**ENSC 427: Communications Network**

**Spring 2015**

**Group 8**

# Performance Analysis of a system during a DDoS attack

**http://www.sfu.ca/~spc12/**

Samuel Chow <spc12 at sfu.ca>
Tenzin Sherpa <tserpa at sfu.ca>
Sam Hoque <shoque at sfu.ca>

# Table of Contents

## List of Figures

## Glossary

**CBR**: Constant Bit Rate

**DDoS**: Distributed Denial of Service

**DNS**: Domain Name System

**DNSBL**: Domain Name System Blacklist

**FIFO**: First In First Out

**IANA**: Internet Assigned Numbers Authority

**ICMP**: Internet Control Message Protocol

**IP**: Internet Protocol

**HTTP**: Hypertext Transfer Protocol

**NTP**: Network Time Protocol

**POD**: Ping of Death

**PQM**: Passive Queue Management

**SFQ**: Stochastic Fair Queuing

**TCP**: Transmission Control Protocol

**UDP**: User Datagram Protocol

## 1. Abstract

Distributed Denial of Service (DDoS) attacks have become a common phenomena and a threat to modern day Internet. Popular web and mail servers face DDoS attacks on a regular basis and after some major attacks, firms have put in place robust prevention strategies to mitigate these threats. With the growth of the Internet, data sharing between host and client machines have amplified over time. A DDoS attack results in denial of service for legitimate users and is achieved by interrupting or suspending services of a host connected to the Internet. Attackers utilize methods to limit bandwidth or cause inefficient packet handling to restrict the servers from communicating with the clients. In this report, we provide a general overview of DDoS attacks and its prevention measures. Additionally, we simulate a DDoS attack on a client-server model and also examine the 'Black hole' preventative technique. Analysis and observations are performed later on the same parameters to note its effect pre and post a DDoS attack.

## 2. Introduction

The term DDoS is inherited from DoS or Denial of Service. A DDoS attack is not the same as a DoS attack. A DoS attack involves a single attacker, thus the attack's magnitude and effect is less severe. A DDoS attack employs multiple attackers (using zombies or botnets) and as a result, has the potential to cause significant inconvenience to clients, especially clients who require urgent response to their requests.

A DDoS attack is an explicit attempt by an attacker to prevent intended users from using resources. An attacker 'floods' a network by sending excessive large quantity of requests to a server and thus reducing the available bandwidth for an actual user. The attacker uses zombies/botnets (infected computers) to allocate server resources. The number of botnets and zombies is directly proportional to the scale of the attack.

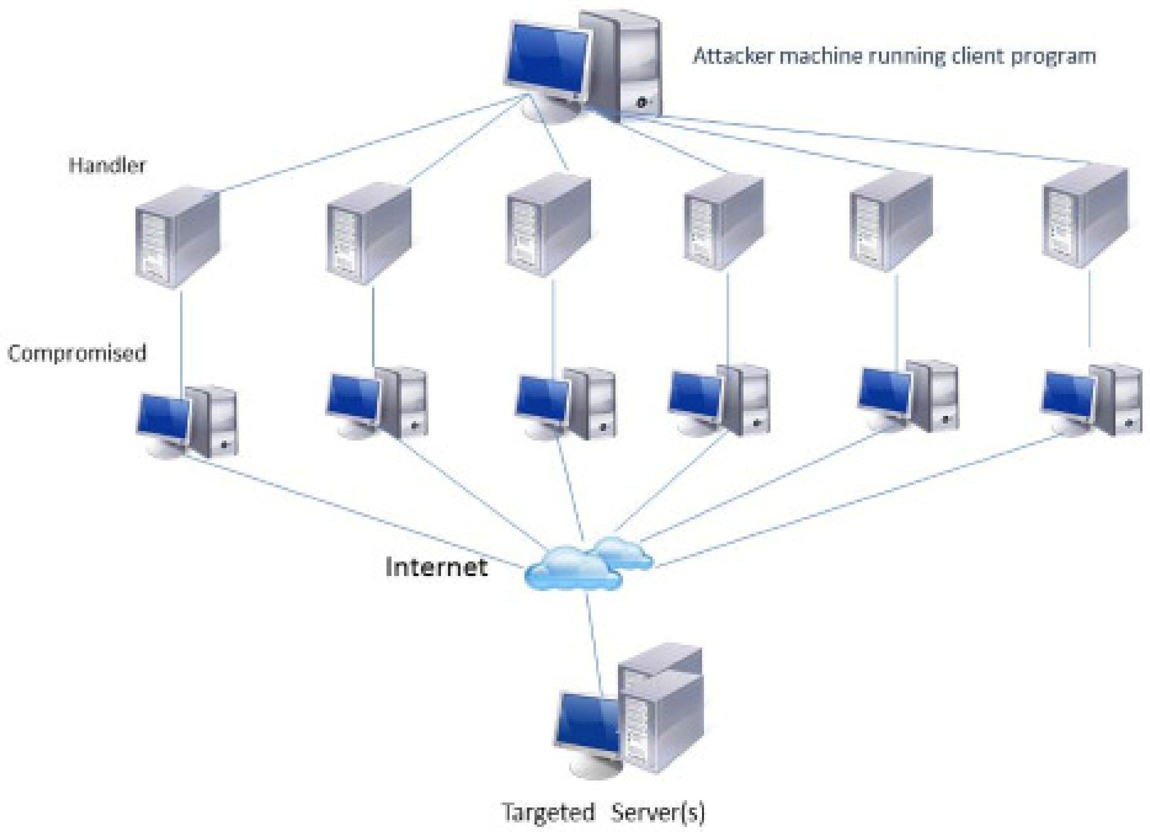

Figure 1: A conceptual diagram of a DDoS attack.

In this paper, we examine:

- Varieties of modern day DDoS attacks
- The importance of analyzing a DDoS attack and its effects
- Example of a DDoS attack
- DDoS attack prevention techniques
- Simulation parameters
- Simulation of a DDoS Reflection attack
- Simulation of 'Black hole' prevention technique.

The scope of this project includes simulating a DDoS attack on ns-2 platform utilizing Drop tail queuing method:

- One attacker, three zombies and six clients with same data transfer rate
- One attacker, three zombies, six clients and a 'Black hole'

We examine the overall effect on clients' bandwidth pre and post a DDoS attack, and note the implications of such an attack.

We also compare these results with the SFQ queuing method and summarize scopes for future work.

## 3. Types of DDoS attack

DDoS attacks can be broadly divided into three categories:

- Volume based attacks
- Protocol attacks
- Application layer attacks.

### 3.1 Volume based attacks

A volume based DDoS attack's goal is to saturate the available bandwidth of the attacked site and limit user access. Types of volume based DDoS attacks include UDP floods, ICMP floods and other spoofed-packet floods. The magnitude of the attack is measured in bits per second (Bps).

### 3.2 Protocol attacks

A protocol attack consumes actual server resources or those of intermediate communication equipment, such as firewalls and load balancers. Types of protocol attack include SYN floods, fragmented packet attacks, Ping of Death, Smurf DDoS and more. This type of attack is measured in packets per second.

### 3.3 Application layer attacks

Application layer attacks are comprised of seemingly legitimate and innocent requests that go unnoticed. The goal of these attacks is to crash the web server. Slowloris, Zero-day DDoS attacks, DDoS attacks that target Apache, Windows or OpenBSD vulnerabilities fall under this umbrella. The magnitude of an application layer attack is measured in requests per second.

## 4. Popular DDoS attack types

Here we look at some specific and particularly popular, dangerous types of DDoS attacks that has recently come in to public awareness:

### 4.1 Reflection attack

A reflection attack is the method of attacking a challenge-response authentication system that uses the same protocol in both directions to authenticate each other. In this sort of DDoS attack, it tricks the target host in to providing an answer to its own challenge.

The general attack outline is as follows:

- The attacker initiates a connection to a target A.
- The target A attempts to authenticate the attacker by sending it a challenge.

- The attacker then opens another connection to the target B, and sends the target B this challenge as its own.
- The target B responds to the challenge.
- The attacker sends that response back to the target A on the original connection.

If the authentication protocol is not carefully designed, the target host will accept the second response as valid. As a result, the attacker will have one fully authenticated channel connection and the other one will be abandoned.

### 4.2 SYN flood

A Transmission Control Protocol (TCP) connection sequence works on the “three-way handshake” principle, where a SYN request is sent to initiate a TCP connection with the host and must be answered by a SYN-ACK response from that host, and then confirmed by an ACK response from the requester. A SYN flood DDoS attack exploits this principle in the TCP connection sequence to initiate an attack. The requester sends multiple SYN requests, but either does not respond to the host’s SYN-ACK response, or sends the SYN requests from a spoofed Internet Protocol (IP) address. The host system continues to wait for acknowledgement for each of the requests, as a result tying up its available resources. There comes a time when no new connections can be made until the previous requests receive acknowledgement. This eventually results in a denial of service.

### 4.3 User Datagram Protocol (UDP) flood

This DDoS attack engages the UDP by flooding random ports on a remote host with numerous UDP packets. As a result, the host repeatedly checks for the application listening at that port, and when no application is found it replies with an Internet Control Message Protocol (ICMP) Destination Unreachable packet. This can ultimately lead to inaccessibility for legitimate users.

### 4.4 ICMP (Ping) flood

An ICMP flood overwhelms the target resource with ICMP Echo Request (ping) packets. During this attack, packets are sent as fast as possible without waiting for replies. It can consume both outgoing and incoming bandwidth, resulting in a significant overall system slowdown.

### 4.5 Ping of Death (POD)

A Ping of Death (POD) attack involves the attacker sending multiple malformed or malicious pings to a computer. The maximum packet length of an IP packet including header is 65,535 bytes. However, the Data Link Layer usually attaches a limit to the maximum frame size - for example 1500 bytes over an Ethernet network. In a POD DDoS attack, a large IP packet is split across multiple IP packets or fragments and sent to the host. When the recipient host reassembles the IP fragments into the complete packet, the recipient ends up with an IP packet larger than 65,535 bytes. This can overflow memory buffers allocated for the packet, causing denial of service for legitimate packets.

## 4.6 Slowloris

Slowloris is a highly targeted attack. It enables one web server to take down another server without affecting other services or ports on the target network. In a Slowloris DDoS attack, it holds as many connections to the target web server open for as long as possible. This is done by creating connections to the target server, but sending only a partial request. Slowloris constantly sends more HTTP headers, but never completes a request. The targeted server, assuming these are legitimate requests, keeps each of these false connections open. Eventually, this overflows the maximum concurrent connection pool possible, and leads to denial of additional connections from legitimate clients.

## 4.7 Network Time Protocol (NTP) Amplification

In an NTP Amplification attack, the perpetrator exploits publically accessible NTP servers to overwhelm the targeted server with UDP traffic. In this attack, the query-to-response ratio is anywhere between 1:20 and 1:200 or more. This means that any attacker that obtains a list of open NTP servers (e.g., by using tool like Metasploit or data from the Open NTP Project) can easily generate a devastating high-bandwidth resulting in a high-volume DDoS attack.

## 4.8 Hypertext Transfer Protocol (HTTP) flood

In HTTP flood DDoS attack, the attacker exploits seemingly legitimate HTTP GET or POST requests to attack a web server or an application. HTTP floods do not use malformed packets, spoofing or reflection techniques. On the contrary, it requires less bandwidth than other attacks to bring down the targeted site or web server. The attack becomes most effective if it can force the server or application to allocate the maximum resources possible in response to each single request. Here is a diagram of a HTTP DDoS attack comprising of over 690 million requests sent from 180 thousand different botnet IP addresses over the duration of 8 days. The different colored circles identify the region and volume of requests.

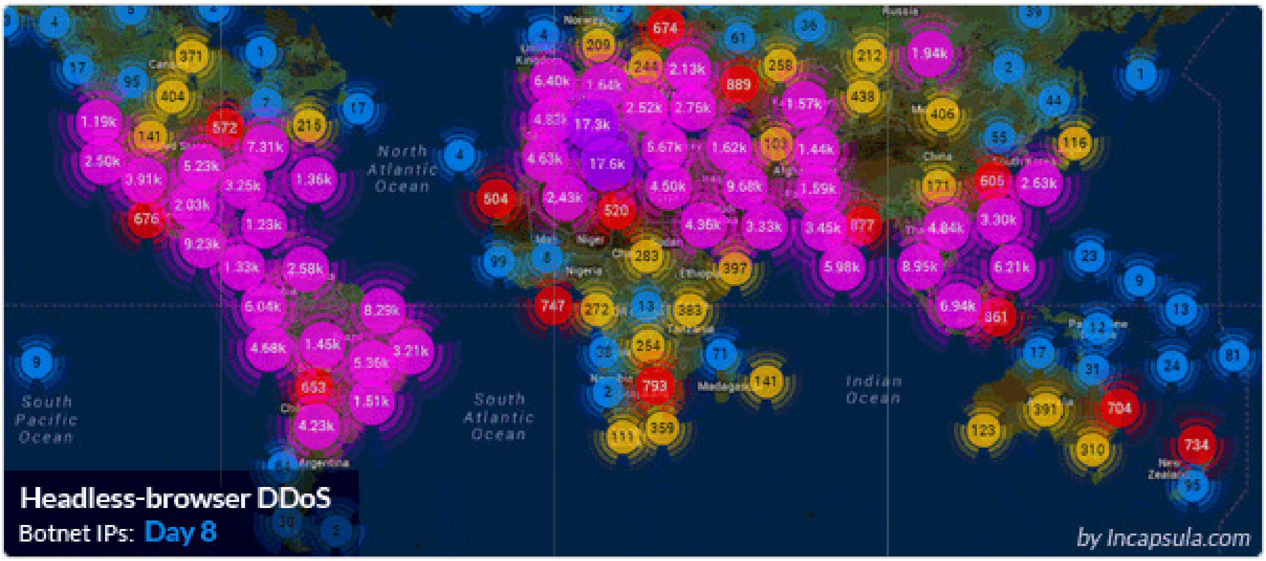

Figure 2: A massive HTTP flood: 690,000,000 DDoS requests from 180,000 botnets IPs.

