ENSC 427: COMMUNICATION NETWORKS
SPRING 2024

FINAL PROJECT
Investigating the Effects of DDoS Attacks on End Users Over Different Ethernet Mediums

https://joshma25.github.io/

Wong, Joshua	301374810	jcw49@sfu.ca
Ma, Joshua	301357872	jma154@sfu.ca
Speers, Andrew	301362782	aspeers@sfu.ca

Group 2

Abstract

Distributed denial-of-service (DDoS) attacks are a great threat in the modern day and can have a major effect on people using the free internet since they can prevent individuals from accessing services and websites. Ethernet connections are how a vast majority of people access the Internet and this project analyzes how different ethernet cables are affected by DDoS attacks using ns-3 simulations. This study hopes to draw relationships between the number of attacker bots and the data rate of a given ethernet cable. After proceeding with our experimentation and simulations, we found that the relationship between these two variables is an inverse linear relationship (1/t). As the data rate of the cable increases, the delay caused by DDoS attack congestion decreases. This relationship was determined by measuring the time taken for a packet to be sent to a server and received back during a DDoS attack. We conclude that choosing an ethernet cable with faster data rates does improve performance under the load of a DDoS attack, however, our simulation environment was not very realistic. Despite this, the relationships drawn from this study may still be used in real world wired network applications.

Table Of Contents

Introduction

DDoS attacks are used by attackers to prevent people from having access to an online service and can happen to any service at any time. While the responsibility of defense against DDoS attacks often falls onto the internet service provider or the owner of the server under attack, we wonder about the effects of the ethernet cables we use when under these attacks as well. In this project, we will be testing different speeds of ethernet cables while under the duress of an increasing number of bots during a DDoS attack. Since ethernet is the most common way that we access the internet at home we will be testing the speeds of the most commonly used cables, Category 5 (Cat5:100Mbps), Category 5e (Cat5e:1Gbps) and Category 6 (Cat6:10Gbps). We will use the software ns-3 to simulate these attacks and then use NetAnim to give the data a visual representation of the DDoS attacks. We will use this data to come to a conclusion on the relationship between the number of daemon bots, cable data rate and the time it takes to receive a response packet from a targeted server. The article "Simulation of DDOS Attacks on P2P Networks" tells us that DDoS attacks target server nodes to bottleneck the server. With this bottleneck, the user must send even more packets to maintain connection, further adding congestion which in turn leads to more packet loss. All of these factors affect the target node as well as the point-to-point network as a whole [1]. This leads us to believe that the speed of the cables used and our simulation of this will give us valuable data in understanding the relationship between the data rate on different factors in a DDoS attack.

Background

DDoS Attack

A DDoS attack is a malicious coordinated attack to deny access to a given service by legitimate users of that service. DDoS attacks are orchestrated by the attacker hacking into hundreds or thousands of compromised machines to create a botnet composed of machines known as daemons or zombies. There are many kinds of DDoS attacks but a large majority are volumetric attacks where large amounts of traffic are targeted at a network to cause massive slowdown or sometimes complete shutdown of that service [2-3]. A smurf attack aims at sending a large volume of ICMP (Internet Control Message Protocol) packets at the targeted server with the source IP address spoofed to be identical to the selected victim's IP address. This causes the victim to be bombarded with ICMP echo packets resulting in the service being made unavailable. Smurf attacks have had plenty of time to have strategies created to combat them so the issue of smurf attacks has largely been solved [4]. A SYN flood attack tries to consume all of the resources of a server by sending a flood of SYN packets. This is achieved by abusing the 3-way handshake protocol by sending thousands of first connection requests and then never sending the third ACK request back. This causes the targeted server to maintain a server port connection for a non-trivial period wasting resources and preventing access to the service [5]. A UDP (User Datagram Protocol) flood is carried out in a fairly simple way, by sending a massive volume of UDP packets to the target and simply overwhelming the server's ability to handle the amount of bogus traffic. The server suffering from the throttle caused by the constant attack of the UDP flood can even have its firewalls overwhelmed by the attack causing denial of legitimate service that tries to reach the server [6]. The image below in Figure 1 shows a typical configuration of a DDoS attack and how one is carried out.

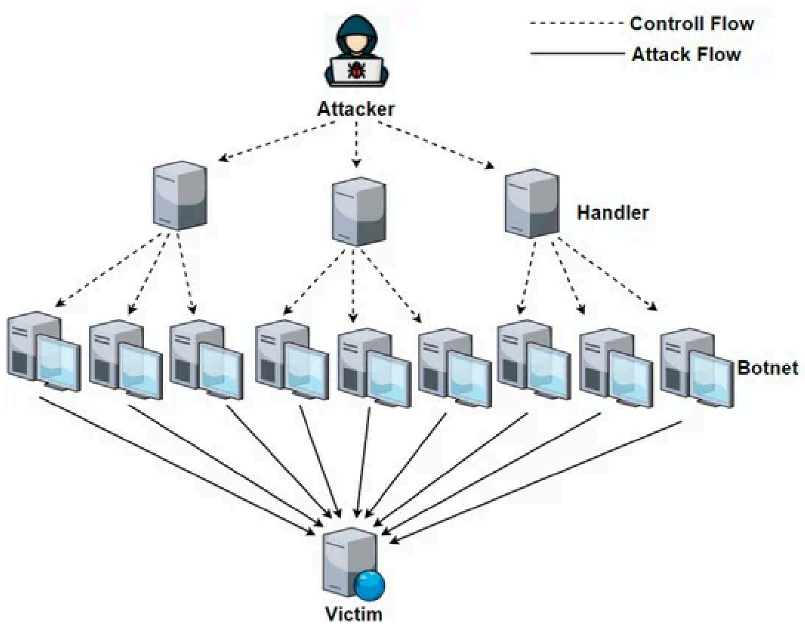

Figure 1: General Structure of a DDoS Attack [9]

Just like the technology we use, DDoS attacks have become more powerful over time. In modern times the typical botnet usually contains 400 - 2000 daemon machines [7]. These attacks also have DDoS attack rates of 5-10 Gbps depending on the severity of the attack but attack rates can reach upwards of 2Tbps [8]. Even with these tremendous rates of attack and with high amounts of attacking bots a majority of DDoS attacks don't completely saturate the node links, usually only reaching 95% saturation for less than 10 minutes at a time.

