DDoS Attack Analysis and Prevention Measures

HTTP://WWW.SFU.CA/~AHMADA/ENSC427/MAIN.HTML

Group 5: Mohammad Ahmad (ahmada@sfu.ca), Ryadh Almuaili (ralmuail@sfu.com) | ENSC 427 | March 19, 2017

Table of Contents

List of Figures ii
List of Tables iii
Glossary iv
1.Abstract 1
2.Introduction 1
3.DDoS Attack Overview 3
4. Types of Attacks 4
5. Attack Methods 4
  5.1 DoS Attack Methods 4
  5.2 DDoS Attack Methods 5
6. Example of a DDoS Attack: BBC website attack 6
7. Importance of analyzing DDoS attacks 6
8. Prevention Methods 7
9. Queuing Algorithms 8
  9.1 DropTail 8
  9.2 RED 8
  9.3 SFQ 9
10. Related Work 10
11. Simulation 10
  11.1 Simulation Topology 11
  11.2 Simulation Parameters 11
  11.3 Simulation Run Time 12
  11.4 Analysis 14
  11.4.1 System performance using DropTail queuing algorithm 14
  11.4.2 System performance using RED queuing algorithm 16
  11.4.3 System performance using SFQ queuing algorithm 17
  11.4.4 DropTail vs RED vs SFQ 17

12. Scope of Future Work 21
13. Conclusion 21
APPENDIX I
CODE LISTING I
DROPTAIL I
RED XIX
SFQ XXXVIII

List of Figures

Figure 1 : Conceptual diagram of DDoS [10] 2
Figure 2 General structure of DDoS attack [1][2] 3
Figure 3 Expected number of connected devices between 2014 - 2021 [15] 7
Figure 4 DropTail Queue algorithm [13] 8
Figure 5 RED Queue algorithm [14] 9
Figure 6 SFQ Queue algorithm [14] 9
Figure 7 Simulation topology before attack 11
Figure 8 Simulation topology before attack 13
Figure 9 Simulation topology after attack 14
Figure 10 Simulation results when DropTail queuing is used 15
Figure11 Simulation results when RED queuing is used 16
Figure 12 Simulation results when SFQ queuing is used 17
Figure 13 Client 4 simulation results using the three different queues 18

List of Tables

Table 1 : Client 4 BW using different queues..20

Glossary

DDoS:	Distributed Denial of Service
DoS:	Denial of Service
SFQ:	Stochastic Fairness Queuing
RED:	Random Early Detection
ICMP:	Internet Control Message Protocol
IP:	Internet Protocol
TCP:	Transmission Control Protocol
SYN:	Synchronize
ACK:	Acknowledge
UDP:	User Datagram Protocol
BW:	Bandwidth
TFN:	Tribe Flood Network
IoT:	Internet of Things
FIFO:	First In First Out
Mbps:	Megabits per second
ns-2:	Network Simulator 2
SFU:	Simon Fraser University
UoC:	University of Calgary
SPAWAR:	Space and Naval Warfare Systems Command

1.Abstract

Distributed Denial of Service (DDOS) attacks have become a major problem for service providers. They undermine the capability to provide efficient and reliable application and services. In this project, using ns-2 we simulated and analyzed a DDOS attack which exploits the bandwidth limitations of a server. We then implemented three different queuing algorithms to mitigate the BW loss to legitimate users. We found that using SFQ queuing algorithm was most effective in ensuring that users got their fair share of connection to the server. RED queueing was second in our BW analysis and Droptail came in last. In future work we suggest use of a bigger topology that is more consistent with an actual DDoS attack and analysis of constant performance achieved by the first clients of each router.

2.Introduction

Distributed denial-of-service (DDoS) attacks are becoming a global issue to businesses nowadays. They are a constant threat to organizations and institutions by threatening service performance and shutting down websites and mail servers. DDoS attack is a type of DoS attack where multiple compromised systems are being used to target a single system. The compromised systems traffic is flooded which leads to service denial to legitimate users. There are different types of DDoS attacks such as Traffic attacks, Bandwidth attacks and Application attacks.

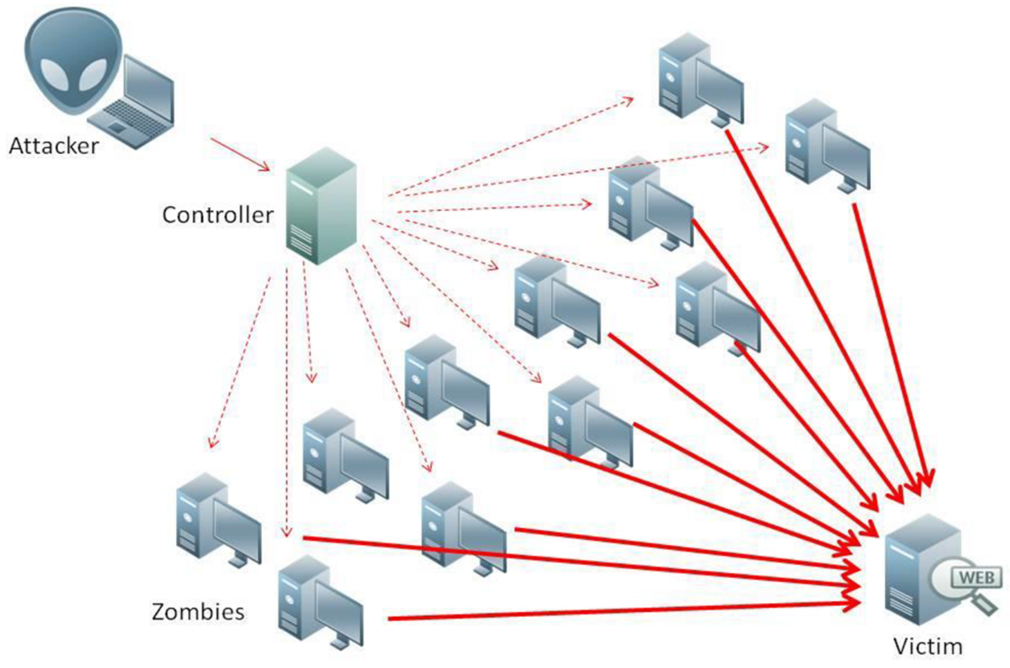


Figure 1 : Conceptual diagram of DDoS [10]

In this project, we examined:

- Various attacks methods
- Approaches to their attacks
- Prevention methods
- Review attack on BBC website
- Why DDoS is an important topic to address?
- Previous work were DDoS attacks are analyzed

The main scope of this project we will be using ns-2 to:

- Simulate a bandwidth DDoS attack and analyze the effect and damages it can cause to the services provided by a system
- Simulate the DDoS attack using DropTail, SFQ and RED queuing algorithms under the same simulation parameters
- Compare the performance of the system after applying the three different queuing algorithms

3.DDoS Attack Overview

A real attacker deploys daemon attack programs in multiple host computers, and deploys a master program, that controls and coordinate the daemons, in another host computer. When the real attacker wants to launch an attack, an execute command is sent to the control master program which will then execute all the daemons under its control. After that, the daemons will attack the victim. The four main components of an attack are:

1. A real attacker
2. A control master program
3. Attack daemon agents
4. The victim

The general structure of a DDoS attack is shown in Figure 2 below.

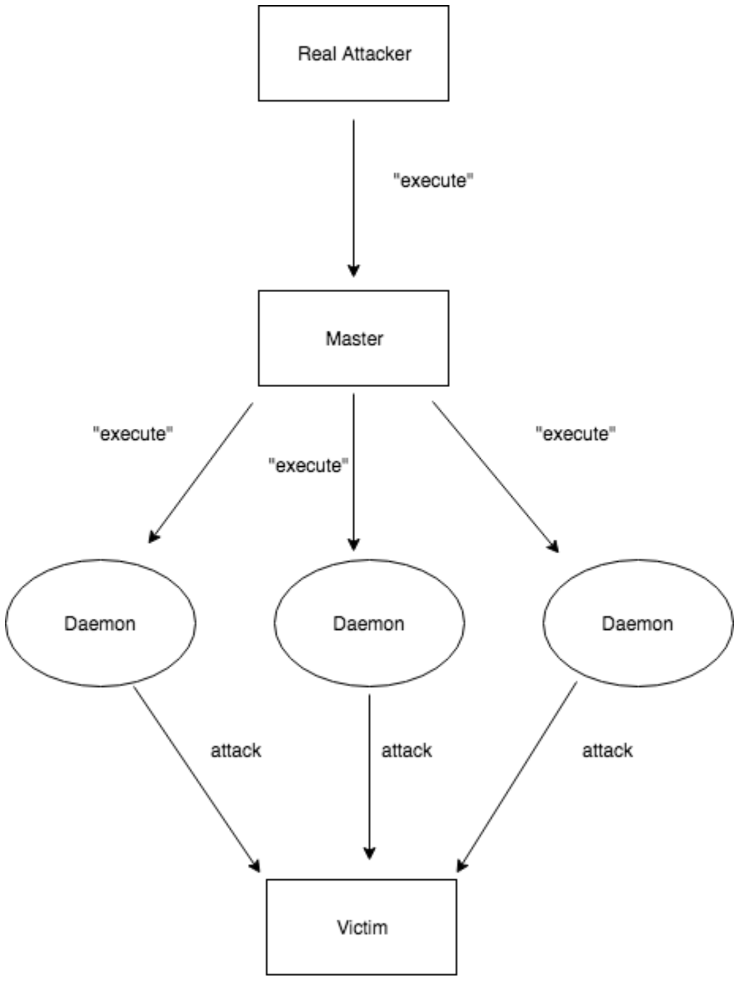


Figure 2 General structure of DDoS attack [1][2]

4. Types of Attacks

There are various types of attack which target different services, such as

- Flooding system traffic which leads to service denial to legitimate users
- Connection disruption between two machines, thereby preventing access to a service
- Preventing a particular system or user from accessing a service

5. Attack Methods

In this section we will go over some the different methods of DoS and DDoS attacks.

5.1 DoS Attack Methods

Different DoS attack methods make use of different protocols to exploit a system's weakness, some of which are:[2][3]

5.1.1 Smurf

Smurf -> ICMP: In smurf attack a large amount of Internet Control Message Protocol (ICMP) echo traffic is sent to a number of Internet Protocol (IP) broadcast addresses. The ICMP echo packets are carry a source address of the target(spoofed address) [7]. Most systems on an IP network accept ICMP echo requests [8] and reply to them with an echo reply to the source address of the victim. This multiplies the traffic by the number of responding hosts. " On a broadcast network, there could potentially be hundreds of machines to reply to each ICMP packet. The process of using a network to elicit many responses to a single packet has been labeled as an amplifier. [2]"

5.1.2 SYN Flood

SYN Flood -> TCP handshake: SYN flood exploits the three way TCP handshake, which is required to establish a connection before a service is made available to a client. In a typical handshake a server receives an initial SYN request and replies with a SYN/ACK and waits for an ACK. In a SYN Flood the server is flooded with

SYN requests which are never acknowledged, this leads to the connection buffer of the server getting filled up leaving no room to process any more incoming connections. [2]

5.1.3 UDP Flood attack

- The attacker uses forged UDP packets to connect attacker and the victim.
- Implemented exchange rate is designed to deplete the Bandwidth(BW) provided by the victim

5.2 DDoS Attack Methods

When a DDoS attack is implemented it makes use of the aforementioned DoS techniques to expand the attack and to facilitate communication between control master program and the attacker. Some of the DDoS attack techniques are discussed in this section.

5.2.1 TFN

ICMP -> (any DoS): in TFN command line interface is used to communicate between the attacker and the control master program. Interaction among the control master and attack daemons is achieved through ICMP echo reply packets. TFN's attack daemons can implement Smurf, SYN Flood, UDP Flood, and ICMP Flood attacks. [2]

5.2.2 Stacheldraht

Stacheldraht is based on TFN except in place of ICMP it uses encrypted TCP connection in the first stage. Control and communication between control master and its daemons is done through TCP and ICMP.

5.2.3 Trinoo

TCP -> UDP Flood: Trinoo uses TCP for connection between the main attacker and the control master program and UDP for communication between the master and attack daemons. It is used in UDP flood attacks.

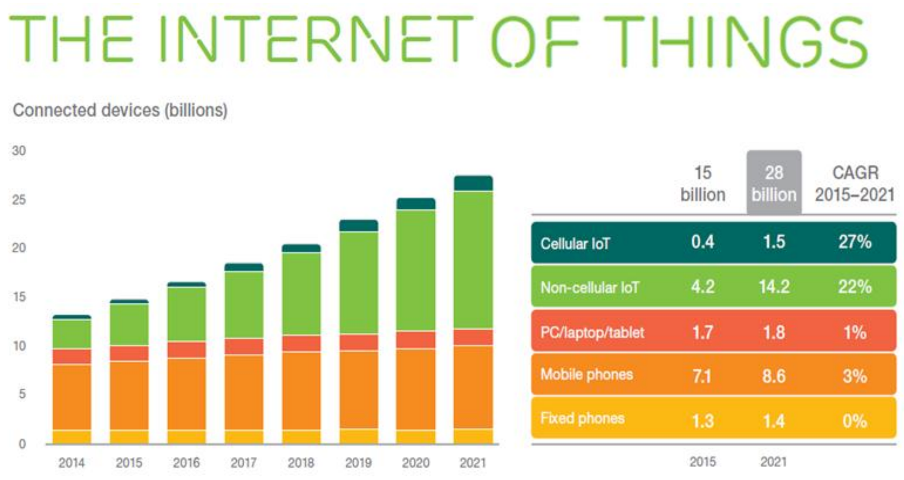
There are other complex variations that incorporate various protocols at various stages of an attack to make detection of the attack and tracing the attacker difficult for the victim.