### 4.9 Zero-day DDoS attacks

“Zero-day” DDoS attacks in a nutshell are unknown or new attacks, exploiting inefficiencies and vulnerabilities for which no patch has yet been released. The term is well-known among the members of the hacker community. Recently, the practice of trading Zero-day vulnerabilities has become a popular activity among hackers.

## 5. Importance of analyzing a DDoS attacks and its effects

DDoS attacks can essentially disable a computer or even a network. Depending on the nature of the enterprise, this can effectively disable an organization resulting in a huge loss. According to a 2013 Neustar survey result, DDoS attacks cost businesses $100,000 per hour in average. This cumulates to an extraordinary $1 million in losses before an internet-reliant company even starts to mitigate the attack. Apart from the financial losses, a DDoS attack can lead to erosion of brand value of a company, skyrocketing operational costs, and a need to invest in new people and advance technologies to manage the risk better in the future. The following figure illustrates the magnitude and the number of DDoS attacks across the world during the third and fourth quarters of 2013:

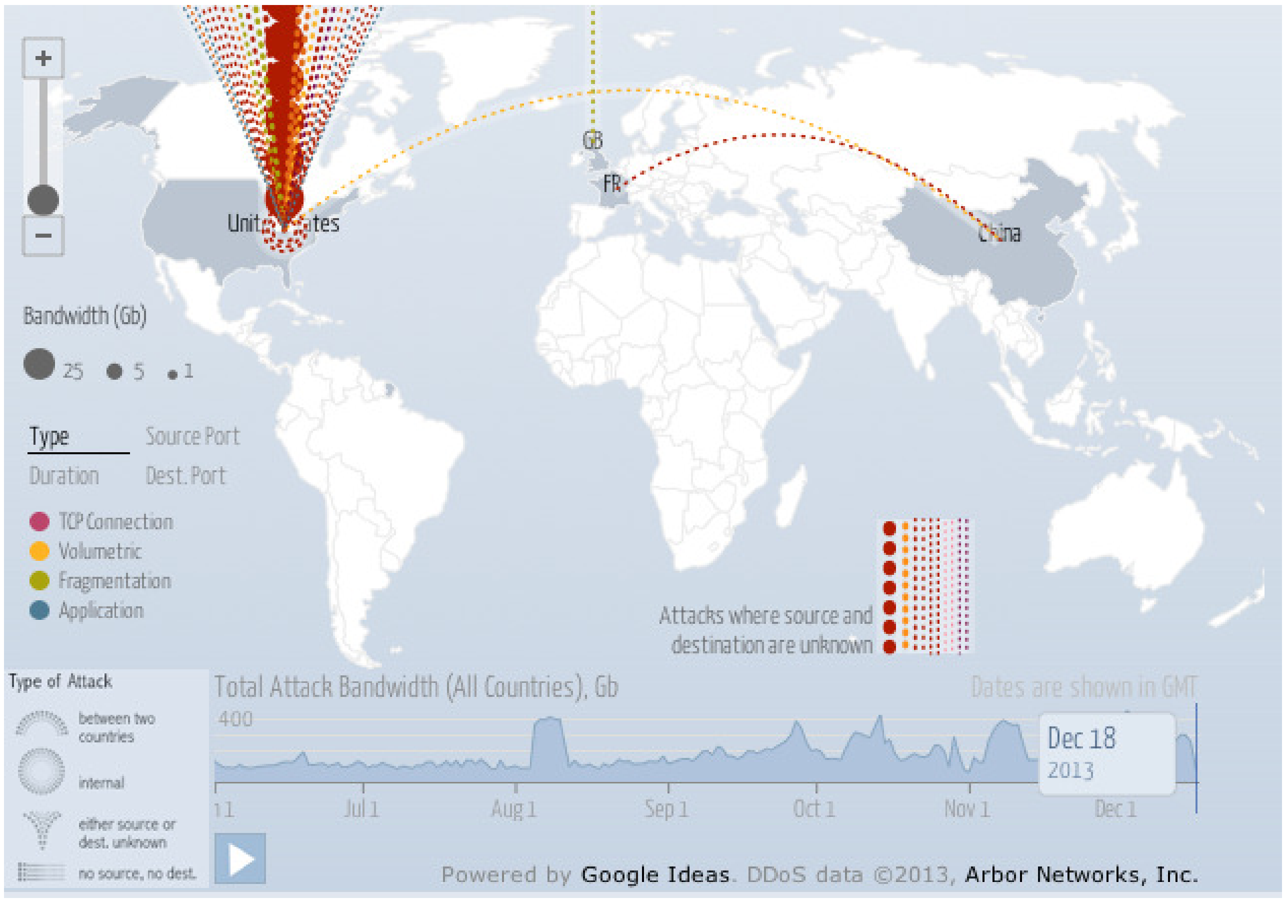

Figure 3: Global DDoS attack visualization from 2013.

A DDoS attack can result in a combination of any of the following fatal losses:

- Network connectivity
- Bandwidth consumption
- Consumption of other scarce, limited, or non-renewable resources
- Destruction or alteration of configuration information
- Physical destruction or alteration of network components.

## 5.1 Network connectivity

Denial-of-service attacks are most frequently executed against network connectivity. As a result the host or the network is prevented from communicating on the network. The attack does not depend on the attacker being able to consume a network's bandwidth. The intruder can consume kernel data structures involved in establishing a network connection and initiate a DDoS attack. Therefore, an intruder can even execute a DDoS attack from a dial-up connection against a machine on a very fast network.

An intruder can also use a computer's own resources against itself in unexpected ways. For example, the intruder can use forged UDP packets to connect the echo service on one machine to the critical services on another machine. These two services will consume all available network bandwidth between them and affect the network connectivity for all future machines.

### 5.2 Bandwidth consumption

An intruder may also be able to consume all the available bandwidth on a network by generating a large number of packets directed to the network. Typically, these packets are ICMP ECHO packets, but in principle they can be anything. Further, the intruder does not need to operate from a single machine; rather he can coordinate several other botnets and zombie machines on different networks to achieve a larger effect.

### 5.3 Consumption of other critical resources

In addition to network connectivity and bandwidth consumption, intruders may be able to consume other critical resources that a system requires in order to operate. These include, but not limited to:

- **Data structures**: By writing a simple program or script that repeatedly creates copies of itself.
- **Disk space**: By generating excessive numbers of mail messages, intentionally generating errors that must be logged or placing files in anonymous ftp areas or network shares.
- **System crash**: By sending unexpected data over the network.
- Essential devices and backup services as printers or tape devices.

### 5.4 Destruction or alteration of configuration information

An improperly configured computer may not perform well or may not operate at all. An intruder can alter or destroy configuration information that prevents one from using a computer or network. For example, if the intruder changes the routing information in a router, the network may be disabled. Or if the registry on a Windows NT machine is modified, certain functions may become unavailable.

### 5.5 Physical destruction or alteration of network components

Physical security is a prime component in guarding against many types of attacks in addition to denial of service. The primary concern with this type of attack is physical security. A DDoS attack can lead to unauthorized access of computers, routers, network wiring closets, network backbone segments, power and cooling stations, and many other critical components of a network.

## 6. Example of a DDoS attack: Spamhaus

Spamhaus provides one of the key tools that performs much of the anti-spam filtering online. It patrols the Internet for spammers and publishes a list of IP addresses that should not be trusted. This empowers email system administrators filter unwanted messages. Spamhaus' services are so efficient and important to the operation of the Internet's email architecture that when a lawsuit threatened to shut their services down, industry experts testified that doing so risked breaking the email service. They attested that Spamhaus is directly or indirectly responsible for filtering as much as 80% of daily spam messages.

On March 18, 2013, Spamhaus was in the midst of a large DDoS attack against their website. It was sufficiently large to fully saturate their connection and knock their site offline. The attack was categorized as a Layer 3 attack. Layer 3 DDoS attacks are types of volumetric DDoS attacks on a network infrastructure and rely on extremely high volumes (floods) of data to slow down web server performance, consume bandwidth, and eventually degrade access for legitimate users. Existent solutions at Spamhaus were ineffective in stopping it. Source of the attack was unknown, but it most likely originated from a group of individuals working together, a botnet of compromised PCs, a botnet of compromised servers, incorrectly configured DNS resolvers, or even home Internet routers with weak passwords.

CloudFlare was assigned to mitigate the attack on March 19, 2013. The following is a snapshot of the attack they observed. The green area represents in-bound requests and the blue line represents out-bound responses.

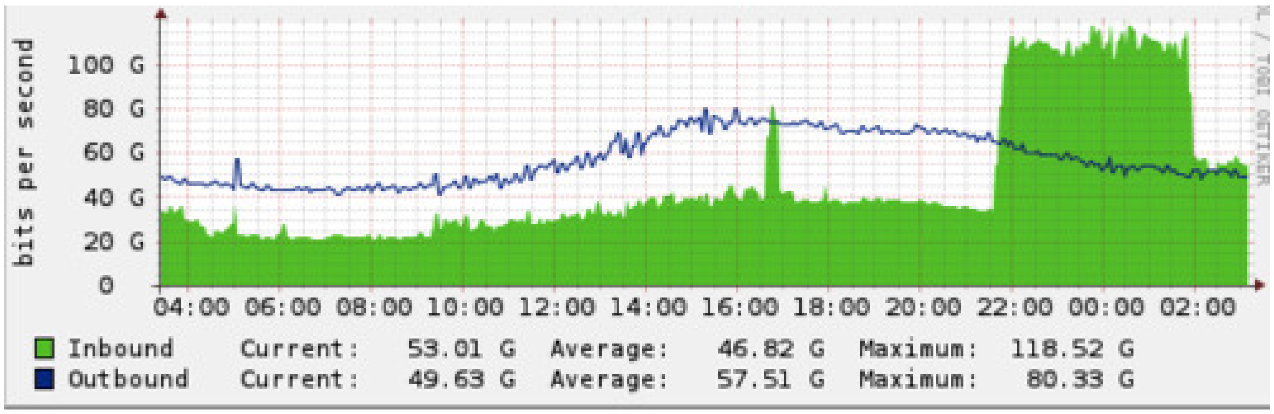

Figure 4: DDoS attack on Spamhaus servers on March 19, 2013.

The largest source of the attack traffic against Spamhaus came from DNS reflection. The attacker was sending requests for the DNS zone file for ripe.net to open DNS resolvers. The requests were likely 36 bytes long and the response was about 3,000 bytes, translating in an amplification factor of approximately 100. There were over 30,000 unique DNS resolvers involved in this attack.

CloudFlare made heavy use of 'Anycast' to mitigate the attack. During the DDoS attack, Anycast diluted the attacker requests by spreading it across CloudFlare's 23 worldwide data centers. These data centers announced the same IP address to CloudFlare customers. As a result, the battle was no longer between many versus one, instead it transformed in to a battle of many versus many. Traffic was not concentrated in any one location and the bottleneck was removed.

During this DDoS attack, the attacker also used SYN flooding methods to initiate supplementary DDoS attacks on SpamHaus. Timely intervention by CloudFlare saved the email exchange servers from going offline that day.

# 7. DDoS attack prevention techniques

There is no one-size-fits-all, unique solution to eliminate the threats posed by various DDoS attacks. Network security firms deploy a combination of preventative measures to nullify a DDoS attack's effect. Here we look at some of the more popular and robust prevention techniques:

- Null route (Black hole)
- Domain Name System Blacklist (DNSBL)
- Filtering techniques
- Other general techniques.

## 7.1 Null route (Black hole)

When a route is defined in a Linux or Unix operating system, it tells the system which specific IP address it needs to communicate to for a successful network connection. When information is sent through the Internet, it goes through an ISP Gateway and then from there, the information is sent through the Internet to another gateway. Each gateways carry a blacklist of IP addresses that should not be trusted. Bad information will be redirected from the gateway to 'Black holes' through null routes. Borrowing on this principle, a 'null route' or a 'Black hole' simply tells the system to drop the network communication from a specific IP address. Once the malicious IP address is identified, the server will no longer send a SYN/ACK reply and the unknown TCP based network connection will be disabled from establishing communication. If the request comes from a UDP based network connection, it will still be received by the server. However, the server will no longer send a response to the originating IP address. As a result, the server will respond to legitimate user requests only and ignore the attackers' requests.

## 7.2 Domain Name System Blacklist (DNSBL)

DNSBLs are chiefly used to publish lists of IP addresses known to be involved in spamming or potentially harmful activities that can negatively impact a user. This list is compiled in such a format that computer programs on the Internet can readily access it. The technology is built on top of the DNS. Similar to the Null route method, all modern mail servers can be designed to reject or flag messages which have been sent from a site listed on one or more such lists.

## 7.3 Filtering techniques

- **SYN proxy**: A SYN proxy server is set up so that during a SYN flood, all connection requests are screened and only those that are legitimate requests are forwarded.
- **Limiting connections**: Preference is only given to existing connections. New connection requests are limited and screened before providing access.
- **Aggressive aging**: Idle connections are removed from the connection tables in firewall and servers to free up resources.
- **Source rate limiting**: If there are limited numbers of IP addresses involved in a DDoS attack, outer IP addresses that break the norm are identified and ignored.
- **Dynamic filtering**: When the nature of the attack and the attackers evolve constantly, undisciplined behavior is identified and punished for a short time by creating a short span-filtering rule. The filtering rule is removed after a specific time span.
- **Active verification**: Combined with SYN proxy, legitimate IP addresses are cached into a memory table for a limited period of time. After verification they are then let out without SYN proxy check.
- **Anomaly recognition**: For scripted and coordinated DDoS attacks, anomaly checks are performed on headers, state and rate. As a result, most attacks will be filtered out.
- **Granular rate limiting**: Rate thresholds are set for attack packets based on past behavior and are adjusted adaptively over time.
- **White list, black list**: Based on the location of an IP address on either of these lists, services will be accordingly allowed or denied.
- **Dark address prevention**: IP addresses not assigned by the Internet Assigned Numbers Authority (IANA) are identified as dark addresses. These IP addresses are thus all blocked and considered as spoofs.