Ethernet Cable

The ethernet protocol provides wired network connections compared to the wireless connections that protocols such as WiFi provide. Wired connections have advantages over wireless connections, in general, wired connections such as ethernet cables provide more reliable and higher speed data transfer when compared to wireless counterparts [10]. The construction of ethernet cables greatly affects the data transfer speed of a given cable and is worth investigating further. Ethernet cables are comprised of four twisted pairs of copper wires. Twisting two separate wires into a twisted pair helps to reduce the noise and crosstalk between neighboring wires when compared to just a standalone wire. This is due to the fact that twisting wires together helps to resist electromagnetic interference (EMI) that occurs between the individual wires within the ethernet cable. To further protect these eight individual data carrying wires, shielding techniques can be employed to further reduce noise within the ethernet cable [10]. Shielding techniques require that the individual copper wires, the ethernet cable as a whole or both are encased in some sort of protective layer. The two most common shielding materials are aluminum foil and aluminum braided screens [11]. As is indicated by the name, aluminum foil shielding utilizes a thin layer of aluminum to protect the wires or cable from external noises. The same concept applies to braided screens with the exception that braided screens are constructed differently when compared to foil. Braided screens, by the name, are a thin jacket of flat aluminum that protects the wires or cable from external noises. Comparing foil to braided shielding, foil is lighter and less expensive than braided shielding. However, braided shielding offers less resistance compared to foil and makes for better shielding in high frequency applications [12]. Another method of shielding is to physically separate the twisted pairs within the cable. This separation is done with the addition of a plastic spline [13]. Referring to the Figures below, we can see the difference between the different types of shielding techniques.

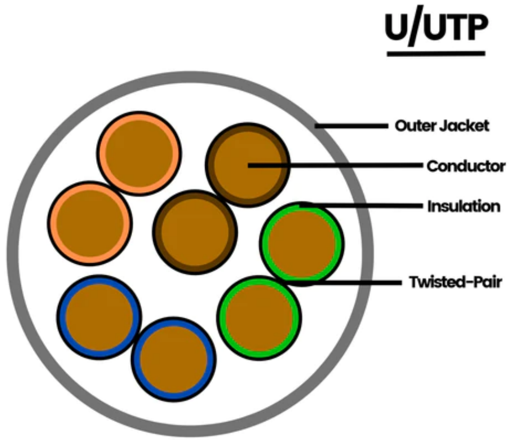

Figure 2: Unshielded Cable [11]

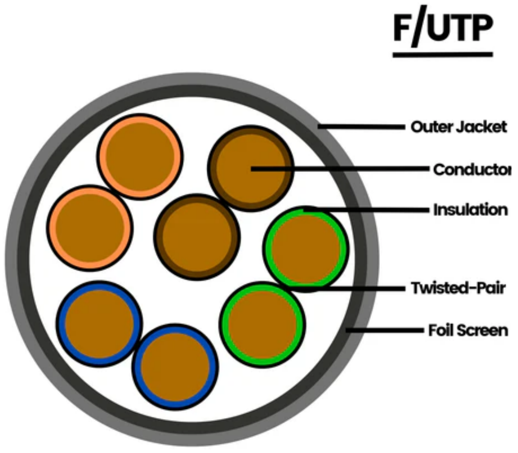

Figure 3: Foil Shielded Cable [11]

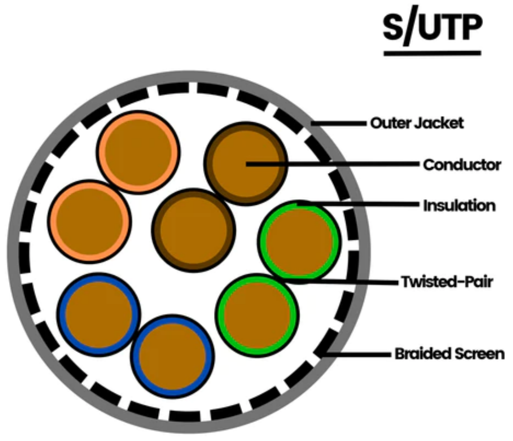

Figure 4: Braided Screen Shielded Cable [11]

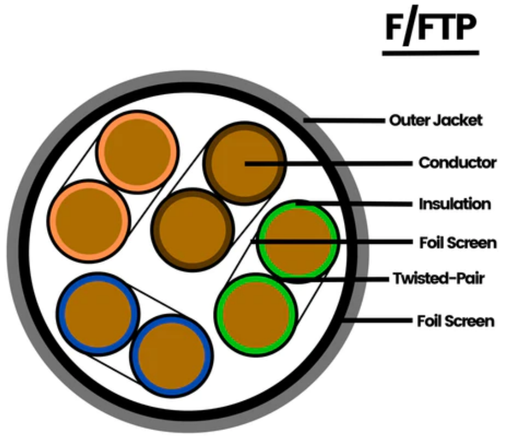

Figure 5: Foil Shielded Cable and Twisted Pairs [11]

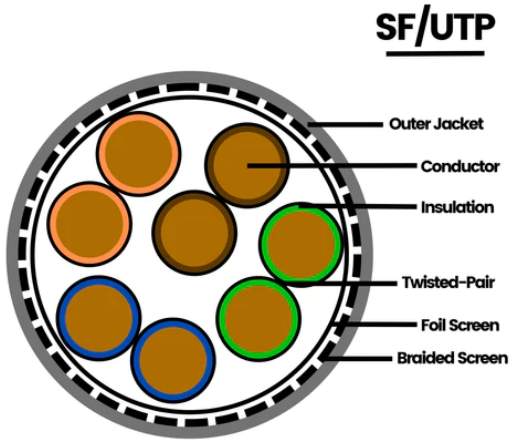

Figure 6: Foil and Braided Screen Shielded Cable [11]

