SUBROSA: AN EXPERIMENTAL PLATFORM FOR STUDYING TIMING ANALYSIS OF REAL TIME ANONYMITY SYSTEMS

by

HATIM ASGER DAGINAWALA

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2007

To my wife Cathy

ACKNOWLEDGEMENTS

I am eternally grateful to Dr. Matthew Wright for being a mentor and guide, for helping me in every aspect of this long process. Without his guidance and support this thesis would not have been published. I would like to thank Dr. Hao Che and Dr. Donggang Liu for their interest in my research and for taking time to serve on my thesis committee. I would also like to acknowledge the quality work of researchers and academicians in the area of network security in general and anonymous communication in particular.

I am grateful to Cyrus Bavarian for his helpful insight and suggestions in the design of SubRosa and for his support and encouragement.

Thank you seems very small for my family and friends without who's encouragement, support, motivation and prayers I would have never been able to come this far.

Last but not the least to Cathy, for everything she did and does for me and us.

November 26, 2007

ABSTRACT

SUBROSA: AN EXPERIMENTAL PLATFORM FOR STUDYING TIMING ANALYSIS OF REAL TIME ANONYMITY SYSTEMS

Publication No. ______

Hatim Asger Daginawala, M.S.

The University of Texas at Arlington, 2007

Supervising Professor: Dr. Matthew K. Wright

Timing analysis poses a significant challenge for the mix based anonymous systems that wish to support low-latency applications like web browsing, instant messaging and Voice over IP (VOIP). Research in this area so far has been done through simulations on ad hoc simulators or non practical local area networks. We developed SubRosa, an experimental platform for studying timing analysis of real time anonymity systems to facilitate the study of Tor like low-latency anonymous systems.

We present results of experiments on a real distributed network test bed PlanetLab, where we studied timing analysis attacks and some of the defenses to protect against such attacks. We validate the simulated results obtained for defensive dropping by Levine et al in 2004 on PlanetLab using SubRosa. We also propose a new light weight defense based on the basic mix principal called γ-buffering and evaluate the initial results from the experiments with γ-buffering on PlanetLab.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Wide acceptance of the Internet as a popular communication medium and information resource has made concerns over the privacy and anonymity of network communication a prominent issue. Sexual abuse and other crime victims might want to seek information on the Internet or share information with other victims. A business might want to hide their communication from their competitors. Law enforcement agencies may want to encourage citizens to provide tips without revealing their identities. Consumers may want to hide their purchasing patterns and habits from businesses. These are some of the reasons the Internet users might be interested in anonymous communication on the Internet. Also, systems like an electronic voting system will not be feasible without ensuring anonymity.

The fundamental design of the Internet and the IP networks, along with the relative ease of usage monitoring, the ease of data collection & data mining abilities of computers, creates a challenge for applying anonymity over the Internet. IP networks use IP addresses to identify computers on the network; IP address is linked with a user's real identity. Internet service providers (ISP) keep track of the relationship between the IP and the user. Anonymous systems try to hide this relationship so an IP address cannot be linked with the real user with certainty.

A mix, as Chaum [1] proposed in 1981, is a communication system used to hide the correspondences between its incoming and outgoing messages. The order of arrival is hidden by outputting the uniformly sized messages in lexicographically ordered batches. Many different mix architectures are proposed and exist. Since any given mix can be compromised, many systems proposed and developed for anonymous communications use a chain of mixes. The number of mixes a message passes through is a system determined parameter. Tor, for instance, uses a path length of 3, which means a message is routed through three mixes before it is delivered to the final destination.

Timing analysis attack refers to collecting timing of messages moving through the network and finding correlations with the objective of revealing senders' and receivers' identity. This study focuses on the timing analysis of low latency mix networks. Low latency mix networks are used for interactive applications like web browsing, instance messaging and voice data communication over IP network (VOIP). Various studies have proved timing analysis to be highly effective against low latency mix networks [2, 3, 20]. The stringent latency requirements of these applications make it challenging to hide timing correlation effectively and hence make it more vulnerable to the timing analysis attacks.

1.1 Contributions

Previous studies in this area have either been done on simulations [2, 3] or non realistic small local area networks [4]. We developed an application, SubRosa, to emulate real application and to run on the distributed test bed PlanetLab. PlanetLab is a global research network that supports the development of new network services. PlanetLab is a collection of computers distributed over the globe and all of the machines are connected to the Internet. SubRosa consists of a server, client and sink component and emulates behavior of Tor like anonymity system [5]. Tor operates on TCP, whereas SubRosa uses UDP as the transport.

We used basics of the mix design and applied it to a low latency mix network. A new light-weight defense against timing analysis, γ-buffering, is implemented in SubRosa and studied on PlanetLab. γ-buffering is designed to remove timing correlation from the network stream without the over head of constant rate cover traffic. We also compare γ -buffering with no defense and defensive dropping [2] on PlanetLab.

1.2 Outline

In chapter 2, prior work and background in anonymous communication and timing analysis is discussed. Chapter 3 discusses new γ -buffering algorithm in depth. Chapter 4 is an overview of SubRosa – test application, PlanetLab – test bed and experiment setup. Results are presented in chapter 5. Conclusions and future work are presented in chapter 6.

CHAPTER 2

BACKGROUND

Anonymity as defined in [6] “Anonymity is the state of being not identifiable within a set of subjects, the anonymity set. The anonymity set is the set of all possible subjects. With respect to acting entities, the anonymity set consists of the subjects who might cause an action. With respect to addressees, the anonymity set consists of the subjects who might be addressed.” In other words, subjects are more anonymous if they can hide in a larger set. Anonymity on the Internet can be compromised by any observer who monitors the network stream, collects the timing of the input and output messages and successfully finds correlation between them. This is referred to as timing analysis attack. Timing analysis is described in more detail later in this section.

2.1 Mix and Mix Network

In an attempt to provide anonymity, Chaum, in 1981, proposed the mix. The mix is a system that hides the correspondence between inputs and outputs. In order to hide the correspondence, the mix applies cryptographic transformation to input messages to their appearance. The mix also delays and reorders messages, so that outputs cannot be correlated with inputs based on timing analysis.

Two broad categories of mix networks exist [9]. The original mix proposed by Chum collects *n* messages and outputs them to the network in a batch. This is the basics of the pool-mixes e.g. Mixmaster10], Mixminion [11]. A Continuous mix or Stop-and-Go mix [12] stores the messages for a user specified delay and reorders messages based on the randomness of the delay distribution before sending it out on to the network, e. g. JAP[13]. Diaz at el has described variant of mixes along with their strengths and weaknesses in [9].

Using a single mix makes the entire anonymous system vulnerable to the compromise of this one mix and hence a chain of mixes are used for anonymous systems. Also, using multiple mixes increases anonymity of the entire system and protects the system from denial of service (DoS). Mix networks can be categorized into cascade networks and free route networks. In a cascade network, a message takes a predefined route determined by the system, e. g. JAP. In a free route network, the path is the selection of mix to be used for routing is done by the user, e. g. Tor.

High latency applications, like emails and message boards, don't have strict timing requirements and can use high latency mixes, which delay the message to provide anonymity. Low latency applications like web-browsing, instant messaging and VOIP have very stringent latency requirements. Latency inducing mix networks can render these applications useless. Empirical evidence shows that low latency anonymity systems attract more users as compared to the high latency systems [14]. A larger user base increases the anonymity set, which increases the anonymity of the mix network. The mix network we focus on in this study is a free route, mix network for low latency applications.