## 7.4 Other general techniques

- **Unused services**: Users may disable the applications and ports that are open on the host system, but are left unused.
- **Security patches**: By installing all relevant latest security patches and updates to the system.
- **IP broadcast**: Achieved by disabling IP broadcast on the host computer.
- **Firewalls**: Firewalls can help prevent users from launching simple flooding type DDoS attacks from their machines. However, this method's efficiency is limited and is not suitable for complex attacks.
- **IP hopping**: By changing the IP address or the location of the active host server proactively within a pool of homogeneous servers or with a pre-specified set of IP address ranges, DDoS attacks may be avoided.

## 8. Simulation

Prior to commencing the simulation we list the:

- Simulation scenarios
- Simulation parameters
- Simulation topologies
- Simulation run time.

### 8.1 Simulation scenarios

- **DDoS attack type:** Reflection attack
- **DDoS prevention technique**: DNSBL (DNS Blacklist) and 'Null route'
- **Platforms**: ns-2 , X-graph , Ubuntu 14.04 LTS
- **2 Scenarios**:
    - Scenario One: One attacker, three zombies, and six clients
    - Scenario Two: One attacker, three zombies, six clients, and one 'Black hole'

### 8.2 Simulation parameters

The following parameters were adhered to in running this simulation. Unless otherwise stated, the parameters are:

- **Data Rates:**
    - Clients: 50 bytes at 0.5 Mbps
    - Zombies: 50 bytes at 5.0 Mbps
    - Attacker: 50 bytes at 0.01 Mbps
- **Agent**: UDP
- **Application**: Constant Bit Rate (CBR)
- **Queue**: Drop tail
- **Delay:** 10 ms
- **Bandwidth:**
    - Client to Gateway: 45 Mbps (T3 Connection)
    - Zombie to Gateway: 45 Mbps (T3 Connection)
    - Gateway to Server: 12.5 Mbps

### 8.3 Simulation topologies

We simulated two topologies to observe the effects of a DDoS attack on a client-server system. In one of the topologies, we included a 'Black Hole' to prevent the DDoS attack from affecting the system.

**Topology 1**: One attacker, three zombies, and six clients:

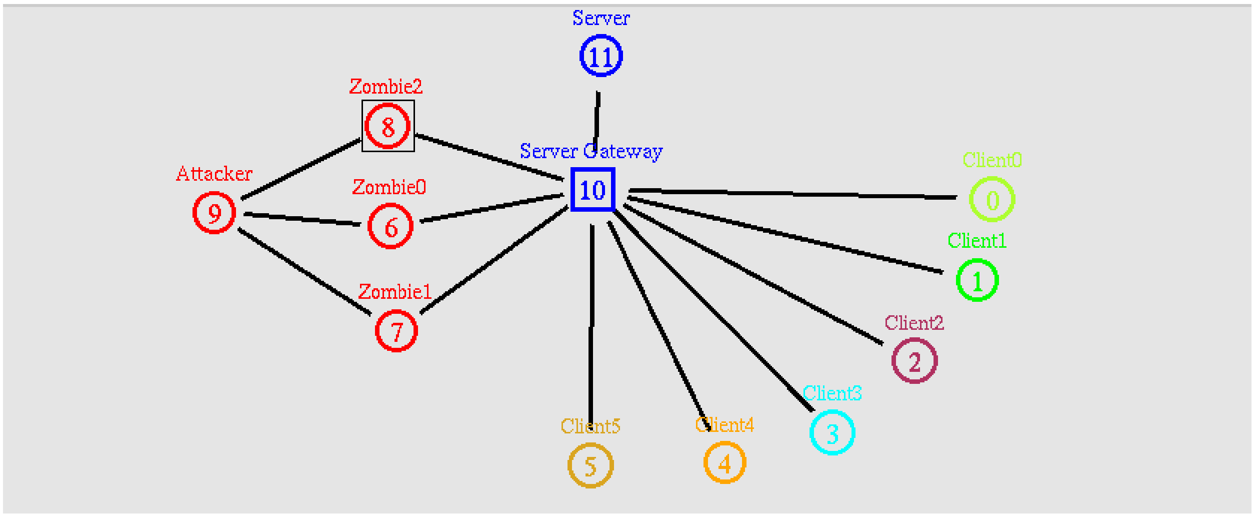

Figure 5: DDoS attack involving one attacker, three zombies, and six clients.

**Topology 2**: One attacker, three zombies, six clients, and one 'Black hole':

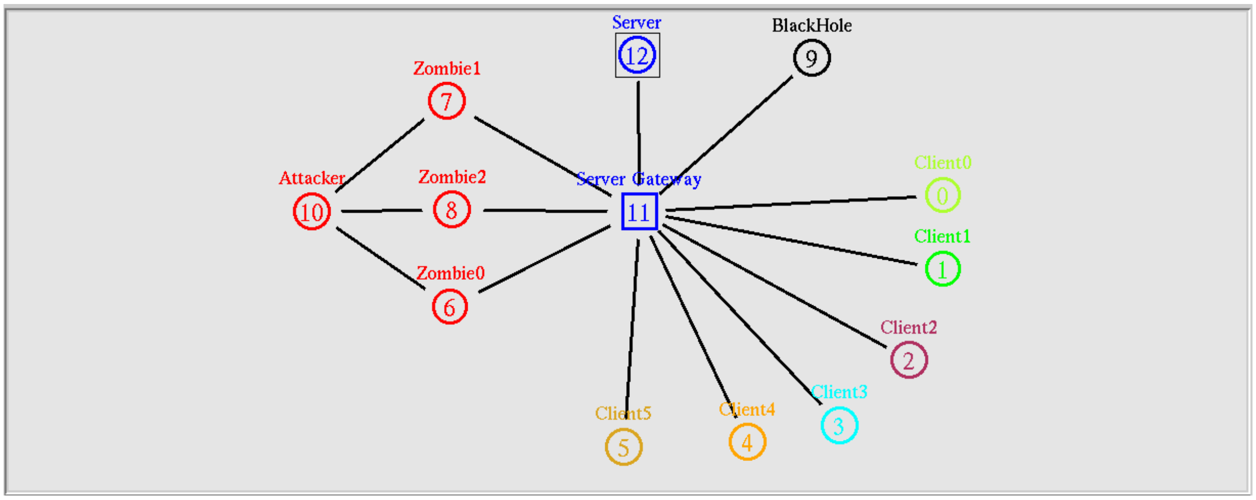

Figure 6: DDoS attack involving one attacker, three zombies, six clients, and one 'Black hole'.

## 8.4 Simulation run time

We simulate the following scenarios to observe the effect of a DDoS attack on a set of clients and server. We also note the efficiency of 'Black hole' prevention technique in nullifying this DDoS attack:

- **0s < t < 10s**:
  No traffic sent.

- **10s < t < 19.9s**:
  Clients send data to server and data reach server.
- **t = 19.9s**
  Attacker sends data to zombies to begin the DDoS attack.
- **20s < t < 40s**
  DDoS occurs and clients' data rate drop.
- **t = 39.9s**
  Attacker sends data to zombies to stop the DDoS attack.
- **t = 40s**
  DDoS stops and clients' data reach the server again.
- **t = 50s**
  All traffic stops.

## 9. Simulation of a DDoS Reflection attack

### 9.1 Analysis

Between 0s < t < 9.9s there is no data flow from the clients to the server. Data flow starts at t=10s and continues until t=50s. DDoS attack is initiated at t=20s. Prior to the DDoS attack, data flow is successful among all the clients and the server. Clients are sending 50 bytes of data at a rate of 0.5 Mbps to the server. Here is a ns-2 network topology of the system between 10s < t < 20s:

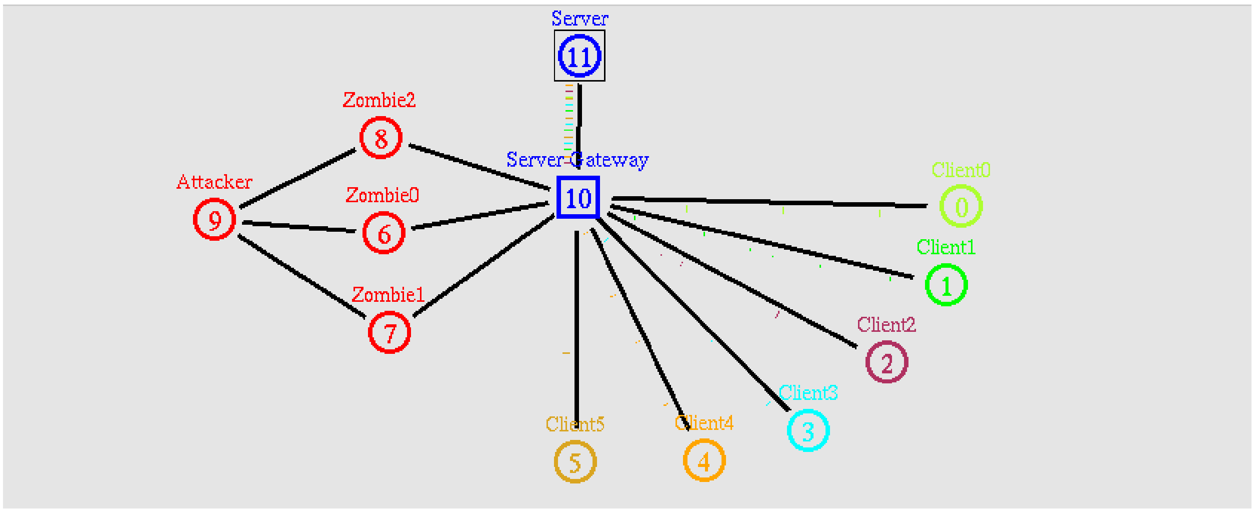

Figure 7: Clients' data flow to the server is successful (10s < t < 20s).

The DDoS attack commences at t=20s and continues until t=40s. During this time, the attacker sends a request (50 bytes at a rate of 0.01 Mbps) to the three zombies at t=19.9s to initiate the attack. The zombies start flooding the network with 50 bytes data at a rate of 5.0 Mbps at t=20s. As a result, the server's bandwidth is overwhelmed and client's data flow gets restricted. Loss of packets continues to occur until the attacker sends another request at t=39.9s to terminate the DDoS attack. The zombies stop sending data at t=40s. All clients are successfully able to send data to the server again after t=40s.

It is noteworthy that, in spite of the attacker sending requests to the zombies at 0.01 Mbps, it is able to exhaust the server's bandwidth of 12.5 Mbps by utilizing the zombies. This suggests that an attacker can initiate a large scale DDoS attack on faster networks, even though he or she may be limited to a slower Internet connection. The scale of the attack depends on the number of zombies involved in the attack and their data transfer rates. Here is a ns-2 network topology of the system between 20s < t < 40s showing data loss:

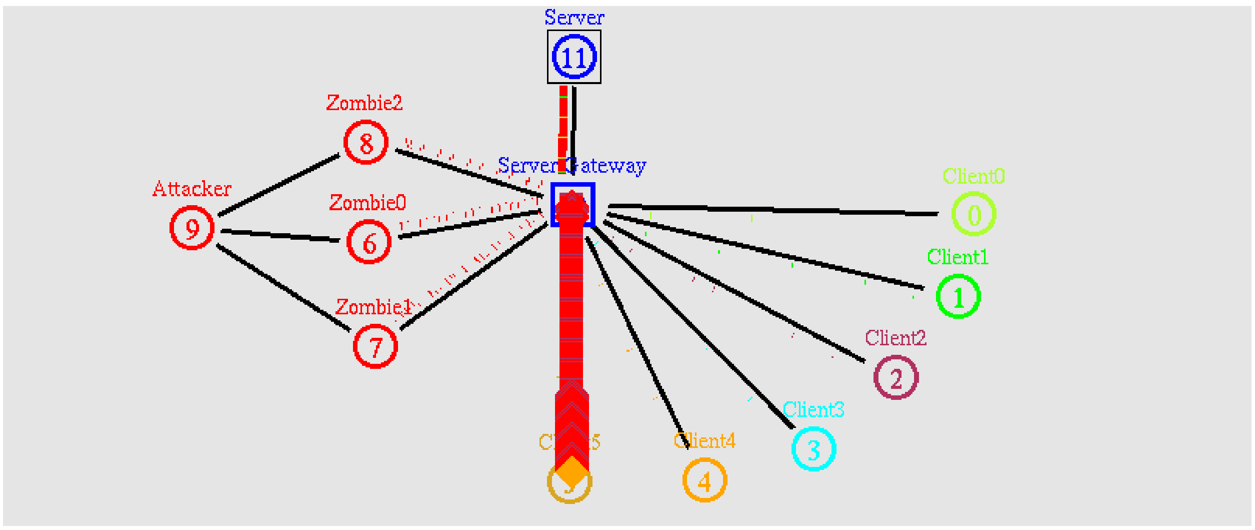

Figure 8: Clients' data flow to the server is unsuccessful (20s < t < 40s).