6. Example of a DDoS Attack: BBC website attack

Back on the new year's eve of 2015, a group of hackers have launched a huge DDoS attack on BBC's website. The entire domain of BBC including television and radio player were down for more than three hours. The tool that was used to deploy the attackers was capable of generating up to 600 Gbps on BBC's website. This attack was believed to be one of the biggest DDoS attacks on history. [11][12]

7. Importance of analyzing DDoS attacks

The Internet of Things (IoT) sensors and devices are constantly increasing and the world is expected to have around 28 billion connected devices by 2021 [15]. This enormous increase in the connected devices and sensors will allow attackers to compromise more connected devices to generate huge volume based attacks in an increasing rates.



	15 billion	28 billion	CAGR 2015–2021
Cellular IoT	0.4	1.5	27%
Non-cellular IoT	4.2	14.2	22%
PC/laptop/tablet	1.7	1.8	1%
Mobile phones	7.1	8.6	3%
Fixed phones	1.3	1.4	0%
	2015	2021	

Figure 3 Expected number of connected devices between 2014 - 2021 [15]

Studying DDoS attacks and their different prevention and attacking methods will help us protecting intellectual properties and corporates as the world of technology develops. DDoS attacks could lead to incredible loses such as[16]:

- Hardware replacements
- Customers trust
- Personal information
- Reputation lose
- Reduce productivity
- Downtime costs

8. Prevention Methods

Some ways to prevent a DDoS attack are:

- Filtering Routers: Filtering all packets passing through the network, protects from attacks conducted from neighboring networks, and prevents the network itself from being an unaware attacker [3]
- Disabling IP Broadcasts: By disabling IP broadcasts, host computers can no longer be used as amplifiers in ICMP Flood and Smurf attacks
- Other common ways: [2]
    - Increase the size of the connection queue,
    - Decrease the time-out waiting for the three-way handshake, and
    - Employ vendor software patches to detect and circumvent the

problem.

- Modifying queuing algorithm in routers

9. Queuing Algorithms

Since the focus of this project is the use of queuing algorithms to analyse performance we introduce here the algorithms that we used in the simulations.

9.1 DropTail

Each packet is treated identically and when queue filled to its maximum capacity the newly incoming packets are dropped until queue have sufficient space to accept incoming traffic, finite FIFO. Figure 4 below illustrates the mechanism of DropTail algorithm. [2]

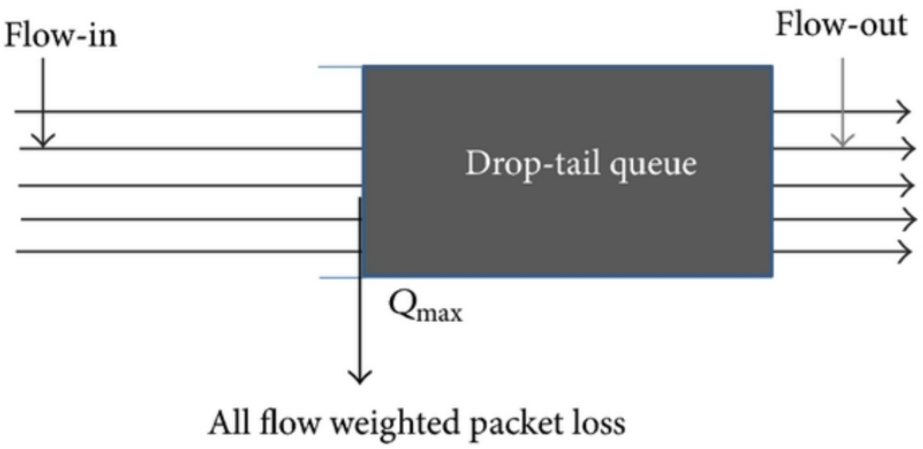


Figure 4 DropTail Queue algorithm [13]

9.2 RED

It operates on the average queue size and drop packets on the basis of statistics information. If the buffer is empty all incoming packets are acknowledged. As the queue size increase the probability for discarding a packet also increase. When buffer is full probability becomes equal to 1 and all incoming packets are dropped. Figure 5 below illustrates the mechanism of RED algorithm. [5]

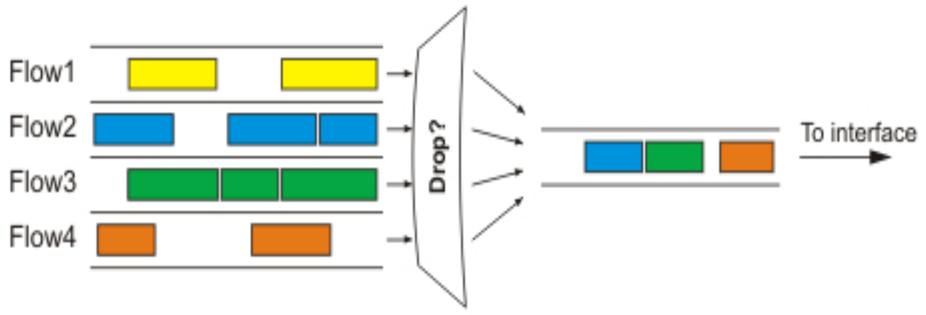


Figure 5 RED Queue algorithm [14]

9.3 SFQ

SFQ queuing algorithm provides fairness between the connected clients. Each client is able to send data in turn, thus preventing any single user from drowning out the rest. The fairness of this queuing algorithm is ensured by hashing algorithm as well as round robin algorithm. Figure 6 below illustrates the mechanism of SFQ algorithm.[5][14]

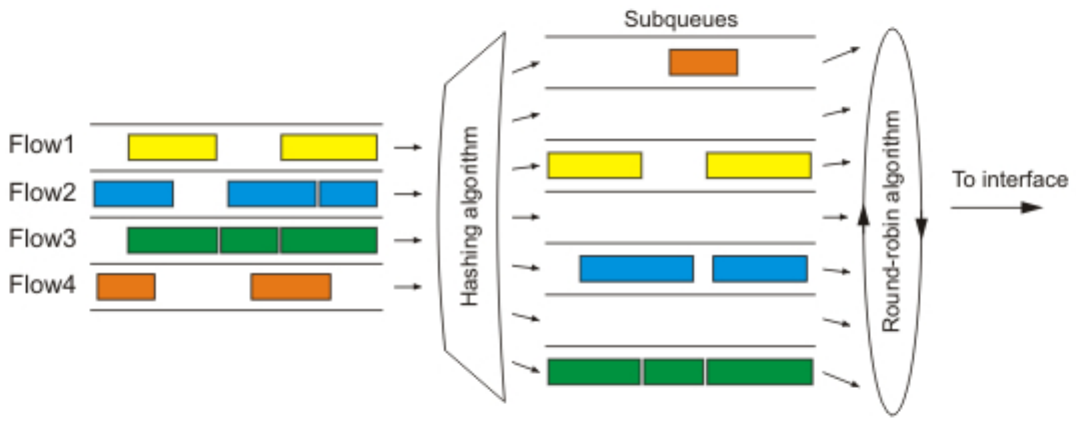


Figure 6 SFQ Queue algorithm [14]

10. Related Work

Two other projects that are comparable to ours are mentioned here. The first paper[2] is published by professor Ljiljana Trajkovic and her peers in SFU, UoC and SPAWAR systems Center in San Diego, CA. This paper analyzes the attacks using different queueing algorithms. In this project the server or the victim is connected directly to the clients and the attacker. We take a different approach where the clients and attackers are connected to routers which are in turn connected to a Gateway which is then connected to the victim. This typifies more of how an actual connection is set up. The queuing algorithms are then applied to all the links.

The second project was done by previous ENSC 427 students, from Spring 2015[9], where they analyzed the effect of using a black hole on a topology similar to the one used in this project. The testing in this report focused more on a black hole solution then the effect of queuing algorithm.

11. Simulation

Our goal for this simulation is to simulate DDoS Scenario using the following:

- **Software Tools**
  - **ns-2** (network simulator)
  - x-graph
  - Excel
- **Attack Method**
  - **UDP Flood**
    - ~**Trinoo** DDoS Implementation
- **Queuing algorithms:**
    - DropTail
    - SFQ
    - RED

An approximation of Trinoo DDoS implementation is used because we are only interested in effect of the attack, namely interaction of daemons and the victim, not the attacker and the control master program.

11.1 Simulation Topology

The network topology consists of a server connected to a Gateway which is further connected to three routers. Two of these routers have three legitimate clients each and the third router is connected to 3 Zombies which are controlled by and connected to an attacker. As seen in the following figure, figure-7 topology in nam.

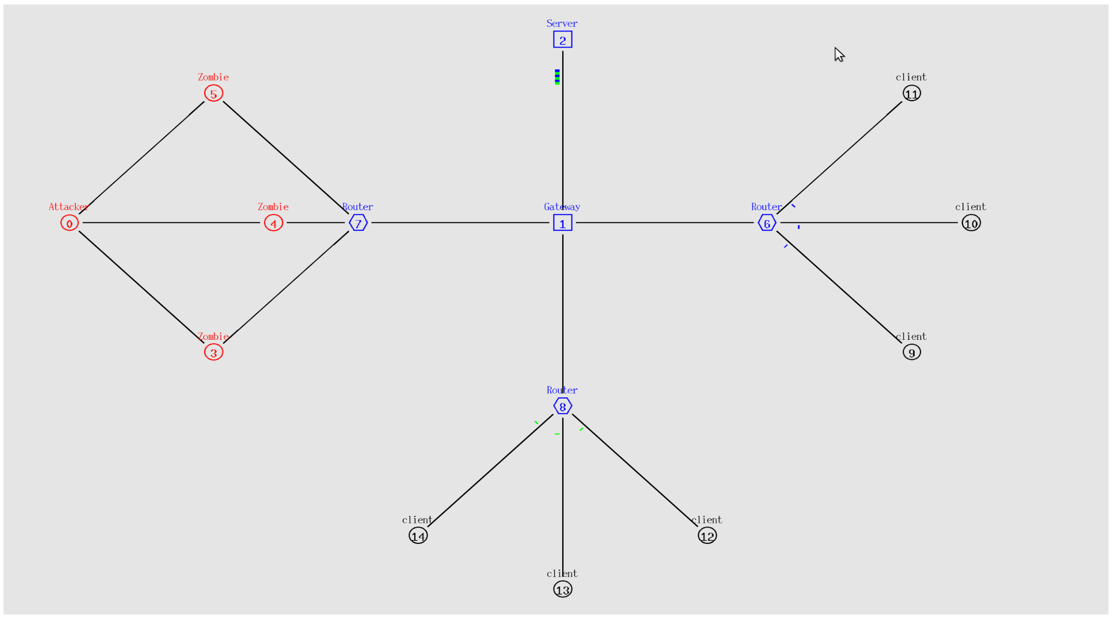


Figure 7 Simulation topology before attack

11.2 Simulation Parameters

The attack is done on a UDP to UDP connection. This is because we do not care about the packets reaching their destination we are interested in filling up the bandwidth. All the nodes are using UDP protocol. Traffic is generated by the six clients and three attackers based on a CBR traffic model. There are two types of sinks setup one is the main server which acts as a sink for all nine traffic generating nodes. The second one is for the zombies which acts as a sink for the attacker, when the attacker wants to initialize the attack.

The link and traffic implementation is as follows,

- **Attacking**
    - 1 Attacker
    - 3 Zombies (Daemons)
    - Rate: 2.4Mbps / Zombie
    - Interval: 20ms
- **Clients**
    - 6 legitimate clients
    - Rate: 0.04 Mbps / client
    - Interval: 200ms
- **Routers**
    - 3 routers in total
    - 2 routers connect legitimate clients to gateway
    - Total rate sent by clients: 0.24 Mbps /router
    - 1 router connects zombies to Gateway
    - Total rate sent by zombies: 7.2 Mbps

- **Links:** 100ms delay and a bandwidth of 10Mbps for all links except gateway to server the bandwidth is 5 Mbps.

Traces are collecting information on these sinks and plotting using X graph and Excel.