2.2 Timing Analysis: Attacks

In over 25 years since Chum introduced mix, many systems have been proposed and exist for anonymous communication. Various studies have been done on breaking the anonymity and proving defenses against them. This study focuses on timing analysis attacks.

Timing analysis attacks can be classified into active and passive. An active attack assumes a global active adversary, who not only can observe traffic on the entire network but can also delay and insert messages into the streams. The power of an active adversary is in the ability to insert and delay messages. A passive adversary, on the other hand, does not have the ability of inserting or delaying messages on the network. Thread model in this study is the same as described in [2]; a passive adversary who is aware of and has access to the entry and exit nodes on a path.

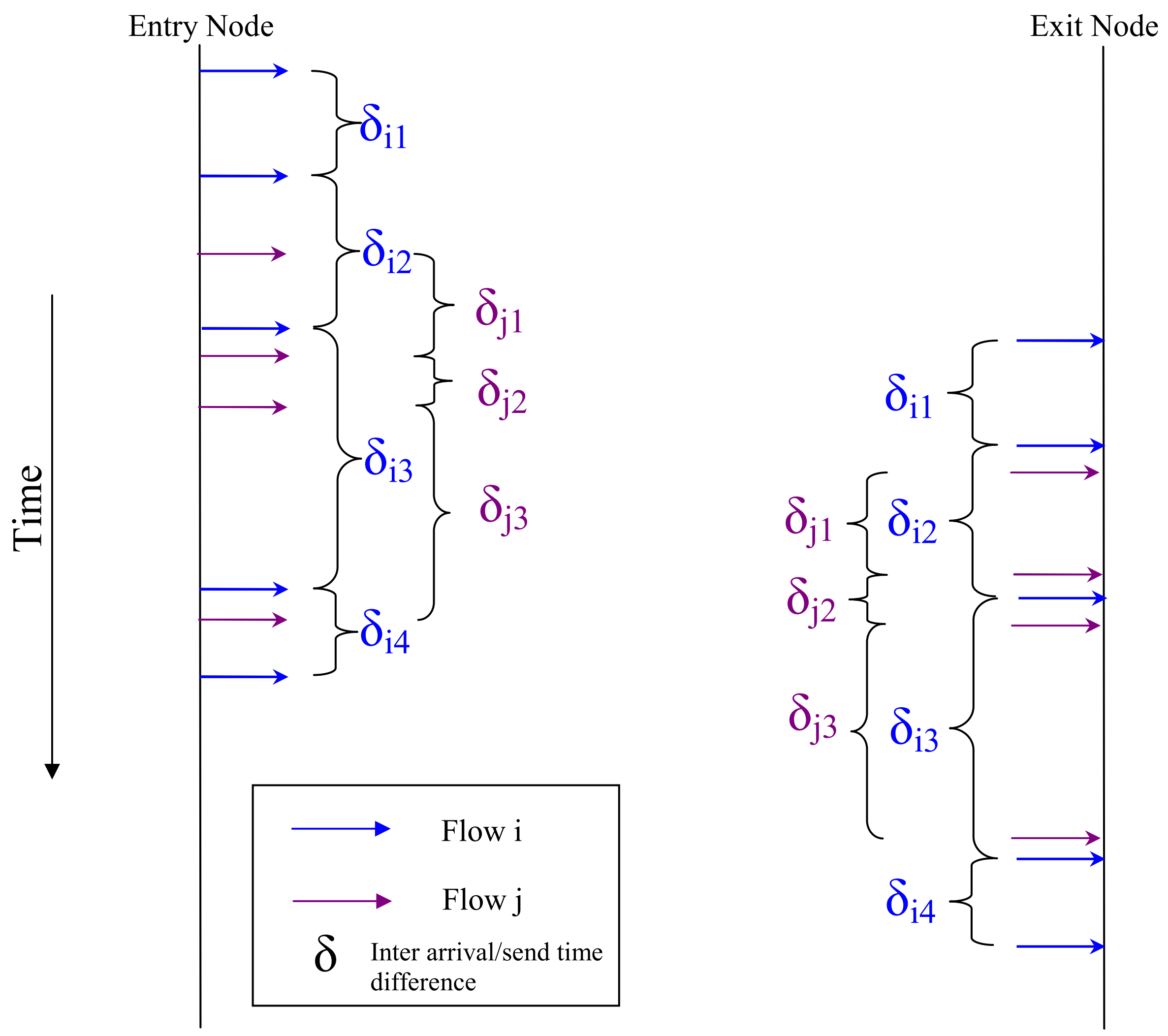

Figure 2.1 - Timing Correlation analyses

Figure 2.1 shows the basis of timing analysis; relative time difference (δ) between two consecutive packets remains the same throughout the network. In other words, δ of packets leaving the entry node will be approximately the same as that of a packet entering the exit node of a mix. This property of the network flow is used for timing analysis. Statistical correlation can be found between various distinct streams to determine the most likely sender and receiver of the stream and compromise the anonymity [2, 3]. A dropped packet on the path, intentional or due to network congestion, will cause the timings of the packets to be off by one. As a result, the correlation will be calculated between the packets that do not match. To avoid this effect, the count of the number of packets received during a time window is used instead of timings of packets, as suggested in [2].

2.3 Timing Analysis: Defenses

Constant rate cover traffic along the entire path is a known defense against timing analysis attacks. When all the participating nodes are synchronized and send data as the same constant rate, cover traffic makes all the streams look the same and makes it difficult to find correlation based on the time difference to isolate the streams. Pipenet [17] and ISDN-Mix [18] use end to end cover traffic where as Tarzan [15] uses hop by hop cover traffic.

Constant rate cover traffic, however, adds tremendous overhead to the network. For the cover traffic to be fool proof, all the nodes must be synchronized and transmit the packets at the same constant rate. Even with synchronized constant rate cover traffic vulnerability to an active adversary still remains. For these reasons, and under the assumption that the routing infrastructure is uniformly busy, systems like Tor [5] and Freedom [16], does not use any cover traffic.

Based on the idea of cover traffic, to reduce the timing correlations, various defense mechanisms have been proposed. In partial-route padding [19], all the cover traffic is dropped at a designated intermediate mix. Defensive dropping [2] generalizes the idea of partial-route padding, where the initiator creates a dummy packet and marks it to be dropped at any intermediate mix at random. When the packets are dropped at random at a sufficiently large frequency, the timing correlation is reduced [2]. Adaptive padding [20] tried to reduce the timing correlation by inserting dummy packets in the network stream, instead of dropping the packets. It is used to fill statistically unlikely gaps in the packet flow without adding latency. A variant of adaptive padding has also been proposed by Shmatikov and Wang [20] to counter attack.

CHAPTER 3

Γ -BUFFERING: BUFFERING FOR LOW-LATENCY CONNECTIONS

In the original mix design, each user must send one packet in each batching period. This has previously been extended to low-latency mixes by the use of constant rate traffic, in which each user emits packets at a constant rate, using dummy packets when real user traffic is not available. Constant rate traffic reduces the correlation between different flows since all the flows receive constant number of packets with the same time difference. In the design of real-world mixes, various batching approaches have been proposed [9]. This chapter introduces a new light weight defense based on the idea of constant rate traffic to prevent timing analysis attacks. It is called γ-Buffering.

3.1 γ-Buffering

γ-Buffering is a technique for buffering traffic that can be used to undermine traffic analysis attacks in low-latency anonymous communications systems. This technique is designed with the aim to maintain low latencies at the cost of some cover traffic and can be adapted for different levels of allowable latency and bandwidth use.