## 9.2 Observations

We plot the results of this simulation on X Graph for the duration of 0s < t < 60s and observe the 'bandwidth vs. time' graph below:

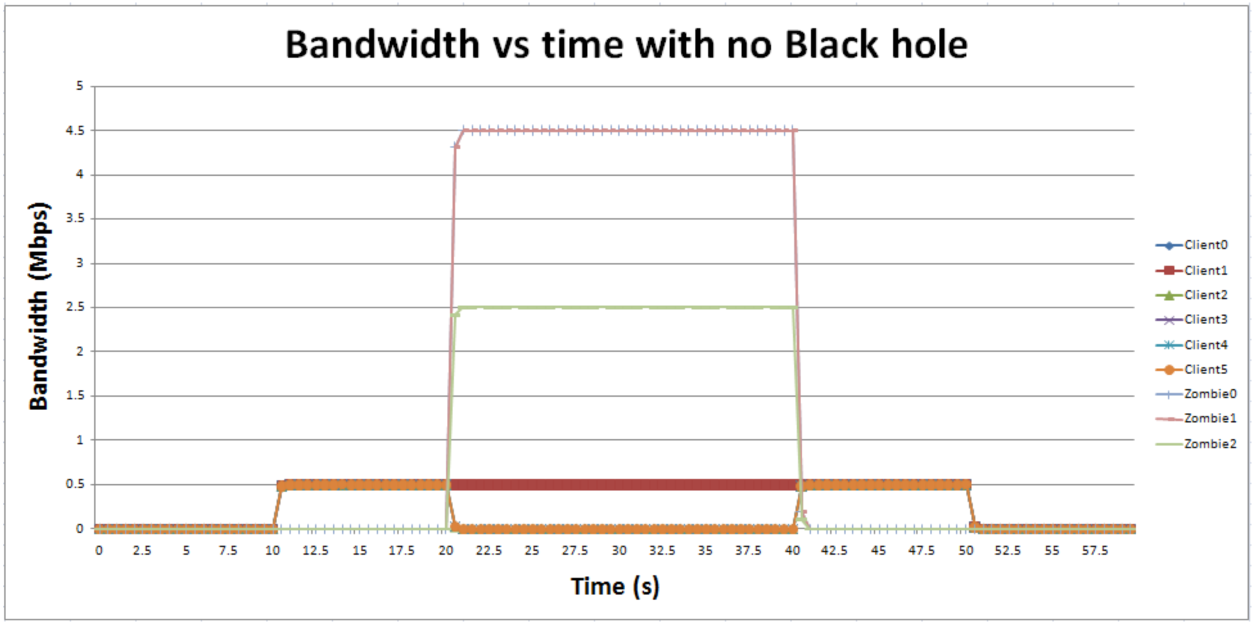

Figure 9: 'Bandwidth vs. time' graph of a DDoS attack (0s < t < 60s).

Some key observations from the diagram are as follows:

- There is no traffic flow to the server between 0s < t < 10s and 50s < t < 60s, as suggested earlier. Total bandwidth used during this time remains zero.
- Data flow from clients initiates at t=10s and ends at t=50s. All the six clients are sending the same size data at the same rate of 0.5 Mbps. As a result, the server's bandwidth is six times the data rate or 3 Mbps. This is true prior and post the DDoS attack, between 10s < t < 20s and 40s < t < 50s.
- The 3 Zombies flood the server with a data flow rate of 5 Mbps between 20s < t < 40s. It is noticeable that this request is beyond the server's capacity and the server is only able to acknowledge 4.5 Mbps request from zombie 0 and zombie 1, and 2.5 Mbps data request from zombie 2. The server's total bandwidth has reached its maximum of 12.5 Mbps at this time and the server is unable to service any further requests.
- Client 2, client 3, client 4 and client 5 are completely disabled from sending data and their transfer rate drops to 0 Mbps. These clients are denied service. However, client 0 and client 1 are able to transmit data at their rates of 0.5 Mbps. Drop Tail is a Passive Queue Management (PQM) algorithm which only sets a maximum length for each queue at the router. Routers decide when to drop packets. Client 0 and Client 1 were the first ones to initiate data transferring. It is possible that they continue to transmit data because of the first in first out (FIFO) algorithm. We are unsure about the exact cause. Further research is necessary and we list this as a scope for future work. The following is a snapshot of data

transfer rates of the clients and zombies during the DDoS attack between t=20.5s to t=25s. Client 0 and client 1 continue to transmit data at the rate of 0.5 Mbps. This pattern continues until the end of the DDoS attack (t=40s):

| **Time (s)** | **Client 0 (Mbps)** | **Client 1 (Mbps)** | **Client 2 (Mbps)** | **Client 3 (Mbps)** | **Client 4 (Mbps)** | **Client 5 (Mbps)** | **Zombie 0 (Mbps)** | **Zombie 1 (Mbps)** | **Zombie 2 (Mbps)** | **Total (Mbps)** |
|---|---|---|---|---|---|---|---|---|---|---|
| 20.5 | 0.498432 | 0.498432 | 0.023808 | 0.023808 | 0.02304 | 0.02304 | 4.310016 | 4.310016 | 2.408448 | 12.11904 |
| 21 | 0.499968 | 0.499968 | 0 | 0 | 0 | 0 | 4.50048 | 4.499712 | 2.49984 | 12.49997 |
| 21.5 | 0.499968 | 0.499968 | 0 | 0 | 0 | 0 | 4.499712 | 4.50048 | 2.49984 | 12.49997 |
| 22 | 0.499968 | 0.499968 | 0 | 0 | 0 | 0 | 4.499712 | 4.499712 | 2.500608 | 12.49997 |
| 22.5 | 0.500736 | 0.499968 | 0 | 0 | 0 | 0 | 4.499712 | 4.499712 | 2.49984 | 12.49997 |
| 23 | 0.499968 | 0.500736 | 0 | 0 | 0 | 0 | 4.499712 | 4.499712 | 2.49984 | 12.49997 |
| 23.5 | 0.499968 | 0.499968 | 0 | 0 | 0 | 0 | 4.50048 | 4.499712 | 2.49984 | 12.49997 |
| 24 | 0.499968 | 0.499968 | 0 | 0 | 0 | 0 | 4.499712 | 4.50048 | 2.500608 | 12.50074 |
| 24.5 | 0.499968 | 0.499968 | 0 | 0 | 0 | 0 | 4.50048 | 4.499712 | 2.49984 | 12.49997 |
| 25 | 0.499968 | 0.499968 | 0 | 0 | 0 | 0 | 4.499712 | 4.50048 | 2.49984 | 12.49997 |

Next, we look at the first derivative of the 'bandwidth vs. time' curves. The first derivative of a function measures dependent variable's (clients' data flow rate, overall bandwidth) sensitivity to change due to an alteration of the independent variable (zombies' data flow rate). The plot of the first derivative further reaffirms the correlation between the zombies' high data transfer rates and the clients' modest data transfer rates and the overall bandwidth. The two smaller spikes at t=10s and t=50s shows the start and end of data transfer among the clients and the server. The two larger spikes at t=20s and t=40s are of more interest. It confirms that:

- The sudden change in the server's bandwidth is due to the attack from the zombies.
- Neither of the zombies are able to achieve their full data flow rate of 5 Mbps. The system is also not able to accommodate three 5 Mbps requests. There is a non-linear correlation between the DDoS attack and its impact on the server.
- Clients' data transfer rates are inversely proportional to this change. At t=20s, the data transfer rate of the zombies increases from 0 to 5 Mbps instantaneously. This rise inversely affects the data transfer rates of the clients and 4 clients' data transfer rates are reduced to 0 Mbps. At t=40s, the opposite happens.

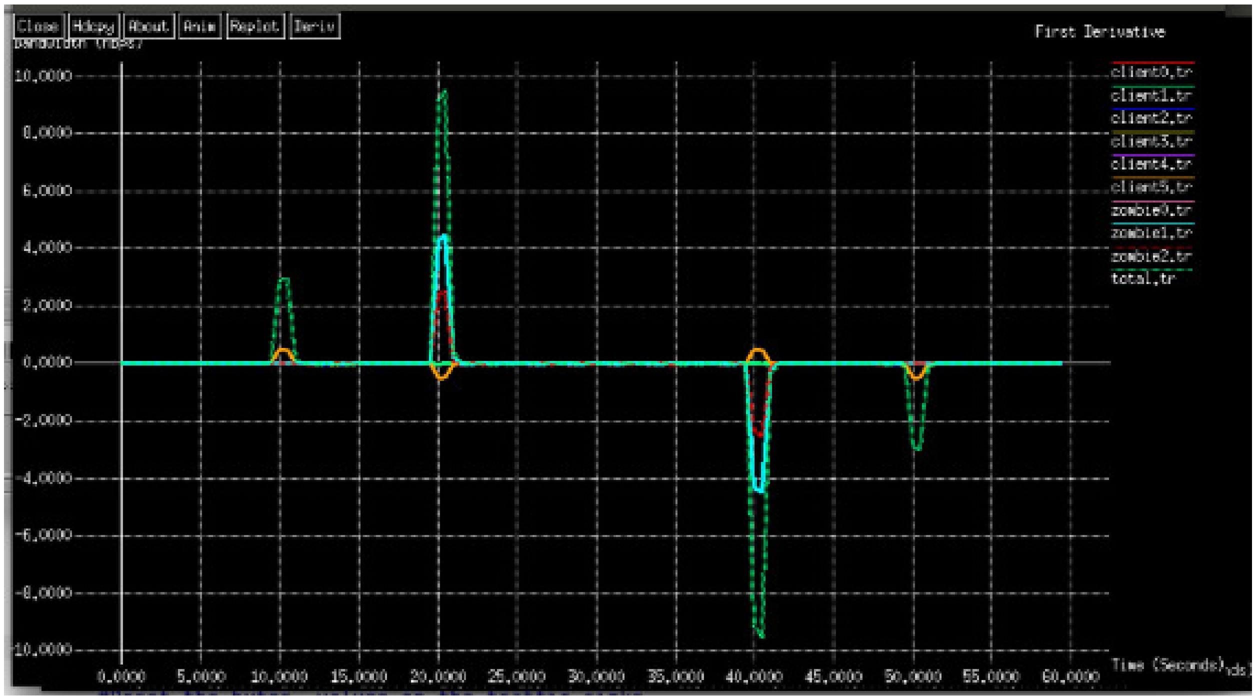

Figure 10: First derivative of the 'bandwidth vs. time' graph of a DDoS attack (0s < t < 60s).

## 10. Simulation of 'Black hole' prevention technique

### 10.1 Analysis

In this simulation, we assume the existence of a 'Black hole' as part of the system and simulate the previous scenario again. The simulation run time stays the same with 0s < t < 50s. Data flow starts at t=10s and continues until t=50s. DDoS attack occurs between 20s < t < 40s. Here is a ns-2 network topology of the system between 10s < t < 20s which is consistent with the previous simulation:

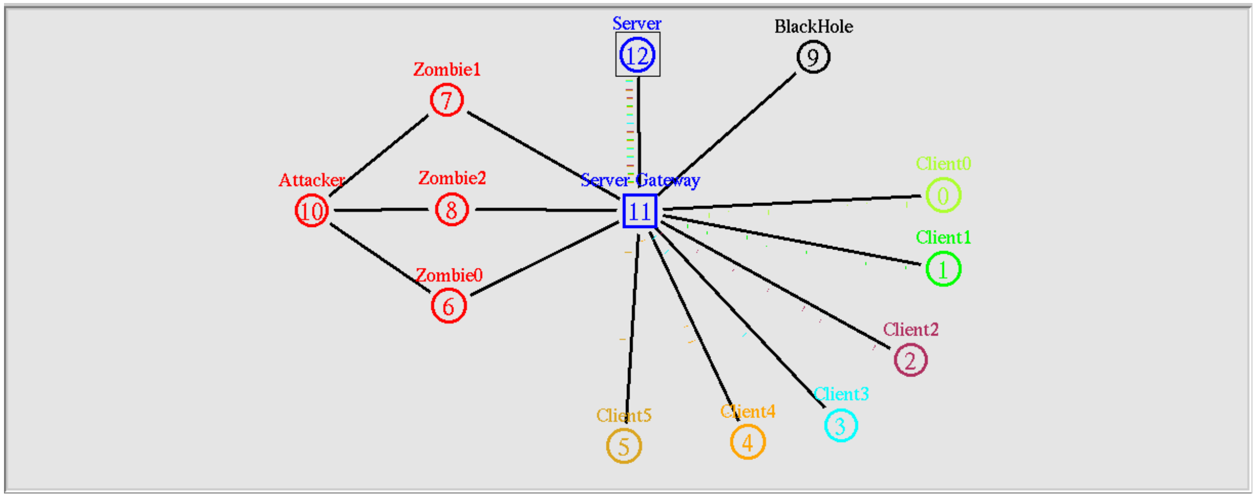

Figure 11: System with 'Black hole' prevention technique (10s < t < 20s).

At t=20s DDoS attack is initiated. It is interesting to note that, in the presence of a 'Black hole' the zombies' traffic are not reaching the server. The malicious requests are being redirected to the 'Black hole' instead and this continues until t=40s. All clients' data flow rates remain steady and undisturbed. The clients are not denied service. Here is a ns-2 network topology of the system between 20s < t < 40s:

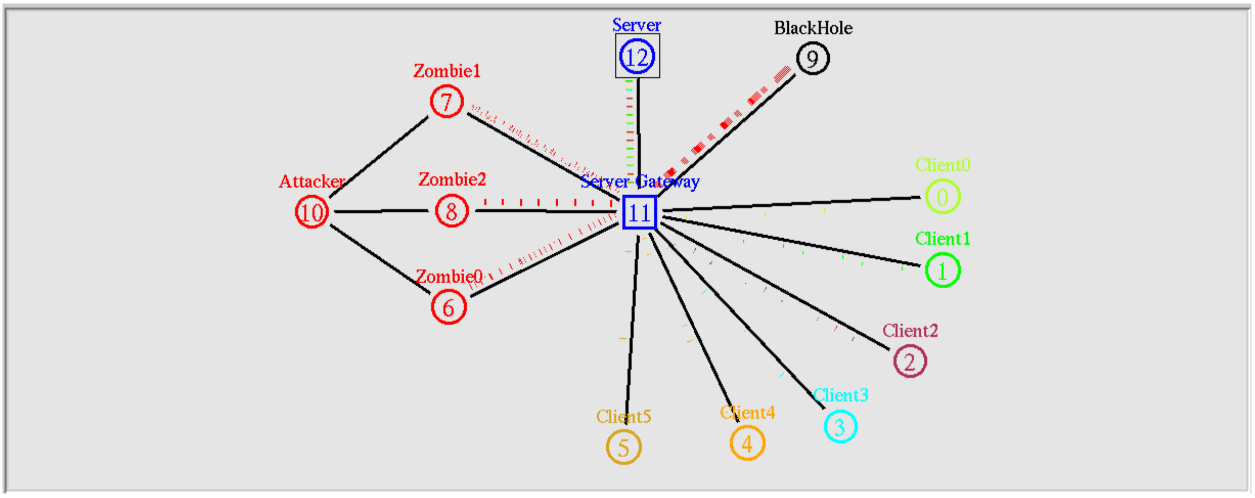

Figure 12: Successful prevention of DDoS attack using a 'Black hole' (20s < t < 40s).