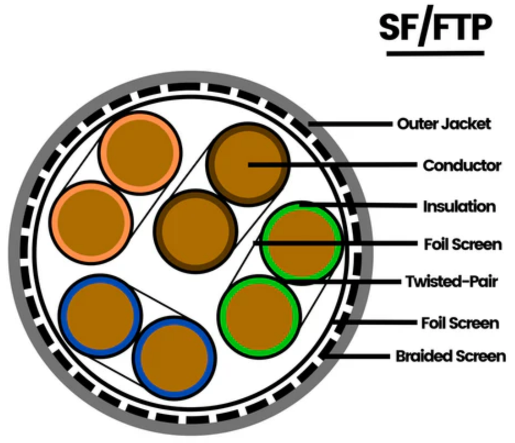

Figure 7: Foil and Braided Screen Shielded Cable and Foil Shielded Twisted Pairs [11]

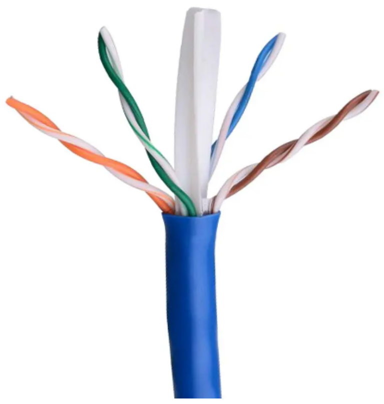
Figure 8: Plastic Spline [14]

Look at figures 2-7, we can see the differences between foil and braided screen shielding. We can also see that the cable or the twisted pairs or both can be shielded. Inspecting Figure 8, we can see the plastic spline that physically separated the twisted pairs directly in the middle of the cable. We can also see the four twisted pairs.

Now that we have discussed the methods of construction for ethernet cables, we can look into the types of ethernet cables and the differences between them. A summary of different types of ethernet cables can be seen in Table 1 below.

Category	Max Speed	Max Bandwidth	Application
Cat4	16 Mbps	20 MHz	Found in older buildings that are unable to upgrade ethernet cables, mostly for IBM Token Ring networks
Cat5	100 Mbps	100 MHz	Older hardware but still used in home networks
Cat5e	1 Gbps	100 MHz	Home networks
Cat6	1-10 Gbps	250 MHz	Home and office networks
Cat6a	10 Gbps	500 MHz	Office networks and small data centers
Cat7	40 Gbps	600 MHz	Data centers and server rooms

Table 1: Ethernet Cable Types and Applications [15-16]

The most common cable types used in home networks are Cat5, Cat5e and Cat6, therefore these three cable types will be the main scope of our experimentation. As mentioned before, ethernet cables provide a high speed connection for data transfer, and for most end users, this is the most important characteristic of a cable. In order to achieve these faster speeds, maximum bandwidth supported by the cable must be increased to accommodate for this increase in speed. However, as a result of this increase in bandwidth, there is also an increase in high frequency noise that is produced by the wires in the cable. So, in order to achieve higher bandwidth, which in turn increases transfer speeds, shielding and noise suppression techniques must be implemented to reduce the noise associated with higher bandwidth. This relationship between transfer speed, bandwidth, and high frequency noise suppression can be seen in the jump from Cat5e to Cat6 cables. From Table 1, we can see that bandwidth increases from 100 MHz to 250 MHz. As a result, max transfer speed increases from 1 Gbps to 10 Gbps. This increase in bandwidth was made possible by the addition of a plastic spline that physically separated the twisted pairs [13]. This addition of the plastic spline as well as other shielding techniques is the main reason behind the increases in bandwidth between ethernet cable types.

Procedure

To carry out our experiment we utilized and adapted some code written by Saket-Upadhyay [17]. This code operates by creating one legitimate user node, one server node, one node that connects the user and server, and a varied number of bots ranging from 10 to 100. The DDoS attack rate was then set to a constant 20480 Kbps. When the code is executed the bots flood the network with packets adding congestion and throttling our legitimate user's packets. The amount of congestion is proportional to the amount of bots. The data rate of the connection from the user node to the connection node is also varied, this connection simulates an ethernet cable. To measure the delay experienced by the user, we measure the time it takes for the first packet entering the congested network to return to the user. Below is some pseudo code explaining how the network is created and configured.

1. Creating 3 nodes, a node for the legitimate user, the destination server, and the interface between the two
2. Create N number of attacker bot nodes
3. Create point-to-point connections between the user node and the server node
4. Create point-to-point connections between attacker bots and server
5. Loop N times to create all attacker bot connections
6. Assign an IP address to the user and the server
7. Assign IP addresses to all attacker bots
8. Loop N times to assign all attacker bot addresses
9. Setup UDPSink on the receiver side
10. Setup TCPSink on the server side
11. Create a .xml file with all of the required data
12. Configure NetAnim with the .xml file

Please refer to Appendix I for the full Source Code

Below we can see in the following figures a simulation of 10 bots for Figure 9 and 30 bots for Figure 10 of the NetAnim program simulating the DDoS attack from the data created using our .xml files. This simulation was run many times to create Table 2 and Table 3 seen in the results section by varying the number of bots used and the data rate of the cable in the simulation.

