

# Dos Attack in Internet of Things

---

**Software Used:** NetSim Standard v12.0 (32/64 bit), Visual Studio 2015/2017/2019

Follow the instructions specified in the following link to clone/download the project folder from GitHub using Visual Studio:

<https://tetcos.freshdesk.com/support/solutions/articles/14000099351-how-to-clone-netsim-file-exchange-project-repositories-from-github->

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

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

**Secure URL for the GitHub repository:**

[https://github.com/NetSim-TETCOS/DOS\\_Attack\\_v12.0.git](https://github.com/NetSim-TETCOS/DOS_Attack_v12.0.git)

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.

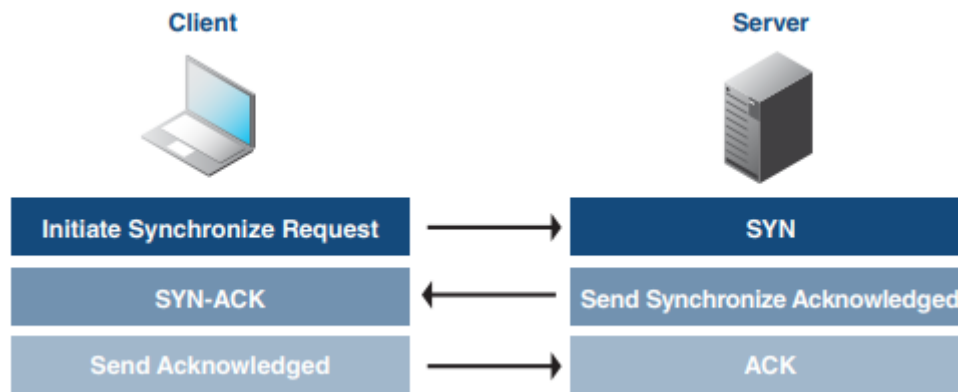
**Standard DDoS Attack types:**

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);`

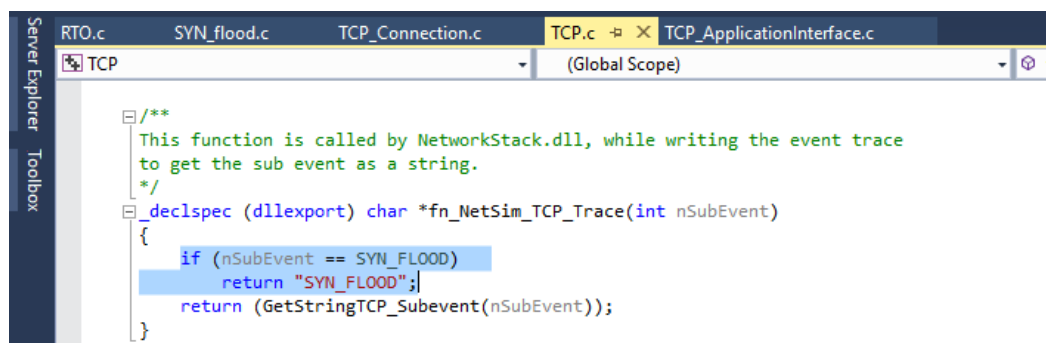
This function is used to create and send SYN packet to the network layer

- `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



```
Server Explorer  Toolbox
RTO.c  SYN_flood.c  TCP_Connection.c  TCP.c  TCP_ApplicationInterface.c
TCP (Global Scope)
/**
 * This function is called by NetworkStack.dll, while writing the event trace
 * to get the sub event as a string.
 */
_declspec(dllexport) char *fn_NetSim_TCP_Trace(int nSubEvent)
{
    if (nSubEvent == SYN_FLOOD)
        return "SYN_FLOOD";
    return (GetStringTCP_Subevent(nSubEvent));
}
```

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

```

static int fn_NetSim_TCP_HandleTimer()
{
    switch (pstruEventDetails->nSubEventType)
    {
        case TCP_RTO_TIMEOUT:
            handle_rto_timer();
            break;
        case TCP_TIME_WAIT_TIMEOUT:
            handle_time_wait_timeout();
            break;
        case SYN_FLOOD:
            syn_flood();
            break;
        default:
            fnNetSimError("Unknown subevent %d in %s\n",
                pstruEventDetails->nSubEventType,
                __FUNCTION__);
            break;
    }
    return 0;
}

```

3. And modified the following lines of code in fn\_NetSim\_TCP\_Init() function resent in TCP.c inside TCP project

```

79  /*
80  _declspec (dllimport) 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      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->rtcb->retransmissionQueue, p, pstruEventDetails->dEventTime);
61     NetSim_EVENTDETAILS pevent;
62     memcpy(&pevent, pstruEventDetails, sizeof pevent);
63     pevent.dEventTime += TCP_RTO(s->rtcb);
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         if (!is_malicious_node(pevent.nDeviceId))
76         {
77             pevent.nProtocolId = TX_PROTOCOL_TCP;
78             pevent.pPacket = fn_NetSim_Packet_CopyPacket(p);
79             pevent.szOtherDetails = NULL;
80             pevent.nSubEventType = TCP_RTO_TIMEOUT;
81             fnAddEvent(&pevent);
82             print_tcp_log("Adding RTO Timer at %.1lf", pevent.dEventTime);
83         }
84     }
85
86 static void handle_rto_timer_for_ctrl(PNETSIM_SOCKET s)
87 {
88     if (isSynbitSet(pstruEventDetails->pPacket))
89         record_syn(s);
90 }

```

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

```

31 * RFC 8312 : CUBIC for Fast Long-Distance Network
32 * https://ieeexplore.ieee.org/document/1354672
33 * https://research.csc.ncsu.edu/netsrv/sites/default/files/hystart_techreport_2008.pdf
34 */
35
36 // #define _TEST_TCP // Enable to test the TCP
37 #ifdef _TEST_TCP
38     static double dropProbability = 0.5;
39     static bool burstDrop = true;
40     static int burstDropSize = 10;
41     static bool isAckDrop = true;
42 #endif
43
44 #pragma comment (lib, "NetworkStack.lib")
45
46 _declspec(dllexport) target_node;
47
48 // USEFUL MACRO
49 #define isTCPConfigured(d) (DEVICE_TRXLayer(d) && DEVICE_TRXLayer(d)->isTCP)
50 #define isTCPControl(p) (p->nControlDataType/100 == TX_PROTOCOL_TCP)
51
52 // Constant
53 #define TCP_DupThresh 3
54 #define NUMBEROFMALIGNOUSNODE 2
55 int is_malicious_node(NETSIM_ID devid);
56 // Typedef
57 typedef struct stru_TCP_Socket NETSIM_SOCKET, *PNETSIM_SOCKET;
58
59 typedef enum enum_tcpstate
60 {
61     TCPCONNECTION_CLOSED,
62     TCPCONNECTION_LISTEN,
63     TCPCONNECTION_SYN_SENT,
64     TCPCONNECTION_SYN_RECEIVED,
65     TCPCONNECTION_ESTABLISHED,
66     TCPCONNECTION_FTM_WAIT_1
67 }

```

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

```

7  * Product or its content without express prior written consent of Tetcos is
8  * prohibited. Ownership and / or any other right relating to the software and all
9  * intellectual property rights therein shall remain at all times with Tetcos.
10 *
11 * Author: Soniya
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] = { 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     return 0;
33 }
34
35 void syn_flood()
36 {
37     /*
38     if (!sockAddr)
39     {
40         sockAddr = caller(1, sizeof * sockAddr);
41     }
42     */
43 }

```

- 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)

```

- SYN\_FLOOD.c file contains the following functions

```

7  * Product or its content without express prior written consent of Tetcos is
8  * prohibited. Ownership and / or any other right relating to the software and all
9  * intellectual property rights therein shall remain at all times with Tetcos.
10 *
11 * Author: Soniya
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] = { 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     return 0;
33 }
34
35 void syn_flood()
36 {
37     /*
38     if (!sockAddr)
39     {
40         sockAddr = caller(1, sizeof * sockAddr);
41     }
42     */
43 }

```

```

SYN_flood.c* TCP.h* RIO.c
TCP (Global Scope) syn_flood()
34 }
35
36 void syn_flood()
37 {
38     extern PSocketAddress anySocketAddr;
39     anySocketAddr->ip = DEVICE_IPADDRESS(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.nSubEventId = SYN_FLOOD;
61         fnpAddEvent(&pEvent);
62     }
63 }
64
65
66
67
68

```

