ENSC 427: COMMUNICATION NETWORKS
Spring 2021

Final Project Report

Analysis of 5G Network under DDoS Attack Using ns-3 Simulator

http://www.sfu.ca/~wailongt/ENSC427/Index.html

Namsakhi Kumar – namsakhi_kumar@sfu.ca – 301336727
Wai Long (Edwin) Tam – wailongt@sfu.ca – 301264849
Kwok Liang Lee – kwokl@sfu.ca – 301326695

TEAM 10

Abstract

With the fifth generation (5G) network becoming popular to the public due to the facts that it has a higher bitrate and wider bandwidth that would allow high-speed communication, it favors 5G network to replace the older generations, and it allows more devices, such as autonomous vehicles and health monitors to be implemented into 5G technology in the future. However, the security on 5G networks to autonomous devices and monitors are important, as a split second of disconnection will contribute to accident or misstep. If the network is under Distributed-Denial-of-Service (DDoS) attack, the device may lose connection due to functional disruption of the server by overflooding it with excessive requests, resulting in traffic overflow which blocks bandwidth capacity. In this paper, we will discuss local Device-to-Device (D2D) communication rather than global D2D. To understand the performance of 5G networks under DDoS attack, we simulate the network model on mmWave in ns-3 simulator and use Wireshark to obtain the data flow of traffic in the network. Finally, the traffic animation is displayed with NETAnim for better visualization.

Table of Contents

Glossary

5G - Fifth Generation Technology
DDoS - Distributed Denial of Service
ICMP - Internet Control Message Protocol
LTE - Long Term Evolution
MAC - Media Access Control
MME - Mobility Management Entity
PDCP - Packet Data Convergence Protocol
PHY - Physical layer
PGW - Packet Data Network Gateway
RLC - Radio Link Control
RRC - Radio Resource Control
SGW - Service Gateway
TCP - Transmission Control Protocol
UDP - User Datagram Protocol
Ue - User Equipment

List of figures

List of Tables

Introduction

In this project, the goal is to simulate Distributed Denial of Service (DDoS) attack on an end-to-end simulation scenario in a 5G network using mmWave module in ns-3. The mmWave module [1] is an open source project, interfaced with Long Term Evolution (LTE) module, named as LTE Evolved Packet Core Network Simulator (LENA) [2].

The structure that is used in mmWave in this project included components of Service Gateway (SGW) / Packet Gateway (PGW) that provides internet connectivity, an access point, clients, and a remote host that will connect to SGW/PGW to generate IP packets [3]. The transport protocol that has been used by the attackers is User Datagram Protocol (UDP) to provide a fast connectionless oriented service. The type of Distributed Denial of Service (DDoS) attack simulated in this project is UDP flood. Simply put, UDP flood is a type of cyber attack that sends a huge amount of UDP Packets in a small fragment of time [4] as illustrated in Figure 1 below [5].

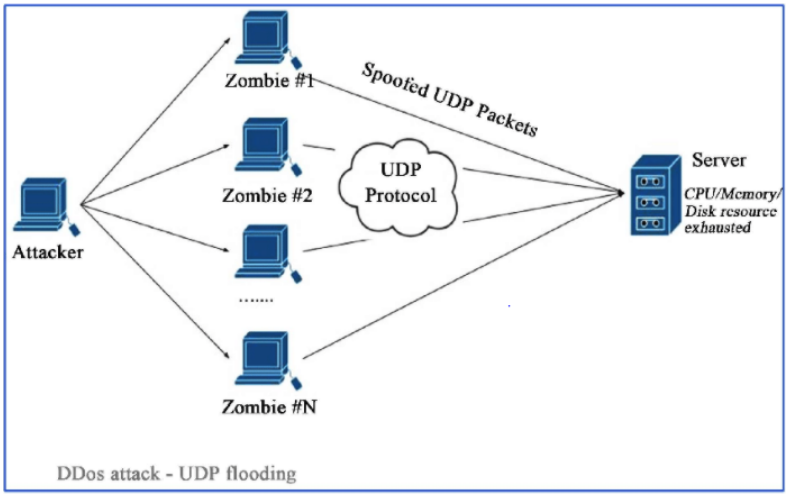


Figure 1: UDP Flood Attack [5]

DDoS Attack

DDoS attacks are categorized into three major types namely volume based attacks, protocol attacks, and application layer attacks. In volume based attacks, the goal of the attacker is to generate massive traffic to completely saturate the bandwidth, making it impossible for traffic to flow between legitimate users and servers [6]. In protocol attacks, the goal of the attacker is to consume the server's resources and eat up the processing capacity of network infrastructure resources like firewalls and load balancers. In application layer attacks, the goal of the attacker is to crash the web server, targeting web applications. Furthermore, as described in [4], DDoS attacks are broadly classified into two methods, direct and reflector. As the name suggests, a direct attack is implemented by sending a huge amount of packets from attacks to the victim [4]. The type of packets can be of type Transmission Control Protocol (TCP), Internet Control Message Protocol (ICMP), UDP, and a mixture of all three [4]. Whereas for reflector attack, the attack is performed with the usage of routers, called reflectors [4]. Attackers will use spoofed addresses to send reflectors to the packets that cause them to send replies to the victim [4].

In our project, we have implemented the UDP flood attack which belongs to the category of volume based attacks and has a direct method for analysis of 5G network under DDoS attack.

Related Works Overview

The ns-3 network simulator is enhanced with several modules and each of these modules is further categorized into subfolders, named as documentation, examples, tests, src (contains C++ scripts), build, helpers (provide IP addresses and ensures connectivity with various class objects) and scratch. Figure 2 below shows the architectural implementation of ns-3

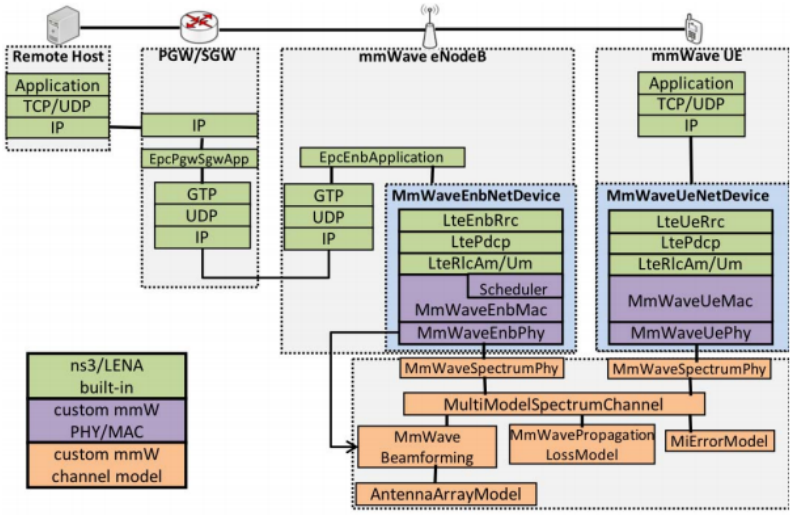


Figure 2: Diagram of mmWave module [2]

mmWave module. As shown in Figure 2 and also mentioned earlier, that mmWave module sits on top of ns-3 LTE module which means that mmWave module itself covers all the detailed implementation of LTE protocols. Thus the user does not need to explicitly add the LTE module while simulating scenarios within mmWave. Furthermore, mmWave also provides its own PHY layer, MAC layer, and custom channel model. The MmWaveEnbNetDevice and MmWaveUeNetDevice are two major parent classes within the mmWave module, used for implementation of eNodeB and ueNodes, respectively. Each of these parent classes contains a child class which are the MmWaveEnbMac in MmWaveEnbNetDevice parent class and MmWaveUeMac in MmWaveUeNetDevice parent class. These child classes support LTE module services including LTE protocol stack (RRC, PDCP, RLC, MAC, PHY) [7]. The MmWavePhy classes, MmWaveEnbPhy and MmWaveUePhy, control the messages of downlink and uplink control channels and handle data-transfer services offered by RLC and MAC layers [8]. The MmWaveSpectrumPhy, an MmWavePhy child class, supports downlink and uplink control channels and also encapsulates various PHY models as objects for MmWaveMiErrorModel, MmWavePropogationLossModel, and MmWaveBeamforming.

