Macalester College
DigitalCommons@Macalester College

Honors Projects | Mathematics, Statistics, and Computer Science

5-1-2006

Analysis of Defenses against Distributed Denial of Service Attacks

D. Eric Chan-Tin
Macalester College, echantin@alumni.macalester.edu

Follow this and additional works at: http://digitalcommons.macalester.edu/mathcs_honors

Recommended Citation
Chan-Tin, D. Eric, "Analysis of Defenses against Distributed Denial of Service Attacks" (2006). *Honors Projects*. Paper 4.
http://digitalcommons.macalester.edu/mathcs_honors/4

This Article is brought to you for free and open access by the Mathematics, Statistics, and Computer Science at DigitalCommons@Macalester College. It has been accepted for inclusion in Honors Projects by an authorized administrator of DigitalCommons@Macalester College. For more information, please contact scholarpub@macalester.edu.

Analysis of Defenses against Distributed Denial of Service Attacks

D. Eric Chan-Tin

Advisor: Dr. G. Michael Schneider
Department of Computer Science
May 1, 2006

Table of Contents

Abstract....1
1. Introduction....2
1.1 Analogy....2
1.2 Why?....2
1.3 Examples....5
1.4 DDoS Attacks....6
1.5 Defenses....13
1.6 Similar Work....18
1.7 This Paper....19
2. Network Simulation....21
2.1 Introduction....21
2.2 Example....22
2.3 Goals....30
2.4 Technical Details....30
2.4.1 Software Used....30
2.4.2 Time Units....31
2.4.3 Events....32
2.4.4 Distributions....34
2.4.5 Parameters....34
2.5 Sample Output....41
2.6 Useful Statistics....41
3. Simulation Runs....44
3.1 Test Runs....44
3.2 Goals....44
3.3 Base Case....45
3.3.1 What is a Base Case?....45
3.3.2 Base Case 1 – Number of Good Clients....45
3.3.3 Base Case 2 – Evil Clients come in!!....48
3.3.4. Message Queue always Empty?....50
3.4 Defenses....52
4 Experiment 1 – Using a Priority Queue....53
4.1 Goals....53
4.2 New Parameters....54
4.3 Runs....54
4.4 Conclusion....63
5 Experiment 2 – Limit Number of Connections....65
5.1 Goals....65
5.2 New Parameter....66
5.3 Runs....66
5.4 Conclusion....74
6 Experiment 3 – Server Reset....75
6.1 Goals....75
6.2 New Parameter....76
6.3 Runs....77

6.4 Conclusion.....83
7 Conclusions.....85
8 Future Work.....88
9. Bibliography.....90
10. Acknowledgments.....93
Appendix A – Raw Data and Tables.....94
Appendix B – Code.....95

Abstract

Distributed Denial of Service (DDoS) attacks are attempts to overwhelm a computer system in order to deny access by legitimate users. They are generally unstoppable, but there is a good deal of on-going research on methods to reduce their negative effects. This paper will deal with the design of a model that simulates such an attack. The simulation model is then used to study possible ways to defend against these attacks. Three experiments are run: 1) using a priority queue to sort messages from clients based on how many connections they have open on the server; 2) limiting the number of connections each client can create; and 3) having the server forcefully delete the oldest established connection, whenever its connection table becomes full. Results show that method 1 is totally ineffective while method 2 somewhat improves the overall performance of the system. However, method 3, combined with method 2, produces significantly improved performance against a DDoS attack.

1. Introduction

1.1 Analogy

Imagine a telephone system which can handle a certain number of calls at any one time. A person, either intentionally (having many telephones at his* disposal and using all of them at the same time) or unknowingly (placing a lot of legitimate calls at the same time), could hog all the available slots, preventing any other person from being able to place a call. This analogy is similar to a Denial of Service (DoS) attack. It tries to use up all the resources of a server or client computer, preventing legitimate use of that system. A DoS attacker is usually using only one computer for the attack. However, a Distributed Denial of Service (DDoS) attack occurs when many computers attack a single one, and it is much more common nowadays.

1.2 Why?

With the current popularity of the Internet, most computers are connected to the Internet via a high speed connection. A DDoS attack could be happening at any time.

* Note that "his" and "he" is used. However, this does not indicate that the person is male. "he" is used for simplicity.

Similar to the analogy of the telephone system, a DDoS attack could also be happening through legitimate use of the Internet. A special phrase is used to denote when this is happening – flash crowds. A DoS attack can thus be defined as an overload that causes resources to be fully utilized by the attack and thus prevent the use of a computer or application. A DDoS attack achieves the same result as a DoS attack but in a more distributed and coordinated way, requiring multiple “attackers” or computers. Thus it is much harder to defend against.

Although the real reasons behind a DDoS attack is rarely or never known because the attacker is very seldom caught. Some potential reasons are

- Fame

  Fame within the hacker community is really important for attackers.

- Financial

  Newspapers report that gambling websites have to pay a certain amount of money per month to blackmailers else their websites will be flooded and inaccessible. The amount paid is thought to be less than the amount they would lose should their websites go down. Competitive companies might also attack each other in an effort to bring down the other company's website. If a popular website is down for even only an hour, that company might lose hundreds of customers and potentially thousands of dollars.

- Political

  Some country might want to cripple another country's servers for some reasons.

- Military

It might be a good idea to overload a sensitive or critical military server during times of wars.

A DDoS attack comes in all flavors and shapes. Each specific attack and its corresponding defense, if any, will be explained later in this paper.

Distributed Denial of Service attacks have often been coined as unstoppable [Farrow, n.d.] [Lau et al, 2000]. The attacks are often classified or further divided into three categories: detection, prevention, and traceability. Detection is about knowing when a DDoS attack is happening, and alerting the appropriate individuals, such as the network administrator, system administrator, and the ISP (Internet Service Provider). To stop such an attack, manual intervention is required. Prevention is a mostly automatic intervention in trying to stop the attack. Both detection and prevention techniques suffer from false positives. Traceability is trying to trace back where the attack came from. The IP addresses of the attacking computers are very often spoofed or faked. The real attacker has to be traced back and this is often very hard to impossible due to the very nature of the Internet.

This thesis will focus more on the prevention aspects with a little bit of detection. Only the TCP connection will be simulated. The research will focus on trying to prevent evil clients from even establishing a connection and allowing availability for good clients. The very basic connection will be simulated, and then prevention features will be added to see whether they provide any help in abating the attack(s) and denying service and connectivity to the evil clients, while providing a constant and good service to the good clients.

1.3 Examples

There are many Internet-related attacks going on every day. There is probably an attempted attack on one or more Macalester College systems right now. Denial of Service attacks probably represent a fair share of these attacks. DDoS attacks happened before and some examples are briefly listed below:

- A University of Minnesota computer was knocked off for more than two days in August 1999, when a DDoS tool called Trinoo was deployed in more than 200 attacking computers.
- In February 2000, Yahoo!, Amazon.com, CNN.com, and other major Web sites were brought down due to a distributed attack [Lau et al, 2000].
- On October 21, 2002, the root Domain Name System (DNS) servers were "pinged to death" for an hour.
- Gibson Research Corporation (http://www.grc.com) was brought down in May 2001.
- On May, 22 2001, CERT (widely regarded as the "Fork Knox of computer security") was knocked off the Internet.
- The Code Red worm in 2001 is also regarded as a DDoS attack.
- In March 2005, on the very first day of its public release, the Sun Grid was hit by a DDoS attack.

Since 1998, there have probably been hundreds of DoS attacks around the world. In 2001, "a quantitative estimate of worldwide DoS attack frequency found 12,000

attacks over a three-week period" [Carl et al, 2006]. Moreover, the 2004 CSI/FBI Computer Crime and Security Survey "listed DoS attacks among the most financially expensive security incidents" [Carl et al, 2006].

A Distributed Denial of Service attack is very real and is a significant threat.

1.4 DDoS Attacks

A Distributed Denial of Service attack can occur in many ways. Some are known as 'brute force' attacks. Others exploit a specific weakness in the network protocol, while some exploit a weakness in a specific application program. A DoS attack does not always have to occur on a server, or be targeted at a server. Some attacks exploit weaknesses in an application run on clients or on the average user's computers, causing the latter to crash and stop functioning.

A list of some of the most well-known types of DoS attacks are described below.

1) Brute Force

   A brute force attack usually just tries to eat up all resources by overloading a server with requests. One example would be a thousand computers trying to access the Macalester website repeatedly every millisecond. This would prevent any legitimate use of the website by any other users. There is no attempt at subtlety – just a flooding of requests to a single system. This is much like the attempt to overwhelm a plumbing system by turning on every tap on campus at the same time!

2) TCP SYN

The Transmission Control Protocol (TCP) requires a three way handshake before a connection can be established. A client sends an initial connect request to a server. When the server receives the request, it acknowledges (acks) back to the client. When the latter receives the ack, the connection has now been established for the client, but not for the server. The client thus sends an ack back to the server. When the latter receives the ack, the connection has been established for both the client and the server and transmission of data can begin. This three way handshake is shown in Figure 1.

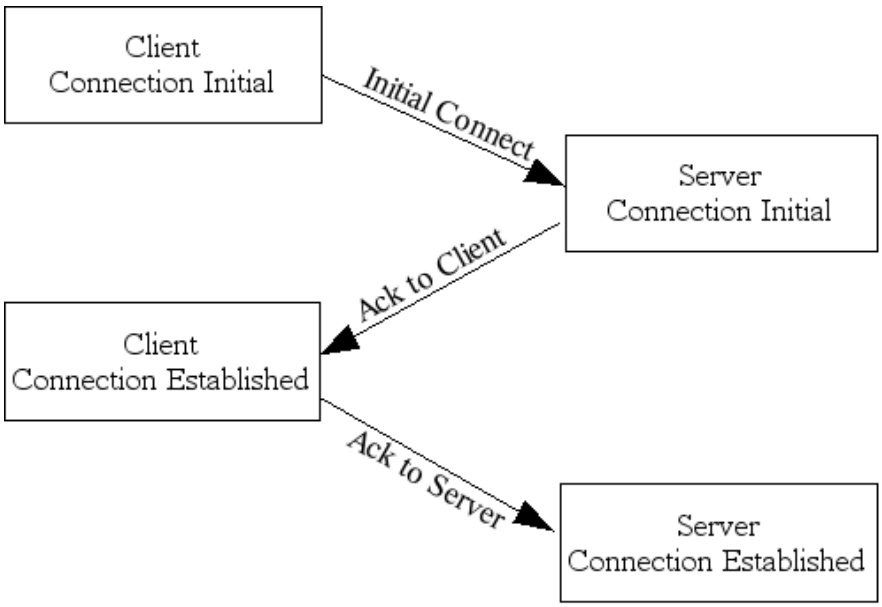

Figure 1: Three way handshake of TCP

The TCP SYN exploit is for a client to send an initial connect request to a server, and although the server acks back, the client never acks back to the server. This leaves the server waiting for an acknowledgment that will never arrive. However,

that initiated connection entry on the server will eventually time-out and be removed from the table, which will free up a port on the server for another connection to be created. The distributed attack is to have hundreds of malicious clients sending an initial connection request to the same server without sending any acknowledgments. Each connection created on the server takes up a port number and there are only 2^16 ports available on any computer nowadays. Eventually, the server will timeout those connections but for a brief amount of time, no other clients can access the server. In a sense, this attack succeeds by using up all connection table entries for a brief period of time.

3) LAND Attack

"A LAND attack is a DoS attack that consists of sending a special spoofed packet to a computer, causing it to lock up. The security flaw was first discovered in 1997 by someone using the alias 'm3lt', and it has resurfaced many years later in operating systems such as Windows Server 2003 and Windows XP SP2." [Land Attacks Still Going Strong, 2005] The spoofed packet is an ICMP (Internet Control Message Protocol) echo packet that has the same source and destination address. The system that receives this packet will just stall and will not know what to do with that packet. After a timeout, that packet will usually be discarded. The word “LAND” is used because the first variant of this attack required the attacker to be on site, but nowadays this attack can be performed remotely.

4) Teardrop Attack

The maximum size of a packet over the Internet is 65536 bytes. Some messages

can be much larger than 65000 bytes and must be fragmented, that is broken down into smaller pieces and sent individually via different routes, when the separate pieces arrive at the destination they are reassembled back into a single logical message. This could be the source of a potential attack. The server holds all the fragments until they can be reassembled. This uses up space in the server's table. A malicious user could send a lot of packets which cannot be reassembled, causing the table to become full and the server rejecting all other packets. Timeouts do help, but just as for the TCP SYN exploit, this prevents legitimate use of resources for a certain period of time. This situation is illustrated in Figure 2, in which three messages M, N, and P – each four fragments long, are all partially arrived but no single message is complete. No new fragments or messages can be processed by the system until the table (also known as a buffer) is freed.

M1	M2	M3
N1	P1	N3
P2	P3	N2

Figure 2: Full Message Buffer

5) Smurf Attack

Every computer on the Internet can be identified by its Internet Protocol (IP) address. The latter is unique for each computer, except for some reserved IP addresses which every router knows about. A malicious computer could spoof its IP address, so that its IP address appears to be the IP address of the victim

computer. The latter will then be overloaded with unrequested packets or messages which might cause it to crash. Figure 3 gives an example. The victim's computer IP address is 141.140.1.5 and the attacker's real IP address is 141.140.121.111. However, the attacker masquerades its IP address as 141.140.1.5 and sends a message to computer 64.236.24.12. The latter replies back to the spoofed IP address (141.140.1.5). This seems rather benign but if thousands of attacking computers perform such a smurf attack, the victim will be overloaded with messages from the same innocent computer (64.236.24.12). However, that computer could be a valid and legitimate system trying to surf the Internet peacefully, but it could be barred from connecting to 141.140.1.5 for no apparent reason.

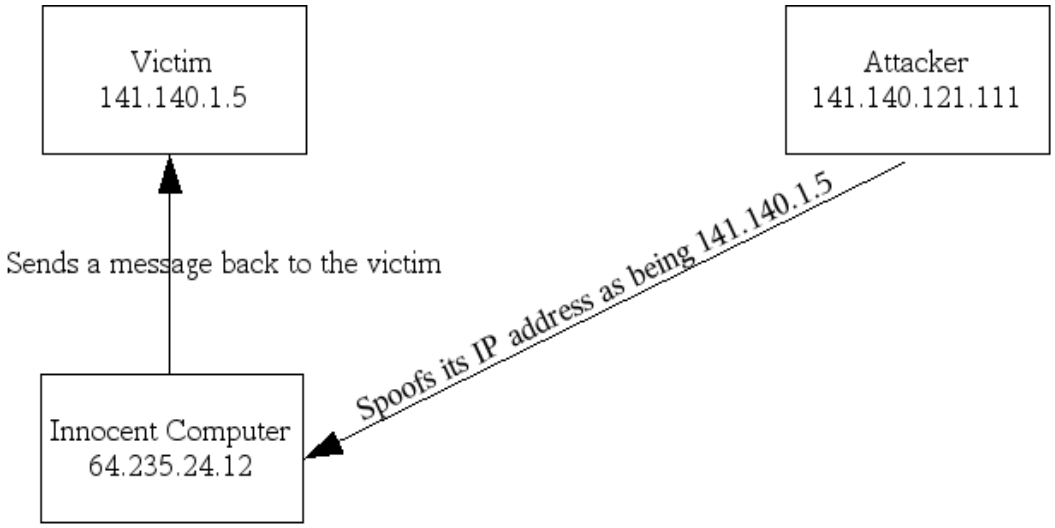

Figure 3: Smurf Attack

6) Email overflow

One of the oldest types of DoS attack is to fill up an email server's disk space preventing any other emails from being received. A lot of emails can be sent to an

email server, which will store all of them until the user retrieves them. If a sufficient amount of email is sent, this will fill up all free disk space on the server, and all future emails will be discarded. However, this is easily countered nowadays with Spam blockers and cheap and easily available disk space. Moreover, when the server's disk is filled up to a certain percentage, it could alert the system administrator.

7) Broadcast

Many routers have a broadcast capability for testing and other purposes. This capability sends an ICMP packet to every computer connected to the router. A malicious user could send a broadcast message to a router, which will then broadcast it on end. This could potentially flood a network with packets. However, many routers have this 'feature' turned off. Also, there is software that can prioritize packets and give higher priority to TCP packets and lower priority to ICMP packets thus thwarting this type of attack (http://www.packeteer.com).

8) Flash Crowds

A non malicious example of an almost unstoppable distributed denial of service attack is flash crowds. As its name implies, flash crowds occur when numerous legitimate users try to connect to a particular server, eventually flooding it with more requests than it can manage, leaving other users unable to access the server. This often happens following an important news, sports, or entertainment event when millions of legitimate users attempt to access the same site. For example, this is what happened to all major news broadcast web sites right after 9/11. Flash

crowds also occur after what is commonly known as the Slashdot effect (http://slashdot.org is a popular news website).

9) Physical Attack

There is of course the possibility of a physical DoS attack, such as an attacker physically cutting an Ethernet cable, or setting fire to the server building. This paper will not elaborate more on these types of physical attack, nor with the possible defenses. Only DoS attacks based on software assaults will be dealt with.

10) Other exploits

Many other DoS attacks are possible which exploit certain weaknesses in some applications. Some of the most interesting ones are briefly listed below:

1. There have been numerous buffer overflow exploits in which applications crash because they cannot handle incoming messages/instructions.
2. A recent (December 2005) exploit in the newly released Mozilla Firefox 1.5 could prevent further use of the web browser until its *history.dat* file was erased.
3. Viruses and worms can crash computers and deny their legitimate use by users. They also clog up network bandwidth.

There are many tools available that can exploit some of the weaknesses mentioned above and which can simulate an overload of requests. Some of those tools are Tribe, Tribe Floodnet, Trinoo, TFN2K, Stacheldraht, and Shaft, and they are all freely available on the Internet. These are all highly specialized tools aimed at creating a Distributed Denial of Service attack. Most of them work by first infecting a victim computer with

some sort of a Trojan, and then the attacker can remotely control those infected computers to launch a massive DDoS attack on unsuspecting systems. A Trojan is a type of virus that opens a back door in a computer to allow a malicious user to gain complete control of the computer at a later time. It is similar to the Trojan horse in Greek mythology.

There are also other packages that can test the performance of a server. Such examples include hping, fping, nmap, and nessus. Although those tools are meant for security purposes, they can also be used by malicious users to find weaknesses in or attack a system.

Moreover, you can buy a product from a company that will test your network and its performance and ensure that your system is secure. Examples include, but are not limited to, http://www.ixiacom.com, and http://dast.nlanr.net/projects/advisor/.

1.5 Defenses

Someone once said “All programs are buggy”. This means that no program ever written is perfect. Security holes, omitted tests or checks, and other weaknesses will eventually be found and exploited. Although one cannot rely on any programs, protocols, or systems to be completely secure from attacks, there are steps that can be taken to prevent a DoS or DDoS attack.

1) The simplest defense step is of course to patch software as manufacturers or developers make them available. However, some patches might be the

cause for an attack. Some patches, for example in the *Sendmail* program, would automatically reset settings back to default values without alerting the system administrator. These default settings could pose a security risk. Thus, patches should not be applied blindly.

