12-2009

Analysis of Distributed Denial of Service Attacks and Countermeasures

Sampada Karandikar
Clemson University, skarand@g.clemson.edu

ANALYSIS OF DISTRIBUTED DENIAL OF SERVICE ATTACKS AND COUNTERMEASURES

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

---

by
Sampada Karandikar
December 2009

Accepted by:
Dr. Richard R. Brooks, Committee Chair
Dr Christopher Griffin
Dr. Ian Walker

TABLE OF CONTENTS

Page

APPENDICES

LIST OF FIGURES

LIST OF TABLES

LIST OF GRAPHS

DEDICATION

This thesis is dedicated to my parents, who have raised me to be the person I am today. Thank you for all the guidance and support that you have always given me. Thank you for everything.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Dr Richard R. Brooks for having believed in me and supporting me during the entire period of this research. Your patience and suggestions about what to do next were highly invaluable.

I would like to thank my family for all the support throughout my undergraduate and graduate academic career. A special thank you to my grandmothers in India for encouraging me during every phone conversation.

This thesis would not have been possible without the support of my friends in Clemson who were with me throughout the completion of the thesis and providing me encouragement when it seemed impossible to continue. A special thank you to my colleagues in my research group – Jason, Ryan, Hari, Lu Chen and my labmates, Juan Deng and Yu Lu for keeping up the spirit and giving me company while working.

ABSTRACT

Network technology has experienced explosive growth in the past two decades. The vast connectivity of networks all over the world poses monumental risks. The generally accepted philosophy in the security world is that no system or network is completely secure [1] which makes network security a critical concern. The work done in this thesis focuses on Distributed Denial of Service Attacks (DDoS) where legitimate users are prevented from accessing network services. Although a lot of research has been done in this field, these attacks remain one of the most common threats affecting network performance.

One defense against DDoS attacks is to make attacks infeasible for an attacker, by increasing either the amount of attack traffic needed to disable a link or the number of attackers needed to disable the network.

Theoretical work has been done previously which focused on quantifying the attack traffic required to disable a set of mincut arcs in a network. In this thesis, we experimentally verify the validity of the analysis performed by running simulations using the SSFNet network simulator. A Distributed Denial of Service attack is simulated by flooding the mincut arcs in the network. From the results, we analyze

- The minimum number of zombie processors (attack sources) required to disable a set of arcs
- The minimum attack traffic volume required to disable the arcs.

CHAPTER ONE

INTRODUCTION

As computer systems have evolved into today's complex, enterprise-wide, solutions, the security risks and protective measures have also become complex [2]. Maintaining the security of a system involves maintaining confidentiality, authentication, integrity, non-repudiation, access control and availability [3]. However, the concept of complete security is an illusion [4]. Almost anyone can reach out to any network which implies that anyone can reach in [5].

The lack of authentication means an attacker can create a fake identity, and send malicious traffic. A Denial of Service (DoS) attack blocks a service for legitimate users and is perpetrated by causing a victim to receive malicious traffic and suffer damage as a consequence [6]. The attack can be launched in multiple forms. The attack could exploit software vulnerabilities of a target thereby crashing the system, or use massive volumes of malicious traffic to consume key resources thus rendering it unavailable to legitimate users, or simply send a few malformed packets to confuse an application or a protocol on the victim machine and force it to freeze or reboot [6]. While it is possible to patch the known vulnerabilities in a system to avoid an attack, it is difficult to prevent the second and third form of attacks. The targets are vulnerable simply because they are connected to the Internet. When the traffic of a DoS attack comes from multiple sources, then it is called as a Distributed Denial of Service (DDoS) Attack [7].

In today's world, botnets are a major source of DDoS problems. Since botnets usually involve computers from many countries, tracking an attack becomes more difficult. Statistics show that about half of the botnets tracked by Arbor networks performed DDoS attacks [8]. A DDoS can have a sustained upload bandwidth of 40 Kb/s as an average from each bot. A relatively small botnet can overwhelm most companies, and a large botnet might be able to take out a fair –sized ISP [9].

1.1 Motivation

In the late 1990's the world was not dependent on the Internet as it is now. The Internet was still limited to research and educational communities. Hence not much attention was paid to Internet security. Today, the traditional role of the Internet has changed. Internet is used for banking, bill payments, tax payments, booking travel reservations, online shopping. It is used by Governments to share information with the world, by researchers as a medium for disseminating their research discoveries rapidly and for establishing worldwide connectivity [7]. Unfortunately, the growth in the Internet has increased the number of attacks on the Internet. Figure 1.1 shows a graph of the number of security incidents reported in the past.

The recent attacks on popular websites like Facebook and Twitter are an example of the rising number of DDoS attacks. One of the major problems with Distributed Denial of Service attacks is the difficulty to detect the source of the attack because of the many components involved. Instead of waiting for an enemy to attack, it is better to use defenses to protect networks or make the networks immune to attacks [11].

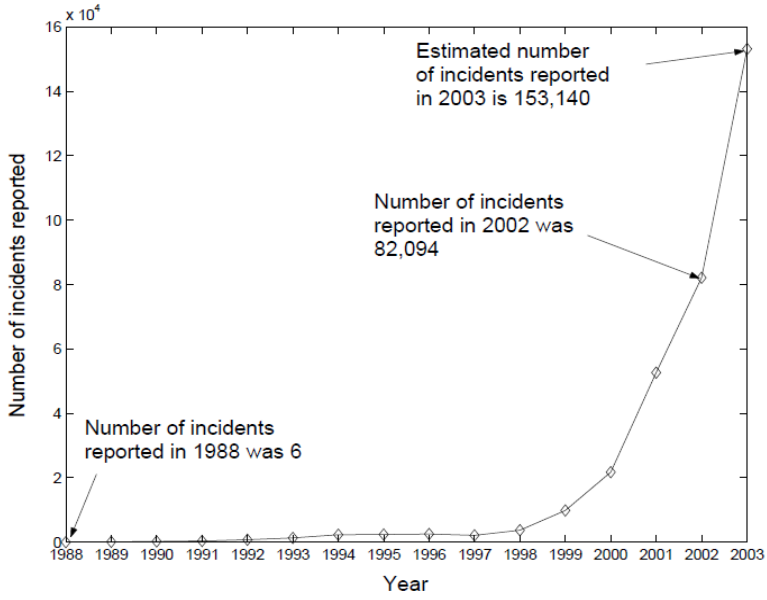


Figure 1.1: The number of Internet security incidents reported from 1988 to 2003

(Data collected from [10])

1.2 Approach

The work done in [11] studies the dynamics of DDoS attacks. The attacker places "zombie" processes on the network that consume network bandwidth. The attacker then attempts to break the legitimate communications links. The legitimate application reconfigures its network to re-establish communications. The authors analyze this board game using the theory of surreal numbers [12]. The authors also quantify the number of zombies and the amount of zombie traffic that an attacker needs to disable a distributed application. We perform simulations to verify if the analysis corresponds to reality. The simulations are performed for large scale complex networks which are generated automatically using Python scripting. We observe from the results that the legitimate

traffic is reduced to a significant amount when the attack traffic is increased beyond the threshold value calculated by the formula.

1.3 Defense Mechanisms

The DDoS defense mechanisms are classified as being reactive and preventive. In reactive measures, the attack sources are identified and are prevented from continuing the attack. The preventive measures focus on eliminating the possibility of performing a DDoS attack. Before concluding that a denial of service attack is under progress, it is necessary to identify and separate DoS attacks from flash events. This is discussed in more detail in the Chapter on DDoS countermeasures.

In this thesis, we provide a countermeasure when an enterprise network wants to maintain communications even though an opponent launches a DDoS attack. We solve this problem using a game theoretical approach which is explained in further detail in the chapter on Bandwidth Limited Co-ordination of games.

1.4 Game Theory based coordination

We look at scenarios when the legitimate application has a set of networks connected by bandwidth limited communication links. The application coordinates amongst its networks by sending only the most important information. If there are multiple messages, then it becomes necessary to prioritize the messages and send the one which is the most consistent with the team goal. We study four different strategies to

make this decision – Maximin, Maximax, Central values and Hotstrat and conclude that the Hotstrat strategy gives the best results.

1.5 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 gives a background on Distributed Denial of Service attacks. Chapter 3 gives a brief overview of the SSFNet simulator and the implementations of the protocols used. It also explains the simulation scenario used for the simulations and explains some simulation scenarios. Chapter 4 details the steps involved in setting up and automating the simulation generation process for large scale networks. Chapter 5 explains the simulation scenario and the results obtained. Chapter 6 focuses on the countermeasures used for DDoS attacks and a summary of the work done. Chapter 7 explains our approach to mitigation using the principles of combinatorial game theory. We conclude the thesis with Chapter 8 presenting our conclusions and future directions for research.

CHAPTER TWO

BACKGROUND ON DISTRIBUTED DENIAL OF SERVICE ATTACKS

In a denial of service attack, an attacker deliberately consumes resources making them unavailable to legitimate users. One common denial of service attack is the 'flooding attack'. Typically, to access a website, a request is sent to the website's server. Since there is an upper limit on the number of requests that a server can process, the request is rejected if this limit is exceeded. In a flooding attack, the attacker floods the website's server with a large number of requests thus preventing legitimate users from accessing information or services [13]. Similarly, for a free mailing service, there is a specific disk quota assigned to an email account. The quota limits the amount of data that a user can store in his account at any point of time. If an attacker sends a large amount of data to the email account, it might prevent the user from receiving legitimate emails. This is a practical example of a Denial of Service attack.

In a Distributed Denial of Service attack, multiple machines launch the attack. The attack thus has a distributed nature. The attacker can make use of the security vulnerabilities of a system to launch the attack. There is a high probability that the machine used to launch the attack is unaware that it is participating in the attack.

Sometimes, it becomes difficult to distinguish a distributed denial of service from normal network activity. At other times, there might be some indications that an attack is under progress.

The single attacker who coordinates the attack is called the Master. The Master coordinates multiple hijacked systems. These hijacked systems are called zombies. Figure 2.1 shows the different components of a DDoS attack.

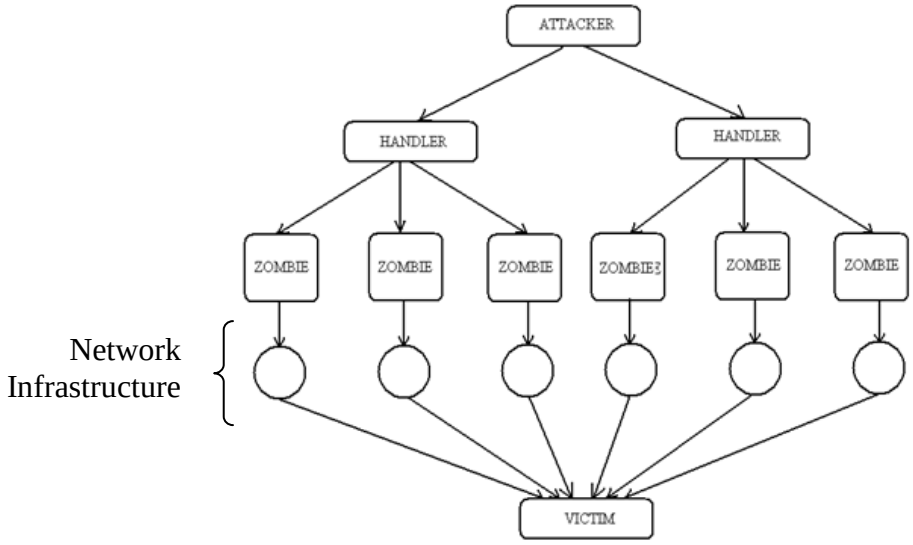


Figure 2.1: DDoS Attack

(Data taken from [11])

DDoS attacks are considered more effective and complicated than their DoS counterparts because the attack can be performed from multiple sites simultaneously and the task of detecting the attacker becomes almost impossible. The next section discusses some of the commonly used methods of Denial of Service attacks.

2.1 Common Denial of Service Attacks

2.1.1 Smurf attack

In a Smurf attack, the attacker sends a large number of ICMP echo requests to a set of IP broadcast addresses. All the echo requests have a spoofed source IP address of the intended victim. On receiving the echo requests, most of the hosts will respond to the request with an echo reply. This increases the flooding traffic by the number of hosts responding.

2.1.2 SYN Flood attack

This attack exploits the standard TCP three-way handshake that is initiated before a TCP transmission. The handshake consists of a three packet exchange sent by client to the server. The server upon receiving the initial SYN from the client responds with a SYN/ACK packet and waits for the client to send a final ACK. If the client sends a huge number of SYNs without sending their corresponding ACKs, then the server keeps waiting for the non-existent ACKs making it impossible for the server to serve other incoming connections.

2.1.3 UDP Flood attack

Here the attacker uses the UDP echo and character generator service. Forged UDP packets are used to connect the echo service on one machine to the character generator

service on another machine. The two services consume a lot of network bandwidth as they exchange characters between themselves. [13]

Some variations of Distributed Denial of Service attacks are mentioned below. These attacks use the techniques mentioned in the section above.

2.2 Known Distributed Denial of Service Attacks

2.2.1 Trinoo

One of the most popular DDoS attacks is the Trinoo attack where the attack daemons use UDP flood attacks to disable the victim. It consists of an attacker system, several compromised systems, which include one or more masters (referred to as handlers), one or more daemon systems (referred to as agents), and one or more victims. The attack begins by loading the Trinoo program on one or more compromised systems. These systems act as handlers and agents. The agents send a UDP packet to let the handler know that the agent systems are ready. When the attack system sends the attack command, the handler sends a message to the agents to launch the attack. After receiving the command to launch an attack, the agent sends a UDP flood to random port numbers on the victim. This attack was experienced in 1999 by University of Massachusetts.

2.2.2 Tribe Flood Network

The TFN attack is more complicated than a Trinoo attack. The TFN software is loaded by the TFN attacker onto compromised systems. In order to launch the attack, the

attack systems simply need remote access to the handler. TFN's attacks daemons can implement Smurf, SYN flood, UDP flood and ICMP flood attacks.

2.2.3 Stacheldraht

Stacheldraht is a combination of Trinoo and TFN attack and relies on TCP for transport. The handlers and agents periodically exchange ICMP reply packets. It encrypts the communication between the attacker and the masters and performs automated update of the agents. It can implement Smurf, SYN flood, UDP flood and ICMP flood attacks. [14]

2.3 Prevalence of Distributed Denial of Service attacks

Businesses have been shut down for several hours by faceless hackers in the past. The DDoS attack slows the system performance and ultimately crashes the system. This section talks about some of the DDoS attacks experienced in the last decade.

DoS attacks crippled high visibility Internet websites like Yahoo, CNN and major ecommerce sites like Amazon.com and Buy.com which were down for three hours as a result of the attack in the February of 2000. The sites started behaving poorly with the Amazon site timing out at various stages throughout the night. Yahoo experienced traffic levels of 1 GB per second. This attack is believed to have been a DDoS where multiple compromised machines were involved.

In January 2001, Microsoft's name server infrastructure was disabled by a Denial of Service assault. The root DNS servers were targeted in the following year and SCO's corporate website was incapacitated in late 2003. [15]

Recently, in August 2009, a string of major websites experienced a DDoS attack. This DDoS was interesting as the attack seems to have been URLs embedded in spam. People clicking on the spam links generated enough traffic to kill the websites. The malicious online attacks affected services of major social networking websites like Facebook, Twitter and Google. Facebook encountered network issues that resulted in degraded service for some of the users. Twitter's website was unavailable for at least two hours. [16]

The military has been the victim of cyber attacks in the past. [3] A National Security Agency red team of hackers was organized to infiltrate the Pentagon systems. The team was able to infiltrate and take control of the Pacific command centre computers, as well as power grids and 911 systems in nine major US cities. Code Red was a worm that first appeared in 2001 and ultimately affected nearly 300,000 computers in the United States. It exploited a hole in Microsoft's IIS web servers. In its first variation, the affected computers were programmed to launch a denial of service attack against the White House website at a certain date and time. [17]

2.3.1 Information Warfare

The cyber attack in Estonia was considered as the first war in cyberspace. It was deemed as a national security situation. In Estonia, ‘ the most wired country’, the Internet is vital and is used routinely to vote, file taxes, and even to shop or pay for parking. The bulk of the cyber assaults used DDoS to bombard the country's Web sites with data. The attackers clogged not only the country's servers, but also made it difficult to direct traffic on the network. In one of the first attacks, a flood of junk messages were thrown at the e-mail server of the Parliament, shutting it down. In another, hackers broke into the Web site of the Reform Party, posting a fake letter of apology from the prime minister. Traffic spiked to thousand times the normal flow. The biggest bank in Estonia had to shut down its online service for more than an hour. It suffered losses of about $1 million. The 10 largest assaults blasted streams of 90 Mbps of data at Estonia’s networks. The attackers used a giant network of bots – perhaps as many as one million computers as far as Vietnam and United States, to amplify the impact of their assault. There is evidence that they rented time on other so-called botnets. [18]

The vast majority of attacks are not even publicized. The victims include a wide range of targets victims from small commercial sites to education institutions. The work in [15] is based on backscatter analysis to estimate the worldwide prevalence of denial of service attacks. They established an alarming presence of roughly 2000 – 3000 active denial of service attacks per week.

2.4 What makes DDoS attacks possible?

We elaborate some of the design issues of the Internet which makes DDoS attacks possible [6].

1. The end-to-end paradigm pushes the complexity to end hosts, leaving the intermediate network simple and optimized for packet forwarding. Thus if one party in a two way communication misbehaves; it can do arbitrary damage to its peer.

2. Attacks are commonly launched from systems that are subverted through security related compromises. So regardless of how well secured the victim system may be, its susceptibility depends on the state of the security of the global internet.

3. Since each Internet entity has limited resources, it can be consumed if there are too many users.

4. The intelligence needed for service guarantees resides with the end hosts. High bandwidth pathways are available in the intermediate network, while the end networks have bandwidth only as much as they need. Thus malicious clients can misuse the abundant resources of the intermediate network for delivery of numerous messages to a less provisioned victim.

