

Dos Attack in 5G NR

Software Used: NetSim Standard v13.1 (64 bit), Visual Studio 2019

Project Download Link:

https://github.com/NetSim-TETCOS/DOS_Attack_in_5G_v13.0/archive/refs/heads/main.zip

Follow the instructions specified in the following link to download and setup the Project in NetSim:

<https://support.tetcos.com/en/support/solutions/articles/14000128666-downloading-and-setting-up-netsim-file-exchange-projects>

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 microseconds 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:

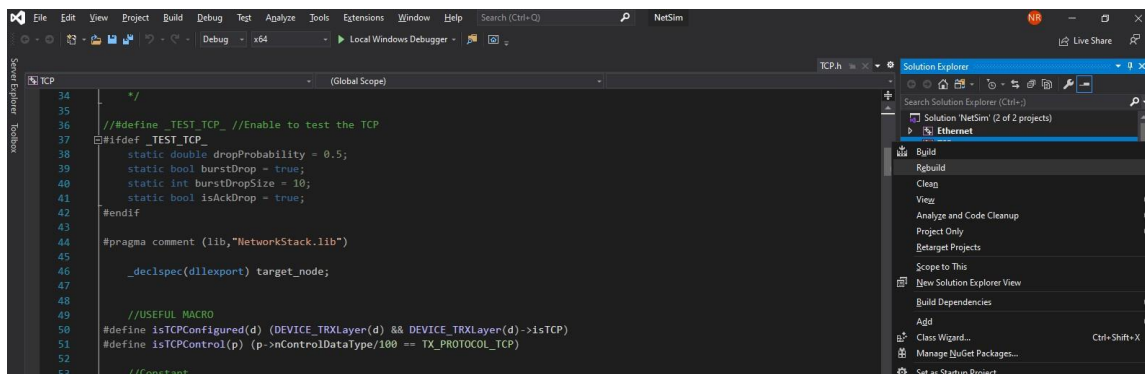
C functions for the SYN_FLOOD attack

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)

Steps to simulate the attack

1. Open the Source codes in Visual Studio by going to Your work-> Source code option and Clicking on Open code in NetSim Home Screen window.
2. In Visual Studio under the **TCP** project in the solution explorer you will be able to see that **SYN_FLOOD.c** file.
3. Right click on the solution in the solution explorer and select Rebuild (Note: first rebuild the TCP project and then rebuild the Ethernet project)



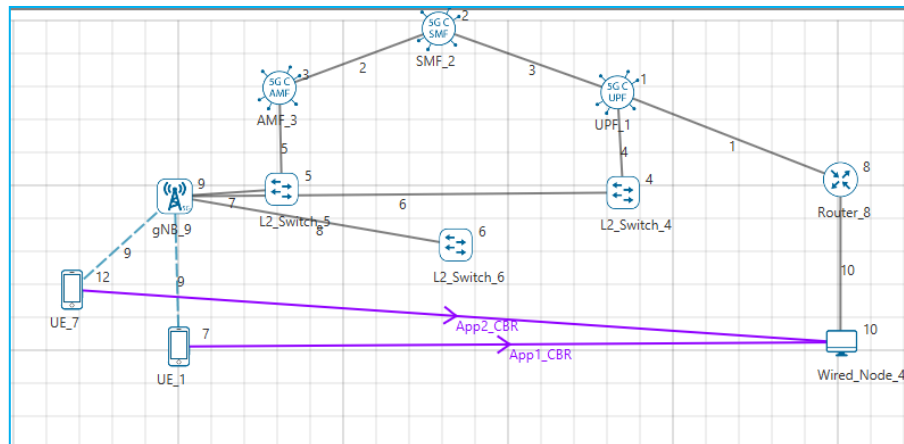
4. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries.

Running Simulations.