```

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

```

```

SYN_flood.c* TCP.h* RIO.c
TCP (Global Scope) socket_creation()
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
91     add_to_socket_list(pstruEventDetails->nDeviceId, newSocket);
92
93     PSocketAddress localsocketAddr = (PSocketAddress)calloc(1, sizeof * localsocketAddr);
94     localsocketAddr->ip = DEVICE_IPADDRESS(pstruEventDetails->nDeviceId, 1);
95     localsocketAddr->port = 0;
96
97     PSocketAddress remotsocketAddr = (PSocketAddress)calloc(1, sizeof * remotsocketAddr);
98     remotsocketAddr->ip = DEVICE_IPADDRESS(target_node, 1);
99     remotsocketAddr->port = 0;
100
101     newSocket->SocketId = s_id;
102     s_id++;
103
104     newSocket->localAddr = localsocketAddr;
105     newSocket->remoteAddr = remotsocketAddr;
106
107     newSocket->localDeviceId = pstruEventDetails->nDeviceId;
108     newSocket->remoteDeviceId = target_node;
109
110     newSocket->sId = sId;
111
112     return newSocket;
113 }
114

```

9. Added PROCESSING\_TIME macro in Ethernet.h file inside ETHERNET project

```

7  * Product or its content without express prior written consent of Tetcos is
8  * prohibited. Ownership and / or any other right relating to the software and all
9  * intellectual property rights therein shall remain at all times with Tetcos.
10 *
11 * Author: Shashi Kant Suman
12 *
13 * -----*/
14
15 #ifndef _NETSIM_ETHERNET_H
16 #define _NETSIM_ETHERNET_H
17 #ifdef __cplusplus
18 extern "C" {
19 #endif
20
21 #pragma comment(lib, "NetworkStack.lib")
22 #pragma comment(lib, "Metrics.lib")
23 #pragma comment(lib, "libTCP")
24
25 #define isETHConfigured(d,i) (DEVICE_MACLAYER(d,i)->nMacProtocolId == MAC_PROTOCOL_IEEE802_3)
26 //Global variable
27 PNETSIM_MACADDRESS multicastSPTMAC;
28
29 #define ETH_IFG 0.960 //Micro sec
30
31 #define Processing_TIME 1000
32

```

10. Modified the following lines of code in fn\_NetSim\_Ethernet\_HandlePhyOut() function present in Ethernet\_Phy.c file inside Ethernet project.

```

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

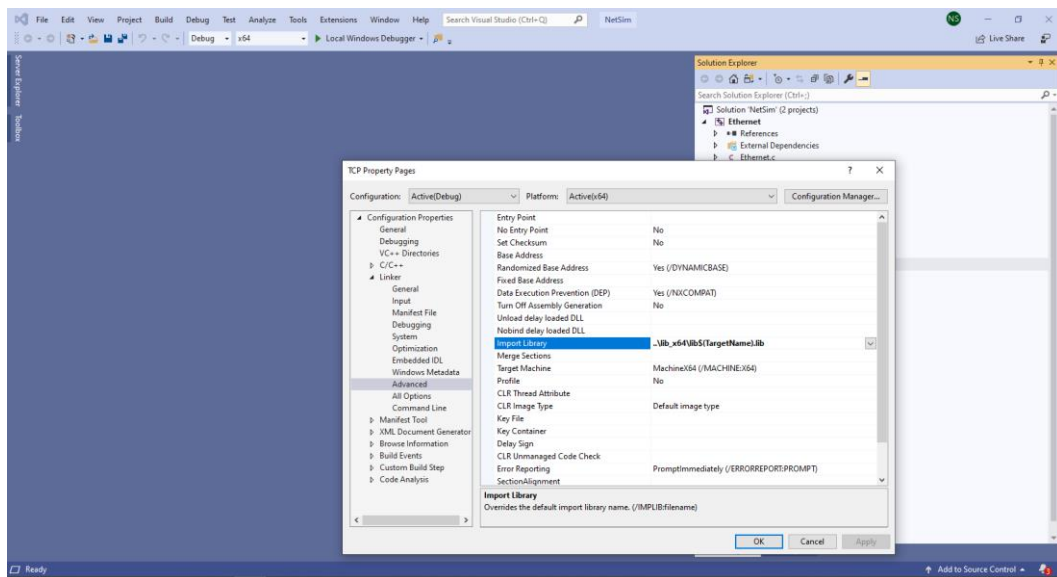
double start;