The main insight behind this technique is that, with sufficiently high traffic rates, standard batching techniques from mixes can be used without slowing down traffic excessively. While low-latency mixes setup and utilize paths for streams of packets, packet-level batching can effectively intertwine the streams' timing characteristics and destroy many of the patterns that attackers would seek to use in timing analysis. Random selection of packets from the batch during the send, independent of the packet arrival order and the latency introduced by buffering would reduce the timing correlation of packets making different streams have similar timing correlations. Furthermore, batching creates a variable intermediate delay that may be able to remove watermarking introduced by an active attacker.

End-to-end cover traffic can provide sufficiently high traffic rates. The amount of traffic entering a node at a given time, however, can still depend on the number of connections entering the node and the network conditions for each connection. We could require that each path send a packet, as proposed in Pipenet [17], but that could lead to long delays and denial-of-service (DoS) attacks. An adversary could prevent a mix from firing by preventing an initiator or number of initiators from sending messages to the mix.

In an attempt to remove enough timing patterns without introducing such risks, γ-buffering uses a more flexible scheme. If there are p incoming connections to a node, then the node buffers at least γ*p packets before sending the batch, where γ is a fraction that can vary depending on the system needs.

When γ=1, each node will get an average of one packet from each incoming connection before sending the batch. For $\gamma > 1$, larger batch sizes help ensure security at the cost of higher average latencies. For $\gamma < 1$, we can ensure a low latency when no more than $(1 - \gamma)$* p connections are blocked or delayed. One of the primary benefits of

γ-buffering is in its flexibility. The buffering parameter γ can be controlled at the system level or by individual nodes. When controlled by nodes, different nodes may offer different γ values, allowing users to select paths with higher or lower amounts of buffering to suit their needs. If the ratio of users to nodes grows, meaning that there are more connections per node, then γ may be dropped due to greater cover traffic.

The other primary benefit of γ-buffering would be its resilience to changing network conditions and DoS attacks. When other users' connections fail or are delayed, this creates delays for the user. Some delay is good; if the user sends traffic too aggressively, then she will be subject to easier traffic analysis. The delay is bounded, however, as long as at least the user's own traffic continues to reach the node. End-to-end attacks will likely fail, even when conducted by a global adversary, as long as some other paths continue to operate.

Another way in which γ-buffering would be resilient to active attacks is that delays introduced along a user's path will multiply to show delay on many other paths in the system. The increased delay, introduced early in the path, will likely delay an entire batch of packets at the next node. These delayed packets cause further upstream delays, with an effect that is exponential in the length of the path. An attacker may observe a delay that has propagated along either the original path or one that has been introduced by the buffering, making it difficult to distinguish false positives from correct matches. Unlike the addition of random delays along the path, γ-buffering introduces delays simultaneous with other delays in the system, so that timing effects occur together and are much less useful to the attacker for differentiating between paths.

CHAPTER 4

EXPERIMENTS

We created an application to work on a real network to collect our data. The experiments were conducted on the distributed platform PlanetLab.

4.1 PlanetLab

PlanetLab is a global research network that supports the development of new network services. PlanetLab is a collection of computers distributed over the globe [21]. Most of the machines are hosted by the participating research institutes and all of the machines are connected to the Internet. All the computers on PlanetLab run a Linux-based operating system from a read-only media. The key objective of PlanetLab's software is to support distributed virtualization—the ability to allocate a *slice* of PlanetLab's network-wide hardware resources to an application. This allows an application to run across all (or some) of the machines distributed over the globe, where at any given time, multiple applications may be running in different slices of PlanetLab [21]. In order to provide consistent expected performance and dedicated resources on a community shared network, PlanetLab allows for reservation of resources through Sirius Calendar Service. Reservation entitles the slice a dedicated 25% of CPU capacity and 2 Mbps of link bandwidth. All other active slices share the remaining resources with equal priority. Sirius is still under development and does not reliably schedule and allocate resources. PlanetLab has over 800 computers located in over 400 locations as of October 2007.

A distributed and geographically disperse Linux-based real network test bed was an ideal platform for collecting data for this research. Global distribution of PlanetLab nodes allows for real network latency and provides a real test bed to test and create robust applications. A slice was created with over 300 nodes for our experiments on PlanetLab. Sites with Internet2 connection were excluded. PlanetLab has various deployment and monitoring tools available, but we developed our own to meet our needs as the existing tools were lacking features needed to manage our experiments.

4.2 SubRosa

4.2.1 Overview

An application, SubRosa, was developed to run on PlanetLab to collect data for this research. We designed SubRosa to emulate the behavior of a Tor-like network over the unreliable UDP transport. It is written from scratch without borrowing code or design from Tor. The primary objective of SubRosa is to collect timing data for our research and hence, encryption is not implemented. Only information visible to an adversary in the presence of encryption has been used to perform timing analysis in our experiments. The Tor project's existing proposal for implementing Tor over UDP, as well as Voice over IP (VOIP) traffic, were factors in choosing UDP as the transport for this application and the research. However, due to the restriction placed by the community-shared, non-dedicated resources of PlanetLab, no experiments were conducted for VOIP data.

SubRosa is written entirely in C and consists of various components; namely, subrosa, srserver, srclient and srsink. subrosa is the controller program. It starts, stops and checks the status of the server, client and sink. The Controller is also used to create a daemon of the server, client and sink. subrosa keeps track of the versions of all components in the system including the configuration file used to run the application. subrosa was used to deploy the application on PlanetLab and keep track of the versions while conducting experiments.

- srsink is the destination of the data. It is a simple UDP echo server designed simplistically for high throughput. Since sink is not a required component for this research, in order to simplify the data collection process and not to waste shared resources, it was not deployed during the experiments.
- srserver is the server piece of the application. It can act as first, last or any intermediary node on the path. The server is discussed in greater detail later in this section.
- srclient is the client piece of the application. It represents the client and is responsible for generating data on the network. The client is discussed in greater detail later in this section.

SubRosa is designed to collect timing data for various defense algorithms against timing analysis and, hence, it is versatile. Hooks and exits are designed to collect timing data as well as easily implement different algorithms for collecting data.

All the variable parameters are read from the configuration file. Appendix A lists the configurable parameters.

Algorithms currently implemented:

- No Defense
- Defensive Dropping
- γ Buffering

Implemented packet generators:

- Constant Bit Rate
- Exponential

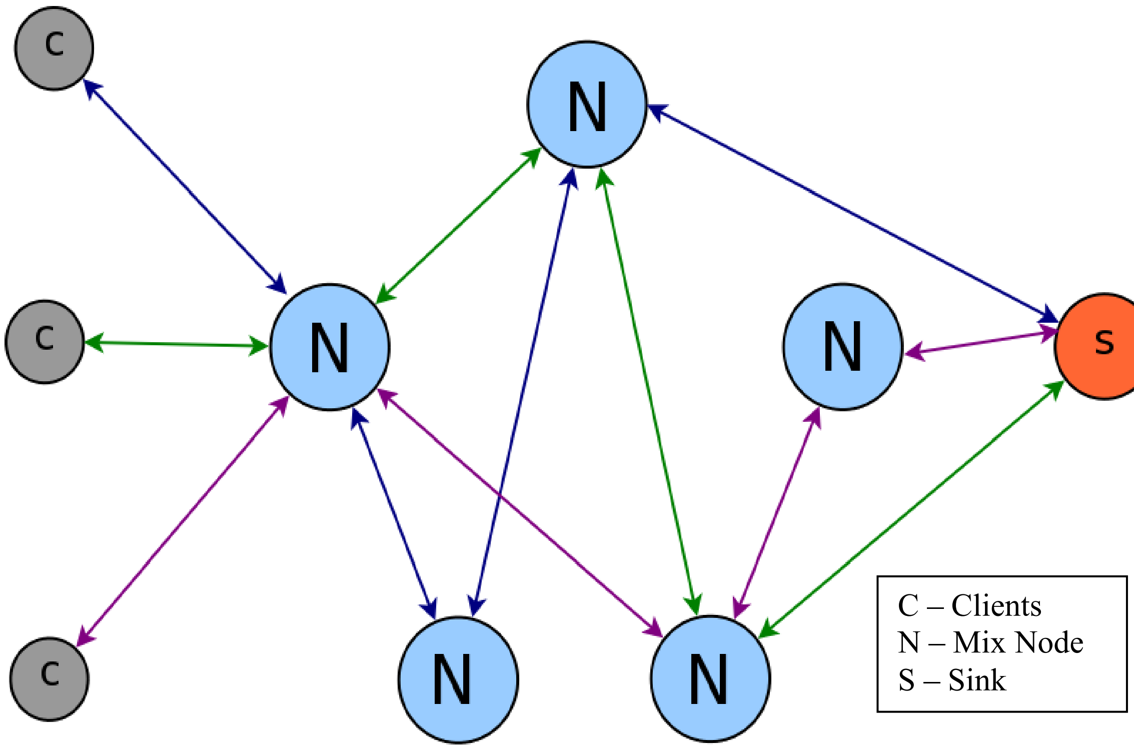

Figure 4.1 - Data flow in a mix network

Data flow in an Onion Router network is shown in Figure 4.1. N represents srserver, C represents srclient and S represents srsink. The client selects the path length and the nodes on the path. After selection, the client starts the circuit building process. Steps involved in the circuit building process are as listed and shown in Figure 4.2 and described here:

Step – 1 establishes the connection with the first node on the path (N1)

Step – 2 extends the connection to the second node on the path (N2) through N1

Step – 3 extends the connection to the third node on the path N3 through N2 using N1 as a relay

Step – 4 opens connection to the destination (S) through N3, using N1 and N2 as a relay N3 on receiving response from S, forwards the response to C, using N2 and N1as a relay

Step – 5 once the circuit is established, C uses N1, N2 and N3 to communicate with S using the same circuit. S is unaware of this handshake.

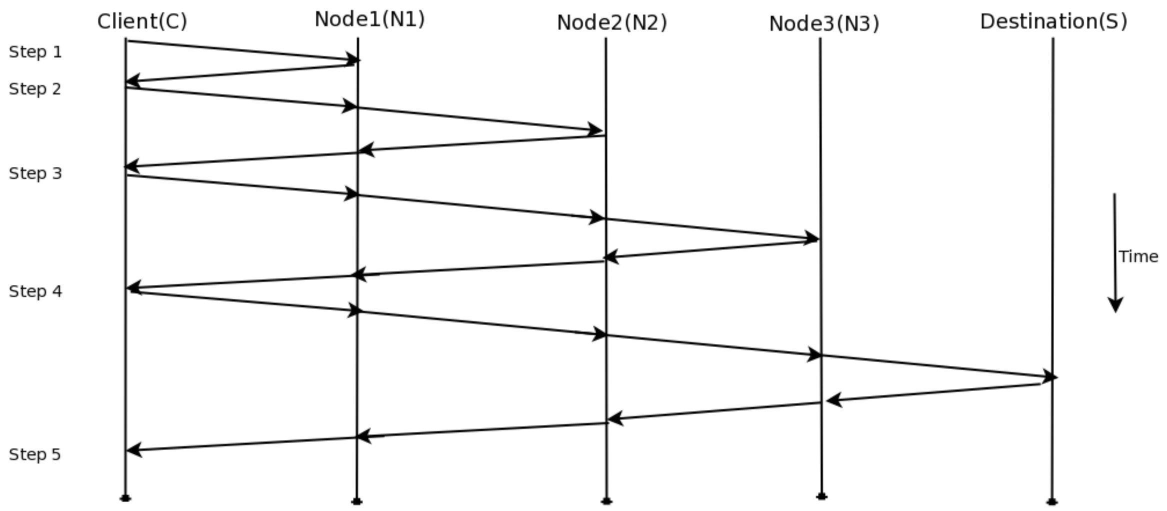

Figure 4.2 - Circuit building process – Steps as described above

4.2.2 Methodology

In our experiments, we fixed the path length to three. Since we did not use the sink, the exit node on the path acts as a sink and generates the response. All the traffic generated for the experiments was constant bit rate.

Five PlanetLab nodes were selected based on the load and the available bandwidth to act as servers for our experiments. Due to the nature of the test bed, if and when a node was not available or was heavily loaded, that node was replaced by another carefully chosen node. Uptime of more then 3 days, no bandwidth restrictions and less then 0.5 seconds response time were the criteria used for selection of the server nodes. Servers were selected by manual use of CoMon monitoring infrastructure for PlanetLab. All the servers were closely monitored during the experiments using CoTop slice based top for PlanetLab. Results were not considered in the analysis if the node did not perform consistently for the duration of the experiment.

More then 300 nodes were added to the slice to act as a client. For each experiment, clients were chosen at random. At the startup, clients randomly choose three out of five available nodes. Circuits are then established with those three nodes on the path.

The duration of our experiments was approximately 15 minutes each. Logs were collected for analysis after each experiment. Timing data were extracted from the logs using Perl scripts. Timing data were converted to zero base time to avoid time zone issues and to accommodate the scripts for analyzing the data which were provided by

[2]. Experiments were conducted for 25, 50, 75 and 100 clients and packets were generated every 300ms and 100ms. Data was also collected for various run with same parameters to take the average.

4.2.3 γ Buffering

γ Buffering is implemented on the servers. Multiplier γ is configured at the start of the application. γ is multiplied with the number of active circuits on the node to obtain the number of packets to buffer, before queuing it on the send queue. During the buffering period all the packets received are stored in a list in random order. Once the desired number of packets is buffered, the packets from the list are put on the send queue and sent out using a pool of threads. The thread pool is created at the startup and is kept alive for the life of the application to avoid overhead and improve the response time. Five threads pool were used for γ buffering. γ buffering was implemented with a delay of 180 seconds after the server started, to avoid lockups during the circuit building process. The packets sent during the first 4 minutes were discarded to compensate for the startup delay. γ values of 0.5, 1.0 and 1.5 were used for the experiments.

4.2.4 Defensive Dropping

Defensive dropping is initiated by the client. The client, at random, selects the packets to be dropped and marks it to be dropped at the intermediary node. The drop command is set in the header for the intermediary node and, hence, no other nodes on the path are aware of the dropped packets. To drop a certain percent of packets, a random number between (0, 1] is compared with the configured drop rate. If the number is less then the drop rate, a packet is marked to be dropped at the intermediary node. The server upon receiving a packet with a drop command set, logs the packet and stops further processing of the packet by discarding it form the receive queue. Drop rates of 20 percent and 50 percent were used for the experiments.

4.2.5 No Defense

Base line data without any defense against the timing analysis were also collected for all the 25, 50, 75 and 100 nodes. Packets were generated at every 100 milliseconds and 300 milliseconds other parameters were kept in line with other experiments.

CHAPTER 5

RESULTS

In this section, we present the results of our experiments on timing analysis of a Tor like network. We show relative effectiveness of various defenses and also compare it against the timing analysis of anonymity systems without any defense.

