# Project Title: DDoS simulation in NS-3

## INTRODUCTION:

Distributed denial of service (DDoS) attacks are a subclass of denial of service (DoS) attacks. A DDoS attack involves multiple connected online devices, collectively known as a botnet, which are used to overwhelm a target website with fake traffic.

Unlike other kinds of cyberattacks, DDoS assaults don't attempt to breach your security perimeter. Rather, a DDoS attack aims to make your website and servers unavailable to legitimate users. DDoS can also be used as a smokescreen for other malicious activities and to take down security appliances, breaching the target's security perimeter.

A successful distributed denial of service attack is a highly noticeable event impacting an entire online user base. This makes it a popular weapon of choice for hacktivists, cyber vandals, extortionists and anyone else looking to make a point or champion a cause.

DDoS attacks can come in short bursts or repeat assaults, but either way the impact on a website or business can last for days, weeks and even months, as the organization tries to recover. This can make DDoS extremely destructive to any online organization. Amongst other things, DDoS attacks can lead to loss of revenues, erode consumer trust, force businesses to spend fortunes in compensations and cause long-term reputation damage.

# OBJECTIVE/PURPOSE:

DDoS attack, the assailant exploits a vulnerability in one computer system, making it the DDoS master. The attack master system identifies other vulnerable systems and gains control of them by infecting them with malware or bypassing the authentication controls through methods like guessing the default password on a widely used system or device.

A computer or network device under the control of an intruder is known as a *zombie*, or *bot*. The attacker creates what is called a *command-and-control server* to command the network of bots, also called a *botnet*. The person in control of a botnet is referred to as the *botmaster*. That term has also been used to refer to the first system recruited into a botnet because it is used to control the spread and activity of other systems in the botnet.

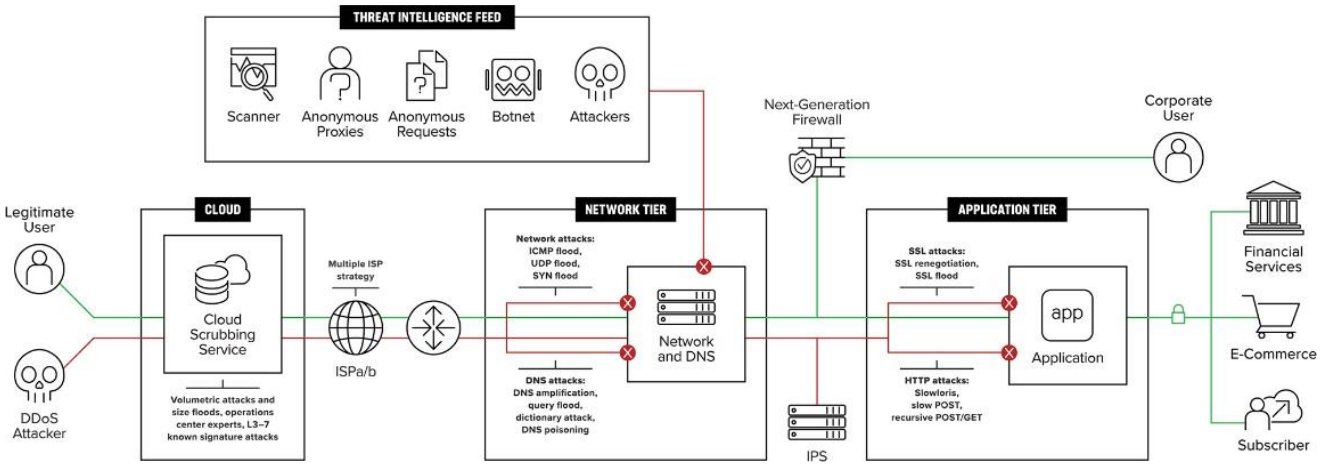
Botnets can be composed of almost any number of bots; botnets with tens or hundreds of thousands of nodes have become increasingly common. There may not be an upper limit to their size. Once the botnet is assembled, the attacker can use the traffic generated by the compromised devices to flood the target domain and knock it offline.

