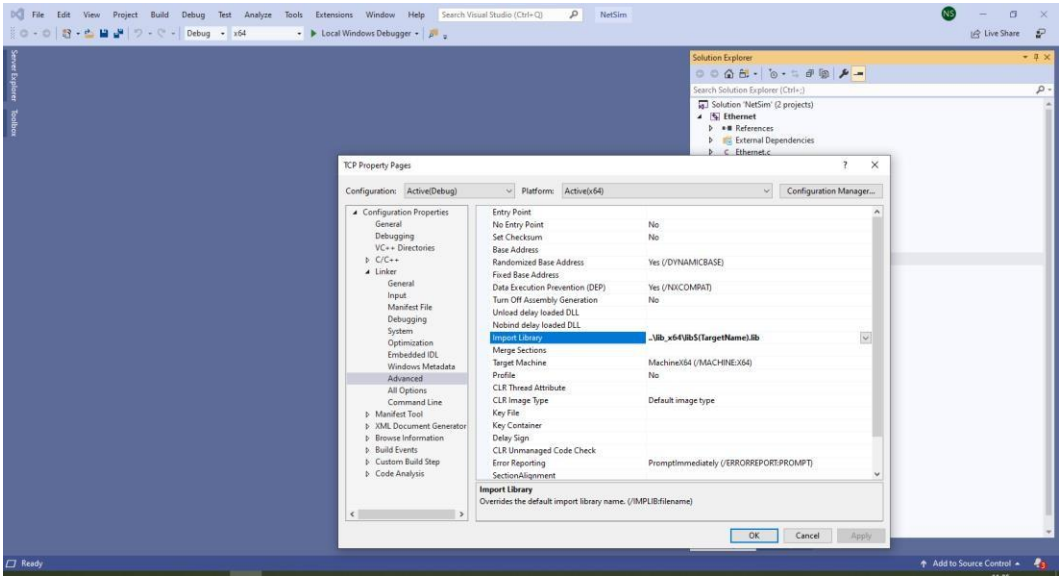
9. Added PROCESSING_TIME macro in Ethernet.h file inside ETHERNET project

```
Ethernet.h
Ethernet                                                    (Global Scope)
22  #pragma comment(lib,"Metrics.lib")
23  #pragma comment (lib,"libTCP")
24  #define isETHConfigured(d,i) (DEVICE_MACLAYER(d,i)->nMacProtocolId == MAC_PROTOCOL_IEEE802_3)
25      //Global variable
26      PNETSIM_MACADDRESS multicastSPTMAC;
27
28  #define ETH_IFG 0.960 //Micro sec
29
30  #define Processing_TIME 1000
31
32      typedef enum enum_eth_packet
33      {
34          ETH_CONFIGBPDU = MAC_PROTOCOL_IEEE802_3 * 100 + 1,
35      }ETH_PACKET;
36
37      /** Enumeration for Switching Technique */
38      typedef enum enum_SwitchingTechnique
39      {
40          SWITCHINGTECHNIQUE_NULL,
41          SWITCHINGTECHNIQUE_STORE_FORWARD,
42          SWITCHINGTECHNIQUE_CUT_THROUGH,
43          SWITCHINGTECHNIQUE_FRAGMENT_FREE,
44      }SWITCHING_TECHNIQUE;
45
```

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

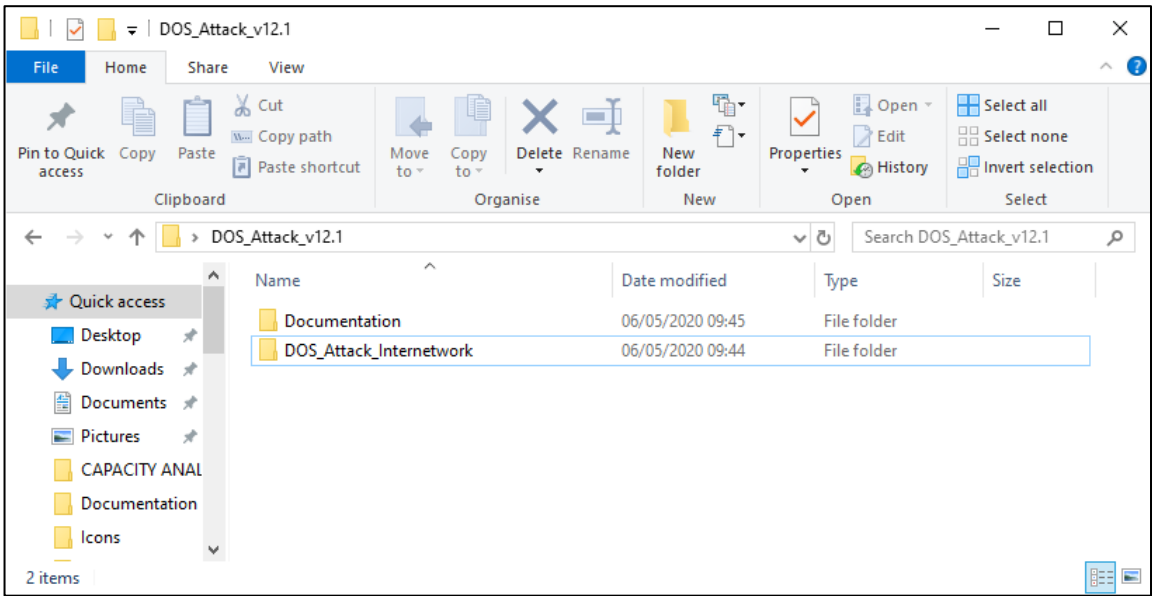
```