We collected data using 25, 50, 75 and 100 clients for various defenses. Data was also collected without implementing any defense against timing analysis to establish the base line for comparisons

A Receiver Operator Characteristic (ROC) curve is a graphical representation of the trade off between the false negative and false positive rates for every possible cut off (threshold). Conventionally, the ROC curves show the false positive rate on the X-axis and 1 – the false negative rate on the Y-axis. We follow the same convention in representing the ROC curves. We plot different ROC curves on the same graph to visualize the relative comparison. Curves that are closer to the upper left-hand corner are better, in our case, better for an adversary and the worse case for an adversary would be a 45^{o} diagonal. Figure 5.1 shows the ROC curve for 100 client configurations for various defenses.

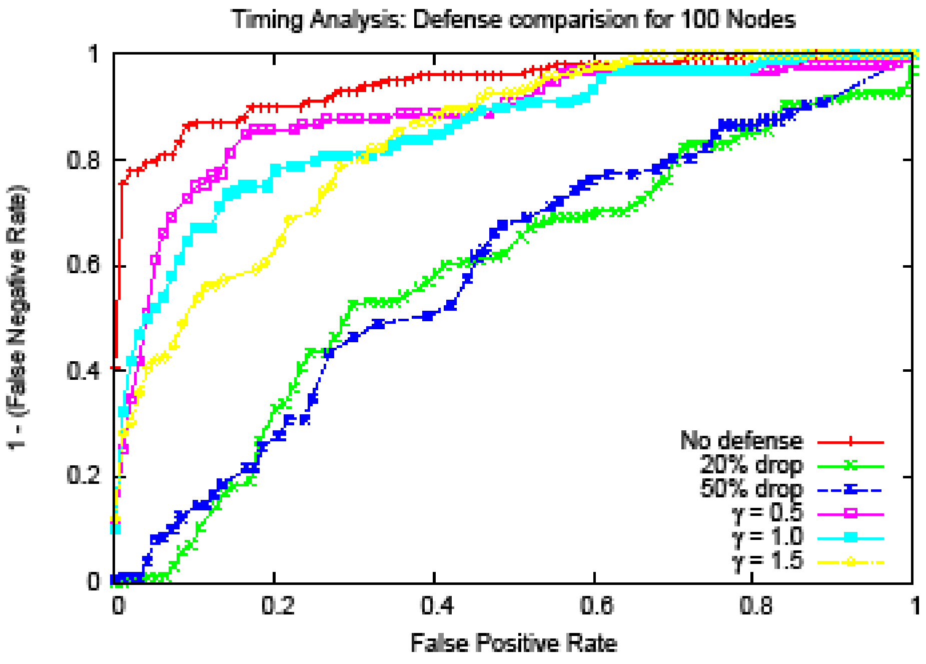

Figure 5.1 – Timing Analysis: Defense comparison for 100 nodes

We can see from Figure 5.1 how various defenses compare with no defense. The curve on the top represents the base line data without any defense, which is the best for an adversary. The defensive dropping curve is almost a diagonal at 45^{o} making defensive dropping the strongest defense among the ones compared. γ-Buffering does not perform as well as defensive dropping as a defense against timing analysis. With the increase in value of γ, the ROC curve is drifting away from the upper left-hand corner, which looks promising. However, due to the lack of data with higher γ values, the effectiveness of γ-buffering is not very clear.

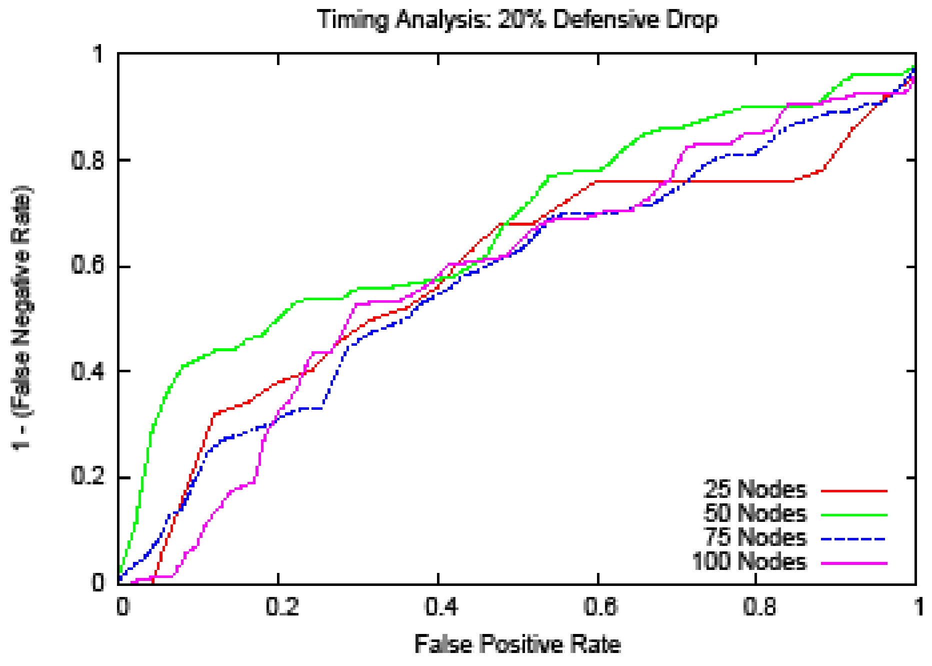

Figure 5.2 – Timing Analysis: 20% Defensive Drop

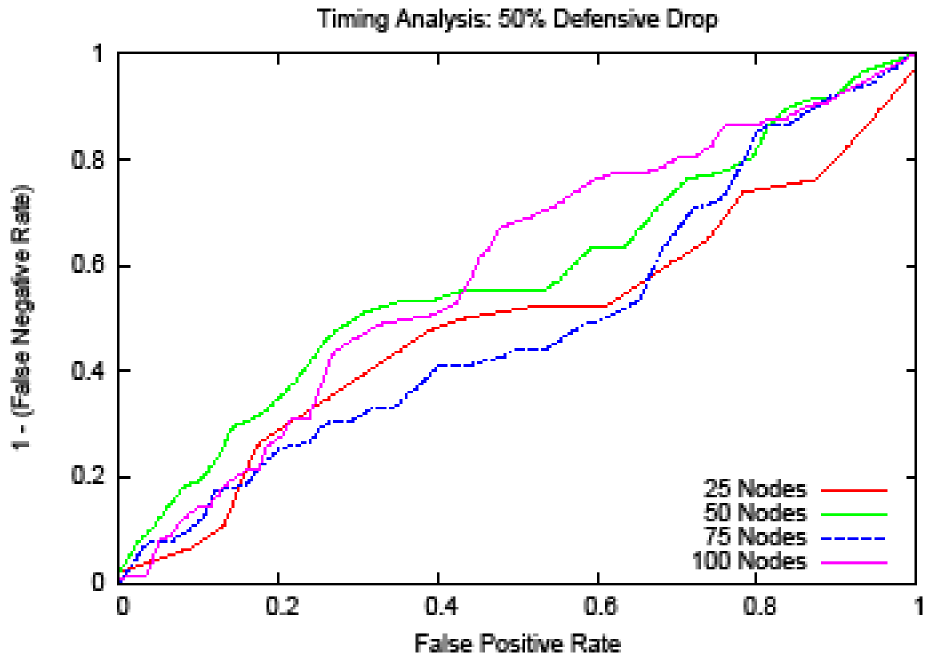

Figure 5.3 – Timing Analysis: 50% Defensive Drop

Figure 5.2 compares the effectiveness of the 20% defensive dropping rate against the timing analysis attacks with the varying number of clients. All the curves are very close to each other and almost at a diagonal. We can infer from Figure 5.2 that number of clients does not impact the defense. Figure 5.3 shows data for 50% defensive dropping rate with similar results.