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;
}

```

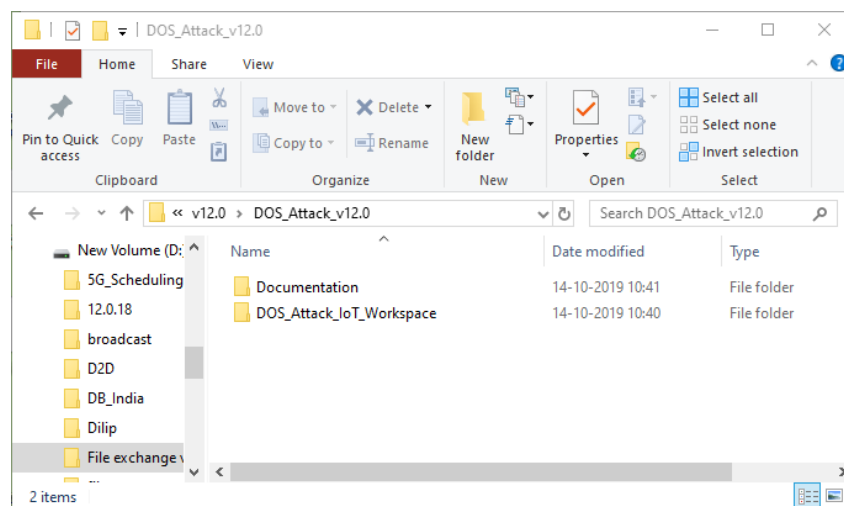


11. Right click on TCP project → Properties→Linker → Advanced →import library 32-bit and 64-bit  
**..\lib\lib\$(TargetName).lib or ..\lib\_x64\lib\$(TargetName).lib**

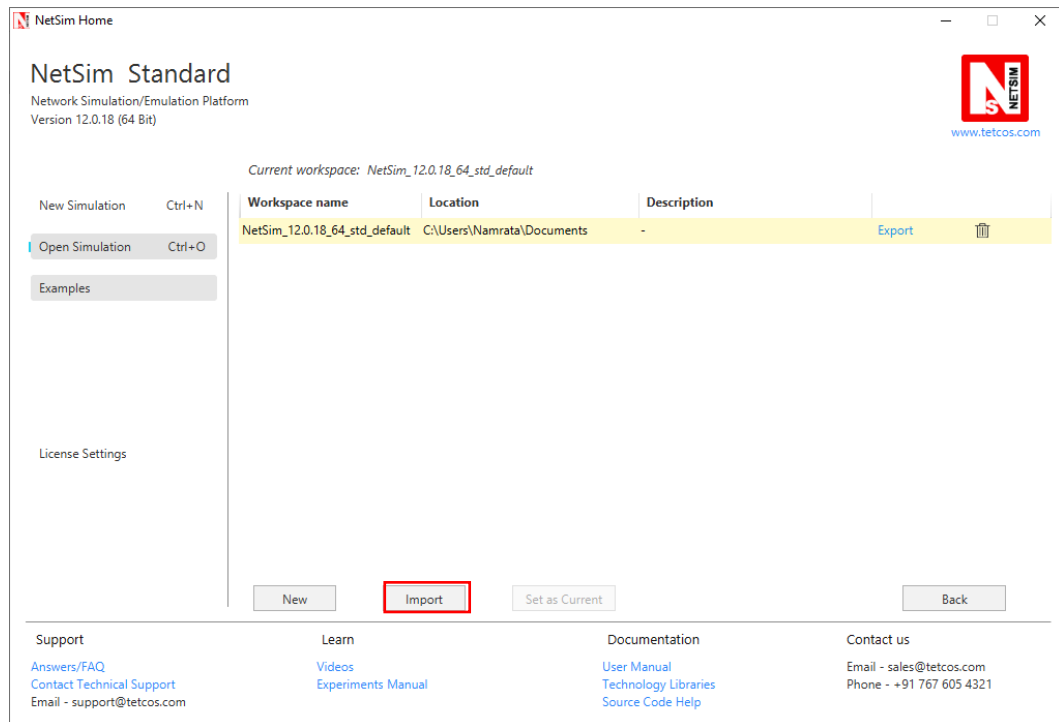


### Steps:

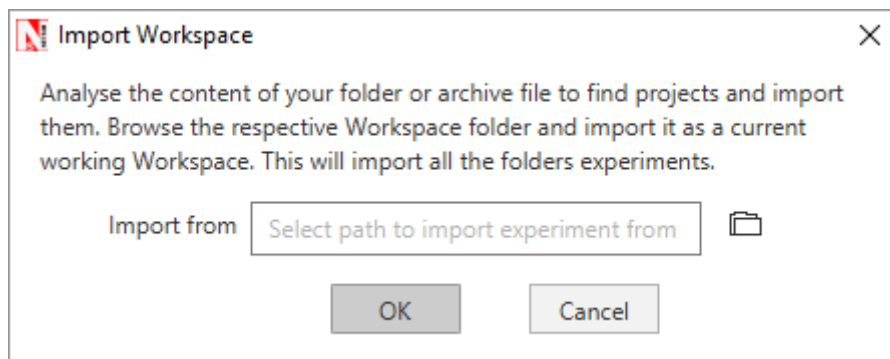
1. The downloaded project folder contains the folders Documentation, and DOS\_Attack\_IoT\_Workspace directory as shown below:



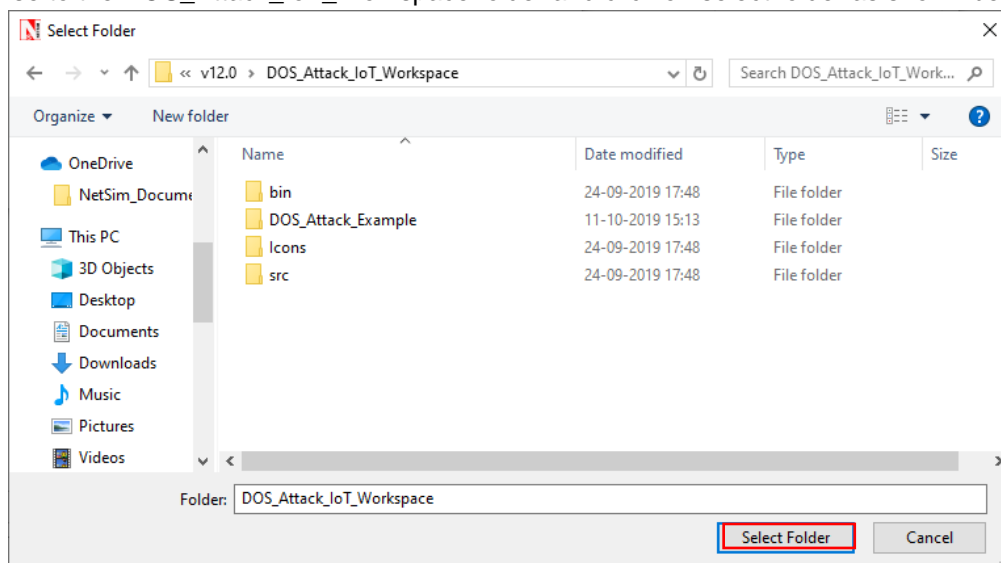
2. Import DOS\_Attack\_IoT\_Workspace by going to Open Simulation->Workspace Options->More Options in NetSim Home window. Then select Import as shown below:



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

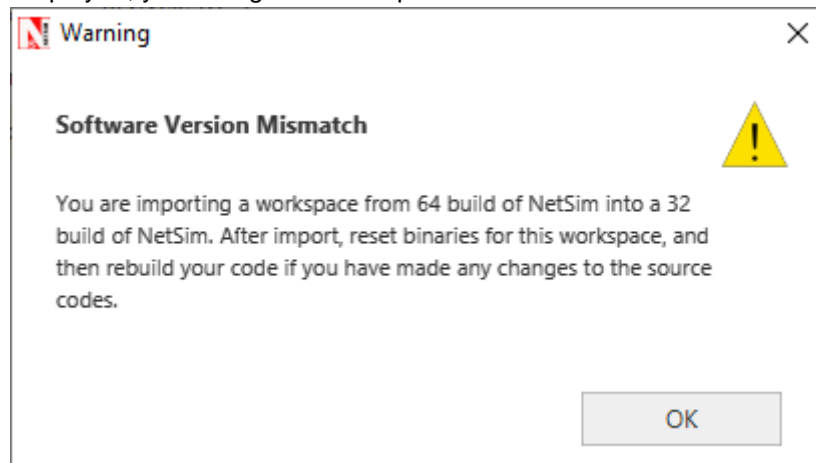


- Browse to the DOS\_Attack\_IoT\_Workspace folder and click on select folder as shown below:

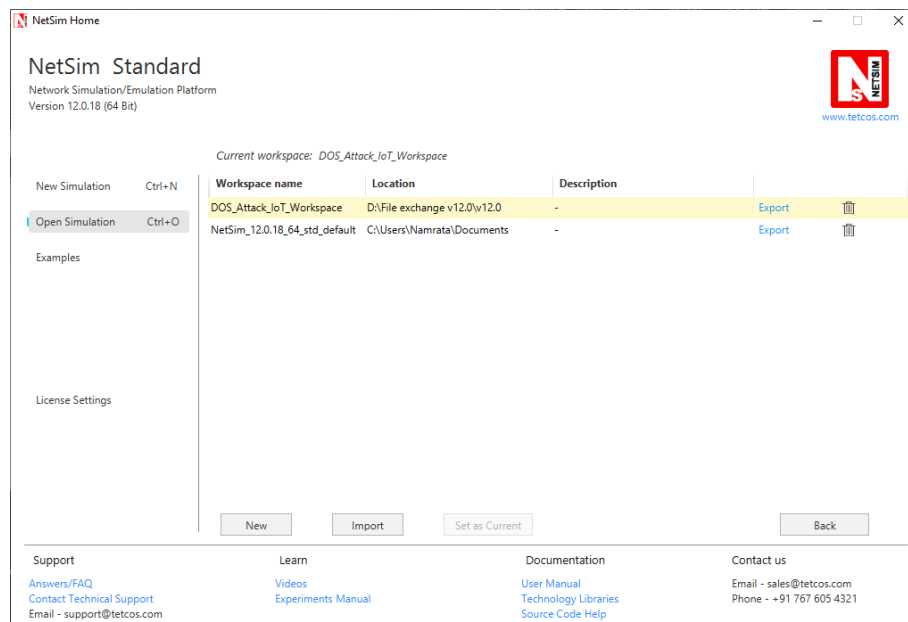


- After this click on OK button in the Import Workspace window.

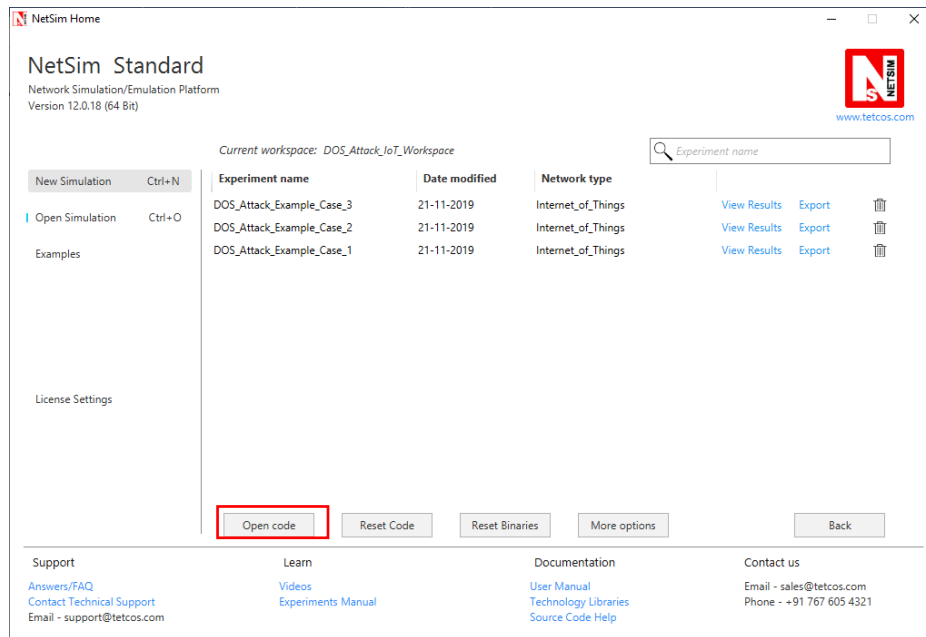
6. While importing the workspace, if the following warning message indicating Software Version Mismatch is displayed, you can ignore it and proceed.



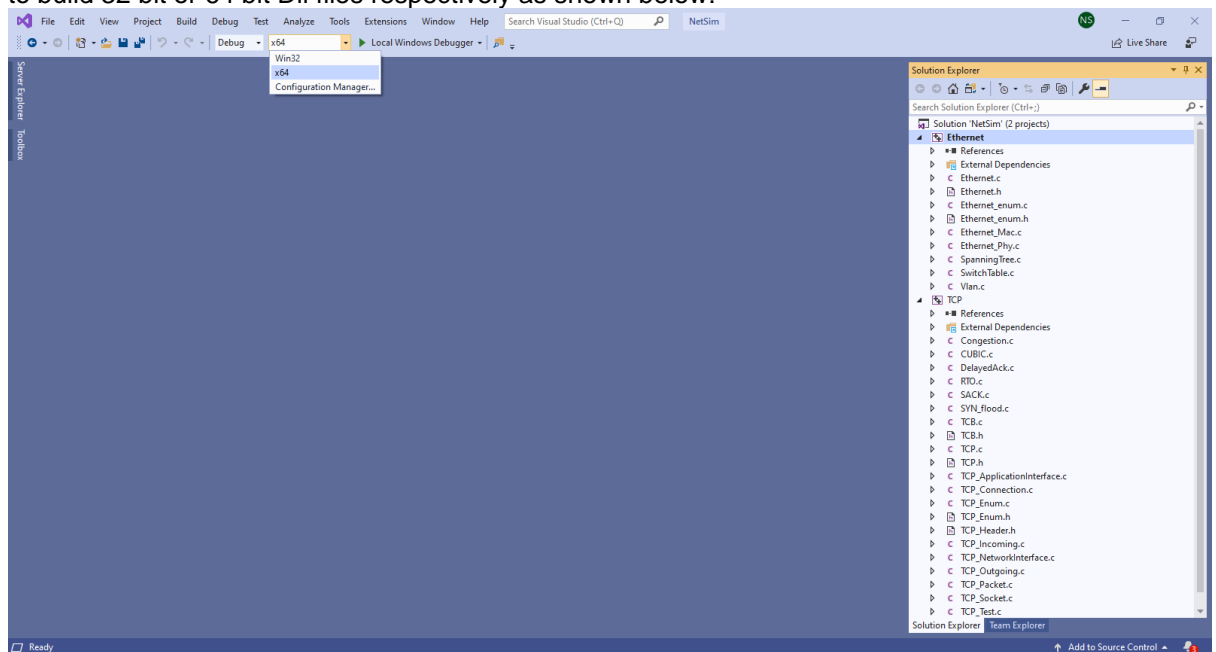