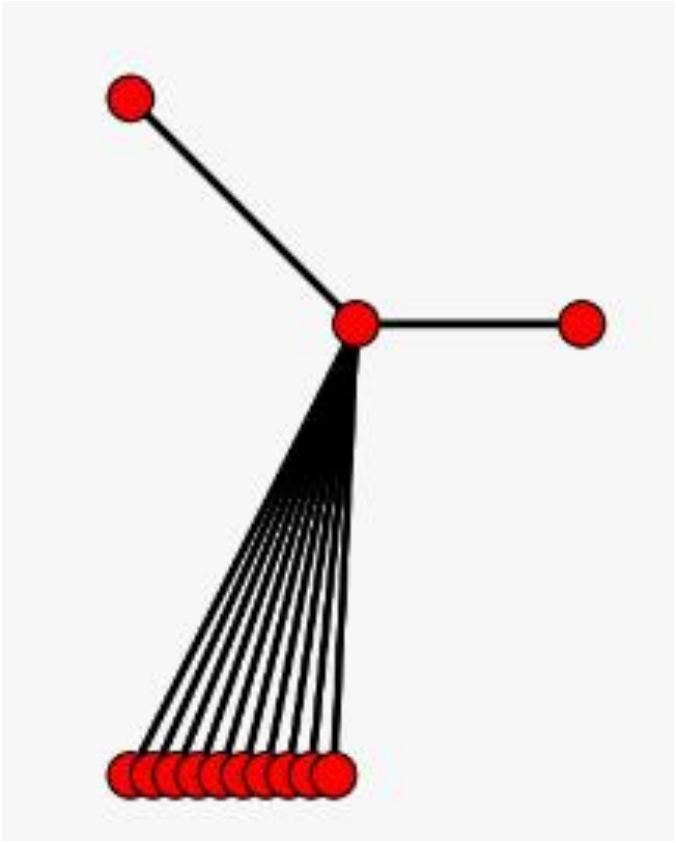
The target of a DDoS attack is not always the sole victim because DDoS attacks involve and affect many devices. The devices used to route malicious traffic to the target may also suffer a degradation of service, even if they aren't the main target.

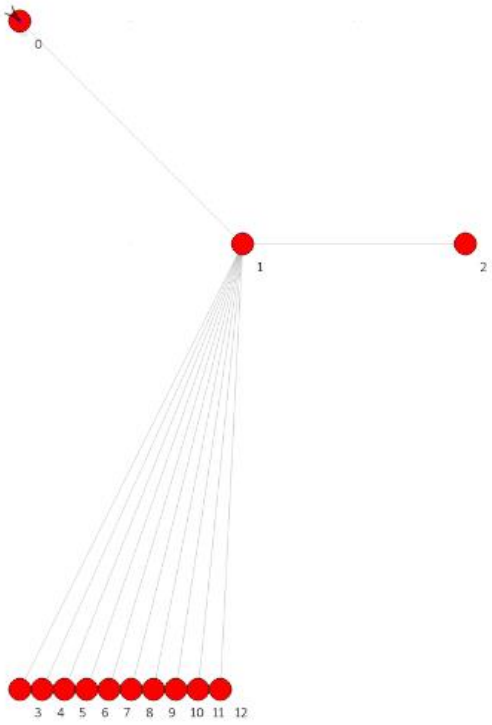
# SYSTEM REQUIREMENTS:

- Ubuntu Linux,
- NS3,
- NetAnim,
- Wireshark
- Python
- Ram: 8 GB
- I3 Processor

# NETWORK DESIGN:



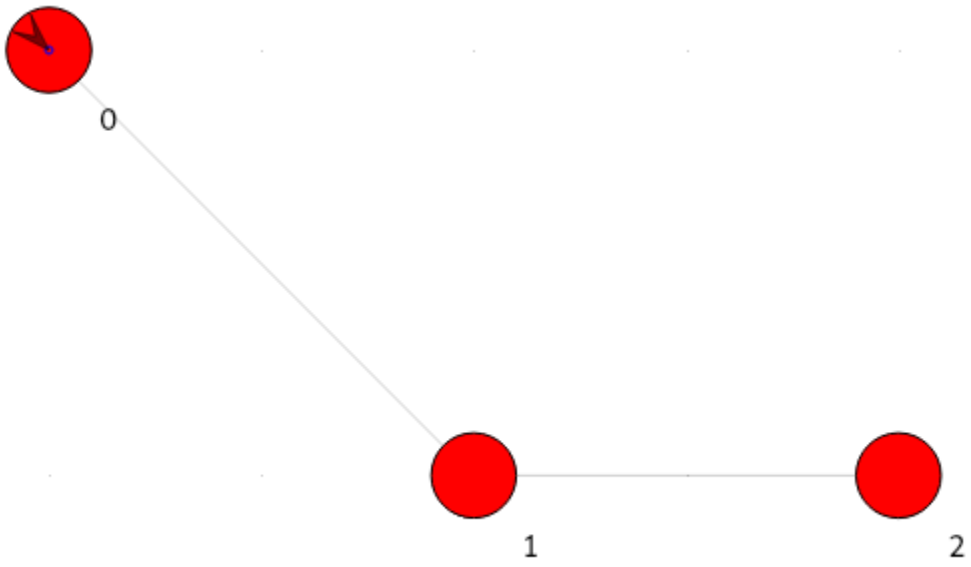



0
1
2
3 4 5 6 7 8 9 10 11 12

# STEPS FOR BUILDING NETWORK:

1) Creating Base Model

   The base model for this attack is relatively simple : We have 3 main nodes Alice [n0] (Legitimate Client), Bob [ n2] (Server Application) and a connecting node in between let's say Dave [n1].