Figures 5.4 - 5.6 compare the effectiveness of various γ values against the timing analysis with the varying number of clients. We see a consistent trend of improved defense with the increase in the value of γ. For γ=1, a 100 client network has the best defense. However, for γ=1.5 25 and 50 nodes seems to have better defense then 100 nodes. The results are not consistent in this regard for γ-buffering as they are with defensive dropping. Similar inconsistencies are also observed in the results for 25, 50 and 75 nodes, which are not shown here. Characteristics of the test bed, PlanetLab may be the factor for the inconsistency. The traffic on the network for γ-buffering was substantially greater than defensive dropping. In defensive dropping, up to 50% of the packets were dropped at the intermediate node.

Figure 5.7 displays the comparison of timing correlation of the base line data for various nodes. Even though over all, the base line data does not seem to have much protection against timing analysis attacks, the inconsistencies mentioned above still exist with 25 nodes networks being the worse and 50 nodes networks being the best, with 75 and 100 nodes falling in between.

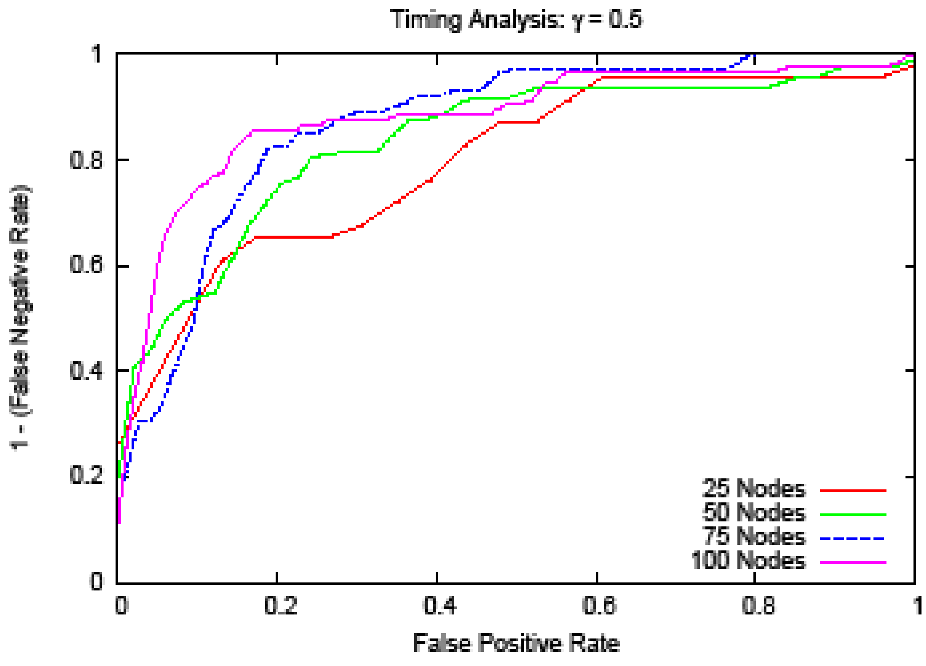

Figure 5.4 – Timing Analysis: γ-buffering for γ=0.5

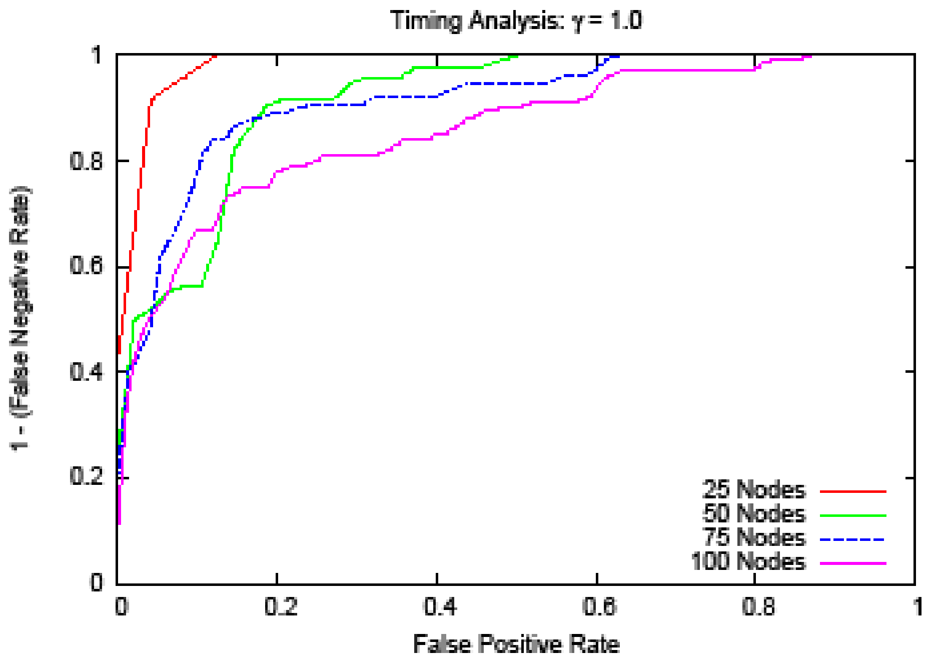

Figure 5.5 – Timing Analysis: γ-buffering for γ=1.0

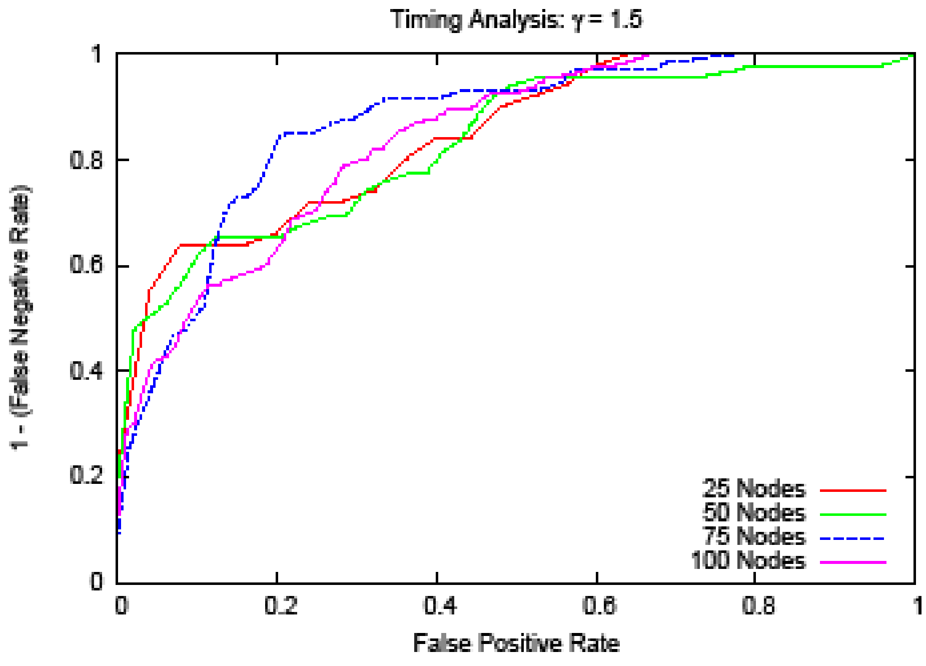

Figure 5.6 – Timing Analysis: γ-buffering for γ=1.5

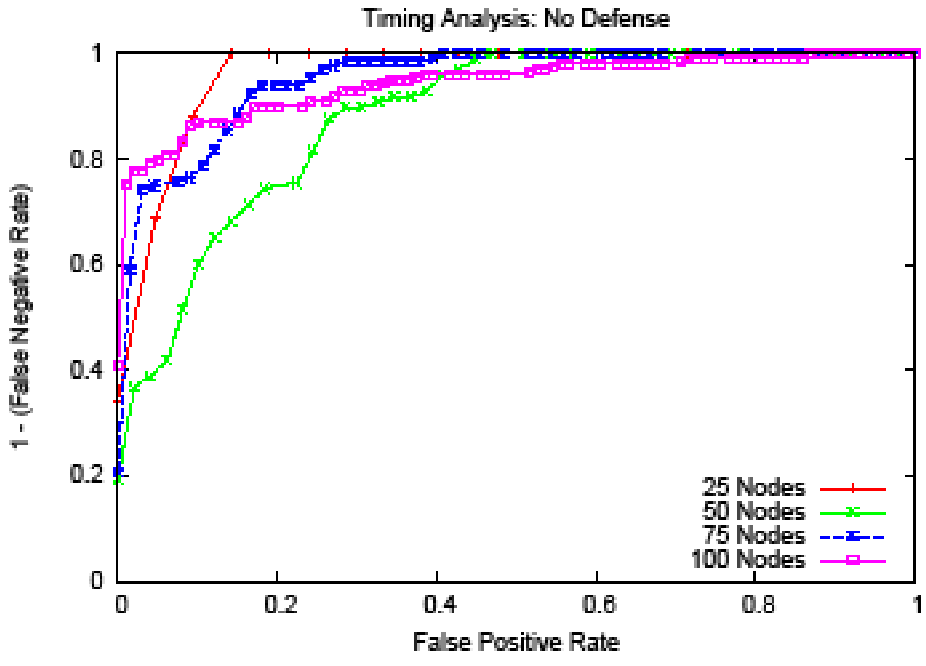

Figure 5.7 – Timing Analysis: No Defense

We conducted more experiments with the same configuration. Data was analyzed collectively to obtain average ROC curve. We kept the number of clients constant at 25. Figure 5.8 – Figure 5.11 shows the average ROC curves of various configurations for 25 clients.

Figure 5.8 shows ROC curve when no defense was used. The curve is on the upper left-hand corner as expected. Figure 5.9 compares 20% defensive dropping with 50% defensive dropping. We can see that 50% drop is marginally better then 20%. Hence 50% drop provides more anonymity at the cost of significantly higher bandwidth.

Figure 5.10 compares various γ values, the results are consistent, but the improvement with the increase in γ is marginal. The variation for ROC of $\gamma=3$ can be attributed to the changed network characteristics since this particular result was taken on a different day using the different set of servers. It is clear that even though γ=buffering has good properties to provide anonymity, it did not perform well as implemented in SubRosa. We assume compromised entry and exit node as a threat model in our study. Delaying packets on all the nodes on the path may provide much needed additional information to the adversary. γ-buffering might perform better if implemented only at the intermediately node, i. e. the delay should occur only at the intermediate node rather than all the nodes. However, we do not have any data at this point to validate the hypothesis and we leave this as a future work.

Figure 5.11 compares various defenses with no defense showing the effectiveness of defensive dropping and non effectiveness of γ-buffering.