Ethernet.h   TCP.h   SYN_flood.c   Ethernet_Phy.c*
Ethernet                (Global Scope)                fn_NetSim_Ethernet_HandlePhyOut()
    if (!packet)
        return 2; // No packet is there for transmission

    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 ➐ Properties ➐ Linker ➐ Advanced ➐ import library 32-bit and 64-bit **..\lib\lib$(TargetName).lib** or **..\lib_x64\lib$(TargetName).lib**

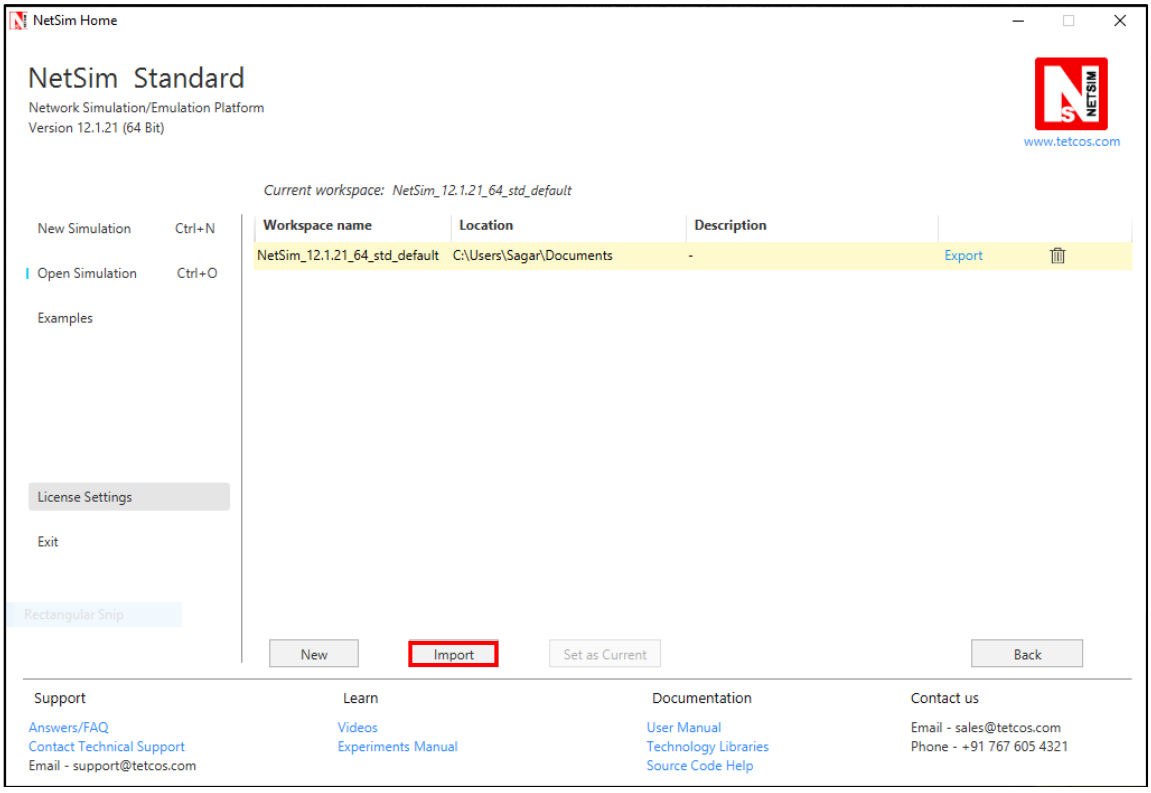


Steps:

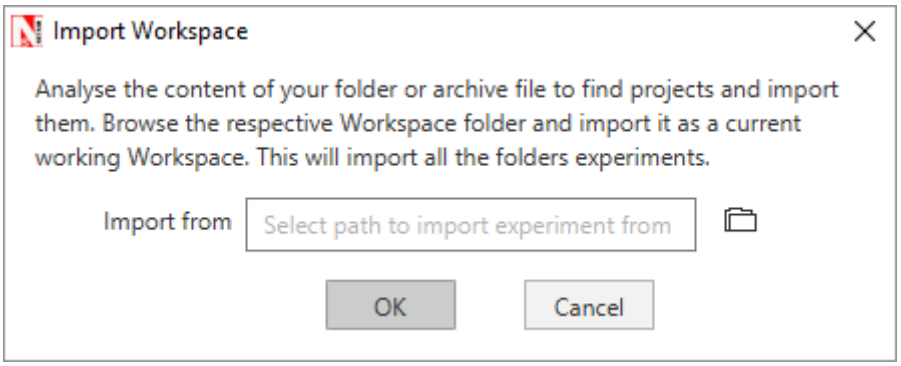
1. The downloaded project folder contains the folders Documentation, and DOS_Attack_Internetworks directory as shown below:



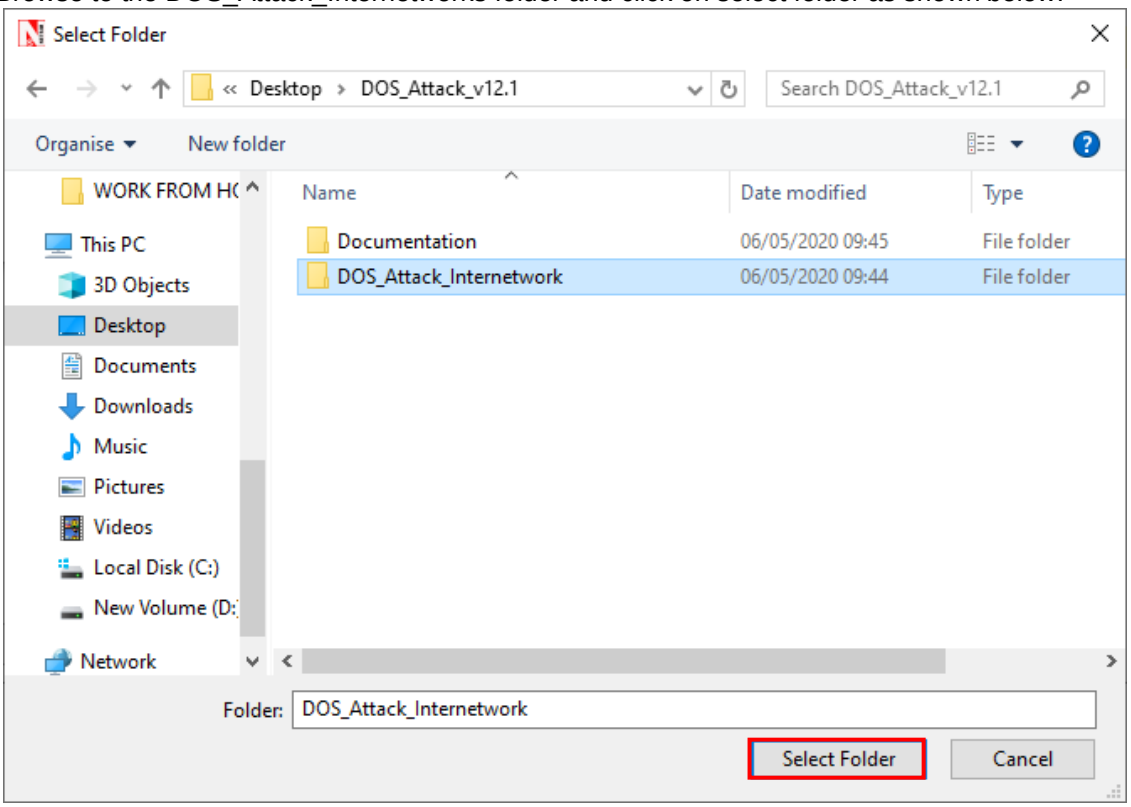
2. Import DOS_Attack_Internetworks by going to Open Simulation->Workspace Options->More Options in NetSim Home window. Then select Import as shown below:



3. It displays a window where users need to give the path of the workspace folder and click on OK as shown below:

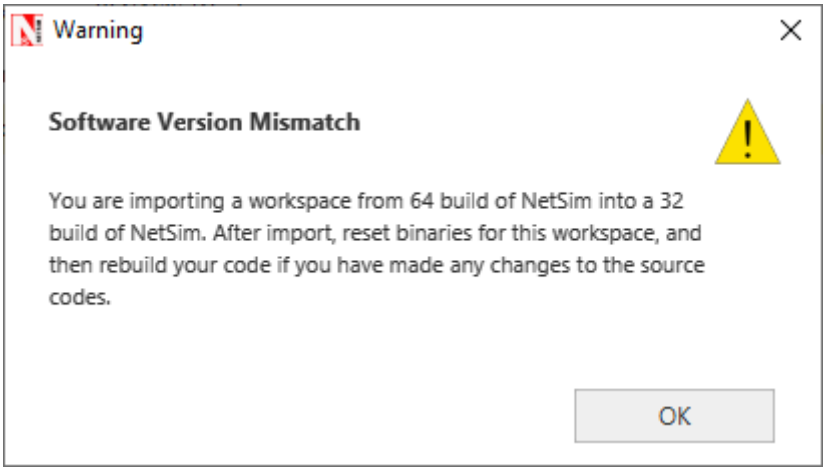


4. Browse to the DOS_Attack_Internetworks folder and click on select folder as shown below:

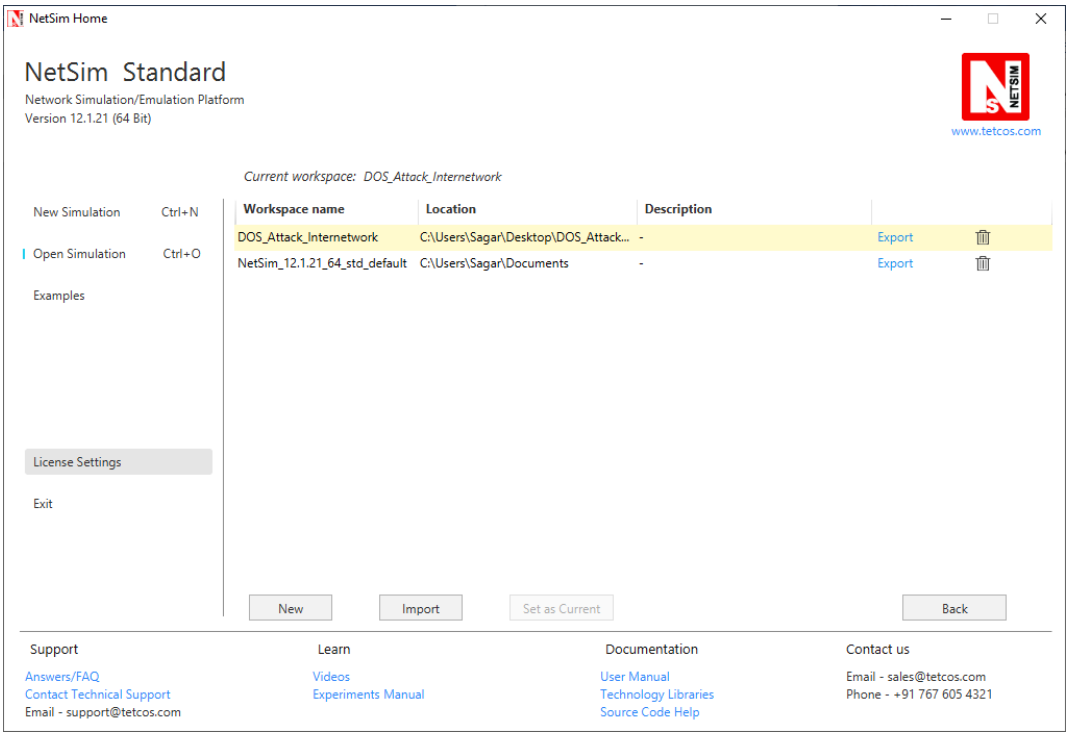