Legitimate Connection Model

2) Now we will add as many bots we want to attack the network and let's call them Mallory [bi | i∈ (N)]. The bots will flood Dave in order to produce network congestion from n1 to n2. This will result in extended communication delay between Alice and Bob and eventually deny the resource for Alice.
3) Include all the required headers.

4) Now we define some global configurations so that we can play around with them later.
5) Now inside typical main() function we will start creating our Base Model designed above.
6) Creating Nodes

   Nodes are nothing but the points of communication we structured in our design phase.

   This will create 3 good nodes and as many Bot nodes defined by NUMBER_OF_BOTS.
7) Connecting Nodes

   We can use point-to-point-helper class to connect different nodes and then install the P2P device in the nodes to actually connect them. One thing to keep in mind here is that our number of bots are dynamic and we don't want the user to change the whole algorithm when changing the number so we can use simple for loop for P2P installation in the bots.

   We will use these "for-loops" wherever we will assign or install something to bot nodes.
8) Assignment of IP Address and Internet Stack
   In this case we are using IPv4 Address for our nodes and can be setup by ipv4-helper class. This will connect our devices to virtual internet stack.
9) Installing Attacker Application in bots
   Now we want to install applications in our nodes so that they can communicate with their neighbors. Let's create attacker application.
10) Setting up Animation in NetAnim Module
11) Configuring mobility in nodes
12) Placing Nodes in Simulation Grid