7. The Imported workspace will be set as the current workspace automatically. To see the imported workspace, click on Open Simulation->Workspace Options->More Options as shown below:



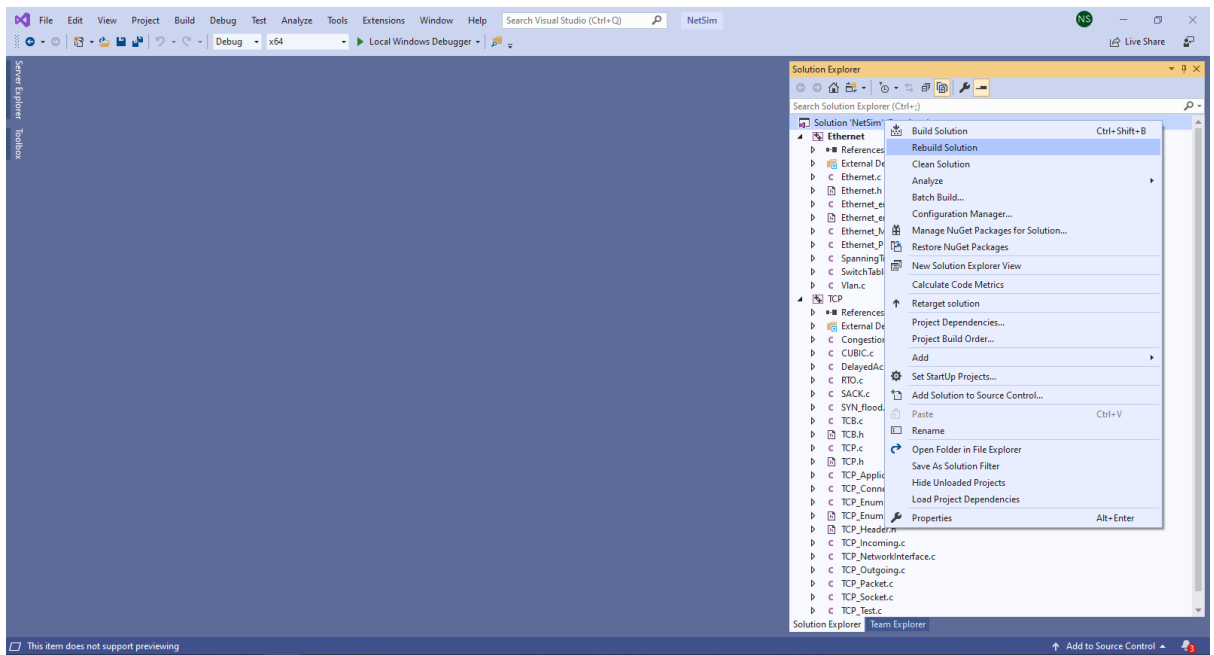
8. Open the Source codes in Visual Studio by going to Open Simulation-> Workspace Options and Clicking on Open code button as shown below:



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:



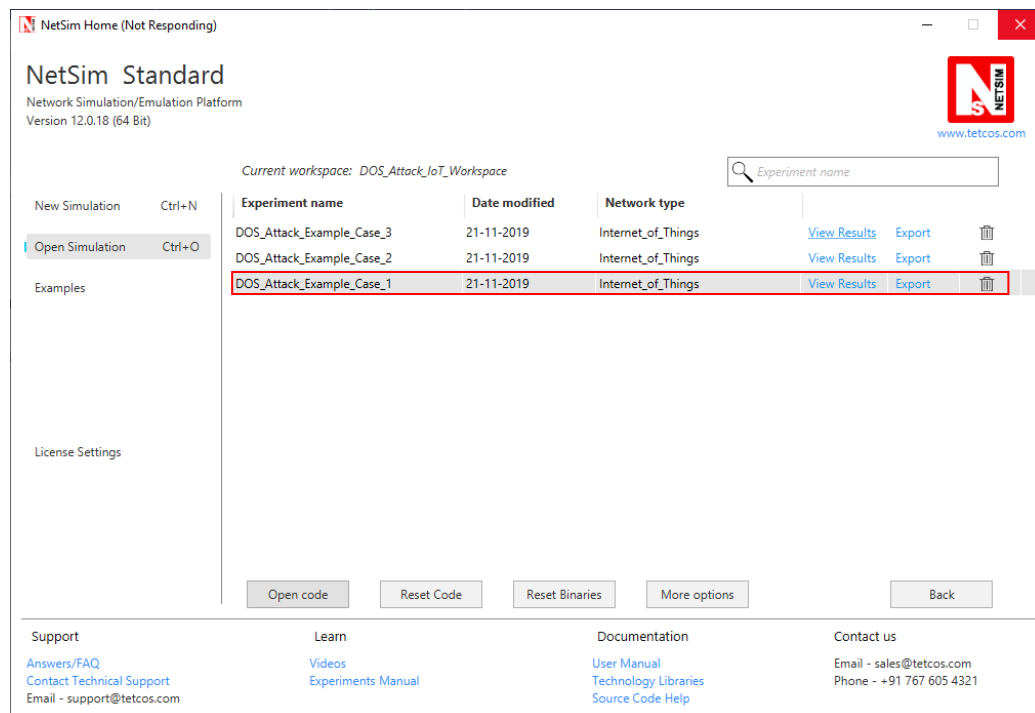
11. Right click on the solution in the solution explorer and select Rebuild.



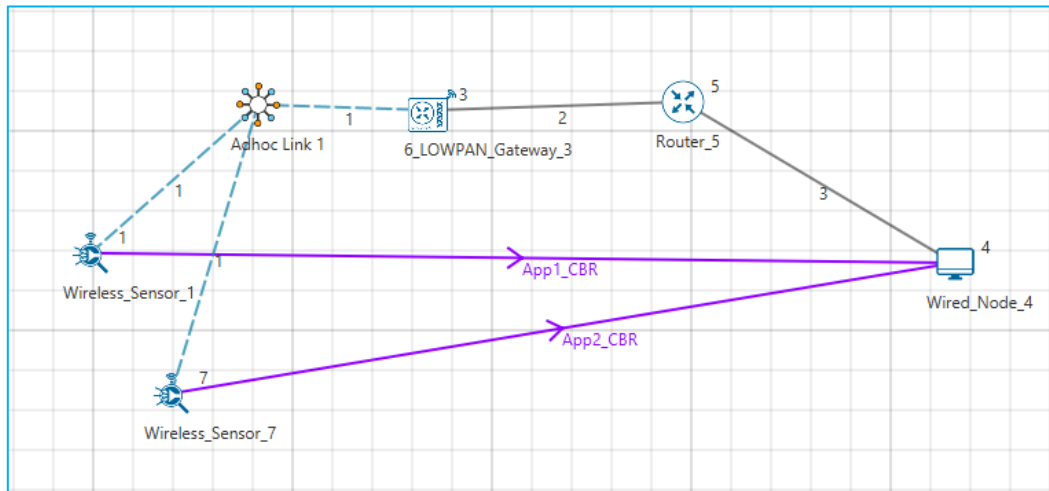
12. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries. (Note: first rebuild the TCP project and then rebuild the Ethernet project)
13. Run NetSim as Administrative mode.

### Case-1: Without Malicious Node

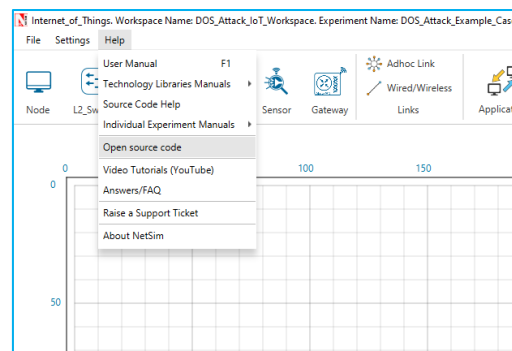
1. Then DOS\_Attack\_IoT\_Workspace comes with a sample configuration that is already saved. To open this example, go to Open Simulation and click on the DOS\_Attack\_Example\_Case\_1 that is present under the list of experiments as shown below:



- 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.



- Help → Open Source code



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