2) Common and logical defenses also have to be applied such as not downloading any attachments or random programs, installing anti-spyware, anti-virus, and firewall programs.
3) IP broadcasting on routers should to be disabled if this feature is not to be used. In some cases, for example a print server broadcasting a “I am printer A. Who wants to use me?” message regularly, this cannot be avoided.
4) Unused services on clients and especially servers should be disabled. Many Linux distributions come with services (SSH, FTP, Samba) turned on.
5) Filtering routers can be used. They could bolster security and help against IP spoofing. Ingress filtering means making sure that all outbound connections from behind the firewall have a valid IP address. Egress filtering means making sure that all inbound connections are coming from the correct place. This does not work if other routers don't use filtering.
6) Although firewalls are good at blocking and filtering packets, and even at protecting the operating system with the newer combined network and OS firewalls, they will only do what they are told and if a system is

compromised, they would not be able to detect or protect it anymore. This is where Intrusion Detection Systems (IDS) come in. They can use signature-based defenses (similar to what most anti-virus software use to detect viruses) to detect and prevent against any known attacks. They can also use heuristic scanning to monitor the processes and the regular usage of a system. If the usage on that system changes significantly, they can alert the system administrator.

7) Network connections are ephemeral and a network administrator has to keep track of what computer is connecting to where, which requests are being sent to the server, and many other network-related tracking. Network monitoring is thus very important for an administrator to detect a premature attack, detect port scanning which are usually a prelude to an attack (although [Vijayan, 2005] shows that this only happens for about 5% of the time), and just to simply monitor network traffic. In some companies, email can be filtered so that important and classified information are not sent outside of the company's network.

8) Many servers are Linux. Since the latter is open source, there are dozens of available Linux distributions which vary from Fedora to EnGarde to Ubuntu to SuSE. Although they all use the same kernel, some distributions have been designed with security from bottom to top. One example is EnGarde (http://www.engardelinux.org) which has been designed with security in mind and geared towards the server end of the market.

9) The honeynet project can be used to track down zombie networks. A zombie (or botnet) is a system that has been infected and is being used by attackers to potential launch massive attacks. A group of infected systems constitutes a zombie network. A honeypot acts like a bait to draw attacks to it so that security experts can analyze the attacks and apply the appropriate defenses.
10) Physical security is also of the utmost importance to protect against attacks.

Although many of the defenses described above are useful against general attacks, not all of them will work specifically for a denial of service attack, and they are even less likely to work against a distributed denial of service attack. A DoS attack against a server is quite easy to defend against. The network administrator can block the IP address of the offending computer, and it is much easier to track down one computer than to try to track down thousands. The simplest defense is of course to just buy more hard disk and memory and processing space and power, and that usually works for bigger companies. However, that approach does not work all the time, and this paper will investigate the inherent problems in a DDoS attack and try to come out with the best possible defense against it.

There are many problems in designing a good defense mechanism against a Distributed Denial of Service attack.

1) One of the most common types of DDoS attacks is to just overload a server with requests. However, that is the exact same definition of a flash

crowd event. It is difficult to distinguish a DDoS attack from a flash crowd event.

2) Although the attack is distributed, that is, many computers participate, defenses are usually solo and not distributed. A distributed defense, sharing the load and information with other servers/computers, might produce a better defense against this type of attack.
3) There is a lack of detailed information available when a DDoS attack occurs. More information would include how many computers were used, where they are found geographically, and what weaknesses, if any, were exploited.
4) Although there is a taxonomy available to classify DDoS attacks and defenses [Mirkovic et al, 2004], there is no current DDoS defense available, therefore there is no benchmark available to determine whether a new product for a DDoS defense is suitable or not.
5) It is very difficult to test software against a DDoS attack. A particular defense could be implemented and be successful when 1000 unique computers are used. However, it might fail if 1001 or 100,000 computers were used.
6) There are economic and social factors involved as to why there is currently no widely accepted DDoS defense mechanisms. Although DDoS attacks are common, many companies do not want to disclose the attacks to public knowledge for fear of bad publicity. Defenses against such

attacks are currently mostly of interest in academic research, not applied security. Morever, DoS attack techiniques are analogous to viruses. Attackers change their signature patterns and defense techniques have to be updated to reflect the latest types of attacks.

There are new defense tools to protect against a DDoS attack. However, there is no benchmark and although the developers claim the tool will help, there is no proof of concept so far. Some examples of those commercial tools are RAZOR (http://www.bindview.com/Services/Razor/), Mu Security (http://www.musecurity.com/), and Melior Inc. (http://www.ddos.com/).

Moreover, there are some freely available tools to detect and prevent the use of the "attacking" tools, such as *gag* – a stacheldraht scanner, and *dds* – a Trinoo/TFN/stacheldraht agent scanner. Other software are also available to track down the source IP, but none of those tools necessarily provide complete protection against a DDoS attack. In fact, there is currently no way to stop such an attack, but there are ways to try to mitigate it.

1.6 Similar Work

When a DDoS attack occurs, there are three main parts in the defense:

1) Detection: The attack has to be detected first. The simplest way to determine that an attack is under way is to check the bandwidth of the network and the usage of computers, as compared to a normal day.

2) Prevention: After the attack has been detected, it has to be stopped or mitigated so that normal use of the services can continue. Although there are currently no known prevention techniques, some steps are to use a firewall and IDS, and also seek the help of the ISP to filter the "evil" packets. This paper will deal mainly in prevention methods.
3) Tracking: The attackers have to be tracked down so that necessary measures can be taken – block them, try to contact the users/owners, or trace them for judicial issues. The IP Source Tracker implemented in most Cisco routers can be useful. Although the attacking computers can be traced back, it does not mean that they can dealt with. International laws might apply and there might language and culture issues.

DDoSVax (http://www.tik.ee.ethz.ch/~ddosvax/) is a research project for developing detection and prevention techniques. The network simulator ns-2 (http://www.isi.edu/nsnam/ns/) can also be used to simulate a real network and try to implement prevention methods.

1.7 This Paper

This paper will deal with simulating a computer network where various clients – both good and evil clients – connect to various servers. Ways in which the server can counteract against the evil clients while providing good service to the good clients will be

researched.

Chapter 2 will explain how the network simulator works and how it is implemented.

Chapter 3 will show the base case against which all experiments are compared.

Chapter 4 will describe the first experiment performed – using a priority queue.

Chapter 5 will detail the second experiment performed – limiting the number of connections per client.

Chapter 6 will describe the third experiment performed – using a server reset.

Chapter 7 will conclude and explain what worked and what did not and why.

Chapter 8 will provide some future research that could be undertaken related to this paper and also provide other guidelines related to Distributed Denial of Service attacks.

Chapter 9 and 10 will deal with references and acknowledgments.

2. Network Simulation

2.1 Introduction

A complete connection-establishment network simulation is designed and built. Only the three way handshake connection establishment process is simulated. The four way handshake connection termination, data transfer, and other negotiations are not simulated. Only ways to prevent malicious clients from establishing a connection will be investigated in this paper.

The network simulation is a discrete event-driven model. It consists of virtual computers, which will make up the basis of the simulation. Each computer can be divided into two categories: server or client. Furthermore, a client can be either a good client or an evil client. The client's purpose is hidden from the server and is only used for statistical purposes.

Moreover, data packets are not actually simulated nor sent, so once a client establishes a connection, there is no actual sending of data or packets or messages, but it is assumed that the data is sent and received and processed, and after a certain amount of time t, whatever needed to be done (for example delivery of a web page, or transferring of a file) has been completed.

When a timeout occurs, the connection is just deleted. There is no retransmission

of messages, or trying to reestablish a connection.

2.2 Example

Each computer in the simulation can be either a client or a server. A client sends requests to the server to create connections. The server just accepts connections from clients. The server is the focus of this paper, as it will become under attack by clients.

Each computer contains a connection table, an inbound message queue, and an outbound message queue.

The connection table holds a list of all the connections – either “in progress” or “established”. The inbound message queue is a First In First Out (FIFO) queue, where the first message in will be the first message out to be processed. This queue is used for messages incoming to the computer only. The outbound message queue is also a FIFO queue, and is only used for outgoing messages from the computer. Both queues hold messages until the computer can process them. Queues are needed because it takes a finite amount of time to process a message to determine what it is, who the sender and receiver are, etc...

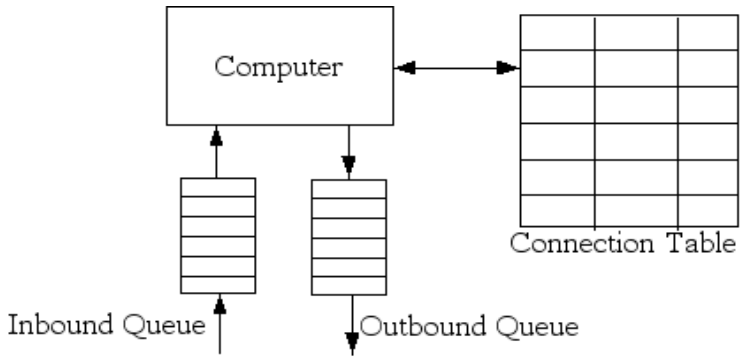

Figure 4: Main Components of the Simulator

There are two types of clients – evil and good. Evil clients act exactly like good clients except in two circumstances. Evil clients never tell the server that they are done with a connection, and thus when an evil client creates a new connection in the server's connection table, that connection stays in the table for the duration of the simulation. Good clients, on the other hand, reset or terminate every connection they establish. The other difference is that evil clients send more connection requests, and thus more messages, to the server than good clients do.

The following is an example of a simulation run:

- Client A wants to establish a connection with server S. Client A thus creates a new Initial Connect message, M1 and puts it in its outbound message queue.

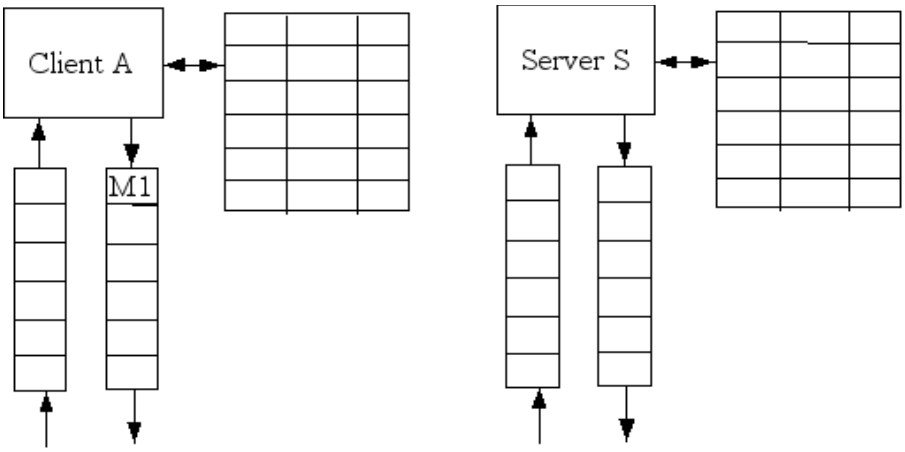

Figure 5: Initial Connect Request

- That message is processed and sent to the server S. A new entry in the client's connection table is created for that new connection. That entry is “in progress”.

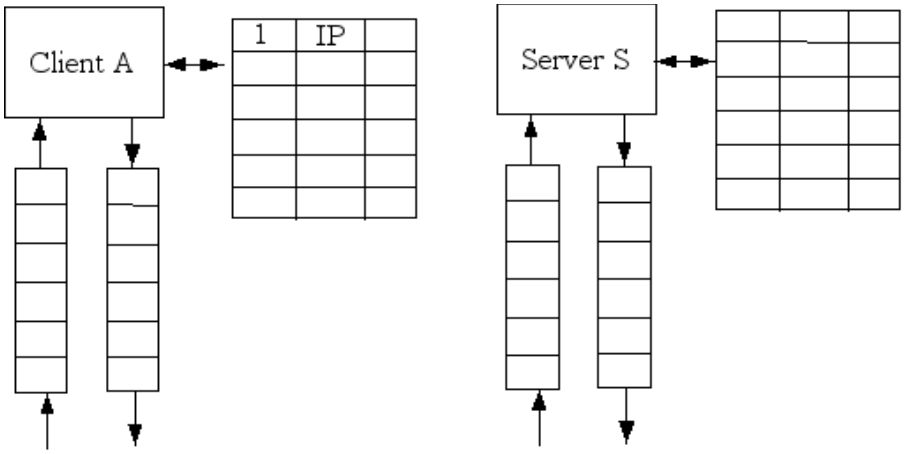

Figure 6: Creation of New Connection for Client

- After a certain amount of time (called the interval time), client A creates another Initial Connect message, M' to the server.
- After the message M1 has been processed and sent, the server S will eventually receive message M1 from client A after a delay time (transmission time).

- Since the server's inbound message queue is empty and its inbound processor is idle, message M1 is processed right away. If the processor was busy, the message would have been put in the queue to be processed later. If the queue is full, the message is just dropped.

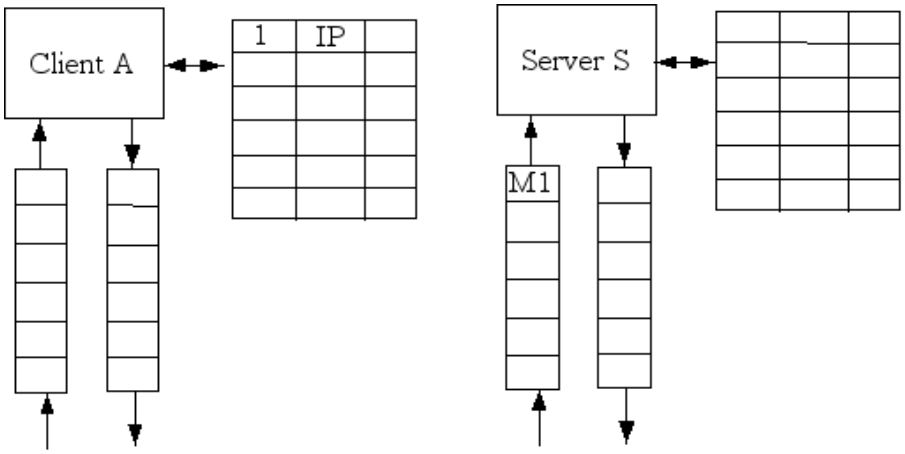

Figure 7: Receive Initial Connect Message

- After the processing time (which is needed to determine what message it is, and who the sender and receiver are), the server attempts to create a new entry in the its connection table. If the table is full, no new connection is created. The new connection is “in progress”. The latter is also marked to be from client A and from index 1 (client A might have multiple connections).

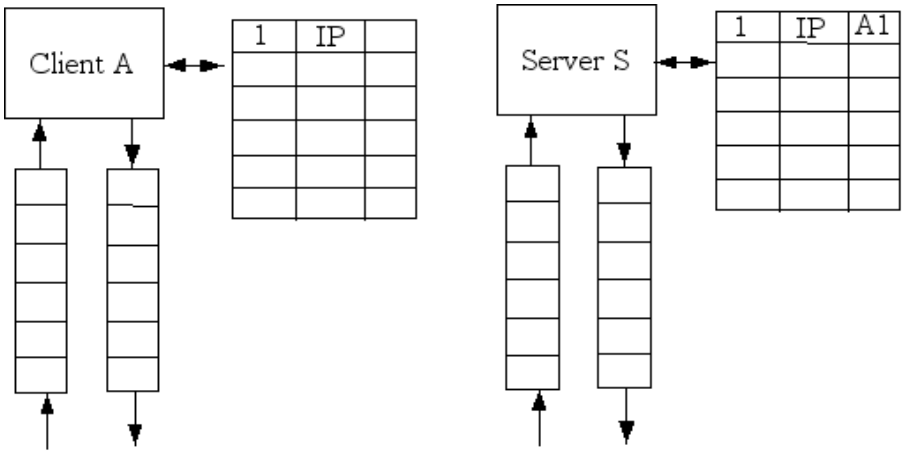

Figure 8: Creation of New Connection for Server

- If a new connection has been created, the server creates a new Ack Client message, M2 to acknowledge the receipt of the initial request from client A. The message is put in the outbound message queue, processed, and sent.

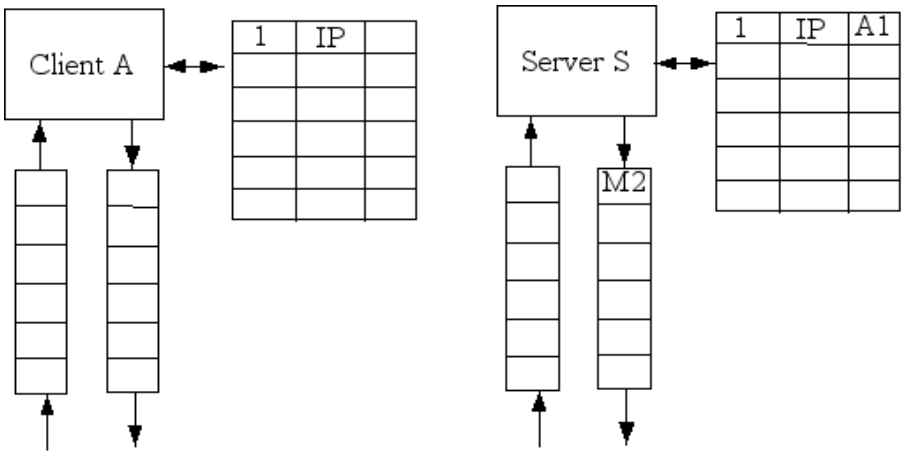

Figure 9: Ack Client Message

- After a delay time, the client receives the message from the server. The message is put in the inbound message queue of the client, and processed. The message has

now been determined to be an acknowledgment. The client's connection table is queried to find the corresponding connection for that message. If it is not found, an error is reported for that client. If the connection is found, it is updated so that the server is marked to be S and the index used on server S is 1. The connection is now “established” on the client's side.

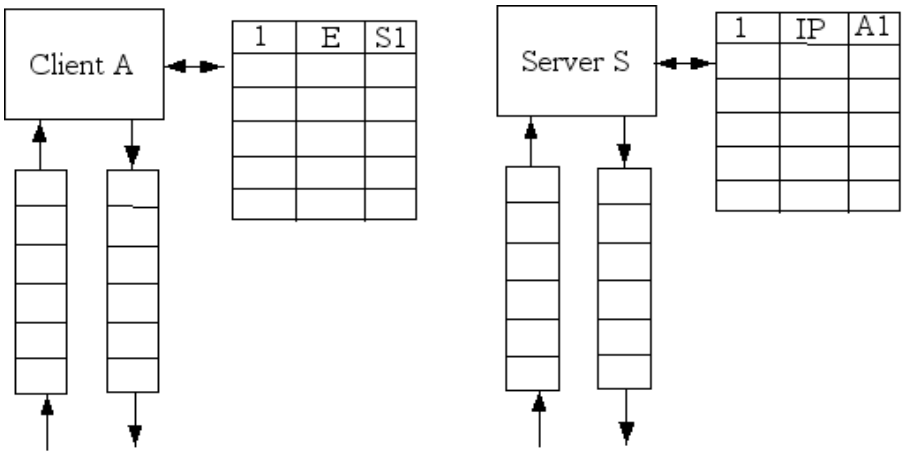

Figure 10: Updating Connection for Client

- The client now creates an Ack Server message, M3 and adds it to its outbound message queue for processing.
- After processing, the message is sent to the server.
- After the reset time (reset time is the time it takes for the Ack Server message to travel from the client to the server, for the server to process the message, and send the appropriate data back. The data could be, for example, a web page), the client creates a Client Reset message M4, adds it to its outbound message queue, processes the message, and sends it to the server.
- After the delay time, the server receives the Ack Server message M3 from the

client. The message is put in the inbound message queue and processed.