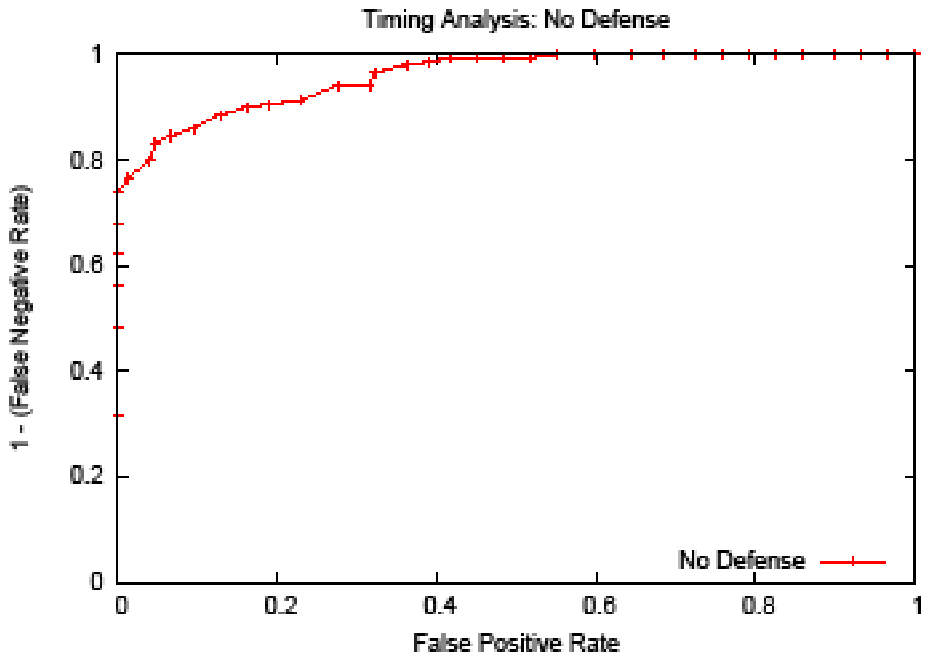

Figure 5.8 – Timing Analysis: No Defense – 25 Nodes

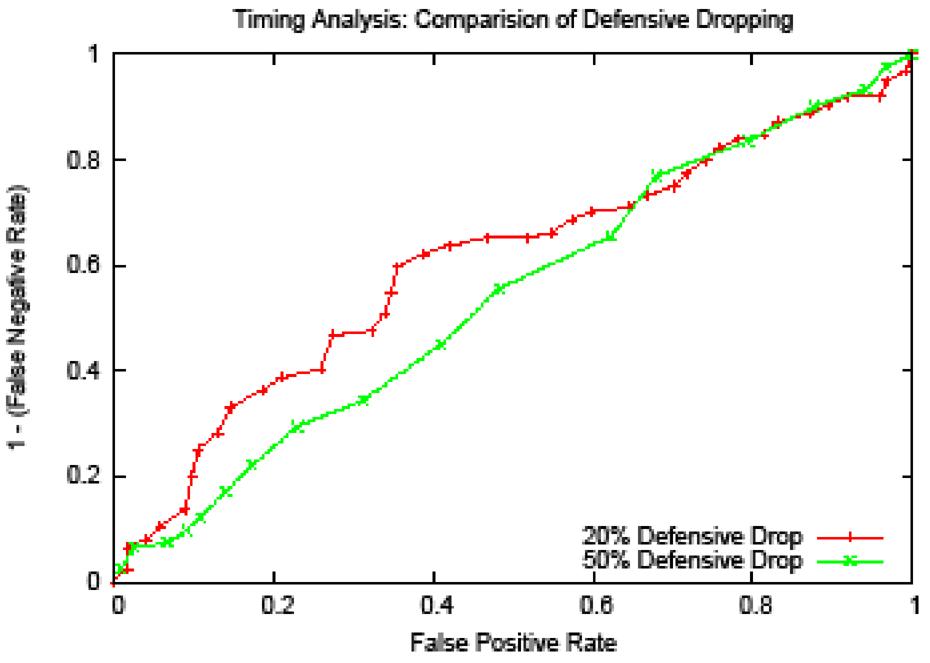

Figure 5.9 – Timing Analysis: Defensive Dropping – 25 Nodes

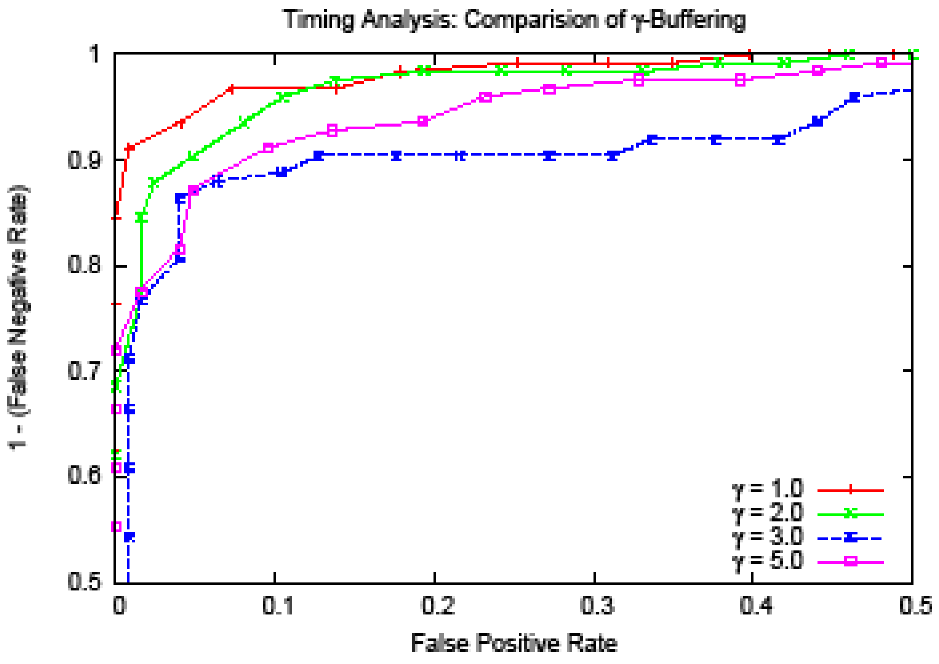

Figure 5.10 – Timing Analysis: γ-buffering – 25 Nodes

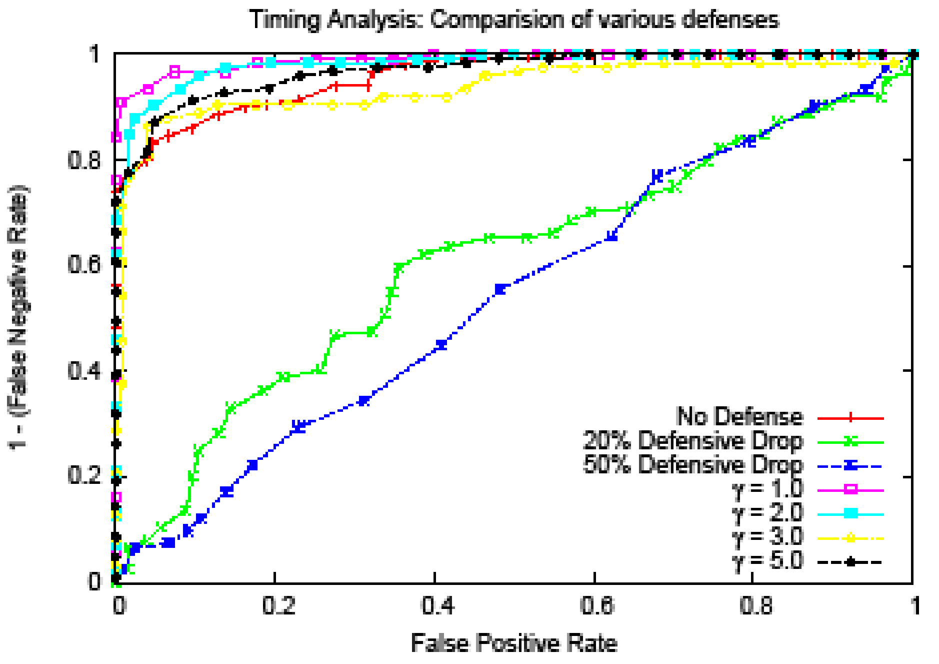

Figure 5.11 – Timing Analysis: Defense comparison – 25 Nodes

CHAPTER 6

CONCLUSIONS

To facilitate the research of low latency anonymous systems in general and timing analysis on low latency systems in particular, we developed SubRosa, an application to study timing analysis of real time anonymity systems. We modeled SubRosa on Tor like systems but choose UDP as the transport protocol with low latency real time application like VOIP in view.

We ran experiments on a real distributed network PlanetLab and collected network timing data. We also performed successful timing analysis attack on the data collected from PlanetLab and implemented known defenses to avoid the attack. We proposed a light weight defense against timing analysis attack on a low latency system, γ-buffering, based on the principle of mix design.

We studied relative comparison of defensive dropping and γ-buffering with no defense. Our results proved defensive dropping to be the best defense against timing analysis attack on a low latency mix network by a passive adversary.

6.1 Future Work

Results presented in this work are not consistent, especially for γ-buffering. More data needs to be collected to prove effectiveness of γ-buffering against timing analysis attack. Better collection of data could be facilitated by developing tools to automate the data collection process on PlanetLab. SubRosa needs to be evaluated in order to eliminate starvation of the packets in the queues for the inconsistent results of γ-buffering defense.

APPENDIX A

SUBROSA CONFIGURATION PARAMETERS

S. No	Section	Parameter	Value	Default	Description
	Common				[COMMON] section of configuration file
1		MaxSendQueue	Integer	1024	Maximum number of packets in the send queue