## 10.2 Observations

We plot the complete simulation results for the duration of 0s < t < 60s and observe the 'bandwidth vs. time' graph below:

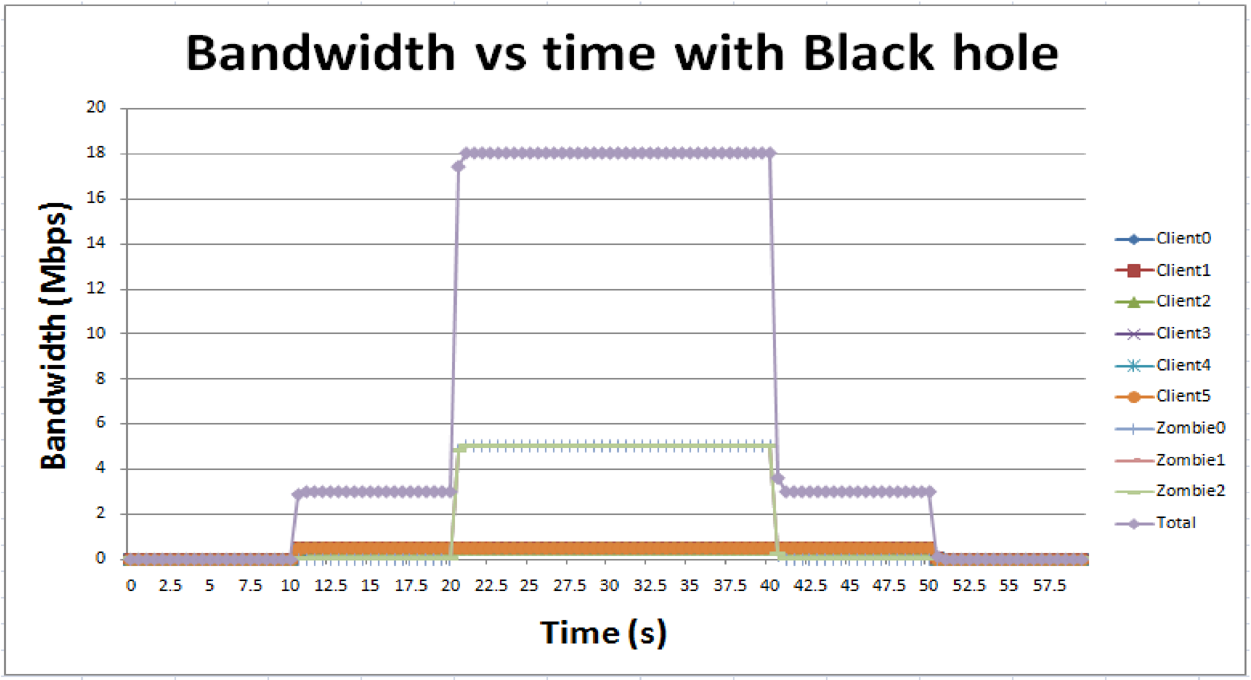

Figure 13: 'Bandwidth vs. time' graph using 'Black hole' prevention technique (0s < t < 60s).

Some key observations from the diagram are:

- Similar to the first scenario, there is no traffic flow to the server between 0s < t < 10s and 50s < t < 60s. The data flow from clients also stay constant at 0.5 Mbps between 10s < t < 20s and 40s < t < 50s.
- The 3 Zombies flood the server with a data flow rate of 5 Mbps between 20s < t < 40s. The X Graph confirms that the zombies are able to send the data at a rate of 5 Mbps. It is now possible because this data flow is not contributing to the overall limit of 12.5 Mbps bandwidth of the server. Instead it is being redirected towards the 'Black hole'.
- Further confirmation of the success of 'Black hole' technique comes from observing the total bandwidth of the system during 20s< t < 40s. The total bandwidth here is 18 Mbps, which is the summation of three 5 Mbps data flow rate contributed by the zombies and six 0.5 Mbps data flow rate of the clients. The 18 Mbps bandwidth shown on the X Graph is not the bandwidth of the link between the gateway and the server. Rather, this is the total bandwidth of the system during this time.
- The 6 clients show consistent data flow of 0.5 Mbps between 10s < t < 50s. Their data flow rates were not affected by the DDoS attack.

Furthermore, if we compare the first derivative of the 'bandwidth vs. time' graph with the one obtained earlier, we will note the same two smaller spikes at t=10s and t=50s are indicating initiation and termination of data transfers respectively. However, the two larger spikes at t=20s and t=40s are different from before and suggest that:

- The non-linear correlation between the zombies' data transfer rates and the system's overall bandwidth is not present and instead there is a linear relationship. The difference in magnitude of the spikes confirms this.
- There is no inverse relationship between the clients' data transfer rates and the DDoS attack anymore.

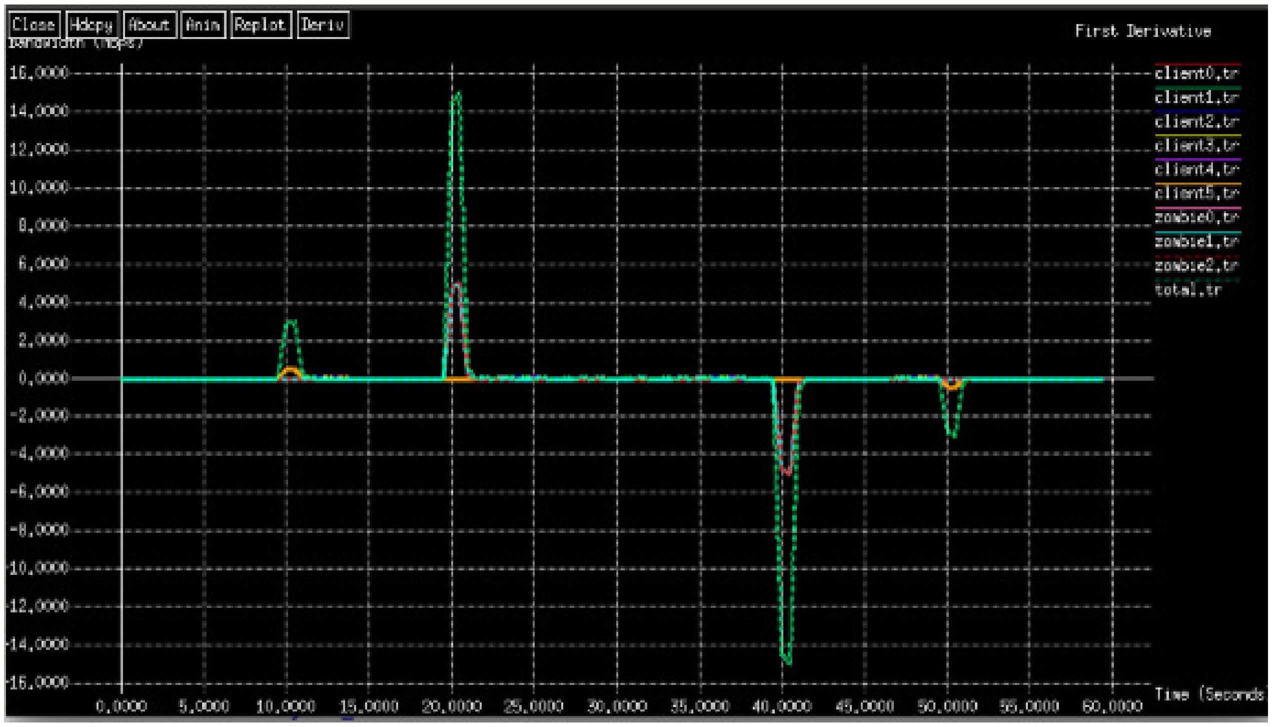

Figure 14: First derivative of the 'Bandwidth vs. time' graph using 'Black hole' prevention technique (0s < t < 60s).

## 11. Comparison of Drop tail and SFQ queuing methods

Keeping the same topologies of one attacker, three zombies, six clients, we simulated the DDoS attack again using Stochastic Fair Queuing (SFQ) method. The simulation run times were kept exactly the same.

### 11.1 Simulation parameters

The simulation parameters are:

- **Data Rates:**
    - Clients 0: 50 bytes at 0.1 Mbps
    - Clients 1: 50 bytes at 0.2 Mbps
    - Clients 2: 50 bytes at 0.3 Mbps
    - Clients 3: 50 bytes at 0.4 Mbps
    - Clients 4: 50 bytes at 0.5 Mbps
    - Clients 5: 50 bytes at 0.6 Mbps
    - Zombie 0: 50 bytes at 4.0 Mbps
    - Zombie 1: 50 bytes at 5.0 Mbps
    - Zombie 2: 50 bytes at 6.0 Mbps
    - Attacker: 50 bytes at 0.01 Mbps
- **Agent**: UDP

- **Application**: Constant Bit Rate (CBR)
- **Queue**: SFQ
- **Delay:** 10 ms
- **Bandwidth:**
    - o Client to Gateway: 45 Mbps (T3 Connection)
    - o Zombie to Gateway: 45 Mbps (T3 Connection)
    - o Gateway to Server: 12.5 Mbps

## 11.2 Analysis

The following were the data flow rates of the clients and zombies between 20.5s < t < 25s. The data flow rates stay same during the entire duration of the DDoS attack (t=20s to t=40s):

| Time (s) | Client 0 (Mbps) | Client 1 (Mbps) | Client 2 (Mbps) | Client 3 (Mbps) | Client 4 (Mbps) | Client 5 (Mbps) | Zombie 0 (Mbps) | Zombie 1 (Mbps) | Zombie 2 (Mbps) | Total (Mbps) |
|---|---|---|---|---|---|---|---|---|---|---|
| 20.5 | 0.100608 | 0.200448 | 0.300288 | 0.400128 | 0.499968 | 0.599808 | 3.327744 | 3.327744 | 3.327744 | 12.08448 |
| 21 | 0.09984 | 0.19968 | 0.300288 | 0.400128 | 0.499968 | 0.599808 | 3.466752 | 3.466752 | 3.466752 | 12.499968 |
| 21.5 | 0.09984 | 0.200448 | 0.29952 | 0.400128 | 0.499968 | 0.600576 | 3.465984 | 3.466752 | 3.466752 | 12.499968 |
| 22 | 0.09984 | 0.19968 | 0.300288 | 0.400128 | 0.499968 | 0.599808 | 3.466752 | 3.466752 | 3.466752 | 12.499968 |
| 22.5 | 0.09984 | 0.19968 | 0.29952 | 0.39936 | 0.499968 | 0.599808 | 3.46752 | 3.466752 | 3.46752 | 12.499968 |
| 23 | 0.100608 | 0.200448 | 0.300288 | 0.400128 | 0.499968 | 0.600576 | 3.465984 | 3.465984 | 3.465984 | 12.499968 |
| 23.5 | 0.09984 | 0.19968 | 0.300288 | 0.400128 | 0.499968 | 0.599808 | 3.466752 | 3.466752 | 3.466752 | 12.499968 |
| 24 | 0.09984 | 0.200448 | 0.29952 | 0.400128 | 0.499968 | 0.599808 | 3.466752 | 3.46752 | 3.466752 | 12.500736 |
| 24.5 | 0.09984 | 0.19968 | 0.300288 | 0.400128 | 0.499968 | 0.599808 | 3.466752 | 3.466752 | 3.466752 | 12.499968 |
| 25 | 0.09984 | 0.19968 | 0.29952 | 0.400128 | 0.499968 | 0.599808 | 3.46752 | 3.466752 | 3.466752 | 12.499968 |

It was interesting to note that, using the SFQ method, the clients' data rates were unaffected. Rather, the zombies were restricted in how much data they could send to the server. Each of their data rates were reduced to approximately 3.466 Mbps and DDoS did not occur.