11.3 Simulation Run Time

- t < 0.4s :
  No data was sent

- 0.4s < t < 4.8 :
  Clients sent packets at a rate of 0.04 Mbps

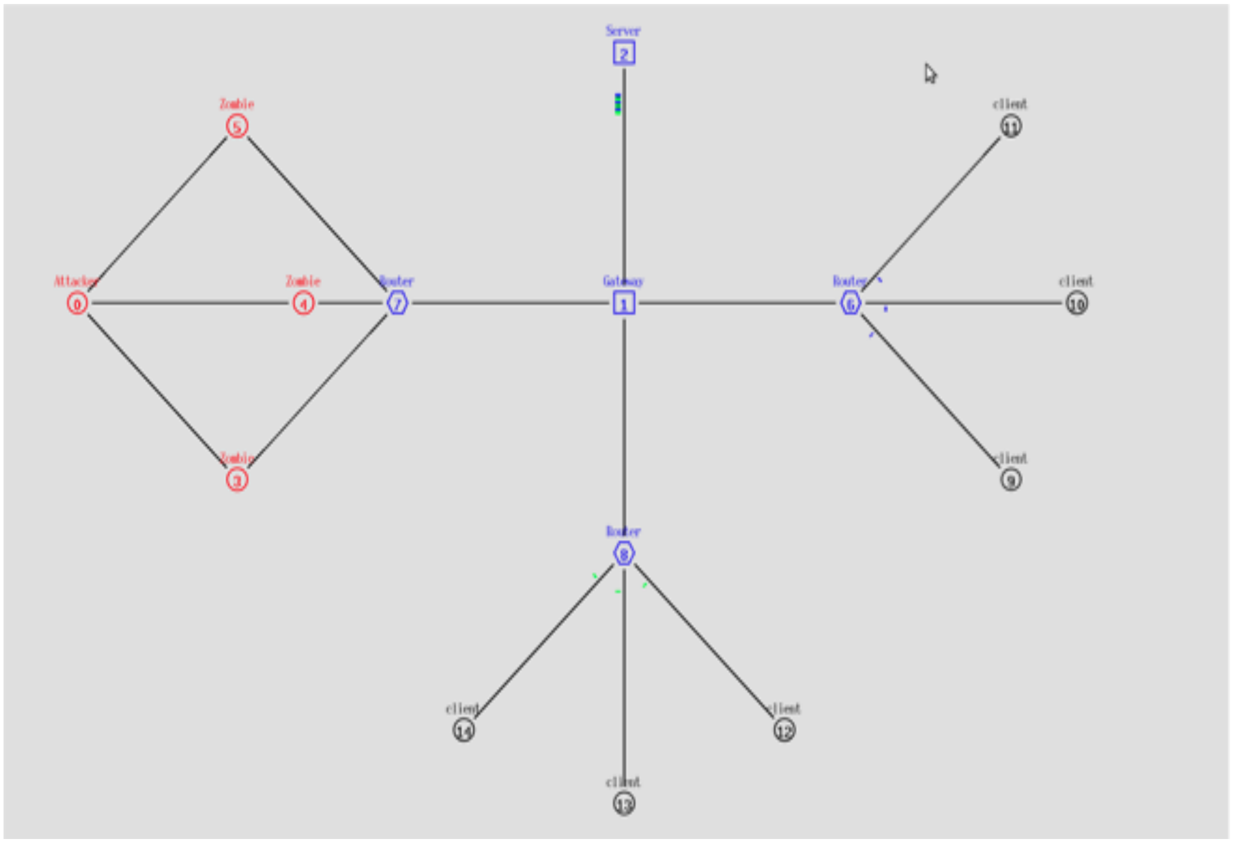


Figure 8 Simulation topology before attack

- at t = 4.8s
  Attack is initiated and Zombies started attacking at a rate of 2.4
Mbps/Zombie

- 4.8s < t < 9.4s
  the gateway was flooded and client's packets were lost

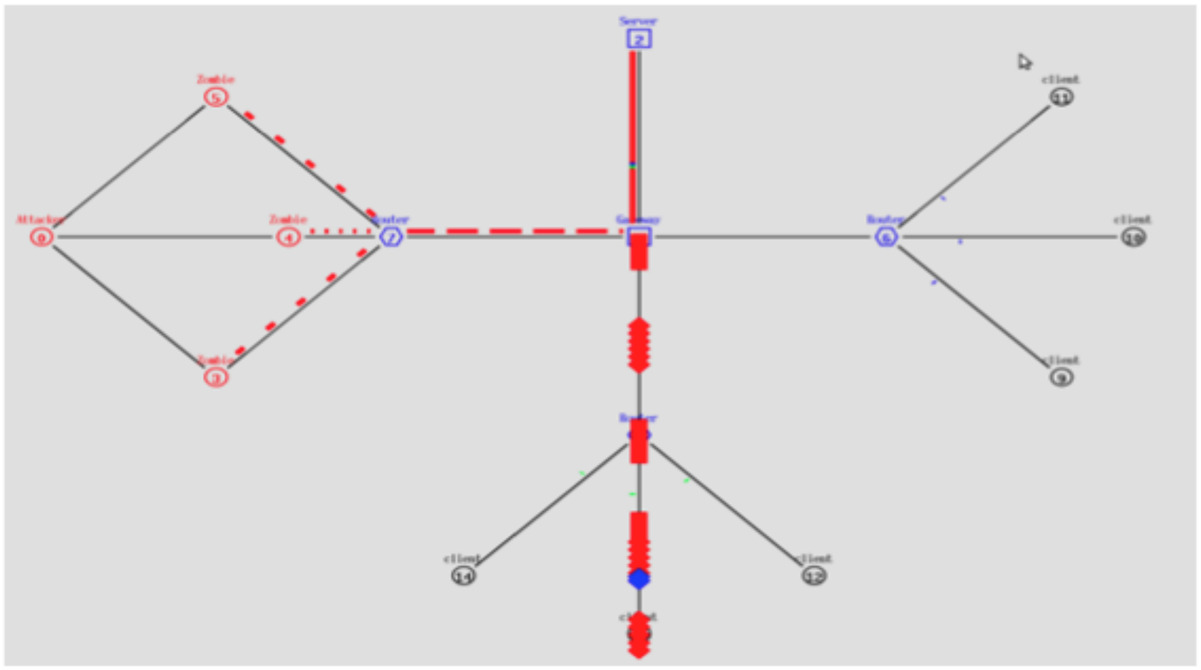


Figure 9 Simulation topology after attack

- at t = 9.4s :
  Zombies stopped the attack

- 9.4s < t < 14.6s
  System was back to behave normally

- at 14.8s
  Simulation was ended

11.4 Analysis

In this section we will analysis the behaviour of the system during the DDoS attack while using DropTail, RED and SFQ queuing algorithms. At the end of this section we will present a comparison between all the three different outputs we have.

11.4.1 System performance using DropTail queuing algorithm

As can be seen from figure 10 below, before 4.8 second every client was able to have its desired bandwidth which is 0.04 Mbps. However, during the attack, between 4.8s < t < 9.4s, most of the clients weren't able to send packets to the server.

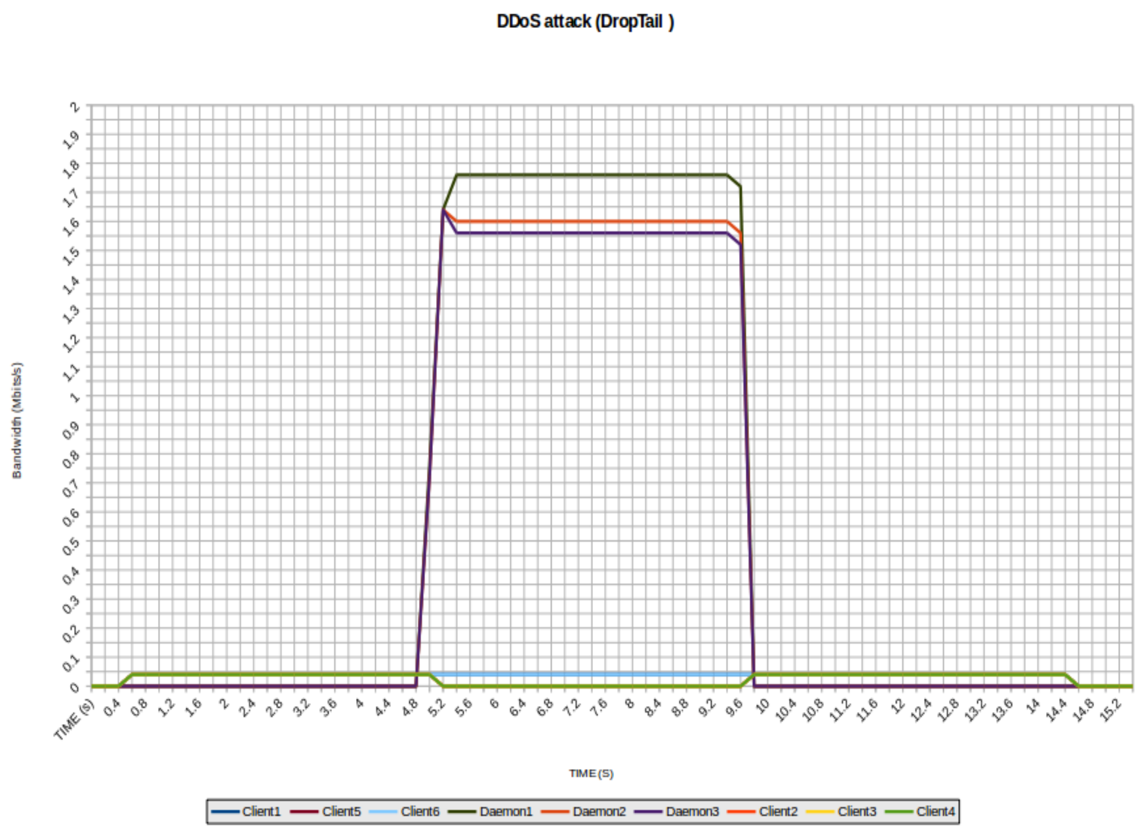


Figure 10 Simulation results when DropTail queuing is used

- Client 2 & 6 were able to send their packets even under the attack, 4.8s < t < 9.4s, and we believe that this is happening because DropTail queuing is FIFO based queuing. Client 2 &Client 6 were the first ones to capture the queue and start their data transmission. It is possible that they continue to transmit data because of the first in first out algorithm. However, to have a fare comparison between the three queueing algorithms, we selected client 4 as a sample to compare between the queues algorithms later in section 10.4.4 .

11.4.2 System performance using RED queuing algorithm

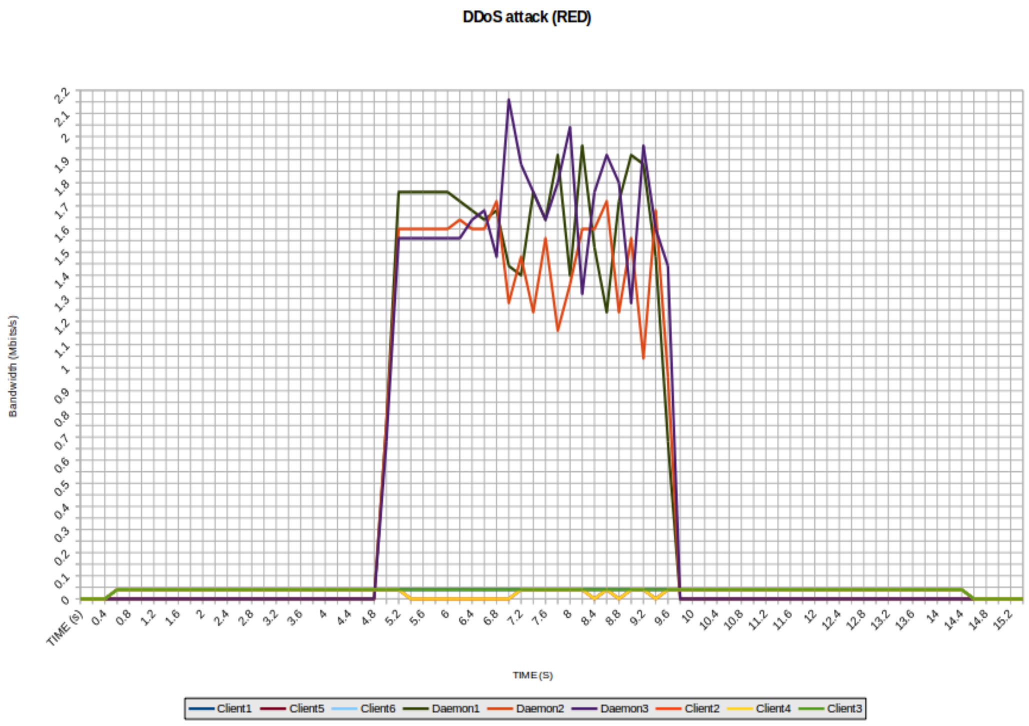


Figure11 Simulation results when RED queuing is used

As can be seen from figure 11 above, there is a noticeable improvement in the performance of the system when we used RED queuing algorithm. Multiple clients were able to send some packets during the attack, between 4.8s < t < 9.4s. However, that is not the optimum queuing algorithm in this case since we can still see a drop in the client's packets during the attack.

11.4.3 System performance using SFQ queuing algorithm

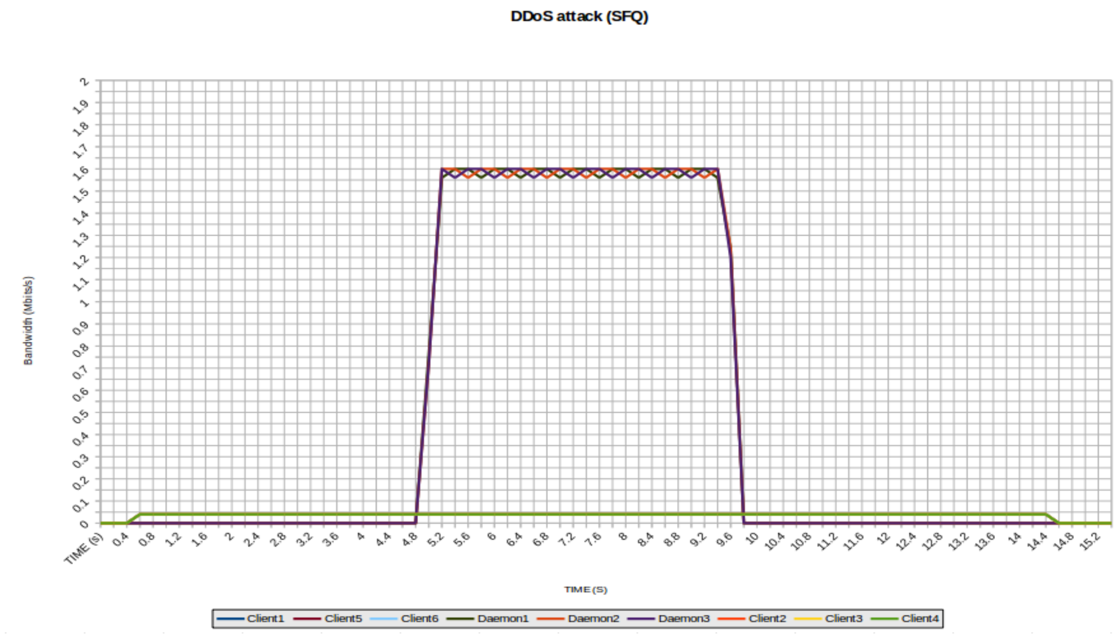


Figure 12 Simulation results when SFQ queuing is used

As can be seen from figure 12 above, SFQ queuing algorithm has allowed all clients to send their desired packets during the attack , between 4.8s < t < 9.4s, without dropping any packets.