The LTE-EPC data plane protocol stack depicted in Figure 3 below is used for LTE in ns-3. It provides the means to simulate an end-to-end data path over the LTE Model that is used in mmWave. Two different layers of IP networking are defined in the EPC model [8]. The first

one involves the connection of eNodeB and SGW/PGW nodes through a set of point-to-point links (S1-U) [8]. The second IP networking is the end-to-end connectivity for SGW/PGW and the remote host (S5) [8]. IP packets are generated at the remote host and addressed to netdevice at the SGW/PGW [8]. Afterward, send to the eNodeB through the S1-U interface and finally to ueNode over the LTE Radio [8].

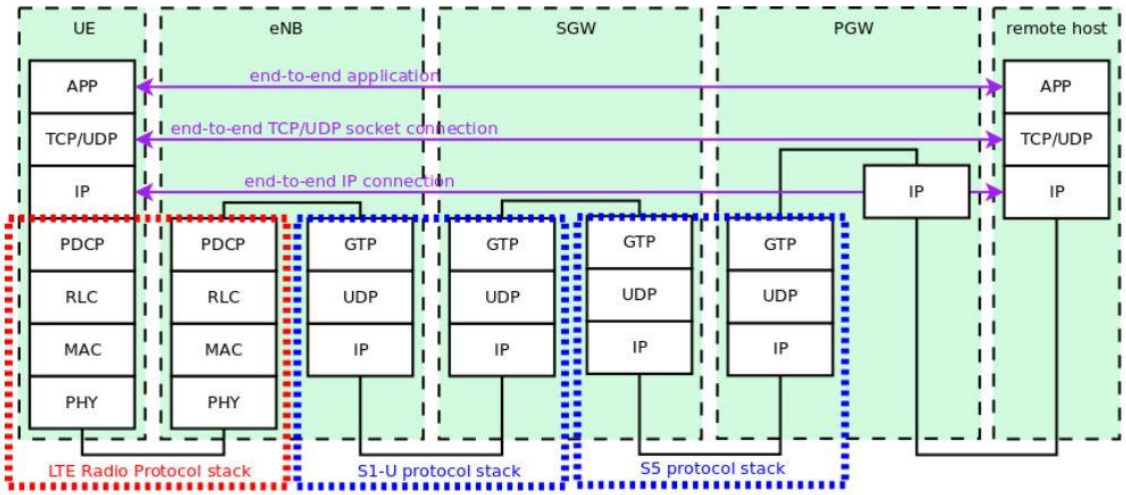


Figure 3: LTE-EPC model [8]

Main Section

Topology

For the implementation of our topology, we have used Figure 4 as a reference, showing DDoS attack on wired topology with 10 attackers, one legitimate user, eNodeB, and one target. In our topology, there is one user equipment (Ue) who is the legitimate user, several attackers (10 -15), one base station (eNodeB), which provides access for users to the network and the logical descendant of the 2G base station as well as the 3G Radio Network Controller. One packet gateway (PGW) provides a gateway through the cellular network and allocates IP addresses to the user, and one mobility management entity (MME) provides device authentication and user mobility management service such as keeping track of user's location. Finally, one remote host (server) that receives TCP/UDP packets and sends ACKS to TCP users. The code (kindly refer to Appendix 1 for the project's code) for the network topology based on the provided examples in mmWave module [1] and the wired DDoS topology from [9].

Throughout all the scenarios, we set the user date rate to 0.5 Gbps, packet size to 1000 bytes for the user and each attacker. In order to saturate the data link between PGW and a remote host with the packets sent by attackers, attacker data rates have been varied in all the scenarios depending on the link capacity. For simulation of each scenario, the simulation time is set to 1.5 seconds and the attack is triggered at 0.4 seconds. The overview of the network layout is shown in Figure 5 below.

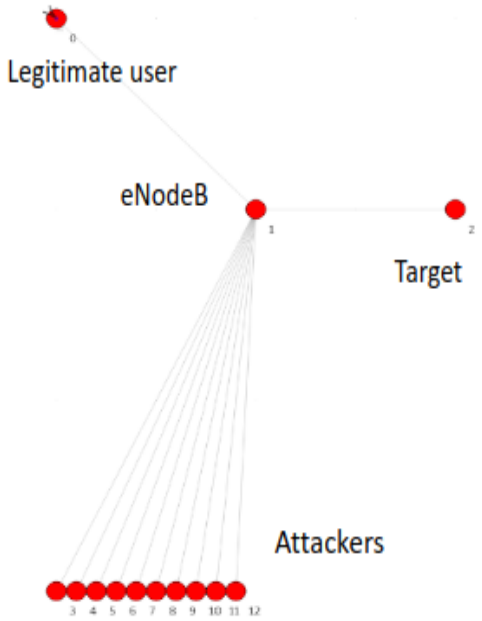


Figure 4: DDoS attack on wired topology [9]

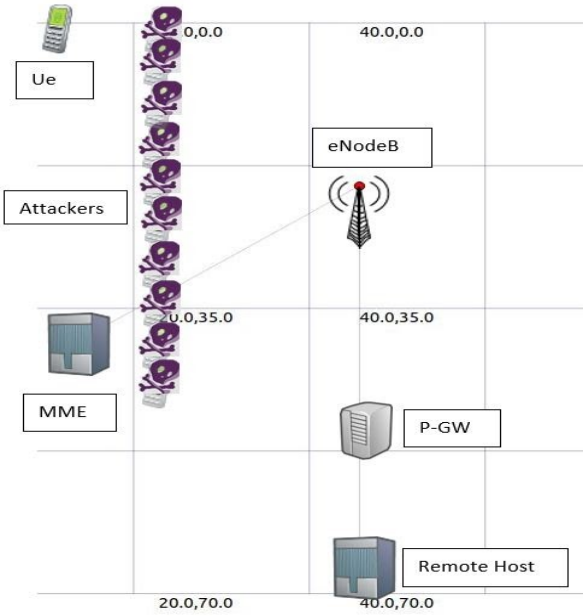


Figure 5: Topology for DDoS attack on 5G network

NetAnim

After the simulation is completed, the packet data is displayed on NetAnim, a software to visualize the dynamics of packets in the network. The NetAnim animation illustrates the topology of the network, same as shown in Figure 5, and the packet travels in the data link will be shown in arrows. However, the arrow can only be shown in the wired link, and wireless transmission will not be displayed. In the normal scenario, for example, the Ue node sends its TCP packets to the nearest eNodeB node, and the packet will then be transmitted along with the wired network to the server via PGW node. Since there is no attacker in the network to interrupt the transmission between the user and server, the user will have the full bandwidth to send packets, as shown in Figure 6. After the DDoS attack happens in the network, shown in Figure 7, the attackers in the network flood the data link between eNodeB node and PGW as well as the link between PGW and the remote host.