```

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);

```

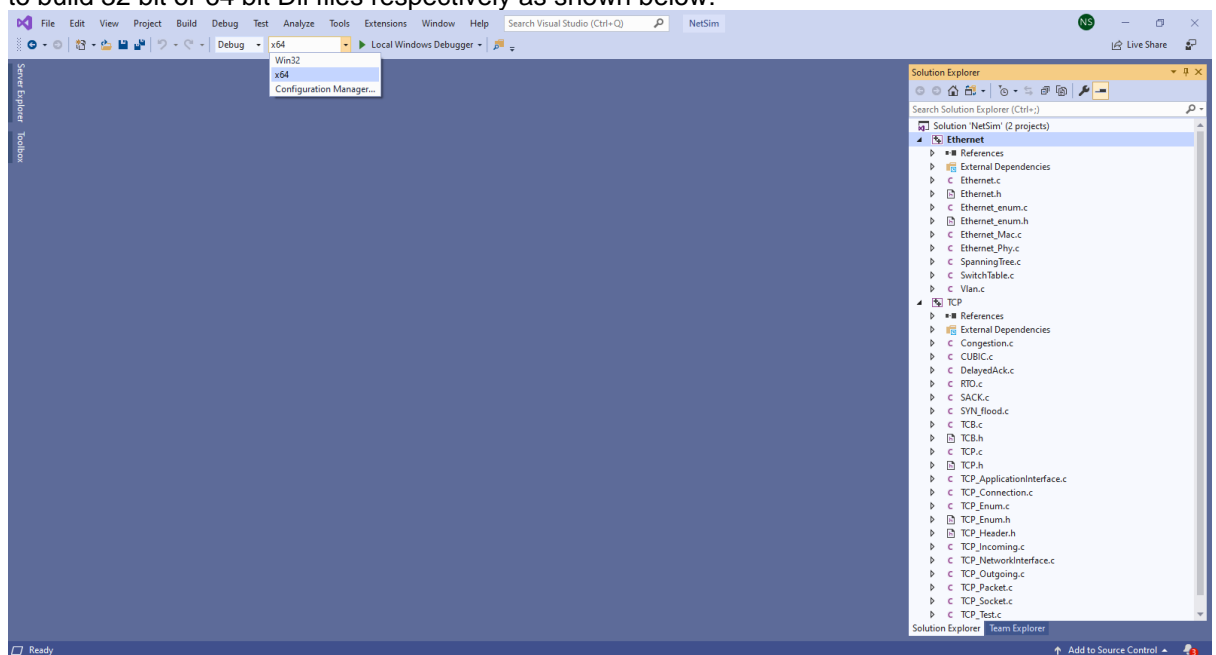
- In SYN\_FLOOD.c set **malicious node** as 0.

