

Dos Attack in 5G NR

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.

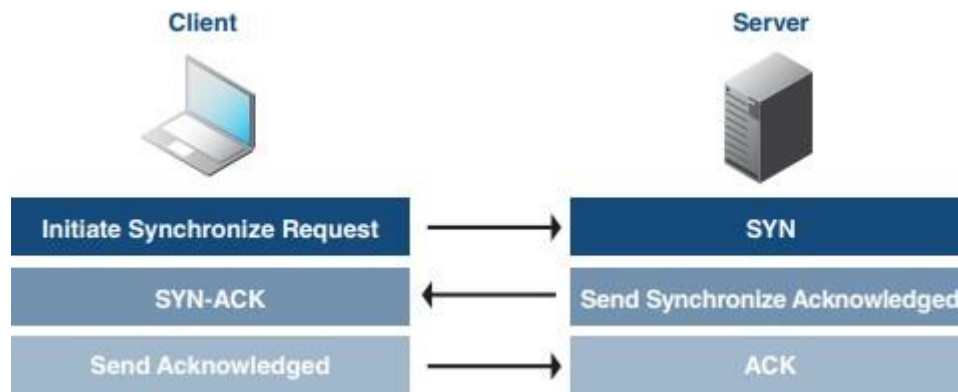
Standard DDoS Attack types:

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 microseconds. This will create and send the TCP_SYN packet for every 1000 microseconds. 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);`

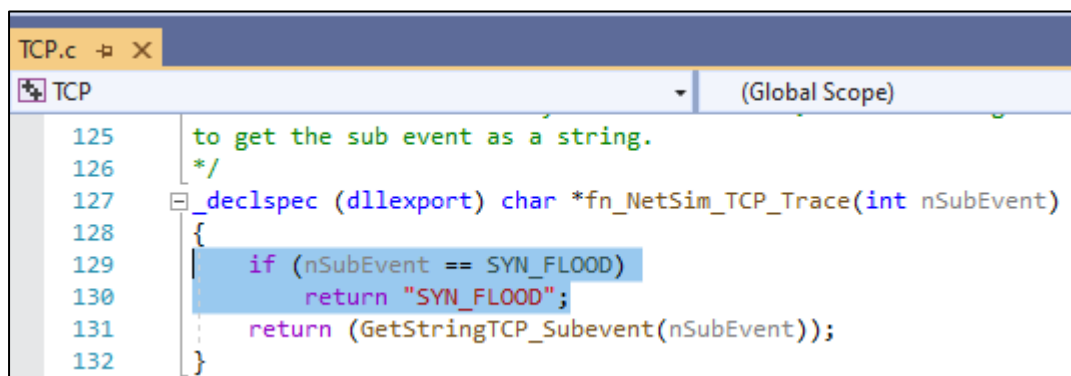
This function is used to create and send SYN packet to the network layer

- `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



```
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

```

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

```

79  /*
80  declspec (dllexport) int fn_NetSim_TCP_Init(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              fnAddEvent(&pevent);
106          }
107      }
108      return 0;
109  }
110
111  /**
112  This function is called by NetworkStack.dll, once simulation end to free the
113  allocated memory for the network.
114  */

```

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

```

52 *rto = min(max((*rto*2), G), (60 * SECOND));
53 print_tcp_log("New RTO = %.2lf", *rto);
54 }
55
56 void add_timeout_event(PNETSIM_SOCKET s,
57                       NetSim_PACKET* packet)
58 {
59     NetSim_PACKET* p = fn_NetSim_Packet_CopyPacket(packet);
60     add_packet_to_queue(&s->tcb->retransmissionQueue, p, pstruEventDetails->dEventTime);
61     NetSim_EVENTDETAILS pevent;
62     memcpy(&pevent, pstruEventDetails, sizeof pevent);
63     pevent.dEventTime += TCP_RTO(s->tcb);
64     pevent.dPacketSize = packet->pstruTransportData->dPacketSize;
65     pevent.nEventType = TIMER_EVENT;
66     pevent.nPacketId = packet->nPacketId;
67     if (packet->pstruAppData)
68     {
69         pevent.nApplicationId = packet->pstruAppData->nApplicationId;
70         pevent.nSegmentId = packet->pstruAppData->nSegmentId;
71     }
72     else
73     {
74         pevent.nSegmentId = 0;
75     }
76     if (!is_malicious_node(pevent.nDeviceId))
77     {
78         pevent.nProtocolId = TX_PROTOCOL_TCP;
79         pevent.pPacket = fn_NetSim_Packet_CopyPacket(p);
80         pevent.szOtherDetails = NULL;
81         pevent.nSubEventType = TCP_RTO_TIMEOUT;
82         fnAddEvent(&pevent);
83         print_tcp_log("Adding RTO Timer at %.1lf", pevent.dEventTime);
84     }
85 }
86
87 static void handle_rto_timer_for_ctrl(PNETSIM_SOCKET s)
88 {
89     if (isSynbitSet(pstruEventDetails->pPacket))
90     {
91         record_syn(-);
92     }
93 }

```

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

```

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 NUMBEROFMALIGNANTNODE 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

```

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, PPOCKETADDRESS addr);
26  static PPOCKETADDRESS 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_IPADDRESS(target_node, 1);
43       * }
44       */
45
46      PNETSIM_SOCKET s = get_Remotesocket(malicious_node, sockAddr);
47  }

```

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

```

TCP_Enum.h  TCP.h  RTO.c  SYN_flood.c  TCP_Connection.c  TCP.c  T
TCP
#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)

```

- SYN_FLOOD.c file contains the following functions

```

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, PPOCKETADDRESS addr);
26  static PPOCKETADDRESS 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  }

```

```

34 }
35
36 void syn_flood()
37 {
38     extern PSOCKETADDRESS anySocketAddr;
39     anySocketAddr->ip = DEVICE_NWADDRESS(target_node, 1);
40     PNETSIM_SOCKET s = get_remotesocket(pstruEventDetails->nDeviceId, anySocketAddr);
41     ptrSOCKETINTERFACE sId = (ptrSOCKETINTERFACE)pstruEventDetails->szOtherDetails;
42     NetSim_EVENTDETAILS pEvent;
43     if (!s)
44     {
45         s = socket_creation();
46         tcp_connect(s, s->localAddr, s->remoteAddr);
47     }
48     else
49     {
50         s->localDeviceId = pstruEventDetails->nDeviceId;
51         s->remoteDeviceId = target_node;
52         s->sId = sId;
53         send_syn_packet(s);
54         memcpy(&pEvent, pstruEventDetails, sizeof pEvent);
55         pEvent.dEventTime = pstruEventDetails->dEventTime + 1000;
56         pEvent.nDeviceId = pstruEventDetails->nDeviceId;
57         pEvent.nPacketId = 0;
58         pEvent.nEventType = TIMER_EVENT;
59         pEvent.nProtocolId = TX_PROTOCOL_TCP;
60         pEvent.nSubEventType = SYN_FLOOD;
61         fnpAddEvent(&pEvent);
62     }
63 }
64
65
66
67
68

```

```

67 }
68
69 static void send_syn_packet(PNETSIM_SOCKET s)
70 {
71     NetSim_PACKET* syn = create_syn(s, pstruEventDetails->dEventTime);
72     s->tcbs->SND.UNA = s->tcbs->ISS;
73     s->tcbs->SND.NXT = s->tcbs->ISS + 1;
74     tcp_change_state(s, TCPCONNECTION_SYN_SENT);
75     s->tcbs->synRetries++;
76     s->tcpMetrics->synSent++;
77     send_to_network(syn, s);
78     add_timeout_event(s, syn);
79 }
80
81
82
83
84

```