This is consistent with the definition of the SFQ method. SFQ is based on a fair queuing algorithm. It uses a hashing algorithm which divides the traffic over a limited number of FIFO queues while being almost perfectly fair. When the three zombies started transmitting data at t=20s, all the six clients and three zombies were transferring data. However, the clients were already transmitting data from t=10s, so they were in first. As per the definition of the SFQ method, the clients' data were let out first. About the zombies' data rates, SFQ divided their data transfer rates almost equally in to three different queues and allowed them to transfer data at a reduced rate. Here is 'bandwidth vs time' graph of the data rates using SFQ method showing that DDoS did not occur:

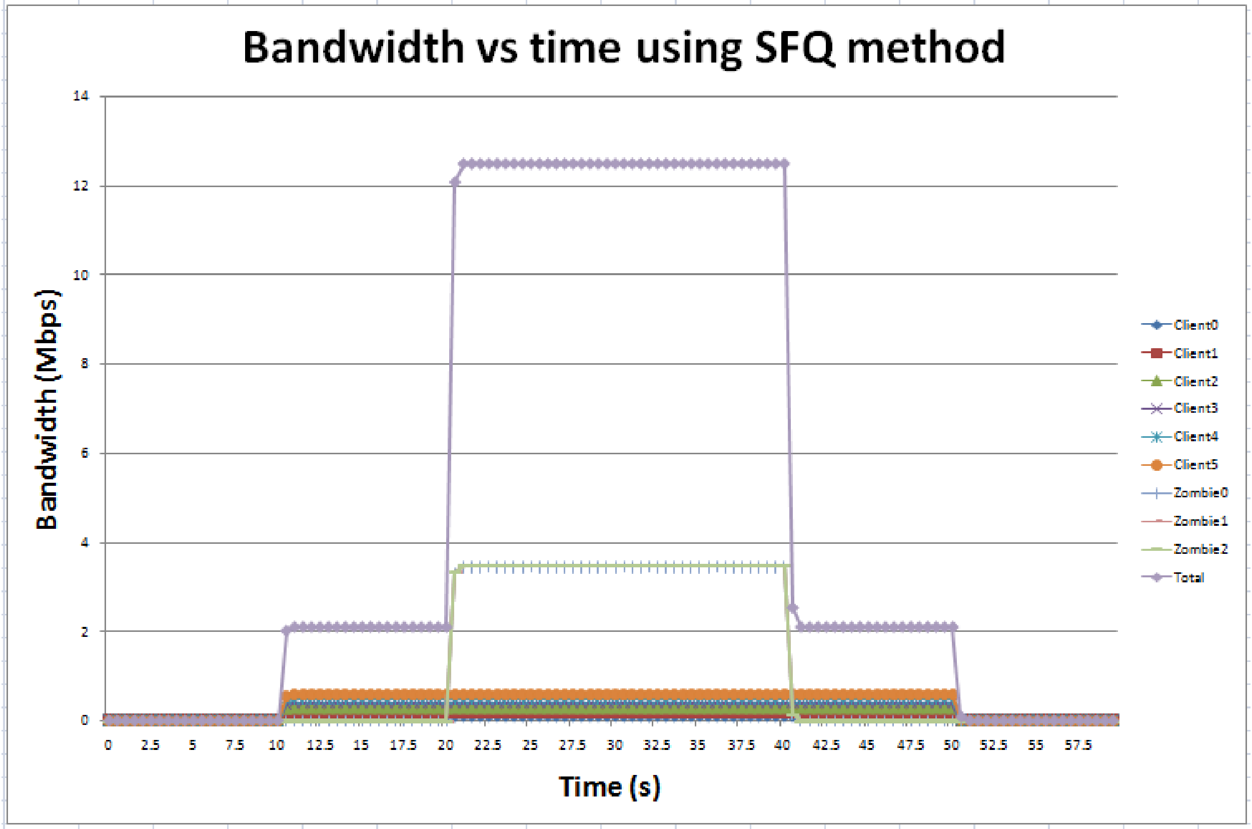

Figure 15: 'Bandwidth vs. time' graph using SFQ method (0s < t < 60s).

## 12. Scope of future work

DDoS attacks are modern day Internet threats that are worth further researching. This project is a small step towards that direction and there are numerous opportunities of future work. Due to limited resources and lack of opportunities we were unable to explore further, but the following are some key project opportunities that are worth considering:

- Using Drop tail queuing method, change the data rates of the clients and observe their effects on the DDoS attack with and without a 'Black hole'.
- Using SFQ queuing method, change the data rates of the clients and zombies and observe their effects on the DDoS attack.
- Using SFQ queuing method, initiate data transfer from a client after the DDoS attack to verify FIFO mechanism.
- Complete writing of C++ code in ns-2 to create a new agent/application model of DDoS attacks.

- Simulate larger, more realistic models of DDoS attacks using more network components.
- Simulate different DDoS attacking methods and compare their impacts.
- Compare the effectiveness of other prevention techniques.
- Create scenarios with other queuing disciplines and compare their performances.

## 13. Conclusion

In this project we had the opportunity to explore the modern day phenomena called DDoS attack. We identified what a DDoS attack is and the numerous varieties of a DDoS attack. There is no one size fits all solution to eliminate the threat of a DDoS attack and network administrators need to be particularly vigilant in their endeavors to limit its impacts. We simulated a simple Reflection method DDoS attack on a client-server model in ns-2 and noted its catastrophic effects. 'Black hole' prevention technique was a robust measure in nullifying the threat of this DDoS attack. Our analysis and observations showed how the client-server relation is impacted over time during a DDoS attack and in the presence of a 'Black hole' prevention method. We also compared Drop tail and SFQ methods and their impacts on a DDoS attack. Additionally, we learnt about TCL and ns-2, which was a plus. Our research has enlightened us about DDoS attacks and has paved the way for more future work in this field.

## References

[1] F. Lau, S. H. Rubin, M. H. Smith, and Lj. Trajkovic, "Distributed denial of service attacks," (invited paper) in *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics, SMC 2000*, Nashville, TN, Oct. 2000, pp. 2275-2280.

[2] X. Rui, M. W. Li, and Z. W. Ling, “SYN flooding detecting using negative selection algorithm based on eigen value sets,” May 2009, [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5138099

[3] M. Blagov, “DDoS Definition,” [Online]. Available: https://www.incapsula.com/ddos/ddos-attacks/

[4] M. Greis, “Tutorials for the Network Simulator 'ns',” [Online]. Available: http://www.isi.edu/nsnam/ns/tutorial/

[5] P. White, “How much traffic can a single server handle,” Mar. 2011, [Online]. Available: http://blog.whitesites.com/How-much-traffic-can-a-single-server-handle__634363981032706250_blog.htm

[6] M. Prince, “The DDoS that knocked Spamhaus Offline (And how we mitigated it),” Mar. 2013, [Online]. Available: https://blog.cloudflare.com/the-ddos-that-knocked-spamhaus-offline-and-ho/

[7] Computer Emergency Response Team (CERT) for the Software Engineer Institute, "Denial of Service Attacks," 1997, [Online]. Available: https://www.cert.org/information-for/denial_of_service.cfm?

[8] A. S. Tanenbaum, "Authentication Protocols" in *Computer Networks*, 4th edition, New Jersey, Prentice Hall, 2003, ch.*8*, sec. *7*, pp. 787-790.

[9] The Anti-Abuse Project, "DNS blacklists," [Online]. Available: http://www.anti-abuse.org/dns-blacklists/

[10] B. Cane, "Mitigating DoS attacks with a null (or black hole) route on Linux," Jan. 2013, [Online]. Available: http://bencane.com/2013/01/14/mitigating-dos-attacks-with-a-null-or-blackhole-route-on-linux/

[11] Global Dots, "DDoS mitigation," [Online]. Available: http://www.globaldots.com/knowledge-base/ddos-mitigation/

[12] E. Ahmed, "Working Mechanism of FQ, RED, SFQ, DRR and Drop-Tail Queues," [Online]. Available: https://sites.google.com/a/seecs.edu.pk/network-technologies-tcp-ip-suite/home/performance-analysis-of-impact-of-various-queuing-mechanisms-on-mpeg-traffic/working-mechanism-of-fq-red-sfq-drr-and-drop-tail-queues

[13] US Department of Homeland Security, "DDoS quick guide," Jan. 2014, [Online]. Available: https://www.us-cert.gov/sites/default/files/publications/DDoS%20Quick%20Guide.pdf

# Appendix

## Code Summary

### Run file

tcp-full.h

```
class AttackerFullTcpAgent : public FullTcpAgent {
public:
	AttackerFullTcpAgent();
protected:
	virtual void sendmsg(int nbytes, const char *flags);
};
```

tcp-full.cc

```
static class AttackerFullTcpClass : public TclClass {
public:
	AttackerFullTcpClass() : TclClass("Agent/TCP/FullTcp/Attacker") {}
	TclObject* create(int, const char*const*) {
		// ns-default sets reno_fastrecov_ to false?
		return (new AttackerFullTcpAgent());
	}
} class_attacker_full;

void
AttackerFullTcpAgent::sendmsg(int nbytes, const char *flags)
{

	//newstate(TCPS_SYN_SENT);	// sending a SYN now
	//sent(iss_, foutput(iss_, REASON_NORMAL));

	if (flags && strcmp(flags, "MSG_EOF") == 0)
		close_on_empty_ = TRUE;
	if (flags && strcmp(flags, "DAT_EOF") == 0)
		signal_on_empty_ = TRUE;

	if (nbytes == -1) {
		infinite_send_ = TRUE;
		newstate(TCPS_SYN_SENT);	// sending a SYN now
		sent(iss_, foutput(iss_, REASON_NORMAL));
		advance_bytes(0);
		//advance_bytes(0);
	} else
		//advance_bytes(nbytes);
		newstate(TCPS_SYN_SENT);	// sending a SYN now
		sent(iss_, foutput(iss_, REASON_NORMAL));
		advance_bytes(0);
}
```

ns-default.tcl

```
	Agent/TCP/FullTcp/Attacker instproc init {} {
		$self next
	}
```

makefile.vc / makefile.in

tcp/tcp-full.o

## Drop tail queuing method, Scenario One: One attacker, three zombies, and six clients

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows
$ns color 1 greenyellow
$ns color 2 green
$ns color 3 maroon
$ns color 4 cyan
$ns color 5 orange
$ns color 6 goldenrod
$ns color 7 red
$ns color 8 black
$ns color 9 blue

#Open the output files
#for {set i 0} {$i < 9} {incr i} {
#    set f($i) [open out($i).tr w]
#}
set f0 [open client0.tr w]
set f1 [open client1.tr w]
set f2 [open client2.tr w]
set f3 [open client3.tr w]
set f4 [open client4.tr w]
set f5 [open client5.tr w]
set f6 [open zombie0.tr w]
set f7 [open zombie1.tr w]
set f8 [open zombie2.tr w]
set f9 [open total.tr w]

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Create 5 nodes
set client0 [$ns node]
set client1 [$ns node]
set client2 [$ns node]
set client3 [$ns node]
set client4 [$ns node]
set client5 [$ns node]
set zombie0 [$ns node]
set zombie1 [$ns node]
set zombie2 [$ns node]
set attacker [$ns node]
set servergateway [$ns node]
set server [$ns node]

$client0 color "greenyellow"
$client0 label "Client0"
```

```
$client1 color "green"
$client1 label "Client1"

$client2 color "maroon"
$client2 label "Client2"

$client3 color "cyan"
$client3 label "Client3"

$client4 color "orange"
$client4 label "Client4"

$client5 color "goldenrod"
$client5 label "Client5"

$zombie0 color "red"
$zombie0 label "Zombie0"

$zombie1 color "red"
$zombie1 label "Zombie1"

$zombie2 color "red"
$zombie2 label "Zombie2"

$attacker color "red"
$attacker label "Attacker"

$servergateway color "blue"
$servergateway shape "square"
$servergateway label "Server Gateway"
$server color "blue"
$server label "Server"

#Connect the nodes
#Client Nodes to Server Gateway
$ns duplex-link $client0 $servergateway 45Mb 10ms DropTail
$ns duplex-link $client1 $servergateway 45Mb 10ms DropTail
$ns duplex-link $client2 $servergateway 45Mb 10ms DropTail
$ns duplex-link $client2 $servergateway 45Mb 10ms DropTail
$ns duplex-link $client3 $servergateway 45Mb 10ms DropTail
$ns duplex-link $client4 $servergateway 45Mb 10ms DropTail
$ns duplex-link $client5 $servergateway 45Mb 10ms DropTail

#Attacker Node to Zombies
$ns duplex-link $attacker $zombie0 45Mb 100ms DropTail
$ns duplex-link $attacker $zombie1 45Mb 100ms DropTail
$ns duplex-link $attacker $zombie2 45Mb 100ms DropTail

#Zombie Node to Server Gateway
$ns duplex-link $zombie0 $servergateway 45Mb 10ms DropTail
```

```
$ns duplex-link $zombie1 $servergateway 45Mb 10ms DropTail
$ns duplex-link $zombie2 $servergateway 45Mb 10ms DropTail
#Gateway to Server
$ns duplex-link $servergateway $server 12.5Mb 10ms DropTail

#Define a 'finish' procedure
proc finish {} {
        global f0 f1 f2 f3 f4 f5 f6 f7 f8 f9
        #Close the output files
        close $f0
        close $f1
        close $f2
        close $f3
        close $f4
        close $f5
        close $f6
        close $f7
        close $f8
        close $f9

        #Call xgraph to display the results
        #exec xgraph client0.tr client1.tr client2.tr client3.tr client4.tr client5.tr zombie0.tr
zombie1.tr zombie2.tr total.tr -geometry 900x500 -bg black -fg white -bw 100 -t "DDoS
Bandwidth of Server Results" -x "Time (Seconds)" -y "Bandwidth (Mbps)" -lw 3 &

    global ns nf
    $ns flush-trace
        #Close the trace file
    close $nf
        #Execute nam on the trace file
    exec nam out.nam &

    exit 0
}

#Define a procedure that attaches a UDP agent to a previously created node
#'node' and attaches an Expoo traffic generator to the agent with the
#characteristic values 'size' for packet size 'burst' for burst time,
#'idle' for idle time and 'rate' for burst peak rate. The procedure connects
#the source with the previously defined traffic sink 'sink' and returns the
#source object.
proc attach-expoo-traffic { node sink color size rate } {
        #Get an instance of the simulator
        set ns [Simulator instance]

        #Create a UDP agent and attach it to the node
        set source [new Agent/UDP]
        $ns attach-agent $node $source

        #Create an Expoo traffic agent and set its configuration parameters
        #set traffic [new Application/Traffic/Exponential]
        set traffic [new Application/Traffic/CBR]
        $traffic set packetSize_ $size
        #$traffic set burst_time_ $burst
        #$traffic set idle_time_ $idle
        $traffic set rate_ $rate
        $traffic set random_ 0

    # Attach traffic source to the traffic generator
    $traffic attach-agent $source
        $source set class_ $color
        #Connect the source and the sink
        $ns connect $source $sink
        return $traffic
}

#Define a procedure which periodically records the bandwidth received by the
#three traffic sinks sink0/1/2 and writes it to the three files f0/1/2.
proc record {} {
    global sink0 sink1 sink2 sink3 sink4 sink5 sinkz0 sinkz1 sinkz2 f0 f1 f2 f3 f4 f5 f6 f7 f8 f9
        #Get an instance of the simulator
        set ns [Simulator instance]
        #Set the time after which the procedure should be called again
        set time 0.5
        #How many bytes have been received by the traffic sinks?
    set bw0 [$sink0 set bytes_]
    set bw1 [$sink1 set bytes_]
    set bw2 [$sink2 set bytes_]
        set bw3 [$sink3 set bytes_]
        set bw4 [$sink4 set bytes_]
        set bw5 [$sink5 set bytes_]
        set bw6 [$sinkz0 set bytes_]
        set bw7 [$sinkz1 set bytes_]
        set bw8 [$sinkz2 set bytes_]