```

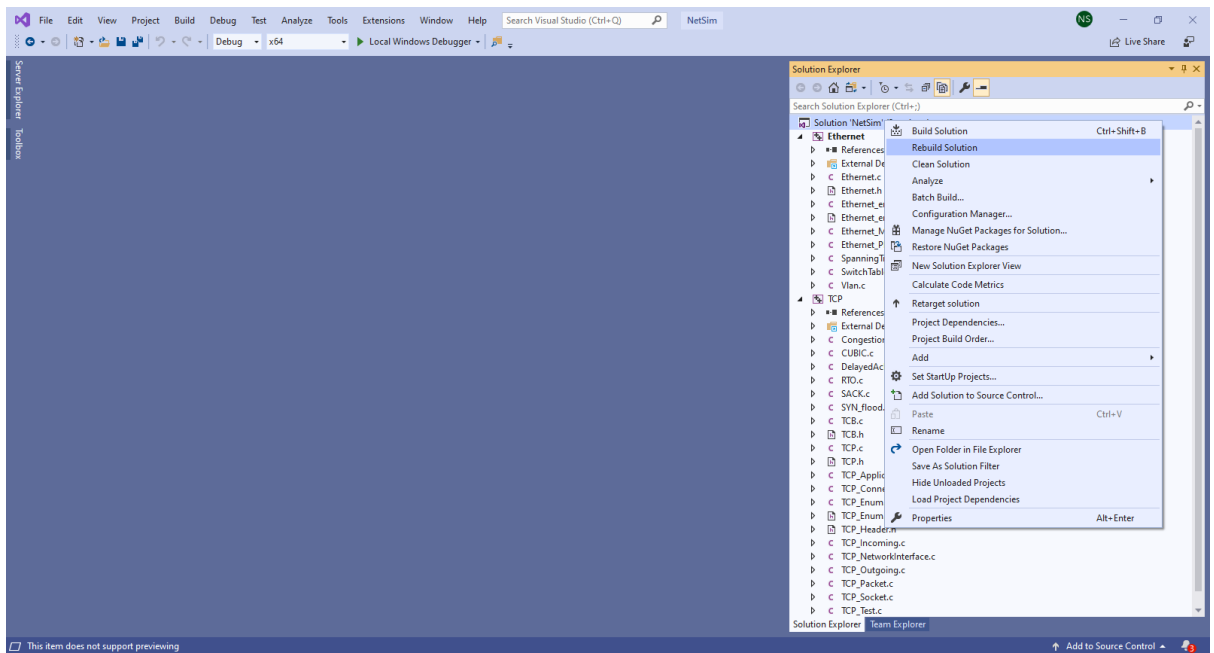
SYN_flood.c* TCP.h RTO.c
TCP (Global Scope)
7  * Product or its content without express prior written consent of Tetcos is
8  * prohibited. Ownership and / or any other right relating to the software and all
9  * intellectual property rights therein shall remain at all times with Tetcos.
10 *
11 * Author: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

```

- 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:



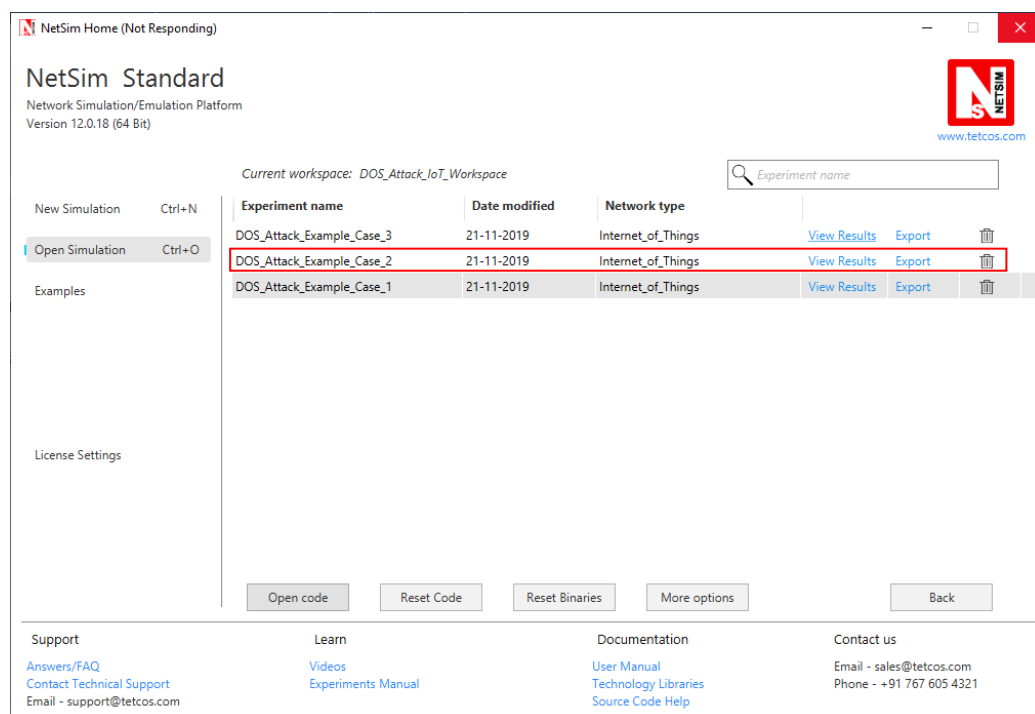
- Right click on the solution in the solution explorer and select Rebuild.



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 100 seconds.

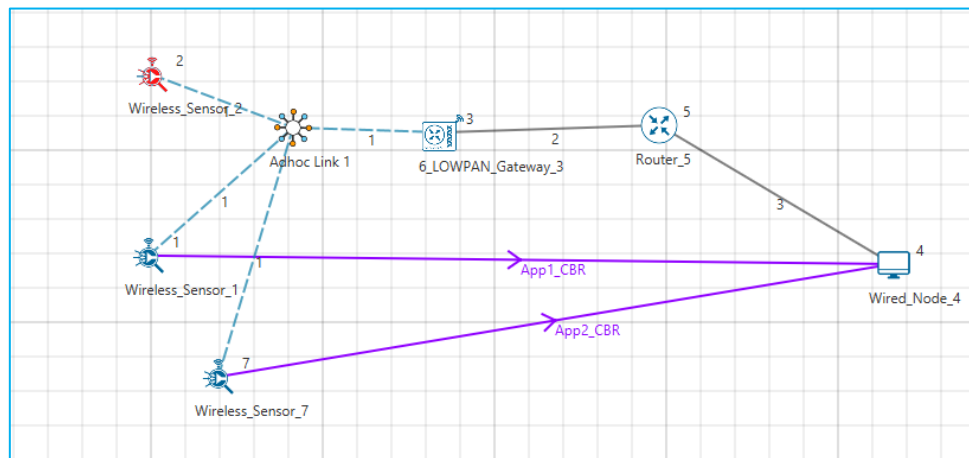
## Case-2: With one Malicious Node

1. Then DOS\_Attack\_IoT\_Workspace comes with a sample configuration that is already saved. To open this example, go to Open Simulation and click on the DOS\_Attack\_Example\_Case\_2 that is present under the list of experiments as shown below:

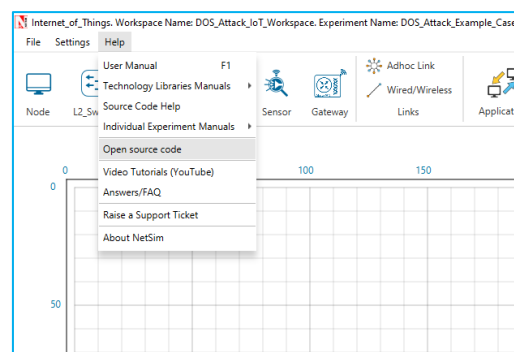




- The saved network scenario consisting of 3 sensors, 1 6LOWPAN Gateway, 1 router, and 1 wired node in the grid environment forming a IoT Network. Traffic is configured from sensor node to the Wired Node.



- Help → Open Source code



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

```

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);
  
```

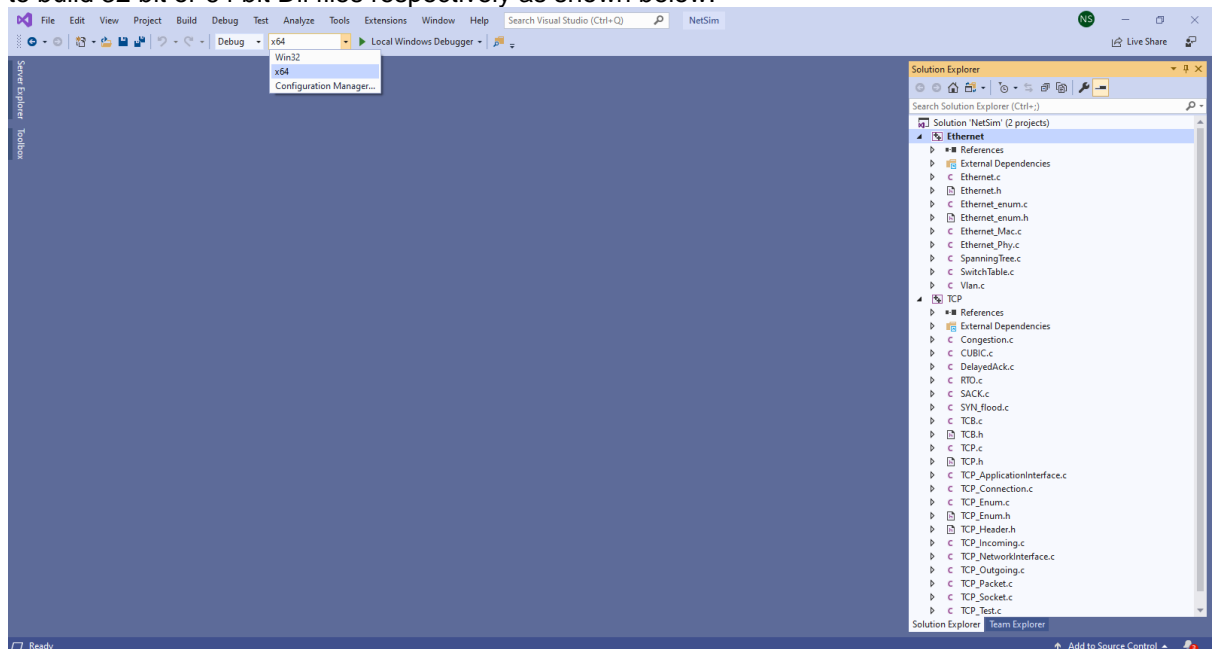
- In SYN\_FLOOD.c set **malicious node** as 2.