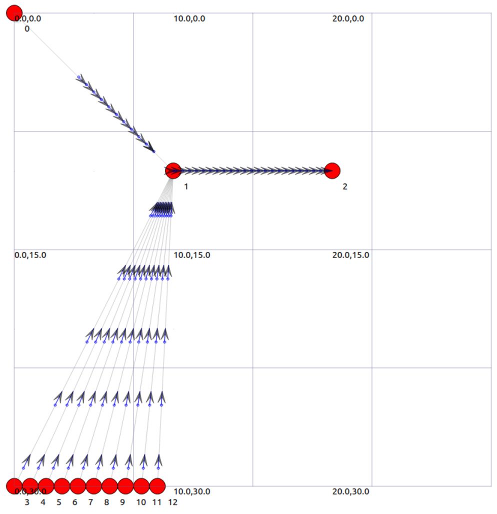

Figure 9: Simulation Structure for 10 Bot Attack

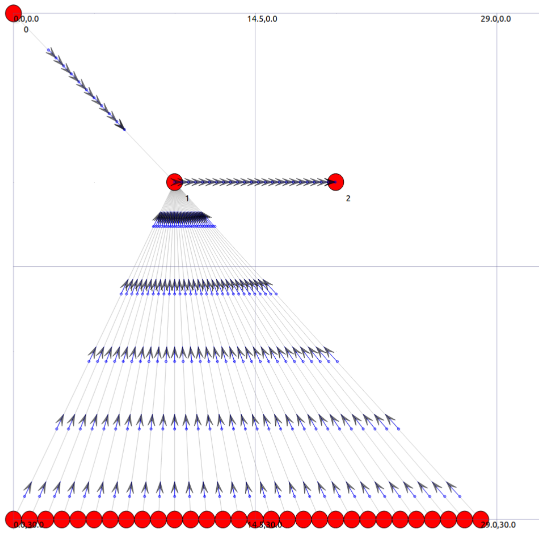

Figure 10: Simulation Structure for 30 Bot Attack

Results

At first, we ran the code with 10 bots and a data rate of 100 Mbps, 1 Gbps, and 100 Gbps which correspond to the Cat5, Cat5e, and Cat6 ethernet cables. There is a clear delay in the Cat5 cable as the time it took for the first packet sent through congestion to return is 0.008698s which is over twice the amount of time it takes for a packet to return with no congestion which is 0.0039861s. In contrast, there is virtually no delay for the Cat5e and Cat6 ethernet cables. To be able to see a delay, we ran the code and increased the number of bots attacking the server up to 100. As we increased the number of bots, the delay in the Cat5 cable further increased up to 0.01416s whereas the delay in the Cat5e cable became more noticeable at 50 bots. At 100 bots, the Cat5e cable experienced a delay of around 0.00043s. However, we still failed to see much difference in transmission time in the Cat6 cable. While the delay was slightly longer at 100 bots, this minuscule delay can also be attributed to human error.

	Data Rate (Cable Type)			
	# of bots	Cat5 (100 Mbps)	Cat5e (1 Gbps)	Cat6 (10 Gbps)
Number of Bots DDoS Rate: 20480 kb/s	10	0.008698s	0.003998s	0.00395662s
	20	0.008438s	0.0040106s	0.00400249s
	30	0.00841s	0.004013s	0.004004s
	40	0.008404s	0.004021s	0.00400003s
	50	0.00842776s	0.00436642s	0.0040013s
	75	0.01208s	0.00443995s	0.00400186s

	100	0.01416s	0.0044132s	0.0040137s
0 bots and 100 Mbps (Control Measurement) = 0.0039861s				

Table 2: Time of Packet from Being Sent to Being Returned

We can see that from our data above an increase in the number of bots clearly increases the delay experienced which is as expected. It also appears that ethernet cables that have a higher data rate are more capable of handling higher levels of congestion since the Cat6 cable is barely affected even when 100 bots are attacking the server whereas the Cat5 cable experienced noticeable congestion even at 10 bots.

In order to be able to see the relationship between the number of bots attacking the server and the delay experienced by the Cat5 cable, we then ran the code with 10 bots and a data rate of 16 Mbps which is the maximum data rate for a Cat4 cable. We then reran the code and increased the data rate to 20Mbps and then by increments of 10 Mbps up to 200Mbps.

Rate	Time
16 Mbps (Cat4)	0.03165964s
20 Mbps	0.03026256s
30 Mbps	0.0201829s
40 Mbps	0.015113s
50 Mbps	0.012863s
60 Mbps	0.0113635s
70 Mbps	0.010327s
80 Mbps	0.009641s
90 Mbps	0.009464s
100 Mbps (Cat 5)	0.008698s
110 Mbps	0.00817121s
120 Mbps	0.00750354s
130 Mbps	0.0069479s
140 Mbps	0.00646959s
150 Mbps	0.00605988s

160 Mbps	0.00569366s
170 Mbps	0.00511622s
180 Mbps	0.00484477s
190 Mbps	0.00458675s
200 Mbps	0.00438176s
300 Mbps	0.00404s

Table 3: Time of Packet from Being Sent to Being Returned

The exact relationship between the data rate and the delay is not immediately obvious since it doesn't appear to be linear. By plotting these results onto a graph with Mbps in the x-axis and the time it takes for the packet to return in the y-axis and comparing it to what we found to be the best-fit equation, we find that the equation $Time\ for\ packet\ return\ = \frac{700}{Mbps}$ matches closely to the points that we plotted.