Simulation

Before starting any test cases, we verified our network topology in two ways. First is the normal scenario which doesn't have any attacker and the second one with DDoS parameters found in [10].

Figure 7 shows the NetAnim result under attack where the attackers flood the links in the network with UDP packets. Both the TCP and UDP packets are 500 bytes each but are sent to the remote host with different send rates. The user will send the TCP packets at 0.1 Mbps while attackers send the UDP packets are 1 Mbps. All the network links have 1 Mbps bandwidth and generate constant bit rate traffic. The I/O graph from Wireshark generated using the ns-3 pcap file is displayed in Figure 8.

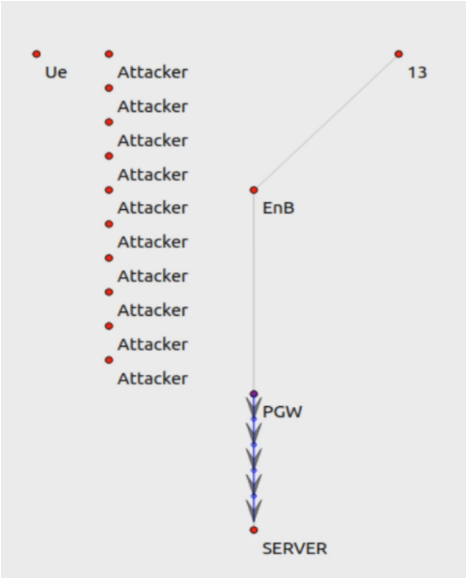


Figure 6: NetAnim Image of normal scenario

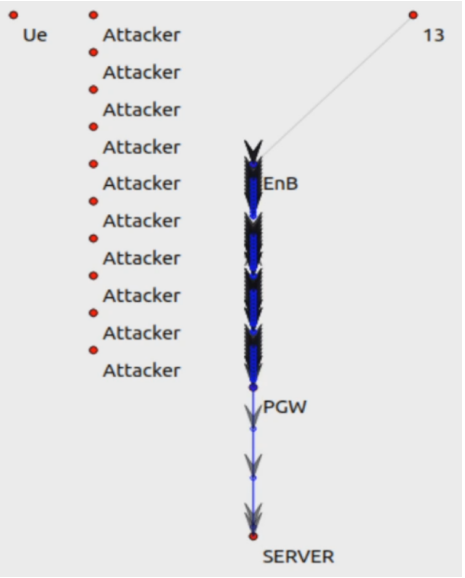


Figure 7: NetAnim Image of DDoS scenario

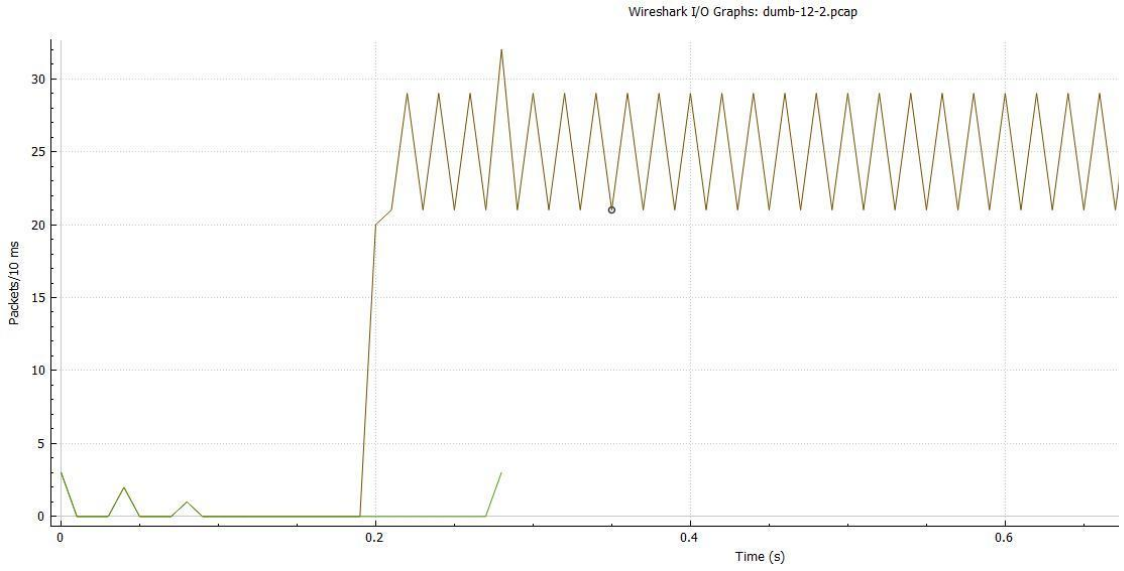


Figure 8: Test result with parameters from [10] with 1.5 sec of simulation time

The green line in Figure 8 depicts the packets sent from the user to the remote host while packets from attackers are plotted in the brown line. As we can see, the attackers started to attack approximately at 0.2 sec. After 0.3 sec, the transmission of the packets between the remote host and the user stopped. This proved that our DDoS implementation works. Another thing to mention from this I/O graph is the peak at 0.3 sec. This is due to the TCP and UDP packets being sent to the same port at the remote host. Therefore, the packet number shown at the peak is the total sum of both TCP and UDP packets.

Test cases

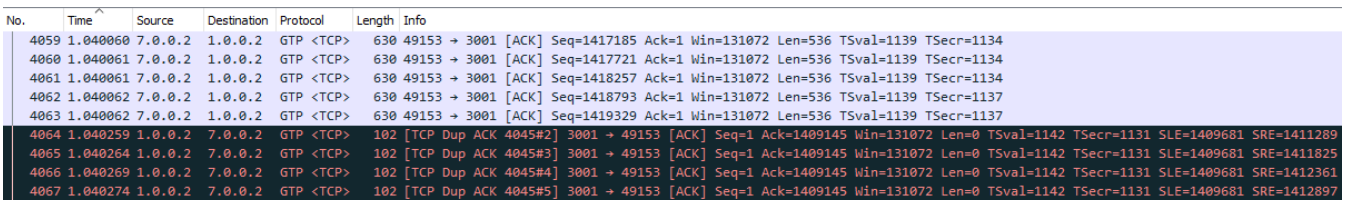
In this project, along with the normal case, other five scenarios are tested as well. In the normal case, a user's normal transmission routine is set up, which will have no attackers in the network, and this will be the control among all other scenarios. The first scenario has a low transmission rate at 100 Mbps and has 10 attackers enabled in the network, so to prove the attacker system is functional. The second and third scenario has 10 attackers each with 250 Mbps and 500 Mbps data rate in the links, respectively. These two scenarios are simulating the sub-5G network and 5G speed in the real world scenario [11]. These first three scenarios will have the user with sending rate at 0.5 Gbps and the attacker with sending rate at 0.5 Gbps. However, the fourth case is using an ideal 5G data rate which is 1 Gbps, and 15 attackers with the attack rate at 1.5 Gbps, such that the attackers can be able to flood the data link with packets. Furthermore, the last scenario tests the behaviour of the 5G network under more attackers and high attacker sending rate making the attackers more powerful. Similarly, the user data rate in the last two cases remains the same as in other scenarios such that the results from all cases can be compared at the end. Finally, all the tests are simulated for 1.5 simulation seconds, and the attack is set up to be triggered 0.4 seconds after the simulation is started.