11.4.4 DropTail vs RED vs SFQ

In this section we will compare client 4 over the three different queuing algorithms. Figure 13, below shows the bandwidth of client 4 before, during and after the attack.

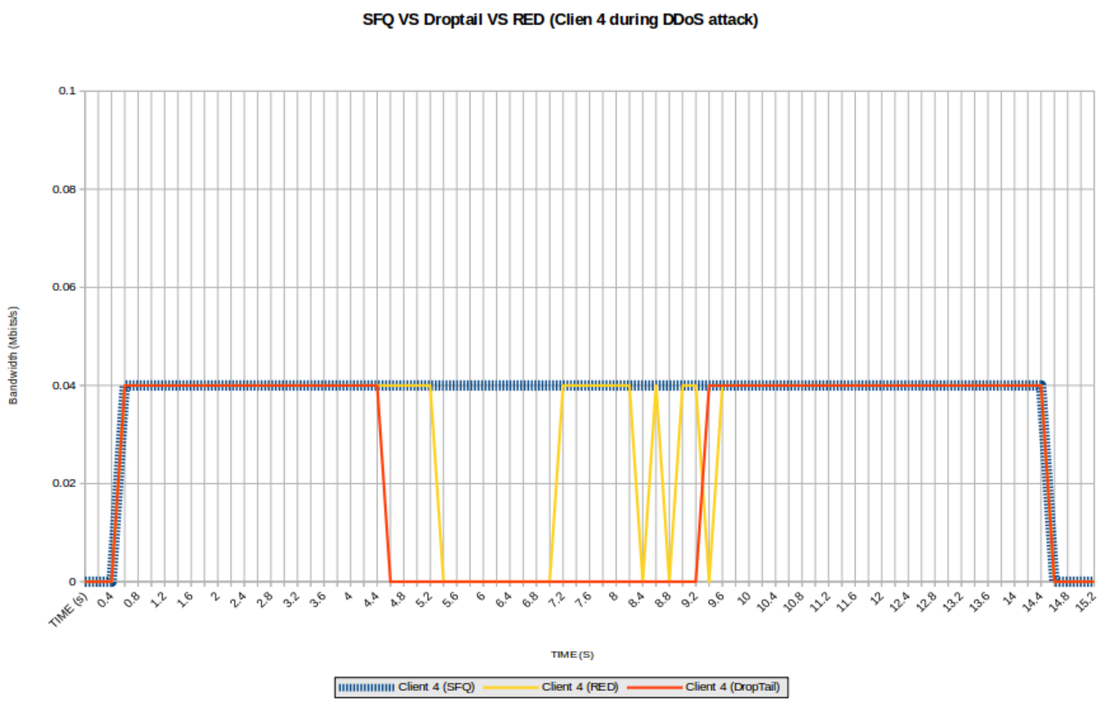


Figure 13 Client 4 simulation results using the three different queues

- The red line shows the bandwidth utilized by client4 using DropTail queuing. During the attack client 4 wasn't able to transfer any packets.
- The yellow line shows the bandwidth utilized by client4 using RED queuing. During the attack client 4 was able to transfer some packets and lost some.
- The blue line shows the bandwidth utilized by client4 using SFQ queuing. During the attack client 4 was able to transfer all packets without any disruption from the attack
- Based on our topology and simulation parameters this is the optimum queuing algorithm among the other algorithms we used so far.

Table 1 below shows the bandwidth of client 4 during the attack using different queues.

Table 1 Client 4 BW using different queues

TIME (Seconds)	DropTail (Mbps)	RED (Mbps)	SFQ (Mbps)
4.8	0.04	0.04	0.04
5.0	0.04	0.04	0.04
5.2	0.04	0.04	0.04
5.3	0.0	0.04	0.04
5.4	0.0	0.0	0.04
5.6	0.0	0.0	0.04
5.8	0.0	0.0	0.04
6.0	0.0	0.0	0.04
6.2	0.0	0.0	0.04
6.4	0.0	0.0	0.04
6.6	0.0	0.0	0.04
6.8	0.0	0.0	0.04
7.0	0.0	0.0	0.04

7.2	0.0	0.04	0.04
7.4	0.0	0.04	0.04
7.6	0.0	0.04	0.04
7.8	0.0	0.04	0.04
8.0	0.0	0.04	0.04
8.2	0.0	0.04	0.04
8.4	0.0	0.04	0.04
8.6	0.0	0.0	0.04
8.8	0.0	0.04	0.04
9.0	0.0	0.0	0.04
9.2	0.0	0.04	0.04
9.4	0.0	0.04	0.04
9.6	0.0	0.04	0.04
9.8	0.0	0.04	0.04

12. Scope of Future Work

In future work we suggest use of a bigger topology that is more consistent with an actual DDoS attack and analysis of constant performance achieved by the first clients of each router. We will also implement different types of DDoS attacks as well as different preventions techniques and determine which once are more useful to implement in a giving application.

13. Conclusion

In this project we have learned about and analyzed DDoS. We have reviewed a recent real life DDoS attack case that targeted BBC website. We have also discussed why DDoS is an important topic to be addressed, and how IoT could play a role in increasing DDoS attacks risk. We also went over various DDoS attacks methods and some prevention techniques. After that, we used ns-2 simulator to simulate a DDoS attack using three different queuing algorithms. We found that DropTail algorithm didn't allow clients to send their packets to the server during the attack time. This is because droptail queue filled up very quickly, and it drop all incoming packets. We have also tried using RED queuing and we noticed an improvement in the system performance where some clients were able to send some packets during the attack time. After that we tried the SFQ algorithm and it allowed all clients to send their packets to the server without any packet losses from clients. This is because SFQ ensures fairness in sending packets using hashing algorithms as well as Round Robin algorithm. Based on our topology and simulation parameters, we found that SFQ is the optimum amongst the other queue types we have tried in this project.

REFERENCES

- [1]S. Bellovin, “Distributed denial of service attacks,” Feb. 2000,http://www.research.att.com/~smb/talks
- [2]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
- [3]D. Dittrich, “The DoS project's ‘Trinoo’ distributed denial of service attack tool,” Oct. 1999; “The ‘Stacheldraht’ distributed denial of service attack tool,” Dec. 1999; “The ‘Tribe Flood Network’ distributed denial of service attack tool,” Oct. 1999, http://www.washington.edu/People/dad.
- [4] P. Ferguson and D. Senie, “RFC 2267: Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing,” Jan. 1998, http://info.internet.isi.edu/innotes/rfc/files/rfc2267.txt
- [5] Kuznetsov, Alexey. "Tc-Sfq(8) - Linux Man Page". Linux man page. N.p., 2017. Web. 3 Apr. 2017.
- [6] "Working Mechanism Of FQ, RED, SFQ, DRR And Drop-Tail Queues - Network Technologies (TCP/IP Suite)". Sites.google.com. N.p., 2017. Web. 3 Apr. 2017.
- [7] Daemon9, Infinity, and Route, “IP-spoofing demystified: trust-relationship exploitation,” Phrack Mag., June 1996,http://www.fc.net/phrack/files/p48/p48-14.html.
- [8] S. Bellovin, Ed., “The ICMP traceback message,” Network Working Group Internet Draft, Mar. 2000, http://www.research.att.com/~smb/papers/draft-bellovin-itrace-00.txt.
- [9] S. Chow, T. Sherpa and S. Hoque, “Performance analysis during a DDoS attack”, http://www.ensc.sfu.ca/~ljilja/ENSC427/Spring15/Projects/team8/ENSC427_team8_report.pdf, April 2015.
- [10] D. Bisson, "DDoSCoin - An Incentive to Launch DDoS Attacks?", BleepingComputer, 2017. [Online]. Available: https://www.bleepingcomputer.com/news/security/ddoscoin-an-incentive-to-launch-ddos-attacks/. [Accessed: 15- Apr- 2017].

- [11] Z. Whittaker, "'Biggest ever' web attack on BBC actually wasn't even close | ZDNet", ZDNet, 2017. [Online]. Available: http://www.zdnet.com/article/tango-down-bbc-was-this-the-largest-ddos-web-attack/. [Accessed: 15- Apr- 2017].
- [12] "3 Famous DDoS Attacks", Cyberdefense Hub, 2017. [Online]. Available: http://www.cyberdefensehub.com/famous-ddos-attacks/. [Accessed: 15- Apr- 2017].
- [13] 2017. [Online]. Available: https://www.researchgate.net/figure/275594356_fig7_Operation-of-a-drop-tail-queue-under-the-fluid-model-simulation. [Accessed: 15- Apr- 2017].
- [14] "3 Famous DDoS Attacks", Cyberdefense Hub, 2017. [Online]. Available: http://www.cyberdefensehub.com/famous-ddos-attacks/. [Accessed: 15- Apr- 2017].
- [15] "Forbes Welcome", Forbes.com, 2017. [Online]. Available: https://www.forbes.com/sites/louiscolumbus/2016/07/09/internet-of-things-on-pace-to-replace-mobile-phones-as-most-connected-device-in-2018/#32c42b3d732c. [Accessed: 16- Apr- 2017].
- [16] "DDoS Attacks: Bigger, Stronger, Scarier", Symantec.com, 2017. [Online]. Available: https://www.symantec.com/connect/blogs/ddos-attacks-bigger-stronger-scarier. [Accessed: 16- Apr- 2017].

