



# **Simon Fraser University**

## **Engineering Science Department**

### **A Simulation Study of DDOS Attacks on Wired Networks**

ENSC 427: Communication Networks,  
Spring 2024  
Final Project  
URL: <https://elainexluu.github.io/ensc427ddos/>

Date: March 6th, 2024  
Instructor: Ljiljana Trajkovic

Prepared By:  
[ENSC] Malhi, Akashroop (301393341/ [asm19@sfu.ca](mailto:asm19@sfu.ca) )  
[ENSC] Ghatarora, Gurnek (301394646/ [gghataro@sfu.ca](mailto:gghataro@sfu.ca) )  
[ENSC] Luu, Elaine (301392121/ [ela64@sfu.ca](mailto:ela64@sfu.ca) )  
Group #4

## **Abstract**

With the increased reliance on wired networks, the vulnerability of these networks to malicious activities such as Distributed Denial of Service (DDoS) attacks becomes a significant concern. In this project, we utilize NS-3 to model accurate simulations of various DDos attack scenarios targeting the communication. The objective is to comprehend and contrast the impact of each attack on the performance of the targeted network.

The simulation setup allows us to deliberately generate and transmit attack traffic to the wired network. This makes it possible to observe and analyze various performance metrics, including throughput, packet loss, and checksum. Through comparing these metrics, we gain insights into the distinct behaviors and possible countermeasures for each type of DDoS attack.

The outcomes of the simulation provide valuable insights for the development of more efficient and effective algorithms, techniques, and procedures to mitigate these attacks in wired networks environments. By understanding the behaviors of these attacks and their effects on network performance, we can design better strategies to enhance the durability and security of wired networks against DDos threats.

## **Acknowledgments**

We extend our gratitude to Professor Ljiljana Trajkovic for her insightful lectures on Communication Networks. Additionally, we appreciate the diligent support and guidance provided by our TA, Soroush Oraki, who offered prompt feedback whenever required.

# Table of Contents

<b>Abstract.....</b>	<b>1</b>
<b>Acknowledgments.....</b>	<b>2</b>
<b>Table of Contents.....</b>	<b>3</b>
<b>Glossary.....</b>	<b>4</b>
<b>Introduction.....</b>	<b>5</b>
1.1 History of DDoS.....	5
1.2 Motivation and Scope.....	5
1.4 High-level Overview.....	6
1.5 Related Work.....	6
1.6 Motivation Behind DDoS Attacks.....	6
1.7 Example of Real Life Problem.....	7
1.8 Difference between DoS and DDoS.....	8
<b>Main Sections: Understand the components of a DDoS (Diving Deeper).....</b>	<b>9</b>
2.1 DDoS Attack Overview.....	9
2.2 Characteristics of Attacks.....	9
2.3 DoS Attack Methods (SYN/UDP/SMURF from single source).....	10
2.4 Queuing Algorithms.....	12
1. Drop Tail.....	12
2. Fair Queuing.....	12
3. Stochastic Fair Queuing.....	12
<b>Main Sections: Implementation.....</b>	<b>12</b>
2.5 NS-3.....	12
2.6 Wireshark.....	13
2.7 Simulation Scenarios.....	13
2.8 Topology.....	14
<b>Discussion and Conclusion.....</b>	<b>15</b>
3.1 Packet Loss.....	15
3.2 Checksum.....	17
3.4 Recommendations.....	21
<b>References.....</b>	<b>23</b>
<b>Contribution.....</b>	<b>25</b>
<b>Appendix:.....</b>	<b>26</b>
6.1 Code Listing.....	26

## **Glossary**

DDoS: Distributed Denial of Service

DoS: Denial of Service

TCP: Transmission Control Protocol

UDP: User Datagram Protocol

SYN: Synchronize

IP: Internet Protocol

IRC: Internet Relay Chat

ISP: Internet Service Provider

IoT: Internet of Things

HTTP: Hypertext Transfer Protocol

WiFi: Wireless Fidelity

WLAN: Wireless Local Area Network

NS-3: Network Simulator 3

# Introduction

## 1.1 History of DDoS

Originally, DDoS attacks were done with experimental purpose, with no ill-intent, or simply not powerful enough to have a large-scale threat to companies as we see today. The first attack occurred in 1974, with more bandwidth-based attacks arising due to the development of IRC in the 90's. In 1996, the ISP Panix became the target for one of the first large-scale DDoS attacks, having their servers overloaded by a SYN flood. Ensuring attacks would expose the threat potential of DDoS attacks and display their growing complexity, as in 1999 an attack was launched by a hacker using a tool called Trinoo, disabling the computer network of the University of Minnesota for over 48 hours using a UDP flood [6][7]. By the 2000s, attacks began targeting major companies, showing how the evolution of networks would inadvertently intensify the effectiveness of DDoS attacks [8].

## 1.2 Motivation and Scope

Being introduced to softwares such as Wireshark through class assignments, it was interesting to be able to see all the traffic traveling across networks. This raised curiosity on how this traffic would behave in reaction to DDoS attacks, and the different security measures specific networks would take in response.

The main scope of this project will be implemented with NS-3. We will be simulating DDoS attacks and analyze the negative effects it causes to the services for a client. Furthermore, we will compare the performance of networks after using various traffic patterns by changing the number of attackers and rates they attack at [10][11].

## 1.3 Project Idea/Objective

Our objective for this simulation is to examine the toll of a common UDP Flood attack on a large scale network to closely view the effects on a more realistic system containing sub-networks, users and a server. In our simulations we will be testing multiple magnitudes of DDoS attacks, from a lower volume to a higher volume of UDP packet flooding, as well as different numbers of bots. In theory, these two variables should display different levels of congestion, which we will measure with throughput and packet loss. We will also be analyzing checksum to see the corruption of packets in order to determine if the DDoS attacker was successful in hiding its own IP address, and see if their location was traceable or not. Following these metrics should allow us to accurately simulate a UDP flood on a large scale network, as we should be able to see if the broadcast channel was flooded with UDP packets as well as if those packets' source addresses were indeed unreachable (corrupted) due to spoofing [16][15][17][10].

## 1.4 High-level Overview

As mentioned, DDoS attacks can occur on different networks. NS-3 will be used for simulation, in order to analyze and decipher comparisons of the various attacks. With NS-3, we are able to deliberately create attacks and transfer attack traffic to a particular network. Our code generates XML files for NetAnim, an offline animator that aids in crafting realistic network topologies. Additionally, we produce PCAP files compatible with Wireshark, allowing us to examine critical metrics like throughput, checksums, and packet loss. Moreover, we are able to acquire insight and create possible countermeasures for the different types of DDoS attacks helping us develop efficient algorithms, techniques and procedures to counteract the attacks [3][5][11].

## 1.5 Related Work

i. Using Graphic Network Simulator 3 For DDos Attacks Simulation by Anatoliy, Balyk, Mikolaj Karpinski, Artur Naglik, Gulmira Shangytbayeva, and Ihor Romanets.

This paper discusses the application of a specific approach to simulating the performance of an HTTP server within a typical enterprise network under a Distributed Denial of Service (DDoS) attack using Graphical Network Simulator-3 (GNS3). The focus is on understanding how an HTTP server behaves and performs under such adverse conditions, allowing for the evaluation of potential vulnerabilities and the effectiveness of mitigation strategies [9].

ii. Modeling distributed denial of service attack in advanced metering infrastructure by Yonghe Guo, Chee-Wooi Ten, Shiyan Hu, and Wayne W. Weaver.

This paper explores the idea of a DDoS cyber attack on an advanced metering infrastructure (AMI). AMI essentially allows two-way communication between utilities and users, and allows remote communication between smart household appliances and these utilities, and here the authors analyze the effect on the latency, throughput, and response times under different attack scenarios. The results give insight on the pros and cons of the different wireless protocols used in AMI [15].

## 1.6 Motivation Behind DDoS Attacks

As the IoT and devices are constantly increasing in use within the world, it is significant for websites and servers to acquire strong security to protect data and prevent any breaches that could occur. Websites and servers that do not have secure protection are prone to web vulnerability attacks. With that being said, the increase of IoT sensors and connected devices allows attackers to perform DDoS attacks at an increasing rate [8].

DDoS attacks are becoming more large and complex and it is important to analyze the intentions behind a DDoS attack as it can cause server outages, making servers inaccessible to legitimate users and generating unnecessary stress to organizations. Here are some reasons as to why an attacker might perform DDoS attacks [12][13]:

- **Financial Gain**

DDoS attacks are often used for financial gain for the attacker. Attackers could launch DDoS attacks to extort money from companies in exchange for stopping the attack [12][8][13].

- **Politics**

DDoS attacks can also be used for political reasons. Attackers could launch DDoS attacks on a targeted website to protest against certain concepts, actions from the government, and organizations [12][13].

- **Revenge**

An attacker might also launch a DDoS attack for revenge against a person or organization [12][13].

However, there are defense mechanisms to prevent a DDoS attack from occurring. By analyzing and studying through DDoS preventions and attacking methods, we are able to better protect businesses and corporations from corruption and major downfalls such as:

- Lost businesses
- Lost of reputation
- Personal information exposure

## 1.7 Example of Real Life Problem

UDP flood attacks present a considerable threat to online gaming servers, primarily due to the real-time nature of gaming communication and the prevalent use of UDP for its low-latency attributes. These attacks can lead to severe consequences, including gameplay interruptions, extended downtime, and a competitive disadvantage for affected players. Moreover, the disruption in service can result in substantial loss of revenue for gaming service providers. Overall, UDP flood attacks jeopardize the stability and performance of online gaming servers, significantly impacting both players and service providers alike.