- The server has now determined that the message is an acknowledgment to the acknowledgment it sent earlier. It thus tries to find the corresponding connection in its connection table. If the connection is not found, an error is reported for the server. If the connection is found, the latter is now "established". The three-way handshake connection establishment procedure is now over.

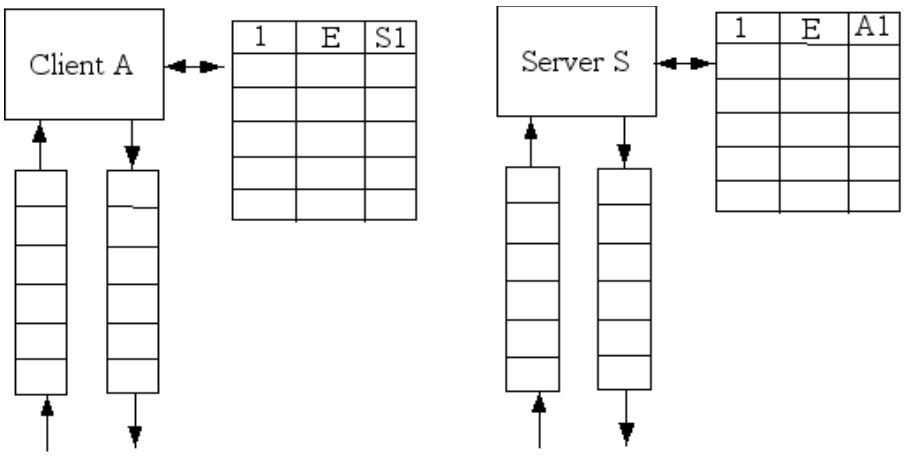

Figure 11: Update Connection for Server

- After the reset time and the delay time, the server receives the Client Reset message M4. The latter is queued in the inbound message queue and processed.

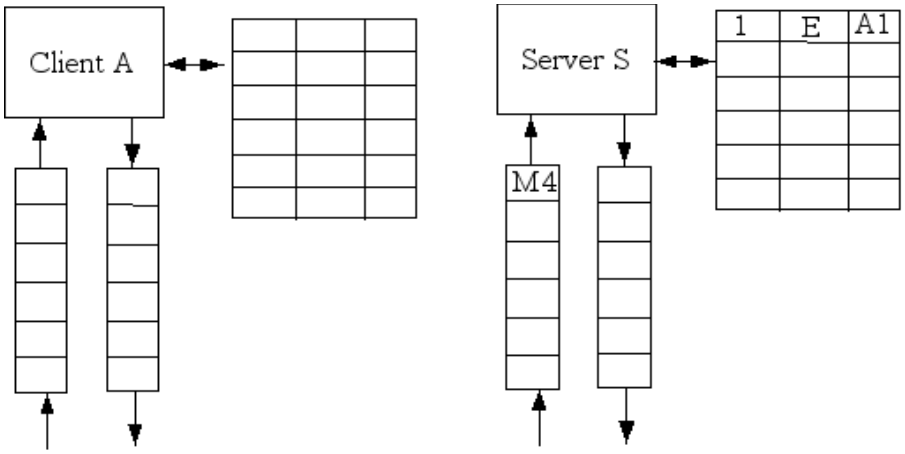

Figure 12: Receive Client Reset Message

- The server has now determined that the message is a reset to an earlier created connection. It thus queries its connection table for the corresponding connection. Once found, that connection is deleted. If not found, an error is reported for the server.

The four-way handshake connection termination procedure is not simulated because this paper is only interested in connection establishments. Thus, a reset without any acknowledgements is sufficient.

A timeout is also scheduled after a new connection is created. It is discarded if the connection is updated/established or removed. If the timeout does occur, the connection in the connection table is deleted. No reset or other message is sent to indicate to the other computer that the connection has timed out.

2.3 Goals

The main goal of this simulation is to keep the server running even while under attack from a distributed network of malicious clients, and also being available to good clients. Evil clients can also be blocked or prevented from connecting to the server.

To achieve this goal, a set of parameters has to be well-defined. As mentioned in Chapter 2.2, there are a lot of variables. The real-world or at least a good approximation value of those variables need to be set, and if needed, tweaked to achieve a better result. Moreover, a number of good clients will be connecting to a server, so that the latter starts to get busy. Evil clients are then added into the simulation, and defense techniques developed to still try to achieve the same availability and dependability as when the simulation was without any evil clients. The result would be to try to maximize the number of good clients serviced, while trying to minimize the number of evil clients serviced. Since the server does not know which clients are good or evil, it will have to act upon what it knows already.

2.4 Technical Details

2.4.1 Software Used

The whole network simulator was built from the ground up using Java. IntelliJ IDE (Integrated Development Environment) by JetBrains was used to improve efficiency, easier debugging, decrease development time, and make refactoring easy. Moreover, test-first programming was applied, as unit tests were developed first before the actual code is written. JUnit was used for the unit tests. This is to ensure that each and every piece of code is working as it should. An automated end-to-end test was not implemented, but the simulation was assumed to be working correctly by going through each event one at a time and making sure that the simulator was doing what it was supposed to do.

2.4.2 Time Units

Since Java is very bad at dealing with decimals, the time is not represented as a double, but rather as an integer. Since the simulator produces a discrete event-driven simulation, each event happens at discrete time and is processed sequentially. Even if two events happen to occur at exactly the same time, the first one will be dealt with first, then the second one. Therefore, it is acceptable to use integers as time units instead of portraying real time in seconds or minutes. However, 10,000 time units is equal to 1 second.

2.4.3 Events

A heap is used to store all the different events, arranged in chronological order. The whole simulation is event-driven, not time-driven. Therefore, there must always be an event in the heap. All clients create new events to establish a connection with a server. The end of the simulation is indicated by a Terminate event. The seven main types of events are Initial Connect, Ack Client, Ack Server, Reset, Timeout, Terminate, and Measure events. Additionally, the first four are further divided into four events:

- Create – The event has just been created and added to the outbound message queue. When the message is to be processed, a Send event is scheduled on the heap.
- Send – The message related to the event is being processed, and a Receive event is scheduled on the heap.
- Receive – The message related to the event has just arrived at the destination and is added to the inbound message queue. When the message is ready to be processed, an End event is scheduled on the heap.
- End – The message related to the event is being processed at the receiver's side. Depending on what event that is, further events can be scheduled.

Therefore, all the events possible in the simulation are:

- Initial Connect – Create, Send, Receive, End

  A client wants to establish a connection with a server, and sends an initial connect

request. This event can only be created by a client and sent to a server.

- Ack Client – Create, Send, Receive, End

  A server has received an initial connect request from a client and sends an acknowledgment. This event can only be created by a server and sent to a client.

- Ack Server – Create, Send, Receive, End

  A client has received the acknowledgment from the server and sends back its own acknowledgment. This event can only be created by a client and sent to a server.

- Reset – Create, Send, Receive, End

  A reset event can be created by either a client or a server and sent to the opposite computer (client to server or server to client). Its function is to delete the connection from the sender and indicate that it wants the connection deleted at the receiver.

- Timeout

  A connection has timed out in a connection table. The latter could belong to either a server or a client, the connection is just deleted from the connection table.

- Terminate

  The end of the simulation.

- Measure

  Regular events that take a snapshot of the simulation. It is used for statistics gathering.

2.4.4 Distributions

In the simulation, just like the real world, it must be really a coincidence for two things to happen at exactly the same time. For example, a message sent from a client to a server at time t would take f seconds, but a second message sent from a client to a server at the same time t might take g seconds, where f is not equal to g.

Most time variables follow an exponential random distribution, given by

*(- log (r) * mean)*, where mean is the mean time provided, and r is a random number and 0 <= r < 1

This returns a double (or a decimal number). The extra decimals can be truncated to return just an integer.

2.4.5 Parameters

As mentioned before, the simulator requires a lot of variables. All the different parameters that can be changed in the simulator is given below.

General Configuration		
measurementInterval	100,000	The Measurement Interval (10 seconds)
terminate	18,000,000	The Termination Time (30 minutes)
verbose	false	Whether to go into verbose/debugging mode
delayTime	20,000	The mean delay/transmission time (2 seconds)

Server Configuration		
numServers	1	The number of servers
computeSendTimeServer	10	The time to process an outgoing message
computeReceiveTimeServer	10	The time to process an incoming message
connectionTableSizeServer	250	The size of the connection table
timeoutServer	300,000	The timeout value (30 seconds)
inQueueSizeServer	500	The size of the inbound message queue
outQueueSizeServer	500	The size of the outbound message queue
Good Clients Configuration		
numClientsGood	1	The number of good clients
computeSendTimeClientGood	10	The time to process an outgoing message
computeReceiveTimeClientGood	10	The time to process an incoming message
connectionTableSizeClientGood	50	The size of the connection table
timeoutClientGood	300,000	The timeout value (30 seconds)
inQueueSizeClientGood	100	The size of the inbound message queue
outQueueSizeClientGood	100	The size of the outbound message queue
intervalTimeClientGood	200,000	The interval time between initial connect requests (20 seconds)
resetTimeClientGood	600,000	The time after which a client reset message is sent (60 seconds)
Evil Clients Configuration		
numClientsEvil	0	The number of evil clients
computeSendTimeClientEvil	10	The time to process an outgoing message
computeReceiveTimeClientEvil	10	The time to process an incoming message
connectionTableSizeClientEvil	5,000	The size of the connection table
timeoutClientEvil	300,000	The timeout value (30 seconds)
inQueueSizeClientEvil	100	The size of the inbound message queue
outQueueSizeClientEvil	100	The size of the outbound message queue
intervalTimeClientEvil	40,000	The interval time between initial connect requests (4 seconds)

resetTimeClientEvil	-1	No client reset is sent

Table 1: Parameters

Even though all the variables can be changed for every simulation run, most of them are fixed because they are the real values used in the different servers currently in use around the world. Macalester College's (http://www.macalester.edu) web and email server's settings were obtained from the current network administrator, and those values are used. A more detailed description of each parameter and what the default or set-in value is is given below.

- measurementInterval = 100000

  Every 100,000 time units (10 seconds), a measure event is scheduled which takes a snapshot of the current state of the message queues and connection table of the servers. Since each simulation is run for 18,000,000, a total of 180 measurements are taken which are deemed sufficient.

- terminate = 18000000

  18,000,000 equals to 30 minutes and that time is neither too short when nothing exciting happens nor too long when the simulation will take too long and not yield any further interesting results.

- verbose = false

  If verbose is true, then each event's occurrence will be displayed along with its information. Unless the simulation has to be debugged, only the output of the results is needed, which is the statistics gathered during the simulation run.

- delayTime = 20000

  This is the mean time it takes a message to travel from a computer (either client or server) to another computer (either client or server). This is 2 seconds in real time and it follows the exponential random distribution aforementioned. A mean time of two seconds is thought to be about analogous to the time it would take in the real world.

- numServers = 1

  Although the simulation could have been run using more servers, only one is needed because the results would have been the same for the different servers. If a more real-world model has been simulated, then more servers would have been needed, but for the simple simulation for this thesis, one server is sufficient.

- ComputeSendTimeServer = 10

  The compute send time is the time it takes the server to process an outgoing message and send it on its way to the client. 10 time units is 1 millisecond. Although no information could be obtained about how fast a computer or router takes to process a message, 1 millisecond has been correct for the simulation. The compute send time is the same for all clients and all servers.

- ComputeReceiveTimeServer = 10

  The compute receive time is the time it takes a server to process an incoming message and determine what to do next with it. The compute receive time is the same for all clients and all servers.

- connectionTableSizeServer = 250

The Macalester College's email and web server both use a maximum number of connections of 250.

- timeoutServer = 300000

  The timeout value for a connection is 30 seconds. If an ack is received, then the timeout event is removed from the events heap. Else, the timed-out connection is removed from the connection table. Once a connection has been established, it cannot be timed out. The Macalester College's email server has a timeout value of 30 seconds.

- inQueueSizeServer = 500

  500 messages can be received at one time. Since the size of the connection table is 250 and it takes only 1 millisecond to process a message, a message queue of size 500 is deemed to be more than enough. Moreover, in a real server, each data message (for example a web page hit) usually takes only one buffer space in the queue.

- outQueueSizeServer = 500

  The outbound message queue is the same as the inbound message queue. Usually, the inbound message queue is the one that receives the most messages, especially during an attack, while the outbound message queue will usually not get full because if the server can process a message and create a new connection, the outbound message queue should not be too overloaded.

- numClientsGood = Variable

  This is a variable and indicates the number of good clients in the simulation.

- computeSendTimeClientGood = 10
- computeReceiveTimeClientGood = 10
- connectionTableSizeClientGood = 50

  A good client can create a maximum number of 50 connections at any one time. This should be more than enough for this simulator.
- timeoutClientGood = 300000

  The timeouts for connections for the clients are also the same as for the server.
- inQueueSizeClientGood = 100

  Since the maximum number of connections is 50, a queue size of 100 is deemed to be plentiful.
- outQueueSizeClientGood = 100

  The outbound message queue is the same as the inbound message queue.
- intervalTimeClientGood = 200000

  A good client tries to establish a new connection with the same server on a mean time of 20 seconds. This time is arbitrary and was chosen because the average user on an average computer will be trying to establish a new connection on an average of 20 seconds. Moreover, since the simulation is run for 30 minutes, this will give a total of about 90 connections per good client. This time follows the exponential random distribution mentioned above.
- resetTimeClientGood = 600000

  Since a good client send an initial connect request about every 20 seconds, after a mean time of 60 seconds, it will send the reset. This will give on average 3

connections per client in the server's connection table. The reset is always sent 60 seconds (mean time) after the acknowledgment back to the server from the client is sent.

- numClientsEvil = Variable

  The number of evil clients in the simulation. This can be varied as needed.

- computeSendTimeClientEvil = 10
- computeReceiveTimeClientEvil = 10
- connectionTableSizeClientEvil = 5000

  Since an evil client tries to establish a lot of connections and those are never reset, a table size of 5000 is chosen. Although this might be on the extreme side, it is to be sure that the evil client's connection table is never full (else what's the point of it being evil?).

- timeoutClientEvil = 300000

  The timeout is still set to 30 seconds.

- inQueueSizeClientEvil = 100

  Although a big connection table is needed, a message queue size of 100 is sufficient since each message takes about 1 millisecond to process.

- outQueueSizeClientEvil = 100
- intervalTimeClientEvil = 40000

  As will be discussed later in chapter 3, the optimal[1] value is 4 seconds.

1 Optimal means a value such that it is not extreme on either the high side or the low side. In this case, optimal means that the evil clients will send an initial connect request at a rate that will not completely overwhelm a server nor at a rate too low that the evil clients do not send enough messages to overload a server.

- resetTimeClientEvil = -1

  An evil client never sends a reset message. Therefore, once an evil connection has been established in the server's connection table, it will never be removed.

Although there are a lot of parameters, only two of them are variables – number of evil clients and number of good clients.

The times that follow an exponential random distribution are *delayTime*, *intervalTime*, and *resetTime*.

2.5 Sample Output

A sample output given the parameters from Chapter 2.4.5, for both verbose and non-verbose mode, is given in Appendix A.

2.6 Useful Statistics

A total of 39 types of statistics are gathered for each simulation run. However, not all 39 are really useful, depending on the runs and test cases. A complete list of all the measurements collected is given in Table 2. Most of them are self-explanatory. A description is given for any ambiguous ones.

Time Independent Data	***Data collected each time such an event occurs***
Total number of inbound total messages	
Total number of inbound good messages	
Total number of inbound evil messages	
Total number of inbound dropped total messages	
Total number of inbound dropped good messages	
Total number of inbound dropped evil messages	
Total number of outbound total messages	
Total number of outbound good messages	
Total number of outbound evil messages	
Total number of oubtound dropped total messages	
Total number of outbound dropped good messages	
Total number of outbound dropped evil messages	
Total number of connections	This reprensents the total number of connections attempted (whether they are created or not)
Total number of good connections	
Total number of evil connections	
Total number of dropped connections	
Total number of dropped good connections	
Total number of dropped evil connections	
Total number of established connections	
Total number of established good connections	
Total number of estbalished evil connections	
Total number of reset connections	
Total number of reset good connections	
Total number of reset evil connections	
Total number of timeouts	
Total number of good timeouts	

Total number of evil timeouts	
Total number of errors	
Total number of good errors	
Total number of evil errors	
Time Dependent Data	***Data collected at every measurement interval***
Average size of inbound message queue with all messages	
Average size of inbound message queue with good messages	
Average size of inbound message queue with evil messages	
Average size of outbound message queue with all messages	
Average size of outbound message queue with good messages	
Average size of outbound message queue with evil messages	
Average size of connection table with all connections	
Average size of connection table with good connections	
Average size of connection table with evil connections	

Table 2: Measurements

Measurements are collected only at the server side since we are only interested in analyzing the performance of the server.

3. Simulation Runs

3.1 Test Runs

All the runs are performed using the parameters given in Chapter 2.4.5. When the parameters are not the same as shown in that chapter, the parameters used will be shown. The statistics are gathered as in Chapter 2.6.

Each simulation test case is run ten times to avoid any statistical fluctuations. Two outliers are removed, and the average of the remaining runs are taken. That average is used as the final measurement for that particular run.

3.2 Goals

The main goals of this thesis' simulation are to maximize the availability and serviceability of the server with regards to the good clients, while blocking or limiting the attack from the evil clients.

The total number of established good connections, total number of dropped evil connections, average number of connections (to maximize usage of the server), and average number of good connections will all be maximized, while the total number of evil connections, total number of dropped good connections, and average number of evil

connections will all be minimized. The total number of good timeouts, the total number of good errors, and the total number of reset good connections should not vary too much from the base case.

3.3 Base Case

3.3.1 What is a Base Case?

There will be two base cases, but they both have the same definition. A base case in this paper means the very basic statistics gathered using a set configuration of parameters. The base case will be used for comparison to determine if any features/improvements/new defense techniques/changes in parameters help the goals mentioned in Chapter 3.2.

3.3.2 Base Case 1 – Number of Good Clients

In this case, there are no evil clients, and the parameters are as shown in Chapter 2. The number of good clients is varied, until the load on the server is right. “Right” meaning that the server is busy enough and is neither swamped nor has nothing to do for periods of time (this can be deduced by the amount of free space in the connection table).

It is possible that the server will not be able to service all requests even if there are only good clients. This is acceptable.