5. After this click on OK button in the Import Workspace window.

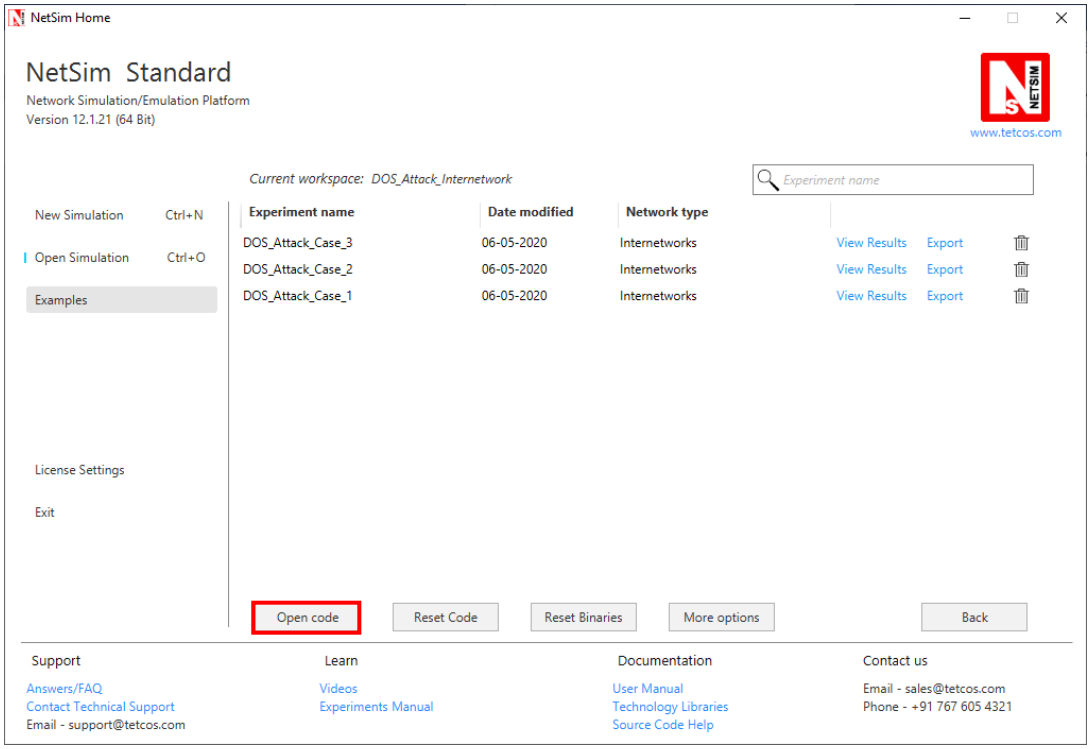
6. While importing the workspace, if the following warning message indicating Software Version Mismatch is displayed, you can ignore it and proceed.



7. The Imported workspace will be set as the current workspace automatically. To see the imported workspace, click on Open Simulation->Workspace Options->More Options as shown below:

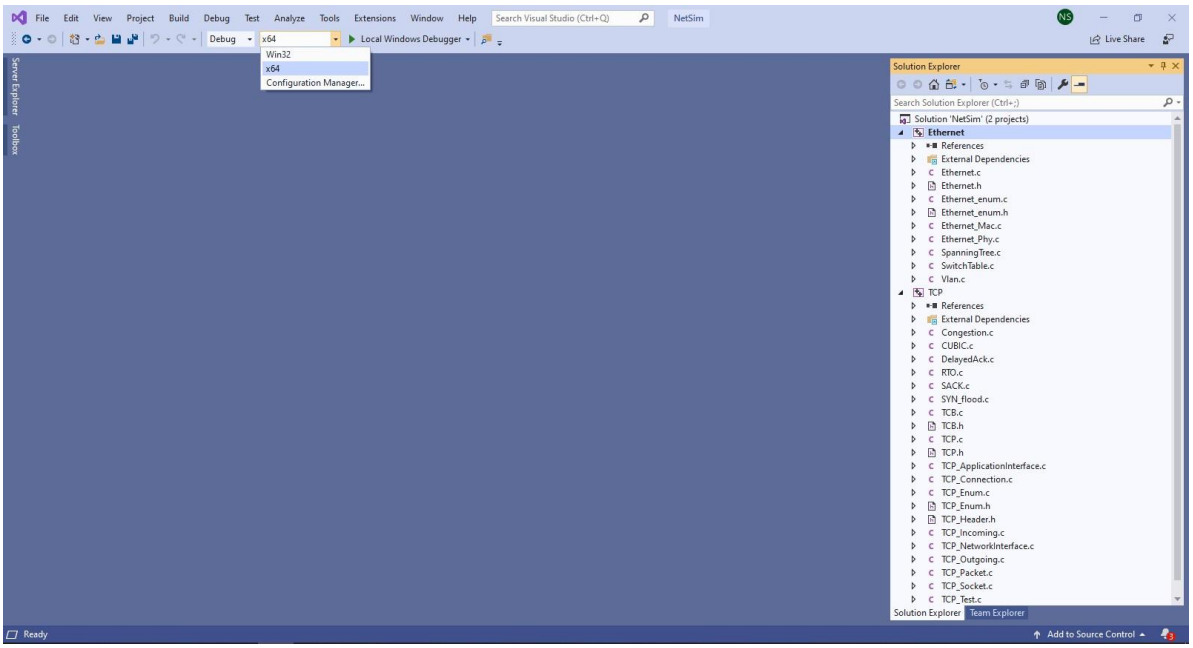


8. Open the Source codes in Visual Studio by going to Open Simulation-> Workspace Options and Clicking on Open code button as shown below:

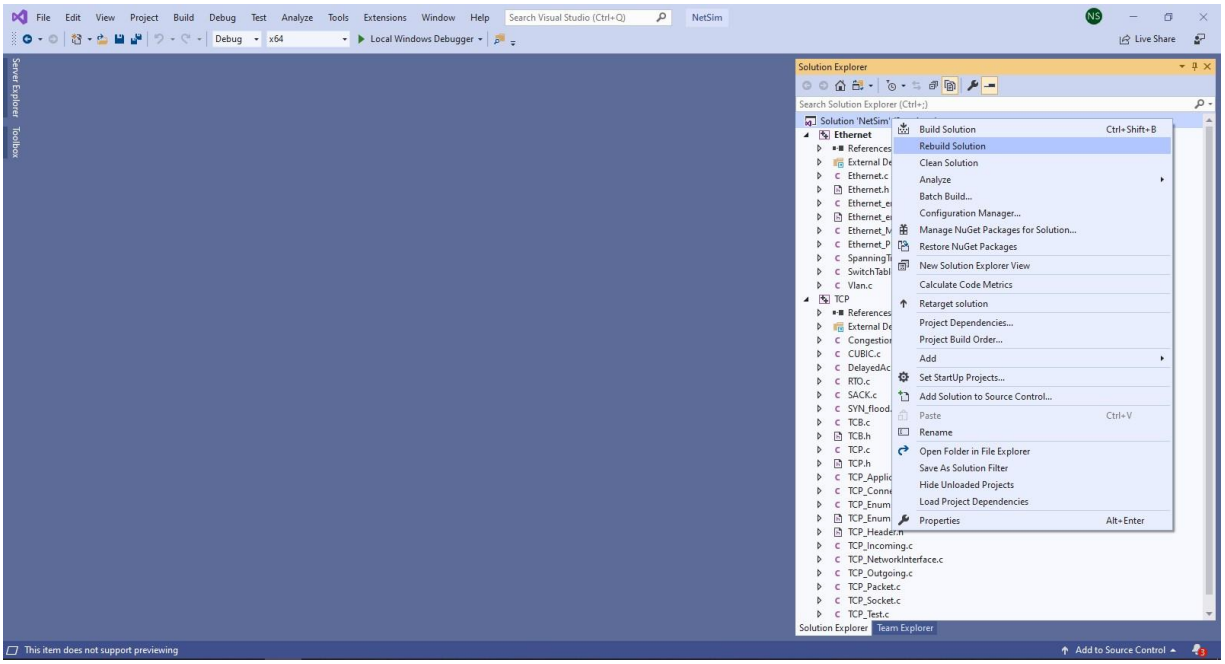


9. Under the **TCP** project in the solution explorer you will be able to see that **SYN_FLOOD.c** file.

10. 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:



11. Right click on the solution in the solution explorer and select Rebuild.

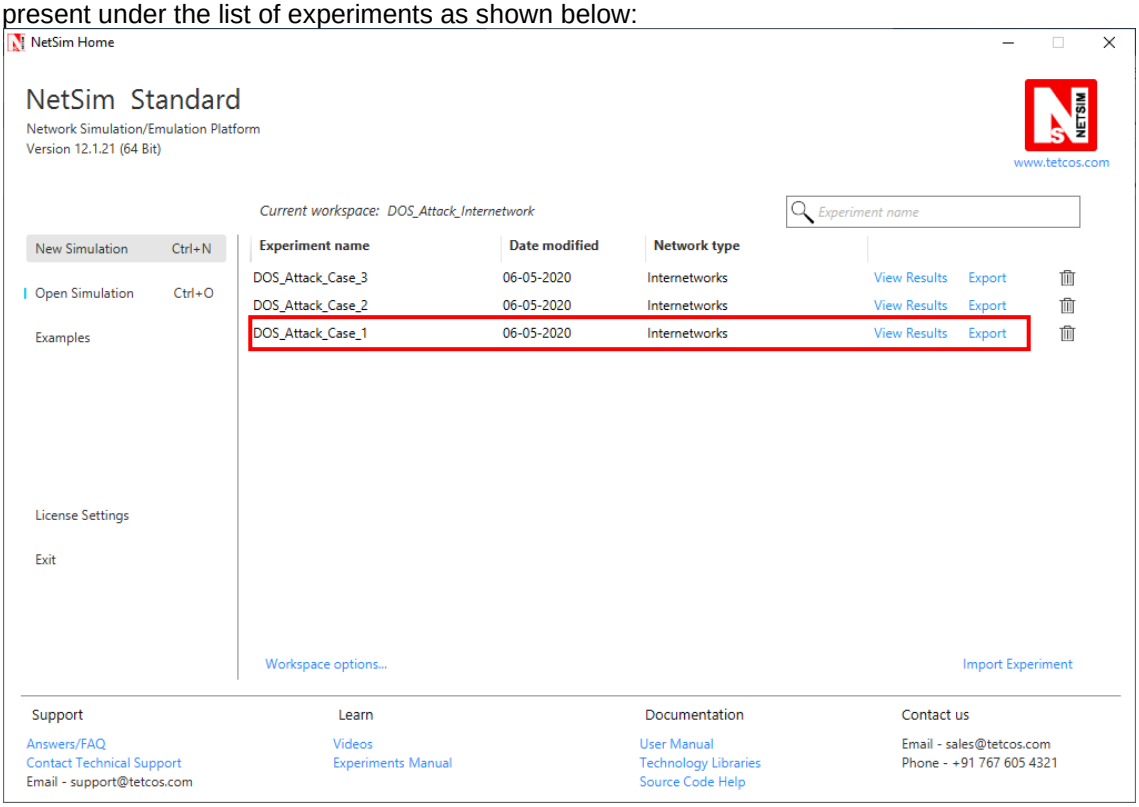


12. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries. (Note: first rebuild the TCP project and then rebuild the Ethernet project)

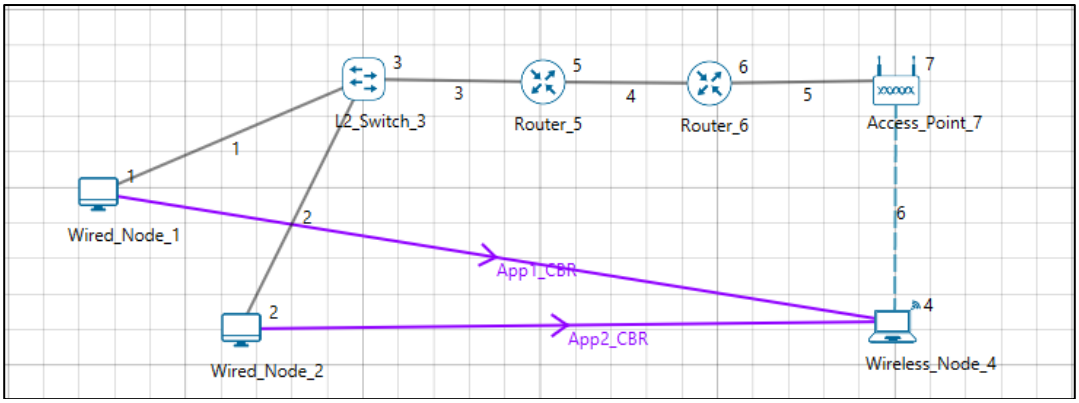
13. Run NetSim as Administrative mode.

Case-1: Without Malicious Node

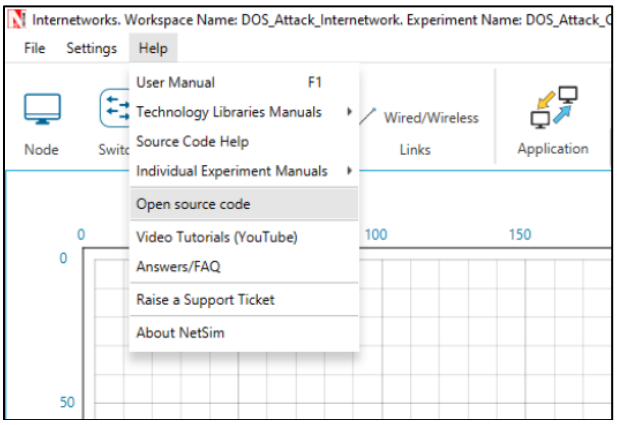
1. Then DOS_Attack_Internetworks comes with a sample configuration that is already saved. To open this example, go to Open Simulation and click on the DOS_Attack_Case_1 that is present under the list of experiments as shown below:



2. The saved network scenario consisting of 2 Wired Nodes, 1 L2 Switch, 2 router, 1 Access Point and 1 wireless node in the grid environment forming a internetworks Network. Traffic is configured from Wired node to the Wireless node.



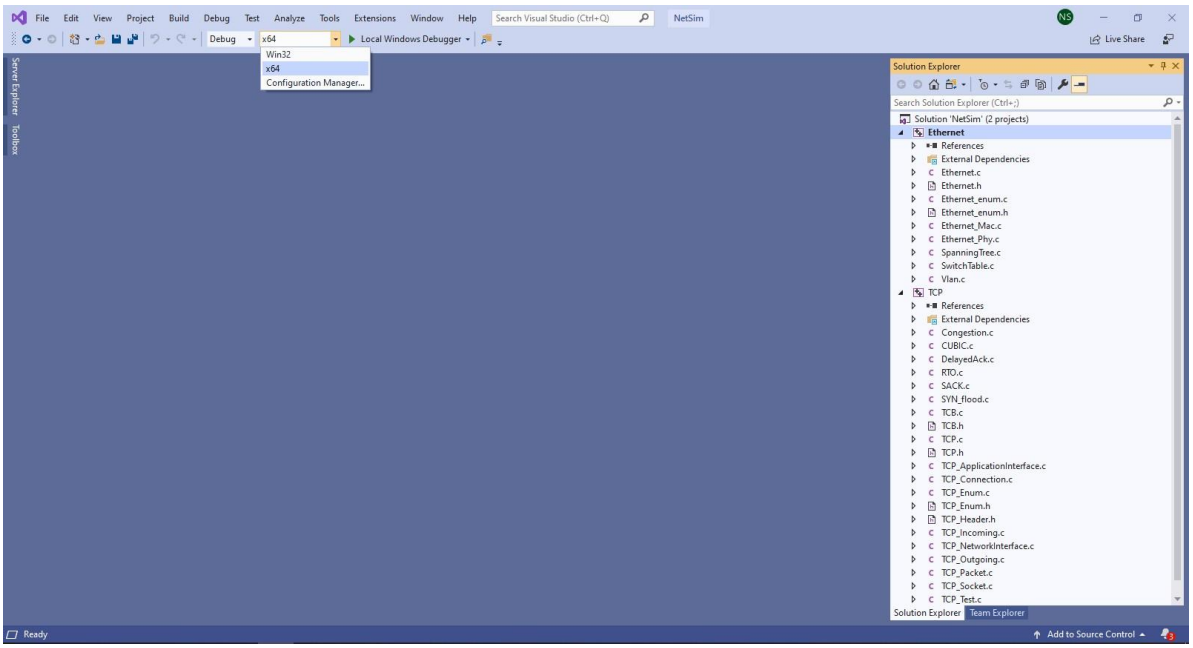
3. Help ➔ Open Source code



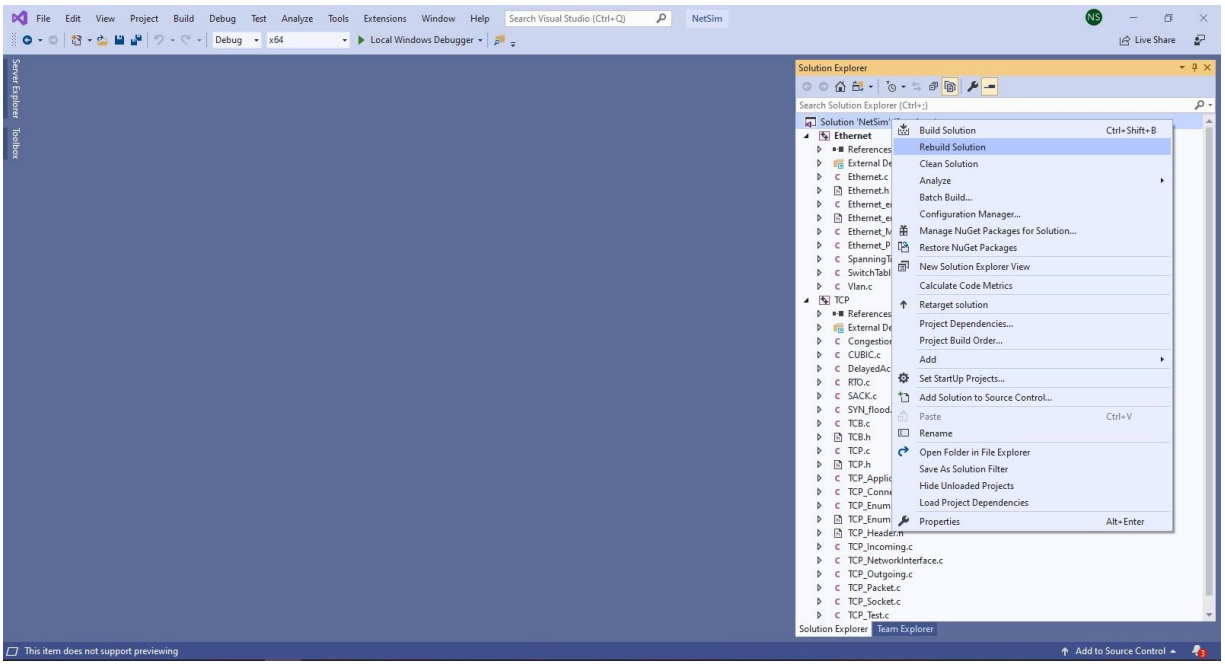
4. In TCP.h set **NUMBEROFMALICIOUSNODE** as 1.

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.