5. Due to IP spoofing, attackers get a powerful mechanism to escape accountability for their actions.

2.5 Reasons for DDoSing

There can be numerous reasons for a DDoS, the primary goal being to inflict damage on the victim. The true victim of the attack might not be the actual target, but others who rely on the target's correct operation. The reasons for DDoS could be personal where an attack can be perpetrated for the purpose of revenge or they could be material in which the attack damages a competitor's resources. The attacks could be performed by hackers simply to gain respect (by successfully attacking popular websites) or may be performed for serious political reasons where a country at war could perpetrate attacks against its enemy's critical resources. Victims may be blackmailed into paying to avoid DDoS attacks. Recent reports have botnets being rented for performing DDoS attacks at a rate of $1000 per spam or DDoS event [39].

2.6 Challenges faced in DDoS Defense

There are several serious factors that hinder the advance of DDoS research. We list some of them in this section [6].

1. There are very few DDoS attacks which can be handled only by the victim. In order to deal with DDoS, it often becomes necessary to have a distributed and coordinated response system. The Internet being a system which itself is administered in a distributed manner, it is difficult to enforce cooperation between networks which discourages researchers from designing distributed solutions.

2. Deployment of a distributed response system implies that parties that will bear the deployment cost are parties that do not suffer direct damage and hence do not benefit directly from the system.
3. There is lack of information on attack parameters used for popular DDoS attacks since publicly reporting DDoS attacks damages the reputation of the victim
4. There is no bench mark suite of attack scenarios that enables comparison between defense systems [6].

2.7 Previous Work

We consider mechanisms for constructing distributed DDoS defense that do not require cooperation among uninvolved parties. The results in this thesis build on previous research documented in [11]. The work in [11] analyzes a two player game played on a computer network in which Player 1 (Blue player) is a legitimate distributed application on a network and Player 2 (Red player) is an attacker who places zombie processes in the network with an intention of attacking node capacities or flooding the arcs between nodes. By finding out the minimum number of zombies needed to disable Blue's network, the authors quantify the resistance of the Blue player to DDoS attacks. This approach helps to design networks with a structure that either resist DDoS attacks or adapt around them. This result is relevant to item 4 in Section 2.6 in that it provides a metric for comparing DDoS countermeasures.

The mincut of a network configuration is a set of network edges whose removal prevents source communication with the destination. The attacker would be interested in

determining the smallest number of arcs that need to be disabled from this mincut. The authors in [11] describe an algorithm to determine the minimum number of arcs that need to be disabled between the source and the destination to avoid the attacker wasting resources attacking arcs that need not be attacked.

Once the arcs that need to be disabled are known, the work further determines the amount of flow to be directed towards these arcs in order to disable them. If RT denotes the minimum amount of traffic that should be generated by the zombies (also called red traffic), λ denotes the Blue (Legitimate) traffic required by Blue's application and C represents that capacity of the physical arc to be attacked, then the total traffic T is given by

$$T = (\lambda + RT) \tag{2.1}$$

The traffic dropped is represented as D

$$D = (\lambda + RT) - C \tag{2.2}$$

Percentage of legitimate traffic in the total traffic P is given by

$$P = \lambda / (\lambda + RT) \tag{2.3}$$

Expected rate of legitimate traffic loss (LTL) is given by

$$LTL = \lambda / (\lambda + RT)[(\lambda + RT) - C] \tag{2.4}$$

If the Blue slack traffic is BS, then the attacker wins in flooding the arc if

$$LTL \geq BS \tag{2.5}$$

where

$$BS = Capacity\ \text{-}[Blue\ Flow] \tag{2.6}$$

From equations (2.4) and (2.5)

$$BS \leq LTL \tag{2.7}$$

Solving further, we get

$$\therefore \frac{BS(\lambda + RT)}{\lambda} \leq (\lambda + RT) - C \tag{2.8}$$

Thus the minimum amount of Red traffic required is given by

$$RT = \frac{C}{1 - (\frac{BS}{\lambda})} - \lambda \tag{2.9}$$

where

$$BS = C - \lambda \tag{2.10}$$

Equation (2.9) represents the threshold value of the zombie traffic that should be generated to disable an arc.

CHAPTER THREE

SSFNET NETWORK SIMULATOR

3.1 Purpose of Simulations

Communication networks are rapidly increasing in complexity, volume, and cost. This has been exponential in the recent past, making it imperative to study the behavior of a network before it is deployed. An experimental network test bed comes across as a practical approach to observe network behavior. However, the cost and time involved in deploying such a test bed is the same as deploying the network itself, making such a study infeasible.

Simulations are a cost effective solution to this problem. They are inexpensive, and quickly deployed. Network simulation tools help researchers and developers estimate network functionality and performance prior to deployment. They are a virtual environment for testing the performance of new networking protocols. They model networks and analyze their performance under different scenarios. To make network operations effective, simulations can inspect the vulnerabilities that may exist in the network. Simulations are often used in test scenarios where it is difficult and infeasible to use network hardware. Simulations provide a controlled and reproducible environment for simulating network attacks.

Simulators allow users to specify the nodes in a network, the links connecting nodes and the flow over links [19]. Most simulators offer a programming framework through which the user can customize the network environment. They reflect the behavior of network components like routers, multiple hosts and various types of network links. The following sections explain the reasons SSFNet is suitable for simulating large networks.

3.2 Scalable Simulation Framework

The Scalable Simulation framework is written in Java and C++. The framework (SSF) allows discrete-event simulation for large complex networks. [20] Researchers have used this framework to design network simulators like DaSSF and SSFNet.

3.3 SSFNet

SSFNet has a single integrated interface which can be used to design networks. It models Internet protocols at and above the network layer.

The Internet consists of a large number of heterogeneous network elements making it difficult to simulate. The Internet is an ‘immense moving target” which grows at an exponential rate undergoing dramatic qualitative changes over time [21]. The scalable simulation framework was developed as a scalable model of the Internet. SSFNet has a modular structure, allowing additional packages to be used to model specific domains. This strategy promotes independence of models from the simulation fabric and liberates the simulation fabric from the specifics of parallel discrete-event simulation engines [19].

SSFNet describes a series of objects which when combined make it possible to define large networks. The simulation sizes that can be handled are directly proportional to the processing power of the system the simulations are run on.

The SSFNet distribution consists of two frameworks - SSF.OS and SSF.Net. Any Internet model can be constructed using these frameworks. The simulator architecture consists of three main components.

1. DML (Domain Modeling Language): The network configuration files needed for running simulations are written in DML. DML files consist of a hierarchical list of recursively defined attribute value pairs. [22]
2. SSF: It is a public domain standard for discrete event simulations of large complex networks.
3. SSFNet: This is a collection of SSF based open source Java models of protocols, network elements and supported classes. This consists of SSF Network models for modeling and simulation. [23]

Figure 3.1 shows the SSFNet simulation layers

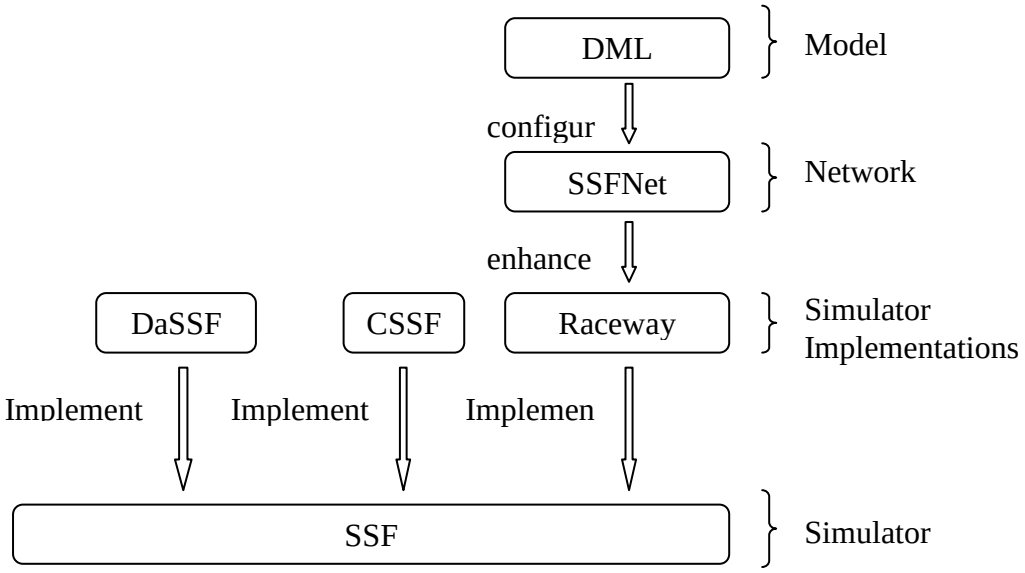


Figure 3.1 Simulation layers in SSFNet

3.4 SSFNet Objects

SSFNet segments the network structure into groups. Groups are repeated as necessary and patched together to create large networks.

3.4.1 Net

The top level Net cannot have an ID. Every network configuration is simply the value of the Net attribute enclosed within Net […]. The included Net is a collection of Hosts, Routers and links and must contain a single ID value or a range of ID values, which identify the network it is configuring. The ID values should be unique.

3.4.2 Frequency

The frequency parameter sets the time resolution of the simulation as the number of simulation ticks per clock second. For instance, if the frequency is set to 1000000, then the simulated time will advance by intervals measured in microseconds. It is used only in the top level Net.

3.4.3 Host

A host could be a client computer or a server. It can have zero or more configured interfaces. The host must have an ID value assigned to it and the ID values must be unique for a particular Net loop.

3.4.4 Graph

The graph component specifies the list of protocols to be used. It is mandatory for every host to have a graph attribute in its definition. The graph attribute has a number of sub attributes within it and there is one graph per installed protocol. The ‘name’ is a symbolic tag by which a protocol implementation finds its configuration and ‘use’ specifies the SSFNet class that should be loaded to do the protocol’s job.

```
graph [
     ProtocolSession [
          name foo
          use SSF.OS.bar
          other protocol-dependent parameters
     ]
]
```

Figure 3.2 Graph attribute

3.4.5 Interface

The interface facilitates the configuration of the network interface of the Ethernet card. The interface also needs to have an ID value which uniquely identifies network interfaces for a particular host. A host may have multiple interfaces, but typically has one or two. A router can have multiple interfaces as in the case of our simulation. An important attribute of the interface is the bitrate which specifies the rate at which packets leave the interface. The default bitrate in SSFNet is 10 Mbps. In our simulation, we have specified the bitrate for every interface to be between 6000000 and 7000000 bps. The latency attribute of an interface specifies the delay introduced by the interface itself. The queue and the buffer attributes are optional. The queue specifies the queue manager for a particular interface whereas the buffer attribute specifies the buffer of the queue in bytes. If the size of the incoming packet is greater than the currently available free buffer space, then the packet is dropped. It is possible to assign an IP address to an interface. If one is not assigned, then the simulator assigns one.

3.4.6 Router

The router component is similar to the host component. The difference is that it will have distinct protocols in its graph component. It could be considered as an intermediate host which cannot originate data.

3.4.7 Link

It specifies a link layer connection. It connects a set of hosts, or router interfaces. It must include the attribute ‘attach’ which specifies the attached network interfaces. The delay attribute specifies the contribution of the link to the total transmission delay.

3.4.8 Traffic

The traffic component specifies the traffic scenario for different client/servers. It is used by protocols like TCP, UDP and HTTP. Traffic could have one or more sub patterns and each pattern should specify one client attribute and one or more server attributes. The client should be specified with the NHI address of the host or client. The format for the traffic attribute is as shown in Figure 3.3

```
traffic [
     pattern [
          client 2
          servers [nhi 1(0) port 1600]
     ]
]
```

Figure 3.3 Traffic attribute

3.5 Addressing

NHI addressing is used as an internal addressing format for model building convenience [33]. It has the form as shown in equation (3.1).

$$N:N:N:.....N:H(I) \tag{3.1}$$

Where N represents the network ID, H is the host ID, and I is the interface ID. The addressing uses concatenated IDs of each network from the outermost network to the innermost network/host which are separated by colons, followed by the interface number (NIC ID) in parenthesis after the host containing the interface. For instance, if a Network with ID 1 contains a host with ID 3 which has an interface ID 4, then the interface would have a NHI address that is represented as 1:3(4). Figure 3.4 shows a simple example of a network with two networks and a host in each network.

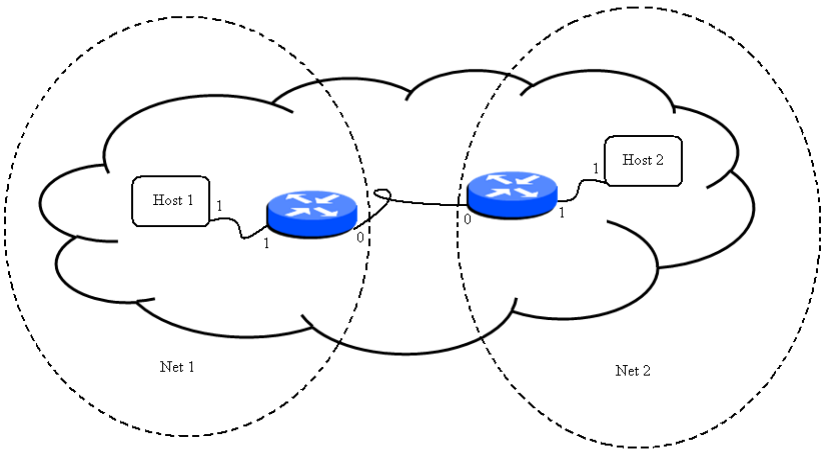


Figure 3.4 A simple network configuration

The individual networks are defined as in Figure 3.5

```
Net[
     Host [id 1 interface [id 1]]
     link [attach 1(1) attach 0(1)]
]
```

Figure 3.5 Net definition

The top level Net is defined as shown in Figure 3.6

```
Net[
     Net [id 1 …..]
     Net [id 2 …..]
     link [attach 1:0(0) attach 2:0(0)]
]
```

Figure 3.6 Top level Net definition.

CIDR addresses may also be explicitly assigned to a network or link. If they are not assigned, then SSF.Net.Net automatically computes them relative to the Net in which they are defined. The IP addresses may be assigned manually, or by using optional attributes to guide the IP address algorithm implemented by SSFNet or could be automatically assigned if no attributes are provided.

3.6 Protocol Implementation in SSFNet

3.6.1 IP implementation

The IP implementation in SSFNet keeps tracks of all the interfaces configured on a particular host or router and the IP addresses of the interfaces that are attached to the links. Routes are not computed by SSF.OS.IP but are computed by routing protocols. The IP protocol session decrements the TTL field in the IP header and drops the packet when the field decrements below zero. The packet is pushed down to the next hop interface by the host/router if the destination is not reached. If the host/router is the destination, then the packet is pushed up to the appropriate protocol mentioned in the Protocol Session.

3.6.2 OSPF implementation

In our simulation, the routers use the OSPF protocol to compute the routing tables. The specification of the OSPF protocol occurs in the router's Protocol Graph specification. At the start of the simulation, protocol finds all the neighbor routers, creates the link state database and computes the routes. The static version of the link state protocol (sOSPF) is used in our simulations, which is a simplified version of the OSPFv2 protocol. This protocol implementation does not perform load balancing between paths of equal cost.

3.6.3 UDP implementation

We use client server models with UDP streaming traffic. The UDP client configuration should specify the earliest time to send a request to the UDP server and must specify the size of the requested file in bytes. The UDP server configures itself with the parameters specified in the DML file. These parameters include the datagram size which is the payload in bytes and the send interval which specifies the interval between two consecutive chunks of data. The client sends one integer specifying the amount of data it wants to the server's well known address. On receiving a client's request, the server spawns a slave server which periodically sends the requested datagram size to the client until all the bytes of the file are sent [24]. UDP is used, since it does not contain flow control. TCP contains end to end flow control, which decreases transmission speed exponentially once packets start being dropped. In the context of our study, this has the following drawbacks:

- It would make flooding DDoS attacks aimed at decreasing available bandwidth easier to implement,
- It makes it impossible to exactly quantify the throughput rates that can be achieved during a DDoS, and
- TCP was designed to provide reasonable throughput for a set of cooperating network flows. It does not provide reasonable strategies for either performing or countering selfish DDoS attacks.

3.7 Simulation software environment

The automation of the network generation process is performed using the Python programming language. The version used is 2.5; however some of the libraries from version 2.6 are also required. The Python plugin - Pydev is integrated in the Eclipse software environment. Pydev is installed by the Eclipse update manager which automatically downloads the plugin from the website entered. The maxflow mincut program is written in Matlab. The input from Python is appropriately converted to a format accepted by Matlab.

3.8 Simple SSFNet Examples

In this section, we discuss some simple simulation examples to illustrate SSFNet.

3.8.1 Configuration 1

The network configuration simulated had the structure shown in Figure 3.7. Since the connections between networks are between the routers of the corresponding networks, the interface of the router of Network 1 is connected to the interface of the router of Network 2. Network 1 and 3 each consist of two hosts as shown in Figure 3.7. The clients in Network 3 request a certain amount of data from the corresponding servers in Network 1. The data flows from the server to the client according to the UDP client server implementation in SSFNet. In the simulation, both clients request a file size of 3000000 bytes and the data gram size is set to 1000 bytes, so 3000 packets are sent by each server to their corresponding clients. The simulations are made to start at the same time. This is

achieved by adjusting the start time and the start window parameters in the client configuration in the DML file.

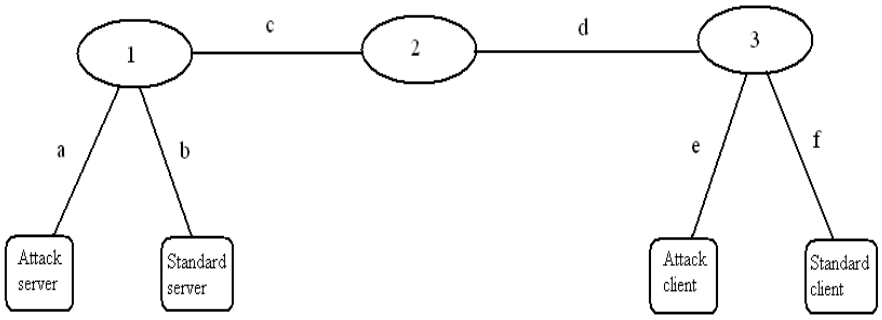


Figure 3.7 Simulation of 3 networks

It is necessary to choose the send interval and the datagram size such that the resulting bandwidth does not exceed the bit rate specified on the server's host link interface. The following parameters need to be changed in the DML script to achieve this.