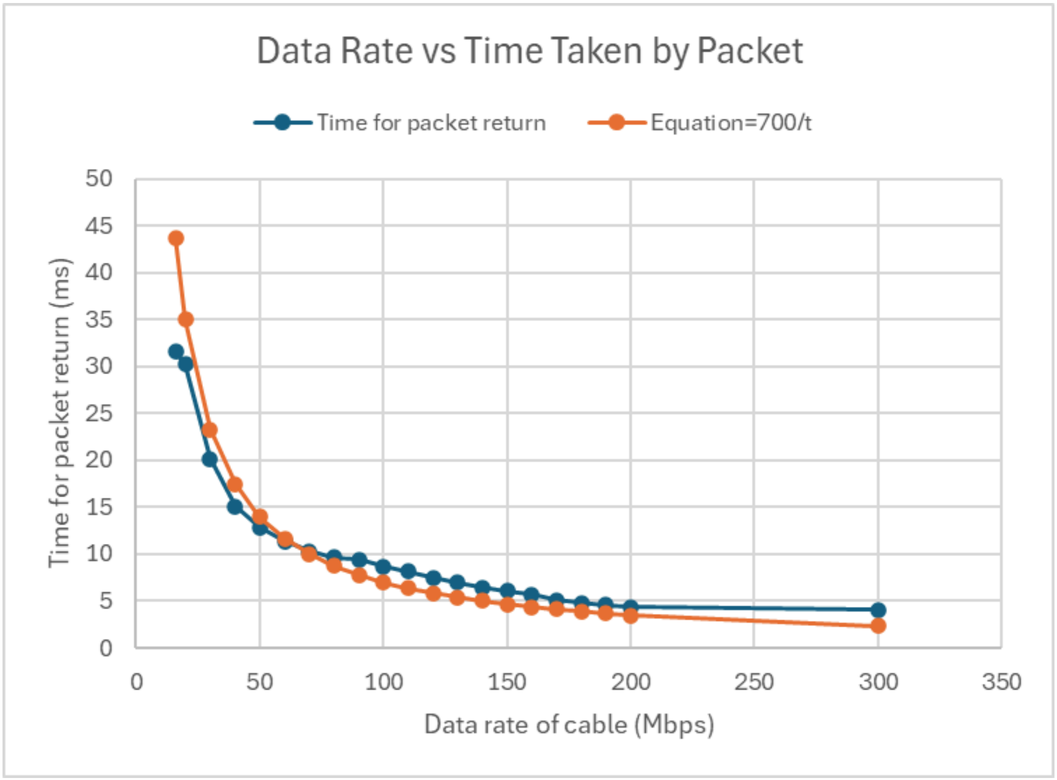

Figure 11: Graph of the relationship of data rate compared to packet transmission rate

The following graph depicts the performance of the Cat5, Cat5e, and Cat6 cables as the number of daemon bots increases from 10 bots to 100 bots compared to the equation $\frac{700}{t}$. The two graphs resemble each other closely which leads us to believe that the relationship between the data rate and the time for a packet to return is inversely proportional.

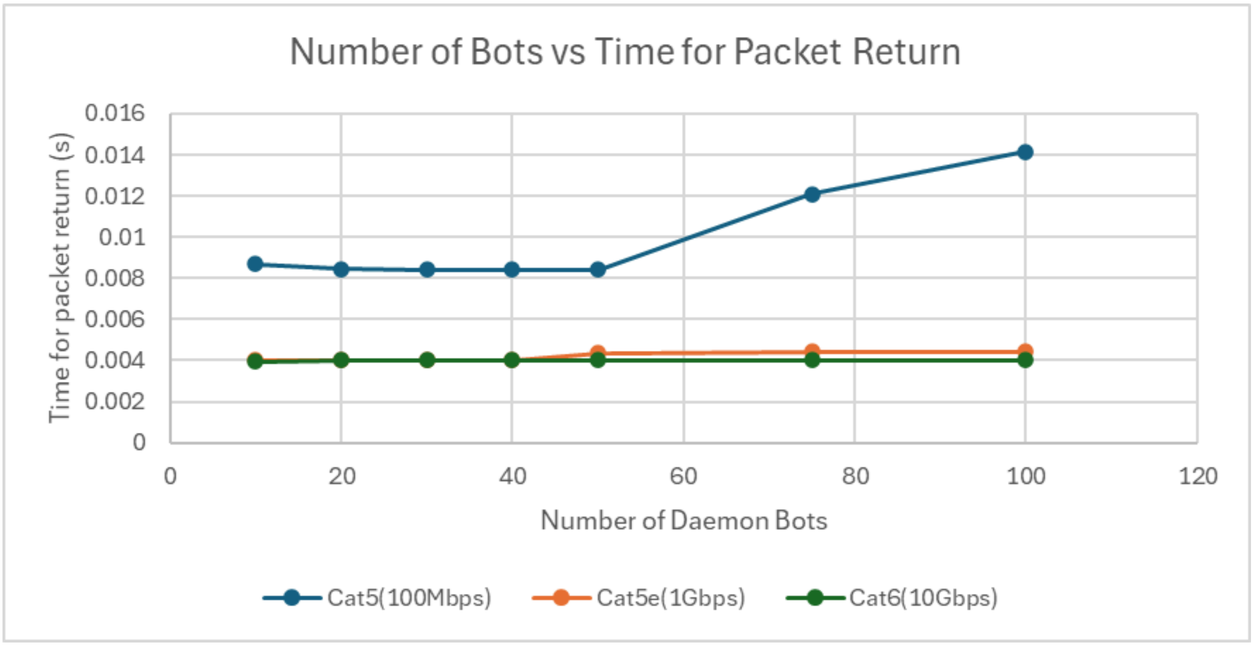

Figure 12: Performance graph of different ethernet cables vs an increasing number of daemon bots

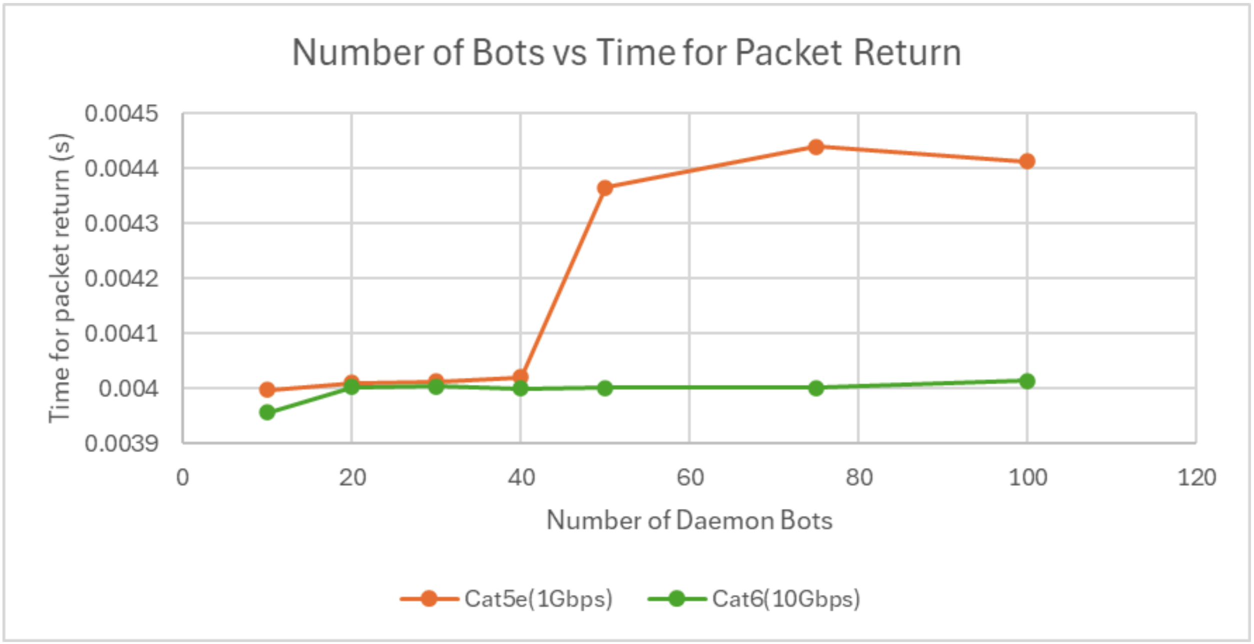

Figure 13: Performance graph of different ethernet cables vs an increasing number of daemon bots (Cat5e and Cat 6 zoom in)