APPENDIX

CODE LISTING

DROPTAIL

```
####################################   DropTail
##############################


#Create a simulator object

set ns [new Simulator]


#Tell the simulator to use dynamic routing

#$ns rtproto DV


#Open the nam trace file

set nf [open out.nam w]

$ns namtrace-all $nf


set f0 [open out0.tr w]

set f1 [open out1.tr w]

set f2 [open out2.tr w]

set f3 [open out3.tr w]

set f4 [open out4.tr w]

set f5 [open out5.tr w]
set f6 [open out6.tr w]

set f7 [open out7.tr w]

set f8 [open out8.tr w]

set during [open outfduring.tr w]

set before [open outfbefore.tr w]

set fafter [open outfafter.tr w]



#Define a 'finish' procedure

proc finish {} {

        global ns nf f0 f1 f2 f3 f4 f5 f6 f7 f8 during before fafter

        $ns flush-trace

   #Close the trace file

        close $nf


        #Close the output files

        close $f0

        close $f1

        close $f2

   close $f3

        close $f4

        close $f5

   close $f6

        close $f7
```

```
          close $f8
     close $during
     close $before
     close $fafter
     #Execute nam on the trace file
     exec nam out.nam &
          #Call xgraph to display the results
          exec xgraph out0.tr out1.tr out2.tr out3.tr out4.tr out5.tr out6.tr out7.tr
out8.tr -geometry 800x400 &

          exit 0
}

#Create nine nodes
set Attacker [$ns node]
set Server   [$ns node]
set Server2   [$ns node]
set Zombie1  [$ns node]
set Zombie2  [$ns node]
set Zombie3  [$ns node]
set Router   [$ns node]
set Router2   [$ns node]
set Router3   [$ns node]
set Client1  [$ns node]
set Client2  [$ns node]

set Client3  [$ns node]

set Client4  [$ns node]

set Client5  [$ns node]

set Client6  [$ns node]


#Colors

$Attacker color  red

$Zombie1  color  red

$Zombie2  color  red

$Zombie3  color  red

$Server   color  blue

$Server2  color  blue

$Router   color  blue

$Router2  color  blue

$Router3  color  blue


# Labelling

$ns at 0.0 "$Attacker label Attacker"

$ns at 0.0 "$Zombie1  label Zombie"

$ns at 0.0 "$Zombie2  label Zombie"

$ns at 0.0 "$Zombie3  label Zombie"

$ns at 0.0 "$Server   label Gateway"

$ns at 0.0 "$Server2  label Server"
```

```
$ns at 0.0 "$Router   label Router"

$ns at 0.0 "$Router2  label Router"

$ns at 0.0 "$Router3  label Router"

$ns at 0.0 "$Client1  label client"

$ns at 0.0 "$Client2  label client"

$ns at 0.0 "$Client3  label client"

$ns at 0.0 "$Client4  label client"

$ns at 0.0 "$Client5  label client"

$ns at 0.0 "$Client6  label client"


# Shape

$Server  shape square

$Server2 shape square

$Router  shape hexagon

$Router2 shape hexagon

$Router3 shape hexagon



#Create links between the nodes

$ns duplex-link $Attacker  $Zombie1 10Mb 100ms DropTail

$ns duplex-link $Attacker  $Zombie2 10Mb 100ms DropTail

$ns duplex-link $Attacker  $Zombie3 10Mb 100ms DropTail

$ns duplex-link $Zombie1   $Router2 10Mb 100ms DropTail

$ns duplex-link $Zombie2   $Router2 10Mb 100ms DropTail
```

```
$ns duplex-link $Zombie3   $Router2 10Mb 100ms DropTail
$ns duplex-link $Router2   $Server  10Mb 100ms DropTail
$ns duplex-link $Router3   $Server  10Mb 100ms DropTail
$ns duplex-link $Server       $Router  10Mb 100ms DropTail
$ns duplex-link $Server       $Server2 5Mb  100ms DropTail
$ns duplex-link $Router      $Client1 10Mb 100ms DropTail
$ns duplex-link $Router      $Client2 10Mb 100ms DropTail
$ns duplex-link $Router      $Client3 10Mb 100ms DropTail
$ns duplex-link $Router3   $Client4 10Mb 100ms DropTail
$ns duplex-link $Router3   $Client5 10Mb 100ms DropTail
$ns duplex-link $Router3   $Client6 10Mb 100ms DropTail

#queue limit
#$ns queue-limit $Server     $Server2  50

# Setting node position
$ns duplex-link-op $Attacker $Zombie1 orient right-down
$ns duplex-link-op $Attacker $Zombie2 orient right
$ns duplex-link-op $Attacker $Zombie3 orient right-up
$ns duplex-link-op $Zombie1  $Router2 orient right-up
$ns duplex-link-op $Zombie2  $Router2 orient right
$ns duplex-link-op $Zombie3  $Router2 orient right-down
$ns duplex-link-op $Router2  $Server  orient right
$ns duplex-link-op $Server   $Router3 orient down
```

```
$ns duplex-link-op $Server   $Router  orient right

$ns duplex-link-op $Server   $Server2 orient up

$ns duplex-link-op $Router   $Client1 orient right-down

$ns duplex-link-op $Router   $Client2 orient right

$ns duplex-link-op $Router   $Client3 orient right-up


$ns duplex-link-op $Router3   $Client4 orient right-down

$ns duplex-link-op $Router3   $Client5 orient down

$ns duplex-link-op $Router3   $Client6 orient left-down



#Create a UDP agent and attach it to nodes

set udpattacker [new Agent/UDP]

$ns attach-agent $Attacker $udpattacker

set udpz1 [new Agent/UDP]

$ns attach-agent $Zombie1 $udpz1

set udpz2 [new Agent/UDP]

$ns attach-agent $Zombie2 $udpz2

set udpz3 [new Agent/UDP]

$ns attach-agent $Zombie3 $udpz3

# set udpRouter [new Agent/UDP]

# $ns attach-agent $Router $udpRouter

#=======================

# set udpRouter2 [new Agent/UDP]
# $ns attach-agent $Router2 $udpRouter2

# set udpRouter3 [new Agent/UDP]

# $ns attach-agent $Router3 $udpRouter3

#========================

# set udpServer [new Agent/UDP]

# $ns attach-agent $Server $udpServer

set udpc1 [new Agent/UDP]

$ns attach-agent $Client1 $udpc1

set udpc2 [new Agent/UDP]

$ns attach-agent $Client2 $udpc2

set udpc3 [new Agent/UDP]

$ns attach-agent $Client3 $udpc3

set udpc4 [new Agent/UDP]

$ns attach-agent $Client4 $udpc4

set udpc5 [new Agent/UDP]

$ns attach-agent $Client5 $udpc5

set udpc6 [new Agent/UDP]

$ns attach-agent $Client6 $udpc6


# Create a CBR traffic source and attach it to udp agents

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize_ 1000

$cbr0 set rate_ 0.2Mb

$cbr0 set interval_ 0.2
```

```
$cbr0 attach-agent $udpattacker


set cbr1 [new Application/Traffic/CBR]

$cbr1 set packetSize_ 6000

$cbr1 set rate_ 2.4Mb;#ignored if interval is specified else interval =
(packetsize*8)/rate

$cbr1 set interval_ 0.02

$cbr1 attach-agent $udpz1

#interval = .01s/packet = 100packets/s, 3000 * 100 = 300000=.3MB/s * 8 = 2.4
Mb/sec

# 2.4 *3 zombies = 7.2 Mb/s

set cbr2 [new Application/Traffic/CBR]

$cbr2 set packetSize_ 6000

$cbr2 set rate_ 2.4Mb

$cbr2 set interval_ 0.02

$cbr2 attach-agent $udpz2


set cbr3 [new Application/Traffic/CBR]

$cbr3 set packetSize_ 6000

$cbr3 set rate_ 2.4Mb

$cbr3 set interval_ 0.02

$cbr3 attach-agent $udpz3


# set cbr4 [new Application/Traffic/CBR]
# $cbr4 set packetSize_ 1000

# $cbr4 set rate_ 0.2Mb

# $cbr4 set interval_ 0.2

# $cbr4 attach-agent $udpRouter


# set cbr5 [new Application/Traffic/CBR]

# $cbr5 set packetSize_ 1000

# $cbr5 set rate_ 0.2Mb

# $cbr5 set interval_ 0.2

# $cbr5 attach-agent $udpServer


set cbr6 [new Application/Traffic/CBR]

$cbr6 set packetSize_ 1000

$cbr6 set rate_ 0.2Mb

$cbr6 set interval_ 0.2

$cbr6 attach-agent $udpc1

# 8000 bits/packet , 0.2 s/packet , 8000/0.2 = 40k bits/s , 0.04 Mbits/s

set cbr7 [new Application/Traffic/CBR]

$cbr7 set packetSize_ 1000

$cbr7 set rate_ 0.04Mb

$cbr7 set interval_ 0.2

$cbr7 attach-agent $udpc2


set cbr8 [new Application/Traffic/CBR]
```

```
$cbr8 set packetSize_ 1000
$cbr8 set rate_ 0.2Mb
$cbr8 set interval_ 0.2
$cbr8 attach-agent $udpc3

set cbr9 [new Application/Traffic/CBR]
$cbr9 set packetSize_ 1000
$cbr9 set rate_ 0.2Mb
$cbr9 set interval_ 0.2
$cbr9 attach-agent $udpc4

set cbr10 [new Application/Traffic/CBR]
$cbr10 set packetSize_ 1000
$cbr10 set rate_ 0.2Mb
$cbr10 set interval_ 0.2
$cbr10 attach-agent $udpc5

set cbr11 [new Application/Traffic/CBR]
$cbr11 set packetSize_ 1000
$cbr11 set rate_ 0.2Mb
$cbr11 set interval_ 0.2
$cbr11 attach-agent $udpc6

#Create a Null agent (a traffic sink) and attach it
```

```
#set sink0 [new Agent/Null]

#$ns attach-agent $Server2 $sink0


#for xgraph=====================

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 sink6 [new Agent/LossMonitor]

set sink7 [new Agent/LossMonitor]

set sink8 [new Agent/LossMonitor]

# Agent/LossMonitor objects as traffic sinks, since they store the amount of bytes
received, which can be used to calculate the bandwidth.

$ns attach-agent $Server2 $sink0

$ns attach-agent $Server2 $sink1

$ns attach-agent $Server2 $sink2

$ns attach-agent $Server2 $sink3

$ns attach-agent $Server2 $sink4

$ns attach-agent $Server2 $sink5

$ns attach-agent $Server2 $sink6

$ns attach-agent $Server2 $sink7

$ns attach-agent $Server2 $sink8
```

```
# set sink1 [new Agent/Null]

# $ns attach-agent $Server2 $sink1

# set sink2 [new Agent/Null]

# $ns attach-agent $Server2 $sink2

$ns connect $udpz1 $sink0

$ns connect $udpz2 $sink1

$ns connect $udpz3 $sink2

$ns connect $udpc1 $sink3

$ns connect $udpc2 $sink4

$ns connect $udpc3 $sink5

$ns connect $udpc4 $sink6

$ns connect $udpc5 $sink7

$ns connect $udpc6 $sink8




#================================



#Color each flow id

#client 1-3 class
```

```
$udpc1  set class_ 1

$udpc2  set class_ 1

$udpc3  set class_ 1

#client 4-6 class

$udpc4  set class_ 3

$udpc5  set class_ 3

$udpc6  set class_ 3

#zombie class

$udpz1  set class_ 2

$udpz2  set class_ 2

$udpz3  set class_ 2


$ns color 1 Blue

$ns color 3 Green

$ns color 2 Red



set c_1_byte 0

set c_2_byte 0

set c_3_byte 0



proc record {} {
        global sink0 sink1 sink2 sink3 sink4 sink5 sink6 sink7 sink8 f0 f1 f2 f3 f4 f5 f6
f7 f8 during c_1_byte before fafter c_2_byte c_3_byte

   #Get an instance of the simulator

   set ns [Simulator instance]

   #Set the time after which the procedure should be called again

        set time 0.2

   #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 [$sink6 set bytes_]

        set bw7 [$sink7 set bytes_]

        set bw8 [$sink8 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]"

   # calculation

   if {$now >= 4.8} {
            if {$now <= 9.4} {
            set c_1_byte [expr {$c_1_byte + $bw8}]
            puts $during "$now $c_1_byte "
            }
   }



   if {$now <= 4.8} {
            if {$now >= 0.2} {
            set c_2_byte [expr {$c_2_byte + $bw8}]
            puts $before "$now $c_2_byte "
            }
   }



   if {$now >= 9.4} {
```

```
            if {$now <= 14.0} {
            set c_3_byte [expr {$c_3_byte + $bw8}]
            puts $fafter "$now $c_3_byte "
            }
    }



    #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
    $sink6 set bytes_ 0
           $sink7 set bytes_ 0
           $sink8 set bytes_ 0
    #Re-schedule the procedure
           $ns at [expr $now+$time] "record"
}

#Schedule events for the CBR agents


$ns at 0.0 "record"
```

```
#$ns at 0.3 "$cbr0 start"
#$ns at 1.0 "$cbr4 start"
#$ns at 1.0 "$cbr5 start"

#start client traffic c1-c6 respectively
$ns at 0.2 "$cbr6 start"
$ns at 0.2 "$cbr7 start"
$ns at 0.2 "$cbr8 start"
$ns at 0.2 "$cbr9 start"
$ns at 0.2 "$cbr10 start"
$ns at 0.2 "$cbr11 start"

#start zombie traffic z1-z3 respectively
$ns at 4.8 "$cbr1 start"
$ns at 4.8 "$cbr2 start"
$ns at 4.8 "$cbr3 start"

#stop zombies
$ns at 9.4 "$cbr3 stop"
$ns at 9.4 "$cbr2 stop"
$ns at 9.4 "$cbr1 stop"

$ns at 14.0 "$cbr11 stop"
```

```
$ns at 14.0 "$cbr10 stop"

$ns at 14.0 "$cbr9 stop"

$ns at 14.0 "$cbr8 stop"

$ns at 14.0 "$cbr7 stop"

$ns at 14.0 "$cbr6 stop"

#$ns at 4.5 "$cbr5 stop"

#$ns at 4.5 "$cbr4 stop"


#$ns at 4.5 "$cbr0 stop"



#Call the finish procedure after 5 seconds of simulation time

$ns at 15.6 "finish"


#Run the simulation

$ns run
```