```

85 int socket_creation()
86 {
87     static int s_id = 100;
88     ptrSOCKETINTERFACE sId = (ptrSOCKETINTERFACE)pstruEventDetails->szOtherDetails;
89     PNETSIM_SOCKET newSocket = tcp_create_socket();
90     add_to_socket_list(pstruEventDetails->nDeviceId, newSocket);
91     PSOCKETADDRESS localsocketAddr = (PSOCKETADDRESS)calloc(1, sizeof * localsocketAddr);
92     localsocketAddr->ip = DEVICE_NWADDRESS(pstruEventDetails->nDeviceId, 1);
93     localsocketAddr->port = 0;
94     PSOCKETADDRESS remotesocketAddr = (PSOCKETADDRESS)calloc(1, sizeof * remotesocketAddr);
95     remotesocketAddr->ip = DEVICE_NWADDRESS(target_node, 1);
96     remotesocketAddr->port = 0;
97     newSocket->SocketId = s_id;
98     s_id++;
99     newSocket->localAddr = localsocketAddr;
100     newSocket->remoteAddr = remotesocketAddr;
101     newSocket->localDeviceId = pstruEventDetails->nDeviceId;
102     newSocket->remoteDeviceId = target_node;
103     newSocket->sId = sId;
104     return newSocket;
105 }
106
107
108
109
110
111
112
113
114

```

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()

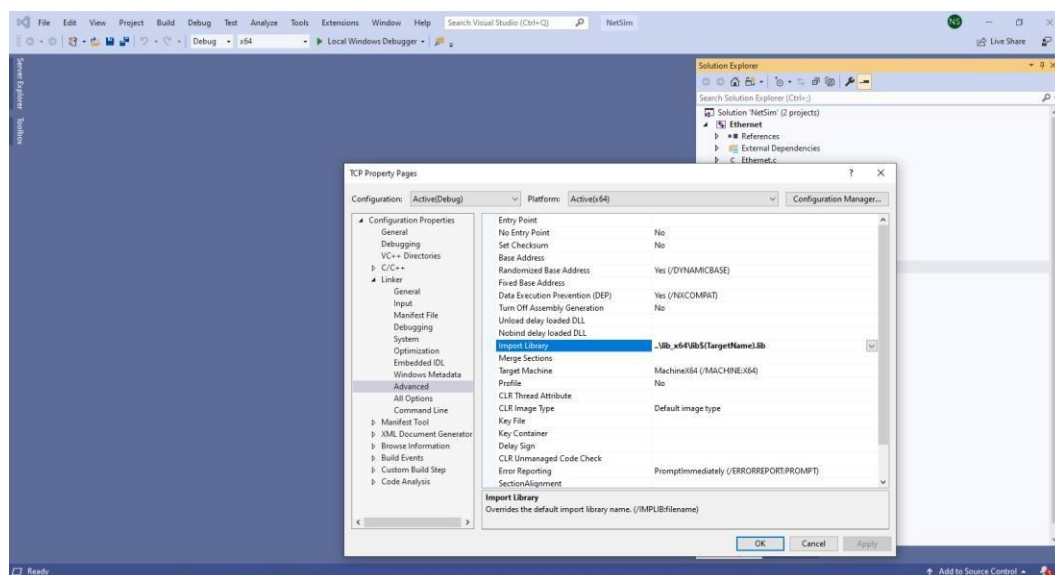
if (!packet)
    return 2; // No packet is there for transmission

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;
}
}

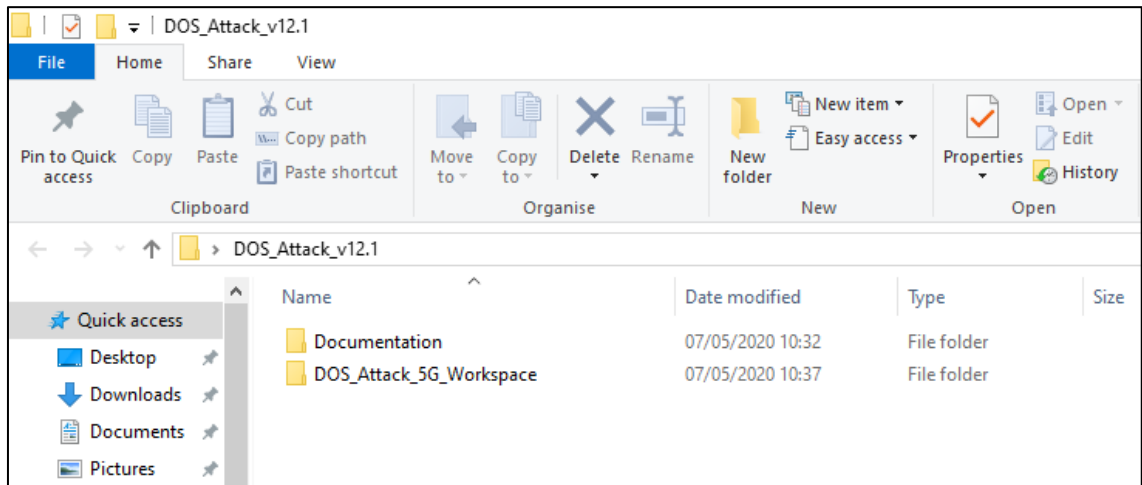
```

11. Right click on TCP project->Properties->Linker->Advanced->import library 32-bit and 64-bit ..\lib\lib\$(TargetName).lib or ..\lib_x64\lib\$(TargetName).lib

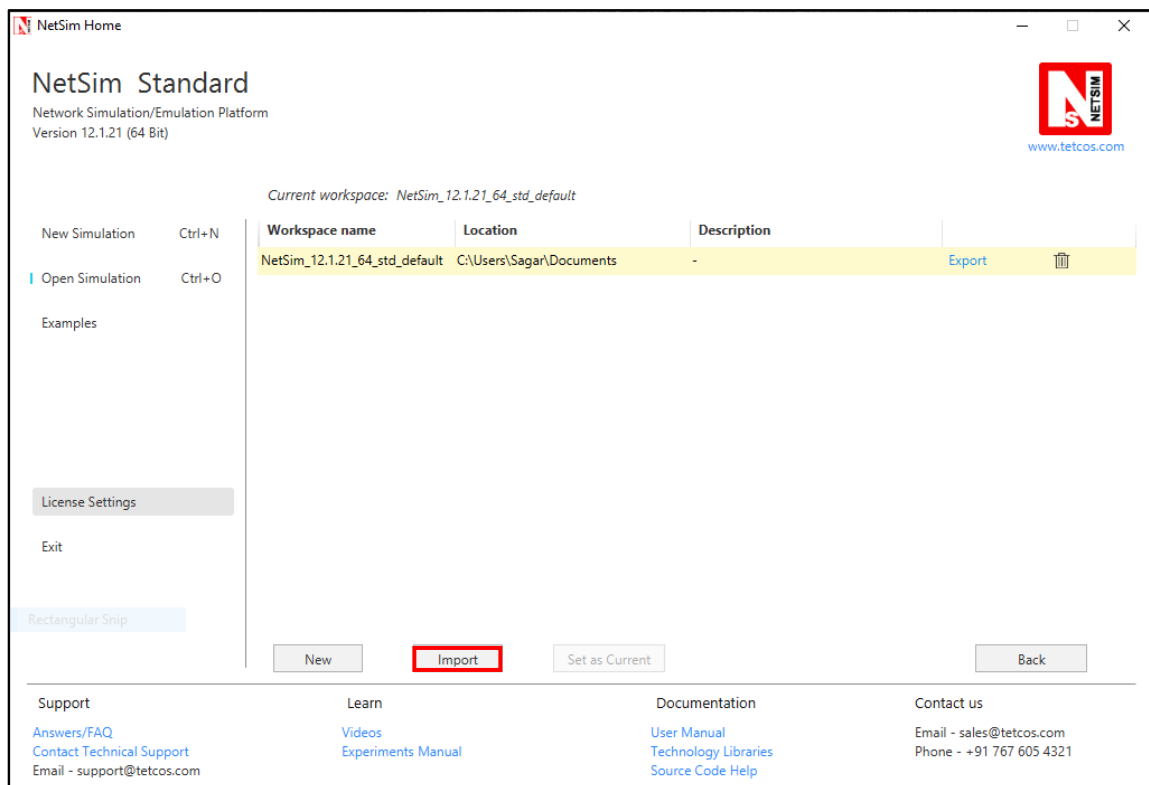


Steps:

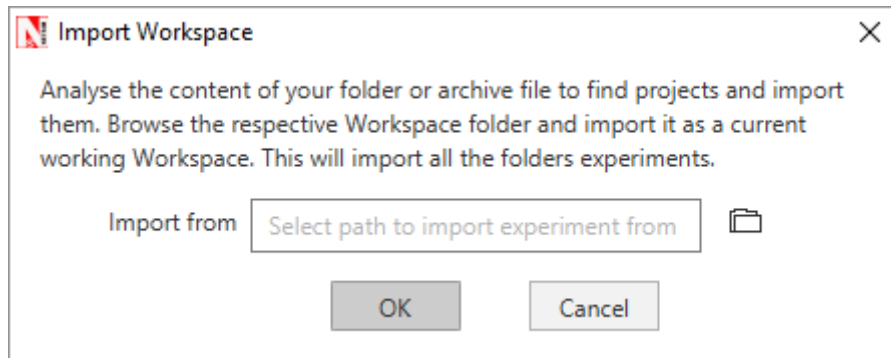
1. The downloaded project folder contains the folders Documentation, and DOS_Attack_5G_Workspace directory as shown below:



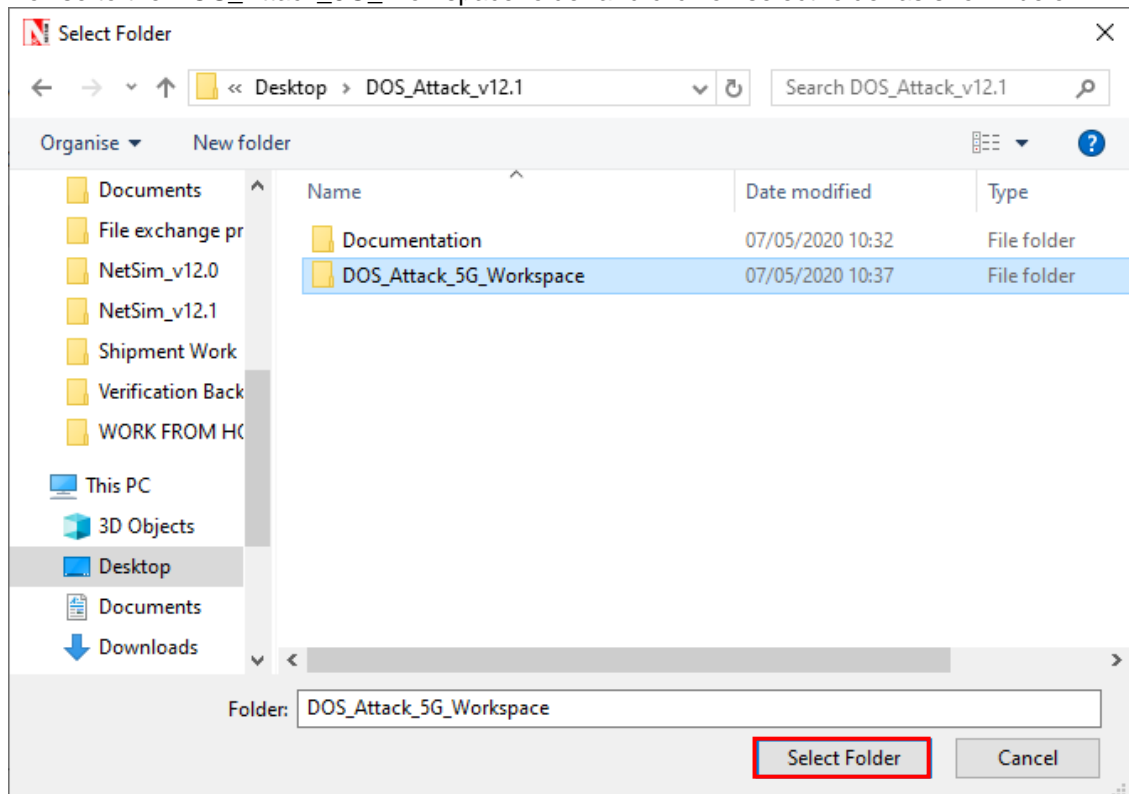
2. Import DOS_Attack_5G_Workspace by going to Open Simulation->Workspace Options->More Options in NetSim Home window. Then select Import as shown below:



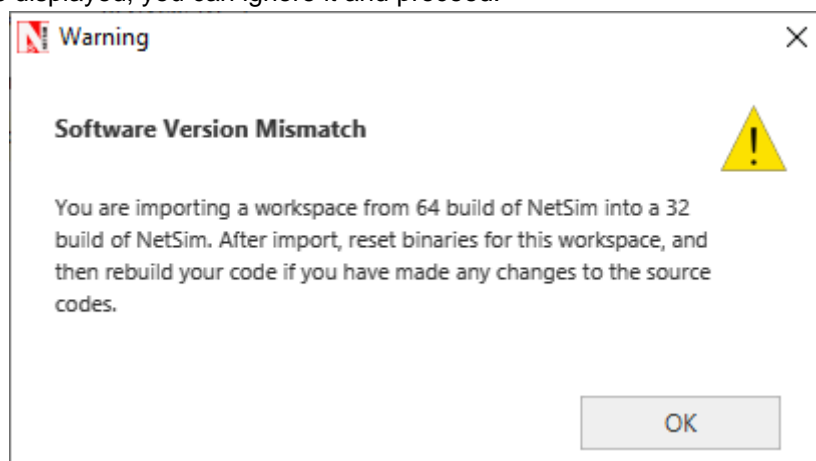
3. It displays a window where users need to give the path of the workspace folder and click on OK as shown below:



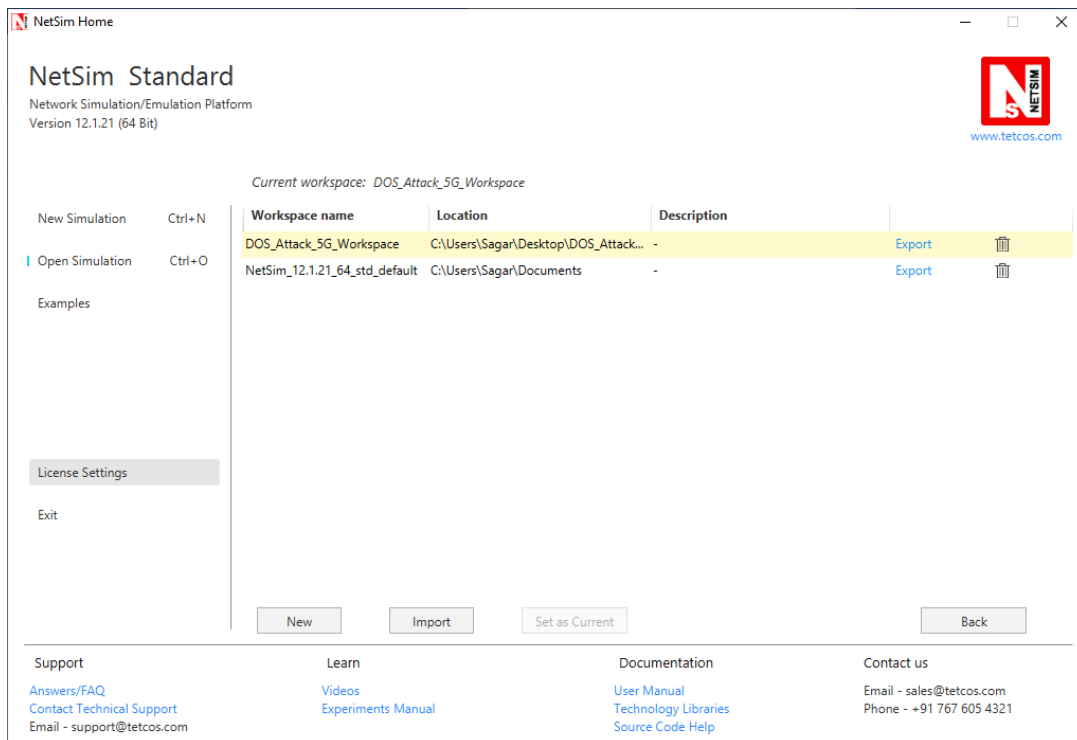
4. Browse to the DOS_Attack_5G_Workspace folder and click on select folder as shown below:



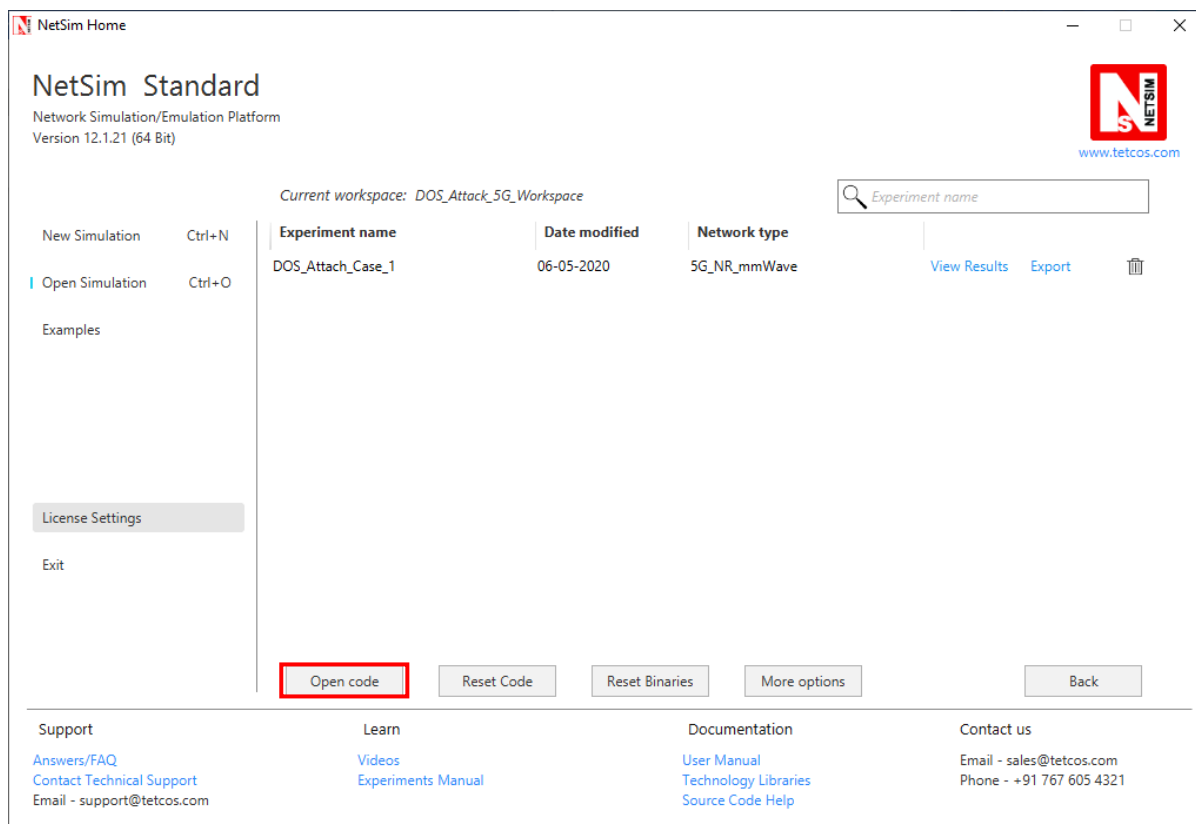
5. After this click on OK button in the Import Workspace window.
6. While importing the workspace, if the following warning message indicating Software Version Mismatch is displayed, you can ignore it and proceed.