## 1.8 Difference between DoS and DDoS

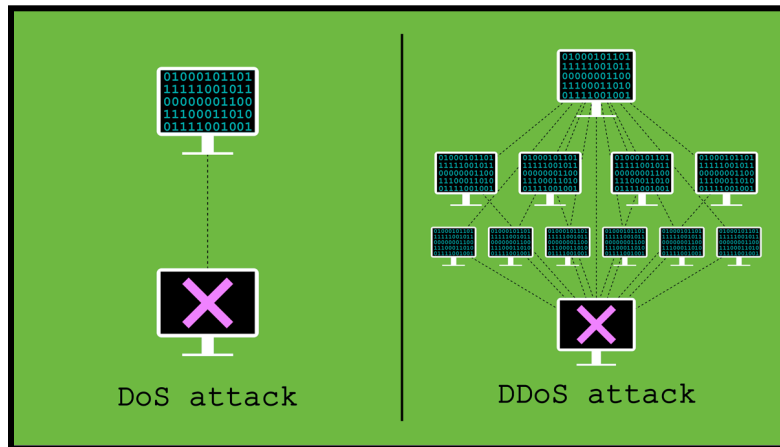


Figure 1: Dos attack versus DDoS attack conceptual diagram

	DoS	DDoS
<b>Description</b>	Attacks are launched from a single source [2].	Attacks, as the name suggests, are distributed across multiple sources [2].
<b>Method</b>	Flood a target system, with a massive amount of traffic, overwhelming its capacity and causing it to become inaccessible to legitimate users .	Attacks amplify the impact of the attack by utilizing the combined bandwidth and resources of the attacking systems, making them more difficult to mitigate [1].
<b>Impact</b>	Moderate	Vigorous
<b>Traceability</b>	More traceable due to their single-source nature	More challenging to trace back to individual attackers because they involve multiple distributed sources
<b>Speed</b>	Slow	Fast
<b>Recovery</b>	Attacks typically require less preparation and originate from a single source, it may be easier to identify the attacker's IP address	Attacks typically require more preparation due to their distributed nature, while recovery from DDoS attacks is often more challenging and time-consuming

## Main Sections: Understand the components of a DDoS (Diving Deeper)

### 2.1 DDoS Attack Overview

In a DDoS attack scenario, there are four components [1][11]:

1. **Real Attacker:** The individual or entity orchestrating the attack.
2. **Master:** Program hosted on a separate computer that manages and coordinates the attack daemons.
3. **Attack Daemons Agents:** Programs deployed across multiple host computers, controlled by the master program, which carry out the attack commands.
4. **Victim:** The target of the attack.

When the real attacker wishes to initiate the attack, they issue an execute command to the control master program, which then activates all the daemons under its control. These daemons subsequently carry out the attack against the victim [1][11].

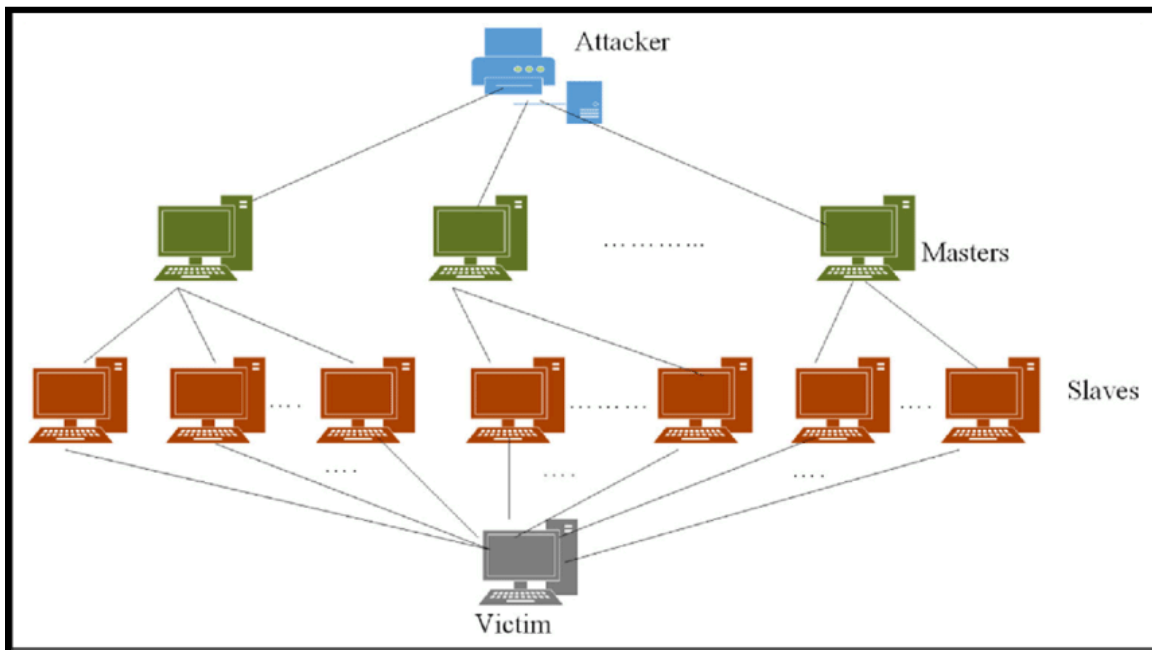


Figure 2: Conceptual Diagram of DDos [2]

### 2.2 Characteristics of Attacks

A Distributed Denial of Service (DDoS) attack has multiple ways to compromise network functionality and hinder user access. It involves flooding a network to diminish available bandwidth for legitimate users and disrupting connections between machines, impeding access to targeted services. These attacks may also be personalized, aiming to block a specific individual's

access to a service. In essence, DDoS attacks are orchestrated to disrupt services, whether by overwhelming a system or targeting a particular user, thereby compromising the stability and accessibility of the affected network or service [1][11].

### 2.3 DoS Attack Methods (SYN/UDP/SMURF from single source)

1. **Smurf:** a smurf attack is where the attacker sends a large amount of Internet Control Message Protocol (ICMP) echo traffic to Internet Protocol (IP) broadcast addresses. The attacker bombards the user's network with responses, overflowing its capacity which causes inaccessibility to legitimate users. In this case the attack will also spoof the user's IP address and request packets on their behalf, which is what causes the flooding when the networks reply with those packets [11].

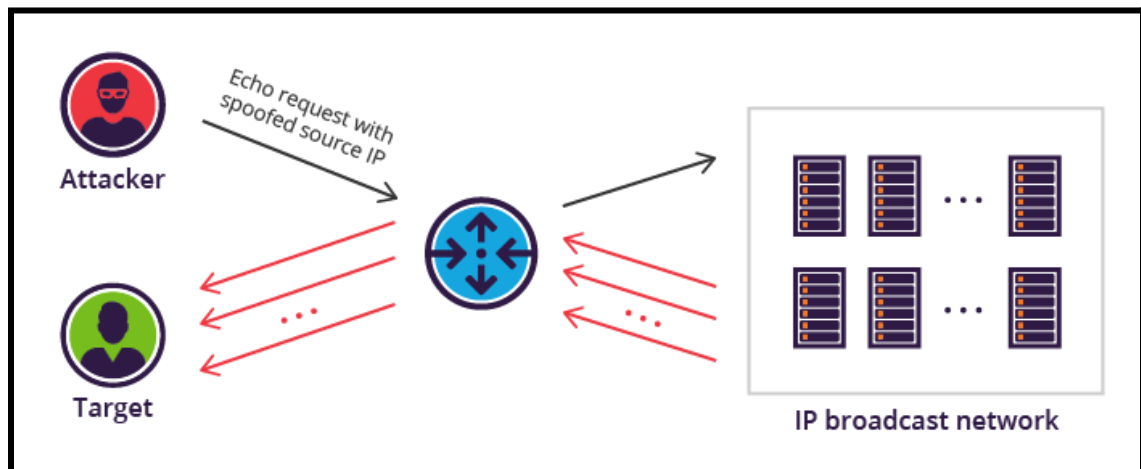


Figure 3: Smurf Flood Diagram [19]

2. **SYN Flood:** a SYN flood is when an attacker sends a flood of TCP SYN packets to a target server. This occurs when an attacker begins to send TCP connections but does not send a final ACK packet as a response, resulting in an incomplete handshake process [18].

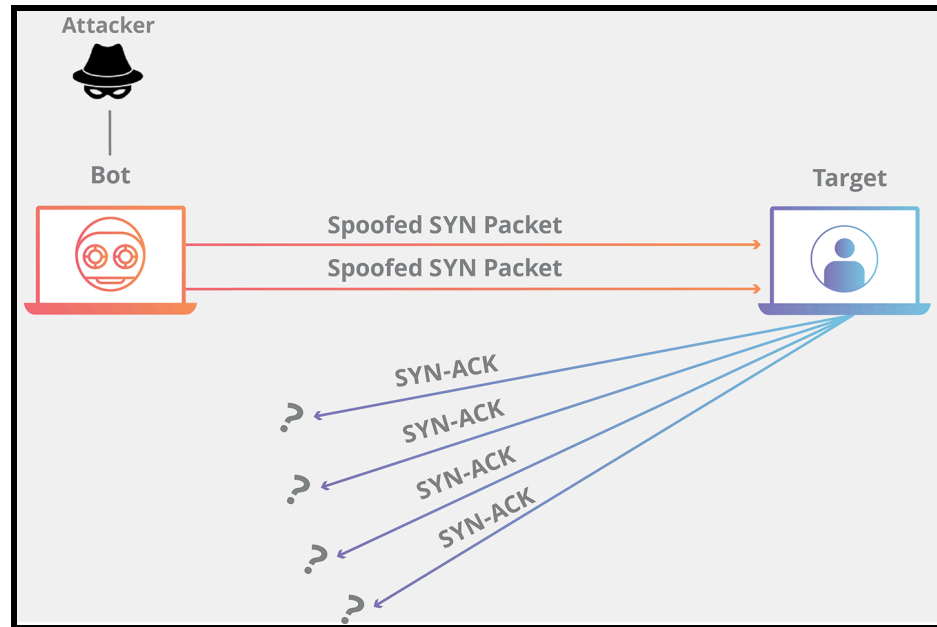


Figure 4: SYN Flood Attack [18]

3. **UDP Flood:** UDP flood is where an attacker sends a flood of User Datagram Protocol (UDP) packets to a server/network. UDP is connectionless which does not require any handshakes to start a connection. Therefore, the attack will overwhelm the target with a large amount of UDP packets, consuming the target's bandwidth and resources. This causes a decrease in performance for legitimate users [17].