Figures 12 and 13 show how drastic an impact increasing the number of bots had on the Cat5 cable, being noticeably worse than Cat5e and Cat6 with any amount of bots. Cat5 also showed the greatest slowdown when the number of bots was increased, being 39% slower with 100 bots compared to

10. While the 1Gbps Cat5e cable experienced a slowdown from the increasing number of bots it didn't suffer as severely as the 100Mbps Cat5 cable. We can also see very clearly that the 10Gbps Cat6 cable was barely slowed down at the levels of bots that we tested showing how effective this cable is at mitigating the effects of the DDoS attack.

Discussions

It is safe to say from our two data tables that an ethernet cable with a high transmission rate relative to the attack rate of the daemon bots is more resistant to the effects of network congestion than an ethernet cable with a lower transmission rate. The difference in performance between the Cat5 cable and the Cat6 cable became more and more noticeable as we increased the number of bots. We attempted to run the code with 150 bots and 200 bots because we wanted to see if the Cat6 cable would experience any performance issues, however, NetAnim would always crash. We theorize that if we were able to run the code with enough bots that the Cat6 cable would experience delays, we would see the same inverse relationship between the number of bots and the time it takes for the packet to return as we did with lower data rates.

Due to computing power and the efficiency of our code, we are unable to realistically simulate a DDoS attack since on average in 2022, DDoS attacks in real life attack at a rate of 5.17 Gbps [16]. A realistic DDoS attack would require 400-2000 bots. In our simulation, the number of bots attacking is also constant with the attack rate also being constant. Realistically, the attack would start at low rates with the number of bots linearly ramping up [15].

Even though data rate is the most significant difference between different types of ethernet cables, there are other characteristics that distinguish them from each other such as shielding, maximum bandwidth, and range. For example, Cat5 cables are generally unshielded while Cat6 cables have a spline in the center of the cable that mitigates crosstalk [8]. We are unable to recreate shielding in ns-3 due to it being a physical layer attribute. We were also unable to find a parameter for PointToPoint that corresponded to maximum bandwidth so we are also unable to account for that parameter. We are also unable to simulate range in ns-3 as it's also a physical layer attribute. We could implement a propagation delay however this delay would be constant and not an accurate representation of real-life conditions. Bandwidth could affect the data rate of the PointToPoint connection in the ns-3 simulation since bandwidth dictates the number of bits that are allowed to transmit per second [14]. Since there does not seem to be a parameter for maximum bandwidth, we do not know if the connection even has a maximum bandwidth.

For future work, we want to be able to observe delays in the Cat6 cable so we would want to refine our code and make it more efficient so that we can run more bots and/or increase the attack rate. Additionally, we can try running our code on a better computer. We would also want to change the code so that it accounts for other parameters that vary among Ethernet cables such as bandwidth as mentioned before. We could look into the BandWidthManager class in ns-3 to try to implement the bandwidth parameter. We would also want to try using alternate simulators such as Riverbed which could possibly be better at simulating different types of ethernet cables since ns-3 does not have a lot of features for physical layer simulations. Lastly, we want to finalize our code for wireless connections so that we would be able to compare how wireless connections such as WiFi respond to DDoS attacks compared to wired connections. Problems with the code for wireless connections are further discussed in Appendix II.

Conclusion

With the world becoming more digitized, DDoS attacks are becoming a common threat. Businesses are disturbed and lose money, and users are unable to access their services. While DDoS protection is generally applied at the network and application layer [18], we wanted to see if users can do anything to protect themselves from an attack. Ethernet cables are commonly used by individuals to connect to the internet so we wanted to see if using a better ethernet cable mitigates the effects of an incoming DDoS attack.

In conclusion, this paper explored the effects of how DDoS attacks affected ethernet cables of different types. We simulated a DDoS attack in ns-3 and measured how long it would take a packet to traverse through a congested network and return to the client. We varied the number of bots and the data rate of the cable to look for the relationship between the DDoS attack rate and the data rate and found that the relationship is approximately 1/t. While we were able to see delays in the Cat5 cable, we were unable to produce a DDoS attack powerful enough to cause a delay in the Cat6 cable. In the future, we would want to either run our program on a more powerful computer or make our program more efficient. Overall, we were able to learn more about how Ethernet functions, how DDoS attacks are carried out, their negative effects on networks, and how to better use ns-3.

References

[1] N. Qwasmi, F. Ahmed, and R. Liscano, “Simulation of DDOS Attacks on P2P Networks" in 2011 IEEE International Conference on High Performance Computing and Communications, Banff, AB, Canada, 02-04 Sept. 2011. DOI: 10.1109/HPCC.2011.86

[2] imperva, “DDoS Attacks.” imperva.com. https://www.imperva.com/learn/ddos/ddos-attacks/ (accessed Apr. 6, 2024).

[3] cloudflare, “What is a DDoS attack?” cloudflare.com. https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/ (accessed Apr. 6, 2024).

[4] cloudflare, “Smurf DDoS attack.” cloudflare.com. https://www.cloudflare.com/learning/ddos/smurf-ddos-attack/ (accessed Apr. 6, 2024).

[5] cloudflare, “SYN flood attack.” cloudflare.com. https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/ (accessed Apr. 6, 2024).

[6] cloudflare, “UDP flood attack.” cloudflare.com https://www.cloudflare.com/learning/ddos/udp-flood-ddos-attack/ (accessed Apr. 6, 2024).