## SOURCE CODE:

```
#include <ns3/csma-helper.h>
#include "ns3/mobility-module.h"
#include "ns3/nstime.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/netanim-module.h"

#define TCP_SINK_PORT 9000
#define UDP_SINK_PORT 9001

//experimental parameters
#define MAX_BULK_BYTES 100000
#define DDOS_RATE "20480kb/s"
#define MAX_SIMULATION_TIME 10.0

//Number of Bots for DDoS
#define NUMBER_OF_BOTS 10

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("DDoSAttack");

int main(int argc, char *argv[])
{
   CommandLine cmd;
   cmd.Parse(argc, argv);

   Time::SetResolution(Time::NS);
   LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
   LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);


   //Legitimate connection bots
   NodeContainer nodes;
```

```
nodes.Create(3);

//Nodes for attack bots
NodeContainer botNodes;
botNodes.Create(NUMBER_OF_BOTS);

// Define the Point-To-Point Links and their Paramters
PointToPointHelper pp1, pp2;
pp1.SetDeviceAttribute("DataRate", StringValue("100Mbps"));
pp1.SetChannelAttribute("Delay", StringValue("1ms"));

pp2.SetDeviceAttribute("DataRate", StringValue("100Mbps"));
pp2.SetChannelAttribute("Delay", StringValue("1ms"));

// Install the Point-To-Point Connections between Nodes
NetDeviceContainer d02, d12, botDeviceContainer[NUMBER_OF_BOTS];
d02 = pp1.Install(nodes.Get(0), nodes.Get(1));
d12 = pp1.Install(nodes.Get(1), nodes.Get(2));

for (int i = 0; i < NUMBER_OF_BOTS; ++i)
{
   botDeviceContainer[i] = pp2.Install(botNodes.Get(i), nodes.Get(1));
}

//Assign IP to bots
InternetStackHelper stack;
stack.Install(nodes);
stack.Install(botNodes);
Ipv4AddressHelper ipv4_n;
ipv4_n.SetBase("10.0.0.0", "255.255.255.252");

Ipv4AddressHelper a02, a12, a23, a34;
a02.SetBase("10.1.1.0", "255.255.255.0");
a12.SetBase("10.1.2.0", "255.255.255.0");

for (int j = 0; j < NUMBER_OF_BOTS; ++j)
{
   ipv4_n.Assign(botDeviceContainer[j]);
   ipv4_n.NewNetwork();
}
```

```
   //Assign IP to legitimate nodes
   Ipv4InterfaceContainer i02, i12;
   i02 = a02.Assign(d02);
   i12 = a12.Assign(d12);

   // DDoS Application Behaviour
   OnOffHelper onoff("ns3::UdpSocketFactory",
Address(InetSocketAddress(i12.GetAddress(1), UDP_SINK_PORT)));
   onoff.SetConstantRate(DataRate(DDOS_RATE));
   onoff.SetAttribute("OnTime",
StringValue("ns3::ConstantRandomVariable[Constant=30]"));
   onoff.SetAttribute("OffTime",
StringValue("ns3::ConstantRandomVariable[Constant=0]"));
   ApplicationContainer onOffApp[NUMBER_OF_BOTS];

   //Install application in all bots
   for (int k = 0; k < NUMBER_OF_BOTS; ++k)
   {
      onOffApp[k] = onoff.Install(botNodes.Get(k));
      onOffApp[k].Start(Seconds(0.0));
      onOffApp[k].Stop(Seconds(MAX_SIMULATION_TIME));
   }

   // Sender Application (Packets generated by this application are throttled)
   BulkSendHelper bulkSend("ns3::TcpSocketFactory",
InetSocketAddress(i12.GetAddress(1), TCP_SINK_PORT));
   bulkSend.SetAttribute("MaxBytes", UintegerValue(MAX_BULK_BYTES));
   ApplicationContainer bulkSendApp = bulkSend.Install(nodes.Get(0));
   bulkSendApp.Start(Seconds(0.0));
   bulkSendApp.Stop(Seconds(MAX_SIMULATION_TIME - 10));

   // UDPSink on receiver side
   PacketSinkHelper UDPsink("ns3::UdpSocketFactory",
                     Address(InetSocketAddress(Ipv4Address::GetAny(),
UDP_SINK_PORT)));
   ApplicationContainer UDPSinkApp = UDPsink.Install(nodes.Get(2));
   UDPSinkApp.Start(Seconds(0.0));
   UDPSinkApp.Stop(Seconds(MAX_SIMULATION_TIME));
```

```
   // TCP Sink Application on server side
   PacketSinkHelper TCPsink("ns3::TcpSocketFactory",
                        InetSocketAddress(Ipv4Address::GetAny(),
TCP_SINK_PORT));
   ApplicationContainer TCPSinkApp = TCPsink.Install(nodes.Get(2));
   TCPSinkApp.Start(Seconds(0.0));
   TCPSinkApp.Stop(Seconds(MAX_SIMULATION_TIME));

   Ipv4GlobalRoutingHelper::PopulateRoutingTables();

   //Simulation NetAnim configuration and node placement
   MobilityHelper mobility;

   mobility.SetPositionAllocator("ns3::GridPositionAllocator",
                         "MinX", DoubleValue(0.0), "MinY", DoubleValue(0.0),
"DeltaX", DoubleValue(5.0), "DeltaY", DoubleValue(10.0),
                         "GridWidth", UintegerValue(5), "LayoutType",
StringValue("RowFirst"));

   mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
   mobility.Install(nodes);
   mobility.Install(botNodes);

   AnimationInterface anim("DDoSim.xml");

   ns3::AnimationInterface::SetConstantPosition(nodes.Get(0), 0, 0);
   ns3::AnimationInterface::SetConstantPosition(nodes.Get(1), 10, 10);
   ns3::AnimationInterface::SetConstantPosition(nodes.Get(2), 20, 10);

   uint32_t x_pos = 0;
   for (int l = 0; l < NUMBER_OF_BOTS; ++l)
   {
      ns3::AnimationInterface::SetConstantPosition(botNodes.Get(l), x_pos++, 30);
   }
   pp1.EnablePcapAll("DDOS-p2p");
   //Run the Simulation
   Simulator::Run();
   Simulator::Destroy();
   return 0;
}
```