Number of Good Clients	1	4	16	32	64	80	96	128
# of Inbound Messages	267	1051	4146	8577	17031	20233	22193	25221
# of Good Inbound Messages	267	1051	4146	8577	17031	20233	22193	25221
# of Evil Inbound Messages	0	0	0	0	0	0	0	0
# of Dropped Inbound Messages	0	0	0	0	0	0	0	0
# of Dropped Good Inbound Messages	0	0	0	0	0	0	0	0
# of Dropped Evil Inbound Messages	0	0	0	0	0	0	0	0
# of Outbound Messages	91	355	1399	2898	5751	6691	6974	7059
# of Good Outbound Messages	91	355	1399	2898	5751	6691	6974	7059
# of Evil Outbound Messages	0	0	0	0	0	0	0	0
# of Dropped Outbound Messages	0	0	0	0	0	0	0	0
# of Dropped Good Outbound Messages	0	0	0	0	0	0	0	0
# of Dropped Evil Outbound Messages	0	0	0	0	0	0	0	0
# of Attempted Connections	91	355	1399	2898	5752	7104	8508	11367
# of Good Attempted Connections	91	355	1399	2898	5752	7104	8508	11367
# of Evil Attempted Connections	0	0	0	0	0	0	0	0
# of Dropped Connections	0	0	0	0	1	413	1534	4308
# of Dropped Good Connections	0	0	0	0	1	413	1534	4308
# of Dropped Evil Connections	0	0	0	0	0	0	0	0
# of Established Connections	89	349	1377	2864	5697	6626	6902	6990
# of Established Good Connections	89	349	1377	2864	5697	6626	6902	6990
# of Established Evil Connections	0	0	0	0	0	0	0	0
# of Reset Connections	86	342	1351	2788	5542	6452	6727	6809
# of Reset Good Connections	86	342	1351	2788	5542	6452	6727	6809
# of Reset Evil Connections	0	0	0	0	0	0	0	0
# of Timeouts	0	0	0	0	1	1	1	1
# of Good Timeouts	0	0	0	0	1	1	1	1
# of Evil Timeouts	0	0	0	0	0	0	0	0
Number of Errors	1	5	18	28	41	51	57	54
Number of Good Errors	1	5	18	28	41	51	57	54
Number of Evil Errors	0	0	0	0	0	0	0	0
Average Size of Inbound Queue	0	0	0	0	0	0	0	0
Average Size of Inbound Queue (Good)	0	0	0	0	0	0	0	0
Average Size of Inbound Queue (Evil)	0	0	0	0	0	0	0	0
Average Size of Outbound Queue	0	0	0	0	0	0	0	0
Average Size of Outbound Queue (Good)	0	0	0	0	0	0	0	0
Average Size of Outbound Queue (Evil)	0	0	0	0	0	0	0	0
Average Size of Connection Table	2	12	48	99	199	233	240	245
Average Size of Connection Table (Good)	2	12	48	99	199	233	240	245
Average Size of Connection Table (Evil)	0	0	0	0	0	0	0	0

Table 3: Base Case 1 - No evil clients

As shown in Table 3 and Figure 13, there are no dropped connections until there are 64 good clients. Moreover, there are no dropped messages. The errors are due to a reset from a client to the server arriving before the acknowledgment. The number of errors increases as the number of good clients increases because if one good client causes one error to occur, two good clients should cause two errors to occur. It is not exactly linear due to statistical fluctuations. Moreover, the number of reset connections plus the number of errors is approximately equal to the number of established connections, which by the definition of an error, is correct. The number of incoming messages does not increase linearly when there are dropped connections because an acknowledgment or reset message counts towards the number of inbound messages. It is also interesting to note that the number of dropped connections plus the number of outbound messages is equal to the number of connections. Each connection that was not dropped should create one and only one outbound message – the acknowledgment back to the client.

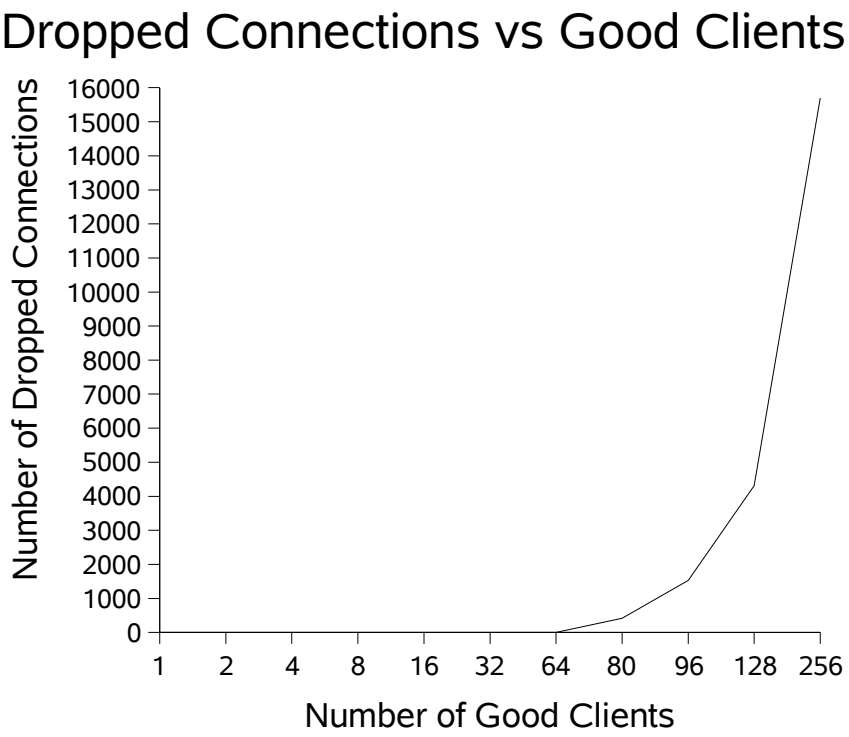

Figure 13: Number of Dropped Connections vs Number of Good Clients

The optimal value for the number of good clients is 80. With 80 good clients and no evil clients, the server will be busy with requests, as shown by almost full connection table on average (233/250 = 93.2%), but not so busy that everything grinds to a halt because only 5.8% (413/7104) of the total number of connections are dropped.

3.3.3 Base Case 2 – Evil Clients come in!!

80 good clients is a base case, but evil clients need to come in the picture. The

second base case is where there is only one evil client. Only one evil client is used because you need to start somewhere. As will be seen later, more evil clients are added. The optimal value of the mean interval time of initial connect requests for evil clients will now be determined using 80 good clients and one evil client. The optimal value with regards to the evil clients would of course be the lowest value possible, with clients bombarding a server with messages as often as possible, and the optimal with regards to the server would be the highest value possible so that the server receives very few requests and is thus available for other requests as long as possible. Thus the optimal value would be one to strike the balance between those two extremes.

As shown in Figure 14 and Appendix A, the optimal value chosen for the mean interval time for evil clients is 40,000 time units, which is 4 seconds. Therefore, on average, an evil client will send an initial connect request every four seconds to the server. Four seconds was chosen because out of 7124 good connections, 3943 (55.3%) good connections were dropped. The average size of the connection table is also 243 (97.2%), with about half of them good and half evil. From Figure 14, it can be seen why the mean time of four seconds was chosen – the server is neither too busy that nothing can really help nor too idle that no noticeable improvements can be performed.

The number of established good connections decreases exponentially and the number of dropped good connections increases logarithmically as the mean interval time decreases. This is due to the exponential distribution. This causes a lot of good clients being denied service, while the single evil client hogs up all the resources. Moreover, evil clients do not reset their connections, thus the evil connections stay in the server's

connection table for the whole duration of the simulation. The number of errors does decrease because as fewer clients can create new connections, the probability of a reset message arriving before an acknowledgment message decreases.

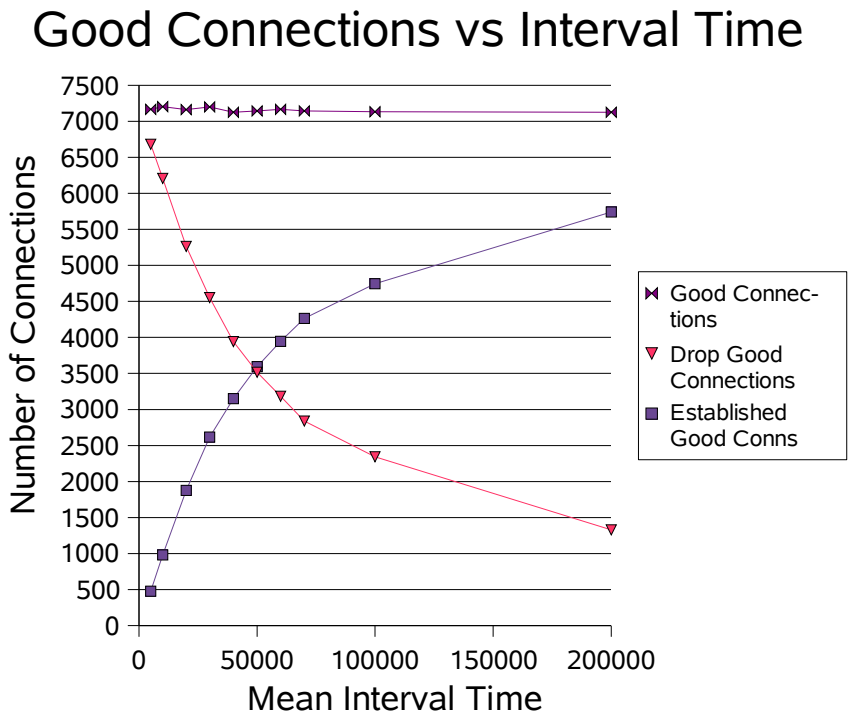

Figure 14: Number of Good Connections vs Mean Evil Interval Time

3.3.4. Message Queue always Empty?

For all the runs completed so far, the average size of the message queues (both inbound and outbound) has always been empty, and there have not been any dropped messages whatsoever.

A reset time of 6 seconds (60000 time units) was used for this run, with no evil clients, and it can be seen from Figure 15 that as the number of good clients increases, the server becomes so overloaded that messages start getting dropped and chaos occurs. The outbound message queue never gets full because it can never happen that there are more outbound messages at any time than the size of the connection table.

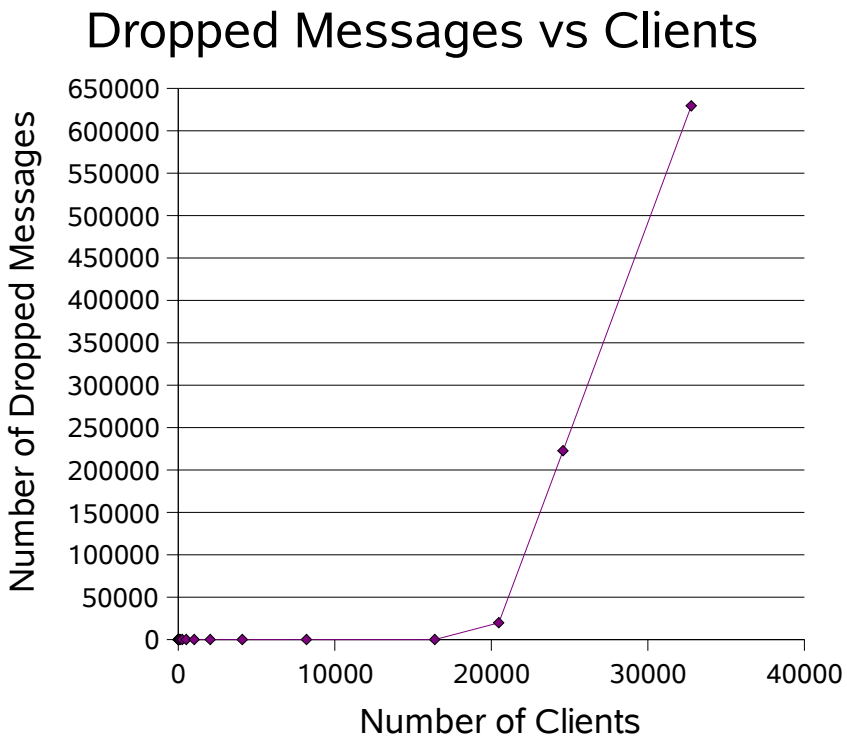

Figure 15: Number of Dropped Messages vs Number of Clients

The number of dropped incoming messages and the average size of the inbound message queue increases exponentially when there are more than 16,000 clients, and everything grinds to a halt. The number of established connections drops from 21311 to 654. The number of timeouts also increases because as messages get dropped, sometimes an acknowledgment would get lost as well.

3.4 Defenses

The parameters provided for the simulation are held constant since these are the real values that some servers in the real world use. The only way to try to improve the serviceability of the server while under attack is to introduce new features in the simulation. The three defense features to be implemented are

- Priority Queue

  Instead of using a first in first out (FIFO) message queue, a priority queue is used instead, to give higher priority to the messages from clients which do not have too many connections in the table at the time.

- Limit the number of Connections

  Each client can only create a certain amount of connections at any time.

- Server Reset

  The server performs a reset when its connection table gets full.

4 Experiment 1 – Using a Priority Queue

4.1 Goals

In the first experiment, a priority queue is used instead of a First In First Out queue for the inbound message queue for the server. Only the inbound message queue is changed since this is the bottleneck as described in the Base Case. All messages will be given a priority based on how many connections they currently have on the server. It does not matter whether a client has 10 in progress connections or 10 established connections – in both cases, the client will be counted as having 10 connections.

A cutoff point (call it x) is used. Clients that have more than x connections will be given a low priority, while clients that have at most x connections in the server's connection table will be given a high priority. Only two priorities are used since there are only two types of clients – good and evil. The priority queue will function as a heap, with highest priority messages at the top and lowest priority messages at the bottom.

The main reason behind using a priority queue instead of a FIFO queue is that evil clients will generally have a lot of connections in the connection table, as compared to the good clients. Thus, all messages from evil clients will end up at the end of the queue. Therefore, good clients are expected to receive better performance and evil clients' messages are blocked or slowed down at the inbound queue. The expectation is that the

number of established good connections would go up, and the number of dropped good connections and established evil connections would go down.

The optimal value of that cutoff point x will be determined to see if any improvements have been made.

4.2 New Parameters

Two new parameters are introduced for this new feature. They are shown in Table 4.

useFIFOQueue	false	Whether or not to use the FIFO queue
NumCutOffPriority	0	The priority cutoff point x to be used. Of course, a negative value does not make sense

Table 4: New Parameters for Experiment 1

4.3 Runs

The simulator was run using the same parameters as mentioned before, except that a priority queue is now used instead. The cutoff point is varied to determine if there is an optimum value.

Cutoff	8	7	6	5	4	3	2	1	0	Base Case
Total Connections	7637	7604	7623	7555	7619	7583	7600	7591	7592	7574
Good Connections	7177	7157	7184	7102	7164	7138	7149	7146	7143	7112
Evil Connections	460	447	439	453	455	445	451	445	450	462
Drop Connections	4284	4258	4186	4209	4220	4104	4193	4206	4210	4237
Drop Good Connections	4029	4015	3947	3960	3970	3860	3944	3961	3965	3983
Drop Evil Connections	254	243	239	249	251	244	249	244	245	254
Total Established Connections	3318	3305	3400	3312	3359	3445	3368	3349	3347	3302
Established Good Connections	3112	3101	3199	3108	3155	3244	3167	3148	3143	3093
Established Evil Connections	206	204	200	204	204	201	201	201	204	209
Total Reset Connections	3103	3096	3187	3096	3149	3229	3157	3135	3132	3087
Reset Good Conns	3103	3096	3187	3096	3149	3229	3157	3135	3132	3087
Average Total Connections	243	243	243	243	243	243	243	243	243	243
Average Good Connections	111	111	115	110	111	114	112	112	111	110
Average Evil Connections	132	132	128	132	131	128	130	130	131	133

Table 5: Using a Priority Queue - 80 Good and 1 Evil Clients

The most important statistics collected for the priority queue are shown in Table 5. The average connection size remains constant throughout all the runs, and the same observation can be made for other test cases (as shown in Appendix A).

In the graphs to follow, the point below the 0 priority cutoff means that no priority queue is used and represents the base case.

From Figure 16, the number of created total connections and the number of created good connections remain more or less constant. Thus, the number of created evil connections also remains constant. The slight decrease at priority cutoff = 5 is due to statistical fluctuations since both the number of created total connections and the number of created good connections decrease.

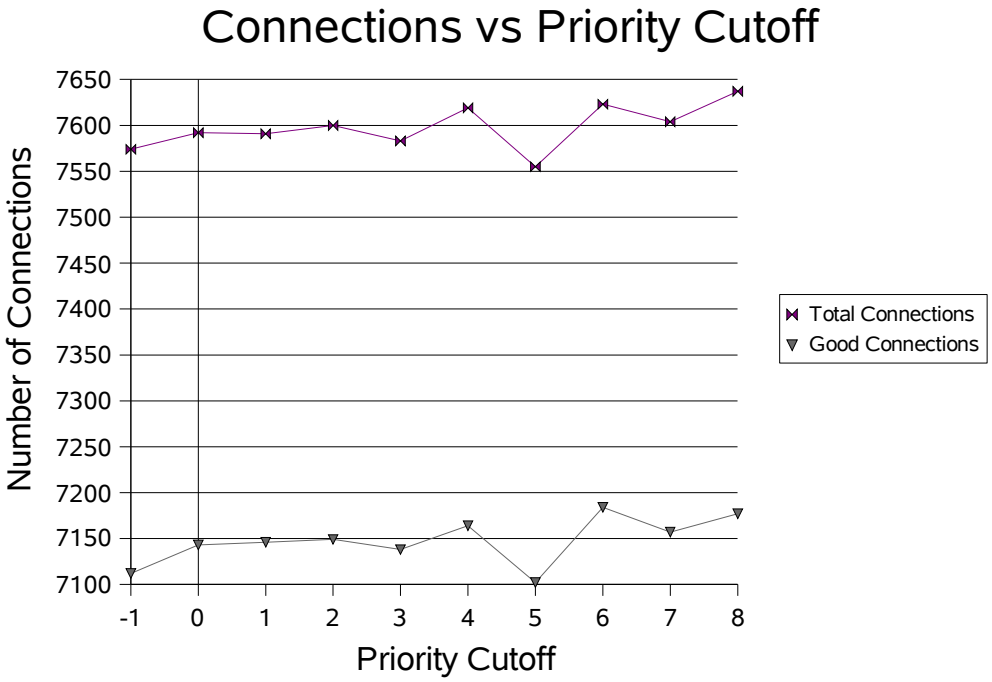

Figure 16: 80 Good Clients and 1 Evil Client (-1 is the base case)

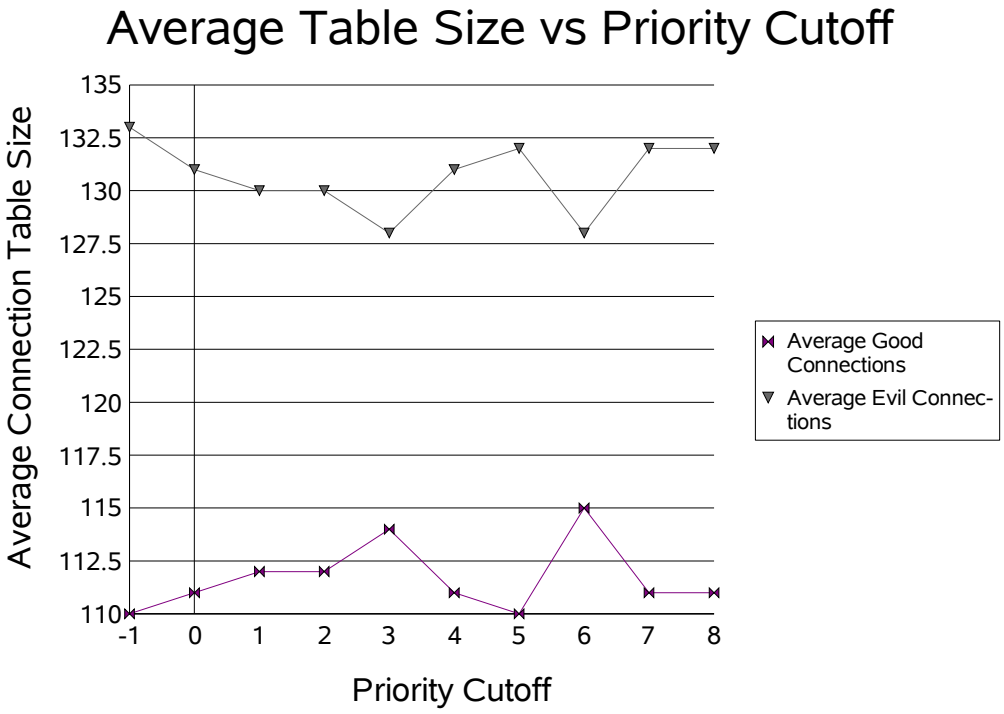

Figure 17: 80 Good Clients and 1 Evil Client (-1 is the base case)