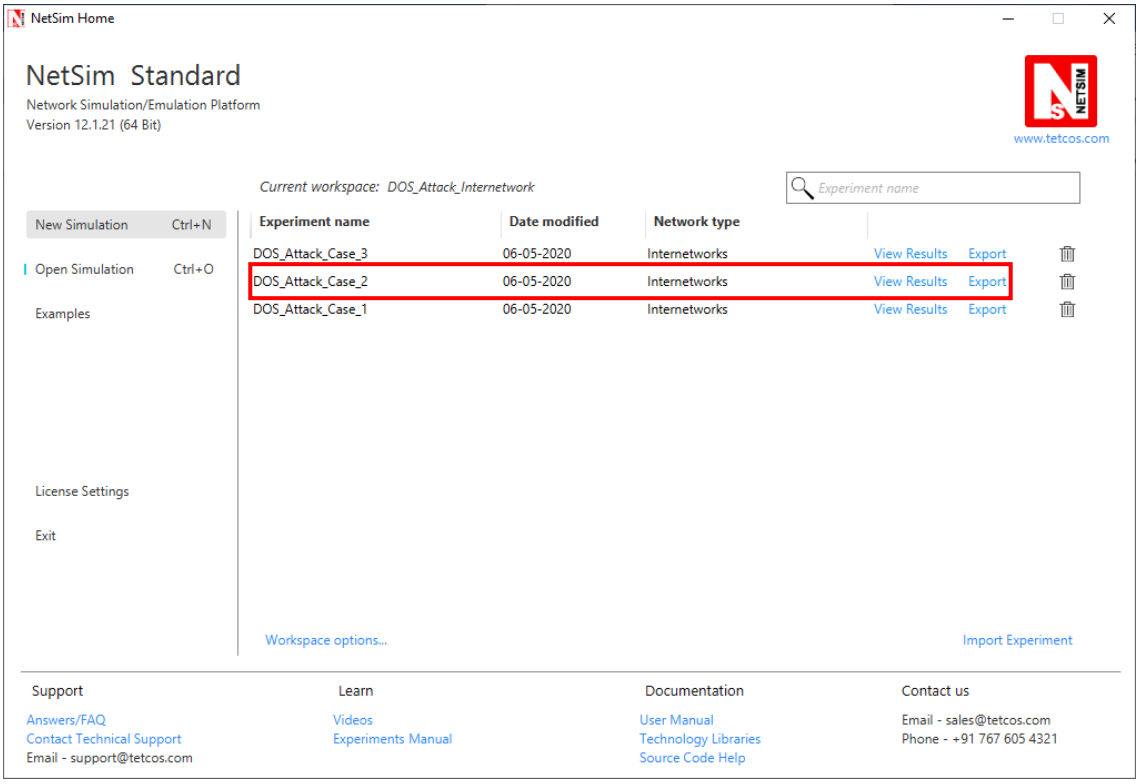


8. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries. (Note: first rebuild the TCP project and then rebuild the Ethernet project)

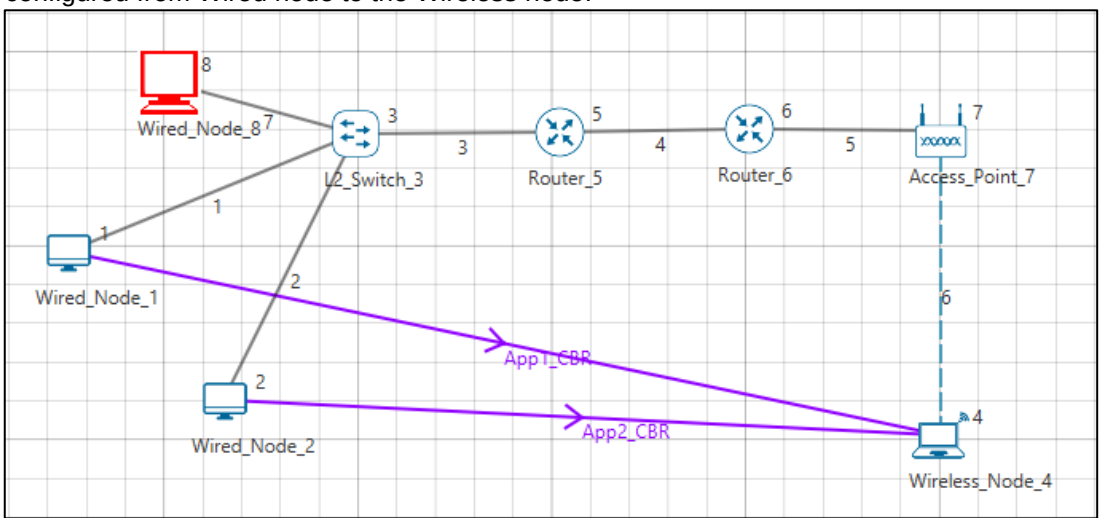
9. Run the simulation for 10 seconds.

Case-2: With one Malicious Node

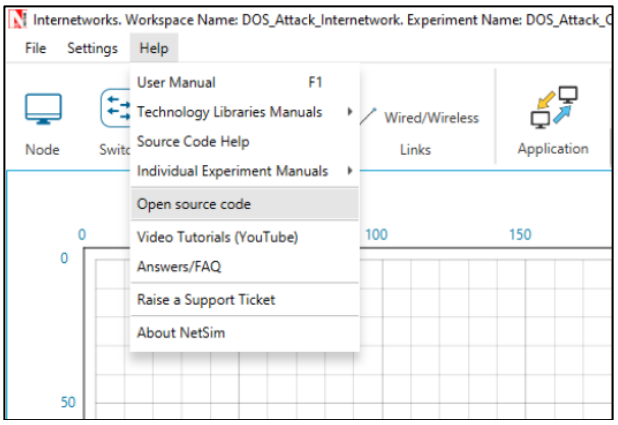
1. Then DOS_Attack_Internetworks comes with a sample configuration that is already saved. To open this example, go to Open Simulation and click on the DOS_Attack_Case_2 that is present under the list of experiments as shown below:



2. The saved network scenario consisting of 3 Wired Nodes, 1 L2 Switch, 2 router, 1 Access Point and 1 wireless node in the grid environment forming a internetworks Network. Traffic is configured from Wired node to the Wireless node.

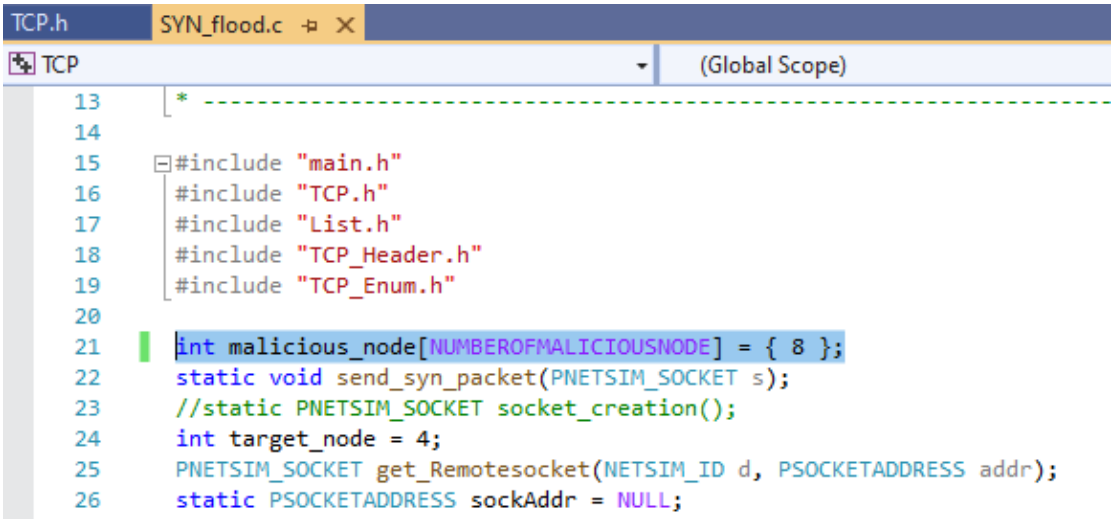


3. Help ➔ Open Source code

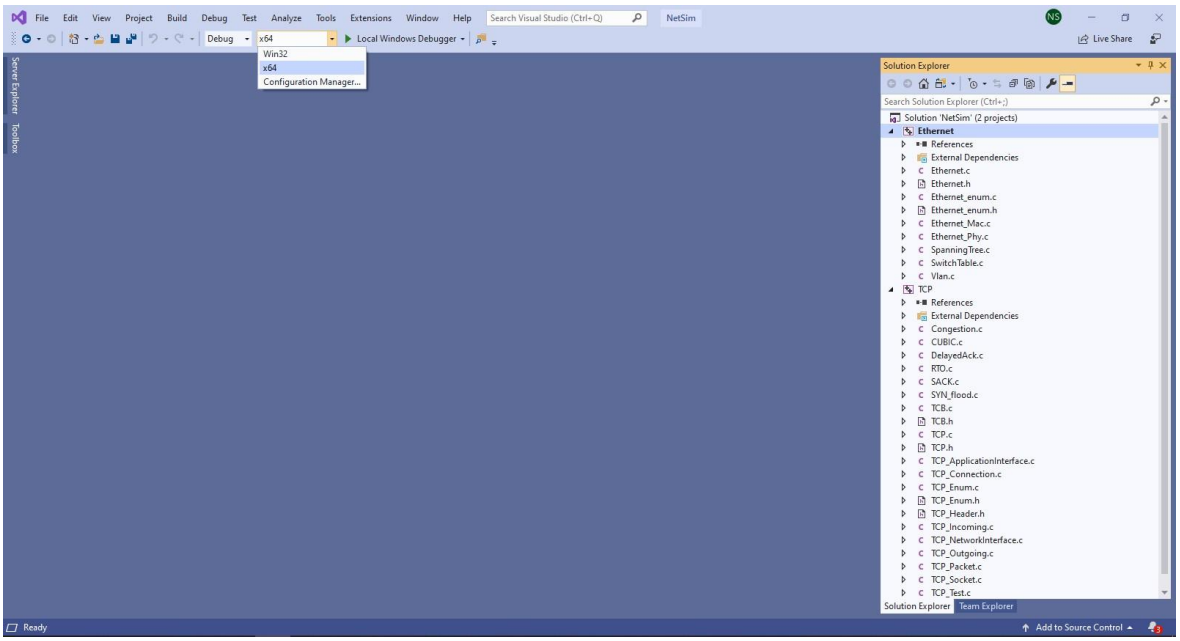