# OUTPUT:

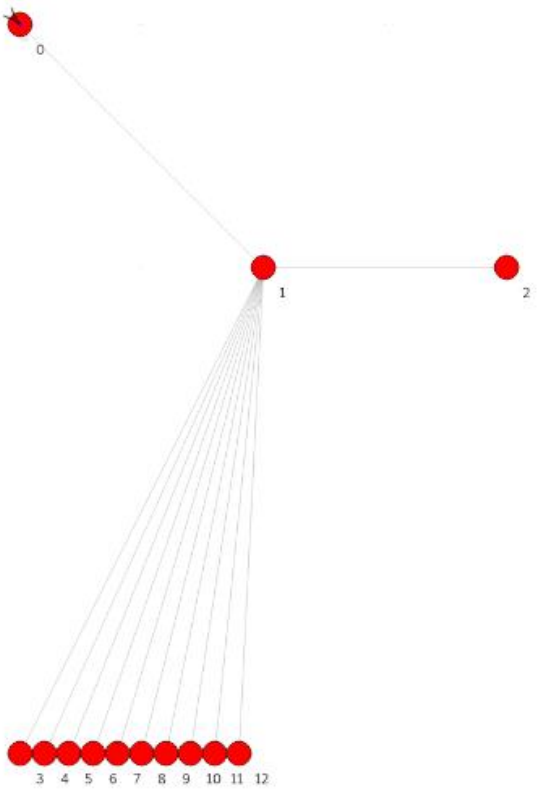
**Commands to write in terminal**

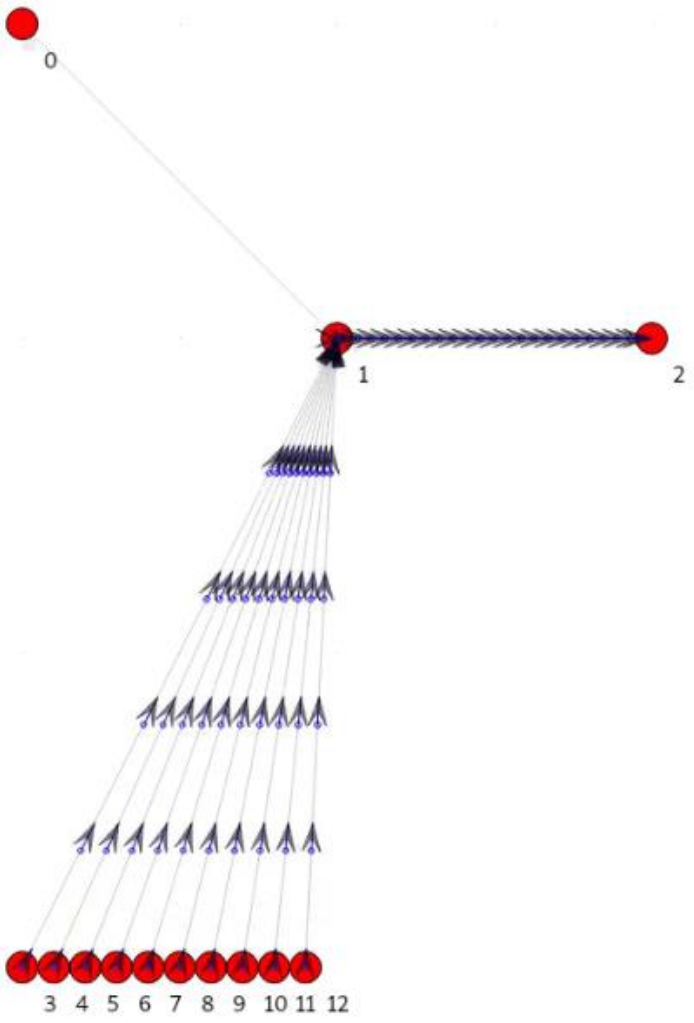
- ./waf --run scratch/ddos

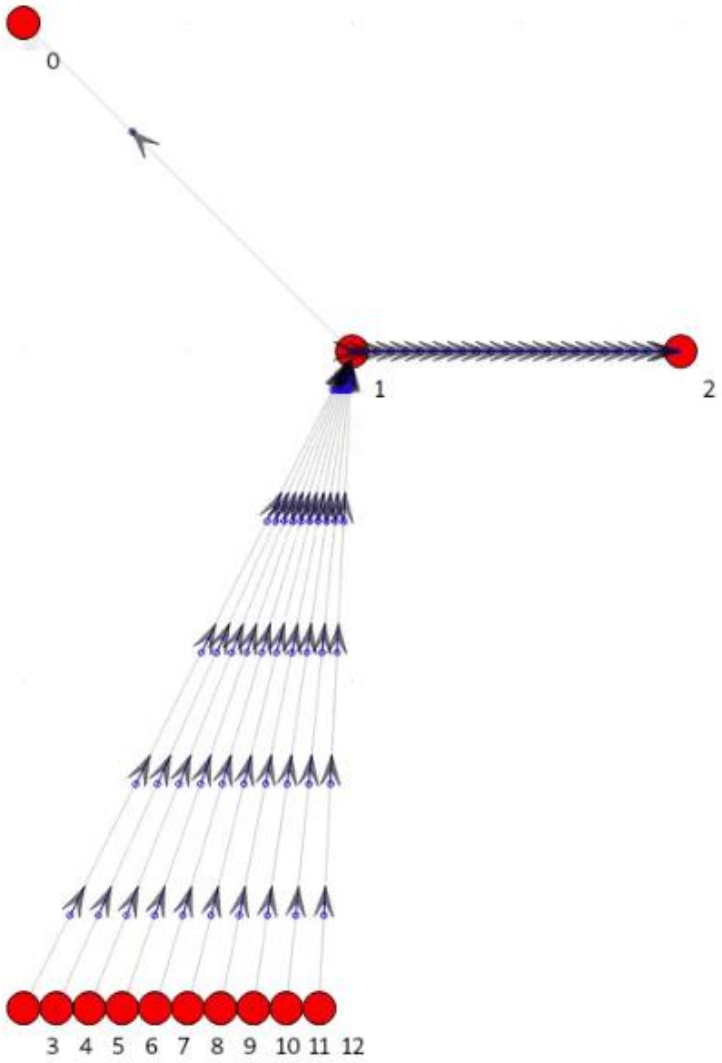
**To run NetAnim simulation of the ddos attack**