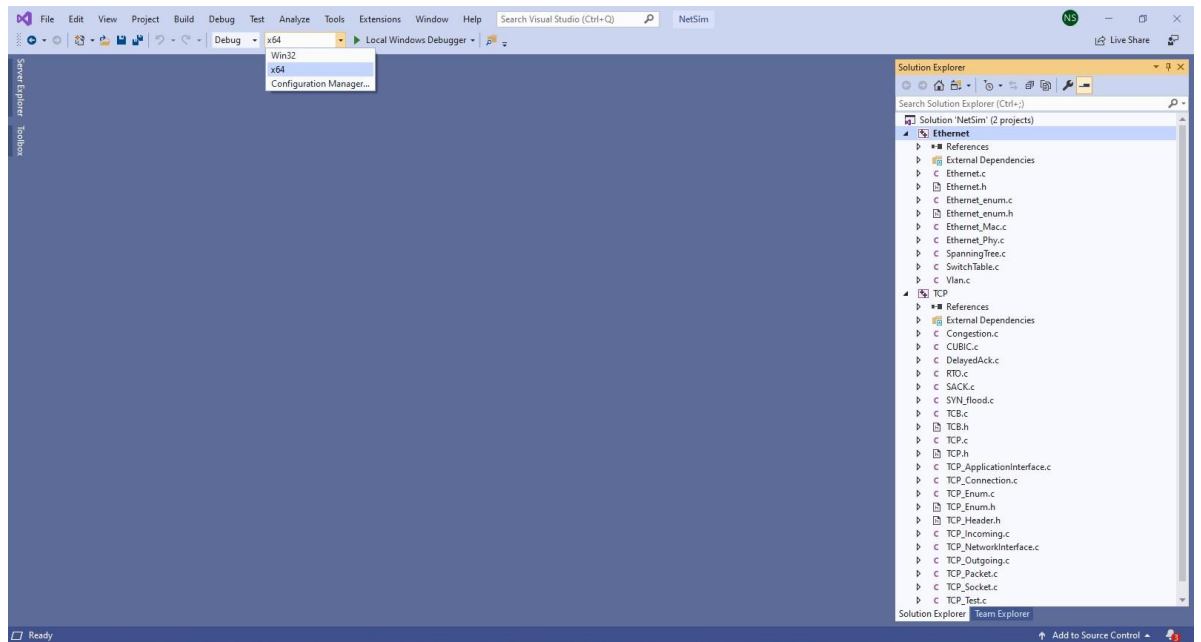
7. The Imported workspace will be set as the current workspace automatically. To see the imported workspace, click on Open Simulation->Workspace Options->More Options as shown below:



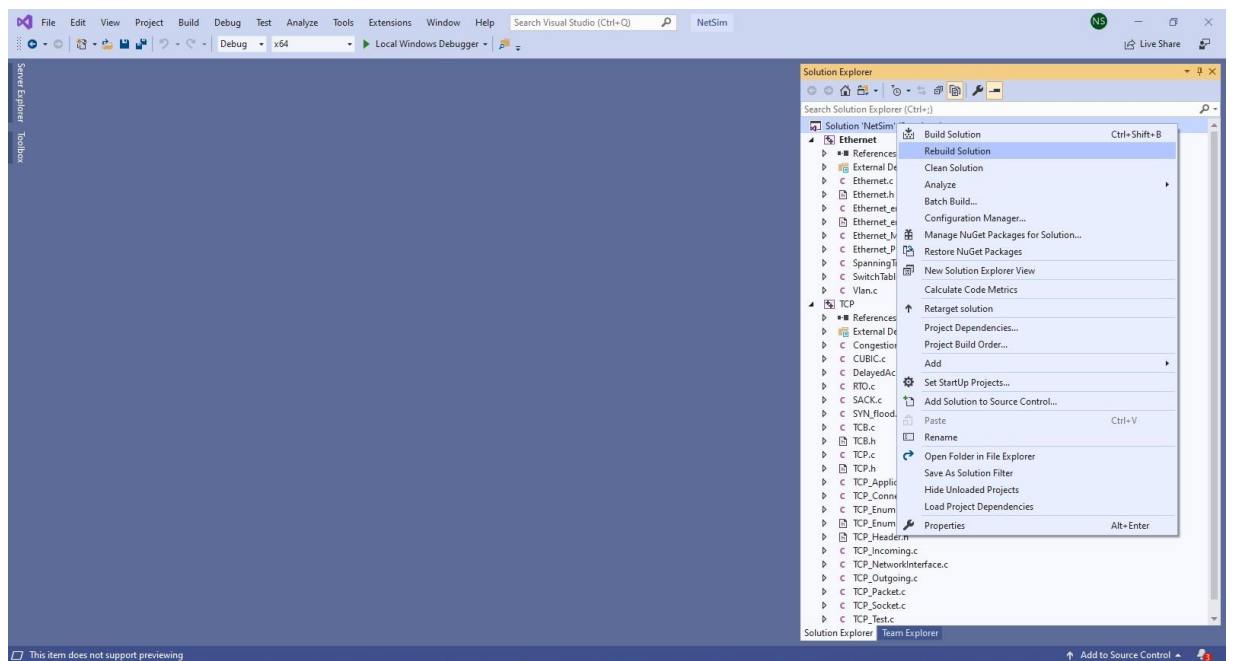
- Open the Source codes in Visual Studio by going to Open Simulation-> Workspace Options and Clicking on Open code button as shown below:



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

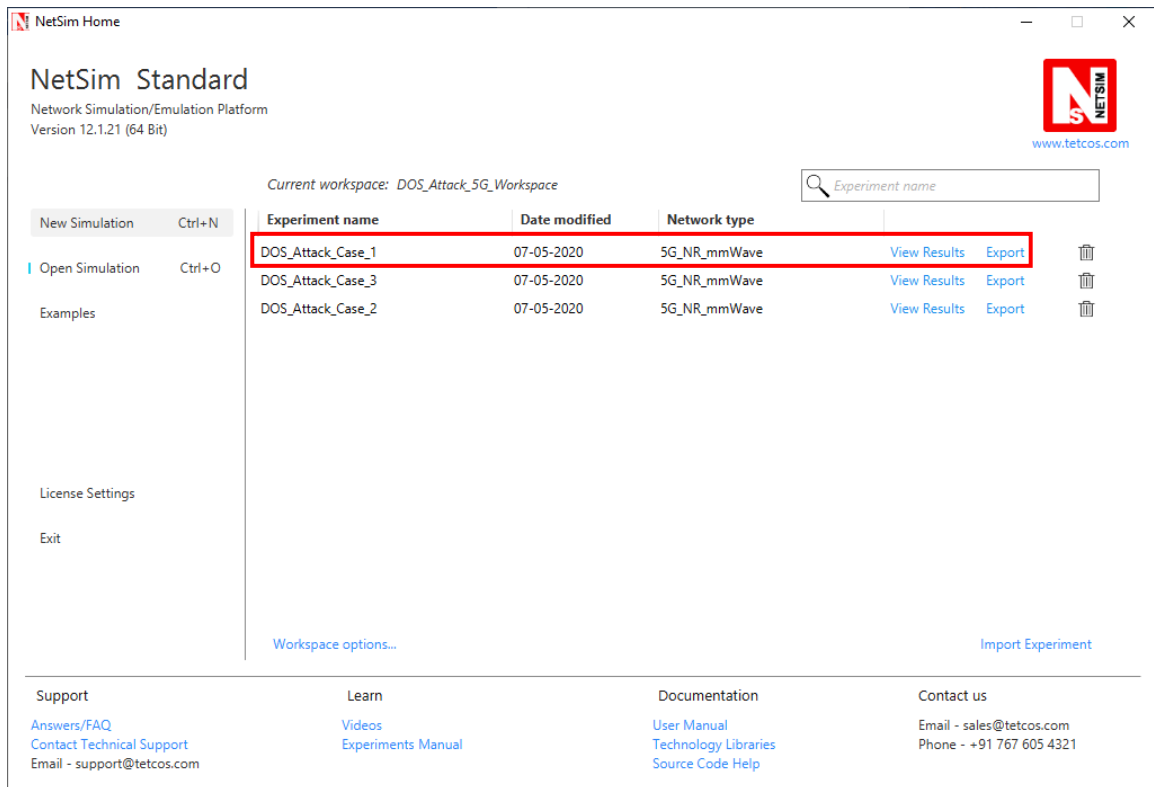


12. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries. (Note: first rebuild the TCP project and then rebuild the Ethernet project)

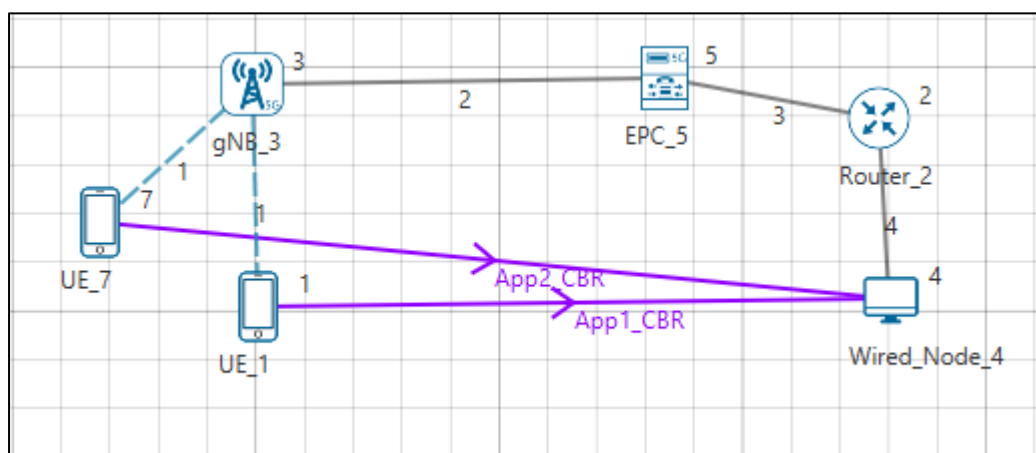
13. Run NetSim as Administrative mode.