4. In TCP.h set **NUMBEROFMALICIOUSNODE** as 1.

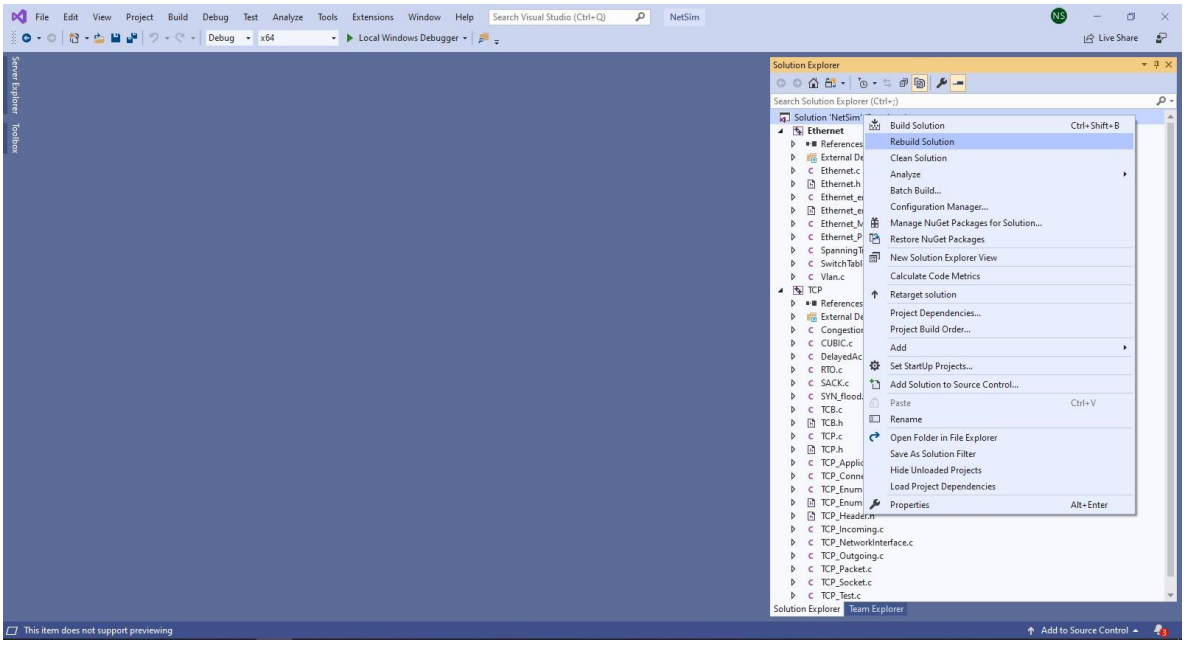
5. In SYN_FLOOD.c set **malicious node** as 8**.**



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.

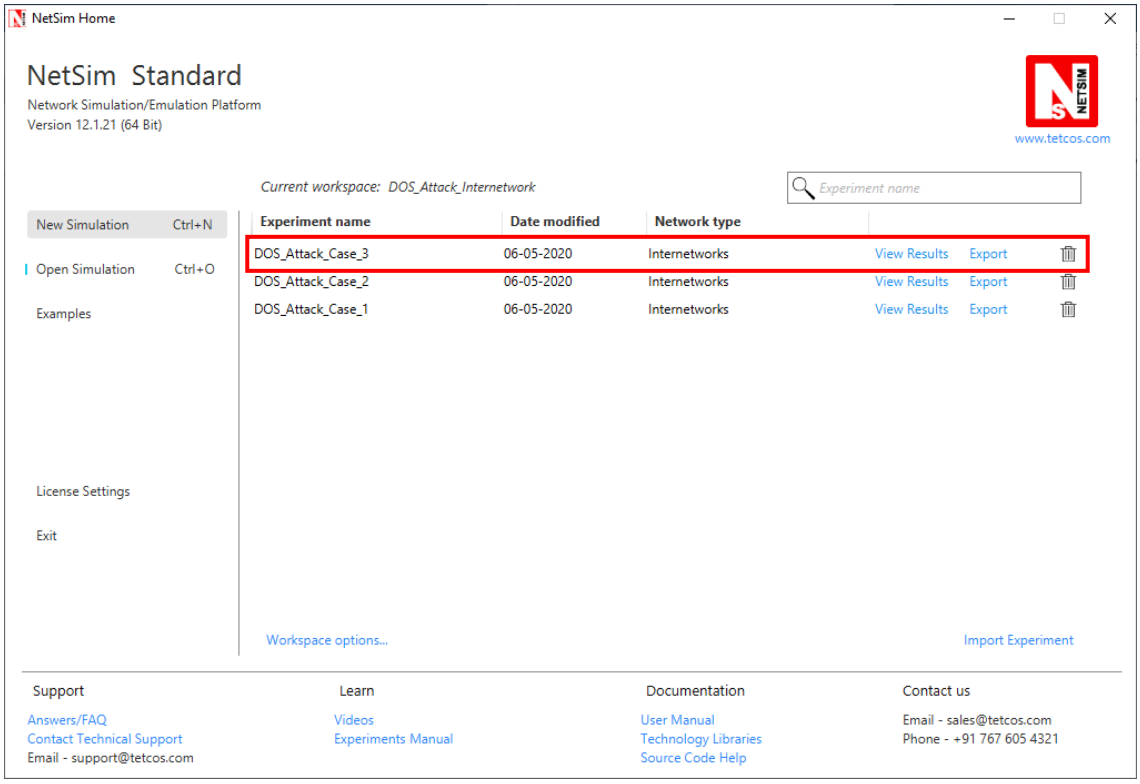


8. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries. (Note: first rebuild the TCP project and then rebuild the Ethernet project)