[7] redwolfsecurity, “Simulating DDoS Attacks - how many attackers is enough?” redwolfsecurity.com. https://www.redwolfsecurity.com/simulating-ddos-attacks-many-attackers-enough/ (accessed Apr. 6 2024).

[8] S. Cook, “20+ DDoS attack trends and statistics in 2024: The rising threat.” comparitech.com. https://www.comparitech.com/blog/information-security/ddos-statistics-facts/ (accessed Apr. 6, 2024).

[9] C. Shieh et al., “Detection of Unknown DDoS Attacks with Deep Learning and Gaussian Mixture Model,” MDPI Applied Science, vol. 11, no. 11, pp. 5213, Jun. 2021, doi: 10.3390/app11115213

[10] vcelink, “Ethernet Cables 101: The Basics of Network Cabling.” vcelink.com. https://www.vcelink.com/blogs/focus/ethernet-cable-101#:~:text=Ethernet%20cables%20come%20in%20different,to%20electromagnetic%20interference%20(EMI). (accessed Apr. 6 2024).

[11] D. Harris. “Ethernet Cable Shielding Types.” truecable.com. https://www.truecable.com/blogs/cable-academy/ethernet-cable-shielding-types (accessed Apr. 6, 2024).

[12] sig4cai, “ Braided, foil, or spiral: which is best for your specific application?” sig4cai.com. https://www.sig4cai.com/how-to-pick-the-right-cable-shielding-for-your-cable-assembly/#:~:text=Foil%20is%20lighter%2C%20less%20expensive,less%20effective%20at%20lower%20frequencies. (accessed Apr 6. 2024)

[13] blackbox, “What’s the difference between CAT5e and CAT6?” blackbox.co.uk. https://www.blackbox.co.uk/gb-gb/page/43869/Resources/Technical-Resources/Black-Box-Explains/Cop

per-Cable/Category-5e-And-6#:~:text=Not%20only%20does%20CAT6%20provide,and%20higher%20data%20transmission%20rates. (accessed Apr. 6, 2024).

[14] lightguru, “CATEGORY 6 (CAT6) Cable UTP Data Blue PVC with Spline (Per meter).” lightguru.sg. https://lightguru.sg/product/category-6-cat6-cable-utp-data-blue-pvc-with-spline/ (accessed Apr. 6, 2024).

[15] vcelink, “Choosing the Best Ethernet Cable for Gaming in 4 Steps.” vcelink.com. https://www.vcelink.com/blogs/focus/best-ethernet-cable-for-gaming (accessed Apr. 6, 2024).

[16] TRIPP LITE, “Ethernet Cables Explained.” tripplite.eaton.com. https://tripplite.eaton.com/products/ethernet-cable-types (accessed Apr. 6, 2024).

[17] DDoSim.cc. (2020), S. Upadhyay. Accessed: Mar. 8, 2024, [C++, online], Available: https://gist.github.com/Saket-Upadhyay/c4c702716233cab91eb31b6d547aaeab

[18] AbdullahBell et al., “Azure Ddos Protection frequently asked questions,” Microsoft Learn, https://learn.microsoft.com/en-us/azure/ddos-protection/ddos-faq (accessed Apr. 12, 2024).

Contributions

	Joshua Ma	Josh Wong	Andrew Speers
References and literature review	0%	50%	50%
Project website	100%	0%	0%
Simulation scenarios, implementation, analysis, and discussion of simulation results	33%	33%	33%
Project presentation	33%	33%	33%
Written final report	33%	33%	33%
Total	33%	33%	33%

Appendix I: Wired DDoS Attack Source Code

```
//ddos.cc
//Credit to S.Upadhyay, Original Code Repository Available at
//https://gist.github/Saket-Upadhyay/c4c702716233cab91eb31b6d547aaeab
//Reference: [17]
#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));

      // Install connections between node and bots
    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");

       // Setup NetAnim Configuration
    anim.SetMaxPktsPerTraceFile(5000000);
    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);
    }

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

Appendix II: Unfinished Wireless DDoS Attack Code and Progress

```
//ddos_wireless.cc
//Credit to TA Soruosh for help creating parts of this code
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/wifi-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/netanim-module.h"

using namespace ns3;

//Experimental Parameters
#define TCP_SINK_PORT 9000
#define UDP_SINK_PORT 9001
#define MAX_BULK_BYTES 100000
#define DDOS_RATE "20480kb/s"
#define MAX_SIMULATION_TIME 10.0
#define NUMBER_OF_BOTS 10

NS_LOG_COMPONENT_DEFINE("WiFiNetworkDDoSAttack");

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

    Time::SetResolution(Time::NS);

       // Crease Wifi Nodes
    NodeContainer wifiStaNodes;
    wifiStaNodes.Create(NUMBER_OF_BOTS);
    NodeContainer wifiApNode;
    wifiApNode.Create(1);

       // Create Wifi Channels
    YansWifiChannelHelper channel = YansWifiChannelHelper::Default();
       // Error Fixed
       // Changed this from YansWifiPhyHelper phy = YansWifiPhyHelper::Default();
    YansWifiPhyHelper phy;
    phy.SetChannel(channel.Create());

       // Wifi Manager
    WifiHelper wifi;
       // Changed from Aarf to Ideal
    wifi.SetRemoteStationManager("ns3::IdealWifiManager");
       //wifi standard 802.11ac may support what we want
       //wifi.SetStandard (WIFI_STANDARD_80211ac);

       //Wifi mac address for STA nodes
    WifiMacHelper mac;
    Ssid ssid = Ssid("wirelessNetwork");
    mac.SetType("ns3::StaWifiMac",
                "Ssid", SsidValue(ssid),
                "ActiveProbing", BooleanValue(false));
       //Install Wifi STA nodes
    NetDeviceContainer staDevices;
    staDevices = wifi.Install(phy, mac, wifiStaNodes);