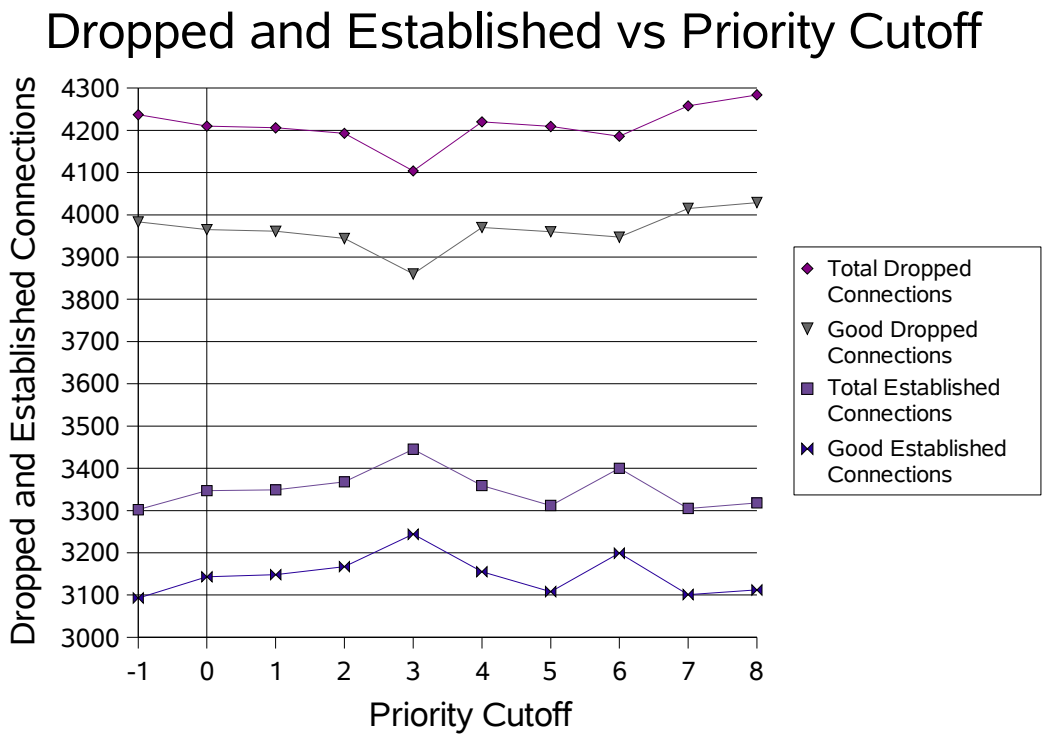

Figure 18: 80 Good Clients and 1 Evil Client (-1 is the base case)

From Figures 17 and 18, it would seem that the optimal cutoff point is at 3 connections. The number of dropped good connections is at the lowest, while the average number of good connections and the number of established connections are at their highest.

However, only an improvement of 2% in the number of dropped good connections is achieved (3983/7112 = 56% to 3860/7138 = 54%), and an improvement of 0.5% in the number of established good connections (3093/3302 = 93.7% to 3244/3445 = 94.2%), and an improvement of 1.6% in the average number of good connections (110/243 = 45.3% to 114/243 = 46.9%).

Statistically, those percentages are not very significant, therefore more evil clients

are added.

80 good clients and 3 evil clients are used in the next test case.

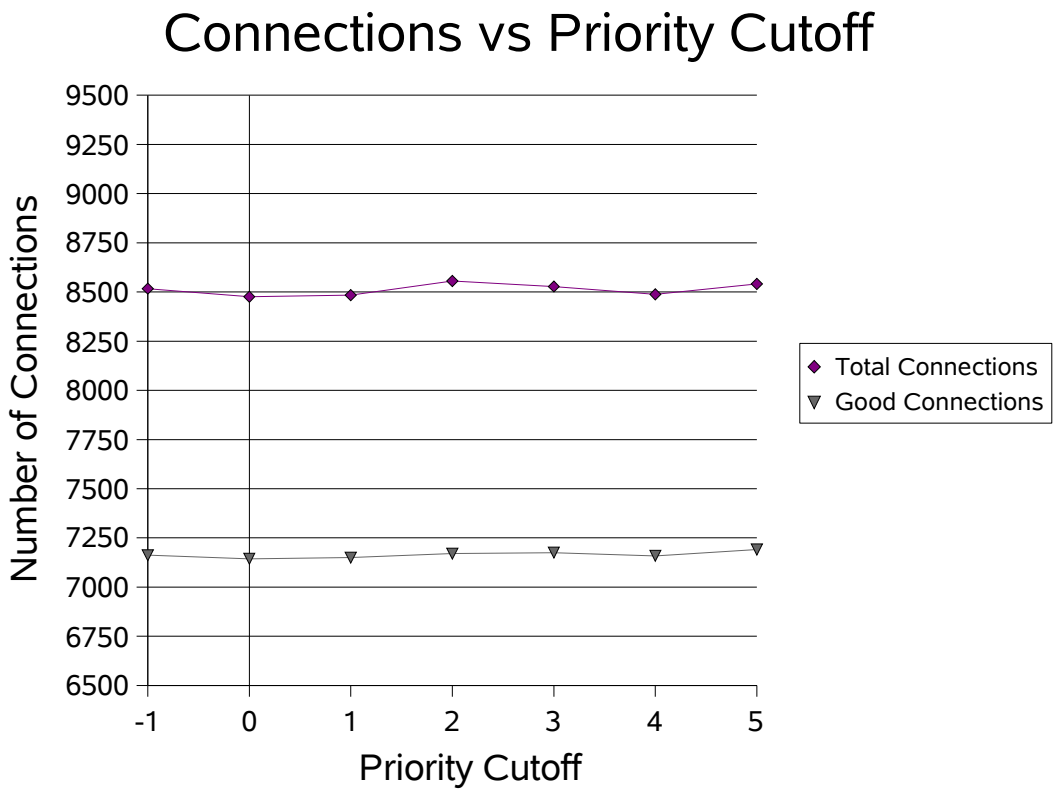

Figure 19: 80 Good Clients and 3 Evil Clients (-1 is the base case)

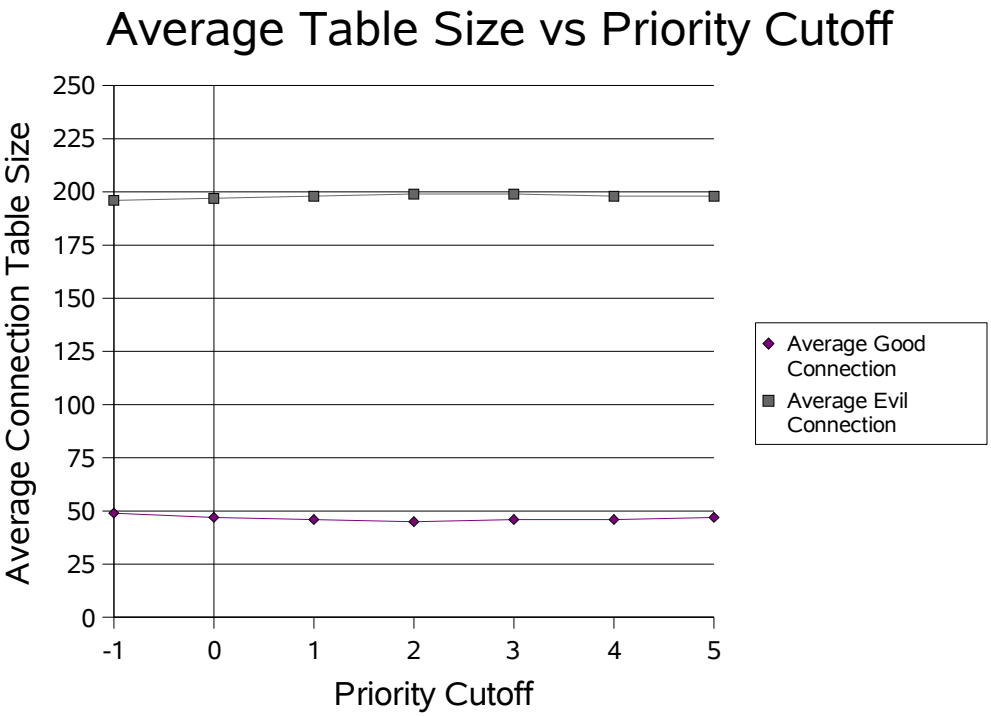

Figure 20: 80 Good Clients and 3 Evil Clients (-1 is the base case)

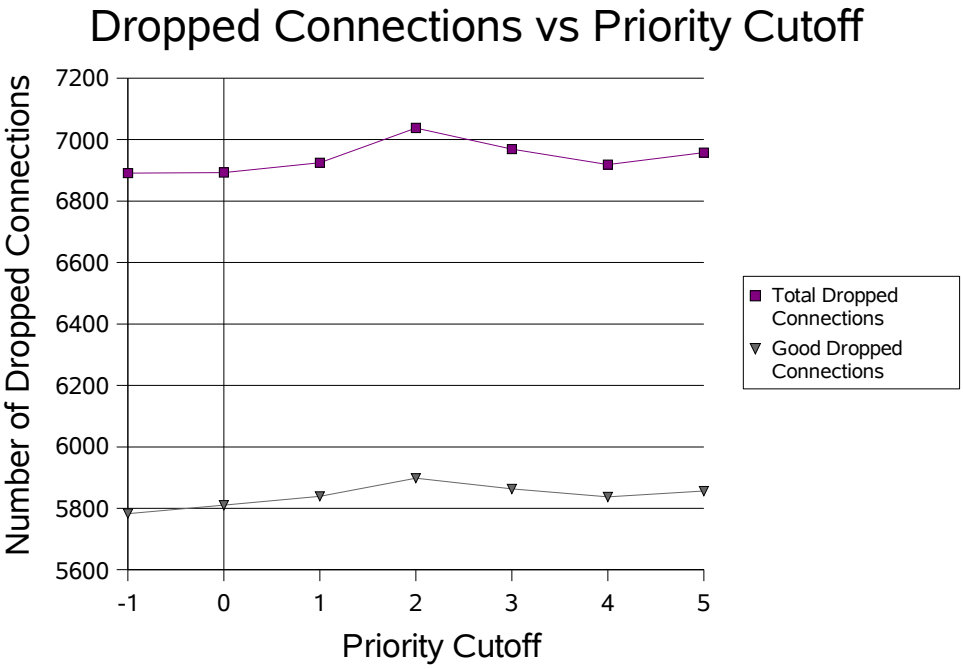

Figure 21: 80 Good Clients and 3 Evil Clients (-1 is the base case)

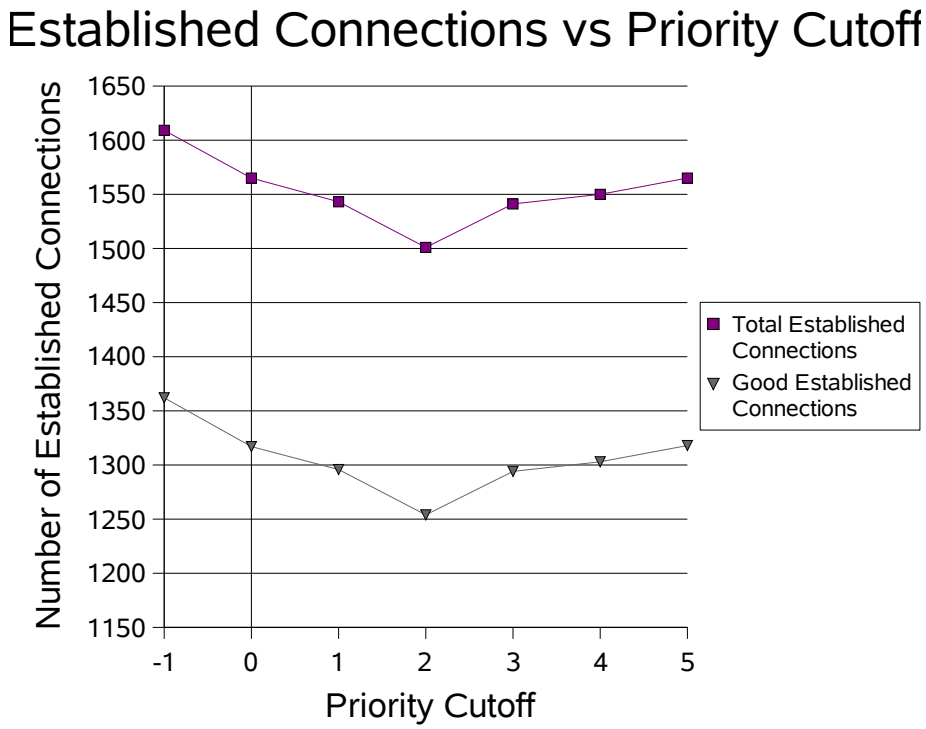

Figure 22: 80 Good Clients and 3 Evil Clients (-1 is the base case)

From Figures 19, 20, 21, and 22, the addition of a priority queue does not seem to help, as shown by the almost horizontal lines in all four graphs.

To prove that a priority queue does not help improve performance of the system, more good clients are added with only 1 evil client. The results are as shown in Figures 23, 24, and 25.

1024 good clients and one evil client interact with one server. The percentage of dropped connections is very high, as expected, since the server just gets overloaded by connection requests. Although there is one evil client, it only has 19 connections on average in the server's connection table. This is because the good clients greatly

outnumber the lonely evil client.

Similar to the previous test case, Figures 25 and 26 show a graph of horizontal lines, indicating that the priority queue does not help improve the performance of the server. On the other hand, Figure 23 shows a display of “zig zag” lines, but these are due to statistical fluctuations.

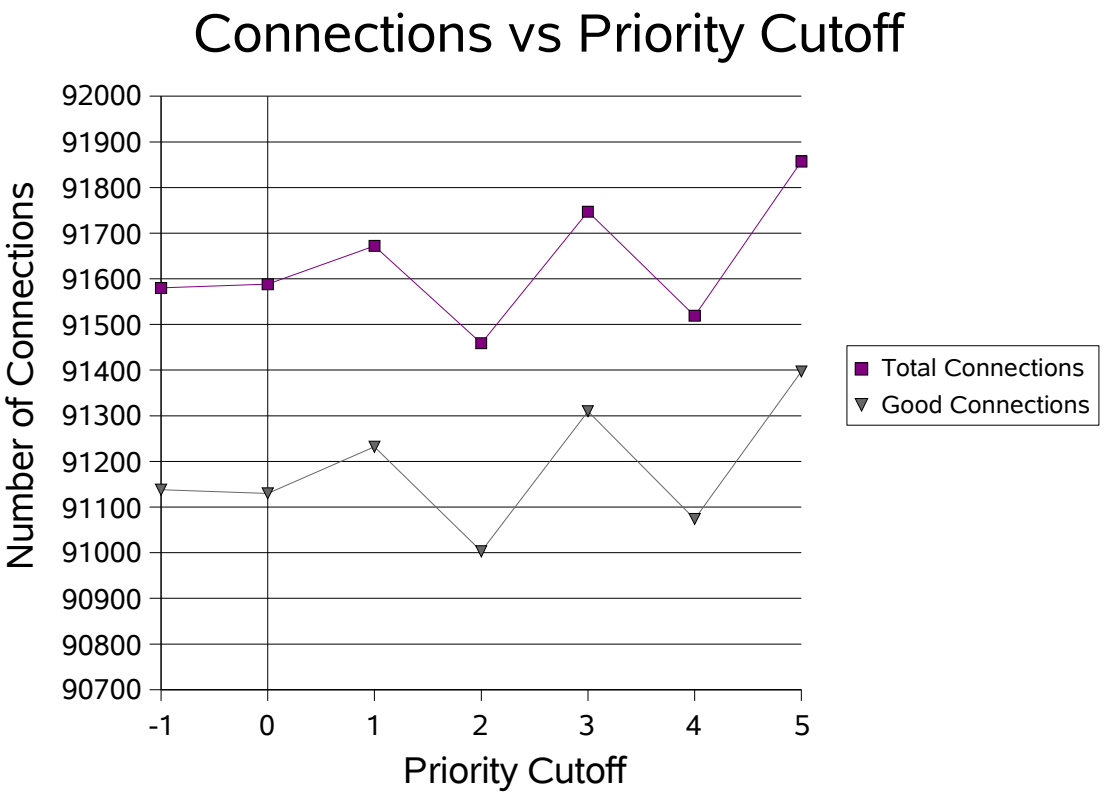

Figure 23: 1024 Good Clients and 1 Evil Client (-1 is the base case)

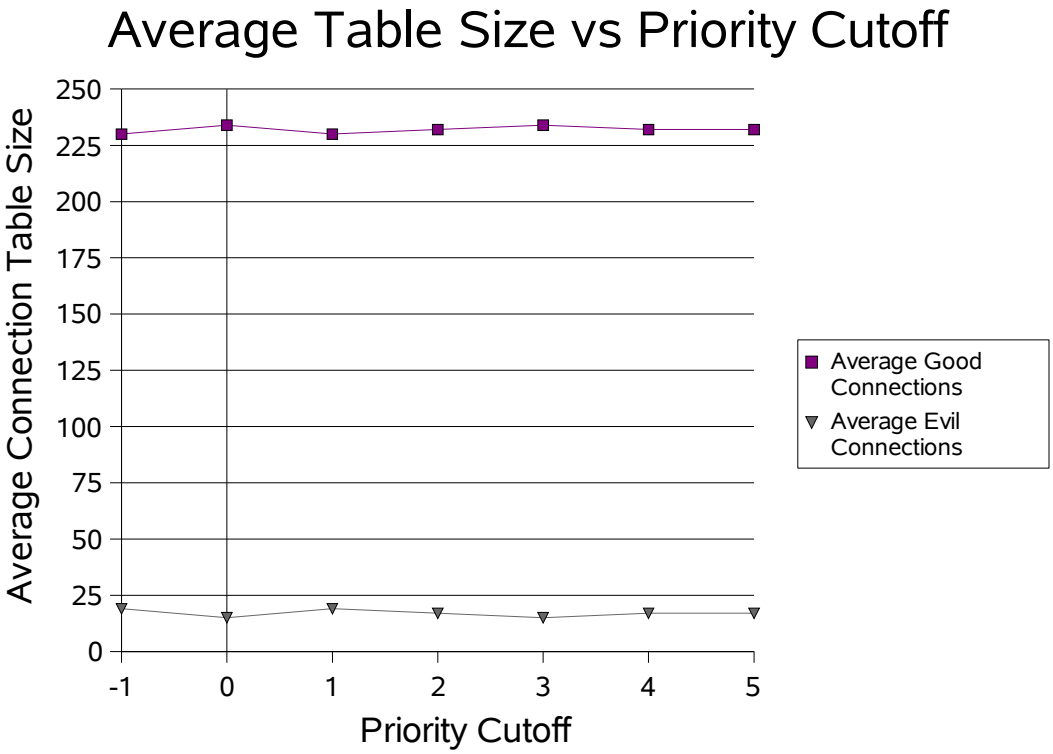

Figure 24: 1024 Good Clients and 1 Evil Client (-1 is the base case)