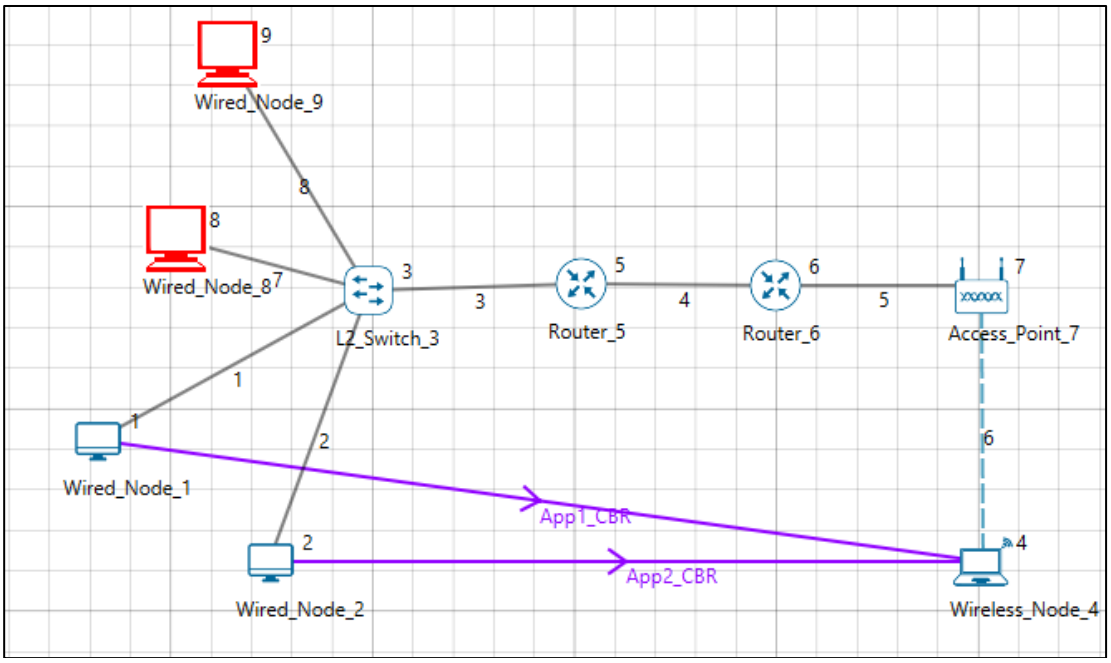
9. Run the simulation for 10 seconds.

Case-3: With two Malicious Node

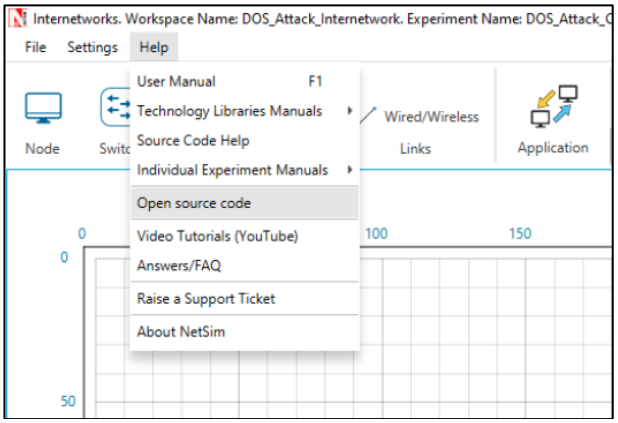
1. Then DOS_Attack_Internetworks comes with a sample configuration that is already saved. To open this example, go to Open Simulation and click on the DOS_Attack_Case_3 that is present under the list of experiments as shown below:



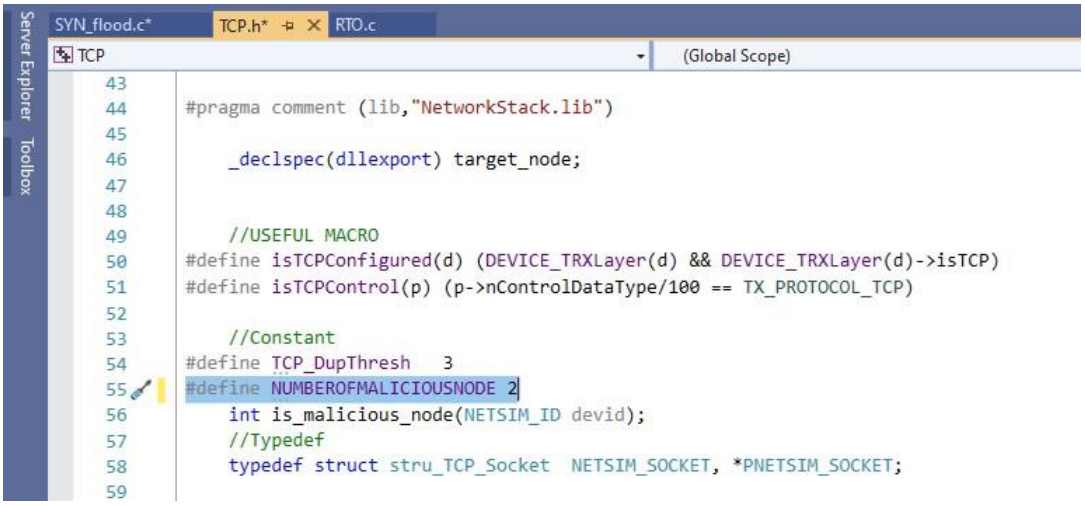
2. The saved network scenario consisting of 4 Wired Nodes, 1 L2 Switch, 2 router, 1 Access Point and 1 wireless node in the grid environment forming a internetworks Network. Traffic is configured from Wired node to the Wireless node.



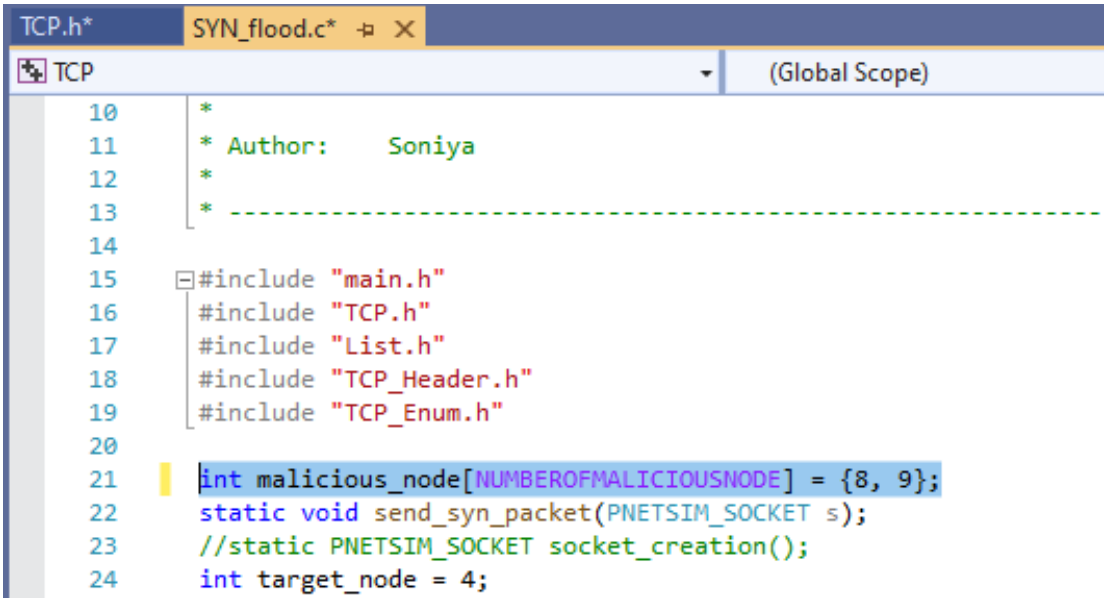
3. Help ➔ Open Source code