```

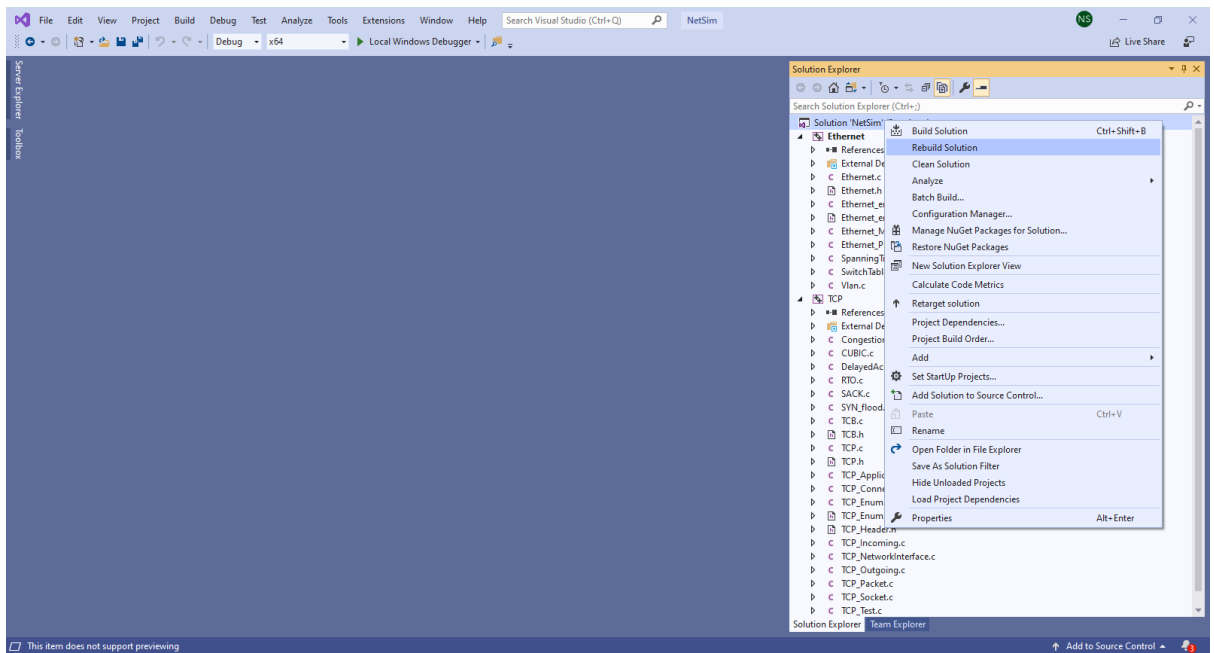
7  * Product or its content without express prior written consent of Tetcos is
8  * prohibited. Ownership and / or any other right relating to the software and all
9  * intellectual property rights therein shall remain at all times with Tetcos.
10 *
11 * Author: Soniya
12 *
13 * -----*/
14
15 #include "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] = { 2 };
22 static void send_syn_packet(PNETSIM_SOCKET s);
23 //static PNETSIM_SOCKET socket_creation();

```

- 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:



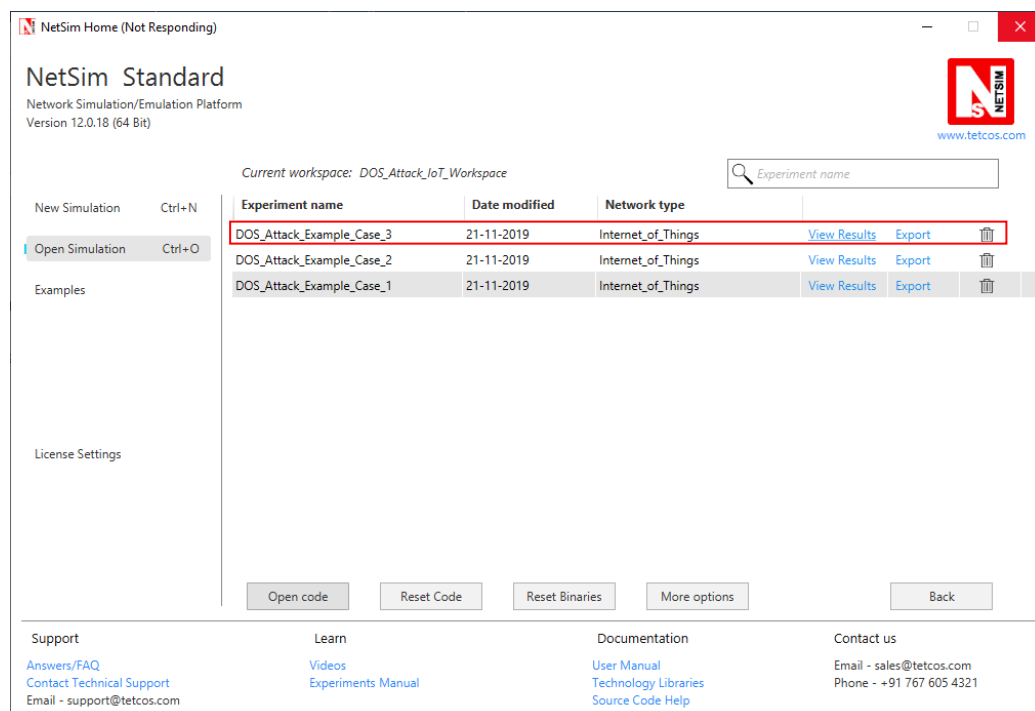
- Right click on the solution in the solution explorer and select Rebuild.



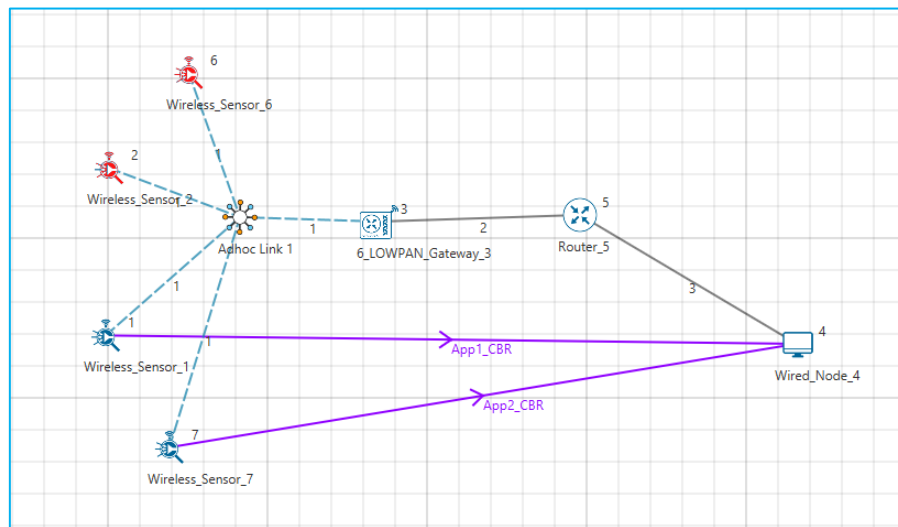
8. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries. (Note: first rebuild the TCP project and then rebuild the Ethernet project)
9. Run the simulation for 100 seconds.

### Case-3: With two Malicious Node

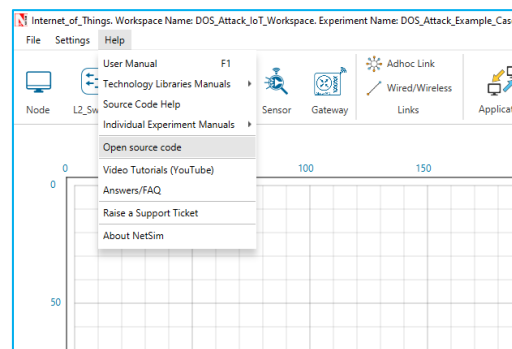
1. Then DOS\_Attack\_IoT\_Workspace comes with a sample configuration that is already saved. To open this example, go to Open Simulation and click on the DOS\_Attack\_Example\_Case\_3 that is present under the list of experiments as shown below:



- 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.



- Help → Open Source code



- 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