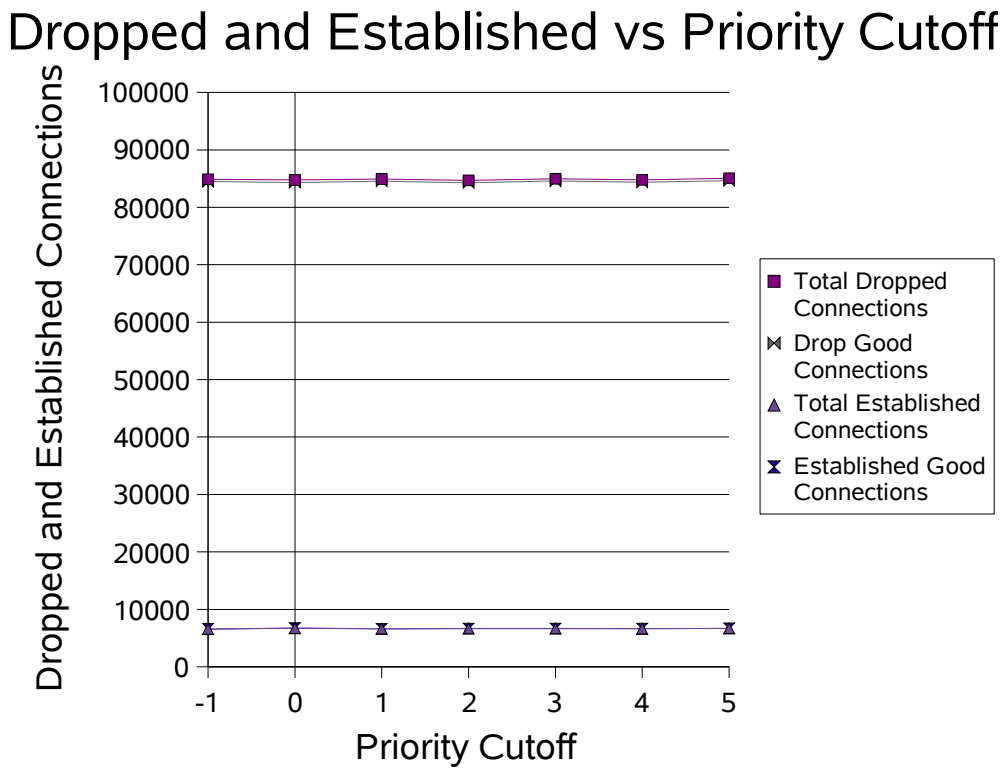

Figure 25: 1024 Good Clients and 1 Evil Client (-1 is the base case)

4.4 Conclusion

As discussed in Chapter 4.3, the priority queue feature did not significantly improve the performance of the server with regards to the good clients. Although intuitively a priority queue should help since all the evil messages are being sent at the back of the queue, the experiment did not work because the evil messages will eventually get to the front of the line and get processed by the server. Processing time of incoming messages is only one millisecond, and even if good clients keep sending messages to the

server, the latter only needs one millisecond to process an evil message, and once a connection from an evil client has been created in the server's connection table, it stays there forever because the evil clients do not send any resets and these connections are never removed. Even though all the evil messages are being pushed back at the end of the queue, once they get to the front, they will get processed by the server and a new entry created in the table. This new entry will never be removed from the connection table, thus the good clients are still being denied access to the server.

Moreover, a message priority queue is not known to be used on any configurations for any servers in the real world, giving further evidence that a priority queue might not work.

We will have to do more rigorous processing to prevent evil clients from getting into the table in the first place.

Next, experiment 2 is going to be implemented and discussed.

5 Experiment 2 – Limit Number of Connections

5.1 Goals

The base case allows a client to create as many connections as needed. This can be harmful and impact performance when an evil client creates dozens of connections, preventing other clients from obtaining a connection space in the table.

Thus, in this second experiment, the number of connections any client can have in the server's table is limited to the limiting value x. Whenever a client tries to create a new connection, the connection table for the server is checked to determine how many connections that client currently has open – both in progress connections and established connections. If the limit x is exceeded, that new connection is dropped. Else, a new entry is created in the server's table for that connection. Only the server uses the limit since it is the object under study.

Since evil clients never reset or remove the connections they create, they could potentially create lots of connections in the server's table, thus denying access by the good clients. Limiting the number of connections each client can create will hamper the negative impacts of the evil clients. However, if the limit is too low, that would impact the good clients in a negative way too since they won't be allowed to create more legitimate connections. On the other hand, if the limit is too high, it would not matter that

the limit is being used since no client will ever have that high number of connections in the table.

The optimal value of the limit x will be determined to see if any improvements have been made.

5.2 New Parameter

For this new feature to work, one new parameter is needed. This is shown in Table 6.

limitNumConnPerClient	0	The limit to be used (0 means no limit), and of course, a negative limit does not make sense

Table 6: New Parameter for Limit

5.3 Runs

The simulator was run using the same parameters as mentioned before, but with the new parameter in Chapter 5.2 introduced. The limit is varied to determine if there is an optimal value.

The priority queue is not used since it was shown in Chapter 4 that adding a priority queue instead of a FIFO queue did not help improve performance on the server.

Table 7 shows the runs with 80 good clients and 1 evil client, with the most important statistics.

Limit	10	9	7	6	5	4	3	2	1	Base Case
Total Inbound Messages	20407	20475	20320	20548	19756	18620	16618	14094	11032	14134
Good Inbound Messages	19955	20010	19866	20094	19298	18173	16160	13635	10596	13486
Evil Inbound Messages	452	465	454	455	458	447	457	459	437	648
Total Outbound Messages	6542	6572	6520	6548	6235	5604	4583	3303	1742	3414
Good Outbound Messages	6532	6563	6513	6542	6230	5600	4580	3301	1741	3210
Evil Outbound Messages	10	9	7	6	5	4	3	2	1	204
Total Connections	7581	7588	7531	7701	7526	7622	7626	7612	7615	7559
Good Connections	7139	7132	7084	7253	7073	7179	7172	7155	7179	7115
Evil Connections	442	456	447	449	453	443	454	457	436	444
Total Dropped Connections	1039	1016	1010	1154	1291	2018	3043	4309	5873	4145
Dropped Good Connections	607	569	570	711	843	1580	2592	3854	5439	3905
Dropped Evil Connections	432	447	440	443	448	439	451	455	435	240
Total Established Connections	6478	6512	6453	6486	6166	5549	4535	3265	1714	3386
Established Good Connections	6468	6503	6446	6480	6161	5545	4532	3263	1713	3182
Established Evil Connections	10	9	7	6	5	4	3	2	1	204
Total Reset Connections	6299	6328	6283	6311	6008	5405	4418	3186	1680	3164
Reset Good Connections	6299	6328	6283	6311	6008	5405	4418	3186	1680	3164
Total Errors	49	48	54	50	56	45	38	31	23	25
Good Errors	49	48	54	50	56	45	38	31	23	25
Evil Errors	0	0	0	0	0	0	0	0	0	0
Average Total Connections	236	236	234	233	219	197	162	115	61	243
Average Good Connections	226	227	227	227	214	193	159	113	60	112
Average Evil Connections	9	8	6	5	4	3	2	2	1	130

Table 7: Limit Number of Connections - 80 Good Clients and 1 Evil Client

In the graphs to follow, a limit of 0 represents the base case scenario.

Figure 26 shows the attempted number of connections. Both curves are similar, as they both have a peak at the same limit and follow the same general pattern. Although there are some fluctuations, those are due to statistical errors. Therefore, the line can be considered to be horizontal, which is what is expected since the number of connections should stay approximately the same, regardless of the limiting value used.

The number of dropped connections is shown in Figure 27. The number of established connections and the average size of the connection table are graphed in Figures 28 and 29 respectively.

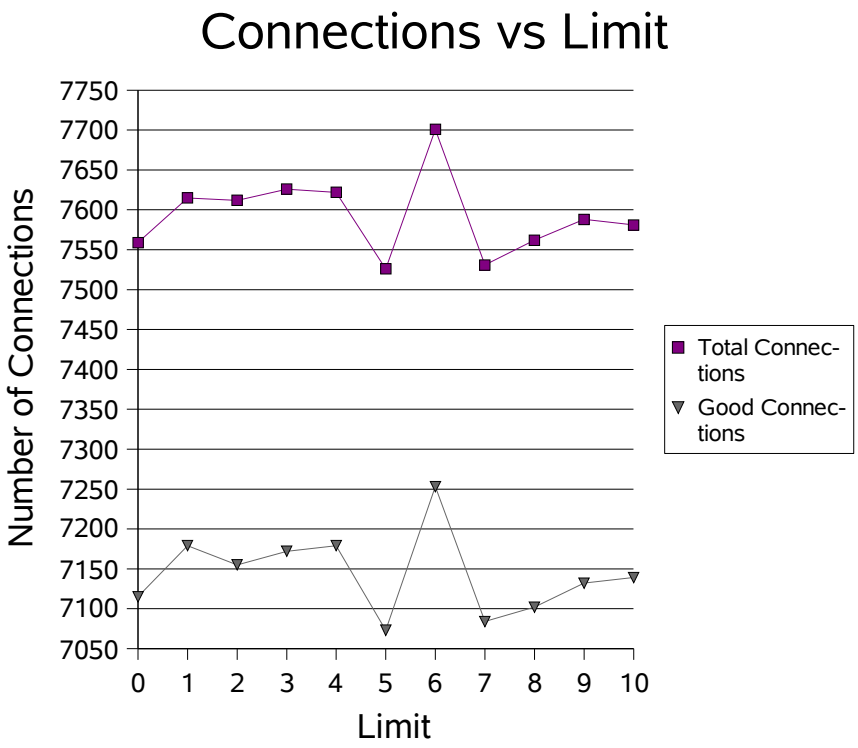

Figure 26: 80 Good Clients and 1 Evil Client (0 is the base case)

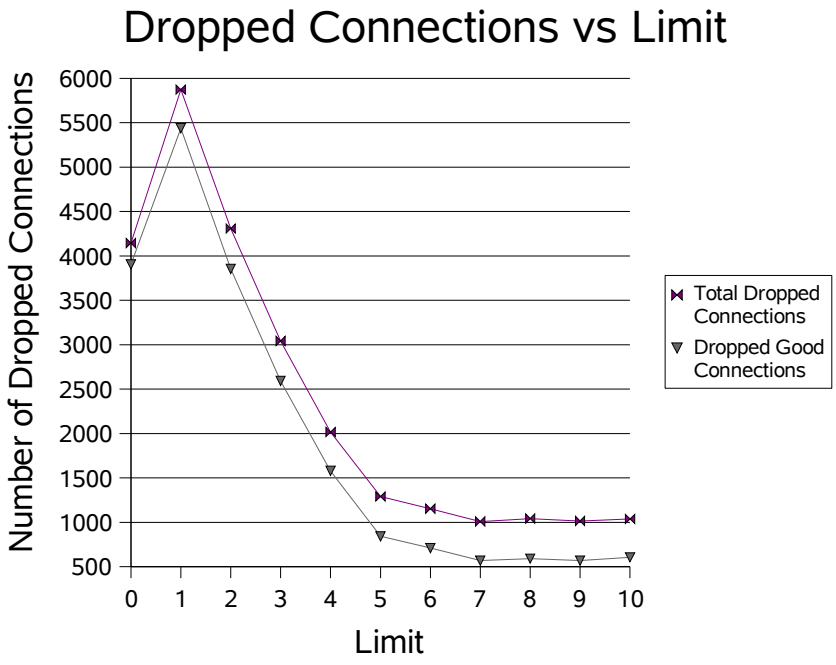

Figure 27: 80 Good Clients and 1 Evil Client (0 is the base case)

When the limit is really low (limit = 1, 2), there are a lot of dropped connections. This is because, each good client would have, on average, about 3 to 4 connections at any time in the connection table of the server since every 20 seconds, it sends a new initial request, and every 60 seconds, it sends a reset message. As mirrored in Figures 28 and 29, there are very few established connections and the connection table is pretty much empty on average for those low limits. This is an extreme case. Figure 28 is a mirror of Figure 27.

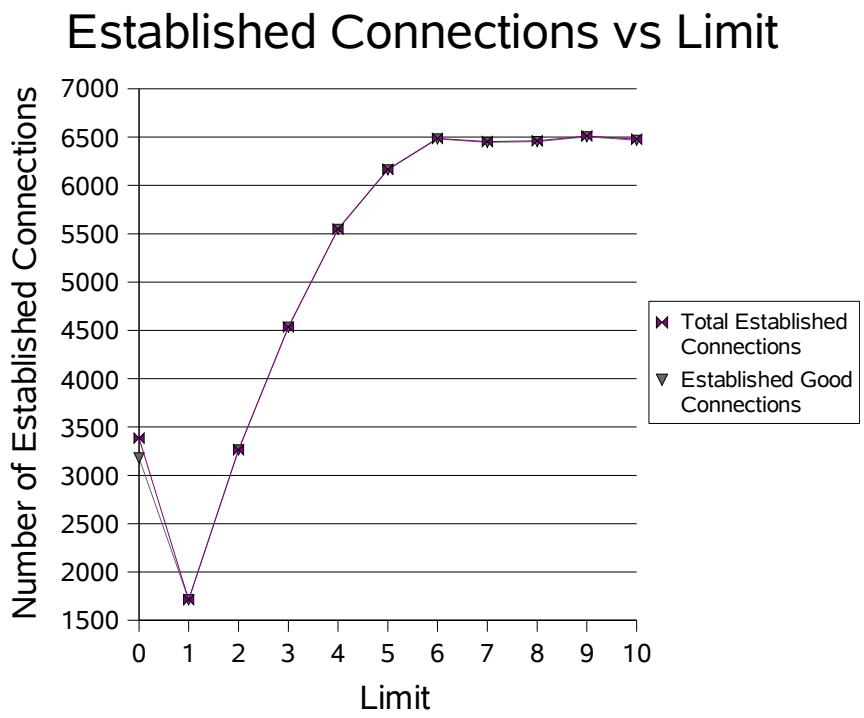

Figure 28: 80 Good Clients and 1 Evil Client (0 is the base case)

However, as the limit increases, the number of dropped connections drops significantly. The number of established connections and the average size of the connection table also increase dramatically. As can be seen in Figure 28, the total number of established connections and the total number of established good connections are on

the same line. This is because the single evil client is being hampered by the introduction of the limit. The client is limited to the number of connections it can create, and from Figure 29, the good clients "hog up" the whole connection table.

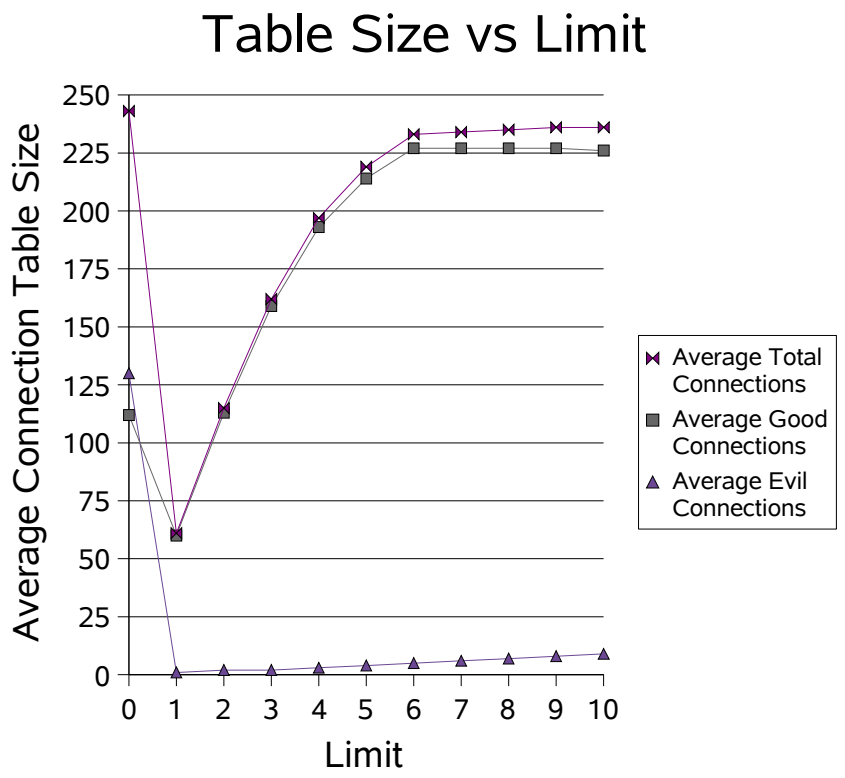

Figure 29: 80 Good Clients and 1 Evil Client (0 is the base case)

As the limit increases, the performance converges back to the base case. This is because as more evil connections can be created, the limit does not help anymore, as the evil client creates as many connections as possible and the good clients suffer from that.

The optimal value of the limit is 6. This is about the average number of connections a good client will have at any time in the server's connection table.

More evil clients are added to the simulation to determine if the limit really helps or not. The next test case contains 80 good clients and 16 evil clients.

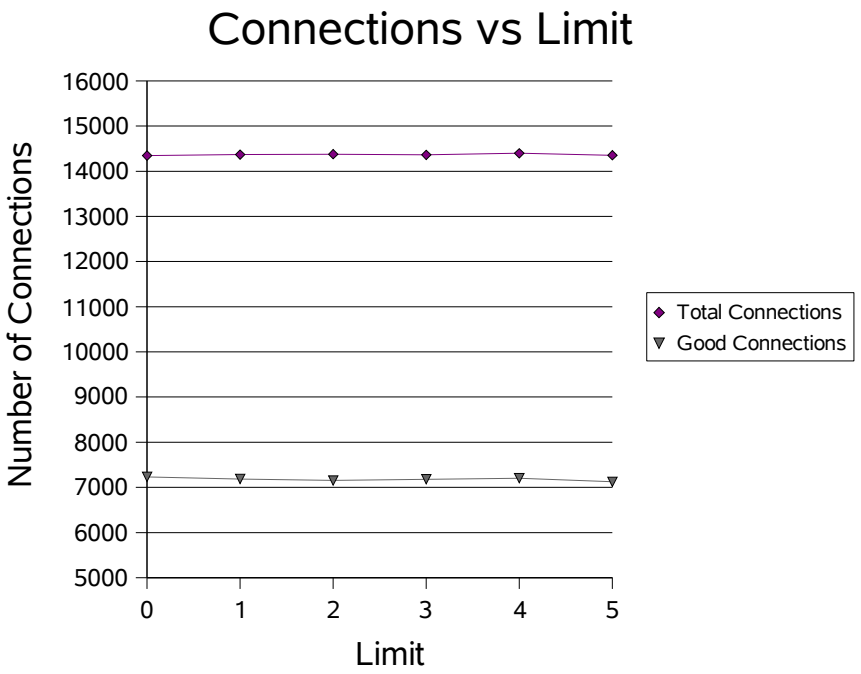

Figure 30: 80 Good Clients and 16 Evil Clients (0 is the base case)

Figure 30 shows that the number of connections is constant, so there are no side-effects to be considered.