Case-1: Without Malicious Node

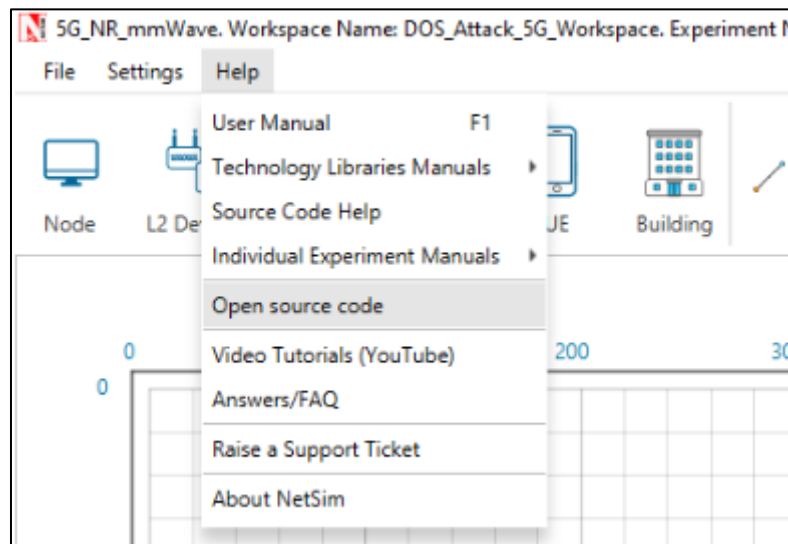
1. Then DOS_Attack_5G_Workspace comes with a sample configuration that is already saved. To open this example, go to Open Simulation and click on the DOS_Attack_Case_1 that is present under the list of experiments as shown below:



2. The saved network scenario consisting of 2 UEs, 1 gNB, 2 router, 1 EPC, 1 Router and 1 wired node in the grid environment forming a 5G NR mmWavw Network. Traffic is configured from Wired node to the Wireless node.



3. Help  Open Source code



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

```

SYN_flood.c* TCP.h RTO.c
TCP (Global Scope)
43
44 #pragma comment (lib,"NetworkStack.lib")
45
46 _declspec(dllexport) target_node;
47
48
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 1
56 int is_malicious_node(NETSIM_ID devid);

```

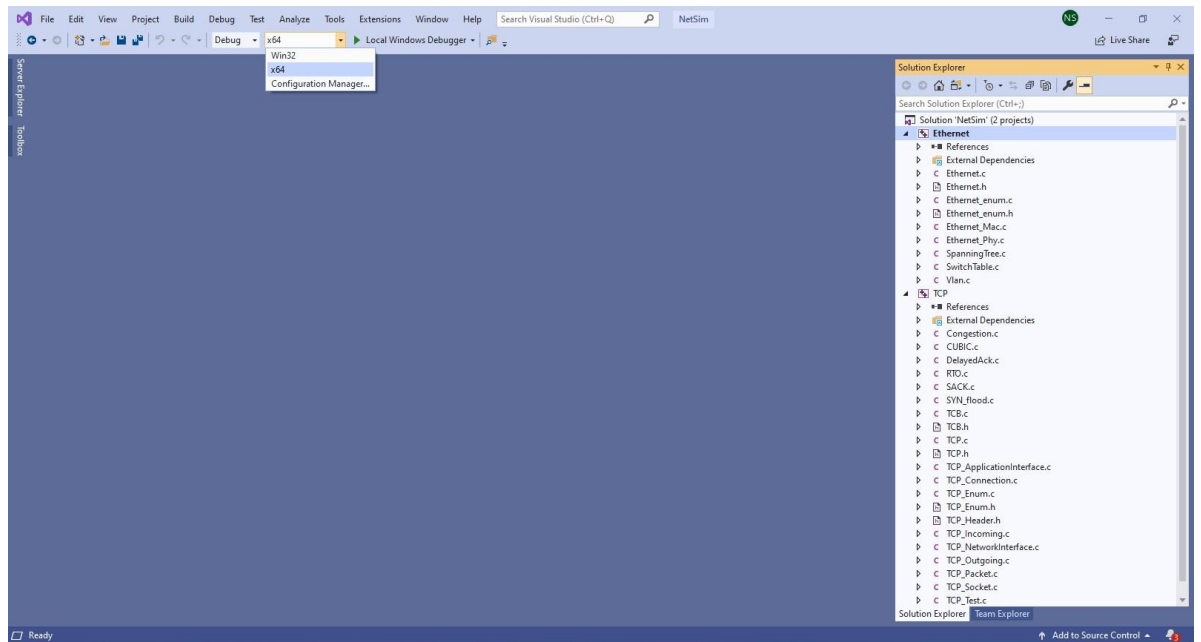
5. In SYN_FLOOD.c set **malicious node** as 0.

```

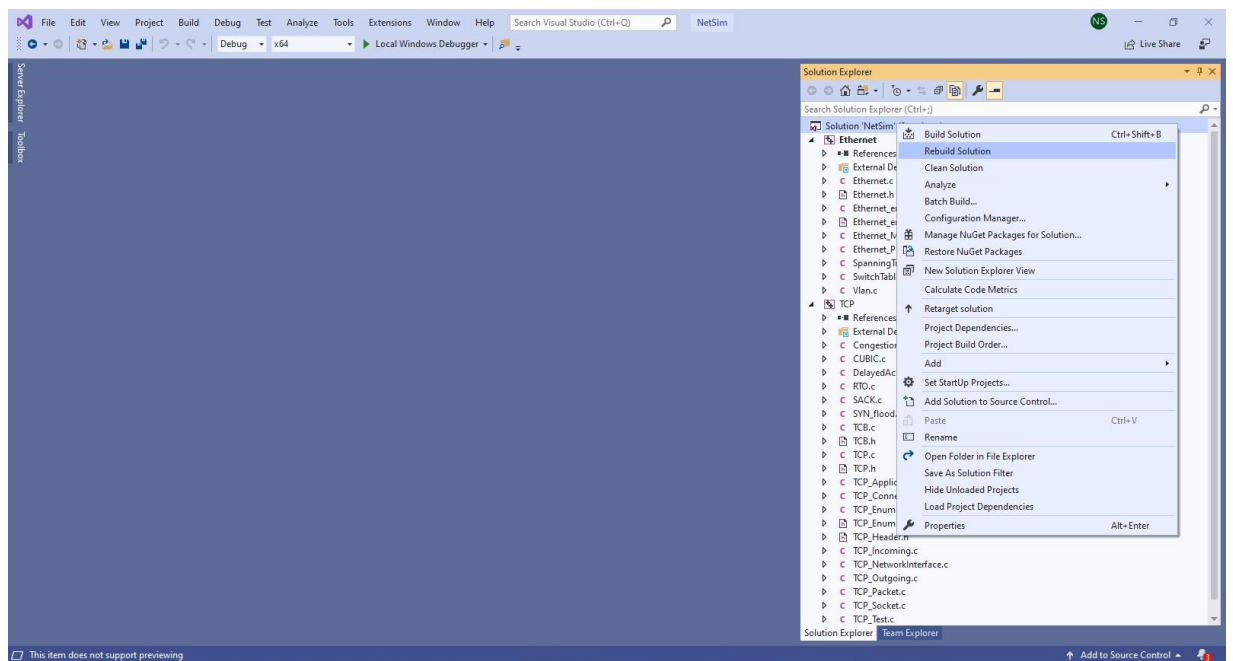
SYN_flood.c* TCP.h RTO.c
TCP (Global Scope)
7
8 * Product or its content without express prior written consent of Tetcos is
9 * prohibited. Ownership and / or any other right relating to the software and all
10 * intellectual property rights therein shall remain at all times with Tetcos.
11 *
12 * Author: Soniya
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] = { 0 };
22 static void send_syn_packet(PNETSIM_SOCKET s);
23 //static PNETSIM_SOCKET socket_creation();

```

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:



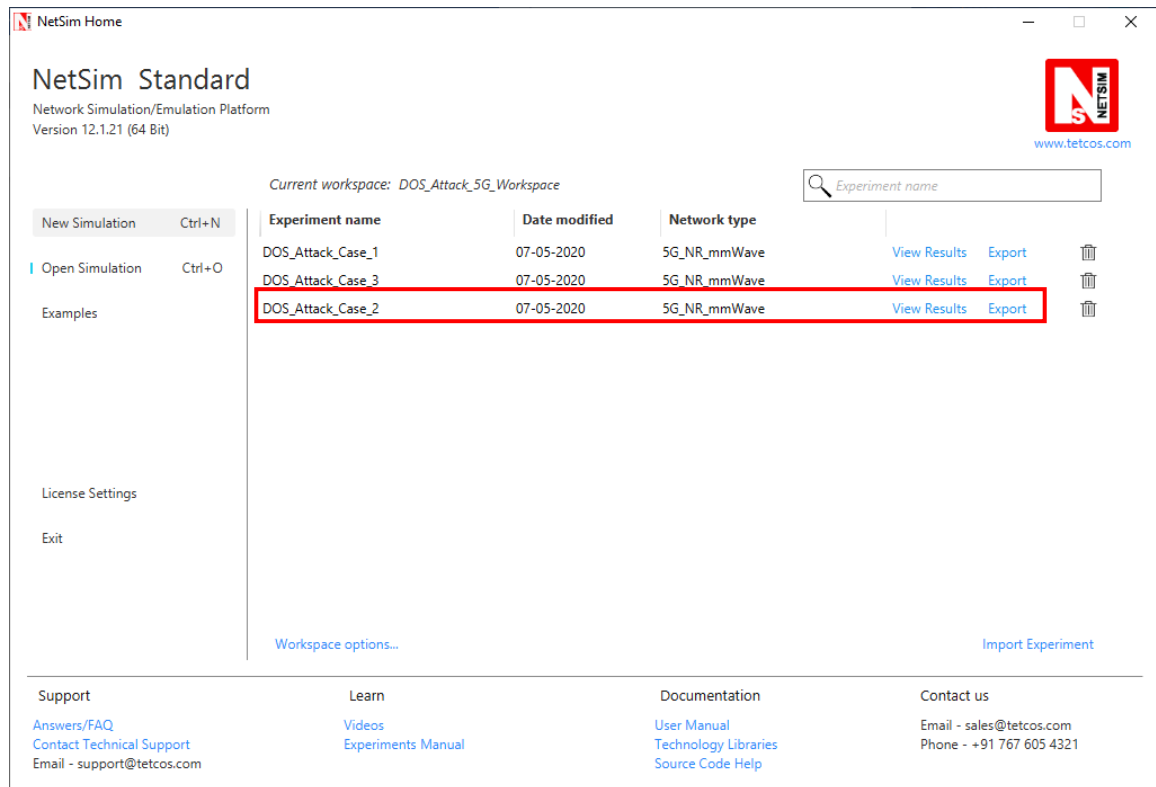
7. Right click on the solution in the solution explorer and select Rebuild.



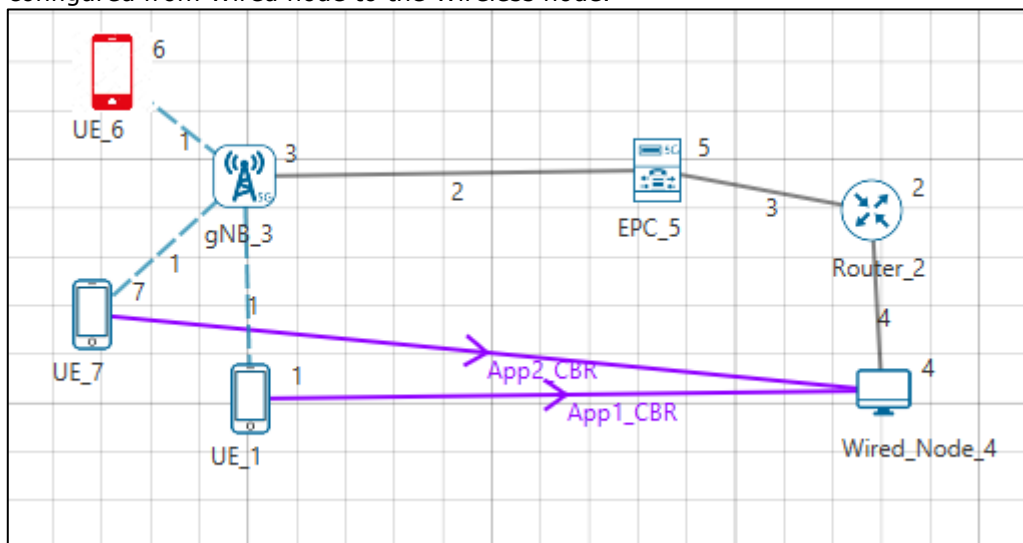
8. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries. (Note: first rebuild the TCP project and then rebuild the Ethernet project)
9. Run the simulation for 5 seconds.

Case-2: With one Malicious Node

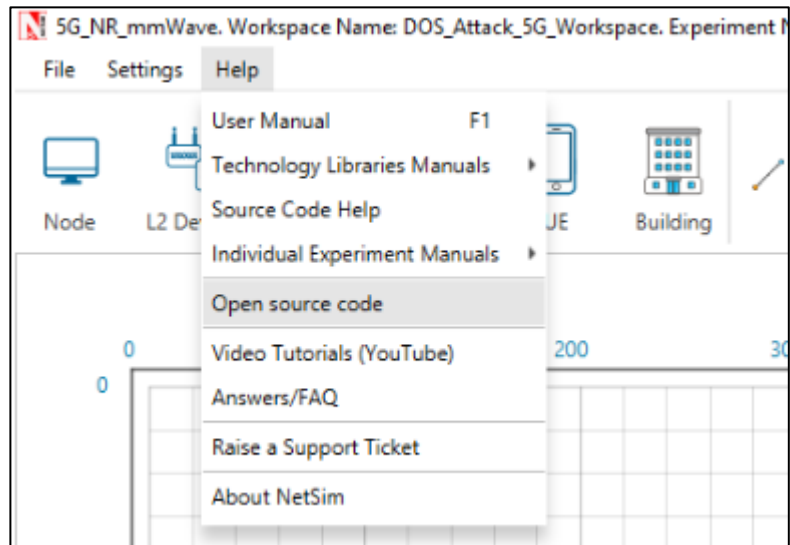
1. Then DOS_Attack_5G_Workspace comes with a sample configuration that is already saved. To open this example, go to Open Simulation and click on the DOS_Attack_Case_2 that is present under the list of experiments as shown below:



2. The saved network scenario consisting of 3 UEs, 1 gNB, 2 router, 1 EPC, 1 Router and 1 wired node in the grid environment forming a 5G NR mmWave Network. Traffic is configured from Wired node to the Wireless node.



3. Help  Open Source code



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

```

SYN_flood.c*  TCP.h  RTO.c
TCP (Global Scope)
43
44 #pragma comment (lib,"NetworkStack.lib")
45
46 _declspec(dllexport) target_node;
47
48
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 1
56 int is_malicious_node(NETSIM_ID devid);

```

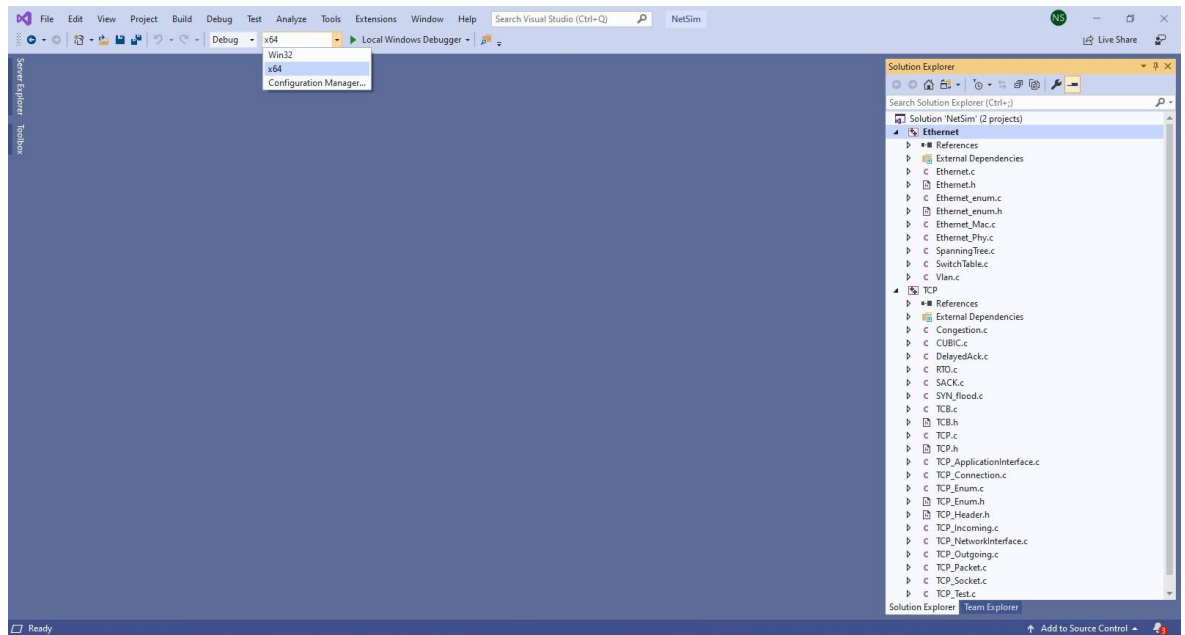
5. In SYN_FLOOD.c set **malicious node** as 6.

```

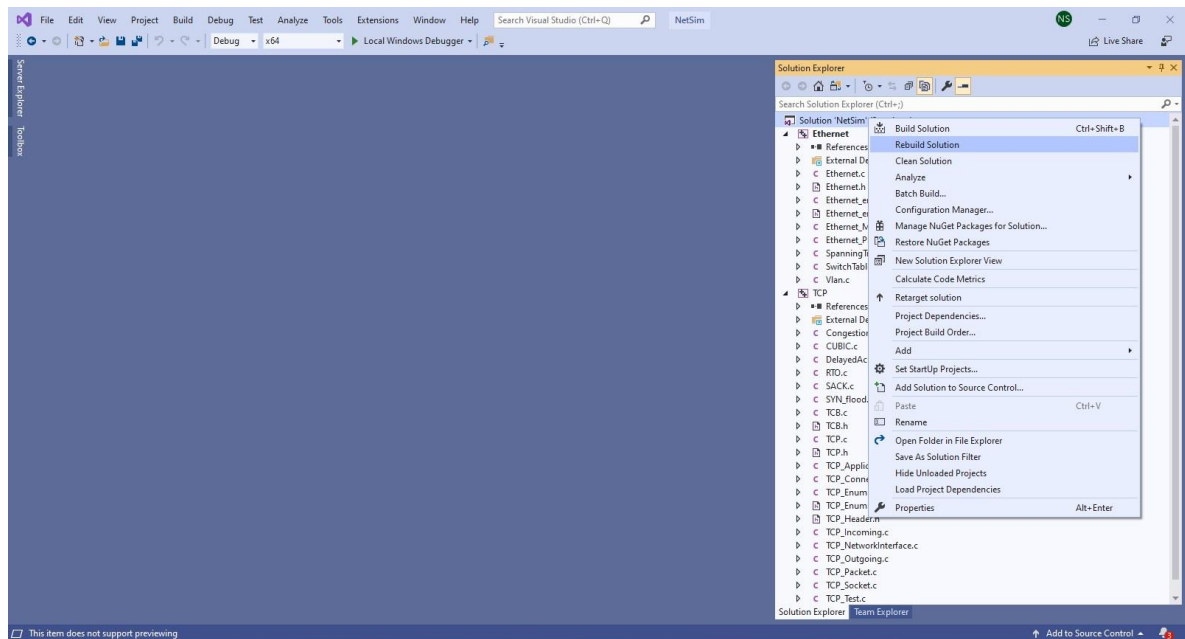
SYN_flood.c  TCP.h
TCP (Global Scope)
12
13
14
15 #include "main.h"
16 #include "TCP.h"
17 #include "List.h"
18 #include "TCP_Header.h"
19 #include "TCP_Enum.h"
20
21 int malicious_node[NUMBEROFMALICIOUSNODE] = {6};
22 static void send_syn_packet(PNETSIM_SOCKET s);
23 //static PNETSIM_SOCKET socket_creation();
24 int target_node = 4;