```

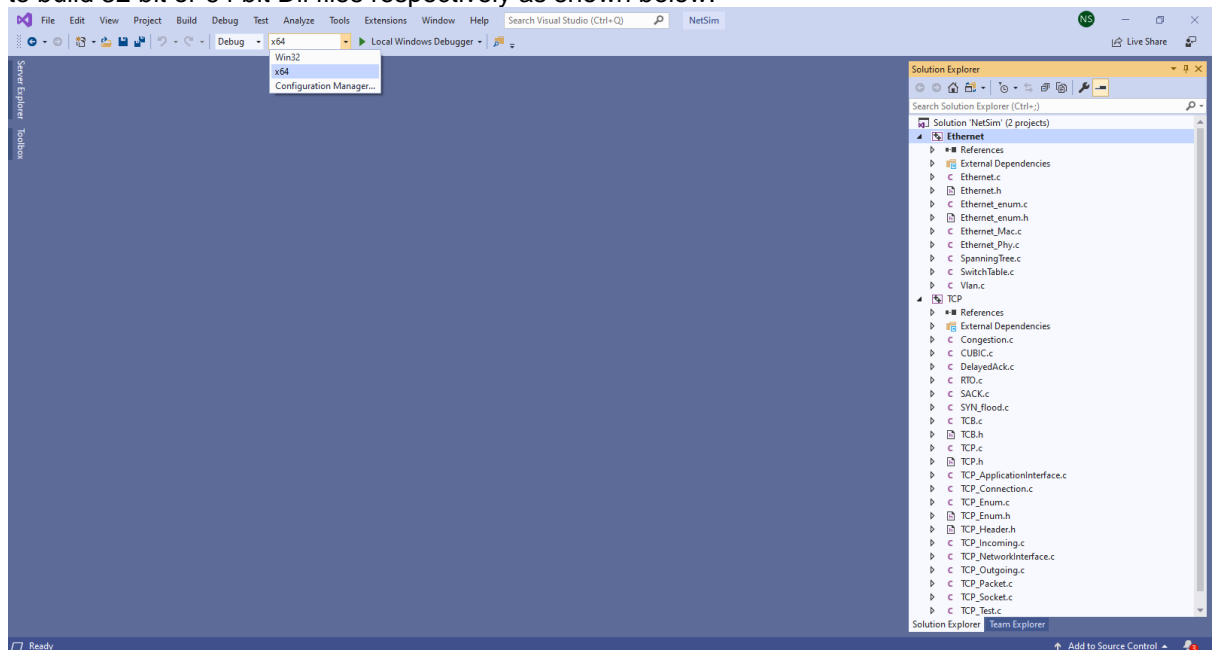
- In SYN\_FLOOD.c set **malicious node** as 2, 6.

```

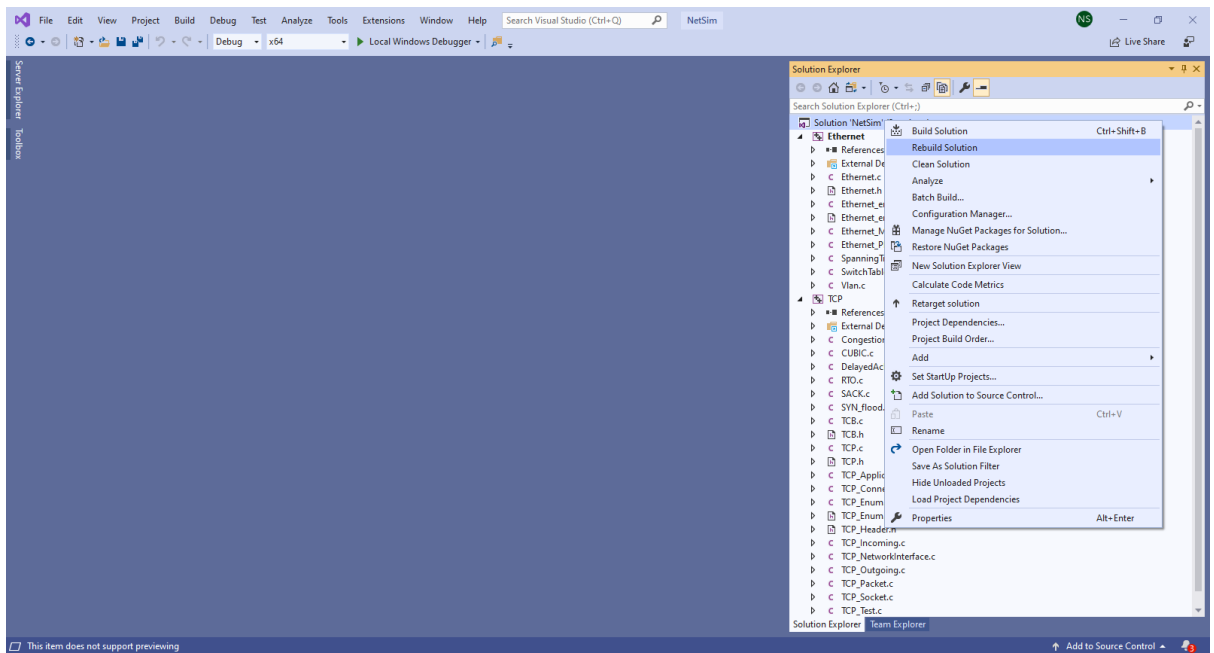
7  * Product or its content without express prior written consent of Tetcos is
8  * prohibited. Ownership and / or any other right relating to the software and all
9  * intellectual property rights therein shall remain at all times with Tetcos.
10
11  * Author:   Soniya
12  *
13  * -----*/
14
15  #include "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] = { 2, 6 };
22  static void send_syn_packet(PNETSIM_SOCKET s);
23  //static PNETSIM_SOCKET socket_creation();
24  int target_node = 4;

```

- 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:



- 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 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 sends from the malicious node, in case 1 there is no malicious node so there will be no SYN\_FLOOD packets.

Simulation Results

Link\_Metrics

Queue\_Metrics

Application\_Metrics

Export Results (.xls/.csv)

Print Results (.html)

Open Packet Trace

Open Event Trace

Log Files

Restore To Original View

Device_id	Port_id	Queued_packet	Dequeued_packet	Dropped_packet
3	1	14	14	0
3	2	18645	18645	0
5	1	16	16	0
5	2	18628	18628	0

Link_id	Link_throughput_plot	Data	Control	Packet_transmit...	Packet_errorred	Packet_collided
All	NA	38244	19632	5	0	795 915
1	NA	13278	7309	0	0	795 915
2	NA	12483	6178	1	0	0 0
3	NA	12483	6145	4	0	0 0

Application Id	Application Name	Packet generated	Packet received	Throughput (Mbps)	Delay(microsec)	Jitter
1	App1_CBR	75000	6220	0.048426	44251048.584468	1523
2	App2_CBR	75000	6259	0.048728	44087098.090275	1517

Source	Destination	Segment Sent	Segment Received	Ack Sent	Ack Received	Duplicate ack r
WIRELESS_SENSOR_2	ANY_DEVICE	0	0	0	0	0
6_LOWPAN_GATEWAY_3	ANY_DEVICE	0	0	0	0	0
WIRED_NODE_4	ANY_DEVICE	0	0	0	0	0
ROUTER_5	ANY_DEVICE	0	0	0	0	0
WIRELESS_SENSOR_6	ANY_DEVICE	0	0	0	0	0

	Throughput_APP1 (Mbps)	Throughput_APP2 (Mbps)
<b>Case-1: Malicious Node =0</b>	0.060768	0.060779
<b>Case-2: Malicious Node =1</b>	0.048549	0.049747
<b>Case-3: Malicious Node =2</b>	0.042493	0.041694

Go to the result window open packet trace, user can find out the SYN\_FLOOD packets via filtering subevent type as SYN\_FLOOD.

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
76	1 TIMER_EVENT	1000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	0
78	94 TIMER_EVENT	2000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	1
80	96 TIMER_EVENT	3000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	94
82	98 TIMER_EVENT	4000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	96
84	100 TIMER_EVENT	5000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	98
86	102 TIMER_EVENT	6000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	100
94	104 TIMER_EVENT	7000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	102
97	116 TIMER_EVENT	8000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	104
128	120 TIMER_EVENT	9000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	116
130	162 TIMER_EVENT	10000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	120
141	164 TIMER_EVENT	11000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	162
144	176 TIMER_EVENT	12000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	164
152	179 TIMER_EVENT	13000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	176
162	186 TIMER_EVENT	14000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	179
176	195 TIMER_EVENT	15000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	186
182	208 TIMER_EVENT	16000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	195
193	213 TIMER_EVENT	17000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	208
213	230 TIMER_EVENT	18000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	213
219	249 TIMER_EVENT	19000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	230
223	257 TIMER_EVENT	20000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	249
281	263 TIMER_EVENT	21000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	257
283	318 TIMER_EVENT	22000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	263
296	320 TIMER_EVENT	23000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	318
301	333 TIMER_EVENT	24000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	320
309	339 TIMER_EVENT	25000	SENSOR	2	0	0	0	0	0 TCP	SYN_FLOOD	0	333
316	346 TIMER_EVENT	26000	SENSOR	2	0	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.