        #Get the current time
    set now [$ns now]
        #Calculate the bandwidth (in MBit/s) and write it to the files
    puts $f0 "$now [expr $bw0/$time*8/1000000]"
    puts $f1 "$now [expr $bw1/$time*8/1000000]"
    puts $f2 "$now [expr $bw2/$time*8/1000000]"
    puts $f3 "$now [expr $bw3/$time*8/1000000]"
    puts $f4 "$now [expr $bw4/$time*8/1000000]"
    puts $f5 "$now [expr $bw5/$time*8/1000000]"
    puts $f6 "$now [expr $bw6/$time*8/1000000]"
    puts $f7 "$now [expr $bw7/$time*8/1000000]"
    puts $f8 "$now [expr $bw8/$time*8/1000000]"
        puts $f9 "$now [expr
($bw0+$bw1+$bw2+$bw3+$bw4+$bw5+$bw6+$bw7+$bw8)/$time*8/1000000]"
```

```
        #Reset the bytes_ values on the traffic sinks
    $sink0 set bytes_ 0
    $sink1 set bytes_ 0
    $sink2 set bytes_ 0
    $sink3 set bytes_ 0
    $sink4 set bytes_ 0
        $sink5 set bytes_ 0
    $sinkz0 set bytes_ 0
    $sinkz1 set bytes_ 0
    $sinkz2 set bytes_ 0


        #Re-schedule the procedure
    $ns at [expr $now+$time] "record"
}


#Create traffic sinks and attach them to the node server
set sink0 [new Agent/LossMonitor]
set sink1 [new Agent/LossMonitor]
set sink2 [new Agent/LossMonitor]
set sink3 [new Agent/LossMonitor]
set sink4 [new Agent/LossMonitor]
set sink5 [new Agent/LossMonitor]

set sinkz0 [new Agent/LossMonitor]
set sinkz1 [new Agent/LossMonitor]
set sinkz2 [new Agent/LossMonitor]

#Sink on Zombies that Attacker sends a start byte too
set sinka0 [new Agent/LossMonitor]
set sinka1 [new Agent/LossMonitor]
set sinka2 [new Agent/LossMonitor]

$ns attach-agent $server $sink0
$ns attach-agent $server $sink1
$ns attach-agent $server $sink2
$ns attach-agent $server $sink3
$ns attach-agent $server $sink4
$ns attach-agent $server $sink5
$ns attach-agent $server $sinkz0
$ns attach-agent $server $sinkz1
$ns attach-agent $server $sinkz2
$ns attach-agent $zombie0 $sinka0
$ns attach-agent $zombie1 $sinka1
$ns attach-agent $zombie2 $sinka2

#Create three traffic sources
#-------------------------------  Node  Sink color size Rate
set source0 [attach-expoo-traffic $client0 $sink0 1 48 500Kb]
set source1 [attach-expoo-traffic $client1 $sink1 2 48 500Kb]
set source2 [attach-expoo-traffic $client2 $sink2 3 48 500Kb]
set source3 [attach-expoo-traffic $client3 $sink3 4 48 500Kb]
set source4 [attach-expoo-traffic $client4 $sink4 5 48 500Kb]
set source5 [attach-expoo-traffic $client5 $sink5 6 48 500Kb]

set sourcez0 [attach-expoo-traffic $zombie0 $sinkz0 7 48 5000Kb]
set sourcez1 [attach-expoo-traffic $zombie1 $sinkz1 7 48 5000Kb]
set sourcez2 [attach-expoo-traffic $zombie2 $sinkz2 7 48 5000Kb]

set sourcea0 [attach-expoo-traffic $attacker $sinka0 7 48 10Kb]
set sourcea1 [attach-expoo-traffic $attacker $sinka1 7 48 10Kb]
set sourcea2 [attach-expoo-traffic $attacker $sinka2 7 48 10Kb]


#Start logging the received bandwidth
$ns at 0.0 "record"
#Start the traffic sources
$ns at 10.0 "$source0 start"
$ns at 10.0 "$source1 start"
$ns at 10.0 "$source2 start"
$ns at 10.0 "$source3 start"
$ns at 10.0 "$source4 start"
$ns at 10.0 "$source5 start"

#Attacker Tells Zombies to Start DDoSing
$ns at 19.9 "$sourcea0 start"
$ns at 19.9 "$sourcea1 start"
$ns at 19.9 "$sourcea2 start"

#Zombies Start DDoSing
$ns at 20.0 "$sourcez0 start"
$ns at 20.0 "$sourcez1 start"
$ns at 20.0 "$sourcez2 start"

#Attacker Tells Zombies to Stop DDoSing
$ns at 39.9 "$sourcea0 stop"
$ns at 39.9 "$sourcea1 stop"
$ns at 39.9 "$sourcea2 stop"

#Zombies Stop DDoSing
$ns at 40.0 "$sourcez0 stop"
$ns at 40.0 "$sourcez1 stop"
$ns at 40.0 "$sourcez2 stop"

#Stop the traffic sources
$ns at 50.0 "$source0 stop"
$ns at 50.0 "$source1 stop"
$ns at 50.0 "$source2 stop"
$ns at 50.0 "$source3 stop"
$ns at 50.0 "$source4 stop"
$ns at 50.0 "$source5 stop"
```

```
#Call the finish procedure after 60 seconds simulation time
$ns at 60.0 "finish"

#Run the simulation
$ns run
```

## Drop tail queuing method, Scenario One: One attacker, three zombies, six clients, and one 'Black hole'

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows
$ns color 1 greenyellow
$ns color 2 green
$ns color 3 maroon
$ns color 4 cyan
$ns color 5 orange
$ns color 6 goldenrod
$ns color 7 red
$ns color 8 black
$ns color 9 blue

#Open the output files
#for {set i 0} {$i < 9} {incr i} {
#    set f($i) [open out($i).tr w]
#}
set f0 [open client0.tr w]
set f1 [open client1.tr w]
set f2 [open client2.tr w]
set f3 [open client3.tr w]
set f4 [open client4.tr w]
set f5 [open client5.tr w]
set f6 [open zombie0.tr w]
set f7 [open zombie1.tr w]
set f8 [open zombie2.tr w]
set f9 [open total.tr w]

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Create 5 nodes
set client0 [$ns node]
set client1 [$ns node]
set client2 [$ns node]
set client3 [$ns node]
set client4 [$ns node]
set client5 [$ns node]
set zombie0 [$ns node]
set zombie1 [$ns node]
set zombie2 [$ns node]
set blackhole [$ns node]
set attacker [$ns node]
set servergateway [$ns node]
set server [$ns node]
```

```
$client0 color "greenyellow"
$client0 label "Client0"

$client1 color "green"
$client1 label "Client1"

$client2 color "maroon"
$client2 label "Client2"

$client3 color "cyan"
$client3 label "Client3"

$client4 color "orange"
$client4 label "Client4"

$client5 color "goldenrod"
$client5 label "Client5"

$zombie0 color "red"
$zombie0 label "Zombie0"

$zombie1 color "red"
$zombie1 label "Zombie1"

$zombie2 color "red"
$zombie2 label "Zombie2"

$attacker color "red"
$attacker label "Attacker"

$blackhole color "black"
$blackhole label "BlackHole"

$servergateway color "blue"
$servergateway shape "square"
$servergateway label "Server Gateway"
$server color "blue"
$server label "Server"

#Connect the nodes
#Client Nodes to Server Gateway
$ns duplex-link $client0 $servergateway 45Mb 10ms DropTail
$ns duplex-link $client1 $servergateway 45Mb 10ms DropTail
$ns duplex-link $client2 $servergateway 45Mb 10ms DropTail
$ns duplex-link $client2 $servergateway 45Mb 10ms DropTail
$ns duplex-link $client3 $servergateway 45Mb 10ms DropTail
$ns duplex-link $client4 $servergateway 45Mb 10ms DropTail
$ns duplex-link $client5 $servergateway 45Mb 10ms DropTail

#Attacker Node to Zombies
$ns duplex-link $attacker $zombie0 45Mb 100ms DropTail
```

```
$ns duplex-link $attacker $zombie1 45Mb 100ms DropTail
$ns duplex-link $attacker $zombie2 45Mb 100ms DropTail

#Zombie Node to Server Gateway
$ns duplex-link $zombie0 $servergateway 45Mb 10ms DropTail
$ns duplex-link $zombie1 $servergateway 45Mb 10ms DropTail
$ns duplex-link $zombie2 $servergateway 45Mb 10ms DropTail

#Gateway to BlackHole
$ns duplex-link $servergateway $blackhole 45Mb 10ms DropTail

#Gateway to Server
$ns duplex-link $servergateway $server 12.5Mb 10ms DropTail

#Define a 'finish' procedure
proc finish {} {
	global f0 f1 f2 f3 f4 f5 f6 f7 f8 f9
	#Close the output files
	close $f0
	close $f1
	close $f2
	close $f3
	close $f4
	close $f5
	close $f6
	close $f7
	close $f8
	close $f9

	#Call xgraph to display the results
	#exec xgraph client0.tr client1.tr client2.tr client3.tr client4.tr client5.tr zombie0.tr
zombie1.tr zombie2.tr total.tr -geometry 900x500 -bg black -fg white -bw 100 -t "DDoS
Bandwidth of Server Results with Blacklist" -x "Time (Seconds)" -y "Bandwidth (Mbps)" -lw 3 &

    global ns nf
    $ns flush-trace
	#Close the trace file
    close $nf
	#Execute nam on the trace file
    exec nam out.nam &

    exit 0
}

#Define a procedure that attaches a UDP agent to a previously created node
#'node' and attaches an Expoo traffic generator to the agent with the
#characteristic values 'size' for packet size 'burst' for burst time,
#'idle' for idle time and 'rate' for burst peak rate. The procedure connects
#the source with the previously defined traffic sink 'sink' and returns the
#source object.
proc attach-expoo-traffic { node sink color size rate } {
        #Get an instance of the simulator
        set ns [Simulator instance]

        #Create a UDP agent and attach it to the node
        set source [new Agent/UDP]
        $ns attach-agent $node $source

        #Create an Expoo traffic agent and set its configuration parameters
        #set traffic [new Application/Traffic/Exponential]
        set traffic [new Application/Traffic/CBR]
        $traffic set packetSize_ $size
        #$traffic set burst_time_ $burst
        #$traffic set idle_time_ $idle
        $traffic set rate_ $rate
        $traffic set random_ 0

    # Attach traffic source to the traffic generator
    $traffic attach-agent $source
        $source set class_ $color
        #Connect the source and the sink
        $ns connect $source $sink
        return $traffic
}

#Define a procedure which periodically records the bandwidth received by the
#three traffic sinks sink0/1/2 and writes it to the three files f0/1/2.
proc record {} {
    global sink0 sink1 sink2 sink3 sink4 sink5 sinkz0 sinkz1 sinkz2 f0 f1 f2 f3 f4 f5 f6 f7 f8 f9
        #Get an instance of the simulator
        set ns [Simulator instance]
        #Set the time after which the procedure should be called again
    set time 0.5
        #How many bytes have been received by the traffic sinks?
    set bw0 [$sink0 set bytes_]
    set bw1 [$sink1 set bytes_]
    set bw2 [$sink2 set bytes_]
        set bw3 [$sink3 set bytes_]
        set bw4 [$sink4 set bytes_]
        set bw5 [$sink5 set bytes_]
        set bw6 [$sinkz0 set bytes_]
        set bw7 [$sinkz1 set bytes_]
        set bw8 [$sinkz2 set bytes_]

        #Get the current time
    set now [$ns now]
        #Calculate the bandwidth (in MBit/s) and write it to the files
    puts $f0 "$now [expr $bw0/$time*8/1000000]"
    puts $f1 "$now [expr $bw1/$time*8/1000000]"
    puts $f2 "$now [expr $bw2/$time*8/1000000]"
    puts $f3 "$now [expr $bw3/$time*8/1000000]"
    puts $f4 "$now [expr $bw4/$time*8/1000000]"
```

```
   puts $f5 "$now [expr $bw5/$time*8/1000000]"
   puts $f6 "$now [expr $bw6/$time*8/1000000]"
   puts $f7 "$now [expr $bw7/$time*8/1000000]"
   puts $f8 "$now [expr $bw8/$time*8/1000000]"
        puts $f9 "$now [expr
($bw0+$bw1+$bw2+$bw3+$bw4+$bw5+$bw6+$bw7+$bw8)/$time*8/1000000]"

        #Reset the bytes_ values on the traffic sinks
   $sink0 set bytes_ 0
   $sink1 set bytes_ 0
   $sink2 set bytes_ 0
   $sink3 set bytes_ 0
   $sink4 set bytes_ 0
        $sink5 set bytes_ 0
   $sinkz0 set bytes_ 0
   $sinkz1 set bytes_ 0
   $sinkz2 set bytes_ 0


        #Re-schedule the procedure
   $ns at [expr $now+$time] "record"
}


#Create traffic sinks and attach them to the node server
set sink0 [new Agent/LossMonitor]
set sink1 [new Agent/LossMonitor]
set sink2 [new Agent/LossMonitor]
set sink3 [new Agent/LossMonitor]
set sink4 [new Agent/LossMonitor]
set sink5 [new Agent/LossMonitor]

set sinkz0 [new Agent/LossMonitor]
set sinkz1 [new Agent/LossMonitor]
set sinkz2 [new Agent/LossMonitor]

#Sink on Zombies that Attacker sends a start byte too
set sinka0 [new Agent/LossMonitor]
set sinka1 [new Agent/LossMonitor]
set sinka2 [new Agent/LossMonitor]

$ns attach-agent $server $sink0
$ns attach-agent $server $sink1
$ns attach-agent $server $sink2
$ns attach-agent $server $sink3
$ns attach-agent $server $sink4
$ns attach-agent $server $sink5
$ns attach-agent $blackhole $sinkz0
$ns attach-agent $blackhole $sinkz1
$ns attach-agent $blackhole $sinkz2
$ns attach-agent $zombie0 $sinka0
```

```
$ns attach-agent $zombie1 $sinka1
$ns attach-agent $zombie2 $sinka2

#Create three traffic sources
#------------------------------  Node  Sink color size Rate
set source0 [attach-expoo-traffic $client0 $sink0 1 48 500Kb]
set source1 [attach-expoo-traffic $client1 $sink1 2 48 500Kb]
set source2 [attach-expoo-traffic $client2 $sink2 3 48 500Kb]
set source3 [attach-expoo-traffic $client3 $sink3 4 48 500Kb]
set source4 [attach-expoo-traffic $client4 $sink4 5 48 500Kb]
set source5 [attach-expoo-traffic $client5 $sink5 6 48 500Kb]

set sourcez0 [attach-expoo-traffic $zombie0 $sinkz0 7 48 5000Kb]
set sourcez1 [attach-expoo-traffic $zombie1 $sinkz1 7 48 5000Kb]
set sourcez2 [attach-expoo-traffic $zombie2 $sinkz2 7 48 5000Kb]

set sourcea0 [attach-expoo-traffic $attacker $sinka0 7 48 10Kb]
set sourcea1 [attach-expoo-traffic $attacker $sinka1 7 48 10Kb]
set sourcea2 [attach-expoo-traffic $attacker $sinka2 7 48 10Kb]


#Start logging the received bandwidth
$ns at 0.0 "record"
#Start the traffic sources
$ns at 10.0 "$source0 start"
$ns at 10.0 "$source1 start"
$ns at 10.0 "$source2 start"
$ns at 10.0 "$source3 start"
$ns at 10.0 "$source4 start"
$ns at 10.0 "$source5 start"

#Attacker Tells Zombies to Start DDoSing
$ns at 19.9 "$sourcea0 start"
$ns at 19.9 "$sourcea1 start"
$ns at 19.9 "$sourcea2 start"

#Zombies Start DDoSing
$ns at 20.0 "$sourcez0 start"
$ns at 20.0 "$sourcez1 start"
$ns at 20.0 "$sourcez2 start"

#Attacker Tells Zombies to Stop DDoSing
$ns at 39.9 "$sourcea0 stop"
$ns at 39.9 "$sourcea1 stop"
$ns at 39.9 "$sourcea2 stop"

#Zombies Stop DDoSing
$ns at 40.0 "$sourcez0 stop"
$ns at 40.0 "$sourcez1 stop"
$ns at 40.0 "$sourcez2 stop"
```

```
#Stop the traffic sources
$ns at 50.0 "$source0 stop"
$ns at 50.0 "$source1 stop"
$ns at 50.0 "$source2 stop"
$ns at 50.0 "$source3 stop"
$ns at 50.0 "$source4 stop"
$ns at 50.0 "$source5 stop"

#Call the finish procedure after 60 seconds simulation time
$ns at 60.0 "finish"

#Run the simulation
$ns run
```