       //Wifi mac address for AP node
    mac.SetType("ns3::ApWifiMac",
                "Ssid", SsidValue(ssid));
       //Install Wifi AP node
    NetDeviceContainer apDevices;
    apDevices = wifi.Install(phy, mac, wifiApNode);
       // Set network mobility
    MobilityHelper mobility;
    mobility.SetPositionAllocator("ns3::GridPositionAllocator",
                                  "MinX", DoubleValue(0.0),
                                  "MinY", DoubleValue(0.0),
                                  "DeltaX", DoubleValue(5.0),
                                  "DeltaY", DoubleValue(10.0),
                                  "GridWidth", UintegerValue(3),
                                  "LayoutType", StringValue("RowFirst"));

    mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
                              "Bounds", RectangleValue(Rectangle(-50, 50, -50, 50)));
    mobility.Install(wifiStaNodes);

    mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
    mobility.Install(wifiApNode);
       //Install Wifi AP and STA nodes on the internet stack
    InternetStackHelper stack;
    stack.Install(wifiApNode);
    stack.Install(wifiStaNodes);
       // Assign IP address to WIfi AP and STA nodes
    Ipv4AddressHelper address;
    address.SetBase("10.1.3.0", "255.255.255.0");
    Ipv4InterfaceContainer staNodeInterfaces = address.Assign(staDevices);
    Ipv4InterfaceContainer apNodeInterface = address.Assign(apDevices);

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

    ApplicationContainer onOffApp = onoff.Install(wifiStaNodes);
    onOffApp.Start(Seconds(1.0));
    onOffApp.Stop(Seconds(MAX_SIMULATION_TIME));

    // Sender Application
    BulkSendHelper bulkSend("ns3::TcpSocketFactory",
InetSocketAddress(apNodeInterface.GetAddress(0), TCP_SINK_PORT));
    bulkSend.SetAttribute("MaxBytes", UintegerValue(MAX_BULK_BYTES));
    ApplicationContainer bulkSendApp = bulkSend.Install(wifiStaNodes.Get(0)); //
Assuming first STA as a legitimate user
    bulkSendApp.Start(Seconds(0.0));
    bulkSendApp.Stop(Seconds(MAX_SIMULATION_TIME));

    // PacketSink on receiver side (AP)
    PacketSinkHelper UDPsink("ns3::UdpSocketFactory",
                             Address(InetSocketAddress(Ipv4Address::GetAny(),
UDP_SINK_PORT)));
    ApplicationContainer UDPSinkApp = UDPsink.Install(wifiApNode);
    UDPSinkApp.Start(Seconds(0.0));
    UDPSinkApp.Stop(Seconds(MAX_SIMULATION_TIME));
    // TCP Sink Application on AP side
    PacketSinkHelper TCPsink("ns3::TcpSocketFactory",
                             InetSocketAddress(Ipv4Address::GetAny(), TCP_SINK_PORT));
    ApplicationContainer TCPSinkApp = TCPsink.Install(wifiApNode);
    TCPSinkApp.Start(Seconds(0.0));
    TCPSinkApp.Stop(Seconds(MAX_SIMULATION_TIME));

    Ipv4GlobalRoutingHelper::PopulateRoutingTables();

    // NetAnim configuration
    AnimationInterface anim("DDoSim.xml");
    anim.SetMaxPktsPerTraceFile(5000000);
    for (uint32_t i = 0; i < wifiStaNodes.GetN(); ++i) {
        anim.SetConstantPosition(wifiStaNodes.Get(i), 5 * i, 30);
    }
    anim.SetConstantPosition(wifiApNode.Get(0), 10, 10);

    Simulator::Run();
    Simulator::Destroy();

    return 0;
}
```

Above is our unfinished code in an attempt to simulate a DDoS attack on a wireless network. Upon receiving part of this code from TA Soroush, we faced numerous build errors. We were able to remedy the build errors and managed to build the code successfully, however, when trying to compile and run the simulation, we got into a state where it appeared the code got stuck. When we open the xml file that was created from the wireless attack, we get the following in NetAnim.

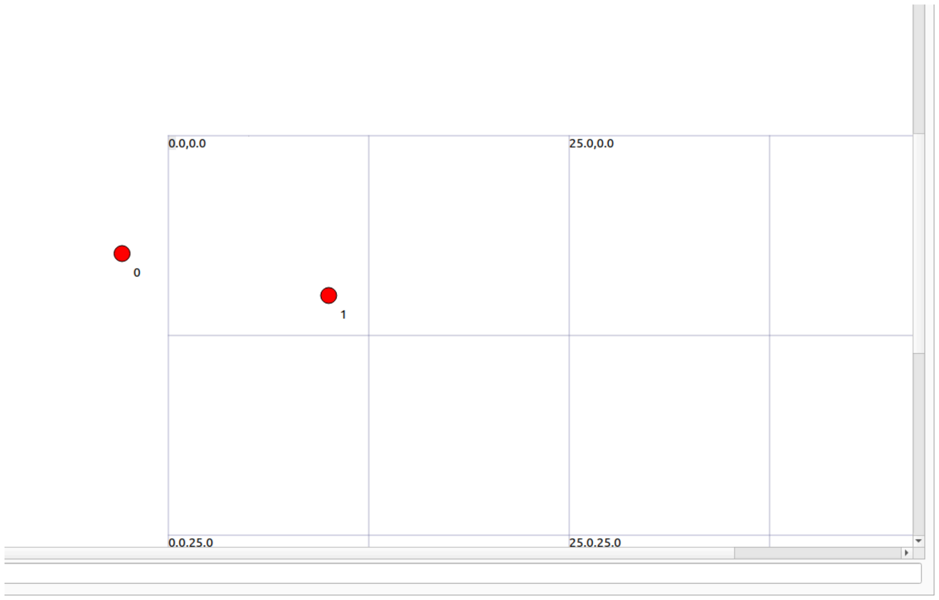

Figure 14: NetAnim of Unfinished Wireless DDoS Code

When we run the animation, it appears that the simulation creates the WiFi STA and AP nodes correctly but due to the mobility of the nodes, their position is not constant throughout the duration of the simulation. Instead, it appears that the nodes “jump” around the defined coordinate limits. As a result, a connection between the two nodes can never be made because of this “jumping” behavior. We believe this is what is causing our wireless simulation to hang because a constant connection between the nodes is never established. We think that if we were able to get the node locations constant, we can establish wireless connections between the nodes. And possibly implement the DDoS code that was created by S. Upadhyay [17].