<table>
<tr><th>Scenarios</th><th>User Data Rate</th><th>Attacker Data Rate</th><th># of Attackers</th></tr>
<tr><td>Control</td><td rowspan="5">0.5 Gbps</td><td rowspan="3">0.5 Gbps</td><td rowspan="3">10</td></tr>
<tr><td>1</td></tr>
<tr><td>2</td></tr>
<tr><td>3</td><td rowspan="3"></td><td rowspan="3"></td></tr>
<tr><td>4</td><td>1.5 Gbps</td><td>15</td></tr>
<tr><td>5</td><td>5 Gbps</td><td>20</td></tr>
</table>

Table 1: Parameters of All Scenarios

Results

In the control (normal) case, the result shows that the user has no interruption in transmitting its packets to the server (Figure 10). However, the user sometimes experiences the packet drop due to retransmission requests from the server and a series of TCP ACK duplications (Figure 9). As a result, the data rate will drop drastically and then gradually increase over time.



No.	Time	Source	Destination	Protocol	Length	Info
4059	1.040060	7.0.0.2	1.0.0.2	GTP <TCP>	630	49153 → 3001 [ACK] Seq=1417185 Ack=1 Win=131072 Len=536 TSval=1139 TSecr=1134
4060	1.040061	7.0.0.2	1.0.0.2	GTP <TCP>	630	49153 → 3001 [ACK] Seq=1417721 Ack=1 Win=131072 Len=536 TSval=1139 TSecr=1134
4061	1.040061	7.0.0.2	1.0.0.2	GTP <TCP>	630	49153 → 3001 [ACK] Seq=1418257 Ack=1 Win=131072 Len=536 TSval=1139 TSecr=1134
4062	1.040062	7.0.0.2	1.0.0.2	GTP <TCP>	630	49153 → 3001 [ACK] Seq=1418793 Ack=1 Win=131072 Len=536 TSval=1139 TSecr=1137
4063	1.040062	7.0.0.2	1.0.0.2	GTP <TCP>	630	49153 → 3001 [ACK] Seq=1419329 Ack=1 Win=131072 Len=536 TSval=1139 TSecr=1137
4064	1.040259	1.0.0.2	7.0.0.2	GTP <TCP>	102	[TCP Dup ACK 4045#2] 3001 → 49153 [ACK] Seq=1 Ack=1409145 Win=131072 Len=0 TSval=1142 TSecr=1131 SLE=1409681 SRE=1411289
4065	1.040264	1.0.0.2	7.0.0.2	GTP <TCP>	102	[TCP Dup ACK 4045#3] 3001 → 49153 [ACK] Seq=1 Ack=1409145 Win=131072 Len=0 TSval=1142 TSecr=1131 SLE=1409681 SRE=1411825
4066	1.040269	1.0.0.2	7.0.0.2	GTP <TCP>	102	[TCP Dup ACK 4045#4] 3001 → 49153 [ACK] Seq=1 Ack=1409145 Win=131072 Len=0 TSval=1142 TSecr=1131 SLE=1409681 SRE=1412361
4067	1.040274	1.0.0.2	7.0.0.2	GTP <TCP>	102	[TCP Dup ACK 4045#5] 3001 → 49153 [ACK] Seq=1 Ack=1409145 Win=131072 Len=0 TSval=1142 TSecr=1131 SLE=1409681 SRE=1412897

Figure 9: WireShark Packet Details of Normal Scenario

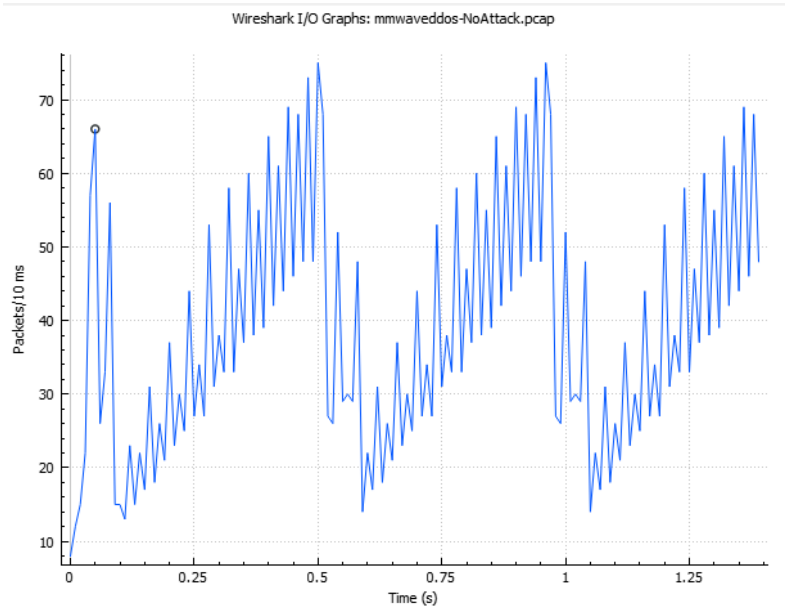


Figure 10: WireShark Image of the Normal Scenario

In the next test case, shown in Figure 11, the legitimate user (labeled in blue line) sends the packets like in the normal case, but then the DDoS attack will be introduced to the network at 0.4 seconds. After the attack occurs (displayed in multiple colors), the data rate of the user drops drastically from 200 packets per 10 milliseconds to about 10 to 20 packets per 10 milliseconds. During the attack, all the attackers generate about 12 to 15 packets per 10 milliseconds. That is about 125 packets per 10 milliseconds in total. After 0.75 seconds, the blue line of the user data rate has ceased, which means the server has not received any packets that are sent out by the user from 0.75 seconds and onward. The second case, in Figure 12 below, is similar to the previous case except the link data rate is increased to 250 Mbps, and the effect of the DDoS on the user is still significant that can completely stop the user transmission and terminate the TCP transmission in 0.7 seconds. Both of the scenarios show DDoS attack can disable the user from accessing the server in the low speed network, but how about in a high speed network like 5G?