**Change directory to netanim-3.108 folder, open terminal and write**

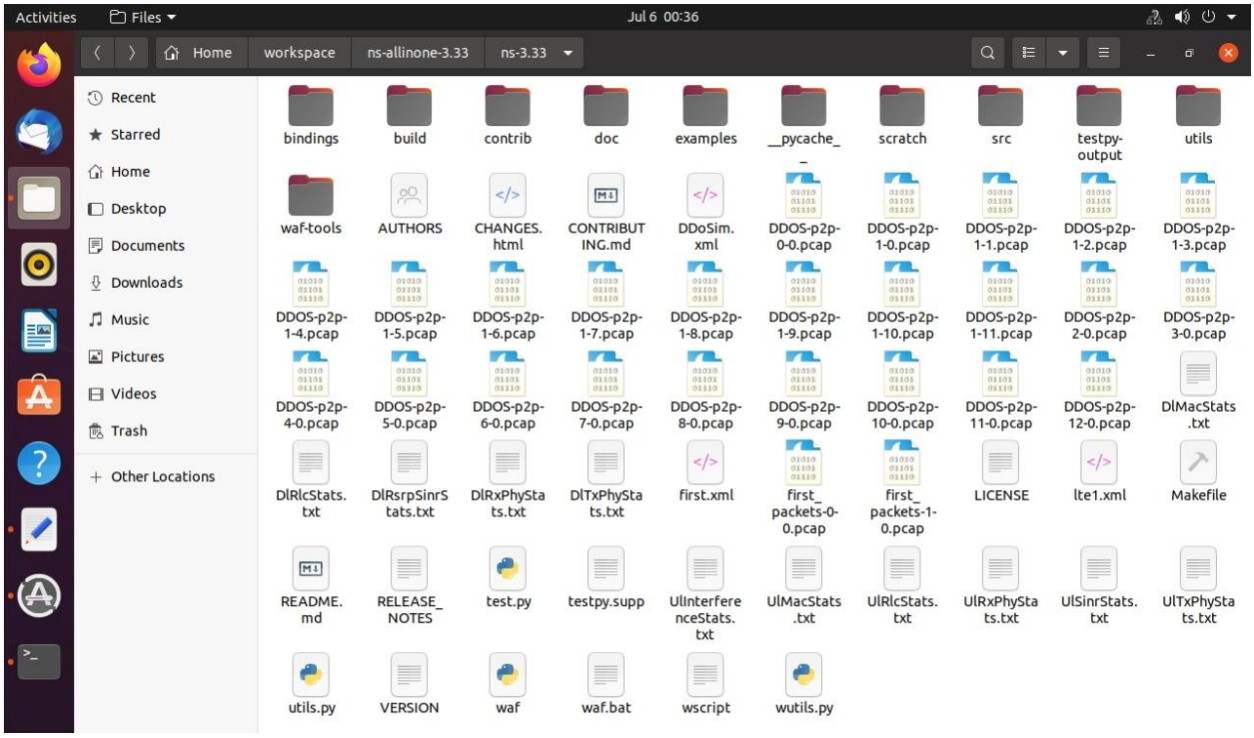
- **./NetAnim**

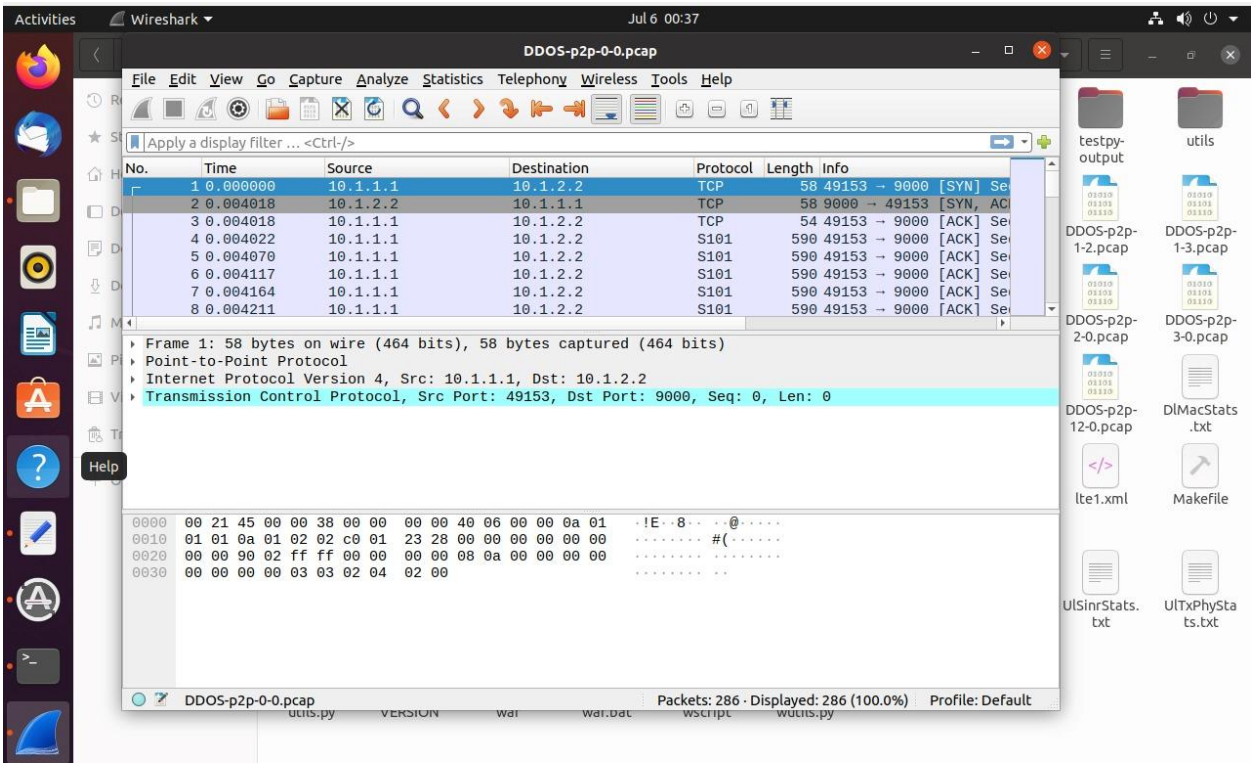


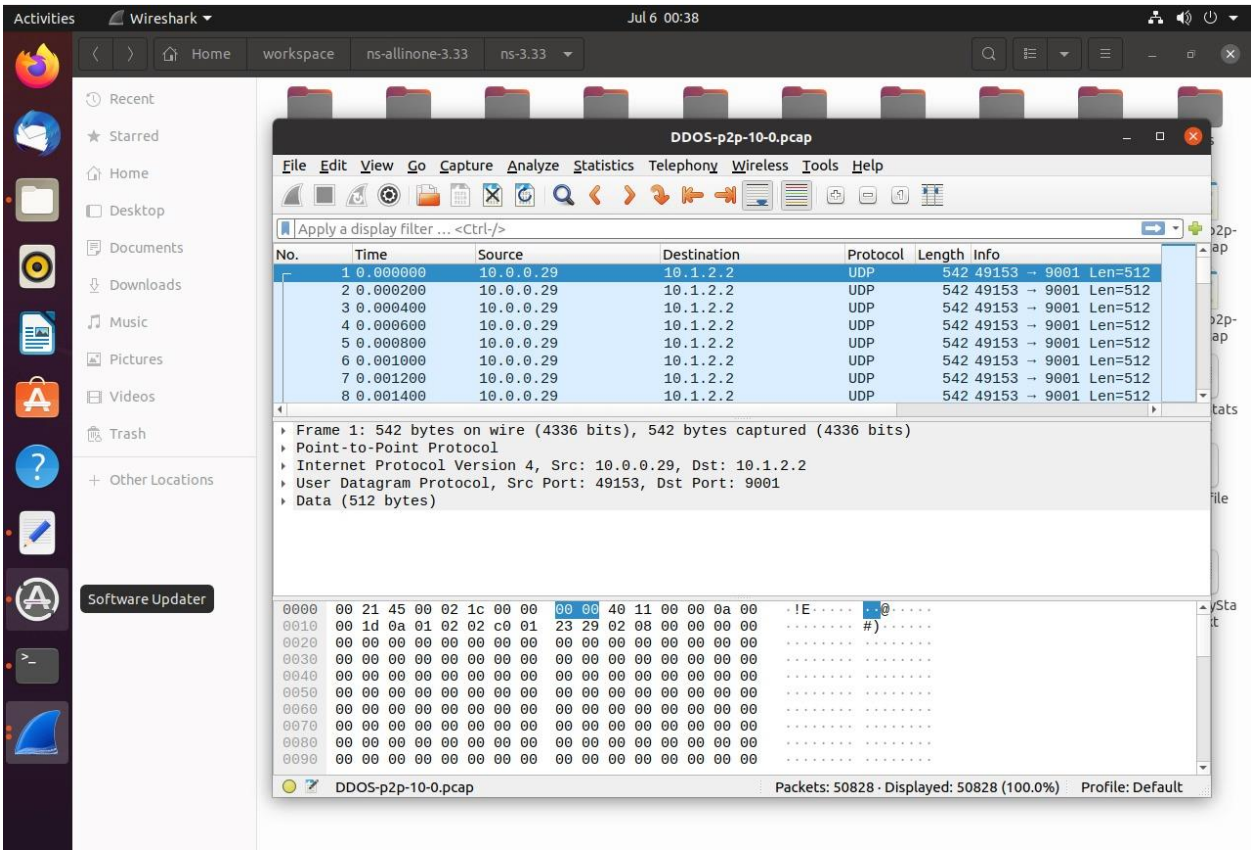

0
1
2
3 4 5 6 7 8 9 10 11 12


0
1
2
3 4 5 6 7 8 9 10 11 12

# WIRESHARK:


Activities
Files
Jul 6 00:36
Home
workspace
ns-allinone-3.33
ns-3.33
Recent
Starred
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Trash
Other Locations
bindings
build
contrib
doc
examples
__pycache__
scratch
src
testpy-output
utils
waf-tools
AUTHORS
CHANGES.html
CONTRIBUTING.md
DDoSim.xml
DDOS-p2p-0-0.pcap
DDOS-p2p-1-0.pcap
DDOS-p2p-1-1.pcap
DDOS-p2p-1-2.pcap
DDOS-p2p-1-3.pcap
DDOS-p2p-1-4.pcap
DDOS-p2p-1-5.pcap
DDOS-p2p-1-6.pcap
DDOS-p2p-1-7.pcap
DDOS-p2p-1-8.pcap
DDOS-p2p-1-9.pcap
DDOS-p2p-1-10.pcap
DDOS-p2p-1-11.pcap
DDOS-p2p-2-0.pcap
DDOS-p2p-3-0.pcap
DDOS-p2p-4-0.pcap
DDOS-p2p-5-0.pcap
DDOS-p2p-6-0.pcap
DDOS-p2p-7-0.pcap
DDOS-p2p-8-0.pcap