## SFQ method: One attacker, three zombies, and six clients

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows
$ns color 1 greenyellow
$ns color 2 green
$ns color 3 maroon
$ns color 4 cyan
$ns color 5 orange
$ns color 6 goldenrod
$ns color 7 red
$ns color 8 black
$ns color 9 blue

#Open the output files
#for {set i 0} {$i < 9} {incr i} {
#    set f($i) [open out($i).tr w]
#}
set f0 [open client0.tr w]
set f1 [open client1.tr w]
set f2 [open client2.tr w]
set f3 [open client3.tr w]
set f4 [open client4.tr w]
set f5 [open client5.tr w]
set f6 [open zombie0.tr w]
set f7 [open zombie1.tr w]
set f8 [open zombie2.tr w]
set f9 [open total.tr w]

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Create 5 nodes
set client0 [$ns node]
set client1 [$ns node]
set client2 [$ns node]
set client3 [$ns node]
set client4 [$ns node]
set client5 [$ns node]
set zombie0 [$ns node]
set zombie1 [$ns node]
set zombie2 [$ns node]
set attacker [$ns node]
set servergateway [$ns node]
set server [$ns node]

$client0 color "greenyellow"
$client0 label "Client0"
```

```
$client1 color "green"
$client1 label "Client1"

$client2 color "maroon"
$client2 label "Client2"

$client3 color "cyan"
$client3 label "Client3"

$client4 color "orange"
$client4 label "Client4"

$client5 color "goldenrod"
$client5 label "Client5"

$zombie0 color "red"
$zombie0 label "Zombie0"

$zombie1 color "red"
$zombie1 label "Zombie1"

$zombie2 color "red"
$zombie2 label "Zombie2"

$attacker color "red"
$attacker label "Attacker"

$servergateway color "blue"
$servergateway shape "square"
$servergateway label "Server Gateway"
$server color "blue"
$server label "Server"

#Connect the nodes
#Client Nodes to Server Gateway
$ns duplex-link $client0 $servergateway 45Mb 10ms SFQ
$ns duplex-link $client1 $servergateway 45Mb 10ms SFQ
$ns duplex-link $client2 $servergateway 45Mb 10ms SFQ
$ns duplex-link $client2 $servergateway 45Mb 10ms SFQ
$ns duplex-link $client3 $servergateway 45Mb 10ms SFQ
$ns duplex-link $client4 $servergateway 45Mb 10ms SFQ
$ns duplex-link $client5 $servergateway 45Mb 10ms SFQ

#Attacker Node to Zombies
$ns duplex-link $attacker $zombie0 45Mb 100ms SFQ
$ns duplex-link $attacker $zombie1 45Mb 100ms SFQ
$ns duplex-link $attacker $zombie2 45Mb 100ms SFQ

#Zombie Node to Server Gateway
$ns duplex-link $zombie0 $servergateway 45Mb 10ms SFQ
```

```
$ns duplex-link $zombie1 $servergateway 45Mb 10ms SFQ
$ns duplex-link $zombie2 $servergateway 45Mb 10ms SFQ
#Gateway to Server
$ns duplex-link $servergateway $server 12.5Mb 10ms SFQ

#Define a 'finish' procedure
proc finish {} {
        global f0 f1 f2 f3 f4 f5 f6 f7 f8 f9
        #Close the output files
        close $f0
        close $f1
        close $f2
        close $f3
        close $f4
        close $f5
        close $f6
        close $f7
        close $f8
        close $f9

        #Call xgraph to display the results
        #exec xgraph client0.tr client1.tr client2.tr client3.tr client4.tr client5.tr zombie0.tr
zombie1.tr zombie2.tr total.tr -geometry 900x500 -bg black -fg white -bw 100 -t "DDoS
Bandwidth of Server Results" -x "Time (Seconds)" -y "Bandwidth (Mbps)" -lw 3 &

   global ns nf
   $ns flush-trace
        #Close the trace file
   close $nf
        #Execute nam on the trace file
   exec nam out.nam &

   exit 0
}

#Define a procedure that attaches a UDP agent to a previously created node
#'node' and attaches an Expoo traffic generator to the agent with the
#characteristic values 'size' for packet size 'burst' for burst time,
#'idle' for idle time and 'rate' for burst peak rate. The procedure connects
#the source with the previously defined traffic sink 'sink' and returns the
#source object.
proc attach-expoo-traffic { node sink color size rate } {
        #Get an instance of the simulator
        set ns [Simulator instance]

        #Create a UDP agent and attach it to the node
        set source [new Agent/UDP]
        $ns attach-agent $node $source

        #Create an Expoo traffic agent and set its configuration parameters
        #set traffic [new Application/Traffic/Exponential]
        set traffic [new Application/Traffic/CBR]
        $traffic set packetSize_ $size
        #$traffic set burst_time_ $burst
        #$traffic set idle_time_ $idle
        $traffic set rate_ $rate
        $traffic set random_ 0

   # Attach traffic source to the traffic generator
   $traffic attach-agent $source
        $source set class_ $color
        #Connect the source and the sink
        $ns connect $source $sink
        return $traffic
}

#Define a procedure which periodically records the bandwidth received by the
#three traffic sinks sink0/1/2 and writes it to the three files f0/1/2.
proc record {} {
   global sink0 sink1 sink2 sink3 sink4 sink5 sinkz0 sinkz1 sinkz2 f0 f1 f2 f3 f4 f5 f6 f7 f8 f9
        #Get an instance of the simulator
        set ns [Simulator instance]
        #Set the time after which the procedure should be called again
        set time 0.5
        #How many bytes have been received by the traffic sinks?
   set bw0 [$sink0 set bytes_]
   set bw1 [$sink1 set bytes_]
   set bw2 [$sink2 set bytes_]
        set bw3 [$sink3 set bytes_]
        set bw4 [$sink4 set bytes_]
        set bw5 [$sink5 set bytes_]
        set bw6 [$sinkz0 set bytes_]
        set bw7 [$sinkz1 set bytes_]
        set bw8 [$sinkz2 set bytes_]



        #Get the current time
   set now [$ns now]
        #Calculate the bandwidth (in MBit/s) and write it to the files
   puts $f0 "$now [expr $bw0/$time*8/1000000]"
   puts $f1 "$now [expr $bw1/$time*8/1000000]"
   puts $f2 "$now [expr $bw2/$time*8/1000000]"
   puts $f3 "$now [expr $bw3/$time*8/1000000]"
   puts $f4 "$now [expr $bw4/$time*8/1000000]"
   puts $f5 "$now [expr $bw5/$time*8/1000000]"
   puts $f6 "$now [expr $bw6/$time*8/1000000]"
   puts $f7 "$now [expr $bw7/$time*8/1000000]"
   puts $f8 "$now [expr $bw8/$time*8/1000000]"
        puts $f9 "$now [expr
($bw0+$bw1+$bw2+$bw3+$bw4+$bw5+$bw6+$bw7+$bw8)/$time*8/1000000]"
```

```
        #Reset the bytes_ values on the traffic sinks
    $sink0 set bytes_ 0
    $sink1 set bytes_ 0
    $sink2 set bytes_ 0
    $sink3 set bytes_ 0
    $sink4 set bytes_ 0
        $sink5 set bytes_ 0
    $sinkz0 set bytes_ 0
    $sinkz1 set bytes_ 0
    $sinkz2 set bytes_ 0


        #Re-schedule the procedure
    $ns at [expr $now+$time] "record"
}


#Create traffic sinks and attach them to the node server
set sink0 [new Agent/LossMonitor]
set sink1 [new Agent/LossMonitor]
set sink2 [new Agent/LossMonitor]
set sink3 [new Agent/LossMonitor]
set sink4 [new Agent/LossMonitor]
set sink5 [new Agent/LossMonitor]

set sinkz0 [new Agent/LossMonitor]
set sinkz1 [new Agent/LossMonitor]
set sinkz2 [new Agent/LossMonitor]

#Sink on Zombies that Attacker sends a start byte too
set sinka0 [new Agent/LossMonitor]
set sinka1 [new Agent/LossMonitor]
set sinka2 [new Agent/LossMonitor]

$ns attach-agent $server $sink0
$ns attach-agent $server $sink1
$ns attach-agent $server $sink2
$ns attach-agent $server $sink3
$ns attach-agent $server $sink4
$ns attach-agent $server $sink5
$ns attach-agent $server $sinkz0
$ns attach-agent $server $sinkz1
$ns attach-agent $server $sinkz2
$ns attach-agent $zombie0 $sinka0
$ns attach-agent $zombie1 $sinka1
$ns attach-agent $zombie2 $sinka2

#Create three traffic sources
#------------------------------  Node  Sink color size Rate
set source0 [attach-expoo-traffic $client0 $sink0 1 48 100Kb]
set source1 [attach-expoo-traffic $client1 $sink1 2 48 200Kb]
set source2 [attach-expoo-traffic $client2 $sink2 3 48 300Kb]
set source3 [attach-expoo-traffic $client3 $sink3 4 48 400Kb]
set source4 [attach-expoo-traffic $client4 $sink4 5 48 500Kb]
set source5 [attach-expoo-traffic $client5 $sink5 6 48 600Kb]

set sourcez0 [attach-expoo-traffic $zombie0 $sinkz0 7 48 4000Kb]
set sourcez1 [attach-expoo-traffic $zombie1 $sinkz1 7 48 5000Kb]
set sourcez2 [attach-expoo-traffic $zombie2 $sinkz2 7 48 6000Kb]

set sourcea0 [attach-expoo-traffic $attacker $sinka0 7 48 10Kb]
set sourcea1 [attach-expoo-traffic $attacker $sinka1 7 48 10Kb]
set sourcea2 [attach-expoo-traffic $attacker $sinka2 7 48 10Kb]


#Start logging the received bandwidth
$ns at 0.0 "record"
#Start the traffic sources
$ns at 10.0 "$source0 start"
$ns at 10.0 "$source1 start"
$ns at 10.0 "$source2 start"
$ns at 10.0 "$source3 start"
$ns at 10.0 "$source4 start"
$ns at 10.0 "$source5 start"

#Attacker Tells Zombies to Start DDoSing
$ns at 19.9 "$sourcea0 start"
$ns at 19.9 "$sourcea1 start"
$ns at 19.9 "$sourcea2 start"

#Zombies Start DDoSing
$ns at 20.0 "$sourcez0 start"
$ns at 20.0 "$sourcez1 start"
$ns at 20.0 "$sourcez2 start"

#Attacker Tells Zombies to Stop DDoSing
$ns at 39.9 "$sourcea0 stop"
$ns at 39.9 "$sourcea1 stop"
$ns at 39.9 "$sourcea2 stop"

#Zombies Stop DDoSing
$ns at 40.0 "$sourcez0 stop"
$ns at 40.0 "$sourcez1 stop"
$ns at 40.0 "$sourcez2 stop"

#Stop the traffic sources
$ns at 50.0 "$source0 stop"
$ns at 50.0 "$source1 stop"
$ns at 50.0 "$source2 stop"
$ns at 50.0 "$source3 stop"
$ns at 50.0 "$source4 stop"
$ns at 50.0 "$source5 stop"
```

```
#Call the finish procedure after 60 seconds simulation time
$ns at 60.0 "finish"

#Run the simulation
$ns run
```