- The rate at which the server generates data. This is defined by the attribute 'send interval' in the server definition
- The bitrate of the interface of the server which connects to the router of the network. This is also included in the definition of the server.
- The bitrate of the interface of the router which goes to the server. This is included in the definition of the Network.

Table 3.1 shows the amount of data received in bytes at the clients.

Link a	Link b	Data received from standard server (bytes)	Data receive from attack server (bytes)
8000000	8000000	1503000 bytes	6000 bytes
8500000	8000000	798000	711000
9000000	8000000	505000	1004000
10000000	8000000	305000	1204000

Table 3.1 Data received for different bitrates for Configuration 1

The DML script for the above configuration can be found in Appendix A.

3.8.2 Configuration 2

In the second configuration, the servers are placed a hop away from the mincut arc. Figure 3.8 shows the configuration that is simulated.

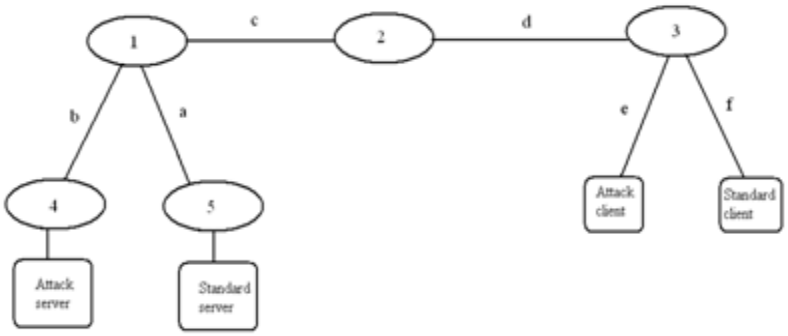


Figure 3.8 Simulation of 5 networks

Link a	Link b	Drops received from attack server (bytes)	Data received from standard server (bytes)
4000000	4000000	9000	3000000
4000000	5000000	613000	2396000
4000000	6000000	1007000	2003000
4000000	8000000	1505000	1505000

Table 3.2 Data received for different bitrates for Configuration 2

From the observations tabulated, it is clear that as the data generation rate of the attack server increases, the number of drops observed for the legitimate traffic also increases. The DML script for Configuration 2 is shown in Appendix B.

3.8.3 Configuration 3

The configuration in figure 3.9 was simulated to understand the working of the OSPF protocol in SSFNet. The static version of the OSPF protocol (sOSPF) in SSFNet uses the hop count as the cost attribute for routing packets from the source to the destination. For all paths having the same cost to the destination, the path having the next hop as the smallest network ID is selected. This is verified by simulating the configuration in Figure 3.9. The packets are routed along path 1 -2 4- 6. The sOSPF protocol in SSFNet does not perform load balancing across equal cost links.

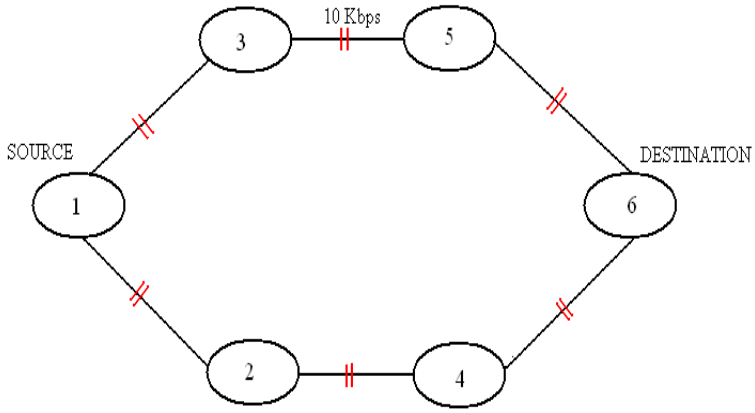


Figure 3.9 Network configuration for OSPF verification

CHAPTER FOUR

DML SCRIPT GENERATION

Our aim is to simulate Distributed Denial of Service attacks for large scale networks. We use the SSFNet simulator for this task, because it is capable of simulating larger networks than its competitors (ex. Ns-2). To run the simulations, network configuration files need to be written using the Domain Modeling Language (DML) in SSFNet.

As the size of the DML script details all the nodes and links in the network, it is impractical to manually generate DML scripts for a number of large networks as the process could be time consuming and error prone. Figure 4.1 shows a configuration of 400 networks.

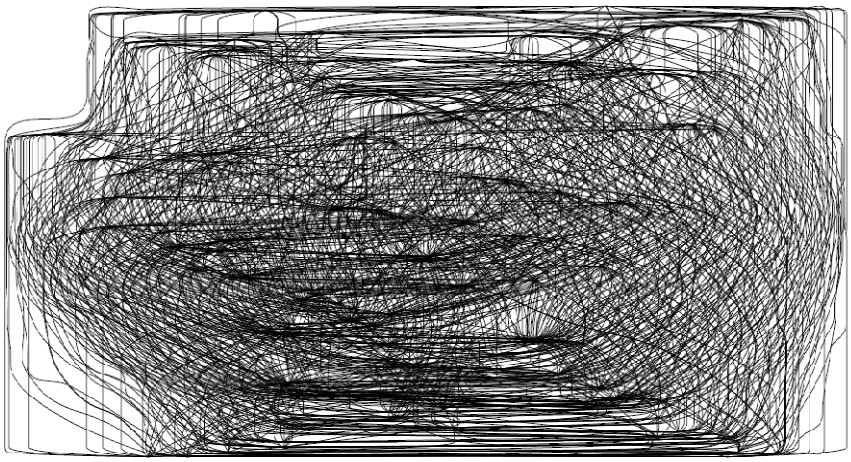


Figure 4.1. Network diagram of 400 nodes.

To generate large scripts that are consistent, we automate the process using Python. Network size is the only input to the script. Python outputs a DML script with the appropriate number of networks. Networks of any size can be generated; the only limiting factor is the processing power of the system which handles the simulation. For this thesis, we created networks of up to 400 sub-networks.

4.1 Graph Theory

Graphs are commonly used to model the structure of the internet for the study of various problems. This section reviews some of the concepts of graph theory.

Graph: A graph is defined as a graphical representation of a network where the hosts are represented as vertices of the graph and the links connecting these hosts are represented by edges of the graph. A graph is traditionally defined as the tuple [V, E] where V is a set of vertices and E is the set of edges. Each edge *e* in defined as (*i, j*) where *i* and *j* represent the two vertices connected by *e.* In the work done in this thesis, we consider graphs that are undirected (where (*i*,*j*) = (*j*,*i*)) and are not multigraphs (where multiple edges connect the same end points)

Connectivity: Two nodes *A* and *B* are connected if the graph contains at least one path from node *A* to node *B* [34].

Source node: A node that is the starting point of a flow is called as a source node or simply a source.

Sink node: A node where the flow terminates is called as the sink node or destination.

Max-flow: In a network graph, the max-flow is the maximum possible flow that one can route from the source to the destination [34].

Min-cut: The min-cut is the smallest set of edges or arcs that are necessary for a source to communicate with a sink. The removal of these edges disconnects the source from the sink.

Connectivity matrix: The connectivity matrix *M* is a square matrix where each element *m(i,j)* is 1(0) if there is (not) an edge connecting the vertices *i* and *j* [35]. In case of undirected graphs, this matrix is symmetric. The diagonal of this matrix could consist of either 0s or 1s depending on the simple assumption that each vertex is connected to itself [35].

Walk: A walk is defined as an ordered list of *z* edges $[(i_0, j_0). (i_1, j_1),\ldots. (i_z, j_z)]$, where each vertex j_a is the same as vertex $i_{a+1.}$

Path: A path of length *z* is a walk where all i_a are unique.

Cycle: If j_z is the same as i_0, the path is a cycle.

A random graph starts as a set of *n* isolated vertices and develops by successively acquiring edges at random. [25] We use this theorem 6.10 from [25] (where *t* is the number of edges in the random graph and *n* the number of nodes):

Theorem 1: The global structure of a typical random graph G_t becomes surprisingly simple as time grows substantially larger than the phase transition time ($t = n/2$): it contains no small components with many edges and all its small components have order $O(\log n)$.

It is important to have a *giant component* for the simulation. The giant component is the largest component of a random graph after the phase change described by Theorem 1. It contains O(n) nodes. The expected number of hops between nodes in these graphs grows proportionally to the log of the number of nodes [36]. If E represents the number of edges in the graph, when $E - n/2 < -n^{2/3}$, the graph is in a subcritical phase and almost certainly not connected. A phase change occurs in the critical phase where $E = n/2 + O(n^{2/3})$ and in the supercritical phase where $E - n/2 > -n^{2/3}$, a single giant component becomes almost certain. When $E = n \log n/2 + O(n)$, the graph is fully connected. [37]

The expected number of edges for a graph is $n(n\text{-}1)p/2$ where p is the uniform probability of an edge existing between any two nodes.

$$\frac{n(n-1)p}{2} = \frac{n}{2} + O(n^{2/3}) \tag{4.1}$$

Thus the probability of an existence of an edge between two nodes at the phase change is given by

$$p \approx \frac{1}{n-1} + O(n^{2/3}) \tag{4.2}$$

Our simulations therefore require p greater than $1/n$ to insure the existence of a giant component.

4.2 Max-flow min-cut

Theorem 2: The max-flow min-cut theorem states that the maximum of all flows is equal to the minimum of all cut capacities. [38]

The concept of max-flow min-cut is illustrated with the help of the example shown in Figure 4.2. If node 1 is the source and node 6 is the destination, then there are two paths from the source to the destination – path A which is 1-2-4-6 and path B which is 1-3-5-6. The maximum flow over path A is bounded by arc 1-2 which has a capacity of 2. The maximum flow over path B is bounded by arc 5-6 which also has a capacity of 2. Since these paths are disjoint the maximum flow from the source to the destination is 6. The removal of arc 1-2 in path A and arc 5-6 in path B completely disconnects the source from the destination. So the mincut is the set of arcs 1-2 and 5-6. [11]

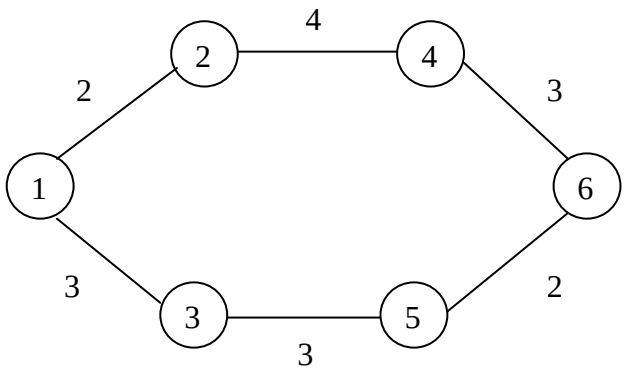


Figure 4.2 Max-flow min-cut for a graph.

4.3 Script Generation Process

We generate a random graph where the probability that an arc exists between nodes *i* and *j* is $p_{i,j}$, which is constant for all *i* and *j*. The most important part in our script generation process is the network connectivity matrix denoted by m_{conn}. $p_{i,j}$ determines how well populated m_{conn} will be and should be varied depending on the network size.

The connectivity matrix is a matrix of all the links that exist between any two networks. It has a form as shown in (4.3).

$$m_{conn} = \begin{bmatrix} 0 & m_{01} & m_{02} & m_{03} & \text{L} & m_{0m} \\ m_{10} & 0 & m_{12} & m_{13} & \text{L} & m_{1m} \\ m_{20} & m_{21} & 0 & m_{23} & \text{L} & m_{2m} \\ m_{30} & m_{31} & m_{32} & 0 & \text{L} & m_{3m} \\ \text{M} & \text{M} & \text{M} & \text{M} & \text{M} & \text{M} \\ m_{m0} & m_{m1} & m_{m2} & m_{m3} & \text{L} & 0 \end{bmatrix} \tag{4.3}$$

The element $m_{conn}[i][j]$ in the connectivity matrix signifies the connection from network *i* to network *j*. A zero element indicates that there is no connection between the networks represented by the element's indices. m_{conn} is a symmetric matrix and since a network need not have a connection to itself, the diagonal elements of the matrix are zero. The non-zero elements of the matrix which are generated with a probability of $p_{i,j}$ have arbitrary values between 6000000 and 7000000. This number specifies the bitrate at which the interfaces between the corresponding networks communicate.

The network IDs have values starting from 1. So, if $m_{conn[1][0]} \neq 0$, there exists a connection between Network 2 and Network 1. Since m_{conn} is a symmetric matrix, elements are generated randomly above the diagonal and the elements below the diagonal have the same values as their corresponding mirror elements. So if

$$m_{conn[1][0]} \neq 0$$
$$then\ m_{conn[0][1]} \neq 0 \tag{4.4}$$

For every network, the description is enclosed in a network loop in the DML script and has a structure as shown in the figure 4.3. There are two hosts, two servers and one router in every network. The following convention has been maintained in every network,

Host 1 is a standard server

Host 2 is an attack server

Host 3 is a standard server

Host 4 is an attack server

All connections between networks are via routers. The router of every network has multiple interfaces and these interfaces are connected via links to other networks. There can be only one link per interface so the number of links originating in or terminating at a network directly determines the number of interfaces required by the router of the network.

The interface details of the router are defined in the router loop which is included in the network loop. The first four interfaces (interface 0/1/2/3) are allocated to the internal hosts of the network. This pattern is followed for all the networks in the script to maintain consistency. The interface details within a network are as shown in the figure 4.3. The interfaces from 4 are free to be assigned to the links between networks.

The interface numbers are assigned sequentially in the connectivity matrix starting from 4 in a column wise fashion. If *N* represents the total number of networks, *m* represents the row number and *n* represents the column number, then the interface number for each non-zero element in m_{conn} is calculated using equation 4.5. The interfaces are generated only for the elements below the diagonal.

$$interface\ number = n * N + m + 4 \tag{4.5}$$

As mentioned earlier, the links in SSFNet are bidirectional so there is only one direct link between any two networks.

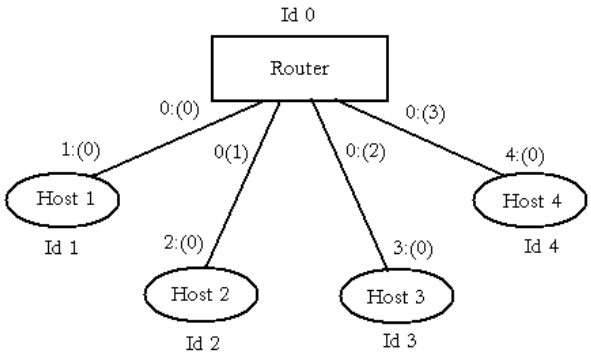


Figure 4.3 Interface details of a network in a simulation

The network loop for every individual network consists of a definition of the router, the hosts in the network and the link which connects the router and the host within the network. The router and the host definitions consist of definition of their own interfaces.

4.3.1 Top level Net

Once all the individual networks are defined, the main Network loop is defined which contains the links which connect the individual networks together. Similar to the interfaces, the links are created only for the elements either above or below the diagonal. The links are defined in the format as in (4.6)

link[attach network_id1:0(interface_no) attach network_id2:0(interface_no) delay 0.0] (4.6)

Here *network_id1* and *network_id2* are the networks between which the link exists.

4.3.2 Traffic

The traffic attribute in the Network loop defines the traffic components that are involved in sending and receiving traffic. The client and the server are defined in the dictionary component of the DML script. When a blue traffic pattern is specified, the standard client directs traffic to a standard server in some other network. Similarly the attack client sends traffic to the attack server when a red traffic pattern is specified. The need for having separate servers in a network arises because of the necessity to adjust the rates of both the servers exclusively.

The dictionary is used to define all the common components that are used throughout the DML script. Once the network is set up, the next phase involves finding two nodes to place the blue nodes.

4.3.3 Determining blue node placement

This is done by the Python script. The blue source and the destination nodes are selected so that no blue destination is less than four hops from the blue source.

Theorem 3: If M = $m_{conn,}$ and if

$$m_{conn}[i][j] \; in \; M^4 + M^3 + M^2 + M \neq 0 \tag{4.7}$$

and

$$m_{conn}[i][j] \; in \; M^3 + M^2 + M = 0 \tag{4.8}$$

then there exists two nodes which are at least four hops away from each other.

Proof: Each non zero element (*j*, *k*) represents the existence of an edge between the nodes *j* and *k.* The result of multiplying M with itself is M^2. Each non-zero element of M^2 except the diagonals represents the existence of a path of length two between the nodes *j* and *k* [35]. By proof of induction, every non zero element of M^3 represents the existence of a path of length three between nodes *j* and *k.* If equation (4.8) is true, then it implies that there exist no paths between nodes *j* and *k* which are connected by three hops or lesser. If equation (4.7) is satisfied, then it implies that the nodes are connected by either

one, two, three or four hops. If both equations are satisfied, then it implies that the nodes are connected by at least four hops.

The chance of finding a combination of blue nodes which are at least four hops away from each other depends on the arc generation probability p_{ij}. For a scenario of 400 networks, p_{ij} is set to 0.03 (since 1/400 < 0.03 according to Theorem 1 in section 4.1) to ensure that there is sufficient connectivity between networks and the required configuration is found. On finding one such configuration, the script stops computing the possible locations for placing the blue nodes. Blue traffic is directed from the blue source to the blue destination and is included in the traffic component of the DML script. If the script fails to find such a combination, then the configuration must be ignored and the program must be rerun till such a situation occurs.

4.4 Mincut Arcs

The non-zero elements are fed as input to the maxflow Matlab program in the format shown in (4.9). The maxflow program determines the mincut set of arcs between two specific network nodes; in this case the two nodes are the blue source and the blue destination.

$$\textit{from\_network} \quad \textit{to\_network} \quad \textit{capacity\_of\_link} \tag{4.9}$$

where *from_network* is the network ID from where the link originates,

to_network is the network ID where the link terminates,

capacity_of_the_link is the capacity of the link between the two networks.

By definition, at least one of the mincut arcs has to be on the four hop path between the blue source and the blue destination. The output of the Matlab code is a set of all mincut arcs between the blue source and the blue destination. Once the mincut arcs are known, the zombie locations can be determined. For a large network, there is a high probability that there will be multiple mincut arcs between the blue source and the blue destination.