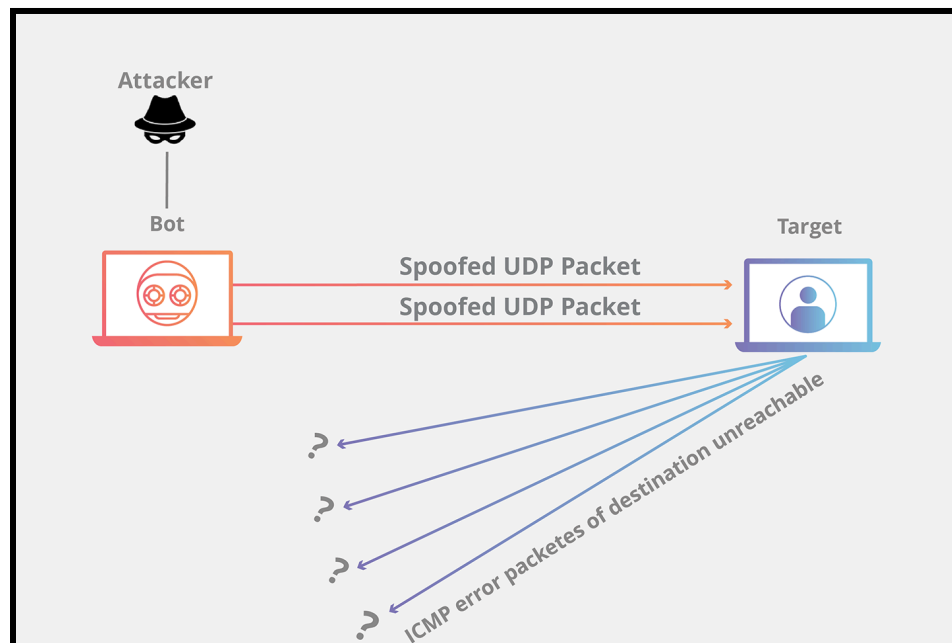


Figure 5: UDP Flood Attack [17]

In all cases the attacker may spoof their own IP address to hide its own identity or even impersonate another user.

## 2.4 Queuing Algorithms

### 1. Drop Tail

Drop tail is commonly used in routers and switches. In drop tail queuing, data buffers within routers and switches temporarily store packets before they are transported to their next destination. When space in the buffer becomes full and new packets arrive, packets at the end of the queue are dropped since there will be no space available to accommodate for the new packets [4][14].

### 2. Fair Queuing

Fair queuing aims to separate queues for the incoming flow of packets. As the name suggests, it allocates resources fairly among competing flows of data packets ensuring that each packet receives an equal amount of bandwidth or buffer space [4][14].

### 3. Stochastic Fair Queuing

Stochastic Fair Queuing is a queuing algorithm used in packet-switched networks, particularly in routers and switches. It is designed to allocate bandwidth fairly among different traffic flows while also providing some level of isolation between flows to prevent one flow from monopolizing resources at the expense of others [4][14].

## **Main Sections: Implementation**

### 2.5 NS-3

NS-3 stands as an open-source platform simulator designed for educational purposes and collaborative research endeavors. Focused primarily on wired and wireless networks, internet protocols, routing algorithms, and traffic models, NS-3 enables researchers to contribute to its development and leverage its capabilities for experimentation. This simulator proves invaluable as it allows users to conduct experiments that may be challenging or unfeasible with real-world systems. Within NS-3, users can access a multitude of libraries under open-source licensing, accessible through the wiki, contributed code page, and GitLab. NS-3 is implemented in C++, offering a robust foundation for simulations, and it further facilitates ease of use with Python bindings, streamlining scripting and configuration tasks. However, prior to utilizing NS-3, users are expected to possess a fundamental understanding of socket programming, including TCP/IP and multicast sockets, to maximize their efficiency and effectiveness within the platform.

## 2.6 Wireshark

Wireshark is an open-source packet analyzer known for its comprehensive capabilities in inspecting network traffic. Supporting an extensive array of protocols, Wireshark meticulously dissects each packet, revealing vital information such as the source and destination addresses, protocol type, and even looking into payload content. This data allows network administrators and developers, enabling them to conduct in-depth analysis of network performance, swiftly detect security threats, and meticulously debug network protocols and applications. With its user-friendly interface and robust feature set, Wireshark remains a go-to tool for professionals seeking exceptional insights into their network's behavior and security posture.

## 2.7 Simulation Scenarios

Scenarios		
Test Case #	Number of bots	DDoS Attack Rate (Mbps)
1	0	0 Mbps
2	3	0.909 Mbps
3	3	200 Mbps
4	20	0.909 Mbps
5	20	200 Mbps

## 2.8 Topology

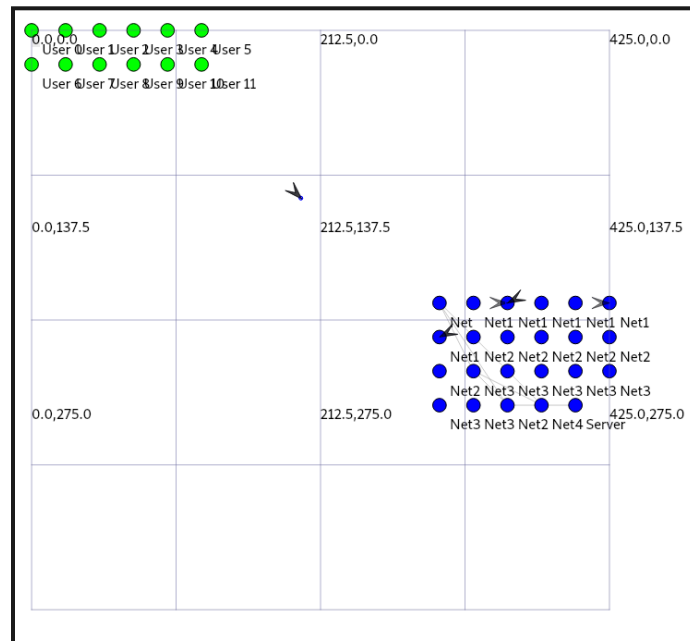


Figure 6: Zero Bots Topology

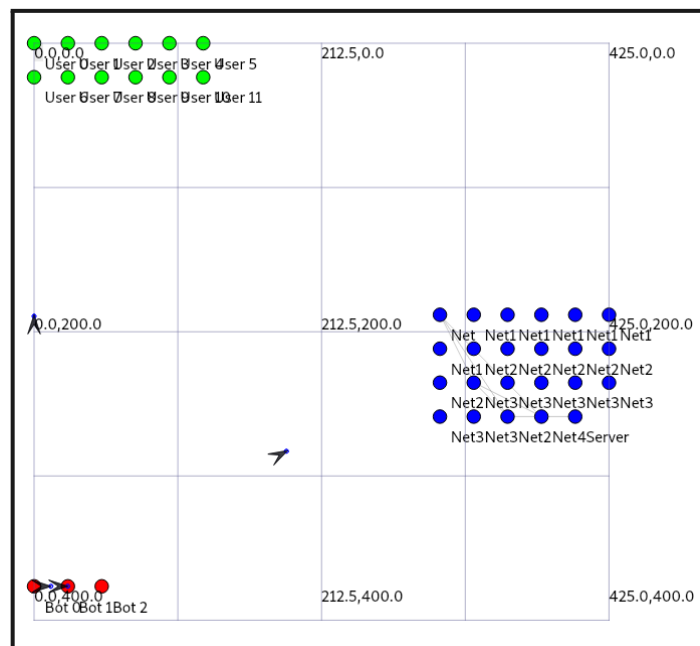


Figure 7: Three Bots Topology

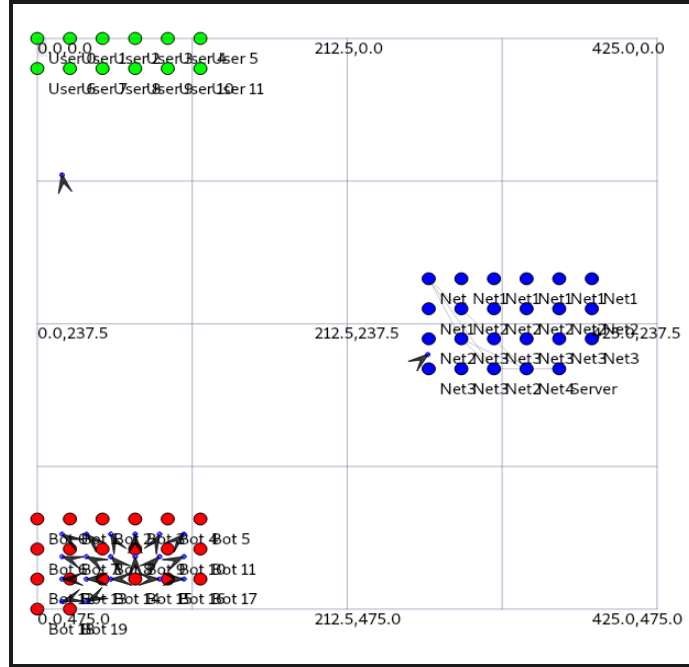


Figure 8: Twenty Bot Topology

Our basic topology consists of three main sections, being the real users, the attack bots, and the network nodes which also contains a server node. The packets being sent out are represented as arrows to indicate the traffic flow. A single bot node will inject corrupted UDP packets into the traffic flow between the target user and its network nodes in order to flood their broadcast channels. Displayed above are three different topologies for our main scenarios, displaying different numbers of bots, including zero bots (Figure 6), three bots (Figure 7), and twenty bots (Figure 8). In the case of zero bots, the flow of traffic is only between users and networks, and represents the typical flow of traffic. Including bots, shows a single bot instructing other bots to send out corrupted UDP packets in order to flood the network. The more bots there are, the more packets are being sent out at a time.

## Discussion and Conclusion

### 3.1 Packet Loss

Due to congestion, packet loss can occur due to deliberate attacks such as UDP floods. In our simulation using ns-3, we observed a significant increase in packet loss in Wireshark traces following a UDP flood. UDP floods involve sending a large volume of UDP packets to a target network or server, overwhelming its capacity to process and forward incoming traffic. Unlike TCP, UDP does not include built-in mechanisms for ensuring reliable delivery, making it particularly vulnerable to flooding attacks. As a result, excessive UDP traffic can quickly lead to packet loss, as network devices drop packets to alleviate congestion and prevent system overload. The impact of packet loss from a UDP flood extends beyond network congestion. It



can disrupt the normal operation of critical services and applications, leading to service outages, degraded performance, and compromised user experiences. In real-time communication applications such as voice communication and video streaming, packet loss can manifest as jitter, stuttering, or complete loss of audio/video data, significantly impairing the quality of the user experience. Furthermore, ongoing monitoring and analysis using tools like Wireshark allow us to identify patterns of packet loss and assess the effectiveness of our mitigation strategies. By understanding the root causes of packet loss and proactively addressing them, we can ensure the reliability and performance of our network infrastructure even in the face of challenging conditions such as DDoS attacks.