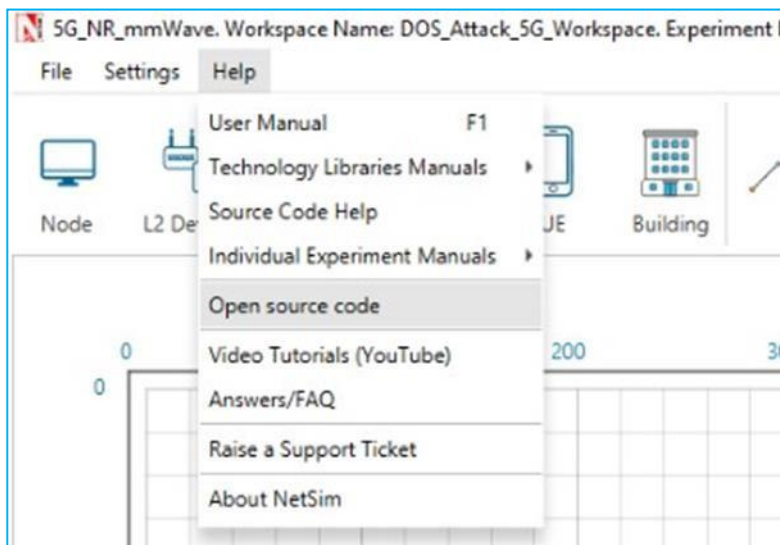
Case-1: Without Malicious Node

1. Then DOS_Attack_5G_Workspace comes with a sample configuration that is already saved. To open this example, go to Your work and click on the DOS_Attack_Case_1 that is present under the list of experiments

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



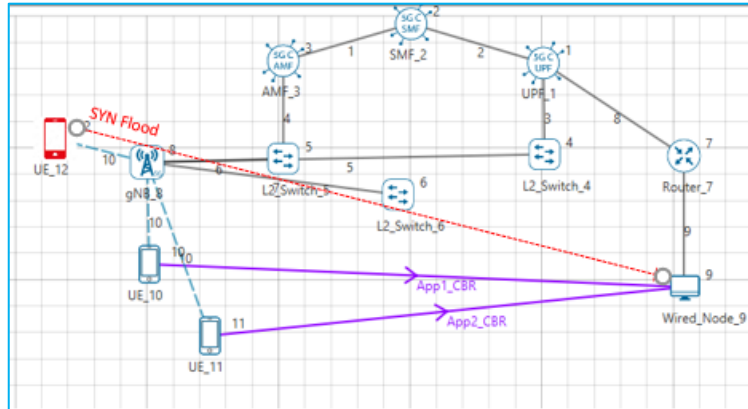
- Help ☐ Open Source code



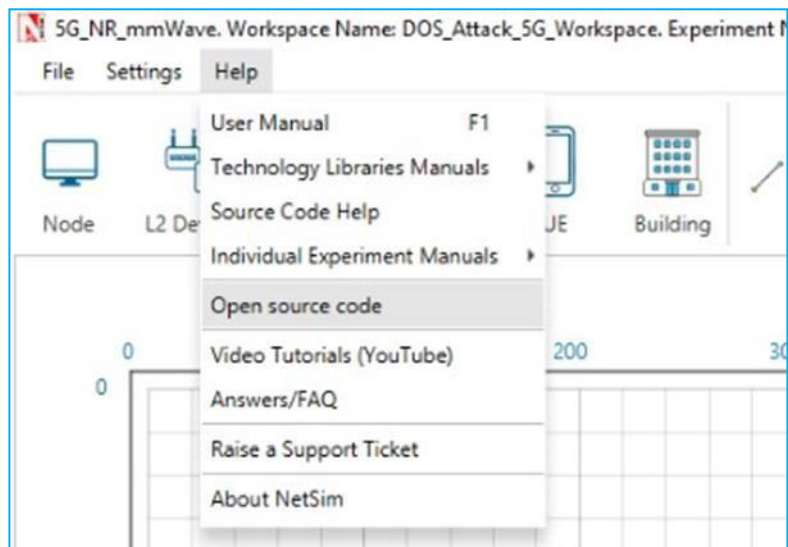
- In TCP.h set **NUMBEROFMALICIOUSNODE** as 1.
- In SYN_FLOOD.c set **malicious node** as 0.
- Right click on the solution in the solution explorer and select Rebuild. (Note: first rebuild the TCP project and then rebuild the Ethernet project)
- Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries.
- Run the simulation for 5 seconds.