RED

```
##################################   RED
###############################
```

```
#Create a simulator object

set ns [new Simulator]


#Tell the simulator to use dynamic routing

#$ns rtproto DV


#Open the nam trace file

set nf [open out.nam w]

$ns namtrace-all $nf


set f0 [open out0.tr w]

set f1 [open out1.tr w]

set f2 [open out2.tr w]

set f3 [open out3.tr w]

set f4 [open out4.tr w]

set f5 [open out5.tr w]

set f6 [open out6.tr w]

set f7 [open out7.tr w]

set f8 [open out8.tr w]

set during [open outfduring.tr w]

set before [open outfbefore.tr w]

set fafter [open outfafter.tr w]
```

```
#Define a 'finish' procedure
proc finish {} {
        global ns nf f0 f1 f2 f3 f4 f5 f6 f7 f8 during before fafter
        $ns flush-trace
   #Close the trace file
        close $nf


        #Close the output files
        close $f0
        close $f1
        close $f2
   close $f3
        close $f4
        close $f5
   close $f6
        close $f7
        close $f8
   close $during
   close $before
   close $fafter
   #Execute nam on the trace file
   exec nam out.nam &
        #Call xgraph to display the results
        exec xgraph out0.tr out1.tr out2.tr out3.tr out4.tr out5.tr out6.tr out7.tr
out8.tr -geometry 800x400 &

        exit 0
}

#Create nine nodes
set Attacker [$ns node]
set Server   [$ns node]
set Server2   [$ns node]
set Zombie1  [$ns node]
set Zombie2  [$ns node]
set Zombie3  [$ns node]
set Router   [$ns node]
set Router2   [$ns node]
set Router3   [$ns node]
set Client1  [$ns node]
set Client2  [$ns node]
set Client3  [$ns node]
set Client4  [$ns node]
set Client5  [$ns node]
set Client6  [$ns node]

#Colors
$Attacker color  red
```

```
$Zombie1  color  red
$Zombie2  color  red
$Zombie3  color  red
$Server   color  blue
$Server2  color  blue
$Router   color  blue
$Router2  color  blue
$Router3  color  blue

# Labelling
$ns at 0.0 "$Attacker label Attacker"
$ns at 0.0 "$Zombie1  label Zombie"
$ns at 0.0 "$Zombie2  label Zombie"
$ns at 0.0 "$Zombie3  label Zombie"
$ns at 0.0 "$Server   label Gateway"
$ns at 0.0 "$Server2  label Server"
$ns at 0.0 "$Router   label Router"
$ns at 0.0 "$Router2  label Router"
$ns at 0.0 "$Router3  label Router"
$ns at 0.0 "$Client1  label client"
$ns at 0.0 "$Client2  label client"
$ns at 0.0 "$Client3  label client"
$ns at 0.0 "$Client4  label client"
$ns at 0.0 "$Client5  label client"
```

```
$ns at 0.0 "$Client6  label client"


# Shape
$Server  shape square
$Server2 shape square
$Router  shape hexagon
$Router2 shape hexagon
$Router3 shape hexagon



#Create links between the nodes
$ns duplex-link $Attacker  $Zombie1 10Mb 100ms RED
$ns duplex-link $Attacker  $Zombie2 10Mb 100ms RED
$ns duplex-link $Attacker  $Zombie3 10Mb 100ms RED
$ns duplex-link $Zombie1   $Router2 10Mb 100ms RED
$ns duplex-link $Zombie2   $Router2 10Mb 100ms RED
$ns duplex-link $Zombie3   $Router2 10Mb 100ms RED
$ns duplex-link $Router2   $Server  10Mb 100ms RED
$ns duplex-link $Router3   $Server  10Mb 100ms RED
$ns duplex-link $Server       $Router  10Mb 100ms RED
$ns duplex-link $Server       $Server2 5Mb  100ms RED
$ns duplex-link $Router       $Client1 10Mb 100ms RED
$ns duplex-link $Router       $Client2 10Mb 100ms RED
$ns duplex-link $Router       $Client3 10Mb 100ms RED
```

```
$ns duplex-link $Router3   $Client4 10Mb 100ms RED

$ns duplex-link $Router3   $Client5 10Mb 100ms RED

$ns duplex-link $Router3   $Client6 10Mb 100ms RED


#queue limit

#$ns queue-limit $Server     $Server2  50


# Setting node position

$ns duplex-link-op $Attacker $Zombie1 orient right-down

$ns duplex-link-op $Attacker $Zombie2 orient right

$ns duplex-link-op $Attacker $Zombie3 orient right-up

$ns duplex-link-op $Zombie1  $Router2 orient right-up

$ns duplex-link-op $Zombie2  $Router2 orient right

$ns duplex-link-op $Zombie3  $Router2 orient right-down

$ns duplex-link-op $Router2  $Server  orient right

$ns duplex-link-op $Server   $Router3 orient down

$ns duplex-link-op $Server   $Router  orient right

$ns duplex-link-op $Server   $Server2 orient up

$ns duplex-link-op $Router   $Client1 orient right-down

$ns duplex-link-op $Router   $Client2 orient right

$ns duplex-link-op $Router   $Client3 orient right-up


$ns duplex-link-op $Router3   $Client4 orient right-down

$ns duplex-link-op $Router3   $Client5 orient down
```

```
$ns duplex-link-op $Router3   $Client6 orient left-down


#Create a UDP agent and attach it to nodes
set udpattacker [new Agent/UDP]
$ns attach-agent $Attacker $udpattacker
set udpz1 [new Agent/UDP]
$ns attach-agent $Zombie1 $udpz1
set udpz2 [new Agent/UDP]
$ns attach-agent $Zombie2 $udpz2
set udpz3 [new Agent/UDP]
$ns attach-agent $Zombie3 $udpz3
# set udpRouter [new Agent/UDP]
# $ns attach-agent $Router $udpRouter
#======================
# set udpRouter2 [new Agent/UDP]
# $ns attach-agent $Router2 $udpRouter2
# set udpRouter3 [new Agent/UDP]
# $ns attach-agent $Router3 $udpRouter3
#========================
# set udpServer [new Agent/UDP]
# $ns attach-agent $Server $udpServer
set udpc1 [new Agent/UDP]
$ns attach-agent $Client1 $udpc1
```

```
set udpc2 [new Agent/UDP]

$ns attach-agent $Client2 $udpc2

set udpc3 [new Agent/UDP]

$ns attach-agent $Client3 $udpc3

set udpc4 [new Agent/UDP]

$ns attach-agent $Client4 $udpc4

set udpc5 [new Agent/UDP]

$ns attach-agent $Client5 $udpc5

set udpc6 [new Agent/UDP]

$ns attach-agent $Client6 $udpc6


# Create a CBR traffic source and attach it to udp agents

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize_ 1000

$cbr0 set rate_ 0.2Mb

$cbr0 set interval_ 0.2

$cbr0 attach-agent $udpattacker


set cbr1 [new Application/Traffic/CBR]

$cbr1 set packetSize_ 6000

$cbr1 set rate_ 2.4Mb;#ignored if interval is specified else interval =
(packetsize*8)/rate

$cbr1 set interval_ 0.02

$cbr1 attach-agent $udpz1
```

```
#interval = .01s/packet = 100packets/s, 3000 * 100 = 300000=.3MB/s * 8 = 2.4
Mb/sec

# 2.4 *3 zombies = 7.2 Mb/s

set cbr2 [new Application/Traffic/CBR]

$cbr2 set packetSize_ 6000

$cbr2 set rate_ 2.4Mb

$cbr2 set interval_ 0.02

$cbr2 attach-agent $udpz2


set cbr3 [new Application/Traffic/CBR]

$cbr3 set packetSize_ 6000

$cbr3 set rate_ 2.4Mb

$cbr3 set interval_ 0.02

$cbr3 attach-agent $udpz3


# set cbr4 [new Application/Traffic/CBR]

# $cbr4 set packetSize_ 1000

# $cbr4 set rate_ 0.2Mb

# $cbr4 set interval_ 0.2

# $cbr4 attach-agent $udpRouter


# set cbr5 [new Application/Traffic/CBR]

# $cbr5 set packetSize_ 1000

# $cbr5 set rate_ 0.2Mb
# $cbr5 set interval_ 0.2
# $cbr5 attach-agent $udpServer

set cbr6 [new Application/Traffic/CBR]
$cbr6 set packetSize_ 1000
$cbr6 set rate_ 0.2Mb
$cbr6 set interval_ 0.2
$cbr6 attach-agent $udpc1
# 8000 bits/packet , 0.2 s/packet , 8000/0.2 = 40k bits/s , 0.04 Mbits/s
set cbr7 [new Application/Traffic/CBR]
$cbr7 set packetSize_ 1000
$cbr7 set rate_ 0.04Mb
$cbr7 set interval_ 0.2
$cbr7 attach-agent $udpc2

set cbr8 [new Application/Traffic/CBR]
$cbr8 set packetSize_ 1000
$cbr8 set rate_ 0.2Mb
$cbr8 set interval_ 0.2
$cbr8 attach-agent $udpc3

set cbr9 [new Application/Traffic/CBR]
$cbr9 set packetSize_ 1000
$cbr9 set rate_ 0.2Mb
```

```
$cbr9 set interval_ 0.2
$cbr9 attach-agent $udpc4

set cbr10 [new Application/Traffic/CBR]
$cbr10 set packetSize_ 1000
$cbr10 set rate_ 0.2Mb
$cbr10 set interval_ 0.2
$cbr10 attach-agent $udpc5

set cbr11 [new Application/Traffic/CBR]
$cbr11 set packetSize_ 1000
$cbr11 set rate_ 0.2Mb
$cbr11 set interval_ 0.2
$cbr11 attach-agent $udpc6

#Create a Null agent (a traffic sink) and attach it
#set sink0 [new Agent/Null]
#$ns attach-agent $Server2 $sink0

#for xgraph====================
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 sink6 [new Agent/LossMonitor]

set sink7 [new Agent/LossMonitor]

set sink8 [new Agent/LossMonitor]

# Agent/LossMonitor objects as traffic sinks, since they store the amount of bytes
received, which can be used to calculate the bandwidth.

$ns attach-agent $Server2 $sink0

$ns attach-agent $Server2 $sink1

$ns attach-agent $Server2 $sink2

$ns attach-agent $Server2 $sink3

$ns attach-agent $Server2 $sink4

$ns attach-agent $Server2 $sink5

$ns attach-agent $Server2 $sink6

$ns attach-agent $Server2 $sink7

$ns attach-agent $Server2 $sink8



# set sink1 [new Agent/Null]

# $ns attach-agent $Server2 $sink1

# set sink2 [new Agent/Null]

# $ns attach-agent $Server2 $sink2

$ns connect $udpz1 $sink0

$ns connect $udpz2 $sink1
```

```
$ns connect $udpz3 $sink2

$ns connect $udpc1 $sink3

$ns connect $udpc2 $sink4

$ns connect $udpc3 $sink5

$ns connect $udpc4 $sink6

$ns connect $udpc5 $sink7

$ns connect $udpc6 $sink8




#===============================



#Color each flow id

#client 1-3 class

$udpc1  set class_ 1

$udpc2  set class_ 1

$udpc3  set class_ 1

#client 4-6 class

$udpc4  set class_ 3

$udpc5  set class_ 3

$udpc6  set class_ 3

#zombie class
```

```
$udpz1  set class_ 2

$udpz2  set class_ 2

$udpz3  set class_ 2


$ns color 1 Blue

$ns color 3 Green

$ns color 2 Red



set c_1_byte 0

set c_2_byte 0

set c_3_byte 0



proc record {} {


        global sink0 sink1 sink2 sink3 sink4 sink5 sink6 sink7 sink8 f0 f1 f2 f3 f4 f5 f6
f7 f8 during c_1_byte before fafter c_2_byte c_3_byte

   #Get an instance of the simulator

   set ns [Simulator instance]

   #Set the time after which the procedure should be called again

        set time 0.2

   #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 [$sink6 set bytes_]
        set bw7 [$sink7 set bytes_]
        set bw8 [$sink8 set bytes_]
#Get the current time
        set now [$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]"

# calculation

if {$now >= 4.8} {
        if {$now <= 9.4} {

        set c_1_byte [expr {$c_1_byte + $bw8}]

        puts $during "$now $c_1_byte "

        }

}



if {$now <= 4.8} {

        if {$now >= 0.2} {

        set c_2_byte [expr {$c_2_byte + $bw8}]

        puts $before "$now $c_2_byte "

        }

}



if {$now >= 9.4} {

        if {$now <= 14.0} {

        set c_3_byte [expr {$c_3_byte + $bw8}]

        puts $fafter "$now $c_3_byte "

        }

}



#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
     $sink6 set bytes_ 0
          $sink7 set bytes_ 0
          $sink8 set bytes_ 0
     #Re-schedule the procedure
          $ns at [expr $now+$time] "record"
}

#Schedule events for the CBR agents

$ns at 0.0 "record"

#$ns at 0.3 "$cbr0 start"
#$ns at 1.0 "$cbr4 start"
#$ns at 1.0 "$cbr5 start"

#start client traffic c1-c6 respectively
$ns at 0.2 "$cbr6 start"
$ns at 0.2 "$cbr7 start"
```

```
$ns at 0.2 "$cbr8 start"
$ns at 0.2 "$cbr9 start"
$ns at 0.2 "$cbr10 start"
$ns at 0.2 "$cbr11 start"

#start zombie traffic z1-z3 respectively
$ns at 4.8 "$cbr1 start"
$ns at 4.8 "$cbr2 start"
$ns at 4.8 "$cbr3 start"

#stop zombies
$ns at 9.4 "$cbr3 stop"
$ns at 9.4 "$cbr2 stop"
$ns at 9.4 "$cbr1 stop"

$ns at 14.0 "$cbr11 stop"
$ns at 14.0 "$cbr10 stop"
$ns at 14.0 "$cbr9 stop"
$ns at 14.0 "$cbr8 stop"
$ns at 14.0 "$cbr7 stop"
$ns at 14.0 "$cbr6 stop"
#$ns at 4.5 "$cbr5 stop"
#$ns at 4.5 "$cbr4 stop"
```

```
#$ns at 4.5 "$cbro stop"



#Call the finish procedure after 5 seconds of simulation time

$ns at 15.6 "finish"


#Run the simulation

$ns run
```