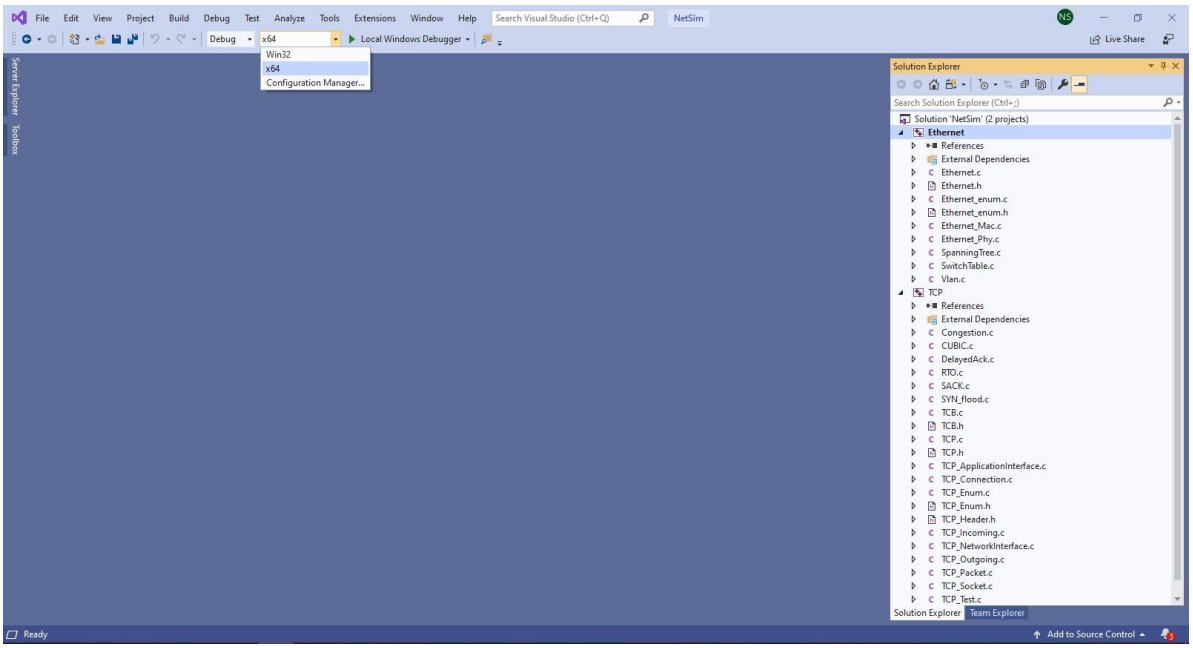
4. In TCP.h set **NUMBEROFMALICIOUSNODE** as 2.



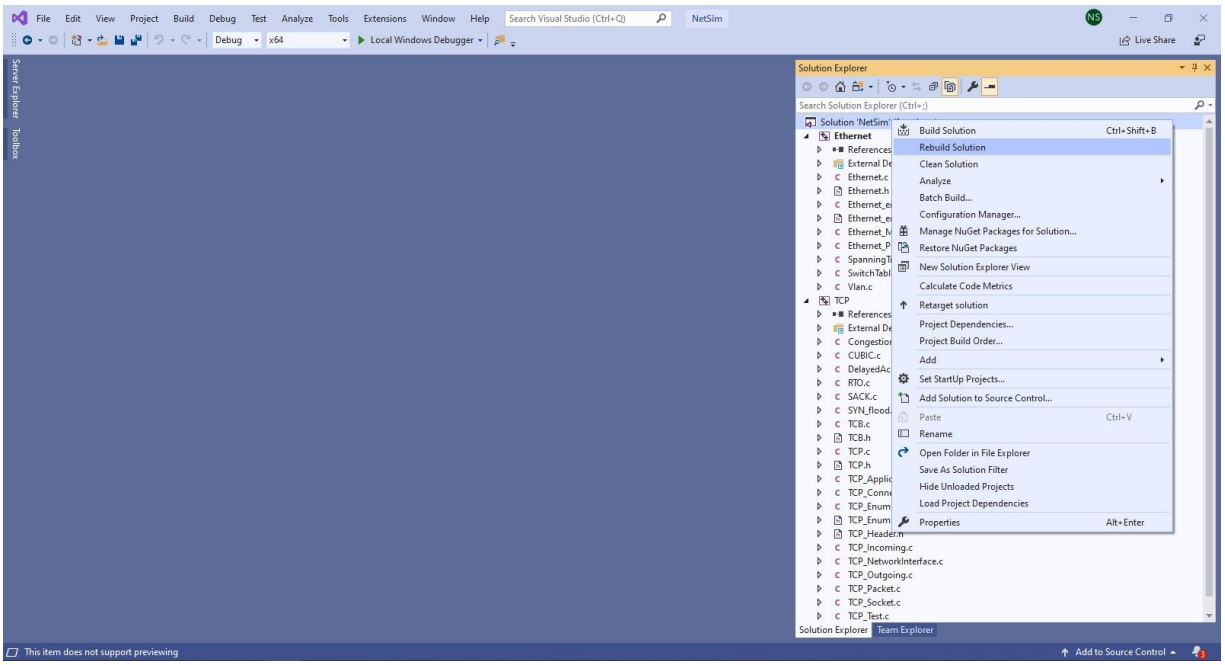
5. In SYN_FLOOD.c set **malicious node** as 8, 9**.**



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:



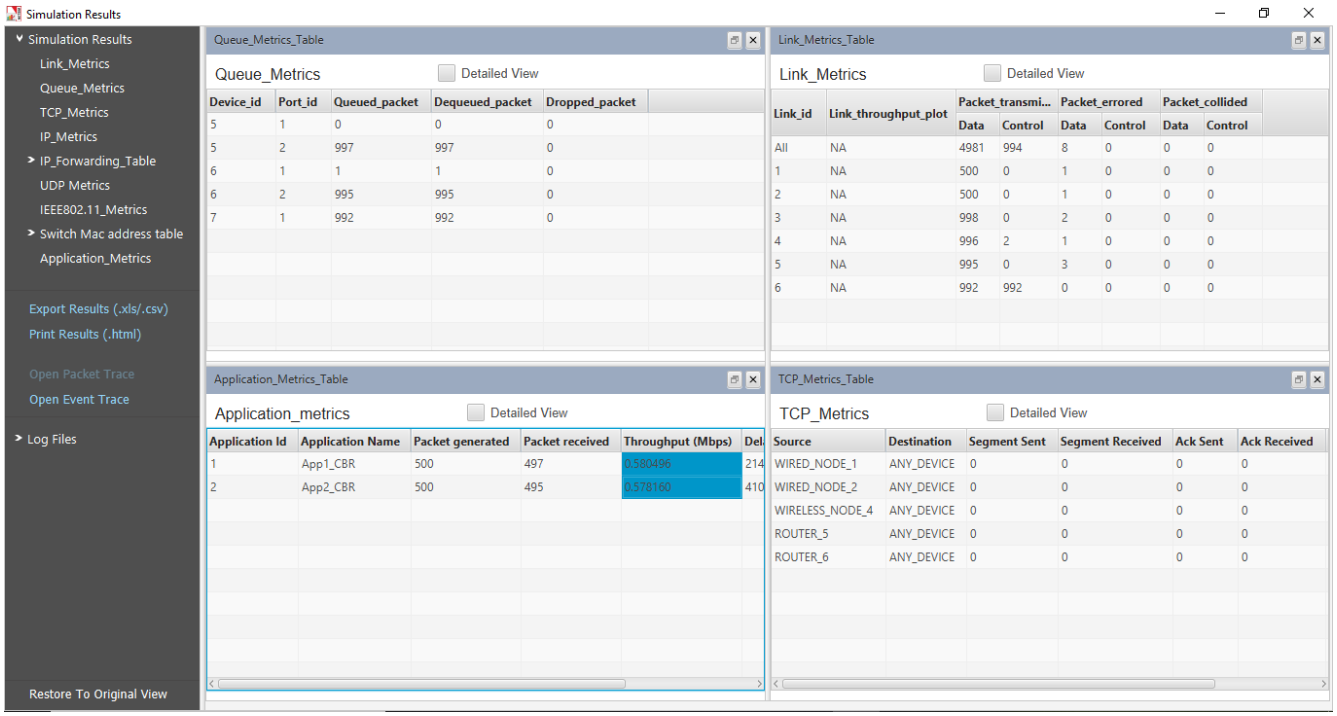
7. Right click on the solution in the solution explorer and select Rebuild.



8. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries. (Note: first rebuild the TCP project and then rebuild the Ethernet project)
9. Run the simulation for 10 seconds.

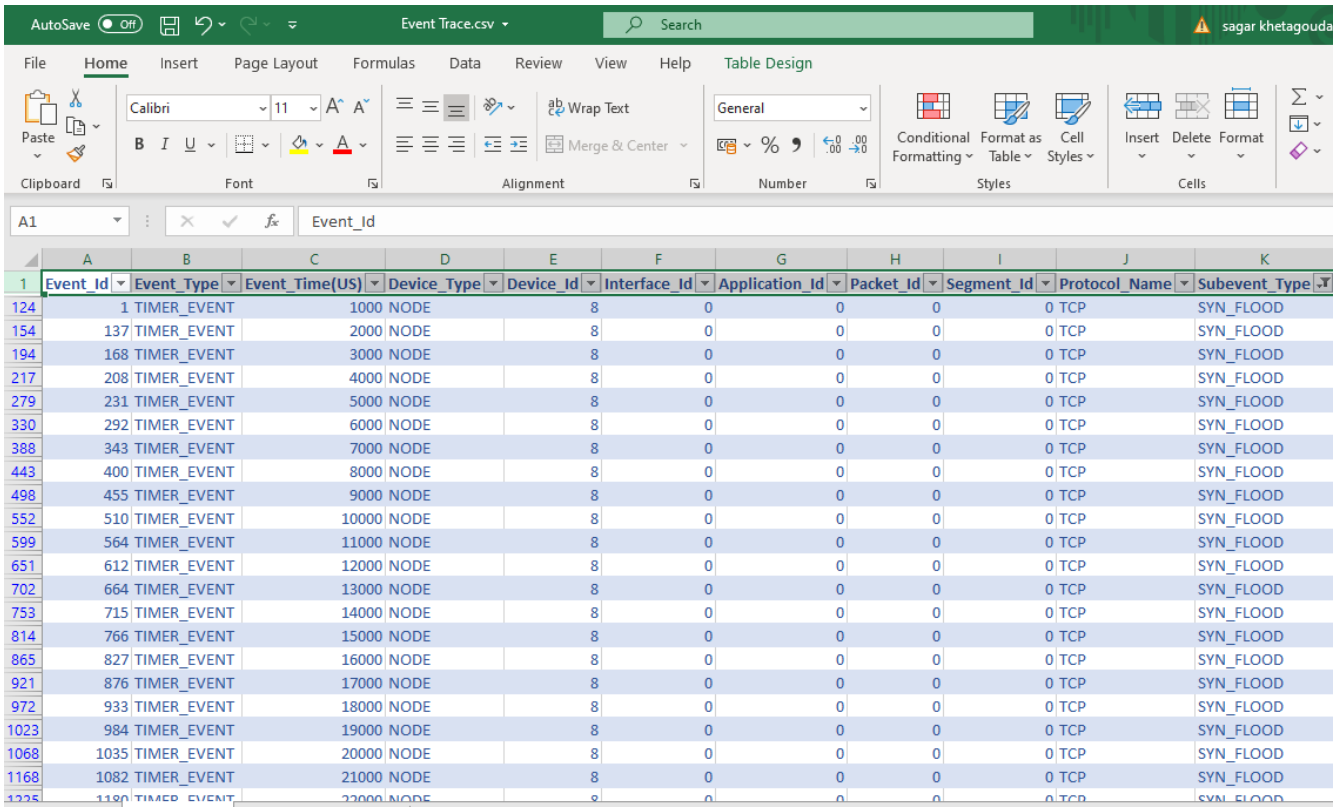
Result:

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 sends 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	0.580496	0.578160
Case-2: Malicious Node =1	0.520928	0.520928
Case-3: Malicious Node =2	0.287328	0.286160

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



Note: Users can also create their own network scenarios in Internetworksand run simulation.