The algorithm in [11] determines how many mincut arcs need to be disabled to stop Blue from sending a given volume of traffic. C_{all} denotes the summation of the capacities of all the mincut arcs and the blue traffic is chosen as a random value between $C_{all}/2$ and C_{all}. This is justified using equations (4.10) and (4.11), where if C is the capacity of a mincut arc, then

$$\frac{C-\lambda}{\lambda}<1 \tag{4.10}$$

$$\frac{C}{2}<\lambda<C \tag{4.11}$$

The number of arcs that need to be disabled to send the calculated amount of blue traffic can then be calculated using the algorithm.

4.5 Zombie placement

With an intention of flooding the mincut arcs, zombies are placed near the arc sources. In our simulations, the zombies (Red nodes) are placed a hop away from the

source node. This is accomplished by inspecting the column corresponding to the source of the mincut arc in m_{conn}. The zombie location is selected by randomly choosing one out of the possible zombie locations. For example, as shown in the figure 4.4, the possible zombie locations could be nodes 1, 2, 3 or 4 as all these have outgoing arcs to the mincut source.

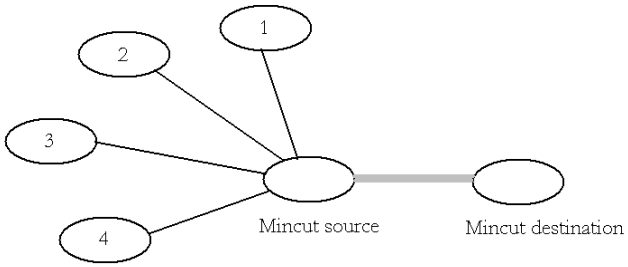


Figure 4.4 Possible zombie locations

The zombie destination is the destination of the mincut arc. The zombie server is placed on the destination. Also, we ensure that no direct connection exists between the zombie source and the zombie destination. Once the zombie source and destination nodes are finalized, they are included in the traffic component as the Red traffic.

This completes the script generation process.

4.6 Selecting one mincut arc at a time

The OSPF protocol in SSFNet does not perform load balancing; it chooses the least cost path to route traffic from the blue source to the blue destination. This is contradictory to the way traffic flows in the Internet. To deal with this problem, we

consider each mincut arc sequentially by disconnecting all but one mincut arc at a time and rerunning the simulation for all the mincut arcs individually.

We start with the mincut arc having the maximum capacity. For a Blue traffic rate of 6000000 bits per second, the Red traffic rate needed to flood the arc is calculated using Equation (2.9). This is the threshold value of the Red traffic denoted by R_{thresh}. Readings are taken by varying the Red traffic rate above and below this value and keeping the Blue traffic rate constant.

This procedure is followed for all of the mincut arcs and the results are recorded.

CHAPTER FIVE

SIMUALTION SCENARIO AND RESULTS

In this chapter, we illustrate the simulations explained in Chapter 4 with an example.

5.1 Simulation Scenario

The scenario consists of 400 nodes. The network graph is generated by setting p_{ij} in the Python script. As mentioned in Chapter 4, Blue players are placed on two nodes which are at least four hops away. The zombie locations are calculated after knowing the details of the mincut arcs in the network. Figure 5.1 details the network configuration obtained. Nodes 10 and 159 are chosen as blue node locations.

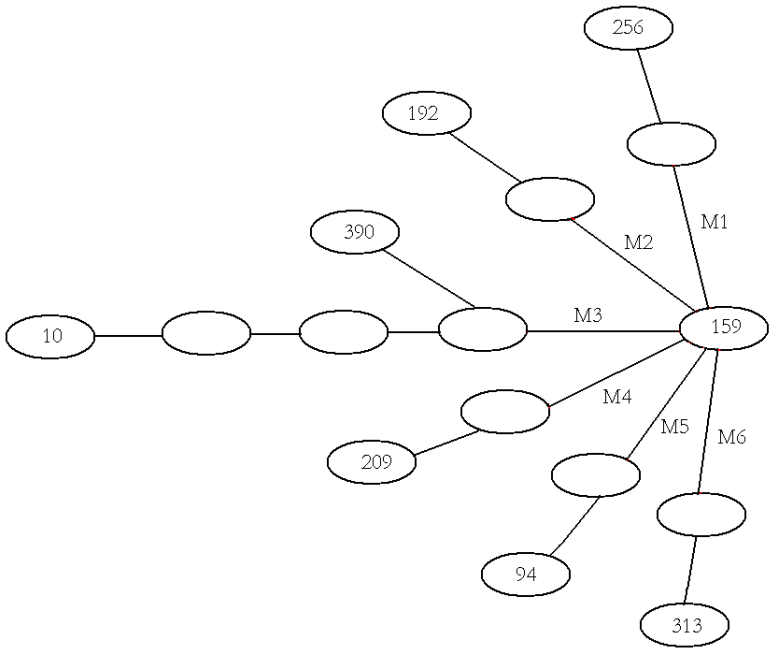


Figure 5.1: 400 Node configuration

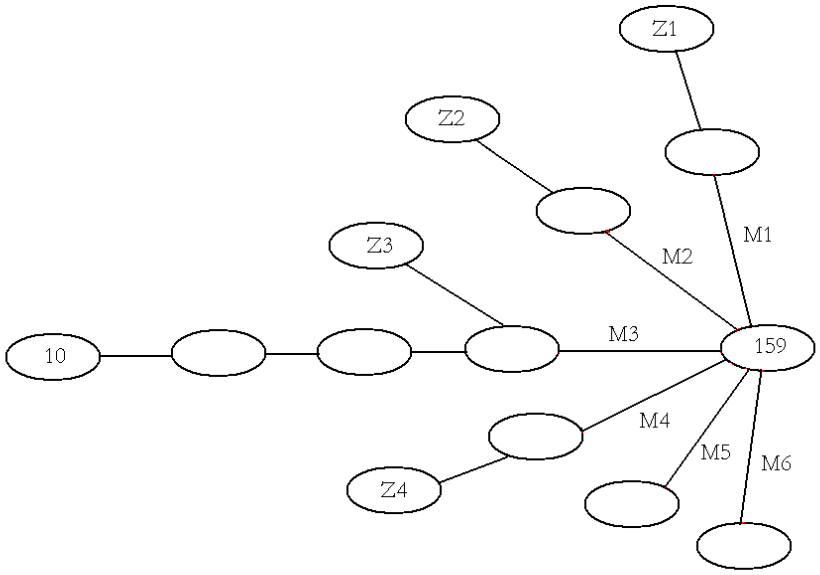


Figure 5.2: Node configuration details of the Network

The node configuration details are shown in the Figure 5.2. Blue traffic flows from node 10 (B1) to node 159 (B2). There are six mincut arcs between the blue source and the blue destination (M1 to M6). The mincut arcs are centered on the destination. According to the algorithm detailed in [11] four out of the six mincut arcs need to be disabled. The zombies are placed on Z1, Z2, Z3 and Z4 corresponding to the four mincut arcs. The Red traffic flows from the zombie source to the zombie destination which is the same as the mincut arc destination in this case. The amount of data requested by the Blue and the Red clients is the same.

We first target the mincut arc with the maximum capacity. In our case, this is link M1, between nodes 256 – 159. The link has a capacity of 6916382 bits per second. The zombie is placed on node 377. The configuration is as shown in Figure 5.3. The attack traffic flows from the red server (node 159) to the red client (node 377) whereas the legitimate traffic flows from the blue server (node 159) to the blue client (node 10).

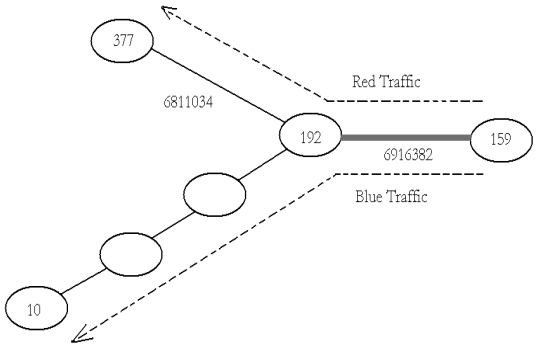
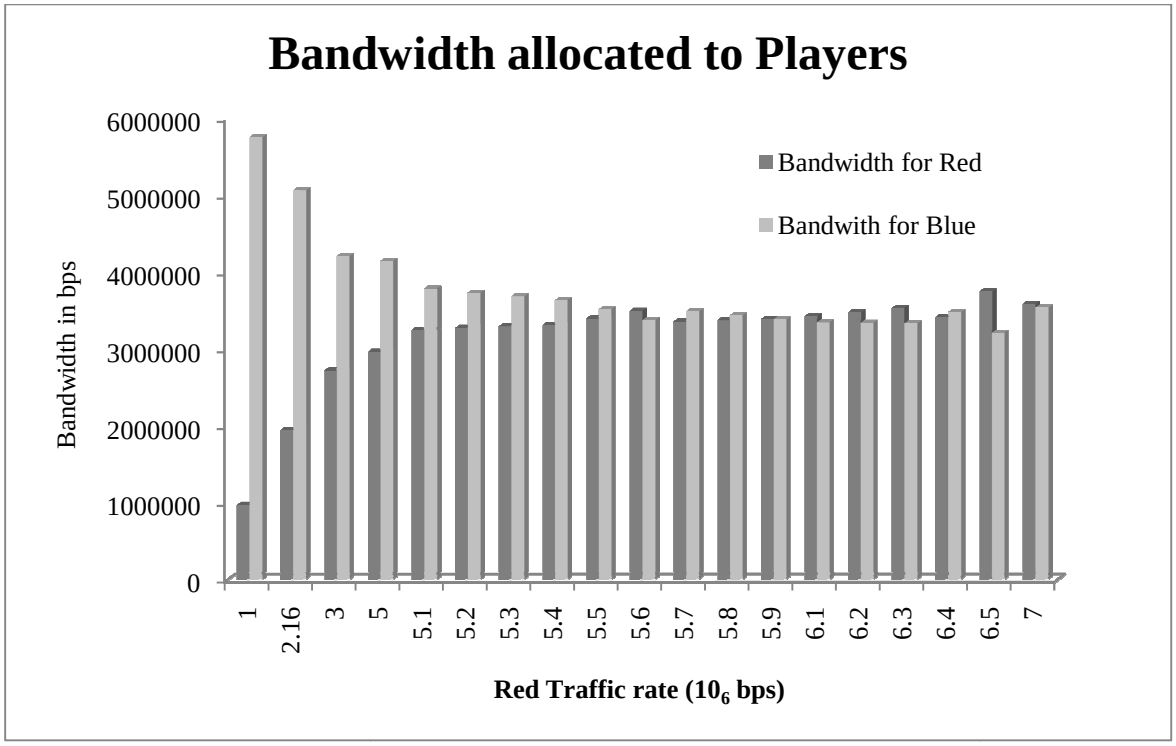


Figure 5.3: Mincut Arc 1

The red traffic rate threshold (R_{thresh}) is calculated for a constant blue traffic rate of 6000000 bps using the Equation (2.9). The red traffic rate is increased from 500000 bps to 7000000 bps and the effective bandwidth allocated to Blue and Red is noted down. On increasing the red traffic value above the threshold value, a significant reduction in blue bandwidth is observed. This is shown in Graph 5.1. Thus by increasing the rate at which an attacker generates data, he can limit the bandwidth allocated to the legitimate source and effectively cause a DDoS attack.



Graph 5.1: Bandwidth allocated to players with increase in Red Traffic rate

The procedure is followed for all three remaining mincut arcs. Figures 5.4, 5.5 and 5.6 detail the configuration of the arcs.

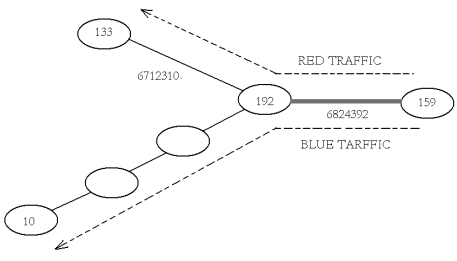


Figure 5.4: Mincut Arc 2

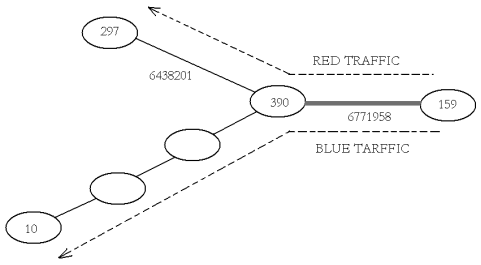


Figure 5.5: Mincut Arc 3

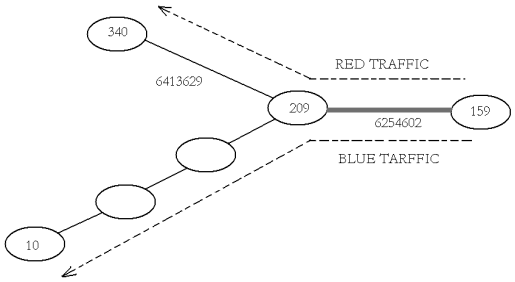
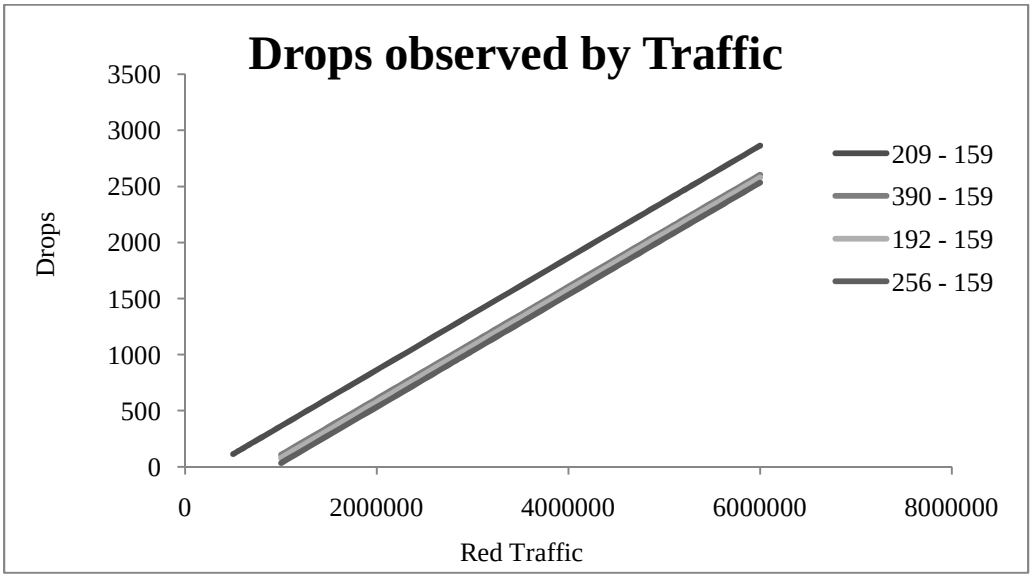


Figure 5.6: Mincut Arc 4

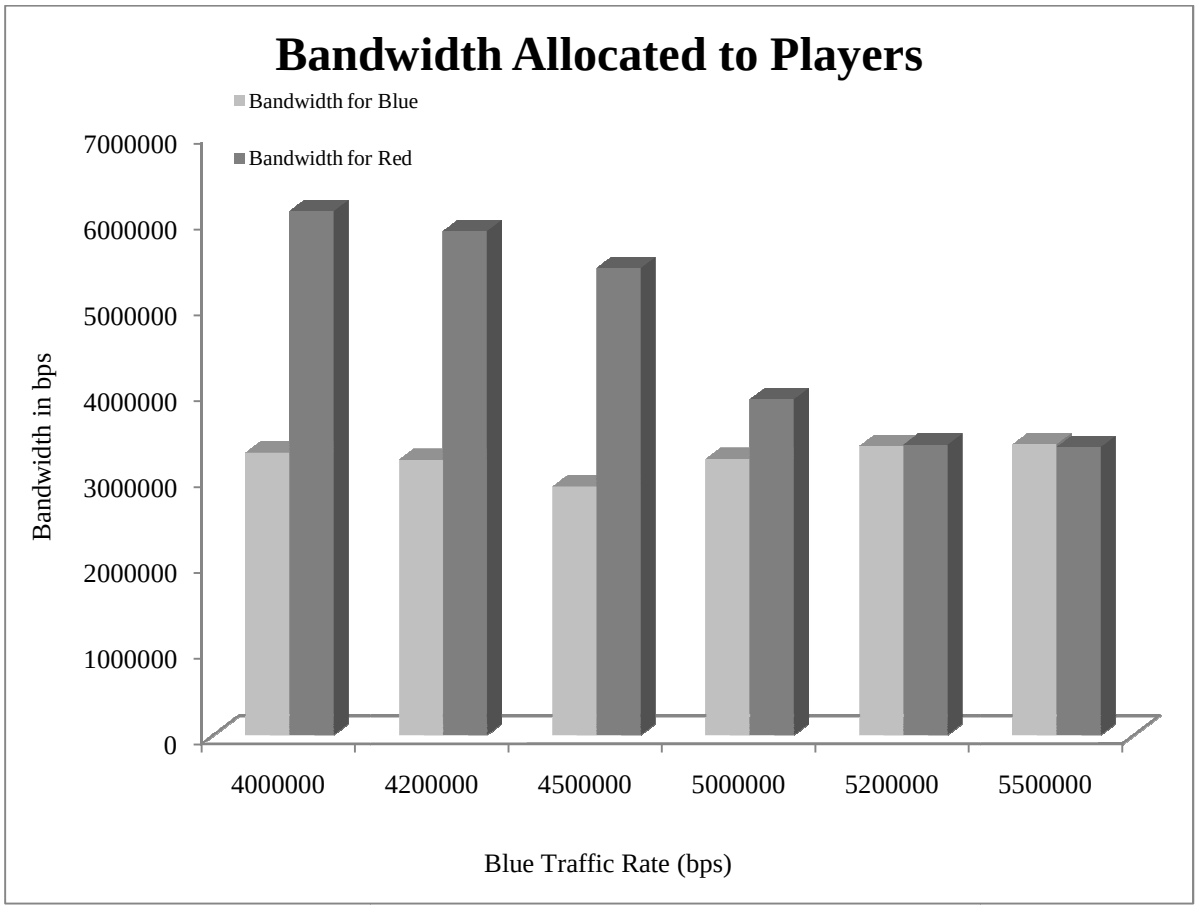
The graphs for these configurations have similar results. To prevent repetition, they have not been included in this thesis.