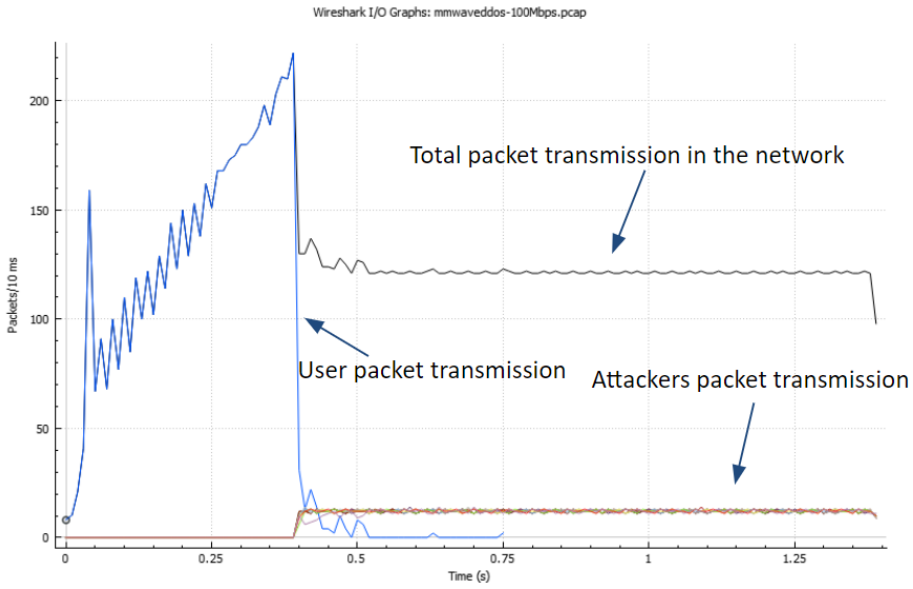


Figure 11: WireShark I/O Graph of Scenario 1

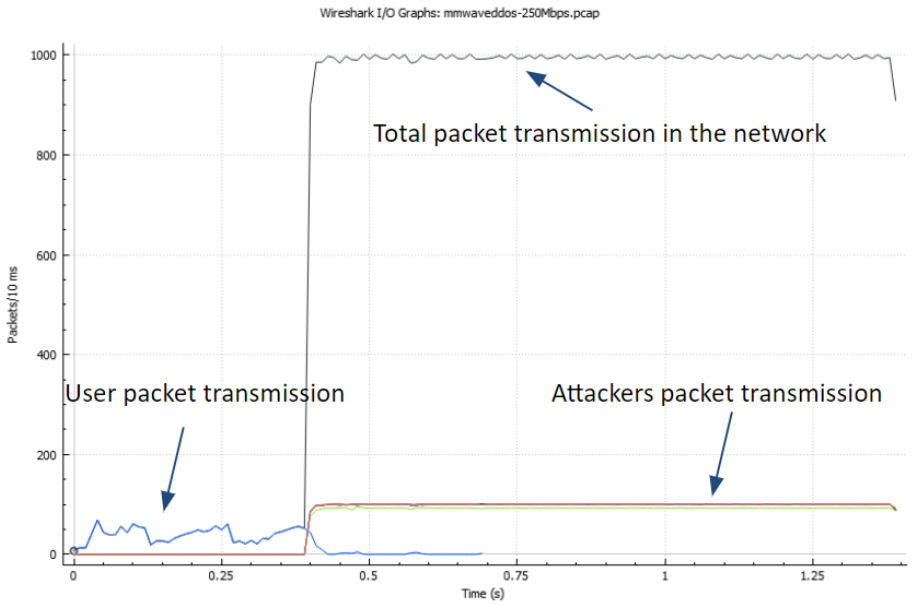


Figure 12: WireShark I/O Graph of Scenario 2

The third and fourth scenarios replicate the average real-world 5G speed and the theoretical 5G speed. In the third scenario, the average real-world 5G speed at 500 Mbps will be simulated. With the 500 Mbps data, shown in Figure 13, users are able to send packets with a rate of 10 to 20 packets per 10 milliseconds after the attack occurs at 0.4 seconds, but the transmission ends at nearly 1 second. This shows that the model of the DDoS attack would still be effective in a real-world 5G network. In an ideal 5G network, shown in scenario 4 and Figure 14, the transmission bandwidth is now extended to 1 Gbps which would allow more data to transmit in the data link. To encounter the higher allowable transmission rate, the attackers would have to increase the packet send rate and the number of attackers would have to increase to 15 in order to increase the traffic in the link. Since the attack, the legitimate user has only a minor effect on the transmission before the data rate returns to the normal routine pattern. This shows this DDoS attack setting has no effect on the user.

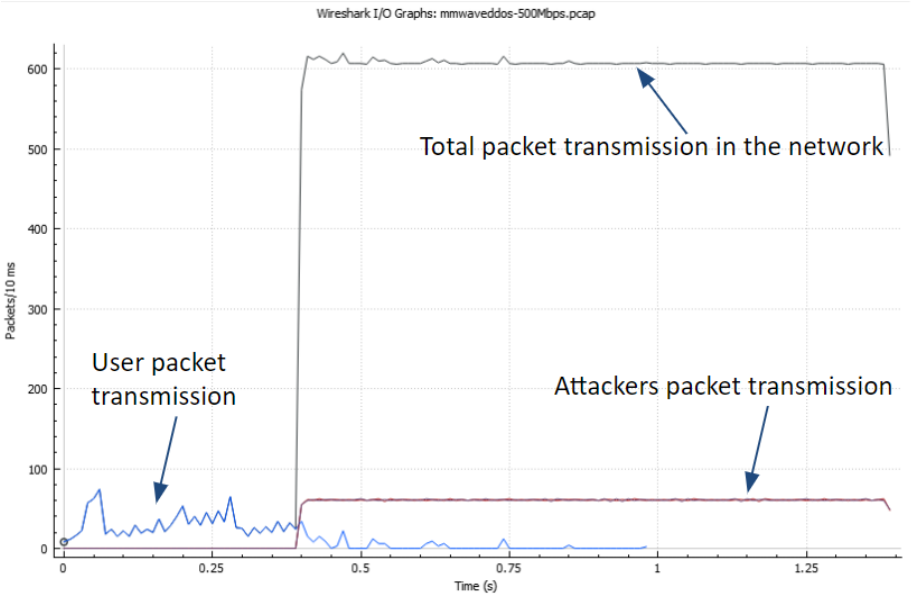


Figure 13: WireShark I/O Graph of Scenario 3

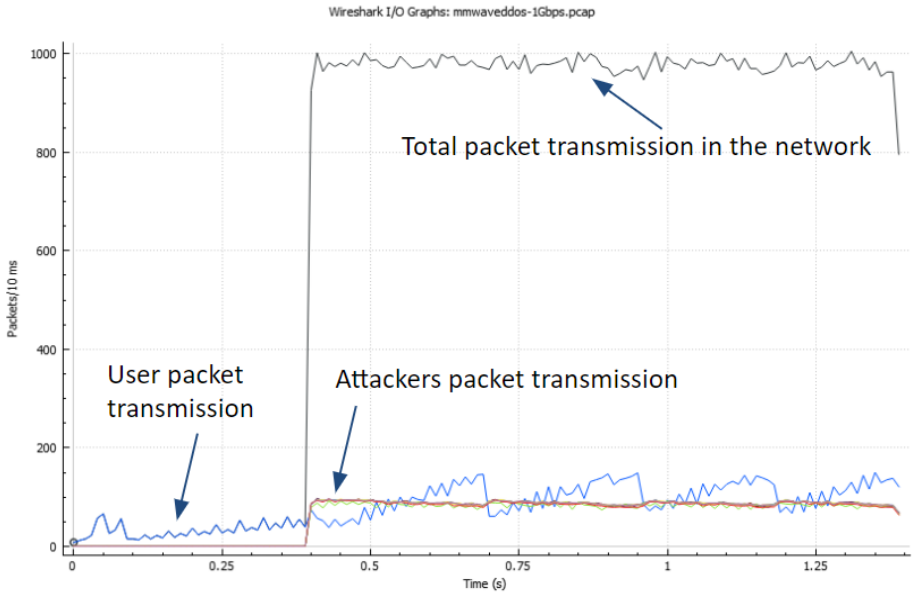


Figure 14: WireShark I/O Graph of Scenario 4

In the last scenario, in Figure 15, the effect of the DDoS attack is increased by setting 20 attackers in the network, and each attacker will have 5 gigabits per second data rate, which will generate about 350 thousand packets per second. At the beginning of the attack, the user data transmission rate has slightly dropped, which can be seen from the first interest point. As the transmission rate of the attacker decreases, the user has some bandwidth to send data to the server, so the user packet received by the server increases. At point 2, the attackers send more packets to the server and make the legitimate user left with less bandwidth to send packets. Hence, the number of packets received by the server from the user decreases. This graph shows our model is possible to create a 5G DDoS attack, but it requires more packets to send from the attackers to the server.