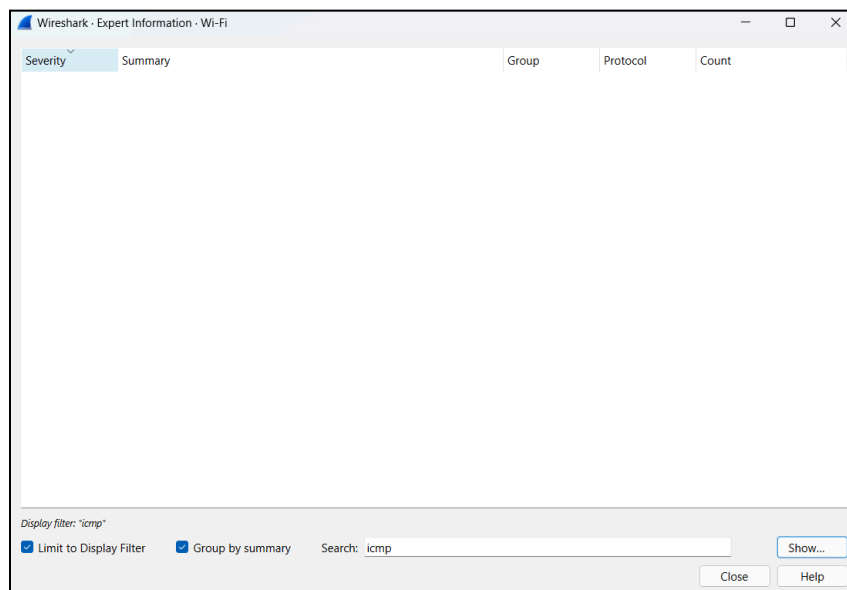


Figure 9: Zero bots, no sign of packet loss

The image shows the 'Wireshark - Expert Information - 909kb\_3Bots-2-0.pcap' window. The 'Summary' tab is selected. The table shows one entry with a warning severity.

Severity	Summary	Group	Protocol	Count
> Warning	Bad checksum	Checksum	ICMP	8509

Figure 10: Packet Loss for Three Bots and 0.909 Mbps

The image shows the 'Wireshark - Expert Information - 909kb\_20Bots-2-0.pcap' window. The 'Summary' tab is selected. The table shows one entry with a warning severity.

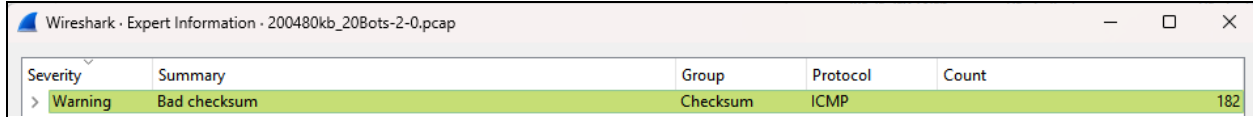
Severity	Summary	Group	Protocol	Count
> Warning	Bad checksum	Checksum	ICMP	10796

Figure 11: Packet Loss for Twenty Bots and 0.909 Mbps

The image shows the 'Wireshark - Expert Information - 200480kb\_3Bots-2-0.pcap' window. The 'Summary' tab is selected. The table shows one entry with a warning severity.

Severity	Summary	Group	Protocol	Count
> Warning	Bad checksum	Checksum	ICMP	344

Figure 12: Packet Loss for Three Bots and 200 Mbps



Severity	Summary	Group	Protocol	Count
> Warning	Bad checksum	Checksum	ICMP	182

Figure 13: Packet Loss for Twenty Bots and 200 Mbps

$$Packet\ Loss = \frac{\# of\ Bad\ Checksum\ Packets}{Total\ Packets\ in\ Simulation} * 100$$

Equation 1: Packet Loss Formula

Number of Attack Bots in Simulation	Packet Loss % (~1 Mbps)	Packet Loss % (~200 Mbps)
0	0 %	0 %
3	37.22%	46.48 %
20	40.9%	46.05%

#### Packet Loss Discussion:

The observed packet loss percentages align with our expectations for this simulation. As we scale up the number of bots and increase the intensity of the DDoS attacks, we're witnessing a corresponding rise in packet loss. Particularly noteworthy is the significant spike in packet loss rates exceeding 6% under both DDoS scenarios. These levels of packet loss are alarming, especially considering that in real-world scenarios, any packet loss exceeding 1% on gaming servers is typically deemed unacceptable. In the realm of online gaming, where real-time communication is paramount for smooth gameplay experiences, even minor instances of packet loss can lead to disruptive gameplay, adversely affecting user experience and engagement. Given the stringent demands of online gaming, maintaining a low packet loss rate is imperative to ensure seamless and uninterrupted gameplay sessions. The implications of high packet loss extend beyond mere inconvenience; they can have profound effects on player satisfaction, game performance, and even revenue streams for gaming companies. Players rely on consistent and reliable network connections to fully immerse themselves in the gaming experience, and any disruption due to packet loss can result in frustration and disillusionment. As technology continues to evolve and gaming experiences become increasingly immersive, ensuring low packet loss rates remains a foundational aspect of delivering optimal gameplay experiences to users worldwide.

### 3.2 Checksum

Checksum is a method that is used to detect errors and is implemented through an algorithm. This method provides information to the receiver about whether or not the full range of data has been transmitted successfully. A checksum involves a value that is calculated prior to

data transmission and is appended to the end of the data before it is sent. Once the data is received by the receiver, the checksum value will be recalculated and the recalculated value will be compared to the value that was appended to the data prior to transmission. Data corruption occurs when the values do not match; if the values do indeed match, it suggests that data was successfully delivered with no alterations.

```
> Frame 32: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
> Ethernet II, Src: 00:00:00_00:00:03 (00:00:00:00:00:03), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)
> Internet Protocol Version 4, Src: 10.1.7.2, Dst: 10.1.4.7
v Internet Control Message Protocol
  Type: 3 (Destination unreachable)
  Code: 3 (Port unreachable)
  v Checksum: 0x0000 incorrect, should be 0x7591
    > [Expert Info (Warning/Checksum): Bad checksum [should be 0x7591]]
    [Checksum Status: Bad]
    Unused: 00000000
  > Internet Protocol Version 4, Src: 10.1.4.7, Dst: 10.1.7.2
  > User Datagram Protocol, Src Port: 49153, Dst Port: 9001
```

Figure 14: Unreachable Destination Packet (invalid Checksum)

#### Checksum Discussion:

In the case of our simulation, all packets that were labeled with an unreachable destination had an incorrect checksum. Since we are doing a UDP Flood simulation, this is expected to see. The attacker had successfully spoofed the IP source address of the packet they were sending, and made it so the target could not trace back to the attacker, hence the unreachable destination type. The attack bots had successfully corrupted the packet in order to hide their location/identity. This fits the aspects of a UDP Flood.

### 3.3 Throughput

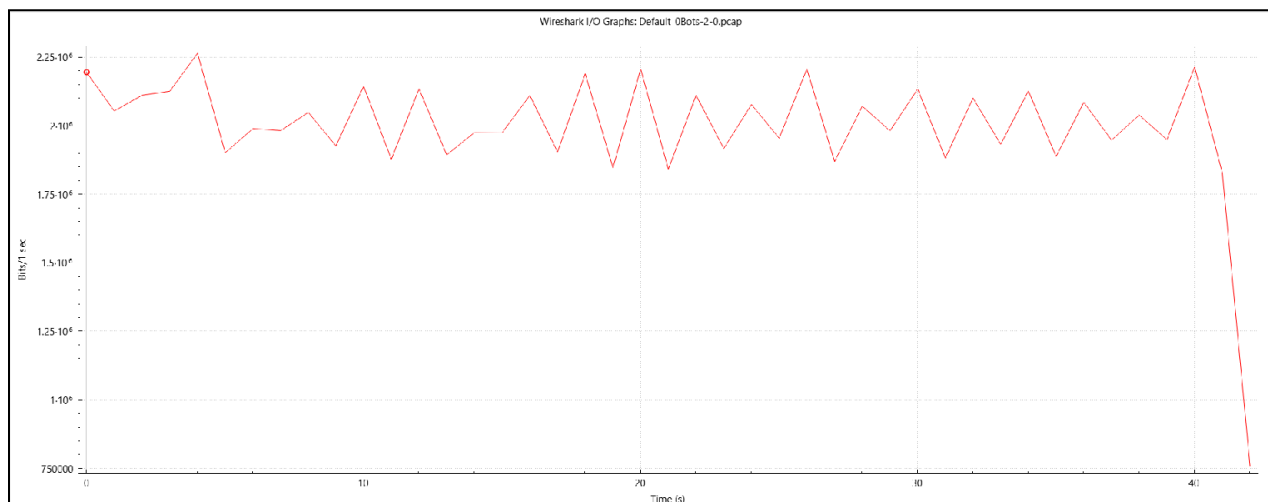


Figure 15: No Bots, Normal Operation

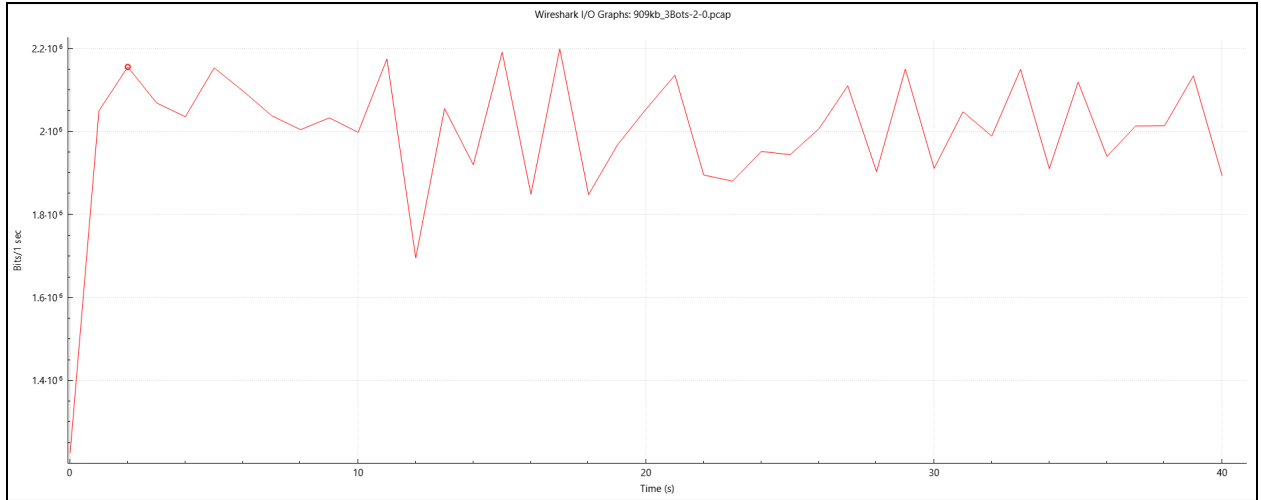


Figure 16: Added 3 bots, with low-rate (0.909 Mbps)