Graph 5.2 shows the drops observed by the total traffic (Red and Blue) over all the four mincut arcs. It can be seen that the arc with the maximum capacity observes the least number of drops. Also as the red traffic rate increases, the number of drops increase linearly which is in line with our understanding.



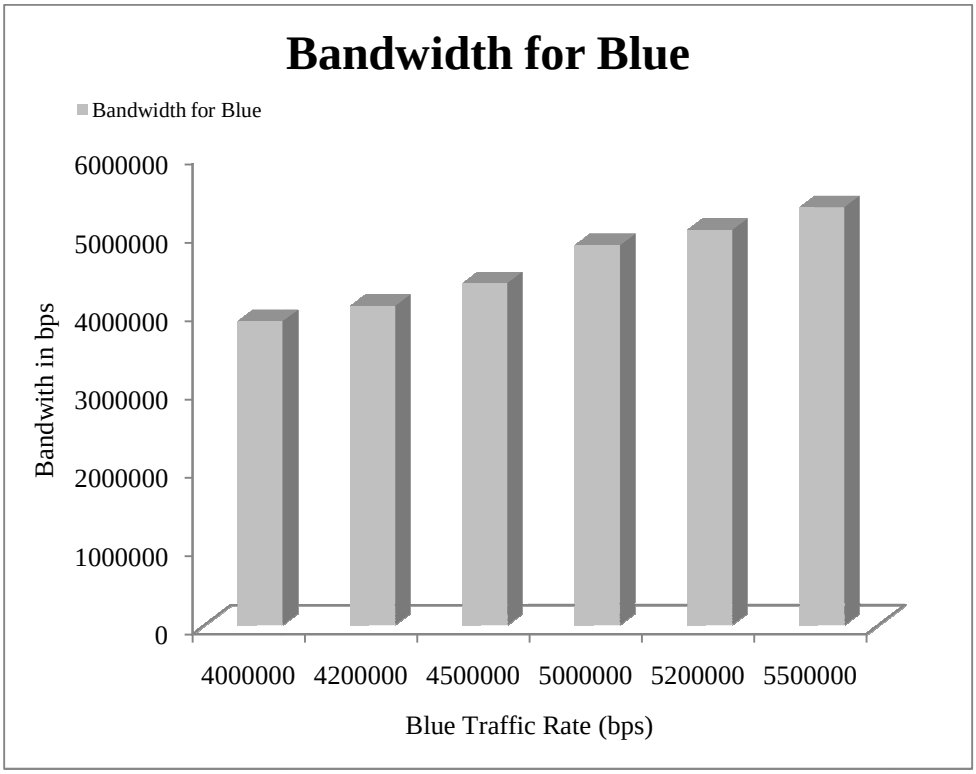
Graph 5.2: Drops observed by the traffic for all mincut arcs

To check the effectiveness of the DDoS attack when the legitimate source increases the amount of traffic we increase the blue traffic rate from 4000000 bps to 5500000 bps. It should be noted that the blue traffic rate has to be greater than half the capacity of the mincut arc. The red traffic rate was set to the value obtained from the formula. It can be observed that the blue bandwidth is limited to a particular value and in spite of the increase in the blue traffic rate, the blue bandwidth does not increase. This is shown in Graph 5.3.



Graph 5.3: Bandwidth allocated to Players with increase in Blue traffic rate

In absence of red traffic, the effective bandwidth occupied by Blue approaches the rate at which the Blue server generates data. This can be verified from graph 5.4. The network overhead contributes some losses and hence the bandwidth occupied is not equal to the blue traffic rate.



Graph 5.4: Effective Bandwidth of Blue in absence of Red traffic

The equation is pessimistic and gives an upper bound on the amount of attack traffic that is required to cause a DDoS attack. It is difficult to estimate a lower bound on the amount of attack traffic needed as it is dependent on the underlying network hardware and software and is thus difficult to estimate.

Similar simulations were performed for two other configurations which were generated randomly by Python. The equation was found to be pessimistic for these configurations as well.

CHAPTER SIX

DISTRIBUTED DENIAL OF SERVICE COUNTERMEASURES

This chapter presents some of the mechanisms provided to defend against Distributed Denial of Service Attacks.

6.1 Defense Mechanisms

DDoS defense may be regarded as a resource allocation problem in which the server resources are fairly allocated to clients to prevent attackers from consuming an excessive amount of resources. DDoS attacks can also be thwarted by filtering or rate limiting attack packets. An attack detection module is used to extract the characteristics of the attack packets and once the characteristics have been summarized, packet filtering modules are used to filter malicious packets. [40]

Some detection techniques use attack source traceback and identification as a response to a DDoS attack. The routers record information about the packets they have seen for later traceback requests or they send additional information about the packets they have seen to the packet's destination. However, traceback is ineffective in DDoS attacks in which the attack traffic comes from legitimate sources. [41]

Activity profiling monitors the average packet rate for a network flow, which consists of consecutive packets with similar packet fields. The total network can be

measured as the sum over the average packets rates of all inbound and outbound flows. An attack can be detected if an increase is observed in the network flows. [42]

In the backscatter analysis, the researchers monitor a wide IP address space for incoming backscatter packets. The backscatter packet's source address is that of the victim, but the packet's destination address is randomly spoofed. An attack that uses uniformly distributed address spoofing leads to a finite probability that any monitored address space will receive backscatter packets. The packets are clustered based on the unique victim source address. To detect attacks, the researchers analyze a cluster's destination address distribution uniformity. [42]

The authors in [26] classify the DDoS defense mechanisms as being reactive and preventive. In reactive measures, the attack sources are identified as early as possible and are prevented from executing further attacks. The countermeasures here may be attack specific, when the attack is consuming fewer resources than available. The preventive measures focus on eliminating the possibility of performing a DDoS attack. This mechanism is not 100 % effective but does ensures a decrease in the frequency and strength of DDoS attacks by making a host resilient to the attacks which includes identifying loopholes in the system and eliminating the vulnerabilities or removing application bugs to prevent intrusions. [6].

6.2 Differentiating between Flash events and DDoS

Before concluding that a denial of service attack is under progress, it is necessary to identify and separate DoS attacks from flash events. If any attempt to undermine a website is considered to be a Denial of Service attack, then the preventive techniques might end up throttling the excess legitimate traffic. Work has been done in the past in this field. A flash event is defined as a sudden increase in traffic for a particular website. This results in a dramatic increase in server load putting severe strain on the network links leading to the server. The end result is considerable increase in packet loss and congestion.

6.3 Summary of work done

The authors in [11] set up a game between the attacker and multiple distributed applications of an enterprise. The attacker might not have sufficient resources to disrupt all the processes of an enterprise. It will try to maximize the number of processes it can disable. In reaction, an enterprise can shift to another configuration that has not been attacked. Both the players have to determine the best process to make a move in. Since the problem is P-Space complete, the authors analyze it using Combinatorial Game Theory and Thermographs. Reconfiguration strategies are provided for distributed applications using Thermostrat strategy.

The work presents an example which consists of 3 distributed applications. The values of the distributed applications are known *a priori*. For determining the process in

which a move has to be made, the authors make use of the Thermostrat strategy. The strategy is explained in detail in Chapter 7. Figure 6.1 shows the three distributed applications.

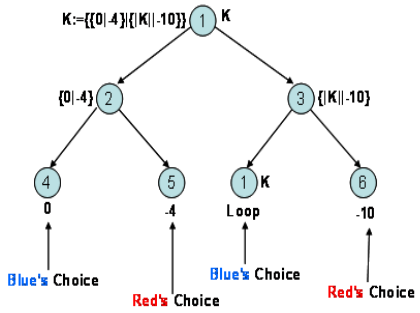


Figure 6.1(a) Configuration 1

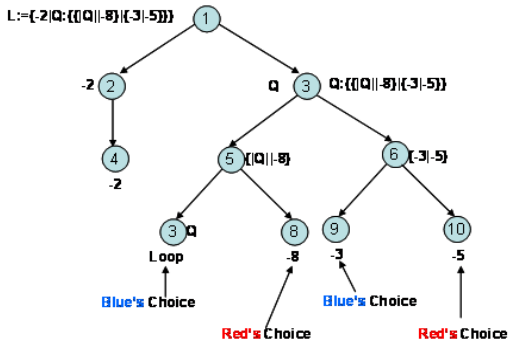


Figure 6.1(b) Configuration 2

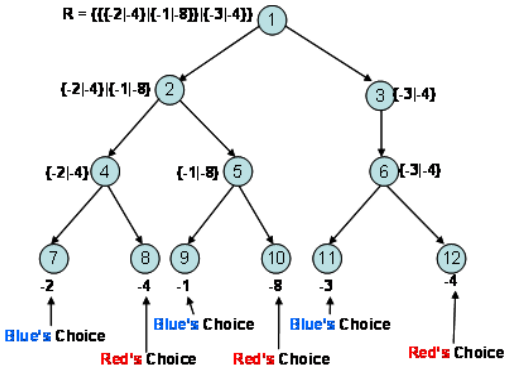


Figure 6.1(c) Configuration 3

CHAPTER SEVEN

BANDWIDTH LIMITED COORDINATION OF GAMES

We now consider what happens when an enterprise (Blue) attempts to maintain operations in spite of an adversary (Red) launching a DDoS attack. Blue has a set of networks it needs to coordinate. For example, it may have accounting, finance, administration, research and development, and manufacturing systems that must remain operational while competing for the same scarce resources. Blue has to maintain communications for these separate networks over bandwidth limited links. The aim of Red is to disrupt Blue's communications. Due to limited bandwidth availability over the links, Blue needs to coordinate among its networks by sending only the most important information. Blue will be successful in achieving this depending on whether it is able to find the most important message that needs to be transmitted. It would be ideal if this could be done without requiring out-of-band coordination messages.

We model this problem using combinatorial game theory. We first explain some of the basic terms required to understand the concepts of game theory.

7.1 Surreal Numbers

The authors in [27] define surreal numbers as an extension of real numbers with a tangible concept of infinity and infinitesimals. They describe surreal numbers as a pair of sets (Left and Right) of previously created surreal numbers such that no member of the right set maybe less than or equal to any member of the left set. By definition Left wins

the game if the final score is greater than zero, or if the final score equals zero and it is Right's turn to play when the game ends [28]. If every element of the left set is not less than every element of the Right set, then it results in an ill-formed surreal number, also called as a game. Every surreal number is a game, but not all games are surreal numbers.

A combinatorial game involves two players – Left and Right. A game tree has a root node which represents the initial position. The root node has zero or more branches going downwards to the left (representing moves for the left player) and downwards to the right (representing moves for the right player). At each point, the player considers the options he has and chooses the one which will maximize his payoff value. Game trees can be typically represented as shown in equation (7.1).

$$\{L_1, L_2, \ldots\ldots L_n \mid R_1, R_2, \ldots\ldots R_m\} \tag{7.1}$$

The options for left are represented as $L_1 \ldots\ L_n$ and the options for right are options from $R_1 \ldots\ R_m$. The equation has a numeric value if

$$\forall L_i \forall R_j : L_i < R_j \tag{7.2}$$

The value of a surreal number where equation (7.2) holds is the "simplest" number between the greatest L value (Lmax) and the smallest R value (Rmin) [29]. If equation (7.2) is not satisfied, then the number is ill formed and it is a game.

$$\exists L_i \exists R_j : L_i \geq R_j \tag{7.3}$$

The value of the game then depends on the sequence of moves taken. Figure 7.1 shows a diagram of a game tree which can be represented by equation (7.4)

$$G = \{15, \{25 \mid 10\} \mid -5\} \tag{7.4}$$

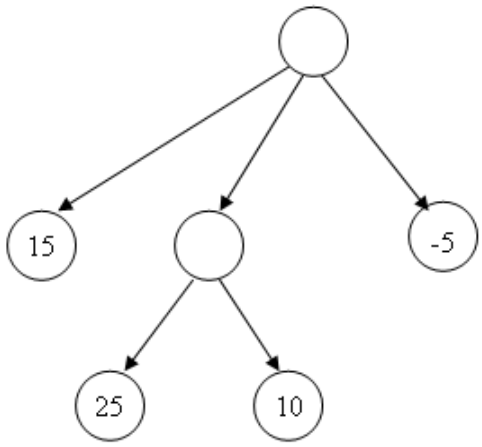


Figure 7.1 Game tree represented by G

If Right plays first, then he has only one option to move and he ends up gaining 5 points from the Left player. If Left makes the first move, then he has two options which are represented by two branches going down leftwards. He can either choose the first option and gain 15 points from Right or choose the second option and move to the game $\{25 \mid 10\}$. If he chooses the second option, then Right plays next and gives the Left player 10 points. The Left player would prefer winning 25 points to 10 and hence if Left plays first, he would choose the first option.

7.2 Combinatorial Game Theory

A combinatorial game involves two players – Left and Right. These are perfect information games in which all players know all the moves that have taken place. Combinatorial game theory does not study games of chance. In our example scenario, there are multiple networks which want to coordinate and communicate over the network links. The Blue player needs to prioritize data and send the most important information. This effectively translates into a Sum of Games problem, where the Blue player is engaged in multiple games with Red and the aim is to maximize the overall payoff function. This sum of games is represented by

$$G_i = \sum_{j=1}^{n} G_{i,j} \tag{7.5}$$

Yedwab proved the following theorems in [28] which state that

Theorem 1: Calculating the value of the Sum of Games is NP-hard.

Theorem 2: Finding the optimal sequence of moves for a Sum of Games problem is PSPACE complete.

These theorems state that a truly optimal strategy for a sum of games is only found by an exhaustive search of alternatives which requires exponential time. Instead of finding the best possible solution, it is possible to find a solution within a constant offset of the optimal.

Mathematical studies have been carried out using game trees to analyze the strategies used for playing games and winning them. We introduce a concept called thermographs which could be used for chilling the games and finding the optimal strategies for the sum of games. In order to understand thermographs, we first explain the concept of 'temperature' of a game.

The temperature of a game signifies the variability of the game. It signifies the amount that stands to be gained by either player initiating a move. A game where a much (little) stands to be gained or lost is called as hot (cold). The variability of a game can be reduced if a tax *t* is imposed for making a move. This is also called as process of cooling. It is done by modifying the game.

$$G_t = \{G_t^L - t \mid G_t^R + t\} \tag{7.6}$$

We use the concept of thermographs in calculating the value of a game. Thermographs are plotted on graphs in which the co-ordinate system used has the tax on the y-axis and the game value on the x-axis. The values on the x-axis are plotted in decreasing order to keep the Left player's options to the left side of the graph. As tax *t* increases, both sides reach a common value which is called as the 'mean value' of the game. The smallest tax needed to reach the game's mean value is called as the temperature of the game.

7.3 Plotting thermographs

The authors in [30] explain the procedure for plotting a thermograph. They start with Left and Right's choices and recurse upwards. For example, Figure 7.2 shows the thermograph of {{5 | -5} | -20}. The thermograph of {5 | -5} is first plotted by marking the Left and Right choices for t=0 on the horizontal axis and then plotting the game values as t increases until the Left and Right values converge [12]. Since the value on the right is already a number (-20), its thermograph is just a vertical mast.

The next step is to plot the thermograph of {{5 | -5} | -20} using the thermograph of {5 | -5}. After Left has moved to {5 | -5} it will be Right's turn so -5 is the starting point on the left. The temperature of the freezing point of {5|-5} is 5. So the left edge of the thermograph starts at point (-5, 5).

The game -20 has value -20 and freezing point t = 0. So the right edge of the thermograph starts at point (-20, 0). We recursively subtract a tax t from the left and add it to the right, until the two values converge. As shown in Figure 7.3, this gives us the freezing point (temperature) of 10 and a mean value of -10.

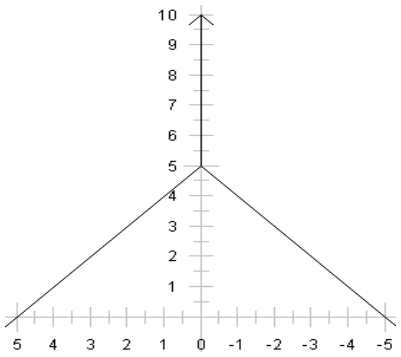


Figure 7.2: Plot of {5|-5}

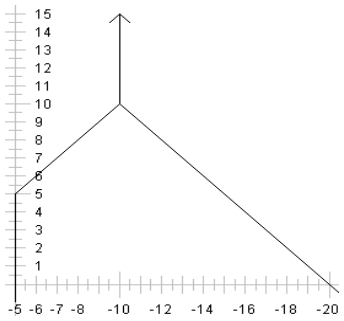


Figure 7.3: Plot of {{5|-5}|-20}

7.4 Berlekamp's Strategies

Choosing a strategy to play the sum of games problem would help to make a decision. In [12], Berlekamp presented three strategies for deciding which game to play in.

Sentestrat: This strategy tells us to respond to the opponent's move by making a move in the same game. This strategy is not of any importance for the work in this thesis, as we do not have an idea of where the opponent is.

Thermostrat: In this strategy, by plotting the friendly side and the enemy side of the sum of games, we find the component game whose thermograph has the maximum width at different temperatures. The temperature at which the widest component occurs is called the ambient temperature. According to the Thermostrat strategy, the component game widest at the ambient temperature is the game that needs to be played in.

For example, in figure 7.4, there are three games being played simultaneously. G1 = {{15|5}|{4|3}}, G2 = {30|20}, G3 = {{50|45}|75/2}. The thermographs for the games are shown from left to right. The left hand side is the sum of the left hand sides of the G1, G2, and G3 thermographs; that is 80 = 45 + 30 + 5. The right hand side is found by subtracting the maximum width of the three thermographs at each temperature. We note that the furthest right point of this graphic has value 69, which occurs at temperature 3/2. Since the thermograph of G2 has the maximum width at this temperature, Thermostrat advises to play in G2.