SFQ

```
###################################   RED
################################


#Create a simulator object

set ns [new Simulator]


#Tell the simulator to use dynamic routing

#$ns rtproto DV


#Open the nam trace file
```

```
set nf [open out.nam w]
$ns namtrace-all $nf

set f0 [open out0.tr w]
set f1 [open out1.tr w]
set f2 [open out2.tr w]
set f3 [open out3.tr w]
set f4 [open out4.tr w]
set f5 [open out5.tr w]
set f6 [open out6.tr w]
set f7 [open out7.tr w]
set f8 [open out8.tr w]
set during [open outfduring.tr w]
set before [open outfbefore.tr w]
set fafter [open outfafter.tr w]


#Define a 'finish' procedure
proc finish {} {
        global ns nf f0 f1 f2 f3 f4 f5 f6 f7 f8 during before fafter
        $ns flush-trace
  #Close the trace file
        close $nf
```

```
          #Close the output files
          close $f0
          close $f1
          close $f2
   close $f3
          close $f4
          close $f5
   close $f6
          close $f7
          close $f8
   close $during
   close $before
   close $fafter
   #Execute nam on the trace file
   exec nam out.nam &
          #Call xgraph to display the results
          exec xgraph out0.tr out1.tr out2.tr out3.tr out4.tr out5.tr out6.tr out7.tr
out8.tr -geometry 800x400 &

          exit 0
}

#Create nine nodes
set Attacker [$ns node]
set Server   [$ns node]

set Server2   [$ns node]

set Zombie1  [$ns node]

set Zombie2  [$ns node]

set Zombie3  [$ns node]

set Router   [$ns node]

set Router2   [$ns node]

set Router3   [$ns node]

set Client1  [$ns node]

set Client2  [$ns node]

set Client3  [$ns node]

set Client4  [$ns node]

set Client5  [$ns node]

set Client6  [$ns node]


#Colors

$Attacker color  red

$Zombie1  color  red

$Zombie2  color  red

$Zombie3  color  red

$Server   color  blue

$Server2  color  blue

$Router   color  blue

$Router2  color  blue
```

```
$Router3  color  blue


# Labelling
$ns at 0.0 "$Attacker label Attacker"
$ns at 0.0 "$Zombie1  label Zombie"
$ns at 0.0 "$Zombie2  label Zombie"
$ns at 0.0 "$Zombie3  label Zombie"
$ns at 0.0 "$Server   label Gateway"
$ns at 0.0 "$Server2  label Server"
$ns at 0.0 "$Router   label Router"
$ns at 0.0 "$Router2  label Router"
$ns at 0.0 "$Router3  label Router"
$ns at 0.0 "$Client1  label client"
$ns at 0.0 "$Client2  label client"
$ns at 0.0 "$Client3  label client"
$ns at 0.0 "$Client4  label client"
$ns at 0.0 "$Client5  label client"
$ns at 0.0 "$Client6  label client"


# Shape
$Server  shape square
$Server2 shape square
$Router  shape hexagon
$Router2 shape hexagon
```

```
$Router3 shape hexagon



#Create links between the nodes
$ns duplex-link $Attacker  $Zombie1 10Mb 100ms RED
$ns duplex-link $Attacker  $Zombie2 10Mb 100ms RED
$ns duplex-link $Attacker  $Zombie3 10Mb 100ms RED
$ns duplex-link $Zombie1   $Router2 10Mb 100ms RED
$ns duplex-link $Zombie2   $Router2 10Mb 100ms RED
$ns duplex-link $Zombie3   $Router2 10Mb 100ms RED
$ns duplex-link $Router2   $Server  10Mb 100ms RED
$ns duplex-link $Router3   $Server  10Mb 100ms RED
$ns duplex-link $Server       $Router  10Mb 100ms RED
$ns duplex-link $Server       $Server2 5Mb  100ms RED
$ns duplex-link $Router      $Client1 10Mb 100ms RED
$ns duplex-link $Router      $Client2 10Mb 100ms RED
$ns duplex-link $Router      $Client3 10Mb 100ms RED
$ns duplex-link $Router3   $Client4 10Mb 100ms RED
$ns duplex-link $Router3   $Client5 10Mb 100ms RED
$ns duplex-link $Router3   $Client6 10Mb 100ms RED

#queue limit
#$ns queue-limit $Server     $Server2  50
```

```
# Setting node position

$ns duplex-link-op $Attacker $Zombie1 orient right-down

$ns duplex-link-op $Attacker $Zombie2 orient right

$ns duplex-link-op $Attacker $Zombie3 orient right-up

$ns duplex-link-op $Zombie1  $Router2 orient right-up

$ns duplex-link-op $Zombie2  $Router2 orient right

$ns duplex-link-op $Zombie3  $Router2 orient right-down

$ns duplex-link-op $Router2  $Server  orient right

$ns duplex-link-op $Server   $Router3 orient down

$ns duplex-link-op $Server   $Router  orient right

$ns duplex-link-op $Server   $Server2 orient up

$ns duplex-link-op $Router   $Client1 orient right-down

$ns duplex-link-op $Router   $Client2 orient right

$ns duplex-link-op $Router   $Client3 orient right-up


$ns duplex-link-op $Router3   $Client4 orient right-down

$ns duplex-link-op $Router3   $Client5 orient down

$ns duplex-link-op $Router3   $Client6 orient left-down



#Create a UDP agent and attach it to nodes

set udpattacker [new Agent/UDP]

$ns attach-agent $Attacker $udpattacker

set udpz1 [new Agent/UDP]
```

```
$ns attach-agent $Zombie1 $udpz1
set udpz2 [new Agent/UDP]
$ns attach-agent $Zombie2 $udpz2
set udpz3 [new Agent/UDP]
$ns attach-agent $Zombie3 $udpz3
# set udpRouter [new Agent/UDP]
# $ns attach-agent $Router $udpRouter
#=======================
# set udpRouter2 [new Agent/UDP]
# $ns attach-agent $Router2 $udpRouter2
# set udpRouter3 [new Agent/UDP]
# $ns attach-agent $Router3 $udpRouter3
#=========================
# set udpServer [new Agent/UDP]
# $ns attach-agent $Server $udpServer
set udpc1 [new Agent/UDP]
$ns attach-agent $Client1 $udpc1
set udpc2 [new Agent/UDP]
$ns attach-agent $Client2 $udpc2
set udpc3 [new Agent/UDP]
$ns attach-agent $Client3 $udpc3
set udpc4 [new Agent/UDP]
$ns attach-agent $Client4 $udpc4
set udpc5 [new Agent/UDP]
```

```
$ns attach-agent $Client5 $udpc5

set udpc6 [new Agent/UDP]

$ns attach-agent $Client6 $udpc6


# Create a CBR traffic source and attach it to udp agents

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize_ 1000

$cbr0 set rate_ 0.2Mb

$cbr0 set interval_ 0.2

$cbr0 attach-agent $udpattacker


set cbr1 [new Application/Traffic/CBR]

$cbr1 set packetSize_ 6000

$cbr1 set rate_ 2.4Mb;#ignored if interval is specified else interval =
(packetsize*8)/rate

$cbr1 set interval_ 0.02

$cbr1 attach-agent $udpz1

#interval = .01s/packet = 100packets/s, 3000 * 100 = 300000=.3MB/s * 8 = 2.4
Mb/sec

# 2.4 *3 zombies = 7.2 Mb/s

set cbr2 [new Application/Traffic/CBR]

$cbr2 set packetSize_ 6000

$cbr2 set rate_ 2.4Mb

$cbr2 set interval_ 0.02
```

```
$cbr2 attach-agent $udpz2

set cbr3 [new Application/Traffic/CBR]
$cbr3 set packetSize_ 6000
$cbr3 set rate_ 2.4Mb
$cbr3 set interval_ 0.02
$cbr3 attach-agent $udpz3

# set cbr4 [new Application/Traffic/CBR]
# $cbr4 set packetSize_ 1000
# $cbr4 set rate_ 0.2Mb
# $cbr4 set interval_ 0.2
# $cbr4 attach-agent $udpRouter

# set cbr5 [new Application/Traffic/CBR]
# $cbr5 set packetSize_ 1000
# $cbr5 set rate_ 0.2Mb
# $cbr5 set interval_ 0.2
# $cbr5 attach-agent $udpServer

set cbr6 [new Application/Traffic/CBR]
$cbr6 set packetSize_ 1000
$cbr6 set rate_ 0.2Mb
$cbr6 set interval_ 0.2
```

```
$cbr6 attach-agent $udpc1

# 8000 bits/packet , 0.2 s/packet , 8000/0.2 = 40k bits/s , 0.04 Mbits/s

set cbr7 [new Application/Traffic/CBR]

$cbr7 set packetSize_ 1000

$cbr7 set rate_ 0.04Mb

$cbr7 set interval_ 0.2

$cbr7 attach-agent $udpc2


set cbr8 [new Application/Traffic/CBR]

$cbr8 set packetSize_ 1000

$cbr8 set rate_ 0.2Mb

$cbr8 set interval_ 0.2

$cbr8 attach-agent $udpc3


set cbr9 [new Application/Traffic/CBR]

$cbr9 set packetSize_ 1000

$cbr9 set rate_ 0.2Mb

$cbr9 set interval_ 0.2

$cbr9 attach-agent $udpc4


set cbr10 [new Application/Traffic/CBR]

$cbr10 set packetSize_ 1000

$cbr10 set rate_ 0.2Mb

$cbr10 set interval_ 0.2
```

```
$cbr10 attach-agent $udpc5

set cbr11 [new Application/Traffic/CBR]
$cbr11 set packetSize_ 1000
$cbr11 set rate_ 0.2Mb
$cbr11 set interval_ 0.2
$cbr11 attach-agent $udpc6

#Create a Null agent (a traffic sink) and attach it
#set sink0 [new Agent/Null]
#$ns attach-agent $Server2 $sink0

#for xgraph=====================
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 sink6 [new Agent/LossMonitor]
set sink7 [new Agent/LossMonitor]
set sink8 [new Agent/LossMonitor]
# Agent/LossMonitor objects as traffic sinks, since they store the amount of bytes
received, which can be used to calculate the bandwidth.
```

```
$ns attach-agent $Server2 $sink0

$ns attach-agent $Server2 $sink1

$ns attach-agent $Server2 $sink2

$ns attach-agent $Server2 $sink3

$ns attach-agent $Server2 $sink4

$ns attach-agent $Server2 $sink5

$ns attach-agent $Server2 $sink6

$ns attach-agent $Server2 $sink7

$ns attach-agent $Server2 $sink8



# set sink1 [new Agent/Null]

# $ns attach-agent $Server2 $sink1

# set sink2 [new Agent/Null]

# $ns attach-agent $Server2 $sink2

$ns connect $udpz1 $sink0

$ns connect $udpz2 $sink1

$ns connect $udpz3 $sink2

$ns connect $udpc1 $sink3

$ns connect $udpc2 $sink4

$ns connect $udpc3 $sink5

$ns connect $udpc4 $sink6

$ns connect $udpc5 $sink7

$ns connect $udpc6 $sink8
```

```
#================================

#Color each flow id
#client 1-3 class
$udpc1  set class_ 1
$udpc2  set class_ 1
$udpc3  set class_ 1
#client 4-6 class
$udpc4  set class_ 3
$udpc5  set class_ 3
$udpc6  set class_ 3
#zombie class
$udpz1  set class_ 2
$udpz2  set class_ 2
$udpz3  set class_ 2

$ns color 1 Blue
$ns color 3 Green
$ns color 2 Red
```

```
set c_1_byte 0

set c_2_byte 0

set c_3_byte 0



proc record {} {


	global sink0 sink1 sink2 sink3 sink4 sink5 sink6 sink7 sink8 f0 f1 f2 f3 f4 f5 f6 
f7 f8 during c_1_byte before fafter c_2_byte c_3_byte

   #Get an instance of the simulator

   set ns [Simulator instance]

   #Set the time after which the procedure should be called again

        set time 0.2

   #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 [$sink6 set bytes_]

        set bw7 [$sink7 set bytes_]
        set bw8 [$sink8 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]"


# calculation


if {$now >= 4.8} {
         if {$now <= 9.4} {
         set c_1_byte [expr {$c_1_byte + $bw8}]
         puts $during "$now $c_1_byte "
         }
}
```

```
if {$now <= 4.8} {
        if {$now >= 0.2} {
        set c_2_byte [expr {$c_2_byte + $bw8}]
        puts $before "$now $c_2_byte "
        }
}


if {$now >= 9.4} {
        if {$now <= 14.0} {
        set c_3_byte [expr {$c_3_byte + $bw8}]
        puts $fafter "$now $c_3_byte "
        }
}


#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
$sink6 set bytes_ 0
```

```
            $sink7 set bytes_ 0
            $sink8 set bytes_ 0
    #Re-schedule the procedure
            $ns at [expr $now+$time] "record"
}

#Schedule events for the CBR agents

$ns at 0.0 "record"

#$ns at 0.3 "$cbr0 start"
#$ns at 1.0 "$cbr4 start"
#$ns at 1.0 "$cbr5 start"

#start client traffic c1-c6 respectively
$ns at 0.2 "$cbr6 start"
$ns at 0.2 "$cbr7 start"
$ns at 0.2 "$cbr8 start"
$ns at 0.2 "$cbr9 start"
$ns at 0.2 "$cbr10 start"
$ns at 0.2 "$cbr11 start"

#start zombie traffic z1-z3 respectively
$ns at 4.8 "$cbr1 start"
```

```
$ns at 4.8 "$cbr2 start"
$ns at 4.8 "$cbr3 start"

#stop zombies
$ns at 9.4 "$cbr3 stop"
$ns at 9.4 "$cbr2 stop"
$ns at 9.4 "$cbr1 stop"

$ns at 14.0 "$cbr11 stop"
$ns at 14.0 "$cbr10 stop"
$ns at 14.0 "$cbr9 stop"
$ns at 14.0 "$cbr8 stop"
$ns at 14.0 "$cbr7 stop"
$ns at 14.0 "$cbr6 stop"
#$ns at 4.5 "$cbr5 stop"
#$ns at 4.5 "$cbr4 stop"

#$ns at 4.5 "$cbr0 stop"


#Call the finish procedure after 5 seconds of simulation time
$ns at 15.6 "finish"

#Run the simulation
```

```
$ns run