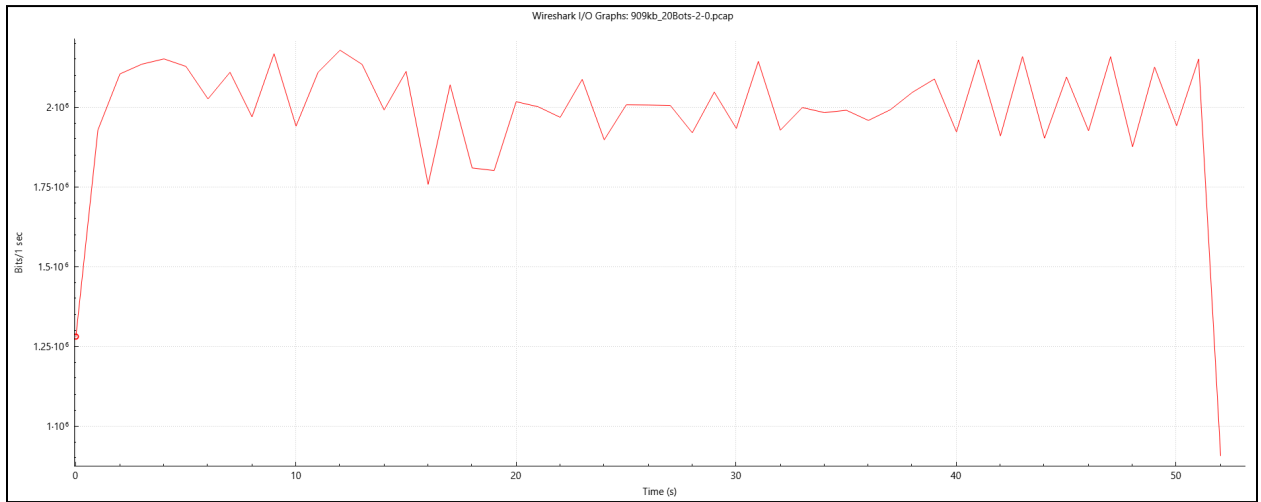


Figure 17: Added 20 bots, with low-rate (0.909 Mbps)

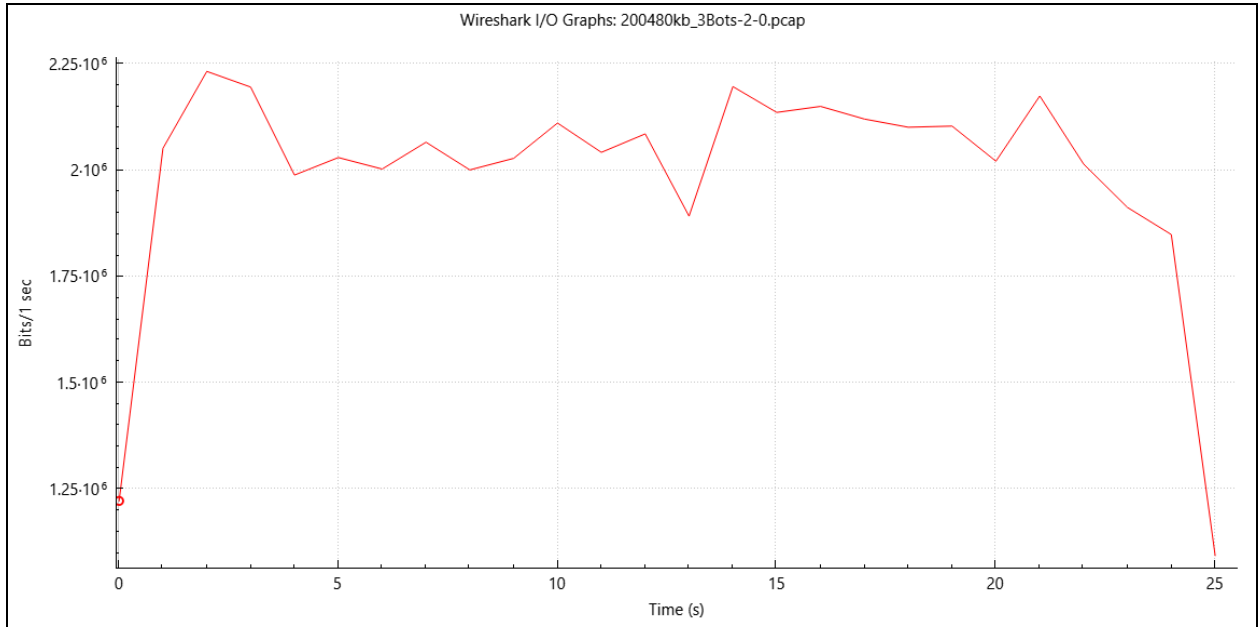


Figure 18: Added 3 bots, with high-rate (200 Mbps)

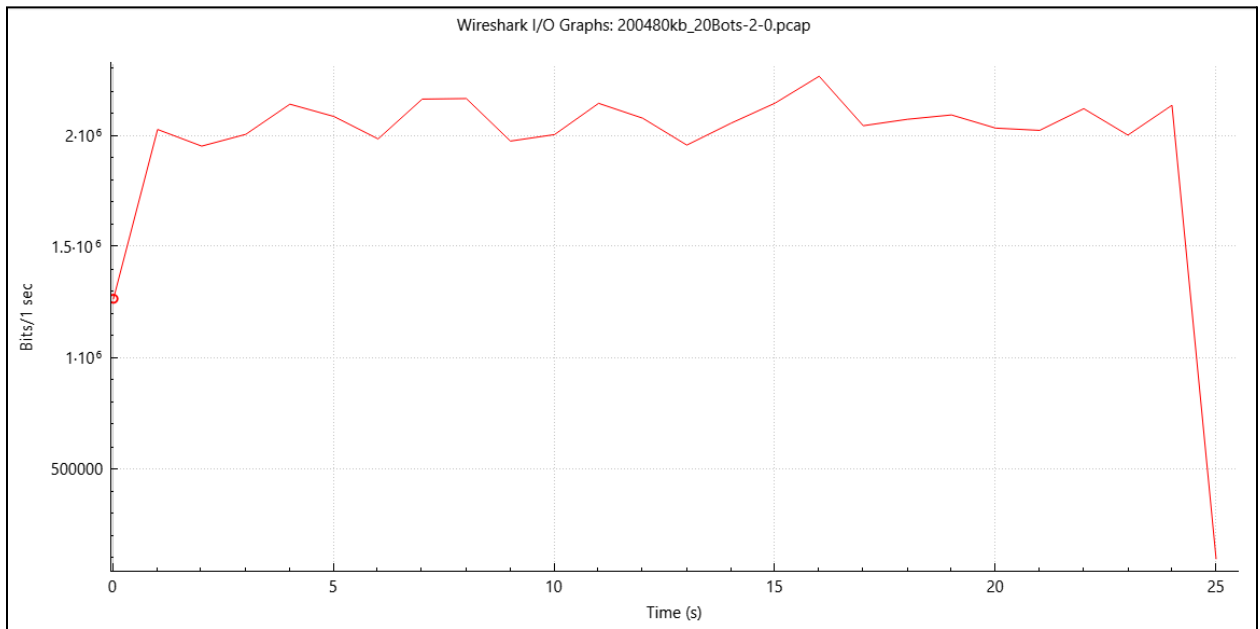


Figure 19: Added 20 bots, with high-rate (200 Mbps)

#### Simulation Results/Discussion:

During our simulations, the key variables under observation were the DDoS rate and the quantity of attack bots deployed. To establish a baseline for comparison across all simulations, we initially observed the normal operation of a large-scale network featuring multiple

sub-networks, a central server, and numerous users (Figure 15). This served as our reference point for assessing the impact of DDoS attacks.

When simulating an attack scenario with only three bots operating at a relatively low DDoS rate of 0.909 Mbps (Figure 16), equivalent to approximately 1 Mbps, the results revealed numerous abrupt drops in bits per second (bps), indicating periods of heightened congestion within the broadcast channel. In contrast, the simulation without attackers exhibited a slightly more consistent flow of data transmission, resulting in a smoother graph. This subtle variation in performance aligns with expectations for this simulation, considering the limited number of bots employed and their lower volume of transmitted data per second. These preliminary findings underscore the importance of further investigation into the effects of varying attack intensities and bot quantities on network performance and stability.

Bringing the number of attack bots up to twenty, with the same DDoS rate (Figure 17) results in roughly the same amount of drops, however the bits per second remain stuck at lower values for longer (for example between time 18s and 20s). We can infer that the channel is remaining congested for a longer amount of time with more bots added, however the congestion amount itself seems roughly the same as three bots. Once again, due to the low volume of bits per second, we do not expect a large amount of variation in the graphs.

Our next step is to simulate the same previous two scenarios at a higher DDoS rate at 200480 kbps or ~200 Mbps which is closer to the average rate of real life DDoS attacks. For three bots at this volume (Figure 18), we see a surprisingly smoother graph. The output seems roughly the same, if not smoother for the case of 20 bots (Figure 19). This is not at all what our expectations were. The graphs seem to remain consistent with bps around  $2 \times 10^6$  to  $2.25 \times 10^6$ , which was roughly the same for our lower rate DDoS attacks, and the drops in volume do not seem to be any more drastic. Our expectation would be that with a much higher rate of packets being sent out, there would be a much lower bps, with many more drops in volume, however this does not seem to be the case. Potential reasons may be due to an error in our code or the amount of time running the simulation causing the sample size of the packets to change.

### 3.4 Recommendations

Our initial project proposal aimed to implement DDoS attacks on wireless networks. However, we encountered several challenges along the way. One major issue was the occurrence of segfaults during the development process. These unexpected crashes significantly impeded our progress and required extensive debugging efforts to identify and resolve the underlying causes. Additionally, we found that debugging consumed a considerable amount of time, diverting resources away from other critical aspects of the project. As a result, our timeline for implementing and testing DDoS attacks on wireless networks was adversely affected.

For future improvements, we aim to enhance our DDoS simulations by incorporating realistic queuing algorithms such as droptail, stochastic fair queuing, and fair queuing. This approach will better reflect real-life scenarios, as larger companies typically employ

sophisticated routing algorithms and security measures to mitigate DDoS attacks. However, NS-3 has inherent limitations in terms of packet handling capacity, restricting the range of results and insights we can obtain. To address this constraint, we plan to explore alternative software solutions that offer greater flexibility and scalability, enabling us to conduct more comprehensive DDoS simulations and analyze a wider array of performance metrics. Despite our efforts, time constraints prevented us from implementing various types of DDoS attacks, including SYN flood and smurf flood. Incorporating these attack vectors into our simulations would provide valuable insights into their impact on different queuing algorithms and network configurations, helping us better understand the effectiveness of defense strategies and mitigation techniques.