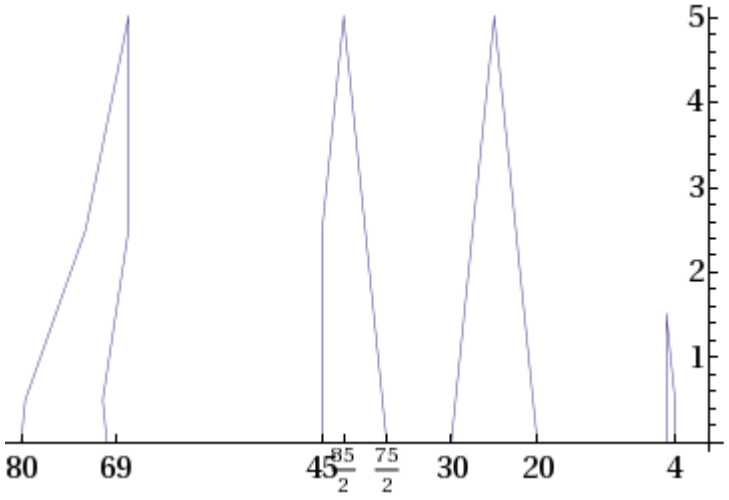


Figure 7.4: Thermostrat strategy example. From right, Thermograph of G1, thermograph of G2, thermograph of G3, and thermograph evaluation of the sum of these three games

In our work, we are playing in a game with one of the components masked. In such cases, the unmasked game might have extreme values or a single surreal number which could

drastically change the width and accordingly the decisions of playing in a game. Thus the Thermostrat strategy fails to be applicable in the work pertinent to this thesis.

Hotstrat: Hotstrat strategy recommends play in a game which has the highest temperature. In other words, the Hotstrat strategy when applied to a sum of games problem would choose a game with the highest variability. Since the variability directly relates to the payoff values, this strategy correctly reflects the most important component game.

The Hotstrat strategy [30] when applied to a sum of games problem would choose a game with the highest temperature and will correspondingly choose that game. Since the temperature of the game signifies the importance and variability of the game, the higher the variability of the game, the higher is the payoff that can be obtained by playing that game. In this example scenario, the result would give us the most important message that needs to be transmitted. This ensures that communication is maintained till the affected links are restored back to their normal state.

7.5 Example Game

We have multiple departments which need to coordinate in order to maintain communication. If the links between the departments experience a DDoS attack, there would be a heavy constraint on the bandwidth that can be assigned to the players. This limits the number of messages that could be transmitted. At such times, it would be of paramount importance to prioritize the messages that need to be sent. We model this as a

two player game with the two players being Red and Blue. The aim of the Blue player is to maintain communication within its departments and the aim of the Red player is to try and disrupt it. If the attacker is able to disrupt the communication between the departments then the Red player wins.

Multiple departments share a limited communications channel and more than one department can simultaneously detect changes in the network. The player needs to prioritize messages before deciding which is the most important. This problem was solved by Virtenen in [31] by considering Maximax, Maximin and central values prioritization schemes. We modify the problem to compare game trees instead of comparing range of values. We represent our set of messages as different branches of a game tree with payoff values assigned to each branch. In essence, *m* departments are simultaneously deciding which of the *n* attackers to engage (one attacker might target multiple links). Thus the message prioritization problem is changed from a team decision problem to a Sum of Games problem from combinatorial game theory.

7.6 Playing in a game with one of the options masked

Since more than one department can simultaneously detect changes in the network, a subset of the game changes, however because of bandwidth limitations, the players can not accurately know the details of all the games in the set. They need to choose the games which are more important. So the players end up playing in a sum of games problem where they are ignorant about the payoffs in a subset of the games.

7.7 Incorporating chance moves

In order to deal with combinatorial game theory, we need to modify Conway's surreal number approach to include chance moves. Surreal number representations of the game assume perfect information. Unfortunately, the underlying nature of this problem is probabilistic in nature. Figure 7.5 shows an extensive form representation of a chance move. Extensive form is a tree structure with each interior node of the tree representing a decision point. Leaves are associated with payoffs. At the root node, Blue wants to maximize the payoff. If Blue chooses the alternative on the left, two choices exist on the left with probability 0.4 and right with probability 0.6. After those chance moves are nodes that represent Red's choices. Since Red wants to minimize, the left (right) node has value 5 (14). We state a theorem which helps us solve this problem.

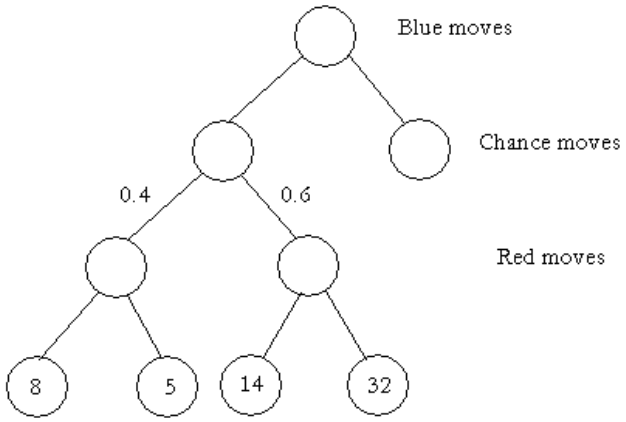


Figure 7.5 Extensive form representation of a chance move

Theorem 3. Given a probability distribution function $\{p_1, p_2, \ldots p_i\}$ where p_k is the probability of surreal number $\{L_k \mid R_k\}$, the expected value is a surreal number.

Since addition and multiplication of well formed surreal numbers is a surreal number [27], for all elements in a game tree,

$$p_k * \textit{surreal number} = \textit{surreal number} \tag{7.7}$$

Blue uses the expected value of its left node 0.4 * 5 + 0.6 * 14 = 10.4 as its expected payoff in calculating which alternative to take. It can also be viewed as compressing the two Red moves into a single information set where Blue cannot know in advance which node in the information set it chooses. In extensive form, each player's possible moves are expressed in alternation with chance moves inserted as necessary. By replacing chance moves in a game tree, we convert an imperfect information game into an equivalent perfect knowledge game. The next section talks about the algorithm to prioritize the messages.

7.8 Message Prioritization Algorithm

1. Each network monitors the state of its links.
2. Each network constructs game trees based on the monitored data
3. Surreal numbers are constructed for each engagement
4. Thermographs are constructed from each surreal number and the freezing point is noted
5. The data is prioritized using the temperature of its associated surreal number
6. An alarm is set proportional to the inverse of the temperature.

7. As soon as the alarm expires, if the bandwidth is not occupied then data is transmitted.

7.9 Simulation of Game Scenario

The simulation game scenario developed using Python helps explain the example game. We also show that the Hotstrat strategy dominates the strategies used in [31] for game theory problems.

Both the players – Red and Blue start the game with a common operating view. The common operating view is a set of three randomly generated games (G1, G2, G3). The two players compete by playing a sum of games problem on this set. On monitoring their links, the players determine that the games G2 and G3 are replaced by games G4 and G5. Since the players have bandwidth enough to transmit information of only one game, they have to choose between game G4 and G5. The decision about which player makes the first move is made randomly with both players having an equal probability. Both players are now playing a sum of games problem which consists of G1 G4 and G5. However, the players are forced to choose between the information sets of {G1, G2, G5} and {G1, G4, G3}. The players choose a strategy from Maximax, Maximin, Central values and Hotstrat to help them decide which game to play in and make a move in that game. This procedure is followed until payoff values are obtained for all three games. The payoff values corresponding to the real scenario are summed to give the payoff for the sum of games. If the sum is greater than or equal to half the maximum payoff possible, then Blue player wins, else Red wins.

The five randomly generated games are

$$G_1 = \{\{12|32\} \mid \{33|32\} \parallel \{48|9\}|\{43|19\}\}$$

$$G_2 = \{\{4|9\} \mid \{49|34\} \parallel \{24|1\}|\{9|4\}\}$$

$$G_3 = \{\{6|37\} \mid \{49|2\} \parallel \{10|34\}|\{21|18\}\}$$

$$G_4 = \{\{10|2\} \mid \{4|45\} \parallel \{32|39\}|\{32|34\}\}$$

$$G_5 = \{\{15|5\} \mid \{14|43\} \parallel \{13|27\}|\{3|27\}\}$$

The steps taken to choose the game which it prefers to see are listed in Table 7.1. The following conventions are used to describe the simulation example.

S: Set of games which the player sees and applies the strategy to.

x: Strategy chosen

I_g: Game which is inconsistent with the real scenario

c_g: Game chosen to be modified after applying *x*

Time step	Action performed by Blue	Action performed by Red
1	$S = \{ G_1, G_2, G_3\}$ x = Maximax	$S = \{ G_1, G_2, G_3\}$ x = Maximax
2	Apply x to G_4 and G_5 $I_g = G_5$ $S = \{ G_1, G_4, G_3\}$	Apply x to G_4 and G_5 $I_g = G_5$ $S = \{ G_1, G_4, G_3\}$

Table 7.1: Procedure to decide the starting scenario

Table 7.2 details the steps followed after choosing the inconsistent game. In our simulation run, Blue player starts the game.

Time step	In action	Scenario before S	Action	Scenario after S
1	Blue	G_1 = {{12|32}|{33|32}|| {48|9}|{43|19}} G_4 = {{10|2}|{4|45}|| {32|39}|{32|34}} G_3 = {{6|37}|{49|2}|| {10|34}|{21|18}}	Apply x to S $c_g = G_3$	G_1 = {{12|32}|{33|32}|| {48|9}|{43|19}} G_4 = {{10|2}|{4|45}|| {32|39}|{32|34}} G_3 = {6|37}|{49|2}

2	Red	G_1 = {{12|32}|{33|32}|| {48|9}|{43|19}} G_4 = {{10|2}|{4|45}|| {32|39}|{32|34}} G_3 = {6|37}|{49|2}	Apply *x* to *S* $c_g = G_4$	G_1 = {{12|32}|{33|32}|| {48|9}|{43|19}} G_4 = {32|39}|{32|34} G_3 = {6|37}|{49|2}
3	Blue	G_1 = {{12|32}|{33|32}|| {48|9}|{43|19}} G_4 = {32|39}|{32|34} G_3 = {6|37}|{49|2}	Apply *x* to *S* $c_g = G_3$	G_1 = {{12|32}|{33|32}|| {48|9}|{43|19}} G_4 = {32|39}|{32|34} G_3 = {6|37}
4	Red	G_1= {{12|32}|{33|32}|| {48|9}|{43|19}} G_4 = {32|39}|{32|34} G_3= {6|37}	Apply *x* to *S* $c_g = G_3$	G_1= {{12|32}|{33|32}|| {48|9}|{43|19}} G_4 = {32|39}|{32|34} G_3= 37

Table 7.2: Steps to play the game

At time step 4, a final payoff value is obtained for G_3. The procedure is continued until payoff values are obtained for all the games. Since the payoff values corresponding to the

real scenario are considered, the payoff value in this case is the summation of the payoff values for games G_1 G_4 and G_5 which is 69 (32 for G1 + 32 for G4 + 5 for G5).

The simulations are run 500 times for each pair of strategies. The percentage wins for Blue player are recorded. The rows represent the strategies chosen by Blue and the columns represent to strategies chosen by Red.

	Maximax	Maximin	Central Values	Hotstrat
Maximax	0.53	0.518	0.55	0.27
Maximin	0.492	0.548	0.488	0.354
Central Values	0.564	0.538	0.534	0.362
Hotstrat	0.75	0.71	0.718	0.542

Table 7.3: Recorded percentage wins for Blue

The test for statistical significance between binomial distributions [7] is used to verify that the values in Table 7.3 are significantly different.

$$\log\left(\frac{p_1(1-p_2)}{p_2(1-p_1)}\right) < 0.41 \tag{7.8}$$

Row wise and column wise comparisons are performed to determine the most optimal strategy for Red and Blue.

For a Red strategy, the most optimal strategy for Blue can be determined by comparing values within columns. This is shown in Table 7.4. Sub columns are created within

columns to show which strategies are being compared. We note that Hotstrat's performance is significantly better than the other three, no matter which strategy was chosen by Red. So Hotstrat is marked as a + and the others are marked as -. When there is no significant difference between the strategies, then they are marked as ≈.

On comparing within rows, we obtain the strategy that performs best for Red against a given Blue strategy. Similarly, Hotstrat causes Blue to win fewer games than other strategies. This is shown in Table 7.5. Thus the Hotstrat provides an effective strategy for determining the priority of the games when competing for bandwidth.

	Maximax		Maximin		Central Values		Hotstrat			
Maximax	-	≈	-	≈	-	≈	-		-	≈
Maximin	-	≈	-	≈	-	≈	-	≈		≈
Central Values	-	≈	-	≈	-	≈	-	≈	+	
Hotstrat	+		+		+		+			

Table 7.4: Choosing an optimal strategy for Blue

	Maximax	Maximin	Central Values	Hotstrat
Maximax	+	+	+	-
	≈	≈	≈	
Maximin	+	+	+	-
	≈	≈	≈	
Central Values	+	+	+	-
	≈	≈	≈	
Hotstrat	+	+	+	-
	≈	≈	≈	

Table 7.5: Choosing an optimal strategy for Red

CHAPTER EIGHT

SUMMARY

We verified, by performing simulations, the work in [11] to quantify the number of resources that an attacker would need to disable a network. Performing a DDoS on a large scale network is more reasonable than a DDoS on a small scale network. We choose the SSFNet simulator over its competitors as it is capable of handling large networks. To simplify the tedious and error prone process of writing script for large networks, we automate the network generation process.

The formula derived in [11] is developed for an ideal network. It does not account for processing and the overhead contributed by the network. Since the network simulator is not really the actual network, the statistics obtained give an upper bound of the amount of attack traffic required to cause a DDoS. It is slightly conservative in quantifying the zombie traffic. The lower bound is dependent on the underlying network implementation and we suspect that a better estimate would have to be empirical. However, considering the fact that we cannot ethically perform a DDoS on a functioning network, it is unlikely that further empirical work can be done.

In Chapter 7, we develop an alternative application of combinatorial game theory in which we allocate bandwidth between processes. We present an example scenario by setting up a game between an attacker and multiple distributed applications of an enterprise. The enterprise coordinates between its different networks by maintaining communication over bandwidth communication links. The limited bandwidth links make

it necessary to determine the most important message that an enterprise needs to transmit. In order to account for the probabilistic nature of the problem, we convert a game with imperfect information into perfect information games. We compare four strategies – Maximin, Maximax, Central value and Hotstrat to determine the priority of the messages and conclude that Hotstrat gives us the best possible results. We verify our understanding by running simulations. The results indicate that our proposed technique will be part of an effective DDoS countermeasure.

Further research can be focused on

1. Introducing background traffic
2. Simulating with a protocol that performs load balancing and more closely simulates the working of the Internet.
3. Implementing a prototype of the Bandwidth Limited coordination of games