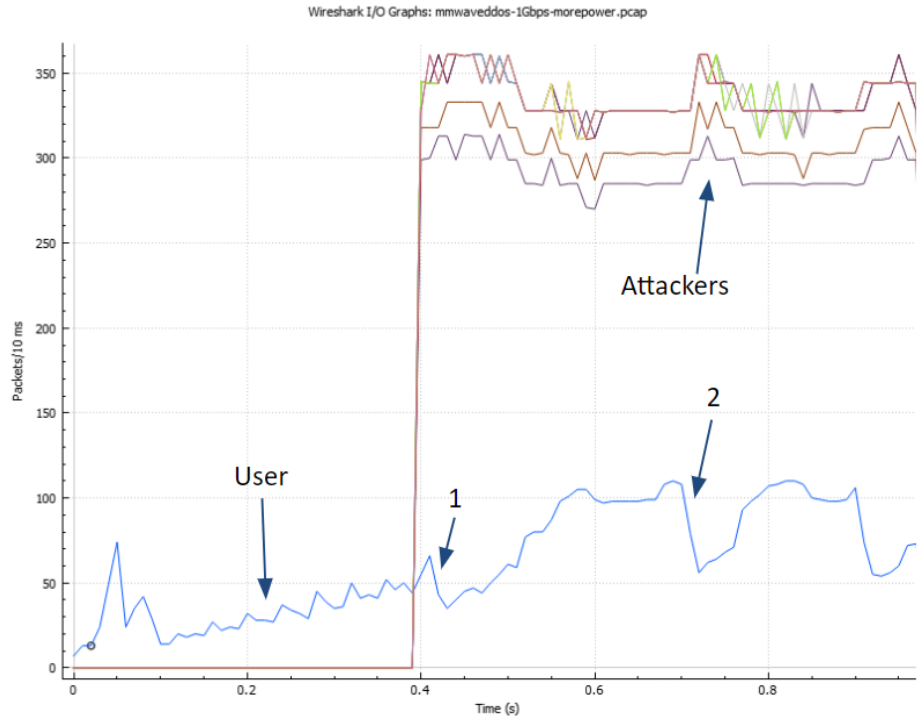


Figure 15: WireShark I/O Graph of Scenario 5

Conclusion

In this project, there are two setups for verification and 5 scenarios for testing the behaviour 5G networks under DDoS attack. Our verifications verify that our network topology works and the attackers successfully occupied the bandwidth available to the user. For the test cases, the first three scenarios with link capacity under sub-5G rate show a significant decrease in user's bandwidth after DDoS attack. In scenarios 4 and 5, the link capacity is set to 1 Gbps. Scenario 4 has 15 attackers while scenario 5 has 20 attackers, and the remaining settings are the same. Compared to the first three scenarios, the DDoS attack didn't decrease the bandwidth of the user significantly in scenarios 4 and 5. This may be possible due to the increase in link capacity which leads to higher tolerance in the number of packets. This shows that it requires the increment of the attacker or the DDoS send rate in order to flood the bandwidth with packets.

For future work, our group wishes to dig deeper into the ns-3 module to have a better understanding of the provided functions, especially functions related to packet generation. In addition, our group will have more scenarios with the link bandwidth set closer to 5G network. Besides this, the number of attackers and users require modification to be more realistic. The exploration and implementation of different types of DDoS attacks and their behaviour should also be considered.

There were many challenges and limitations faced in completing this project. One of the major challenges was the long run time of the simulation. When the attacker send rate, user send rate and link bandwidth are all set to be in Gbps, the simulation can take up to 12 hours to complete. The discontinuity of attackers' packets is another main issue that our group faced. On the other hand, we couldn't figure out the way to show the packet flow between wireless connections in NetAnim. Finally, our group has difficulty in understanding the 5G mmWave framework due to lack of description which led our group to write our code.

References

[1] Nyuwireless-Unipd, “nyuwireless-unipd/ns3-mmwave,” *GitHub*. [Online]. Available: https://github.com/nyuwireless-unipd/ns3-mmwave. [Accessed: 14-Mar-2021].

[2] M. Mezzavilla et al., "End-to-End Simulation of 5G mmWave Networks," in *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2237-2263, thirdquarter 2018, doi: 10.1109/COMST.2018.2828880. [Accessed: 14-Mar-2021].

[3] N. Tan, S. Makina, and M. Zaki, Simulation of DDoS attacks in 4G networks. [Online]. Available: https://sites.google.com/view/ensc427team05/. [Accessed: 14-Mar-2021].

[4] B. Nagpal, P. Sharma, N. Chauhan and A. Panesar, "DDoS tools: Classification, analysis and comparison," *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, India, 2015, pp. 342-346. [Accessed: 14-Mar-2021].

[5] S. Alzahrani and L. Hong, “Generation of DDoS Attack Dataset for Effective IDS Development and Evaluation,” *Journal of Information Security*, vol. 09, no. 04, pp. 225–241, 2018. [Accessed: 14-Mar-2021].

[6] D. Fang, Y. Qian and R. Q. Hu, "Security for 5G Mobile Wireless Networks," in *IEEE Access*, vol. 6, pp. 4850-4874, 2018, doi: 10.1109/ACCESS.2017.2779146. [Accessed: 20-Mar-2021].

[7] “LTE Channels: Logical, Transport and Physical Channels Details and Mapping,” *Techplayon*, 21-Jul-2020. [Online]. Available: http://www.techplayon.com/2411-2/. [Accessed: 15-Mar-2021].

[8] “Design Documentation” *Design Documentation - Model Library*. [Online]. Available: https://www.nsnam.org/docs/models/html/lte-design.html#overview. [Accessed: 15-Mar-2021].

[9] S. U., “DDoS simulation in NS-3 [C++],” *Medium*, 07-Aug-2020. [Online]. Available: https://infosecwriteups.com/ddos-simulation-in-ns-3-c-12f031a7b38c. [Accessed: 18-Apr-2021].

[10] F. Lau, S. H. Rubin, M. H. Smith, and Lj. Trajkovic, ``Distributed denial of service attacks,'' (invited paper) *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics, SMC 2000*, Nashville, TN, Oct. 2000, pp. 2275-2280. [Accessed: 16-Mar-2021].