##################################SFQ##############################
########


#Create a simulator object

set ns [new Simulator]


#Tell the simulator to use dynamic routing

#$ns rtproto DV


#Open the nam trace file

set nf [open out.nam w]

$ns namtrace-all $nf


set f0 [open out0.tr w]

set f1 [open out1.tr w]

set f2 [open out2.tr w]

set f3 [open out3.tr w]

set f4 [open out4.tr w]
set f5 [open out5.tr w]

set f6 [open out6.tr w]

set f7 [open out7.tr w]

set f8 [open out8.tr w]

set during [open outfduring.tr w]

set before [open outfbefore.tr w]

set fafter [open outfafter.tr w]



#Define a 'finish' procedure

proc finish {} {

        global ns nf f0 f1 f2 f3 f4 f5 f6 f7 f8 during before fafter

        $ns flush-trace

   #Close the trace file

        close $nf


        #Close the output files

        close $f0

        close $f1

        close $f2

   close $f3

        close $f4

        close $f5

   close $f6
```

```
        close $f7
        close $f8
    close $during
    close $before
    close $fafter
    #Execute nam on the trace file
    exec nam out.nam &
        #Call xgraph to display the results
        exec xgraph out0.tr out1.tr out2.tr out3.tr out4.tr out5.tr out6.tr out7.tr
out8.tr -geometry 800x400 &

        exit 0
}

#Create nine nodes
set Attacker [$ns node]
set Server   [$ns node]
set Server2   [$ns node]
set Zombie1  [$ns node]
set Zombie2  [$ns node]
set Zombie3  [$ns node]
set Router   [$ns node]
set Router2   [$ns node]
set Router3   [$ns node]
set Client1  [$ns node]

set Client2  [$ns node]

set Client3  [$ns node]

set Client4  [$ns node]

set Client5  [$ns node]

set Client6  [$ns node]


#Colors

$Attacker color  red

$Zombie1  color  red

$Zombie2  color  red

$Zombie3  color  red

$Server   color  blue

$Server2  color  blue

$Router   color  blue

$Router2  color  blue

$Router3  color  blue


# Labelling

$ns at 0.0 "$Attacker label Attacker"

$ns at 0.0 "$Zombie1  label Zombie"

$ns at 0.0 "$Zombie2  label Zombie"

$ns at 0.0 "$Zombie3  label Zombie"

$ns at 0.0 "$Server   label Gateway"
```

```
$ns at 0.0 "$Server2  label Server"
$ns at 0.0 "$Router   label Router"
$ns at 0.0 "$Router2  label Router"
$ns at 0.0 "$Router3  label Router"
$ns at 0.0 "$Client1  label client"
$ns at 0.0 "$Client2  label client"
$ns at 0.0 "$Client3  label client"
$ns at 0.0 "$Client4  label client"
$ns at 0.0 "$Client5  label client"
$ns at 0.0 "$Client6  label client"

# Shape
$Server  shape square
$Server2 shape square
$Router  shape hexagon
$Router2 shape hexagon
$Router3 shape hexagon


#Create links between the nodes
$ns duplex-link $Attacker  $Zombie1 10Mb 100ms SFQ
$ns duplex-link $Attacker  $Zombie2 10Mb 100ms SFQ
$ns duplex-link $Attacker  $Zombie3 10Mb 100ms SFQ
$ns duplex-link $Zombie1   $Router2 10Mb 100ms SFQ
```

```
$ns duplex-link $Zombie2   $Router2 10Mb 100ms SFQ
$ns duplex-link $Zombie3   $Router2 10Mb 100ms SFQ
$ns duplex-link $Router2   $Server  10Mb 100ms SFQ
$ns duplex-link $Router3   $Server  10Mb 100ms SFQ
$ns duplex-link $Server       $Router  10Mb 100ms SFQ
$ns duplex-link $Server       $Server2 5Mb  100ms SFQ
$ns duplex-link $Router       $Client1 10Mb 100ms SFQ
$ns duplex-link $Router       $Client2 10Mb 100ms SFQ
$ns duplex-link $Router       $Client3 10Mb 100ms SFQ
$ns duplex-link $Router3   $Client4 10Mb 100ms SFQ
$ns duplex-link $Router3   $Client5 10Mb 100ms SFQ
$ns duplex-link $Router3   $Client6 10Mb 100ms SFQ

#queue limit
#$ns queue-limit $Server     $Server2  50

# Setting node position
$ns duplex-link-op $Attacker $Zombie1 orient right-down
$ns duplex-link-op $Attacker $Zombie2 orient right
$ns duplex-link-op $Attacker $Zombie3 orient right-up
$ns duplex-link-op $Zombie1  $Router2 orient right-up
$ns duplex-link-op $Zombie2  $Router2 orient right
$ns duplex-link-op $Zombie3  $Router2 orient right-down
$ns duplex-link-op $Router2  $Server  orient right
```

```
$ns duplex-link-op $Server   $Router3 orient down
$ns duplex-link-op $Server   $Router  orient right
$ns duplex-link-op $Server   $Server2 orient up
$ns duplex-link-op $Router   $Client1 orient right-down
$ns duplex-link-op $Router   $Client2 orient right
$ns duplex-link-op $Router   $Client3 orient right-up

$ns duplex-link-op $Router3   $Client4 orient right-down
$ns duplex-link-op $Router3   $Client5 orient down
$ns duplex-link-op $Router3   $Client6 orient left-down



#Create a UDP agent and attach it to nodes
set udpattacker [new Agent/UDP]
$ns attach-agent $Attacker $udpattacker
set udpz1 [new Agent/UDP]
$ns attach-agent $Zombie1 $udpz1
set udpz2 [new Agent/UDP]
$ns attach-agent $Zombie2 $udpz2
set udpz3 [new Agent/UDP]
$ns attach-agent $Zombie3 $udpz3
# set udpRouter [new Agent/UDP]
# $ns attach-agent $Router $udpRouter
#======================
# set udpRouter2 [new Agent/UDP]

# $ns attach-agent $Router2 $udpRouter2

# set udpRouter3 [new Agent/UDP]

# $ns attach-agent $Router3 $udpRouter3

#========================

# set udpServer [new Agent/UDP]

# $ns attach-agent $Server $udpServer

set udpc1 [new Agent/UDP]

$ns attach-agent $Client1 $udpc1

set udpc2 [new Agent/UDP]

$ns attach-agent $Client2 $udpc2

set udpc3 [new Agent/UDP]

$ns attach-agent $Client3 $udpc3

set udpc4 [new Agent/UDP]

$ns attach-agent $Client4 $udpc4

set udpc5 [new Agent/UDP]

$ns attach-agent $Client5 $udpc5

set udpc6 [new Agent/UDP]

$ns attach-agent $Client6 $udpc6


# Create a CBR traffic source and attach it to udp agents

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize_ 1000

$cbr0 set rate_ 0.2Mb
```

```
$cbr0 set interval_ 0.2

$cbr0 attach-agent $udpattacker


set cbr1 [new Application/Traffic/CBR]

$cbr1 set packetSize_ 6000

$cbr1 set rate_ 2.4Mb;#ignored if interval is specified else interval =
(packetsize*8)/rate

$cbr1 set interval_ 0.02

$cbr1 attach-agent $udpz1

#interval = .01s/packet = 100packets/s, 3000 * 100 = 300000=.3MB/s * 8 = 2.4
Mb/sec

# 2.4 *3 zombies = 7.2 Mb/s

set cbr2 [new Application/Traffic/CBR]

$cbr2 set packetSize_ 6000

$cbr2 set rate_ 2.4Mb

$cbr2 set interval_ 0.02

$cbr2 attach-agent $udpz2


set cbr3 [new Application/Traffic/CBR]

$cbr3 set packetSize_ 6000

$cbr3 set rate_ 2.4Mb

$cbr3 set interval_ 0.02

$cbr3 attach-agent $udpz3
```

```
# set cbr4 [new Application/Traffic/CBR]

# $cbr4 set packetSize_ 1000

# $cbr4 set rate_ 0.2Mb

# $cbr4 set interval_ 0.2

# $cbr4 attach-agent $udpRouter


# set cbr5 [new Application/Traffic/CBR]

# $cbr5 set packetSize_ 1000

# $cbr5 set rate_ 0.2Mb

# $cbr5 set interval_ 0.2

# $cbr5 attach-agent $udpServer


set cbr6 [new Application/Traffic/CBR]

$cbr6 set packetSize_ 1000

$cbr6 set rate_ 0.2Mb

$cbr6 set interval_ 0.2

$cbr6 attach-agent $udpc1

# 8000 bits/packet , 0.2 s/packet , 8000/0.2 = 40k bits/s , 0.04 Mbits/s

set cbr7 [new Application/Traffic/CBR]

$cbr7 set packetSize_ 1000

$cbr7 set rate_ 0.04Mb

$cbr7 set interval_ 0.2

$cbr7 attach-agent $udpc2
```

```
set cbr8 [new Application/Traffic/CBR]

$cbr8 set packetSize_ 1000

$cbr8 set rate_ 0.2Mb

$cbr8 set interval_ 0.2

$cbr8 attach-agent $udpc3


set cbr9 [new Application/Traffic/CBR]

$cbr9 set packetSize_ 1000

$cbr9 set rate_ 0.2Mb

$cbr9 set interval_ 0.2

$cbr9 attach-agent $udpc4


set cbr10 [new Application/Traffic/CBR]

$cbr10 set packetSize_ 1000

$cbr10 set rate_ 0.2Mb

$cbr10 set interval_ 0.2

$cbr10 attach-agent $udpc5


set cbr11 [new Application/Traffic/CBR]

$cbr11 set packetSize_ 1000

$cbr11 set rate_ 0.2Mb

$cbr11 set interval_ 0.2

$cbr11 attach-agent $udpc6
```

```
#Create a Null agent (a traffic sink) and attach it

#set sink0 [new Agent/Null]

#$ns attach-agent $Server2 $sink0


#for xgraph=====================

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 sink6 [new Agent/LossMonitor]

set sink7 [new Agent/LossMonitor]

set sink8 [new Agent/LossMonitor]

# Agent/LossMonitor objects as traffic sinks, since they store the amount of bytes
received, which can be used to calculate the bandwidth.

$ns attach-agent $Server2 $sink0

$ns attach-agent $Server2 $sink1

$ns attach-agent $Server2 $sink2

$ns attach-agent $Server2 $sink3

$ns attach-agent $Server2 $sink4

$ns attach-agent $Server2 $sink5

$ns attach-agent $Server2 $sink6

$ns attach-agent $Server2 $sink7
```

```
$ns attach-agent $Server2 $sink8



# set sink1 [new Agent/Null]

# $ns attach-agent $Server2 $sink1

# set sink2 [new Agent/Null]

# $ns attach-agent $Server2 $sink2

$ns connect $udpz1 $sink0

$ns connect $udpz2 $sink1

$ns connect $udpz3 $sink2

$ns connect $udpc1 $sink3

$ns connect $udpc2 $sink4

$ns connect $udpc3 $sink5

$ns connect $udpc4 $sink6

$ns connect $udpc5 $sink7

$ns connect $udpc6 $sink8





#===============================



#Color each flow id
```

```
#client 1-3 class
$udpc1  set class_ 1
$udpc2  set class_ 1
$udpc3  set class_ 1
#client 4-6 class
$udpc4  set class_ 3
$udpc5  set class_ 3
$udpc6  set class_ 3
#zombie class
$udpz1  set class_ 2
$udpz2  set class_ 2
$udpz3  set class_ 2

$ns color 1 Blue
$ns color 3 Green
$ns color 2 Red


set c_1_byte 0
set c_2_byte 0
set c_3_byte 0


proc record {} {
        global sink0 sink1 sink2 sink3 sink4 sink5 sink6 sink7 sink8 f0 f1 f2 f3 f4 f5 f6
f7 f8 during c_1_byte before fafter c_2_byte c_3_byte

   #Get an instance of the simulator

   set ns [Simulator instance]

   #Set the time after which the procedure should be called again

        set time 0.2

   #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 [$sink6 set bytes_]

        set bw7 [$sink7 set bytes_]

        set bw8 [$sink8 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]"

# calculation

if {$now >= 4.8} {
         if {$now <= 9.4} {
         set c_1_byte [expr {$c_1_byte + $bw8}]
         puts $during "$now $c_1_byte "
         }
}



if {$now <= 4.8} {
         if {$now >= 0.2} {
         set c_2_byte [expr {$c_2_byte + $bw8}]
         puts $before "$now $c_2_byte "
         }
}
```

```
    if {$now >= 9.4} {
            if {$now <= 14.0} {
            set c_3_byte [expr {$c_3_byte + $bw8}]
            puts $fafter "$now $c_3_byte "
            }
    }



    #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
    $sink6 set bytes_ 0
           $sink7 set bytes_ 0
           $sink8 set bytes_ 0
    #Re-schedule the procedure
           $ns at [expr $now+$time] "record"
}

#Schedule events for the CBR agents
```

```
$ns at 0.0 "record"

#$ns at 0.3 "$cbr0 start"
#$ns at 1.0 "$cbr4 start"
#$ns at 1.0 "$cbr5 start"

#start client traffic c1-c6 respectively
$ns at 0.2 "$cbr6 start"
$ns at 0.2 "$cbr7 start"
$ns at 0.2 "$cbr8 start"
$ns at 0.2 "$cbr9 start"
$ns at 0.2 "$cbr10 start"
$ns at 0.2 "$cbr11 start"

#start zombie traffic z1-z3 respectively
$ns at 4.8 "$cbr1 start"
$ns at 4.8 "$cbr2 start"
$ns at 4.8 "$cbr3 start"

#stop zombies
$ns at 9.4 "$cbr3 stop"
$ns at 9.4 "$cbr2 stop"
$ns at 9.4 "$cbr1 stop"
```

```
$ns at 14.0 "$cbr11 stop"

$ns at 14.0 "$cbr10 stop"

$ns at 14.0 "$cbr9 stop"

$ns at 14.0 "$cbr8 stop"

$ns at 14.0 "$cbr7 stop"

$ns at 14.0 "$cbr6 stop"

#$ns at 4.5 "$cbr5 stop"

#$ns at 4.5 "$cbr4 stop"


#$ns at 4.5 "$cbr0 stop"



#Call the finish procedure after 5 seconds of simulation time

$ns at 15.6 "finish"


#Run the simulation

$ns run
```