## References

- [1] Cloudflare, "What is a DDoS attack? " cloudflare, <https://www.cloudflare.com/en-ca/learning/ddos/what-is-a-ddos-attack/> [accessed Feb. 23, 2024].
- [2] FORTINET, "What is a DDoS Attack?" Fortinet, <https://www.fortinet.com/resources/cyberglossary/ddos-attack> [accessed Feb. 23, 2024].
- [3] I. Kotenko and A. Ulanov, "Simulation of Internet DDoS Attacks and Defense." *Information Security*, [https://doi.org/10.1007/11836810\\_24](https://doi.org/10.1007/11836810_24). [accessed Feb. 23, 2024].
- [4] L. Arockiam Lawrence and B. Vani, "A Survey of Denial of Service Attacks and its Countermeasures on Wireless Network." *International Journal on Computer Science and Engineering*, [https://www.researchgate.net/publication/49965401\\_A\\_Survey\\_of\\_Denial\\_of\\_Service\\_Attacks\\_and\\_it%27s\\_Countermeasures\\_on\\_Wireless\\_Network](https://www.researchgate.net/publication/49965401_A_Survey_of_Denial_of_Service_Attacks_and_it%27s_Countermeasures_on_Wireless_Network) [accessed Feb. 23, 2024].
- [5] M. Poongothai and M. Sathyakala, "Simulation and analysis of DDoS attacks." *IEEE Xplore*, <https://ieeexplore.ieee.org/abstract/document/6513885> [accessed Feb. 23, 2024].
- [6] R. Davis, "The History and Future of DDoS Attacks." *CyberSecurity Magazine*, <https://cybersecurity-magazine.com/the-history-and-future-of-ddos-attacks/> [accessed Feb. 23, 2024].
- [7] Radware, "DDoS Attacks History." Radware, <https://www.radware.com/security/ddos-knowledge-center/ddos-chronicles/ddos-attacks-history/> [accessed Feb. 23, 2024].
- [8] M. Raza, "Denial-of-Service Attacks: History, Techniques, & Prevention." [https://www.splunk.com/en\\_us/blog/learn/dos-denial-of-service-attacks.html](https://www.splunk.com/en_us/blog/learn/dos-denial-of-service-attacks.html) [accessed Feb. 23, 2024].
- [9] A. Balyk, et al., "Using graphic network simulator 3 for DDoS attacks simulation." *International Journal of Computing*, 16.4 (2017): 219-225 [accessed Feb. 23, 2024].
- [10] B. Singh, K. Kumar and A. Bhandari, "Simulation study of application layer DDoS attack." *International Conference on Green Computing and Internet of Things (ICGCIoT)*, <https://dl.acm.org/doi/10.1109/ICGCIoT.2015.7380589> [accessed Feb. 23, 2024].
- [11] L. Trajkovic et al., "Distributed Denial of Service Attacks." [https://www.sfu.ca/~ljilja/papers/smc00\\_edited.pdf](https://www.sfu.ca/~ljilja/papers/smc00_edited.pdf) [accessed Feb. 23, 2024].



- [12] T. Kenyon, “How are DDoS attacks impacting businesses and services?” Cybermagazine, <https://cybermagazine.com/cyber-security/how-are-ddos-attacks-impacting-businesses-and-services> [accessed Mar. 20, 2024].
- [13] E. Fahmy, “How DDoS attacks impact business continuity.” edge, <https://www.edgemiddleeast.com/security/how-ddos-attacks-impact-business-continuity> [accessed Mar. 20, 2024].
- [14] L. Peterson and B. Davie, “6.2 Queuing Disciplines.” Computer Networks: A Systems Approach, <https://book.systemsapproach.org/congestion/queuing.html> [accessed Mar. 20, 2024].
- [15] Y. Guo et al., “Modeling distributed denial of service attack in advanced metering infrastructure.” *IEEE Xplore*, <https://ieeexplore.ieee.org/abstract/document/7131828/authors> [accessed Mar. 20, 2024].
- [16] Ali, S (2023) DDOS\_attacks\_simulation\_using\_NS3 [Source Code]. [https://github.com/shahdzzz/DDOS\\_attacks\\_simulation\\_using\\_NS3](https://github.com/shahdzzz/DDOS_attacks_simulation_using_NS3) [accessed Mar. 20, 2024].
- [17] Cloudflare, “UDP flood DDoS attack.” cloudflare, <https://www.cloudflare.com/learning/ddos/udp-flood-ddos-attack/> [accessed Mar. 21, 2024].
- [18] Cloudflare, “SYN flood attack.” cloudflare, <https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/> [accessed Mar. 21, 2024].
- [19] imperva, “Smurf DDoS attack.” imperva, <https://www.imperva.com/learn/ddos/smurf-attack-ddos/> [accessed Mar. 21, 2024].

## Contribution

The project workload was evenly distributed among team members. We utilized Google Drive for managing and sharing our work, while collaborative efforts on presentations and reports were facilitated through Google Docs and Slides. Weekly meetings were held to review deadlines and project progress, ensuring accountability. Each team member actively engaged in creating and simulating networks using NS-3, contributing to valuable hands-on experience for all. The project's success can be attributed to our effective teamwork.

	Akash Malhi	Gurnek Ghatarora	Elaine Luu
References and Literature Review	33.33%	33.33%	33.33%
Project Website	33.33%	33.33%	33.33%
Simulation scenarios, implementation, analysis, and discussion of simulation result	33.33%	33.33%	33.33%
Project Presentation	33.33%	33.33%	33.33%
Written final report	33.33%	33.33%	33.33%

## Appendix:

### 6.1 Code Listing [16]

```
/*
    Reference Source:

    ORIGINAL AUTHOR:  Shahd Mohamed Ali
    ORIGINAL GITHUB:
    https://github.com/shahdzzz/DDOS_attacks_simualtion_using_NS3

    Our adaptations:
    Changed topology organizations
    Changed DDoS Rates for simulations
    Changed Node labels
    Changed Simulation variables
    Changed Bot allocation to networks
    Removed un-needed code
    Export pcap files in segments -> single pcap resulted in segfault
    Changed IP Addresses
*/

// Include necessary header files
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/applications-module.h"
#include "ns3/netanim-module.h"
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/internet-stack-helper.h"
#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/ipv4-global-routing-helper.h"
#include "ns3/node-container.h"
#include "ns3/pcap-file.h"
#include "ns3/ipv4-header.h"
#include "ns3/ipv4-address.h"

// Define constants
#define UDP_SINK_PORT 9001
#define MAX_BULK_BYTES 100000
#define DDOS_RATE "909kb/s"
```

```

#define MAX_SIMULATION_TIME 10.0
#define NUMBER_OF_BOTS 20

NS_LOG_COMPONENT_DEFINE("DDoSAttack");
using namespace ns3;

int main(int argc, char * argv[]) {
    // Create Network nodes
    NodeContainer nodes;
    nodes.Create(23);

    // Create Users
    NodeContainer users;
    users.Create(12);

    // Create Bots
    NodeContainer botNodes;
    botNodes.Create(NUMBER_OF_BOTS);

    // Setting up network topology with normal users, network nodes, and bots
    // Point-to-point Network (Net1)
    CsmaHelper PointToPoint;
    PointToPoint.SetChannelAttribute("DataRate", StringValue("100Mbps"));
    PointToPoint.SetChannelAttribute("Delay", StringValue("2ms"));
    NodeContainer containerPP = NodeContainer(nodes.Get(0), nodes.Get(1),
nodes.Get(2));
    NetDeviceContainer devicesPP = PointToPoint.Install(containerPP);

    // PCAP tracing
    PointToPoint.EnablePcapAll("909kbps_20Bots");

    // Point-to-point sub-network 1
    CsmaHelper PointToPointNet1;
    PointToPointNet1.SetChannelAttribute("DataRate", StringValue("100Mbps"));
    PointToPointNet1.SetChannelAttribute("Delay", StringValue("2ms"));
    NodeContainer containerPPNet1 = NodeContainer(nodes.Get(1), nodes.Get(3),
nodes.Get(4));
    NetDeviceContainer devicesPPNet1 = PointToPointNet1.Install(containerPPNet1);

    // Point-to-point sub-network 2
    CsmaHelper PointToPointNet2;
    PointToPointNet2.SetChannelAttribute("DataRate", StringValue("100Mbps"));
    PointToPointNet2.SetChannelAttribute("Delay", StringValue("2ms"));
    NodeContainer containerPPNet2 = NodeContainer(nodes.Get(2), nodes.Get(5),
nodes.Get(6));
    NetDeviceContainer devicesPPNet2 = PointToPointNet2.Install(containerPPNet2);

```

```

// Inserting users and attack bots. User 0 will be the target node.
CsmaHelper csmaPPNet1R1;
csmaPPNet1R1.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csmaPPNet1R1.SetChannelAttribute("Delay", StringValue("2ms"));
NodeContainer containerPPNet1R1;
containerPPNet1R1.Add(nodes.Get(3));
containerPPNet1R1.Add(users.Get(0));
containerPPNet1R1.Add(botNodes.Get(0));
containerPPNet1R1.Add(botNodes.Get(1));
containerPPNet1R1.Add(botNodes.Get(2));
containerPPNet1R1.Add(botNodes.Get(3));
containerPPNet1R1.Add(botNodes.Get(4));
containerPPNet1R1.Add(botNodes.Get(5));
containerPPNet1R1.Add(botNodes.Get(6));
containerPPNet1R1.Add(botNodes.Get(7));
containerPPNet1R1.Add(botNodes.Get(8));
containerPPNet1R1.Add(botNodes.Get(9));
containerPPNet1R1.Add(botNodes.Get(10));
containerPPNet1R1.Add(botNodes.Get(11));
containerPPNet1R1.Add(botNodes.Get(12));
containerPPNet1R1.Add(botNodes.Get(13));
containerPPNet1R1.Add(botNodes.Get(14));
containerPPNet1R1.Add(botNodes.Get(15));
containerPPNet1R1.Add(botNodes.Get(16));
containerPPNet1R1.Add(botNodes.Get(17));
containerPPNet1R1.Add(botNodes.Get(18));
containerPPNet1R1.Add(botNodes.Get(19));

NetDeviceContainer devicesPPNet1R1 = csmaPPNet1R1.Install(containerPPNet1R1);

//Rest of point-to-point subnets
CsmaHelper csmaPPNet1C1;
csmaPPNet1C1.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csmaPPNet1C1.SetChannelAttribute("Delay", StringValue("2ms"));
NodeContainer containerPPNet1C1 = NodeContainer(nodes.Get(4), users.Get(1));
NetDeviceContainer devicesPPNet1C1 = csmaPPNet1C1.Install(containerPPNet1C1);

CsmaHelper csmaPPNet2R2;
csmaPPNet2R2.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csmaPPNet2R2.SetChannelAttribute("Delay", StringValue("2ms"));
NodeContainer containerPPNet2R2 = NodeContainer(nodes.Get(5), users.Get(2));
NetDeviceContainer devicesPPNet2R2 = csmaPPNet2R2.Install(containerPPNet2R2);

CsmaHelper csmaPPNet2C2;
csmaPPNet2C2.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csmaPPNet2C2.SetChannelAttribute("Delay", StringValue("2ms"));