```

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:



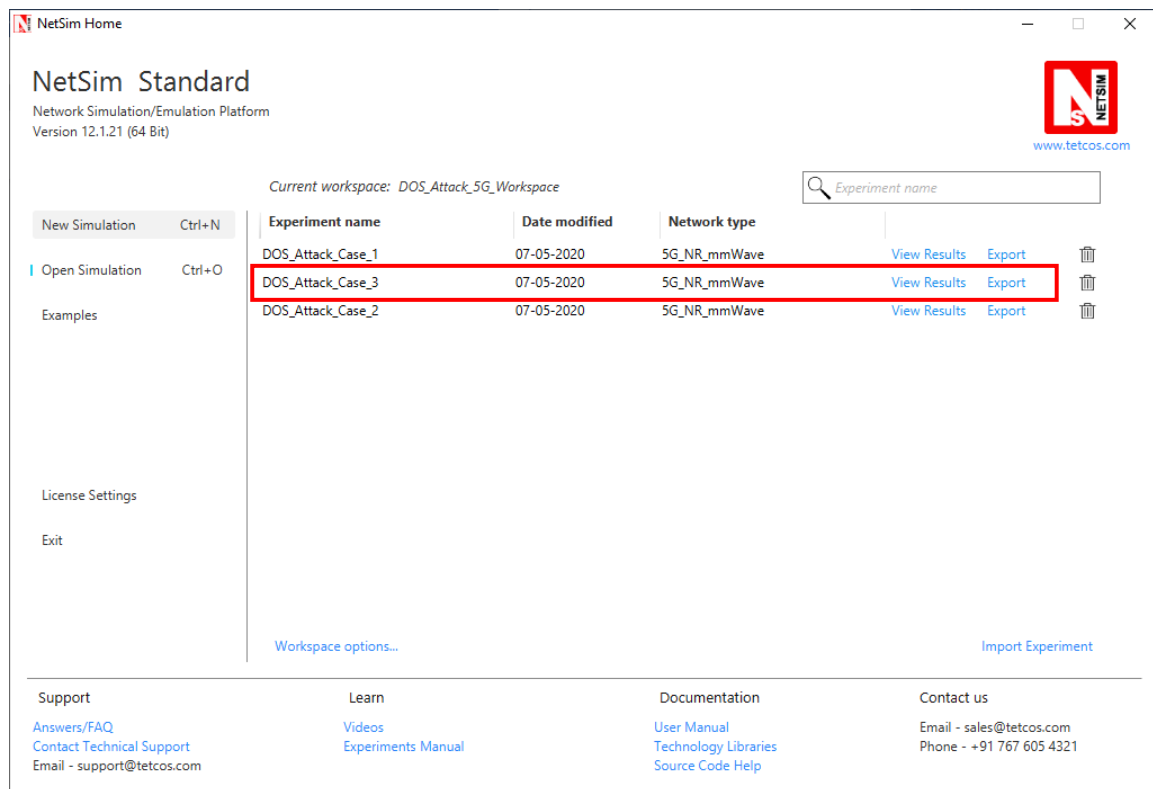
7. Right click on the solution in the solution explorer and select Rebuild.



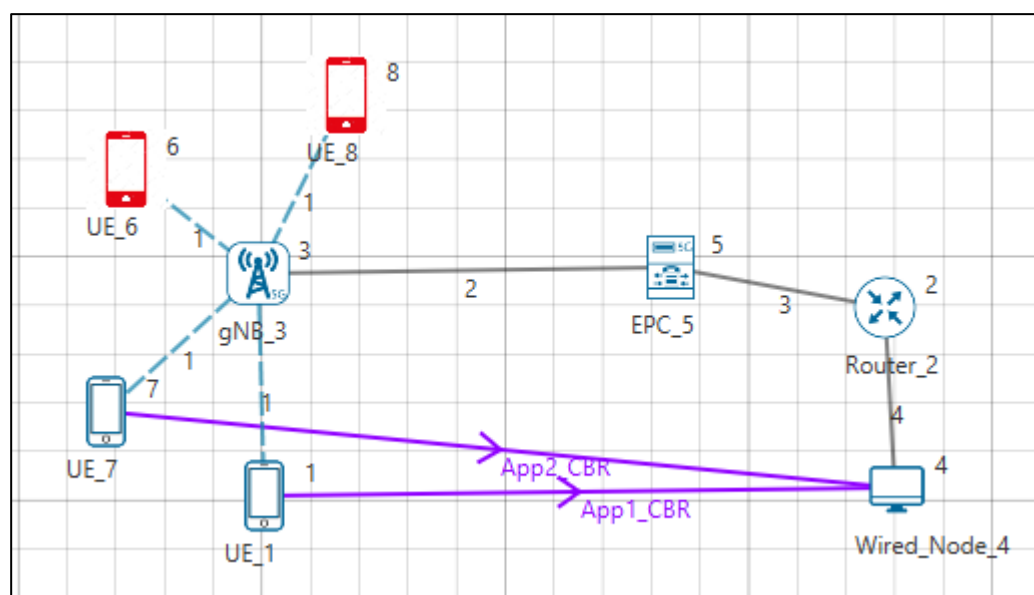
8. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries. (Note: first rebuild the TCP project and then rebuild the Ethernet project)
9. Run the simulation for 5 seconds.

Case-3: With two Malicious Node

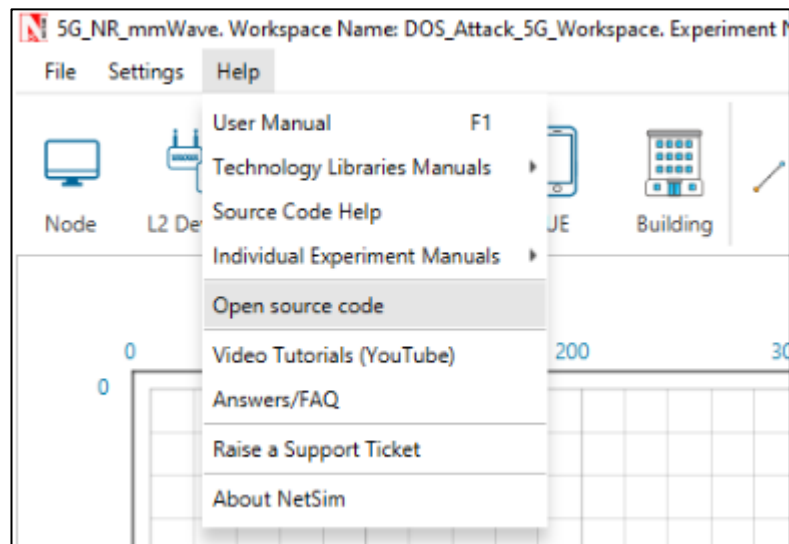
1. Then DOS_Attack_5G_Workspace comes with a sample configuration that is already saved. To open this example, go to Open Simulation and click on the DOS_Attack_Case_3 that is present under the list of experiments as shown below:



2. The saved network scenario consisting of 4 UEs, 1 gNB, 2 router, 1 EPC, 1 Router and 1 wired node in the grid environment forming a 5G NR mmWavw Network. Traffic is configured from Wired node to the Wireless node.