APPENDIX A – DML SCRIPT OF 3 NETWORKS

```
#Starting to write the dml file
schema [_find .schemas.Net]

Network1 [
Net [
      router[
      id 0
      interface [id 0 bitrate 4000000 latency 0.0]
      interface [id 1 bitrate 8000000 latency 0.0]
      interface [id 2 bitrate 4000000 latency 0.0]
      interface [id 3 bitrate 9000000 latency 0.0]
      interface [id 4 bitrate 4000000.0 latency 0.0
            queue [
            use  SSF.Net.droptailQueue
            ]
            monitor[
            use SSF.Net.droptailQueueMonitor_1
            probe_interval 0.1
            debug true
            ]
      buffer 10000
      ] #end of interface

      _find .dictionary.routerGraphFlowMonitored.graph

      ] #end of the router loop


      # starting of udp standard client declaration
      host[id 1
            _extends .dictionary.standardClient
            nhi_route [dest default interface 0 next_hop 0(0)]
      ] #end of udp standard client


      # starting of udp standard server declaration
      host[id 2
            _extends .dictionary.standardServer
      nhi_route [dest default interface 0 next_hop 0(1)]
      ] #end of host2


      # starting of udp attack client declaration
      host[id 3
            _extends .dictionary.attackClient
            nhi_route [dest default interface 0 next_hop 0(2)]
      ] #end of udp attack client
```

```
      # starting of udp attack server declaration
      host[id 4
            _extends .dictionary.attackServer
      nhi_route [dest default interface 0 next_hop 0(3)]
      ] #end of host4

       link [attach 0(0) attach 1(0)]
       link [attach 0(1) attach 2(0)]

       link [attach 0(2) attach 3(0)]
       link [attach 0(3) attach 4(0)]

      graphics [
       collapsed false
       render [
        net [
            expanded [
            ]
       ]
       ]
      x 100.0
      y 100.0
      transform [
       affine 0.66,0.0,0.0,0.66,-300.0,-400.0
      ]
      ]

] #end of the Net loop
] #end of Network loop


Network2 [
Net [
      router[
      id 0
      interface [id 0 bitrate 4000000 latency 0.0]
      interface [id 1 bitrate 4000000 latency 0.0]
      interface [id 2 bitrate 4000000 latency 0.0]
      interface [id 3 bitrate 4000000 latency 0.0]
      interface [id 4 bitrate 4000000.0 latency 0.0
            queue [
            use  SSF.Net.droptailQueue
            ]
            monitor[
            use SSF.Net.droptailQueueMonitor_1
            probe_interval 0.1
            debug true
            ]
      buffer 10000
      ] #end of interface
      interface [id 5 bitrate 8000000.0 latency 0.0
            queue [
            use  SSF.Net.droptailQueue
```

```
            ]
            monitor[
            use SSF.Net.droptailQueueMonitor_1
            probe_interval 0.1
            debug true
            ]
      buffer 10000
      ] #end of interface

      _find .dictionary.routerGraphFlowMonitored.graph

      ] #end of the router loop


      # starting of udp standard client declaration
      host[id 1
            _extends .dictionary.standardClient
            nhi_route [dest default interface 0 next_hop 0(0)]
      ] #end of udp standard client


      # starting of udp standard server declaration
      host[id 2
            _extends .dictionary.standardServer
      nhi_route [dest default interface 0 next_hop 0(1)]
      ] #end of host2


      # starting of udp attack client declaration
      host[id 3
            _extends .dictionary.attackClient
            nhi_route [dest default interface 0 next_hop 0(2)]
      ] #end of udp attack client


      # starting of udp attack server declaration
      host[id 4
            _extends .dictionary.attackServer
      nhi_route [dest default interface 0 next_hop 0(3)]
      ] #end of host4

       link [attach 0(0) attach 1(0)]
       link [attach 0(1) attach 2(0)]

       link [attach 0(2) attach 3(0)]
       link [attach 0(3) attach 4(0)]

      graphics [
       collapsed false
       render [
        net [
            expanded [
            ]
```

```
       ]
       ]
      x 100.0
      y 100.0
      transform [
       affine 0.66,0.0,0.0,0.66,-300.0,-400.0
      ]
      ]

] #end of the Net loop
] #end of Network loop


Network3 [
Net [
      router[
      id 0
      interface [id 0 bitrate 4000000 latency 0.0]
      interface [id 1 bitrate 4000000 latency 0.0]
      interface [id 2 bitrate 4000000 latency 0.0]
      interface [id 3 bitrate 4000000 latency 0.0]
      interface [id 5 bitrate 8000000.0 latency 0.0
            queue [
            use  SSF.Net.droptailQueue
            ]
            monitor[
            use SSF.Net.droptailQueueMonitor_1
            probe_interval 0.1
            debug true
            ]
      buffer 10000
      ] #end of interface


      _find .dictionary.routerGraphFlowMonitored.graph

      ] #end of the router loop


      # starting of udp standard client declaration
      host[id 1
            _extends .dictionary.standardClient
            nhi_route [dest default interface 0 next_hop 0(0)]
      ] #end of udp standard client


      # starting of udp standard server declaration
      host[id 2
            _extends .dictionary.standardServer
      nhi_route [dest default interface 0 next_hop 0(1)]
      ] #end of host2
```

```
      # starting of udp attack client declaration
      host[id 3
            _extends .dictionary.attackClient
            nhi_route [dest default interface 0 next_hop 0(2)]
      ] #end of udp attack client


      # starting of udp attack server declaration
      host[id 4
            _extends .dictionary.attackServer
      nhi_route [dest default interface 0 next_hop 0(3)]
      ] #end of host4

       link [attach 0(0) attach 1(0)]
       link [attach 0(1) attach 2(0)]

       link [attach 0(2) attach 3(0)]
       link [attach 0(3) attach 4(0)]

      graphics [
       collapsed false
       render [
        net [
            expanded [
            ]
       ]
       ]
      x 100.0
      y 100.0
      transform [
       affine 0.66,0.0,0.0,0.66,-300.0,-400.0
      ]
      ]

] #end of the Net loop
] #end of Network loop



Net [
      frequency 1000000000000000
       AS_status boundary
       ospf_area 0

      #random number generation
      randomstream [
      generator "MersenneTwister"
      stream DefaultStream
      ]


      Net [id 1 _extends .Network1.Net]
```

```
      Net [id 2 _extends .Network2.Net]

      Net [id 3 _extends .Network3.Net]


link [attach 1:0(4) attach 2:0(4) delay 0.0]

link [attach 2:0(5) attach 3:0(5) delay 0.0]

traffic [
      pattern [
       client 3:1
       servers [port 10 nhi 1:2(0)]
      ]

      pattern [
       client 3:3
       servers [port 10 nhi 1:4(0)]
      ]

]

] #Net loop closes

dictionary[

      standardClient [
       interface [id 0 _extends .dictionary.10BaseT]
       route [dest default interface 0]
       graph [
            ProtocolSession [
             name client use SSF.OS.UDP.test.udpStreamClient
             start_time 30.0
             start_window 0.0
             file_size 3000000
             _find .dictionary.appsession.request_size
             _find .dictionary.appsession.datagram_size
             _find .dictionary.appsession.show_report
             _find .dictionary.appsession.debug
            ]
            ProtocolSession [name socket use
SSF.OS.Socket.socketMaster]
            ProtocolSession [name udp use SSF.OS.UDP.udpSessionMaster
             _find .dictionary.udpinit]
            ProtocolSession [name ip use SSF.OS.IP]
       ]
      ]

      attackClient [
       interface [id 0 _extends .dictionary.10BaseT]
       route [dest default interface 0]
       graph [
        ProtocolSession [
```

```
            name client use SSF.OS.UDP.test.udpStreamClient
            start_time 30.0
            start_window 0.0
            file_size 3000000
            _find .dictionary.appsession.request_size
            _find .dictionary.appsession.datagram_size
            _find .dictionary.appsession.show_report
            _find .dictionary.appsession.debug
            ]
          ProtocolSession [name socket use SSF.OS.Socket.socketMaster]
          ProtocolSession [name udp use SSF.OS.UDP.udpSessionMaster
           _find .dictionary.udpinit]
          ProtocolSession [name ip use SSF.OS.IP]
         ]
        ]

        10BaseT [
         bitrate 4000000
         latency 0.0
        ]

        10BaseTBT [
         bitrate 8000000
         latency 0.0
        ]

        10BaseTRT [
         bitrate 9000000
         latency 0.0
        ]

        udpinit [
              max_datagram_size 100000
              debug false
        ]

        standardServer [
               interface [id 0 _extends .dictionary.10BaseTBT]
               route [dest default interface 0]
               graph [
                    ProtocolSession [
                     name server use SSF.OS.UDP.test.udpStreamServer
                     port 10
                     client_limit 10
                     _find .dictionary.appsession.request_size
                     _find .dictionary.appsession.datagram_size
                     _find .dictionary.appsession.send_interval
                     _find .dictionary.appsession.show_report
                     _find .dictionary.appsession.debug
                    ]
                    ProtocolSession [name socket use
SSF.OS.Socket.socketMaster]
                    ProtocolSession [name udp use
```

```
SSF.OS.UDP.udpSessionMaster
                  _find .dictionary.udpinit]
                 ProtocolSession [name ip use SSF.OS.IP]
             ]
            ]

      attackServer [
       interface [id 0 _extends .dictionary.10BaseTRT]
       route [dest default interface 0 ]
       graph [
            ProtocolSession [
             name server use SSF.OS.UDP.test.udpStreamServer
             port 10
             client_limit 10
             _find .dictionary.appsession.request_size
             _find .dictionary.appsession.datagram_size
             _find .dictionary.appsession.send_attk_interval
             _find .dictionary.appsession.show_report
             _find .dictionary.appsession.debug
             ]
            ProtocolSession [name socket use
SSF.OS.Socket.socketMaster]
            ProtocolSession [name udp use SSF.OS.UDP.udpSessionMaster
             _find .dictionary.udpinit]
            ProtocolSession [name ip use SSF.OS.IP]
            ]
      ]

      hostLANinterfaceMonitored [interface [id 0 _extends
.dictionary.100Gb
      _find .dictionary.queueMonitor.monitor
      ]]

      100Gb [
      bitrate 900000000
      latency 0.0
      ]

      baseRouterGraph [
      ProtocolSession [name ip use SSF.OS.IP]
#changed ospf version
      ProtocolSession [name ospf use SSF.OS.OSPF.sOSPF]
      ]

      routerGraphFlowMonitored [graph [
      _extends .dictionary.baseRouterGraph
      ProtocolSession [
            name ip use SSF.OS.IP
            monitor [
            name ipnetflow use SSF.OS.NetFlow.IpFlowCollector
            protocol_type all
            max_inactive_time 10
            max_flow_time 100000
```

```
      ]
]
ProtocolSession [
name probe use SSF.OS.ProbeSession
file "sampada.dat"
stream netflow
]
]]

baseServerGraph [
ProtocolSession [
      name server use SSF.OS.TCP.test.tcpServer
      port 10
      _find .dictionary.appsession.request_size
      _find .dicitonary.appsession.show_report
      _find .dictionary.appsession.debug
      _find .dictionary.appsession.qlimit
]
ProtocolSession [name socket use SSF.OS.Socket.socketMaster]
ProtocolSession [name tcp use SSF.OS.TCP.tcpSessionMaster
      _find .dicitonary.tcpinit]
ProtocolSession [name ip use SSF.OS.IP
monitor [
use SSF.App.DDoS.RequestsMonitor
probe_interval 100.0
debug true
]
]]

serverGraphNICMonitored [graph [
_extends .dictionary.baseServerGraph
ProtocolSession [
      name probe use SSF.OS.ProbeSession
      file "sampada.dat"
      stream netflow
]
]]

#TCP initial parameters
tcpinit[
ISS 10000
MSS 1000
RcvWndSize 32
SendWndSize 32
SendBufferSize 128
MaxRexmitTimes 12
TCP_SLOW_INTERVAL 0.5
TCP_FAST_INTERVAL 0.2
MSL 60.0
MaxIdleTime 600.0
delayed_ack false
fast_recovery false
show_report true
```

```
      ]

      queueMonitor [monitor [
      use SSF.Net.droptailQueueMonitor_1
      probe_interval 0.1
      protocol_type udp
      debug true
      ]]

      appsession [
      request_size 500
      datagram_size 1000
      send_attk_interval 0.00088888
      send_interval 0.001
      qlimit 5000
      show_report true
      debug true
      ]
] #dictionary loop closes

graphics [
render [ ]
transform [
 affine 1.0,0.0,0.0,1.0,495.0,396.0
]
]
background "197,246,251(T):126,235,246(B)"
width 600 height 600
```

APPENDIX B – DML SCRIPT OF 5 NETWORKS

```
#Starting to write the dml file
schema [_find .schemas.Net]

Network1 [
Net [
      router[
      id 0
      interface [id 0 bitrate 8000000 latency 0.0]
      interface [id 1 bitrate 8000000 latency 0.0]
      interface [id 2 bitrate 8000000 latency 0.0]
      interface [id 3 bitrate 8000000 latency 0.0]
      interface [id 4 bitrate 4000000.0 latency 0.0
            queue [
            use  SSF.Net.droptailQueue
            ]
            monitor[
            use SSF.Net.droptailQueueMonitor_1
            probe_interval 0.1
            debug true
            ]
      buffer 10000
      ] #end of interface

      interface [id 5 bitrate 4000000.0 latency 0.0
            queue [
            use  SSF.Net.droptailQueue
            ]
            monitor[
            use SSF.Net.droptailQueueMonitor_1
            probe_interval 0.1
            debug true
            ]
      buffer 10000
      ] #end of interface

      interface [id 6 bitrate 5000000.0 latency 0.0
            queue [
            use  SSF.Net.droptailQueue
            ]
            monitor[
            use SSF.Net.droptailQueueMonitor_1
            probe_interval 0.1
            debug true
            ]
      buffer 10000
      ] #end of interface

      _find .dictionary.routerGraphFlowMonitored.graph

      ] #end of the router loop
```

```
      # starting of udp standard client declaration
      host[id 1
            _extends .dictionary.standardClient
            nhi_route [dest default interface 0 next_hop 0(0)]
      ] #end of udp standard client


      # starting of udp standard server declaration
      host[id 2
            _extends .dictionary.standardServer
      nhi_route [dest default interface 0 next_hop 0(1)]
      ] #end of host2


      # starting of udp attack client declaration
      host[id 3
            _extends .dictionary.attackClient
            nhi_route [dest default interface 0 next_hop 0(2)]
      ] #end of udp attack client


      # starting of udp attack server declaration
      host[id 4
            _extends .dictionary.attackServer
      nhi_route [dest default interface 0 next_hop 0(3)]
      ] #end of host4

       link [attach 0(0) attach 1(0)]
       link [attach 0(1) attach 2(0)]

       link [attach 0(2) attach 3(0)]
       link [attach 0(3) attach 4(0)]

      graphics [
       collapsed false
       render [
        net [
            expanded [
            ]
       ]
       ]
      x 100.0
      y 100.0
      transform [
       affine 0.66,0.0,0.0,0.66,-300.0,-400.0
      ]
      ]

] #end of the Net loop
] #end of Network loop
```

```
Network2 [
Net [
      router[
      id 0
      interface [id 0 bitrate 8000000 latency 0.0]
      interface [id 1 bitrate 8000000 latency 0.0]
      interface [id 2 bitrate 8000000 latency 0.0]
      interface [id 3 bitrate 8000000 latency 0.0]
      interface [id 4 bitrate 4000000.0 latency 0.0
            queue [
            use  SSF.Net.droptailQueue
            ]
            monitor[
            use SSF.Net.droptailQueueMonitor_1
            probe_interval 0.1
            debug true
            ]
      buffer 10000
      ] #end of interface
      interface [id 5 bitrate 8000000.0 latency 0.0
            queue [
            use  SSF.Net.droptailQueue
            ]
            monitor[
            use SSF.Net.droptailQueueMonitor_1
            probe_interval 0.1
            debug true
            ]
      buffer 10000
      ] #end of interface

      _find .dictionary.routerGraphFlowMonitored.graph

      ] #end of the router loop


      # starting of udp standard client declaration
      host[id 1
            _extends .dictionary.standardClient
            nhi_route [dest default interface 0 next_hop 0(0)]
      ] #end of udp standard client


      # starting of udp standard server declaration
      host[id 2
            _extends .dictionary.standardServer
      nhi_route [dest default interface 0 next_hop 0(1)]
      ] #end of host2


      # starting of udp attack client declaration
      host[id 3
            _extends .dictionary.attackClient
```

```
            nhi_route [dest default interface 0 next_hop 0(2)]
      ] #end of udp attack client


      # starting of udp attack server declaration
      host[id 4
            _extends .dictionary.attackServer
      nhi_route [dest default interface 0 next_hop 0(3)]
      ] #end of host4

       link [attach 0(0) attach 1(0)]
       link [attach 0(1) attach 2(0)]

       link [attach 0(2) attach 3(0)]
       link [attach 0(3) attach 4(0)]

      graphics [
       collapsed false
       render [
        net [
            expanded [
            ]
       ]
       ]
      x 100.0
      y 100.0
      transform [
       affine 0.66,0.0,0.0,0.66,-300.0,-400.0
      ]
      ]

] #end of the Net loop
] #end of Network loop


Network3 [
Net [
      router[
      id 0
      interface [id 0 bitrate 8000000 latency 0.0]
      interface [id 1 bitrate 8000000 latency 0.0]
      interface [id 2 bitrate 8000000 latency 0.0]
      interface [id 3 bitrate 8000000 latency 0.0]
      interface [id 5 bitrate 8000000.0 latency 0.0
            queue [
            use  SSF.Net.droptailQueue
            ]
            monitor[
            use SSF.Net.droptailQueueMonitor_1
            probe_interval 0.1
            debug true
            ]
      buffer 10000
```

```
] #end of interface


_find .dictionary.routerGraphFlowMonitored.graph

] #end of the router loop


# starting of udp standard client declaration
host[id 1
      _extends .dictionary.standardClient
      nhi_route [dest default interface 0 next_hop 0(0)]
] #end of udp standard client


# starting of udp standard server declaration
host[id 2
      _extends .dictionary.standardServer
nhi_route [dest default interface 0 next_hop 0(1)]
] #end of host2


# starting of udp attack client declaration
host[id 3
      _extends .dictionary.attackClient
      nhi_route [dest default interface 0 next_hop 0(2)]
] #end of udp attack client


# starting of udp attack server declaration
host[id 4
      _extends .dictionary.attackServer
nhi_route [dest default interface 0 next_hop 0(3)]
] #end of host4

 link [attach 0(0) attach 1(0)]
 link [attach 0(1) attach 2(0)]

 link [attach 0(2) attach 3(0)]
 link [attach 0(3) attach 4(0)]

graphics [
 collapsed false
 render [
  net [
      expanded [
      ]
 ]
 ]
x 100.0
y 100.0
transform [
 affine 0.66,0.0,0.0,0.66,-300.0,-400.0
```

```
      ]
      ]

] #end of the Net loop
] #end of Network loop

Network4 [
Net [
      router[
      id 0
      interface [id 0 bitrate 8000000 latency 0.0]
      interface [id 1 bitrate 4000000 latency 0.0]
      interface [id 2 bitrate 8000000 latency 0.0]
      interface [id 3 bitrate 8000000 latency 0.0]
      interface [id 5 bitrate 4000000.0 latency 0.0
            queue [
            use  SSF.Net.droptailQueue
            ]
            monitor[
            use SSF.Net.droptailQueueMonitor_1
            probe_interval 0.1
            debug true
            ]
      buffer 10000
      ] #end of interface

      _find .dictionary.routerGraphFlowMonitored.graph

      ] #end of the router loop


      # starting of udp standard client declaration
      host[id 1
            _extends .dictionary.standardClient
            nhi_route [dest default interface 0 next_hop 0(0)]
      ] #end of udp standard client


      # starting of udp standard server declaration
      host[id 2
            _extends .dictionary.standardServer
      nhi_route [dest default interface 0 next_hop 0(1)]
      ] #end of host2


      # starting of udp attack client declaration
      host[id 3
            _extends .dictionary.attackClient
            nhi_route [dest default interface 0 next_hop 0(2)]
      ] #end of udp attack client


      # starting of udp attack server declaration
```

```
      host[id 4
            _extends .dictionary.attackServer
      nhi_route [dest default interface 0 next_hop 0(3)]
      ] #end of host4

       link [attach 0(0) attach 1(0)]
       link [attach 0(1) attach 2(0)]

       link [attach 0(2) attach 3(0)]
       link [attach 0(3) attach 4(0)]

      graphics [
       collapsed false
       render [
        net [
            expanded [
            ]
       ]
       ]
      x 100.0
      y 100.0
      transform [
       affine 0.66,0.0,0.0,0.66,-300.0,-400.0
      ]
      ]

] #end of the Net loop
] #end of Network loop

Network5 [
Net [
      router[
      id 0
      interface [id 0 bitrate 8000000 latency 0.0]
      interface [id 1 bitrate 8000000 latency 0.0]
      interface [id 2 bitrate 8000000 latency 0.0]
      interface [id 3 bitrate 5000000 latency 0.0]
      interface [id 6 bitrate 5000000.0 latency 0.0
            queue [
            use  SSF.Net.droptailQueue
            ]
            monitor[
            use SSF.Net.droptailQueueMonitor_1
            probe_interval 0.1
            debug true
            ]
      buffer 10000
      ] #end of interface

      _find .dictionary.routerGraphFlowMonitored.graph

      ] #end of the router loop
```

```
      # starting of udp standard client declaration
      host[id 1
            _extends .dictionary.standardClient
            nhi_route [dest default interface 0 next_hop 0(0)]
      ] #end of udp standard client


      # starting of udp standard server declaration
      host[id 2
            _extends .dictionary.standardServer
      nhi_route [dest default interface 0 next_hop 0(1)]
      ] #end of host2


      # starting of udp attack client declaration
      host[id 3
            _extends .dictionary.attackClient
            nhi_route [dest default interface 0 next_hop 0(2)]
      ] #end of udp attack client


      # starting of udp attack server declaration
      host[id 4
            _extends .dictionary.attackServer
      nhi_route [dest default interface 0 next_hop 0(3)]
      ] #end of host4

       link [attach 0(0) attach 1(0)]
       link [attach 0(1) attach 2(0)]

       link [attach 0(2) attach 3(0)]
       link [attach 0(3) attach 4(0)]

      graphics [
       collapsed false
       render [
        net [
            expanded [
            ]
       ]
       ]
      x 100.0
      y 100.0
      transform [
       affine 0.66,0.0,0.0,0.66,-300.0,-400.0
      ]
      ]

] #end of the Net loop
] #end of Network loop
```

```
Net [
      frequency 1000000000000000
       AS_status boundary
       ospf_area 0

      #random number generation
      randomstream [
      generator "MersenneTwister"
      stream DefaultStream
      ]


      Net [id 1 _extends .Network1.Net]

      Net [id 2 _extends .Network2.Net]

      Net [id 3 _extends .Network3.Net]

      Net [id 4 _extends .Network4.Net]

      Net [id 5 _extends .Network5.Net]



link [attach 1:0(4) attach 2:0(4) delay 0.0]

link [attach 2:0(5) attach 3:0(5) delay 0.0]

link [attach 4:0(5) attach 1:0(5) delay 0.0]

link [attach 5:0(6) attach 1:0(6) delay 0.0]

traffic [

      pattern [
       client 3:3
       servers [port 10 nhi 5:4(0)]
      ]

      pattern [
       client 3:1
       servers [port 10 nhi 4:2(0)]
      ]

]

] #Net loop closes

dictionary[

      standardClient [
       interface [id 0 _extends .dictionary.10BaseT]
```

```
       route [dest default interface 0]
       graph [
            ProtocolSession [
             name client use SSF.OS.UDP.test.udpStreamClient
             start_time 30.0
             start_window 0.0
             file_size 3000000
             _find .dictionary.appsession.request_size
             _find .dictionary.appsession.datagram_size
             _find .dictionary.appsession.show_report
             _find .dictionary.appsession.debug
            ]
            ProtocolSession [name socket use
SSF.OS.Socket.socketMaster]
            ProtocolSession [name udp use SSF.OS.UDP.udpSessionMaster
             _find .dictionary.udpinit]
            ProtocolSession [name ip use SSF.OS.IP]
       ]
      ]

      attackClient [
       interface [id 0 _extends .dictionary.10BaseT]
       route [dest default interface 0]
       graph [
        ProtocolSession [
            name client use SSF.OS.UDP.test.udpStreamClient
            start_time 30.0
            start_window 0.0
            file_size 3000000
            _find .dictionary.appsession.request_size
            _find .dictionary.appsession.datagram_size
            _find .dictionary.appsession.show_report
            _find .dictionary.appsession.debug
            ]
        ProtocolSession [name socket use SSF.OS.Socket.socketMaster]
        ProtocolSession [name udp use SSF.OS.UDP.udpSessionMaster
         _find .dictionary.udpinit]
        ProtocolSession [name ip use SSF.OS.IP]
       ]
      ]

      10BaseT [
       bitrate 8000000
       latency 0.0
      ]

      10BaseTRT [
       bitrate 5000000
       latency 0.0
      ]

      10BaseTBT [
       bitrate 4000000
```

```
       latency 0.0
      ]


      udpinit [
            max_datagram_size 100000
            debug false
      ]

      standardServer [
             interface [id 0 _extends .dictionary.10BaseTBT]
             route [dest default interface 0]
             graph [
                  ProtocolSession [
                   name server use SSF.OS.UDP.test.udpStreamServer
                   port 10
                   client_limit 10
                   _find .dictionary.appsession.request_size
                   _find .dictionary.appsession.datagram_size
                   _find .dictionary.appsession.send_interval
                   _find .dictionary.appsession.show_report
                   _find .dictionary.appsession.debug
                  ]
                  ProtocolSession [name socket use
SSF.OS.Socket.socketMaster]
                  ProtocolSession [name udp use
SSF.OS.UDP.udpSessionMaster
                   _find .dictionary.udpinit]
                  ProtocolSession [name ip use SSF.OS.IP]
             ]
            ]

      attackServer [
       interface [id 0 _extends .dictionary.10BaseTRT]
       route [dest default interface 0 ]
       graph [
            ProtocolSession [
             name server use SSF.OS.UDP.test.udpStreamServer
             port 10
             client_limit 10
             _find .dictionary.appsession.request_size
             _find .dictionary.appsession.datagram_size
             _find .dictionary.appsession.send_attk_interval
             _find .dictionary.appsession.show_report
             _find .dictionary.appsession.debug
             ]
            ProtocolSession [name socket use
SSF.OS.Socket.socketMaster]
            ProtocolSession [name udp use SSF.OS.UDP.udpSessionMaster
             _find .dictionary.udpinit]
            ProtocolSession [name ip use SSF.OS.IP]
            ]
      ]
```

```
      hostLANinterfaceMonitored [interface [id 0 _extends
.dictionary.100Gb
      _find .dictionary.queueMonitor.monitor
      ]]

      100Gb [
      bitrate 900000000
      latency 0.0
      ]

      baseRouterGraph [
      ProtocolSession [name ip use SSF.OS.IP]
      ProtocolSession [name ospf use SSF.OS.OSPF.sOSPF]
      ]

      routerGraphFlowMonitored [graph [
      _extends .dictionary.baseRouterGraph
      ProtocolSession [
            name ip use SSF.OS.IP
            monitor [
            name ipnetflow use SSF.OS.NetFlow.IpFlowCollector
            protocol_type all
            max_inactive_time 10
            max_flow_time 100000
            ]
      ]
      ProtocolSession [
      name probe use SSF.OS.ProbeSession
      file "sampada.dat"
      stream netflow
      ]
      ]]

      baseServerGraph [
      ProtocolSession [
            name server use SSF.OS.TCP.test.tcpServer
            port 10
            _find .dictionary.appsession.request_size
            _find .dicitonary.appsession.show_report
            _find .dictionary.appsession.debug
            _find .dictionary.appsession.qlimit
      ]
      ProtocolSession [name socket use SSF.OS.Socket.socketMaster]
      ProtocolSession [name tcp use SSF.OS.TCP.tcpSessionMaster
            _find .dicitonary.tcpinit]
      ProtocolSession [name ip use SSF.OS.IP
      monitor [
      use SSF.App.DDoS.RequestsMonitor
      probe_interval 100.0
      debug true
      ]
      ]]
```

```
      serverGraphNICMonitored [graph [
      _extends .dictionary.baseServerGraph
      ProtocolSession [
            name probe use SSF.OS.ProbeSession
            file "sampada.dat"
            stream netflow
      ]
      ]]

      #TCP initial parameters
      tcpinit[
      ISS 10000
      MSS 1000
      RcvWndSize 32
      SendWndSize 32
      SendBufferSize 128
      MaxRexmitTimes 12
      TCP_SLOW_INTERVAL 0.5
      TCP_FAST_INTERVAL 0.2
      MSL 60.0
      MaxIdleTime 600.0
      delayed_ack false
      fast_recovery false
      show_report true
      ]

      queueMonitor [monitor [
      use SSF.Net.droptailQueueMonitor_1
      probe_interval 0.1
      protocol_type udp
      debug true
      ]]

      appsession [
      request_size 500
      datagram_size 1000
      send_attk_interval 0.0016
      send_interval 0.002
      qlimit 5000
      show_report true
      debug true
      ]
] #dictionary loop closes

graphics [
render [ ]
transform [
 affine 1.0,0.0,0.0,1.0,495.0,396.0
]
]
background "197,246,251(T):126,235,246(B)"
width 600 height 600
```