```

```

NodeContainer containerPPNet2C2 = NodeContainer(nodes.Get(6), users.Get(3));
NetDeviceContainer devicesPPNet2C2 = csmaPPNet2C2.Install(containerPPNet2C2);

// Network 3
CsmaHelper csmaNet3;
csmaNet3.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csmaNet3.SetChannelAttribute("Delay", StringValue("2ms"));
NodeContainer containerNet3 = NodeContainer(nodes.Get(20), nodes.Get(10),
nodes.Get(7));
NetDeviceContainer devicesNet3 = csmaNet3.Install(containerNet3);

// Sub-nets for Network 3
CsmaHelper csmaNet3Net1;
csmaNet3Net1.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csmaNet3Net1.SetChannelAttribute("Delay", StringValue("2ms"));
NodeContainer containerNet3Net1 = NodeContainer(nodes.Get(7), nodes.Get(8),
nodes.Get(9));
NetDeviceContainer devicesNet3Net1 = csmaNet3Net1.Install(containerNet3Net1);

CsmaHelper csmaNet3Net2;
csmaNet3Net2.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csmaNet3Net2.SetChannelAttribute("Delay", StringValue("2ms"));
NodeContainer containerNet3Net2 = NodeContainer(nodes.Get(10), nodes.Get(11),
nodes.Get(12));
NetDeviceContainer devicesNet3Net2 = csmaNet3Net2.Install(containerNet3Net2);

CsmaHelper csmaNet3Net1HH1;
csmaNet3Net1HH1.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csmaNet3Net1HH1.SetChannelAttribute("Delay", StringValue("2ms"));
NodeContainer containerNet3Net1HH1 = NodeContainer(nodes.Get(8), users.Get(4));
NetDeviceContainer devicesNet3Net1HH1 =
csmaNet3Net1HH1.Install(containerNet3Net1HH1);

CsmaHelper csmaNet3Net1H1;
csmaNet3Net1H1.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csmaNet3Net1H1.SetChannelAttribute("Delay", StringValue("2ms"));
NodeContainer containerNet3Net1H1 = NodeContainer(nodes.Get(9), users.Get(5));
NetDeviceContainer devicesNet3Net1H1 =
csmaNet3Net1H1.Install(containerNet3Net1H1);

CsmaHelper csmaNet3Net2HH2;
csmaNet3Net2HH2.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csmaNet3Net2HH2.SetChannelAttribute("Delay", StringValue("2ms"));
NodeContainer containerNet3Net2HH2 = NodeContainer(nodes.Get(11),
users.Get(6));
NetDeviceContainer devicesNet3Net2HH2 =
csmaNet3Net2HH2.Install(containerNet3Net2HH2);

```

```

CsmaHelper csmaNet3Net2H2;
csmaNet3Net2H2.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csmaNet3Net2H2.SetChannelAttribute("Delay", StringValue("2ms"));
NodeContainer containerNet3Net2H2 = NodeContainer(nodes.Get(12), users.Get(7));
NetDeviceContainer devicesNet3Net2H2 =
csmaNet3Net2H2.Install(containerNet3Net2H2);

// Network 4
CsmaHelper csmaNet4;
csmaNet4.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csmaNet4.SetChannelAttribute("Delay", StringValue("2ms"));
NodeContainer containerNet4 = NodeContainer(nodes.Get(13), nodes.Get(14),
nodes.Get(15));
NetDeviceContainer devicesNet4 = csmaNet4.Install(containerNet4);

// Subnets for Network 4
CsmaHelper csmaNet4Net1;
csmaNet4Net1.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csmaNet4Net1.SetChannelAttribute("Delay", StringValue("2ms"));
NodeContainer containerNet4Net1 = NodeContainer(nodes.Get(15), nodes.Get(18),
nodes.Get(19));
NetDeviceContainer devicesNet4Net1 = csmaNet4Net1.Install(containerNet4Net1);

CsmaHelper csmaNet4Net2;
csmaNet4Net2.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csmaNet4Net2.SetChannelAttribute("Delay", StringValue("2ms"));
NodeContainer containerNet4Net2 = NodeContainer(nodes.Get(14), nodes.Get(16),
nodes.Get(17));
NetDeviceContainer devicesNet4Net2 = csmaNet4Net2.Install(containerNet4Net2);

CsmaHelper csmaNet4Net1V1;
csmaNet4Net1V1.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csmaNet4Net1V1.SetChannelAttribute("Delay", StringValue("2ms"));
NodeContainer containerNet4Net1V1 = NodeContainer(nodes.Get(18), users.Get(8));
NetDeviceContainer devicesNet4Net1V1 =
csmaNet4Net1V1.Install(containerNet4Net1V1);

CsmaHelper csmaNet4Net1PTM1;
csmaNet4Net1PTM1.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csmaNet4Net1PTM1.SetChannelAttribute("Delay", StringValue("2ms"));
NodeContainer containerNet4Net1PTM1 = NodeContainer(nodes.Get(19),
users.Get(9));
NetDeviceContainer devicesNet4Net1PTM1 =
csmaNet4Net1PTM1.Install(containerNet4Net1PTM1);

CsmaHelper csmaNet4Net2V2;

```

```

        csmaNet4Net2V2.SetChannelAttribute("DataRate", StringValue("100Mbps"));
        csmaNet4Net2V2.SetChannelAttribute("Delay", StringValue("2ms"));
        NodeContainer containerNet4Net2V2 = NodeContainer(nodes.Get(16),
users.Get(10));
        NetDeviceContainer devicesNet4Net2V2 =
csmaNet4Net2V2.Install(containerNet4Net2V2);

        CsmHelper csmaNet4Net2PTM2;
        csmaNet4Net2PTM2.SetChannelAttribute("DataRate", StringValue("100Mbps"));
        csmaNet4Net2PTM2.SetChannelAttribute("Delay", StringValue("2ms"));
        NodeContainer containerNet4Net2PTM2 = NodeContainer(nodes.Get(17),
users.Get(11));
        NetDeviceContainer devicesNet4Net2PTM2 =
csmaNet4Net2PTM2.Install(containerNet4Net2PTM2);

// Network 4
PointToPointHelper mesh;
mesh.SetDeviceAttribute("DataRate", StringValue("100Mbps"));
mesh.SetChannelAttribute("Delay", StringValue("2ms"));
NetDeviceContainer devicesMesh;
devicesMesh.Add(mesh.Install(nodes.Get(20), nodes.Get(0)));
devicesMesh.Add(mesh.Install(nodes.Get(20), nodes.Get(21)));
devicesMesh.Add(mesh.Install(nodes.Get(20), nodes.Get(13)));
devicesMesh.Add(mesh.Install(nodes.Get(0), nodes.Get(20)));
devicesMesh.Add(mesh.Install(nodes.Get(0), nodes.Get(21)));
devicesMesh.Add(mesh.Install(nodes.Get(0), nodes.Get(13)));
devicesMesh.Add(mesh.Install(nodes.Get(13), nodes.Get(0)));
devicesMesh.Add(mesh.Install(nodes.Get(13), nodes.Get(20)));
devicesMesh.Add(mesh.Install(nodes.Get(13), nodes.Get(21)));
devicesMesh.Add(mesh.Install(nodes.Get(21), nodes.Get(0)));
devicesMesh.Add(mesh.Install(nodes.Get(21), nodes.Get(20)));
devicesMesh.Add(mesh.Install(nodes.Get(21), nodes.Get(13)));

// Server
PointToPointHelper p2pServer;
p2pServer.SetDeviceAttribute("DataRate", StringValue("100Mbps"));
p2pServer.SetChannelAttribute("Delay", StringValue("2ms"));
NetDeviceContainer devicesServer = p2pServer.Install(nodes.Get(21),
nodes.Get(22));

// Install Internet stack on all nodes
InternetStackHelper stack;
stack.Install(users);
stack.Install(botNodes);
stack.Install(nodes);

// Assign IP addresses for Point-to-point network (Net1)