Figures 31, 32, and 33 show the total number of dropped connections, the total number of established connections, and the average size of the connection table. They are all very much similar to Figures 27, 28, and 29.

The base case was not very successful in diminishing the impact of the evil clients because they were able to create so many connections, thereby reducing the effective size of the server's connection table. As shown in Figure 33, the evil clients were taking up most of the connection space in the table.

Therefore, the introduction of the limit helped right away, even with a low limit since evil clients are effectively being denied.

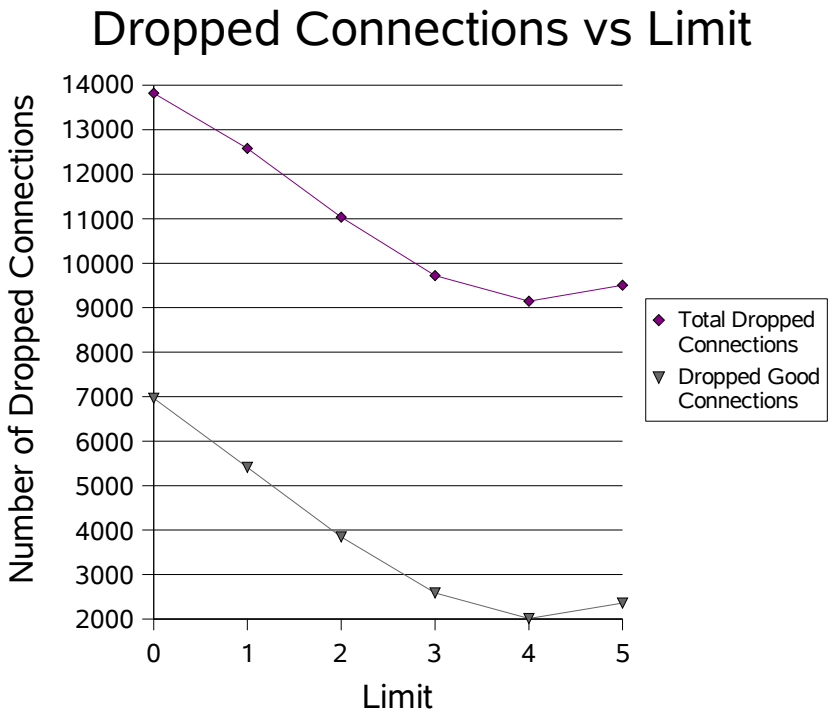

Figure 31: 80 Good Clients and 16 Evil Clients (0 is the base case)

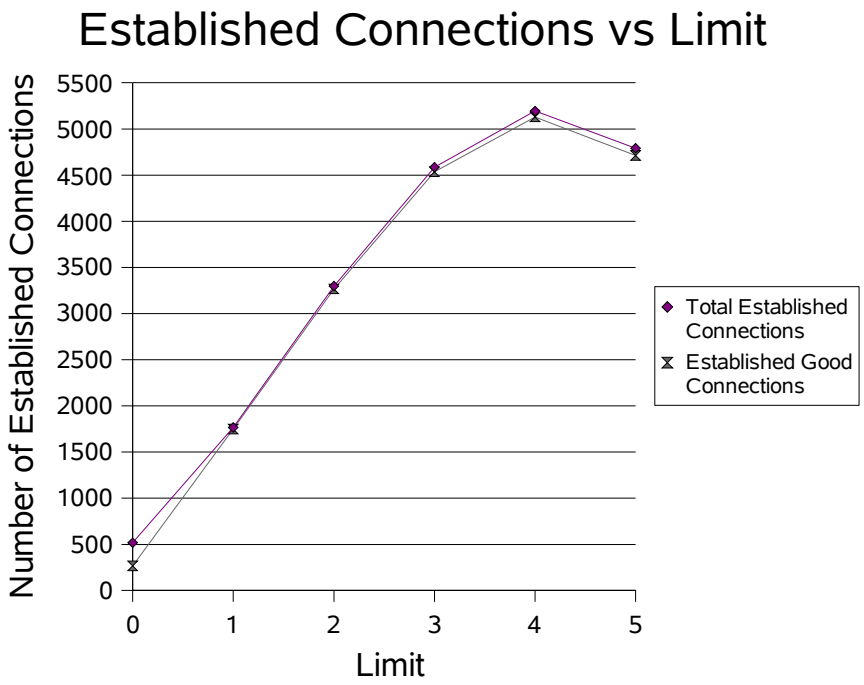

Figure 32: 80 Good Clients and 16 Evil Clients (0 is the base case)

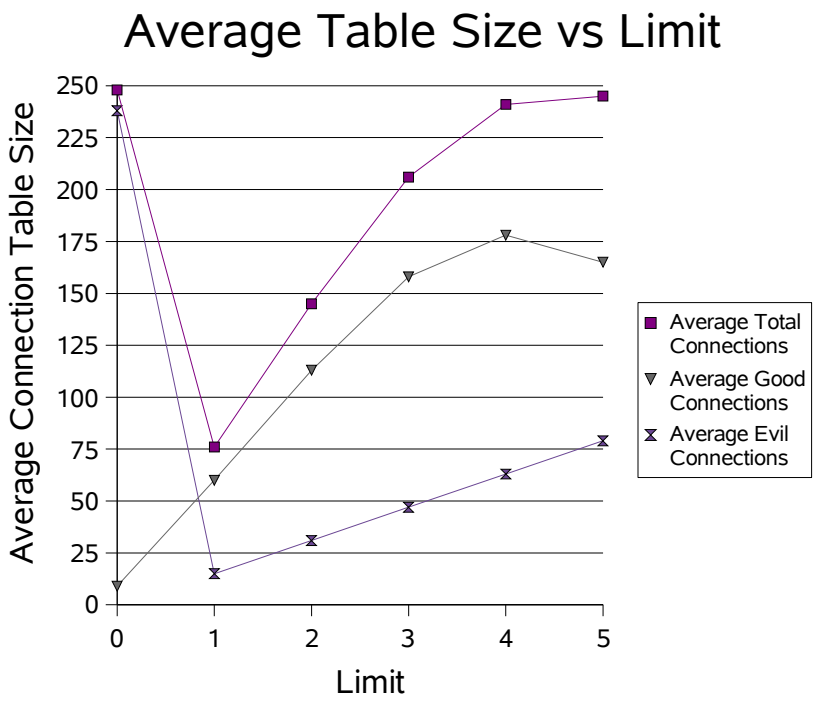

Figure 33: 80 Good Clients and 16 Evil Clients (0 is the base case)

However, with a low limit, the connection table was not being used to its full capacity, but as the limit is increased, the average size of the connection table increases to its maximum. As the limit is increased beyond 5, the evil clients are again allowed to hamper the server's availability.

In Figure 28, both lines overlap each other. However, in Figure 32, the “good established connections” line is a bit below the “total established connections” line. This is due to more evil connections being established since there are more evil clients.

With only 1 evil client, the optimal value for the limit is around 6 connections. With 16 evil clients, the optimal value for the limit decreases to 4 connections. The same trend can be seen as the number of evil clients increases.

5.4 Conclusion

Limiting the number of connections a client can have in the server's connection table does help in reducing the impact of an attack by evil clients. Moreover, the good clients receive a boost in their ability to create new connections in the table. However, as was shown with the two different cases in Chapter 5.3, it is very hard to determine what the optimal value of the limit should be since there is no way of knowing who the evil clients are.

Decreasing the limit as the number of clients increases might work if the number of clients is increased. But a lot more tests have to be run to determine what the optimal value should be for each number of clients. Moreover, if there are no evil clients, the limit should not be decreased because good clients do not connect that often to the server.

Since there is no way for the server to determine whether a client is good or evil, another method has to be found to improve the performance of the server.

6 Experiment 3 – Server Reset

6.1 Goals

As was discussed in Chapter 4, a priority queue did not help hinder evil clients' attack. In Chapter 5, using a limiting value to limit the number of connections a client can create did help improve the performance of the server but it was difficult to determine what the optimal value of the limit should be since the server does not know how many good and evil clients there are.

The third and last experiment to be implemented is the server reset. Whenever its connection table becomes full and a new connection needs to be created, the server will forcefully remove the oldest established connection in its connection table, regardless of which client created that connection. In-progress connections are not removed. If an evil client does not acknowledge back to the server, that in-progress connection will eventually time out.

Since evil clients never reset any of their established connections, the latter stay in the server's connection table for the duration of the simulation. Therefore, the server will try to shoulder the responsibility of flushing out the evil connections. The oldest established connection is removed since this is the most likely connection to be evil. Good clients will eventually reset their connections after about 60 seconds. The oldest

connection is just deleted and no reset message is sent from the server to the client whose connection is being removed. The server reset is performed only when the server's table is full, which implies that it is busy, under attack, or under a big load of requests. Therefore, having the server create a new message and send it will just create more overhead for the server.

It is expected that the evil connections will get reset which will allow more good connections to be created. However, it is also expected that some good connections will be reset prematurely as well. If this happens, when the real reset from the client reaches the server, this will be reported as an error.

The priority queue will still not be used in this experiment but the limit will be implemented as well as the server reset. The limiting value will be varied to determine whether the server reset helps in all cases, including the base case.

6.2 New Parameter

One parameter needs to be added to the parameter list to implement the server reset. It is shown in Table 8.

useServerReset	true	Whether or not to use the server reset

Table 8: New Parameter for Server Reset

6.3 Runs

The same parameters are used as before, except that both the limit and the server reset are used. It will be determined whether the addition of the server reset helps improve the performance of the server or not.

Table 9 shows a sample of the important statistics measured.

Limit	10 w/ Reset	10	6 w/ Reset	6	1 w/ Reset	1	Base Case w/ Reset	Base Case
Total Connections	7633	7581	7627	7701	7745	7615	7564	7559
Good Connections	7179	7139	7184	7253	7282	7179	7116	7115
Evil Connections	454	442	443	449	463	436	448	444
Total Dropped Connections	348	1039	760	1154	5971	5873	0	4145
Dropped Good Connections	4	607	365	711	5509	5439	0	3905
Dropped Evil Connections	344	432	396	443	462	435	0	240
Total Established Connections	7211	6478	6781	6486	1745	1714	7483	3386
Established Good Connections	7101	6468	6733	6480	1744	1713	7036	3182
Established Evil Connections	109	10	48	6	1	1	447	204
Total Reset Connections	7038	6299	6632	6311	1710	1680	7317	3164
Reset Good Connections	6938	6299	6590	6311	1710	1680	6897	3164
Reset Evil Connections	100	0	42	0	0	0	420	0
Total Errors	687	49	312	50	25	23	972	25
Good Errors	687	49	312	50	25	23	972	25
Average Total Connections	237	236	229	233	62	61	240	243
Average Good Connections	227	226	223	227	61	60	211	112
Average Evil Connections	9	9	5	5	1	1	29	130

Table 9: Server Reset

As shown in Appendix A, the total number of connections for both cases – without server reset and with server reset – for all the limit values is constant. The small changes are only due to statistical fluctuations.

As shown in Figure 34, there are no dropped good connections for the base case when the server reset is used. This is not quite surprising although it was expected that at least a couple of connections would get dropped, because the server deletes the oldest established connection when its connection table gets full.

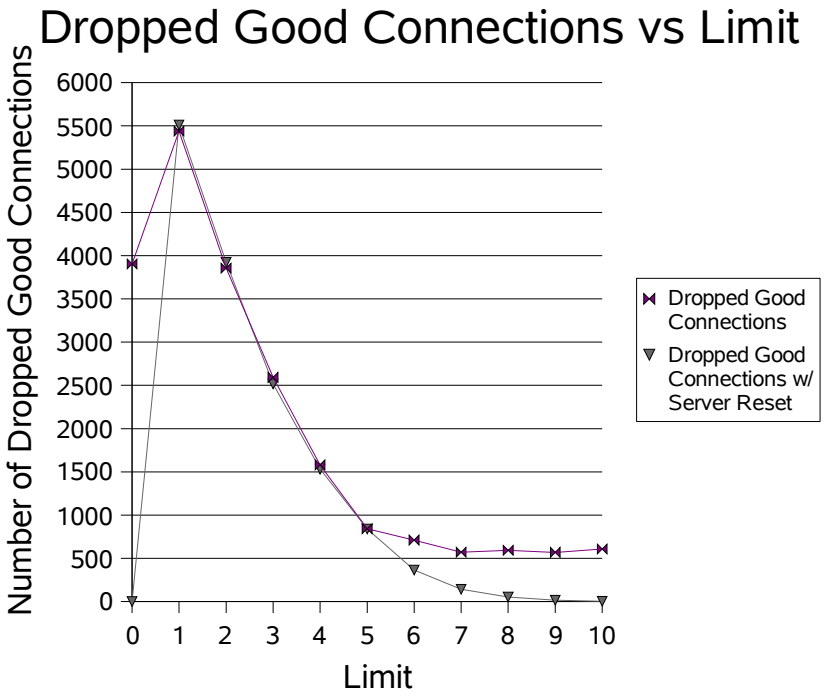

Figure 34: 80 Good Clients and 1 Evil Client (0 is the base case)

For low limit values, the server reset does not help at all, but for higher values, it helps to reduce the number of dropped connections significantly, even more than the limit without the server reset does. However, without using the server reset, as the limit is increased, the number of dropped connections increases, whereas with the server reset, the number of dropped connections stays at zero. However, this is not the only measure that is to be considered as shown later.

Figure 35 shows the number of dropped evil connections. It is analogous to Figure 34 with fewer dropped evil connections as the limit is increased, but evil connections are still being dropped, unless the limit is really big, then it converges back to the base case.

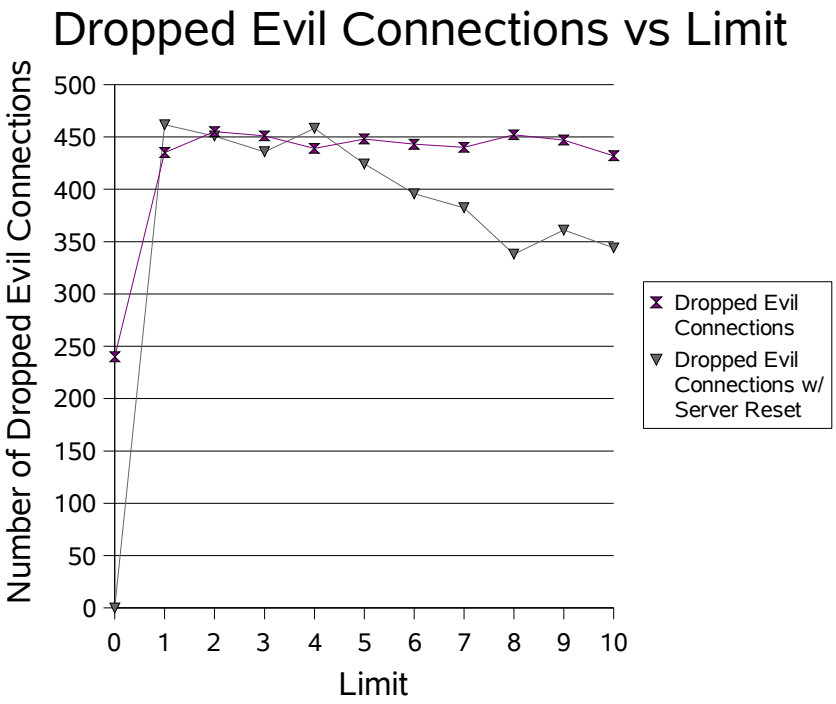

Figure 35: 80 Good Clients and 1 Evil Client

Figures 36 and 37 show the number of established connections. Compared to the base case, the limit helps decrease the number of established evil connections while the number of established good connections increases. Using the server reset, the number of established good connections is increased even more with the optimal value of limit. Although the number of established evil connections also increases, the increase in the established good connections is more significant than the increase in the established evil connections.

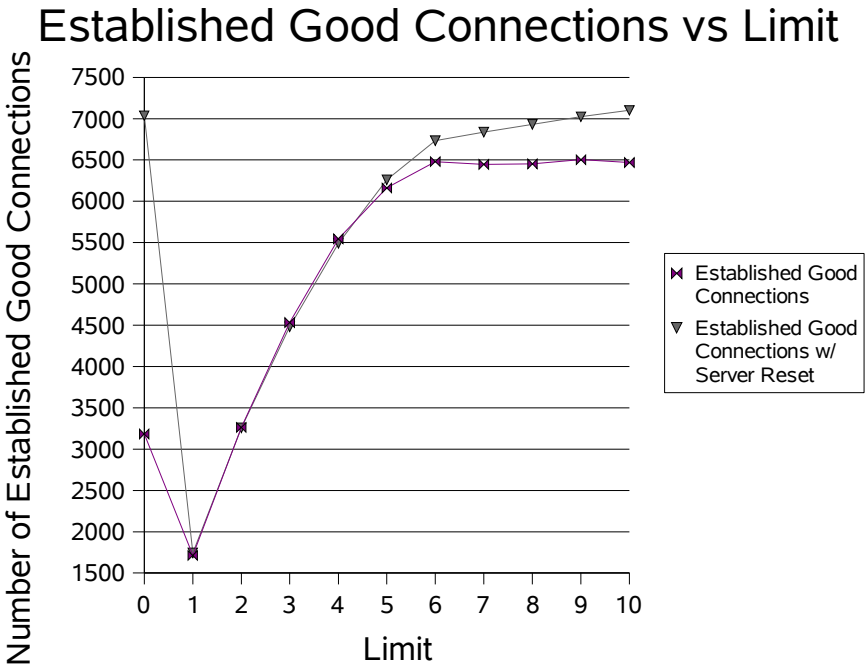

Figure 36: 80 Good Clients and 1 Evil Client (0 is the base case)

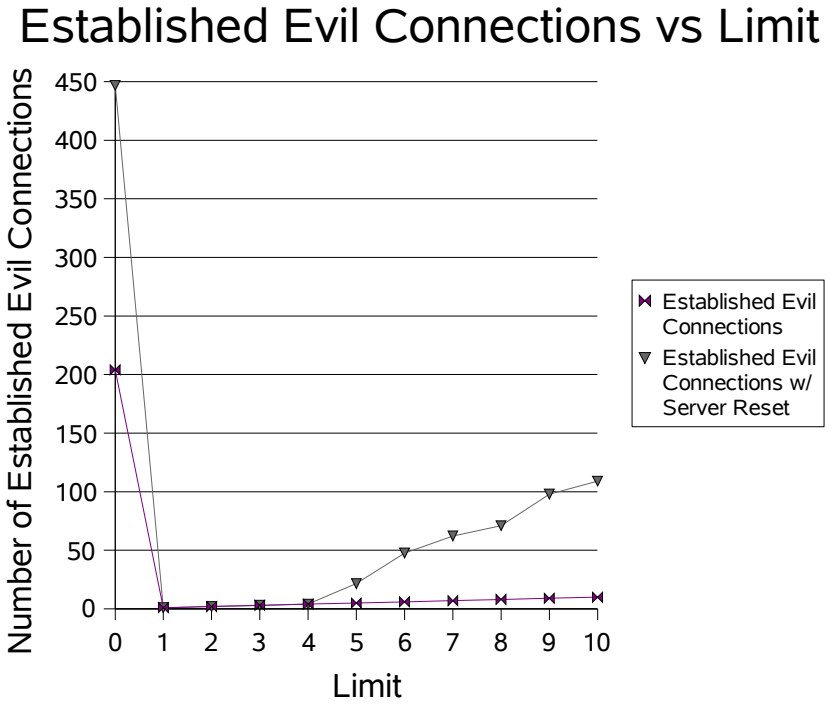

Figure 37: 80 Good Clients and 1 Evil Client (0 is the base case)