[11] M. Sung, J. Kim, E.-S. Kim, S.-H. Cho, Y.-J. Won, B.-C. Lim, S.-Y. Pyun, H. Lee, J. K. Lee, and J. H. Lee, “RoF-Based Radio Access Network for 5G Mobile Communication Systems in 28 GHz Millimeter-Wave,” *Journal of Lightwave Technology*, vol. 38, no. 2, pp. 409–420, 2020, doi: 10.1109/JLT.2019.2942636. [Accessed: 16-Mar-2021].

Appendix 1: ns-3 Code

```
/*

 *    (ue)
 *      \
 *       \
 *         ------- (eNodeB) ------(pgw) ------(remote host)/server
 *                 / |  \
 *                /  |   \
 *               /   |    \
 *             (B0),(B2)...(Bn)/attackers

Network topology and DDoS attacks are created by our group memebers
with the reference to mmwave-simple-epc.cc file created by M. Miozzo
and N. Baldo (Modified by: M. Mezzavilla, S. Dutta, R. Ford, M. Zhang)
in [1] and the NS3 Cybersecurity Simulations created by Saket Upadhyay
([9] from reference list above).

Author: Namsakhi Kumar, Kwok Liang Lee, Edwin Tam

*/
#include "ns3/mmwave-helper.h"
#include "ns3/nstime.h"
#include "ns3/epc-helper.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/applications-module.h"
#include "ns3/point-to-point-helper.h"
#include "ns3/config-store.h"
#include "ns3/mmwave-point-to-point-epc-helper.h"

#include "ns3/netanim-module.h"
#include <sstream>
#include <string>

using namespace std;
using namespace ns3;
using namespace mmwave;

NS_LOG_COMPONENT_DEFINE ("MmWaveDDoS");


int main(int argc, char* argv[])
{
      uint16_t numEnb = 1; //Number of EnB (Base Station)
      uint16_t numUe = 1; //Number of Ue (User Equipment)
```

```
      uint16_t numAtt = 10; //Number of Attacker
      double simTime = 1.5; //Simulation Time
      bool EnableAttack = true; //Enable DDoS Attack
      bool harqEnabled = true; //Hybrid ARQ

      //Creating Nodes for Topology
      NodeContainer ueNodes; //for Ue nodes
      NodeContainer attNodes; //for attacker nodes
      NodeContainer enbNodes; //for EnB nodes

      ueNodes.Create(numUe); //Creating numUe amount of ue nodes
      attNodes.Create(numAtt); //Creating numAtt amount of attacker
nodes
      enbNodes.Create(numEnb); //Creating numEnb amount of enb nodes

      Ptr<MmWaveHelper> mmwaveHelper = CreateObject<MmWaveHelper>();
//Creating mmwave object for using MmWave functions
      Ptr<MmWavePointToPointEpcHelper> epcHelper =
CreateObject<MmWavePointToPointEpcHelper>(); //Creating Envolved Packet
Core object(EPC) for using LTE functions
      mmwaveHelper->SetEpcHelper(epcHelper);  //Link LTE to MmWave
      mmwaveHelper->SetHarqEnabled(harqEnabled); //To enable IP address
and MAC address conversion (Setting ARQ)

      Ptr<Node> pgw = epcHelper->GetPgwNode(); //Using EPC function to
create a Packet Data Network Gateway(PGW) node

      NodeContainer remoteHostContainer; //Create container for
remotehost/server
      remoteHostContainer.Create(1); //Creating 1 remote host node
      Ptr<Node> remoteHost = remoteHostContainer.Get(0); //Set the
first Remote Host Container to be a remoteHost node

      //Create the internet
      InternetStackHelper internet;
      internet.Install(remoteHostContainer); //Install internet to the
remote host such that the remote host has access to the internet
      internet.Install(ueNodes); //Install internet to User Equipments
such that User Eqiupments have access to the internet
      internet.Install(attNodes); //Install internet to Attacker such
that Attacker have access to the internet


      //Install mobility for each nodes
      //Install mobility model for Enb node
      Ptr<ListPositionAllocator> enbPositionAlloc =
CreateObject<ListPositionAllocator>();
      enbPositionAlloc->Add(Vector(40, 20, 0.0));//Set enb location
      MobilityHelper enbmobility;

enbmobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
```

```
      enbmobility.SetPositionAllocator(enbPositionAlloc);
      enbmobility.Install(enbNodes);//Install enbnode to its location

      //Install mobility model for Ue nodes
      MobilityHelper uemobility;
      Ptr<ListPositionAllocator> uePositionAlloc =
CreateObject<ListPositionAllocator>();
      for (unsigned i = 0; i < numUe; i++)
      {
            uePositionAlloc->Add(Vector(10, 10 * i, 0.0));//Location of
the ue node
      }

uemobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
      uemobility.SetPositionAllocator(uePositionAlloc);
      uemobility.Install(ueNodes);//Install the ue node to its location

      //Install mobility model for Attacker nodes
      MobilityHelper attackmobility;
      Ptr<ListPositionAllocator> attackPositionAlloc =
CreateObject<ListPositionAllocator>();
      for (unsigned i = 0; i < numAtt; i++)
      {
            attackPositionAlloc->Add(Vector(20, 5 * i, 0.0));//Location
of the attackers
      }

attackmobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
      attackmobility.SetPositionAllocator(attackPositionAlloc);
      attackmobility.Install(attNodes);//Install attack nodes to its
location

      //Install mobility model for pgw node
      MobilityHelper pgwmobility;
      Ptr<ListPositionAllocator> pgwPositionAlloc =
CreateObject<ListPositionAllocator>();
      pgwPositionAlloc->Add(Vector(40, 50, 0.0));//Set location for pgw
node

pgwmobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
      pgwmobility.SetPositionAllocator(pgwPositionAlloc);
      pgwmobility.Install(pgw);//Install pgw node to its location

      //Install mobility model for remote host
      MobilityHelper rmhmobility;
      Ptr<ListPositionAllocator> rmhPositionAlloc =
CreateObject<ListPositionAllocator>();
      rmhPositionAlloc->Add(Vector(40, 70, 0.0));//Set remotehost
location

rmhmobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
```

```
      rmhmobility.SetPositionAllocator(rmhPositionAlloc);
      rmhmobility.Install(remoteHost);//Install remotehost node to its
location

      //Install mMWave Device to the EnB nodes, Ue nodes, and Attacker
nodes, so Ue and Attacker will be wireless
      NetDeviceContainer enbmmWaveDevs =
mmwaveHelper->InstallEnbDevice(enbNodes);
      NetDeviceContainer uemmWaveDevs =
mmwaveHelper->InstallUeDevice(ueNodes);
      NetDeviceContainer attmmWaveDevs =
mmwaveHelper->InstallUeDevice(attNodes);

      //Create the Transmission line for remoteHost to pgw.*
      PointToPointHelper p2ph;
      p2ph.SetDeviceAttribute("DataRate",