DDOS-p2p-9-0.pcap
DDOS-p2p-10-0.pcap
DDOS-p2p-11-0.pcap
DDOS-p2p-12-0.pcap
DlMacStats.txt
DlRlcStats.txt
DlRsrpSinrStats.txt
DlRxPhyStats.txt
DlTxPhyStats.txt
first.xml
first_packets-0-0.pcap
first_packets-1-0.pcap
LICENSE
lte1.xml
Makefile
README.md
RELEASE_NOTES
test.py
testpy.supp
UlInterferenceStats.txt
UlMacStats.txt
UlRlcStats.txt
UlRxPhyStats.txt
UlSinrStats.txt
UlTxPhyStats.txt
utils.py
VERSION
waf
waf.bat
wscript
wutils.py


Activities
Wireshark
Jul 6 00:37
DDOS-p2p-0-0.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 0.000000 10.1.1.1 10.1.2.2 TCP 58 49153 → 9000 [SYN] Se
2 0.004018 10.1.2.2 10.1.1.1 TCP 58 9000 → 49153 [SYN, AC
3 0.004018 10.1.1.1 10.1.2.2 TCP 54 49153 → 9000 [ACK] Se
4 0.004022 10.1.1.1 10.1.2.2 S101 590 49153 → 9000 [ACK] Se
5 0.004070 10.1.1.1 10.1.2.2 S101 590 49153 → 9000 [ACK] Se
6 0.004117 10.1.1.1 10.1.2.2 S101 590 49153 → 9000 [ACK] Se
7 0.004164 10.1.1.1 10.1.2.2 S101 590 49153 → 9000 [ACK] Se
8 0.004211 10.1.1.1 10.1.2.2 S101 590 49153 → 9000 [ACK] Se
Frame 1: 58 bytes on wire (464 bits), 58 bytes captured (464 bits)
Point-to-Point Protocol
Internet Protocol Version 4, Src: 10.1.1.1, Dst: 10.1.2.2
Transmission Control Protocol, Src Port: 49153, Dst Port: 9000, Seq: 0, Len: 0
0000 00 21 45 00 00 38 00 00 00 00 40 06 00 00 0a 01
0010 01 01 0a 01 02 02 c0 01 23 28 00 00 00 00 00 00
0020 00 00 90 02 ff ff 00 00 00 00 08 0a 00 00 00 00
0030 00 00 00 00 03 03 02 04 02 00
DDOS-p2p-0-0.pcap
Packets: 286 · Displayed: 286 (100.0%)
Profile: Default
Help




## CONCLUSION:

We can experiment with the above program and increase or decrease number of bots, bandwidth etc and observe the effect on the network.

In this way the simulation was performed successfully and DDoS was demonstrated.

## REFERENCE:


- https://www.nsnam.org/docs/tutorial/html
- https://infosecwriteups.com/ddos-simulation-in-ns-3-c-12f031a7b38c