Case-2: With one Malicious Node

- Then DOS_Attack_5G_Workspace comes with a sample configuration that is already saved. To open this example, go to Your work and click on the DOS_Attack_Case_2 that is present under the list of experiments.
- The saved network scenario consisting of 5G Core, 3 UEs, 1 gNB, 1 Router and 1 wired node in the grid environment forming a 5G NR Network. Traffic is configured from UE to Wired node.



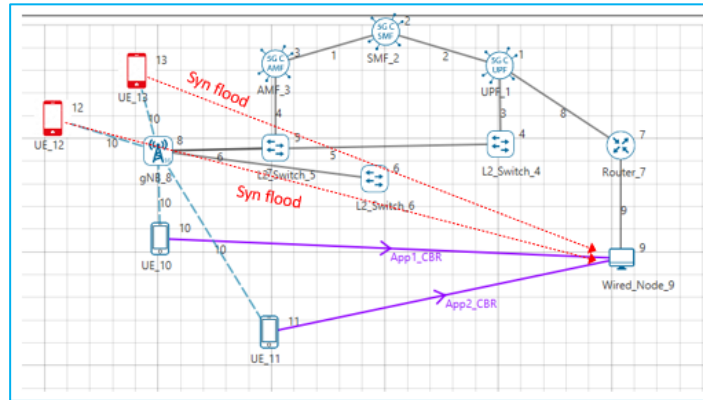
3. Help ☐ Open Source code



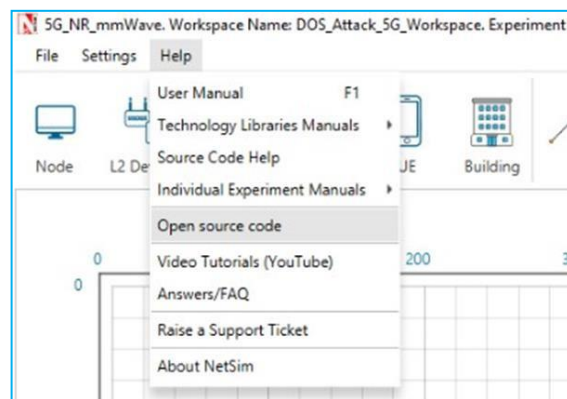
4. In TCP.h set **NUMBEROFMALICIOUSNODE** as 1.
5. In SYN_FLOOD.c set **malicious node** as 12.
6. Right click on the solution in the solution explorer and select Rebuild. (Note: first rebuild the TCP project and then rebuild the Ethernet project)
7. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries.
8. 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 Your work and click on the DOS_Attack_Case_3 that is present under the list of experiments :
2. The saved network scenario consisting of 5G Core, 4 UEs, 1 gNB, 1 Router and 1 wired node in the grid environment forming a 5G NR Network. Traffic is configured from UE to Wired node.



3. Help ☐ Open Source code



4. In TCP.h set **NUMBEROFMALICIOUSNODE** as 2.
5. In SYN_FLOOD.c set **malicious node** as 12, 13.
6. Right click on the solution in the solution explorer and select Rebuild. (Note: first rebuild the TCP project and then rebuild the Ethernet project)
7. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries.
8. Run the simulation for 5 seconds.

Results and discussion

After simulation, open metrics window and observe the Throughput.

Simulation Results

Application Metrics Table

Application ID	Application Name	Packets Generated	Packets Received	Throughput (Mbps)
1	App1_CBR	10289	2164	5.588504
2	App2_CBR	10289	2169	5.594175

TCP Metrics Table

Source	Destination	Segment Sent	Segment Received	Ack Sent	Ack Received
UE_12	ANY_DEVICE	0	0	0	0
SMF_2	ANY_DEVICE	0	0	0	0
AMF_3	ANY_DEVICE	0	0	0	0
ROUTER_7	ANY_DEVICE	0	0	0	0
WIRELESS_NODE_9	ANY_DEVICE	0	0	0	0
UE_10	ANY_DEVICE	0	0	0	0
UE_11	ANY_DEVICE	0	0	0	0
UE_12	ANY_DEVICE	0	0	0	0