APPENDIX C – SIMULATION ISSUES

Some of the issues with the SSFNet simulator that were encountered while gathering results are listed below.

1. Lack of adaptive routing: SSF.OS.OSPF is a partial implementation of OSPFv2, based on the Internet Engineering Task Force's Request for Comments number 2328 (RFC 2328). It is designed to quickly compute the routing tables for arbitrary topologies in SSFNet network models. The unsupported requirements include dynamic neighbor discovery and link state updates in response to dynamic topology changes. We use this OSPF version in our simulations. This does not reflect the way the Internet works in reality. Future research could include implementation of protocols that accurately simulate the working of the Internet.

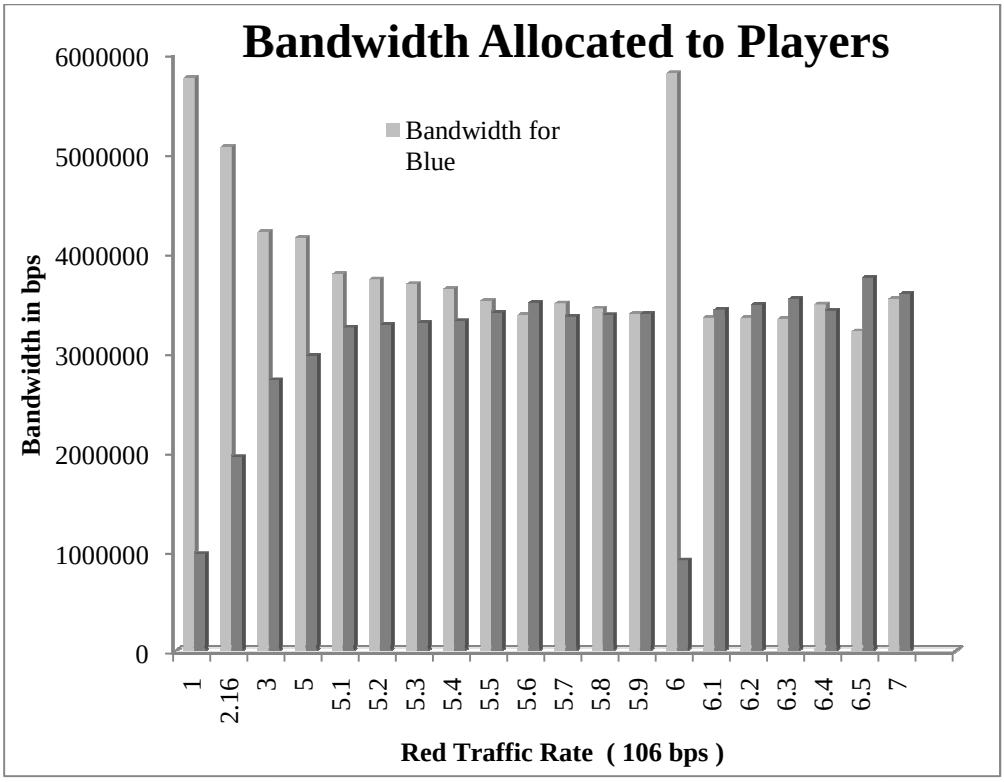


Figure: Glitch observed when Blue traffic rate equals Red Traffic rate

2. As shown in the figure above, a glitch is observed at a specific network traffic level when the rate at which blue server generates data is equal to the rate at which red server generates data. The bandwidth allocated to Blue spikes unexpectedly whereas the bandwidth allocated to Red is significantly less. This behavior is observed for all the mincut arcs of all the configurations. We believe this to be an artifact of the simulator. The simulator fails to behave as expected at this point.

REFERENCES

[1] The Anatomy of the Twitter Attack. (online). http://www.washingtonpost.com/wp-dyn/content/article/2009/07/19/AR2009071900341.html. Last accessed: 12/03/2009

[2] T. Pardoe and G. Snyder, *Network Security*, Thomson Delmar Learning. 2005, pp.xiv.

[3] R. R. Brooks, *Disruptive Security Technologies with Mobile Code and Peer-to Peer Networks*, Boca Raton, FL: CRC Press, 2005.

[4] J. Albanese and W. Sonnenreich, *Network Security Illustrated*, McGraw-Hill Companies. 2004, pp.4

[5] P. W. Dowd and J. T. Mchenry, Network security: it's time to take it seriously. Computer, 31(9):24–28, 1998

[6] J. Mirkovic, J. Martin, and P. Reiher, "A taxonomy of DDoS attacks and DDoS defense mechanisms," in *ACM CCR*, April 2004.

[7] T. Peng, "Defending against Distributed Denial of Service Attacks", University of Melbourne, April 2004.

[8] J. Nazario, J. Linden, Botnet tracking techniques and tools. (online). *www.blackhat.com/presentations/bh-dc-07/.../bh-dc-07-Nazario.pdf. Last accessed: 12/03/2009*

[9] D.Barroso, Botnets - The Silent Threat. ENISA Position Paper No. 3. November 2007.

[10] CERT/CC Statistics, http://www.cert.org/stats/cert stats.html. Last accessed: April 2004.

[11] C. Dingankar, Enterprise Security Analysis Including Denial of Service Countermeasures, Clemson University, August 2007.

[12] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for your mathematical plays Volume 1: Games in General*, Academic Press, New York, 1982, pp.383-391

[13] F. Lau, S. H. Rubin, M. H. Smith, and L. Trajkovic, "Distributed Denial of Service Attacks," in *IEEE International Conference on Systems, Man, and Cybernetics*, pp. 2275-2280, Nashville, TN, USA, October 2000

[14] Cyber Crime – It could happen to you. (online). *www.novell.com/ncmagopenxtest/2000/04/cyber40.pdf. Last accessed: 12/03/2009*

[15] D. Moore, G. M. Voelker, and S. Savage, "Inferring Internet Denial-of-Service Activity," *Usenix Security Symposium*, 2001.

[16] National Cyber Alert System. (online). http://www.us-cert.gov/cas/tips/ST04-015.html. Last accessed: 12/03/2009

[17] Frontline, "Cyber War!" (online). http://www.pbs.org/wgbh/pages/frontline/shows/cyberwar/warnings/. Last accessed: 12/03/2009

[18] NY times (online). http://www.nytimes.com/2007/05/28/business/worldbusiness/28iht-cyberwar.4.5901141.html?pagewanted=1&_r=1. Last accessed: 10/01/2009

[19] J.Guo, W.Xiang, S.Wang, Reinforce Networking Theory with OPNET Simulation, *Journal of Information technology Education*, Volume 6, 2007

[20] Open source SSFNet simulator (online). http://www.ssfnet.org/homePage.html. Last accessed: 12/03/2009

[21] J. Cowie, D. Nicol, and A. Ogielski, "Modeling the global internet," *Comput. Sci. & Eng.*, vol. 1, no. 1, pp. 42–50, Jan. 1999.

[22] D.M. Nicol , J Liu , M Liljenstam , G Yan, Simulation of large scale networks I: simulation of large-scale networks using SSF, *Proceedings of the 35th conference on Winter simulation: driving innovation*, December 07-10, 2003, New Orleans, Louisiana

[23] S Yoon, YB Kim, A design of network simulation environment using SSFNet, *Proceedings of the 2009 First International Conference on Advances in System Simulation* - vol 00, pp. 73-78 , 2009

[24] DaSSF. (online). http://www.cs.dartmouth.edu/research/DaSSF/. Last accessed: 12/03/2009

[25] B. Bollobas ,*Random Graphs*, Cambridge University Press, Second Edition, 2001, pp.143

[26] Kevin, J., Weaver, G. M., Long, N., and Thomas, R, "Trends in denial of service attack technology," Technical report, CERT Coordination Center, Carnegie Mellon University, October, 2001.

[27] C. Tondering, "Surreal Numbers - An Introduction," Version 1.5, January 2005 (online). Available: http://www.tondering.dk/claus/sur15.pdf. Last accessed 12/01/2009

[28] L. J. Yedwab, "On playing well in a sum of games," M.S. Thesis, MIT, 1985, MIT/LCS/TR-348.

[29] J. H. Conway, *On Numbers and Games*, AK Peters, LTD, 2000, Ch 2.

[30] B. C. A. Milvang-Jensen, "Combinatorial Games, Theory and Applications," Thesis, IT University of Copenhagen, 2000.

[31] K. Virtanen., R. P. Hamalainen, and V. Mattila, "Team Optimal Signaling Strategies in Air Combat," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 36, no. 4, pp. 643-660, July, 2006.

[32] Pydev Extensions (online). http://pydev.org. Last accessed: 12/03/2009

[33] B J Premore, Dartmouth College, SSFNet and Routing Simulation (online). Last accessed: 12/03/2009

[34] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, *Network Flows*, Prentice Hall, Upper Saddle River, NJ, 1993.

[35] R. R. Brooks, B. Pillai, S. Rai and S. Racunas, "Mobile Network Analysis Using Probabilistic Connectivity Matrices", IEEE Transactions on Systems Man and Cybernetics, Part C, Accepted for Publication, Nov. 2005.

[36] R. Albert and A.L. Barabasi, Statistical mechanics of complex networks, Reviews of Modern Physics 74, 47 (2002)

[37] S. Jensen, T. Luczak, and A, Rucinski, Random Graphs. New York:Wiley, 2000.

[38] S. Joseph, Max-flow min-cut Theorem (online).
http://www.math.uri.edu/~eaton/maxflowmincut.pdf. Last accessed: 12/03/2009

[39] Botnets for Rent (online). http://www.pbs.org/kcet/wiredscience/story/12-botnets_for_rent.html. Last accessed: 12/02/2009

[40] R. Chen, J. Park, and R. Marchany, "A Divide-and-Conquer Strategy for Thwarting Distributed Denial-of-Service Attacks," *IEEE Transactions on Parallel and Distributed Systems***,** vol 18, issue 5, May 2007, pp 577-588D.

[41] R. Chang, "Defending against flooding-based, distributed denial-of-service attacks: a tutorial," *IEEE Communications Magazine*, 40(10), 2002.

[42] G. Carl, G. Kesidis, R. Brooks and S. Rai, "Denial-of-Service Attack-Detection Techniques," *IEEE Internet Computing*, vol. 10, nos. 1, pp. 82-89, Jan. 2006.