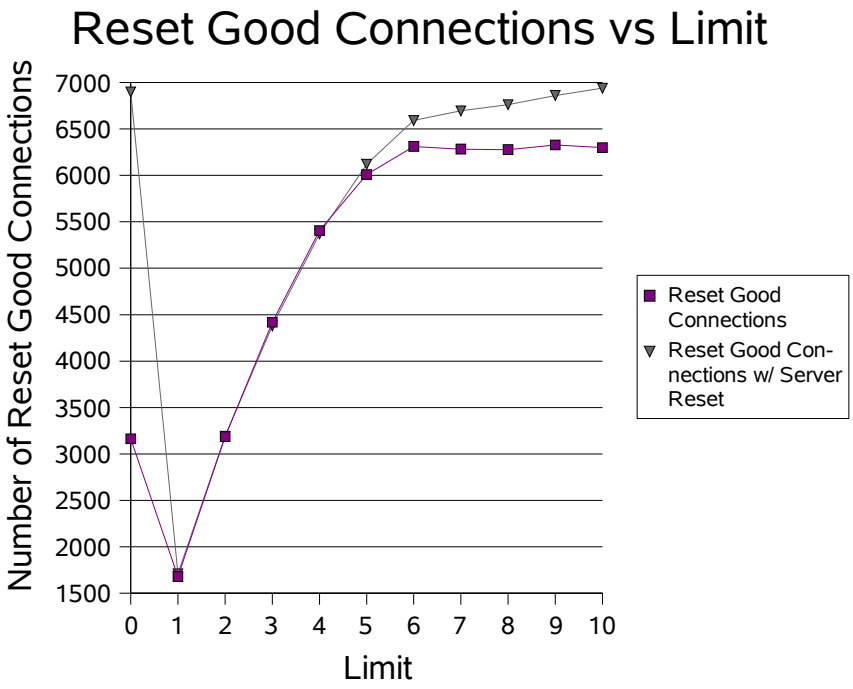

Figure 38: 80 Good Clients and 1 Evil Client (0 is the base case)

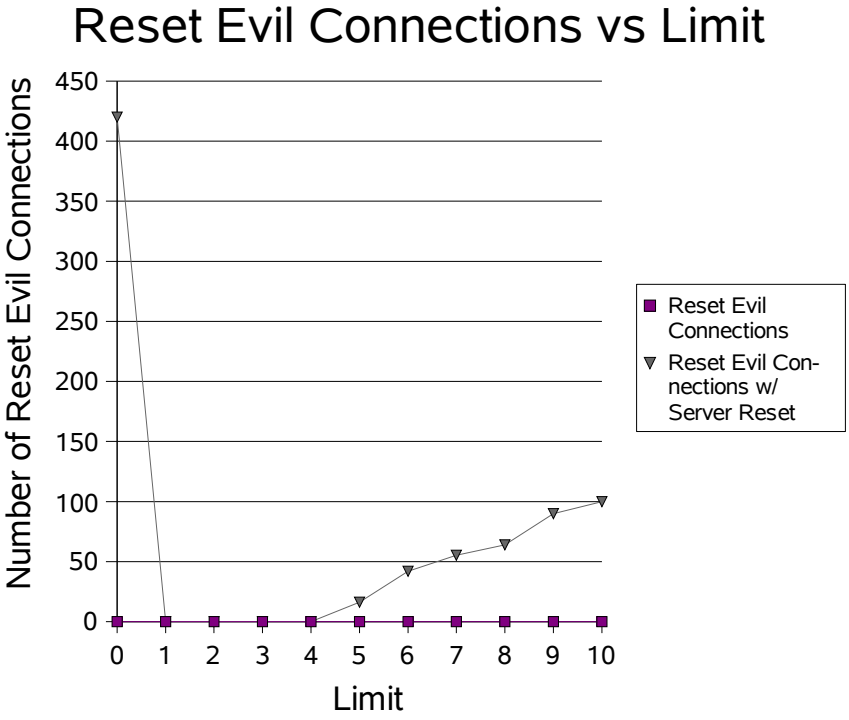

Figure 39: 80 Good Clients and 1 Evil Client (0 is the base case)

Figures 38 and 39 show the number of reset connections. With the server reset, the number of reset good connections goes up. The extra reset good connections are due to the server prematurely removing connections when its connection table gets full. It just so happened that a good client's connection was the oldest established connection at that time. The extra reset connections match the extra errors reported by the server, as shown in Figure 40. On the other hand, evil connections, which were never reset before, are now being removed from the server's connection table. This is the main reason why there are more established and fewer dropped connections overall.

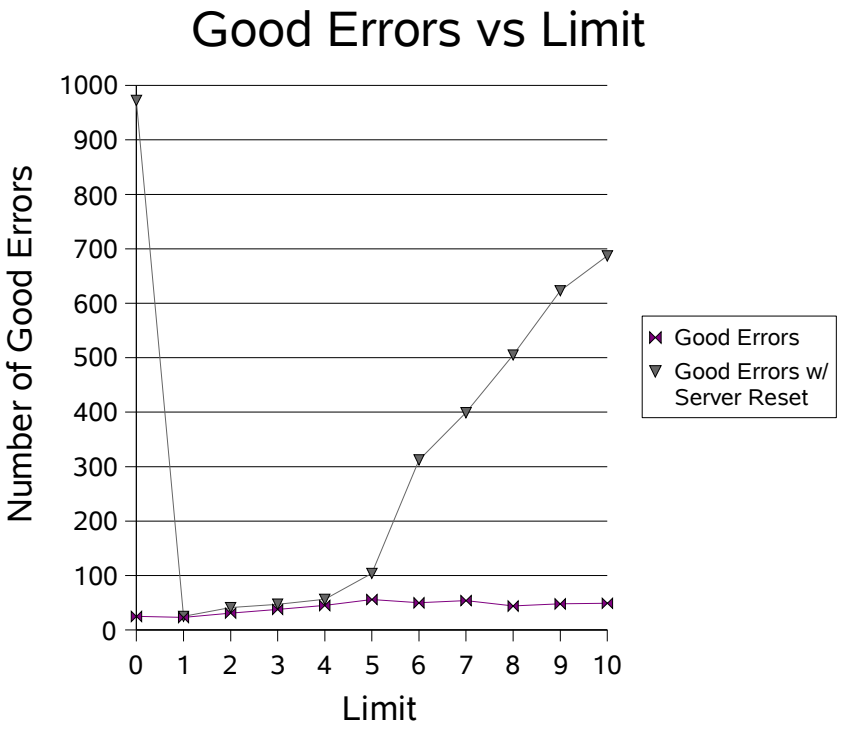

Figure 40: 80 Good Clients and 1 Evil Client (0 is the base case)

Figure 41 shows the average number of good connections in the server's table. The average number of good connections is about the same with or without the server

reset, except in the base case. In the latter, evil clients are effectively being removed from the connection table all the time. However, the number of errors also goes up dramatically.

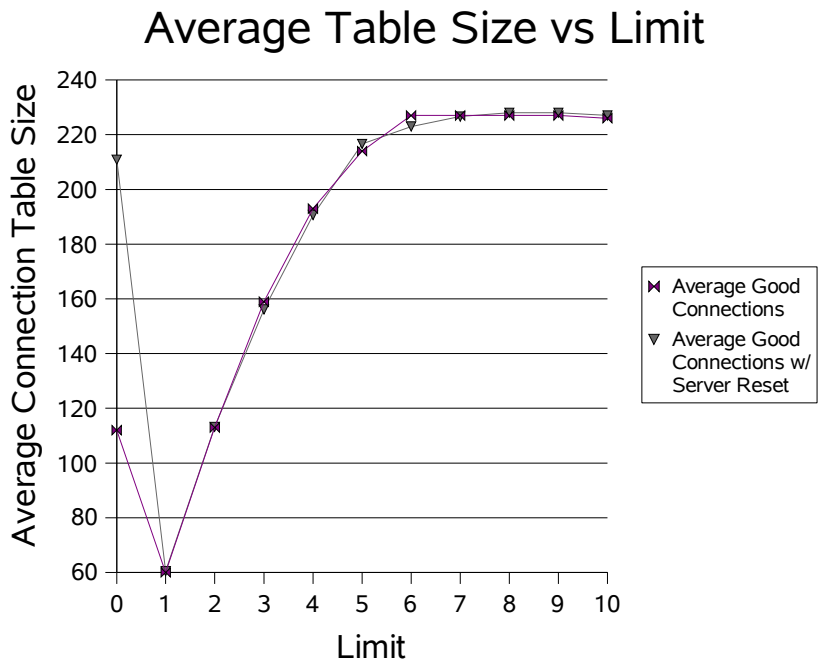

Figure 41: 80 Good Clients and 1 Evil Client (0 is the base case)

The same results are observed when the number of evil clients is increased. The data is shown in Appendix A.

6.4 Conclusion

Using the server reset with no limit is a bit drastic, since good connections are ended prematurely. Although there is no way of knowing in the current simulation

whether that had any effect or whether the reset from the client was on its way already when the server removed the connection, the big increase in the number of errors is something to be analyzed deeper.

When limits are used, an even better improvement is seen when the server reset is used, which was expected. As mentioned above, there is no way of knowing whether the increase in errors caused any problems such as a connection ending prematurely and the client having to reestablish the connection.

Moreover, the optimal value for the limit cannot be effectively determined for the same reasons mentioned in Chapter 5.4.

7 Conclusions

Distributed Denial of Service attacks are a real threat as described in Chapter 1. They are currently generally unstoppable, and there are a lot of academic research and government-funded research going on in this area. This thesis only analyzed a very small portion of the whole field of DDoS attacks.

The simulation performed consisted of a lot of parameters or input values but most of them were fixed and were obtained from the real world (Macalester College's servers). Although a lot of statistics were taken, not all of them are applicable or mean anything for every test case. The simulation also focuses mainly on brute force attack and varying the number of clients – good and evil.

The base case was carefully chosen after numerous tests and analysis and was such that the server is not completely overloaded that any defense techniques implemented would not really help or not busy at all that any improvements by implementing some defenses would not be noticeable. 80 good clients and variable evil clients with an interval time of initial connect requests of 4 seconds were chosen as the base case.

The first experiment performed was implementing an inbound message priority queue for the server instead of an inbound first in first out message queue. As shown in Chapter 4, the addition of a priority queue did not help improve the performance of the

server. This was due to the fact that evil messages will eventually get to the front of the line and once an evil connection has been created, it is never reset.

In light of the failures of the first experiment, the second experiment performed implemented a limit. The limit value limits the number of connections each client can have in the server's connection table at any time. As described in Chapter 5, the limit did improve the performance of the server, but sometimes it was too extreme, such as if the limit value is too low, there are no improvements, but in fact there are deteriorations in the availability of the server. Moreover, if the limit value is too large, the limit is not really used as the test case resembles the base case without the limit. Finding the optimal value of the limit is hard because it depends on the number of clients, the number of good clients, and the number of evil clients. However, if somehow it could be predicted in advance how many good and evil clients there will be in the simulation, then the limit does help a lot. The same results are observed for increasing number of evil clients.

Since evil connections stay in the connection table for the duration of the simulation because evil clients never reset their own connections, a forced server reset is implemented in experiment 3. Whenever its connection table becomes full and a new connection needs to be created, the server will forcefully remove the oldest established connection, regardless of how old it is and what client created that connection. Chapter 6 describes in more details the effects of the server reset. The main aspects of this implementation are that the server reset along with the limit further improves the availability of the server. Also, the limit is not as important as it was when it was used alone as in experiment 2. Although the number of established evil connections increases,

the increase in the number of established good connections more than offsets it. Even with an increase in the number of evil clients, the same results are observed. The downside to using the server reset is that the number of errors also increases. An error means that a message arrived at the server and the latter does not know what to do with it. In the server reset case, the increase in errors is due to a good connection being prematurely removed, and when the reset from the client comes in, the connection has already been deleted and the server does not know what to do with the message. With the way the simulator has been implemented, it cannot be determined whether the connection has been terminated prematurely or whether the client reset was already on its way when the server deleted the connection.

The limit experiment is in fact implemented in real life: The macalester SMTP (outgoing email) server uses a limit of 10. The server reset is also used in the real world. It probably happened to everyone when their connection gets timed out, when logged in to a bank's website, or when idling in an FTP session. Since no data transfer is simulated, there is no way to set the “idle” time value, but just the oldest established connection being removed.

In both experiments 2 and 3, the limit and the server reset were performed regardless of who the client is. They don't try to differentiate between a good and an evil client. Thus, there are a lot of other defense techniques that could have been implemented and those are discussed in the next chapter.

8 Future Work

A lot of other work could build from the simulator to examine other aspects. Detection techniques could be implemented. There are currently a lot of ways to try to detect when a DDoS attack is happening and also who the attacker(s) are. When that is known, the limit could be tuned to the optimal value for each set of parameters. Also, the server can reset only the evil connections.

A firewall could also be implemented to act like a proxy between the server and the clients. It will thus only pass in completely established connections to the server, thereby reducing the load on the latter. The processing time would probably be lower for the firewall and it can have in-built mechanisms to detect and mitigate DDoS attacks. That would be its single task which could then be optimized, in contrast with the server, which has to be more of a general purpose service.

Multiple servers could be used to allow for greater connection space, higher bandwidth, and the ability to cope with a larger attack force. A load balancer thus has to be implemented which then distributes messages to all the servers in a fair way, such as using round robin or determining which server is the least busy and giving it the message to process.

Other attacks could also be implemented or analyzed such as the TCP SYN exploit, LAND attack, smurf attacks, and all those other types of attacks outlined in Chapter 1. The current simulation uses only brute force attack which might be the hardest type of attack to defend against, but the other types of attacks are also interesting.

Least but not last, the simulator could be extended to allow for virtual data transfer, the four-way handshake connection termination, and other real aspects of a network. A current simulator that possesses all those abilities will be the ns-2 simulator, but other features can easily be added to the current simulator so that it simulates a real traffic network.

9. Bibliography

1. [Carl et al, 2006] Carl, G., Kesidis, G., Brooks, R. R., Rai, S., "Denial of Service Attack-Detection Techniques", IEEE Internet Computing, Jan/Feb 2006
2. [Challita et al, 2004] Challita, A., Hassan, M. E., Maalouf, S., Zouheiry, A. (2004), "A Survey of DDoS Defense Mechanisms", Department of Electrical and Computer Engineering, American University of Beirut, Retrieved from http://webfea.fea.aub.edu.lb/proceedings/2004/SRC-ECE-39.pdf
3. [Cheswick et al, 2003] Cheswick, W., Bellovin, S., Rubin, A., *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley Professional, Second Edition, 2003
4. [Denial of Service, n.d.] Denial of Service (n.d.), Retrieved on May 10, 2005 from http://whatis.techtarget.com/definition/0,289893,sid9_gci213591,00.html
5. [Farrow, n.d.] Farrow, R. (n.d.), "Distributed Denial of Service Attacks (DDoS)", Retrieved from http://chinese-school.netfirms.com/computer-article-denial-of-service.html
6. [Forouzan, 2000] Forouzan, B. A., *TCP/IP Protocol Suite*. McGraw-Hill Companies, Inc., 2000
7. [Kessler, 2000] Kessler, G. C., "Defenses Against Distributed Denial of Service Attacks", SANS/GIAC Security Essentials Certification, November 2000
8. [Land Attacks Still Going Strong, 2005] Land Attacks Still Going Strong (2005),

Retrieved on December 14, 2005 from
http://www.securiteam.com/securitynews/6H00E15EUE.html

9. [Lau et al, 2000] Lau, F., Rubin, S. H., Smith, M. H., Trajkovic, L., "Distributed Denial of Service Attacks", 2000 IEEE International Conference on Systems, Man, and Cybernetics, Volume 3: 2275-2280, 2000
10. [Mell et al, 2000] Mell, P., Marks, D., McLarnon, M., "A denial-of-service resistant intrusion detection architecture", Computer Networks 34: 641-658, 2000
11. [Mirkovic et al, 2004] Mirkovic, J. and Reiher, P., "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms", ACM SIGCOMM Computer Communications Review, Volume 34, Issue 2, April 2004
12. [Mutaf, n.d.] Mutaf, P. (n.d.), "Defending against a Denial-of-Service Attack on TCP", Retrieved from
http://www.raid-symposium.org/raid99/PAPERS/ParsMutaf.pdf
13. [Oliver, 2001] Oliver, R. (2001), "Countering SYN Flood Denial-of-Service Attacks", Retrieved on August 21, 2001 from
http://www.tech-mavens.com/synflood.htm
14. [Rogers, n.d.] Rogers, L. R. (n.d.), "What is a Distributed Denial of Service (DDoS) Attack and What Can I Do About It?", Retrieved on September 10, 2005 from http://www.cert.org/archive/pdf/homeusers/ddos.pdf
15. [Stein et al, 2002] Stein, L., Stewart, J. (2002), "The World Wide Web Security FAQ", Retrieved on February 4, 2002 from http://www.w3.org/Security/Faq
16. [Tanenbaum, 2002] Tanenbaum, A. S., *Computer Networks*. Pearson Education,

Inc, Fourth Edition, 2002

17. [Templeton et al, 2003] Templeton, S. J., Levitt, K. E. (2003) "Detecting Spoofed Packets", DARPA Information Survivability Conference and Exposition - Volume I: p. 164, 2003
18. [Tupakula et al, 2003] Tupakula,U. K. and Varadharajan, V., “A Practical Method to Counteract Denial of Service Attacks”, ACM International Conference Proceeding Series, Volume 35: 275-284, 2003
19. [Vijayan, 2005] Vijayan, J. (2005), “Port scan may not always signal attacks”, Retrieved on Dec, 7, 2005 from http://www.computerworld.com/securitytopics/security/story/0,10801,106849,00.html

10. Acknowledgments

I would like to thank my advisor Professor Michael Schneider, and my committee members Professor Richard Molnar and Mr. Barron Koralesky. A special thanks also goes to Mr. Ted Fines for all the information pertaining to the Macalester network.

I would also like to thank my brother, Sebastien Chan-Tin, for his help; my parents and Youa Yang for their love and affection; Jesse Harman, Jacob Dorer, Rita Lee, Dang Vang, Pakou Vang, and Thao Huynh for their friendship and support during my thesis research.

Last but not least, I convey my thanks to the OpenOffice community for such a great product.

Appendix A – Raw Data and Tables

In an effort to save paper, the raw data and tables are not printed. They are available in electronic format at the Macalester College library. You can also contact the author at echantin@alumni.macalester.edu

Appendix B – Code

Similar to the reason mentioned in Appendix A, the code is not printed but is available in electronic format at Macalester College. It can also be obtained from the author by contacting him via email at echantin@alumni.macalester.edu