Link Metrics Table

Link ID	Link Throughput (Mbps)	Packets Transmitted	Packets Received	Packets Collected
AB	NA	25625	25625	0
1	NA	0	0	0
2	NA	0	0	0
3	NA	1111	1111	0
4	NA	0	0	0
5	NA	1111	1111	0
6	NA	0	0	0
7	NA	0	0	0
8	NA	1111	1111	0
9	NA	1111	1111	0

Queue Metrics Table

Device ID	Port ID	Queue ID	Queue Size	Queue Length	Queue Delay	Queue Drop
No content in table						

Go to the result window open Event 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
78	1	1000	UE	6	0	0	0	0	TCP	SYN_FLOOD
109	97	2000	UE	6	0	0	0	0	TCP	SYN_FLOOD
132	129	3000	UE	6	0	0	0	0	TCP	SYN_FLOOD
173	152	4000	UE	6	0	0	0	0	TCP	SYN_FLOOD
275	195	5000	UE	6	0	0	0	0	TCP	SYN_FLOOD
316	295	6000	UE	6	0	0	0	0	TCP	SYN_FLOOD
418	338	7000	UE	6	0	0	0	0	TCP	SYN_FLOOD
463	438	8000	UE	6	0	0	0	0	TCP	SYN_FLOOD
607	485	9000	UE	6	0	0	0	0	TCP	SYN_FLOOD
652	627	10000	UE	6	0	0	0	0	TCP	SYN_FLOOD
771	674	11000	UE	6	0	0	0	0	TCP	SYN_FLOOD
816	791	12000	UE	6	0	0	0	0	TCP	SYN_FLOOD
960	838	13000	UE	6	0	0	0	0	TCP	SYN_FLOOD
1003	980	14000	UE	6	0	0	0	0	TCP	SYN_FLOOD
1147	1025	15000	UE	6	0	0	0	0	TCP	SYN_FLOOD
1190	1167	16000	UE	6	0	0	0	0	TCP	SYN_FLOOD
1315	1212	17000	UE	6	0	0	0	0	TCP	SYN_FLOOD
1366	1335	18000	UE	6	0	0	0	0	TCP	SYN_FLOOD
1510	1388	19000	UE	6	0	0	0	0	TCP	SYN_FLOOD
1555	1530	20000	UE	6	0	0	0	0	TCP	SYN_FLOOD
1699	1577	21000	UE	6	0	0	0	0	TCP	SYN_FLOOD
1743	1739	22000	UE	6	0	0	0	0	TCP	SYN_FLOOD

Case 1 shows the results when there is no attack. The two user applications, attain a throughput of about 6.25 Mbps. In the table we see the throughput of for these two applications falling as we increase the number of attack nodes. This is because the server's resources are being used up in handling the SYN-FLOOD packets and the server is unable to sustain packet transmissions for the regular applications. In this example, with a coordinated attack involving 4 systems the throughputs are down 70%.

	Throughput_APP1(Mbps)	Throughput_APP2(Mbps)
Case-1: Malicious Node =0	6.25	6.27
Case-2: Malicious Node =1	5.99	5.99
Case-3: Malicious Node =2	5.71	5.73

Table 1: Throughputs seen by the user applications. The first row is the throughput when there is no attack. In other samples show the fall in throughputs as the number of attacker systems are increased

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

Appendix: NetSim source code modifications

Changes to fn_NetSim_TCP_Trace(), in TCP.c file, within TCP project

/* This is used to add the SYN_FLOOD sub-events in Event Trace file */

_declspec (dllexport) char *fn_NetSim_TCP_Trace(int nSubEvent)

{

```

if (nSubEvent == SYN_FLOOD)
return "SYN_FLOOD";
return (GetStringTCP_Subevent(nSubEvent));
}

```

Changes to fn_NetSim_TCP_HandleTimer(), in TCP.c file, within TCP project

```

/* This is used to call the syn_flood() function periodically */
static int fn_NetSim_TCP_HandleTimer()
{
switch (pstruEventDetails->nSubEventType)
{
case SYN_FLOOD:
syn_flood();
break;
case TCP_RTO_TIMEOUT:
handle_rto_timer();
break;