DataRateValue(DataRate("1Gb/s")));//Data rate of the transmission line
      p2ph.SetDeviceAttribute("Mtu", UintegerValue(1500));//Maximum
transmission unit of each packet
      p2ph.SetChannelAttribute("Delay",
TimeValue(MicroSeconds(100.0)));//Delay of the link

      NetDeviceContainer internetDevices = p2ph.Install(pgw,
remoteHost); // Connecting PGW to RemoteHost
      Ipv4AddressHelper ipv4h;
      ipv4h.SetBase("1.0.0.0", "255.0.0.0");
      Ipv4InterfaceContainer internetIpIfaces =
ipv4h.Assign(internetDevices); //Assigning IP address to PGW and
RemoteHost

      Ipv4Address remoteHostAddr = internetIpIfaces.GetAddress(1);
//Get RemoteHost IP address
      Ipv4StaticRoutingHelper ipv4RoutingHelper; //Setting up ipv4
      Ptr<Ipv4StaticRouting> remoteHostStaticRouting =
ipv4RoutingHelper.GetStaticRouting(remoteHost->GetObject<Ipv4>());

remoteHostStaticRouting->AddNetworkRouteTo(Ipv4Address("7.0.0.0"),
Ipv4Mask("255.0.0.0"), 1);

      //Install the IP stack on the UEs and attackers
      Ipv4InterfaceContainer ueIpIface, attIpIface;
      ueIpIface =
epcHelper->AssignUeIpv4Address(NetDeviceContainer(uemmWaveDevs));
      attIpIface =
epcHelper->AssignUeIpv4Address(NetDeviceContainer(attmmWaveDevs));

      // Assign IP address to UEs
      for (uint32_t u = 0; u < ueNodes.GetN(); ++u)
      {
            Ptr<Node> ueNodePtr = ueNodes.Get(u);
            // Set the default gateway for the Ue
```

```
            Ptr<Ipv4StaticRouting> ueStaticRouting =
ipv4RoutingHelper.GetStaticRouting(ueNodePtr->GetObject<Ipv4>());

ueStaticRouting->SetDefaultRoute(epcHelper->GetUeDefaultGatewayAddress(
), 1);
      }

      // Assign IP address to attackers
      for (uint32_t u = 0; u < attNodes.GetN(); ++u)
      {
            Ptr<Node> attNodePtr = attNodes.Get(u);
            // Set the default gateway for the Attacker
            Ptr<Ipv4StaticRouting> attStaticRouting =
ipv4RoutingHelper.GetStaticRouting(attNodePtr->GetObject<Ipv4>());

attStaticRouting->SetDefaultRoute(epcHelper->GetUeDefaultGatewayAddress
(), 1);
      }

      mmwaveHelper->AttachToClosestEnb(uemmWaveDevs, enbmmWaveDevs);
//Attach Ue to Enb node by using mmwave (wireless)
      mmwaveHelper->AttachToClosestEnb(attmmWaveDevs, enbmmWaveDevs);
//Attach attackers to Enb node by using mmwave (wireless)

      //Ipv4GlobalRoutingHelper::PopulateRoutingTables();
      //Application
      uint32_t UDPport = 3001;
      uint32_t TCPport = 3001; //Set Socket port to 3001 (tcp)
      #define DDOS_RATE "1Gb/s"
      uint32_t PacketSize = 1000;

      //Setting for DDoS
      if (EnableAttack)
      {
            // DDoS Application Behaviour
            OnOffHelper onoff("ns3::UdpSocketFactory",
Address(InetSocketAddress(remoteHostAddr, UDPport)));//Setting up UDP
socket for application
//On time = 1 & off time = 0 below will set traffic into constant bit
rate
            onoff.SetAttribute("OnTime",
StringValue("ns3::ConstantRandomVariable[Constant=1]"));
            onoff.SetAttribute("OffTime",
StringValue("ns3::ConstantRandomVariable[Constant=0]"));
            onoff.SetConstantRate(DataRate(DDOS_RATE), PacketSize);
//Setting up a sending rate

            ApplicationContainer onOffApp[numAtt]; //Setting up the
number of containers for each attacker

            //Install application in all bots
```

```
            for (int k = 0; k < numAtt; ++k)
            {
                  onOffApp[k] = onoff.Install(attNodes.Get(k)); //Socket
link the K-th attacker to the remoteHost
                  onOffApp[k].Start(Seconds(0.5)); //Attack Transmission
Time Start
                  onOffApp[k].Stop(Seconds(simTime)); //Attack
Transmission Time End
            }
      }

      // Normal Application Behaviour
      OnOffHelper TCPonoff("ns3::TcpSocketFactory",
Address(InetSocketAddress(remoteHostAddr, TCPport)));//Setting up UDP
socket for application

      TCPonoff.SetConstantRate(DataRate("0.5Gb/s"),1000); //Setting up
a sending rate

//On time = 1 & off time = 0 below will set traffic into constant bit
rate
      TCPonoff.SetAttribute("OnTime",
StringValue("ns3::ConstantRandomVariable[Constant=1]"));

      TCPonoff.SetAttribute("OffTime",
StringValue("ns3::ConstantRandomVariable[Constant=0]"));

      ApplicationContainer TCPonOffApp[numUe]; //Setting up the number
of containers for each user

      for (int k = 0; k < numUe; ++k)
      {
                  TCPonOffApp[k] = TCPonoff.Install(ueNodes.Get(k));
//Socket link the K-th attacker to the remoteHost
                  TCPonOffApp[k].Start(Seconds(0.1)); //Attack
Transmission Time Start
                  TCPonOffApp[k].Stop(Seconds(simTime)); //Attack
Transmission Time End
      }

      //TCP sink on receiver side
      PacketSinkHelper TCPsink("ns3::TcpSocketFactory",
InetSocketAddress(Ipv4Address::GetAny(), TCPport)); //the first
parameter is the protocol that is used to receive traffic, and used to
create sockets for the applications

      ApplicationContainer TCPsinkApp = TCPsink.Install(remoteHost);
//Socket link to remoteHost

      TCPsinkApp.Start(Seconds(0.1)); //UDP Transmission Time Start
      TCPsinkApp.Stop(Seconds(simTime)); //UDP Transmission Time End
```

```
	//UDP sink on receiver side
	PacketSinkHelper UDPsink("ns3::UdpSocketFactory",
InetSocketAddress(Ipv4Address::GetAny(), UDPport)); //the first
parameter is the protocol that is used to receive traffic, and used to
create sockets for the applications

	ApplicationContainer UDPsinkApp = UDPsink.Install(remoteHost);
//Socket link to remoteHost

	UDPsinkApp.Start(Seconds(0.1)); //UDP Transmission Time Start
	UDPsinkApp.Stop(Seconds(simTime)); //UDP Transmission Time End

	//Trace file for each node
	mmwaveHelper->EnableTraces();
	p2ph.EnablePcapAll("Trace_file");


	//NetAnim setup and nodes labelling
	AnimationInterface anim("NetAnime_file.xml");

	anim.UpdateNodeDescription (pgw->GetId(), "PGW");
	anim.UpdateNodeDescription(remoteHost->GetId(), "SERVER");

	for(uint32_t i = 0; i < numAtt ; i++)
	{
		anim.UpdateNodeDescription(attNodes.Get(i)->GetId(),
"Attacker");

	}

	anim.UpdateNodeDescription(enbNodes.Get(0)->GetId(), "EnB");
	anim.UpdateNodeDescription(ueNodes.Get(0)->GetId(), "Ue");

	Simulator::Stop(Seconds(simTime));

	//Run the Simulation
	Simulator::Run();
	Simulator::Destroy();
	return 0;

}
```