```



```

    Ipv4AddressHelper addressPP, addressPPNet1, addressPPNet2, addressPPNet1R1,
    addressPPNet1C1, addressPPNet2R2, addressPPNet2C2, addressPPNet1R1C1;
    addressPP.SetBase("192.168.1.0", "255.255.255.0");
    addressPPNet1.SetBase("192.168.2.0", "255.255.255.0");
    addressPPNet2.SetBase("192.168.3.0", "255.255.255.0");
    addressPPNet1R1.SetBase("192.168.4.0", "255.255.255.0");
    addressPPNet1C1.SetBase("192.168.5.0", "255.255.255.0");
    addressPPNet2R2.SetBase("192.168.6.0", "255.255.255.0");
    addressPPNet2C2.SetBase("192.168.7.0", "255.255.255.0");

    Ipv4InterfaceContainer interfacesPP = addressPP.Assign(devicesPP);
    Ipv4InterfaceContainer interfacesPPNet1 = addressPPNet1.Assign(devicesPPNet1);
    Ipv4InterfaceContainer interfacesPPNet2 = addressPPNet2.Assign(devicesPPNet2);
    Ipv4InterfaceContainer interfacesPPNet1R1 =
    addressPPNet1R1.Assign(devicesPPNet1R1);
    Ipv4InterfaceContainer interfacesPPNet1C1 =
    addressPPNet1C1.Assign(devicesPPNet1C1);
    Ipv4InterfaceContainer interfacesPPNet2R2 =
    addressPPNet2R2.Assign(devicesPPNet2R2);
    Ipv4InterfaceContainer interfacesPPNet2C2 =
    addressPPNet2C2.Assign(devicesPPNet2C2);

    // Assign IP addresses for Network 3
    Ipv4AddressHelper addressNet3, addressNet3Net1, addressNet3Net2,
    addressNet3Net1HH1, addressNet3Net1H1, addressNet3Net2HH2, addressNet3Net2H2;
    addressNet3.SetBase("172.16.1.0", "255.255.255.0");
    addressNet3Net1.SetBase("172.16.2.0", "255.255.255.0");
    addressNet3Net2.SetBase("172.16.3.0", "255.255.255.0");
    addressNet3Net1HH1.SetBase("172.16.4.0", "255.255.255.0");
    addressNet3Net1H1.SetBase("172.16.5.0", "255.255.255.0");
    addressNet3Net2HH2.SetBase("172.16.6.0", "255.255.255.0");
    addressNet3Net2H2.SetBase("172.16.7.0", "255.255.255.0");

    Ipv4InterfaceContainer interfacesNet3 = addressNet3.Assign(devicesNet3);
    Ipv4InterfaceContainer interfacesNet3Net1 =
    addressNet3Net1.Assign(devicesNet3Net1);
    Ipv4InterfaceContainer interfacesNet3Net2 =
    addressNet3Net2.Assign(devicesNet3Net2);
    Ipv4InterfaceContainer interfacesNet3Net1HH1 =
    addressNet3Net1HH1.Assign(devicesNet3Net1HH1);
    Ipv4InterfaceContainer interfacesNet3Net1H1 =
    addressNet3Net1H1.Assign(devicesNet3Net1H1);
    Ipv4InterfaceContainer interfacesNet3Net1HH2 =
    addressNet3Net2HH2.Assign(devicesNet3Net2HH2);
    Ipv4InterfaceContainer interfacesNet3Net1H2 =
    addressNet3Net2H2.Assign(devicesNet3Net2H2);

```

```

// Assign IP addresses for Network 4
Ipv4AddressHelper addressNet4, addressNet4Net1, addressNet4Net2,
addressNet4Net1V1, addressNet4Net1PTM1, addressNet4Net2V2, addressNet4Net2PTM2;
addressNet4.SetBase("192.167.1.0", "255.255.255.0");
addressNet4Net1.SetBase("192.167.2.0", "255.255.255.0");
addressNet4Net2.SetBase("192.167.3.0", "255.255.255.0");
addressNet4Net1V1.SetBase("192.167.4.0", "255.255.255.0");
addressNet4Net1PTM1.SetBase("192.167.5.0", "255.255.255.0");
addressNet4Net2V2.SetBase("192.167.6.0", "255.255.255.0");
addressNet4Net2PTM2.SetBase("192.167.7.0", "255.255.255.0");

Ipv4InterfaceContainer interfacesNet4 = addressNet4.Assign(devicesNet4);
Ipv4InterfaceContainer interfacesNet4Net1 =
addressNet4Net1.Assign(devicesNet4Net1);
Ipv4InterfaceContainer interfacesNet4Net2 =
addressNet4Net2.Assign(devicesNet4Net2);
Ipv4InterfaceContainer interfacesNet4Net1V1 =
addressNet4Net1V1.Assign(devicesNet4Net1V1);
Ipv4InterfaceContainer interfacesNet4Net1PTM1 =
addressNet4Net1PTM1.Assign(devicesNet4Net1PTM1);
Ipv4InterfaceContainer interfacesNet4Net1V2 =
addressNet4Net2V2.Assign(devicesNet4Net2V2);
Ipv4InterfaceContainer interfacesNet4Net1PTM2 =
addressNet4Net2PTM2.Assign(devicesNet4Net2PTM2);

// Assign IP addresses for Net4 and Server
Ipv4AddressHelper addressCT, addressServer;
addressCT.SetBase("192.166.1.0", "255.255.255.0");
addressServer.SetBase("192.169.1.0", "255.255.255.0");

Ipv4InterfaceContainer interfacesCT = addressCT.Assign(devicesServer.Get(0));
// node 21
Ipv4InterfaceContainer interfacesServer =
addressServer.Assign(devicesServer.Get(1)); // node 22

MobilityHelper mobility;
// Adjust position allocator for bots (bottom)
mobility.SetPositionAllocator("ns3::GridPositionAllocator",
"MinX", DoubleValue(0.0),
"MinY", DoubleValue(400.0),
"DeltaX", DoubleValue(25.0),
"DeltaY", DoubleValue(25.0),
"GridWidth", UIntegerValue(6),
"LayoutType", StringValue("RowFirst"));
mobility.Install(botNodes);

```

```

// Adjust position allocator for users (left)
mobility.SetPositionAllocator("ns3::GridPositionAllocator",
"MinX", DoubleValue(0.0),
"MinY", DoubleValue(0.0),
"DeltaX", DoubleValue(25.0),
"DeltaY", DoubleValue(25.0),
"GridWidth", UIntegerValue(6),
"LayoutType", StringValue("RowFirst"));
mobility.Install(users);

// Adjust position allocator for network nodes (right)
mobility.SetPositionAllocator("ns3::GridPositionAllocator",
"MinX", DoubleValue(300.0),
"MinY", DoubleValue(200.0),
"DeltaX", DoubleValue(25.0),
"DeltaY", DoubleValue(25.0),
"GridWidth", UIntegerValue(6),
"LayoutType", StringValue("RowFirst"));
mobility.Install(nodes);

// Create Connections
UdpEchoServerHelper echoServer(9);
ApplicationContainer serverApps = echoServer.Install(nodes.Get(4));
serverApps.Start(Seconds(1.0));
serverApps.Stop(Seconds(10.0));

UdpEchoClientHelper echoClient(interfacesPP.GetAddress(0), 9);
echoClient.SetAttribute("MaxPackets", UIntegerValue(100));
echoClient.SetAttribute("Interval", TimeValue(Seconds(1.0)));
echoClient.SetAttribute("PacketSize", UIntegerValue(1024));

ApplicationContainer clientApps = echoClient.Install(users.Get(5));
clientApps.Start(Seconds(2.0));
clientApps.Stop(Seconds(10.0));

// Main Attacker
// Inject DDoS Traffic
OnOffHelper onoff("ns3::UdpSocketFactory",
Address(InetSocketAddress(interfacesPPNet2C2.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];

```

```

//Pulse DDoS traffic periodically
for (int k = 0; k < NUMBER_OF_BOTS; ++k) {
    onOffApp[k] = onoff.Install(botNodes.Get(k));
    onOffApp[k].Start(Seconds(2.0));
    onOffApp[k].Stop(Seconds(MAX_SIMULATION_TIME));
}

// Receiver Application (UDP Packet Sink)
PacketSinkHelper packetSinkHelper("ns3::UdpSocketFactory",
InetSocketAddress(interfacesPPNet2C2.GetAddress(1), UDP_SINK_PORT));
ApplicationContainer packetSinkApp = packetSinkHelper.Install(nodes.Get(1));
packetSinkApp.Start(Seconds(1.0));
packetSinkApp.Stop(Seconds(MAX_SIMULATION_TIME));

// Sender Application (Packets generated by this application are throttled)
OnOffHelper onoffHelper("ns3::UdpSocketFactory",
InetSocketAddress(interfacesPPNet2C2.GetAddress(1), UDP_SINK_PORT));
onoffHelper.SetAttribute("DataRate", DataRateValue(DataRate("5Mbps"))); // Set
the desired data rate for the sender
onoffHelper.SetAttribute("PacketSize", UintegerValue(1024)); // Set the packet
size for the sender
ApplicationContainer senderApp = onoffHelper.Install(nodes.Get(0));
senderApp.Start(Seconds(2.0));
senderApp.Stop(Seconds(MAX_SIMULATION_TIME - 10));

// Populate the routing tables
Ipv4GlobalRoutingHelper::PopulateRoutingTables();

// PCAP Tracing for device
csmaPPNet2R2.EnablePcap("427Test", devicesPPNet2R2.Get(0), true);

// Set up NetAnim
AnimationInterface anim("427Test.xml");

// Label the network nodes
anim.UpdateNodeDescription(nodes.Get(0), "Net1");
anim.UpdateNodeDescription(nodes.Get(1), "Net1");
anim.UpdateNodeDescription(nodes.Get(2), "Net1");
anim.UpdateNodeDescription(nodes.Get(3), "Net1");
anim.UpdateNodeDescription(nodes.Get(4), "Net1");
anim.UpdateNodeDescription(nodes.Get(5), "Net1");
anim.UpdateNodeDescription(nodes.Get(6), "Net1");
anim.UpdateNodeDescription(nodes.Get(20), "Net2");
anim.UpdateNodeDescription(nodes.Get(7), "Net2");
anim.UpdateNodeDescription(nodes.Get(10), "Net2");

```

```

anim.UpdateNodeDescription(nodes.Get(8), "Net2");
anim.UpdateNodeDescription(nodes.Get(9), "Net2");
anim.UpdateNodeDescription(nodes.Get(11), "Net2");
anim.UpdateNodeDescription(nodes.Get(12), "Net2");
anim.UpdateNodeDescription(nodes.Get(13), "Net3");
anim.UpdateNodeDescription(nodes.Get(15), "Net3");
anim.UpdateNodeDescription(nodes.Get(14), "Net3");
anim.UpdateNodeDescription(nodes.Get(18), "Net3");
anim.UpdateNodeDescription(nodes.Get(19), "Net3");
anim.UpdateNodeDescription(nodes.Get(16), "Net3");
anim.UpdateNodeDescription(nodes.Get(17), "Net3");
anim.UpdateNodeDescription(nodes.Get(21), "Net4");
anim.UpdateNodeDescription(nodes.Get(22), "Server");

//set network node colours
for (int i = 0; i < 23; i++) {
    anim.UpdateNodeColor(nodes.Get(i), 0, 0, 255);
}

// set the user nodes labels and colours
for (int i = 0; i < 12; i++) {
    std::ostringstream oss;
    oss << "User " << i;
    anim.UpdateNodeDescription(users.Get(i), oss.str());
    anim.UpdateNodeColor(users.Get(i), 0, 255, 0);
}

// set the bot nodes labels and colours
for (int i = 0; i < 20; i++) {
    std::ostringstream oss;
    oss << "Bot " << i;
    anim.UpdateNodeDescription(botNodes.Get(i), oss.str());
    anim.UpdateNodeColor(botNodes.Get(i), 255, 0, 0);
}

// Change the size of the nodes
for (int i = 0; i < 55; i++) {
    anim.UpdateNodeSize(i, 10, 10);
}

Simulator::Run();
Simulator::Destroy();
return 0;
}

```