```

Changes to fn_NetSim_TCP_Init(), in TCP.c file, within TCP project

```

/* This is used to register the first SYN_FLOOD event */
_declspec (dllexport) int fn_NetSim_TCP_Init(struct stru_NetSim_Network
*NETWORK_Formal,
NetSim_EVENTDETAILS *pstruEventDetails_Formal,
char *pszAppPath_Formal,
char *pszWritePath_Formal,
int nVersion_Type,
void **fnPointer)
{
fn_NetSim_TCP_Init_F(NETWORK_Formal,
pstruEventDetails_Formal,
pszAppPath_Formal,
pszWritePath_Formal,
nVersion_Type,
fnPointer);
NetSim_EVENTDETAILS pevent;
memcpy(&pevent, pstruEventDetails, sizeof pevent);
for (int i = 0; i < NETWORK->nDeviceCount; i++)
{
if (is_malicious_node(i + 1))

```



```

{
pevent.nDeviceId = i + 1;
pevent.dEventTime += 1000;
pevent.nEventType = TIMER_EVENT;
pevent.nSubEventType = SYN_FLOOD;
pevent.nProtocolId = TX_PROTOCOL_TCP;
fnpAddEvent(&pevent);
}
}
return 0;
}

```

Changes to add_timeout_event() in RTO.c file, within TCP project

```

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

```

```
}  
}
```

Changes to TCP.h file, within TCP project

```
/* This is used to define the number of malicious nodes */  
#pragma comment (lib,"NetworkStack.lib")  
_declspec(dllexport) target_node;  
//USEFUL MACRO  
#define isTCPConfigured(d) (DEVICE_TRXLayer(d) && DEVICE_TRXLayer(d)->isTCP)  
#define isTCPControl(p) (p->nControlDataType/100 == TX_PROTOCOL_TCP)
```

```
//Constant  
#define TCP_DupThresh 3  
#define NUMBEROFMALICIOUSNODE 2  
Addition of SYN_flood.c file, within TCP project  
/* This is used to define the malicious node ID's and the target node ID */  
/* This has functions defined for SYN flood attack*/  
#include "main.h"  
#include "TCP.h"  
#include "List.h"  
#include "TCP_Header.h"  
#include "TCP_Enum.h"  
int malicious_node[NUMBEROFMALICIOUSNODE] = {2,6};  
static void send_syn_packet(PNETSIM_SOCKET s);  
//static PNETSIM_SOCKET socket_creation();  
int target_node = 4;  
PNETSIM_SOCKET get_Remotesocket(NETSIM_ID d, P SOCKETADDRESS addr);  
static P SOCKETADDRESS sockAddr = NULL;  
int is_malicious_node(NETSIM_ID devid){}  
void syn_flood(){}  
static void send_syn_packet(PNETSIM_SOCKET s){}  
int socket_creation(){}  

```

Changes to TCP_Enum.h file, within TCP project

```
/* This is used to a new SYN_FLOOD subevent in TCP_Subevent */  
#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),
}

```

Changes to Ethernet.h file, within ETHERNET project

```

/* This is used to define processing time for syn_flood packets */
#ifndef _NETSIM_ETHERNET_H_
#define _NETSIM_ETHERNET_H_
#ifdef __cplusplus
extern "C" {
#endif

```

```

#pragma comment(lib, "NetworkStack.lib")
#pragma comment(lib, "Metrics.lib")
#pragma comment (lib, "libTCP.lib")
#define isETHConfigured(d,i) (DEVICE_MACLAYER(d,i)->nMacProtocolId ==
MAC_PROTOCOL_IEEE802_3)
//Global variable
PNETSIM_MACADDRESS multicastSPTMAC;
#define ETH_IFG 0.960 //Micro sec
#define Processing_TIME 1000

```

Changes to fn_NetSim_Ethernet_HandlePhyOut() in Ethernet_Phy.c file, within ETHERNET project

```

/* This is used to add processing delay for TCP SYN packets */
/* This is used to add processing delay for TCP SYN packets */
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;
}

```

TCP Project Properties:

- Right click on TCP project and select Properties.
- In Linker section go to Advanced

The import library value has been updated for 64-bit source code settings.

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