3. Help  Open Source code



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

```

SYN_flood.c* TCP.h* RTO.c
TCP (Global Scope)
43
44 #pragma comment (lib,"NetworkStack.lib")
45
46 _declspec(dllexport) target_node;
47
48
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

```

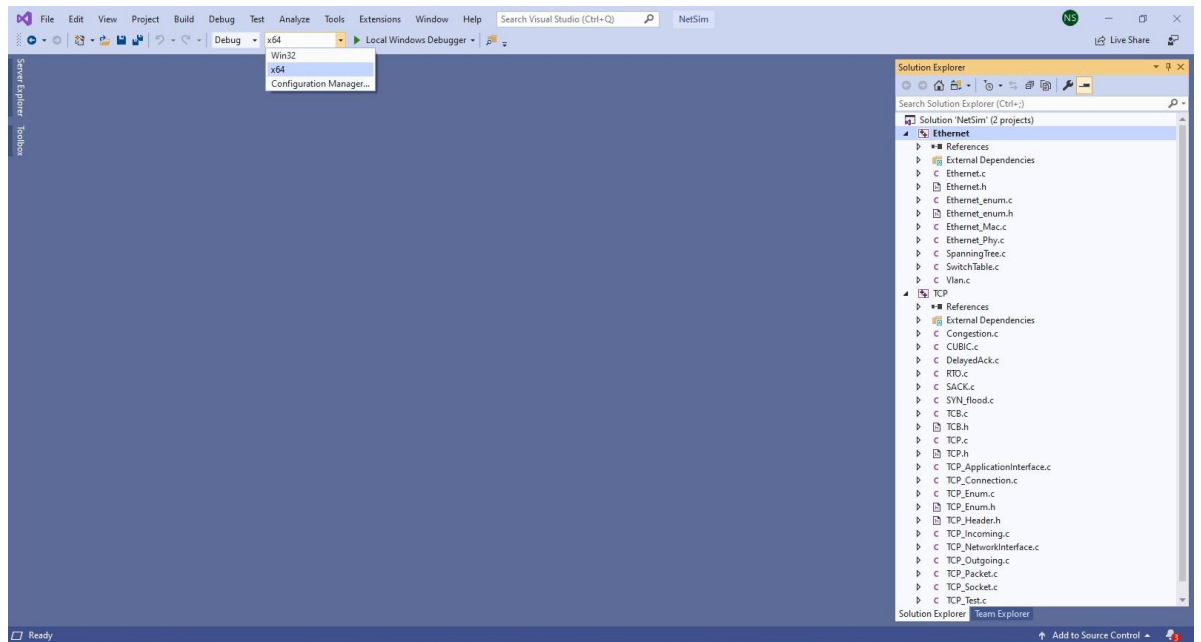
5. In SYN_FLOOD.c set **malicious node** as 6 , 8.

```

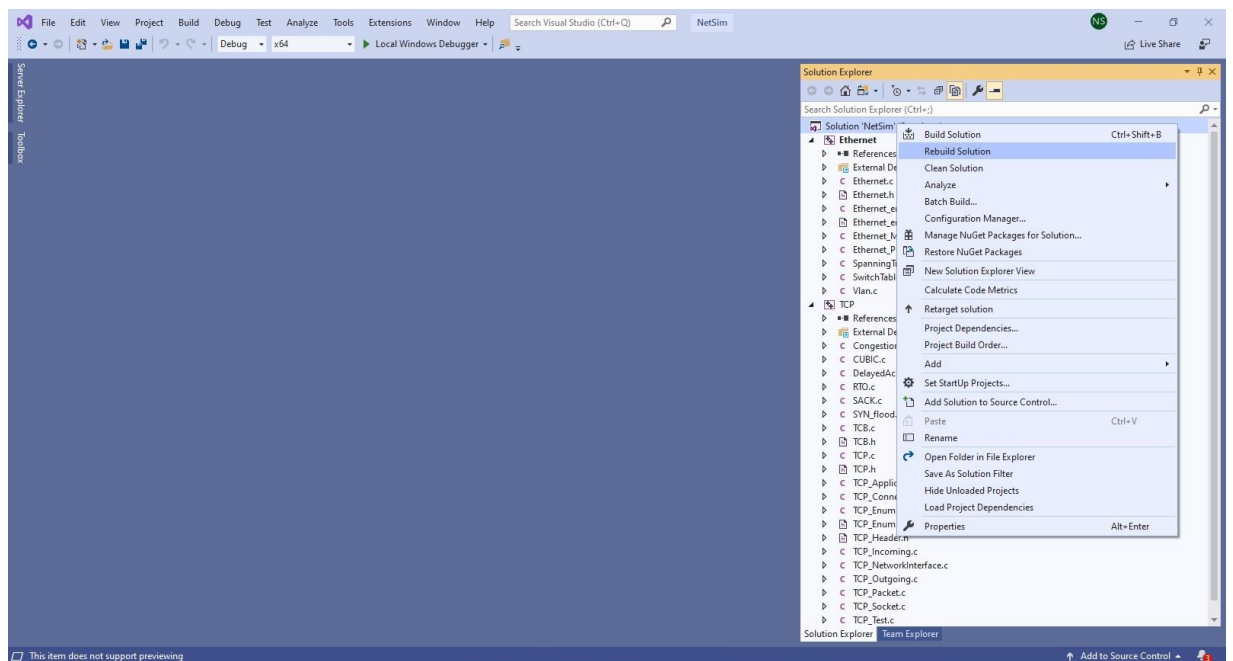
SYN_flood.c TCP.h
TCP (Global Scope)
12
13 *
14 * -----
15 #include "main.h"
16 #include "TCP.h"
17 #include "List.h"
18 #include "TCP_Header.h"
19 #include "TCP_Enum.h"
20
21 int malicious_node[NUMBEROFMALICIOUSNODE] = {6,8};
22 static void send_syn_packet(PNETSIM_SOCKET s);
23 //static PNETSIM_SOCKET socket_creation();
24 int target_node = 4;
25 PNETSIM_SOCKET get_Remotesocket(NETSIM_ID d, PSOCKETADDRESS addr);
26 static PNETSIM_SOCKET sockAddr = NULL;

```

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 5 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.

Simulation Results

- Simulation Results
 - Link_Metrics
 - Queue_Metrics**
 - TCP_Metrics
 - IP_Metrics
 - IP_Forwarding_Table
 - UDP_Metrics
 - Application_Metrics
 - LTENR_SDAP
 - Plots
 - Export Results (.xls/.csv)
 - Print Results (.html)
 - Open Packet Trace
 - Open Event Trace
 - Log Files
 - Restore To Original View

Queue_Metrics_Table

Queue_Metrics

Device_id	Port_id	Queued_packet	Dequeued_packet	Dropped_packet
2	1	1	1	0
2	2	10307	10307	0
5	1	0	0	0
5	2	10324	10324	0

Link_Metrics_Table

Link_Metrics

Link_id	Link_throughput_plot	Packet_transmit...		Packet_errored		Packet_collided	
		Data	Control	Data	Control	Data	Control
All	NA	46290	2	39	0	0	0
1	NA	15325	0	0	0	0	0
2	NA	10335	0	12	0	0	0
3	NA	10323	2	16	0	0	0
4	NA	10307	0	11	0	0	0

Application_Metrics_Table

Application_metrics

Application Id	Application Name	Packet generated	Packet received	Throughput (Mbps)	Del
1	App1_CBR	10289	5143	12.014045	124
2	App2_CBR	10289	5153	12.037408	124

TCP_Metrics_Table

TCP_Metrics

Source	Destination	Segment Sent	Segment Received	Ack Sent	Ack Received	Du
UE_1	ANY_DEVICE	0	0	0	0	0
ROUTER_2	ANY_DEVICE	0	0	0	0	0
WIRED_NODE_4	ANY_DEVICE	0	0	0	0	0
EPC_5	ANY_DEVICE	0	0	0	0	0
UE_7	ANY_DEVICE	0	0	0	0	0

	Throughput_APP1 (Mbps)	Throughput_APP2 (Mbps)
Case-1: Malicious Node =0	12.01	12.03
Case-2: Malicious Node =1	11.87	11.85
Case-3: Malicious Node =2	11.67	11.68

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

AutoSave ☐ Off

Event Trace.csv

Search

sagar.khetagouda

File Home Insert Page Layout Formulas Data Review View Help Table Design

Clipboard Font Alignment Number Styles Cells

	A	B	C	D	E	F	G	H	I	J	K
	Event_Id	Event_Type	Event_Time(US)	Device_Type	Device_Id	Interface_Id	Application_Id	Packet_Id	Segment_Id	Protocol_Name	Subevent_Type
78	1	TIMER_EVENT	1000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
109	97	TIMER_EVENT	2000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
132	129	TIMER_EVENT	3000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
173	152	TIMER_EVENT	4000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
275	195	TIMER_EVENT	5000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
316	295	TIMER_EVENT	6000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
418	338	TIMER_EVENT	7000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
463	438	TIMER_EVENT	8000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
607	485	TIMER_EVENT	9000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
652	627	TIMER_EVENT	10000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
771	674	TIMER_EVENT	11000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
816	791	TIMER_EVENT	12000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
960	838	TIMER_EVENT	13000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
1003	980	TIMER_EVENT	14000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
1147	1025	TIMER_EVENT	15000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
1190	1167	TIMER_EVENT	16000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
1315	1212	TIMER_EVENT	17000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
1366	1335	TIMER_EVENT	18000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
1510	1388	TIMER_EVENT	19000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
1555	1530	TIMER_EVENT	20000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
1699	1577	TIMER_EVENT	21000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD
1742	1719	TIMER_EVENT	22000	UE	6	0	0	0	0	0 TCP	SYN_FLOOD

Event Trace Pivot Table(Custom)

Note: Users can also create their own network scenarios in 5G NR mmWave and run simulation.