2		MaxRecvQueue	Integer	1024	Maximum number of packets in the receive queue
	Server				[SERVER] section of configuration file
3		BindIP	String	Loopback	IP on which server listens to, if blank, IP will be determined from hostname
4		ListenPort	Integer	Random	Port on which server listens to, if blank, random port greater then 1024 will be used
5		SendBufferType	Integer	0	0 – No buffering, 1 – Fixed number of packets, 2 – packets based on γ multiplier
6		MaxSendPoolWorker	Integer	1	Number of threads in the thread pool for sending data.
7		MaximumSendBuffer	Integer	1024	Maximum number of packets to buffer, value obtained by γ multiplier must be less then this value.
8		Buffer γ	Float	0.0	γ buffer multiplier, used only when SendBufferType = 2
	Client				[CLIENT] section of configuration file
9		BindIP	String	Loopback	IP on which client listens to, if blank, IP will be determined from hostname
10		ListenPort	Integer	Random	Port on which client listens to, if blank, random port greater then 1024 will be used
11		NoOfHops	Integer	3	Path Length
12		HOP	String	None	List of server nodes to select route from in the pattern HOPn=hostname:port, $n > 0$, $n < 11$
13		Destination	String	Loopback	Destination address
14		PacketGenerationType	Integer	1	Packet Generation:1 – Constant, 2 - Exponential
15		PacketGenerationRate	Integer	100	Packet generation rate in

S. No	Section	Parameter	Value	Default	Description
					milliseconds
16		PacketDropRate	Float	0.0	Percentage of packets to be dropped
17		BackoffPacketGen	Boolean	No	Stop generating packets if response is not received within 10 seconds
18		RetryCircuit	Boolean	No	Retry circuit building process if circuit was not established in 30 seconds
	Log				[LOG] section of configuration file
19		LogPath	String	.	Log output directory
20		NetLog	Boolean	Off	Network log on/off
21		NetLogFile	String	-	Network log file name, if blank, IP.nlog for server,N1_N2_N3_IP.nlog for client
22		NetLogSize	Integer	0	
23		SysLog	Boolean	Off	System log on/off
24		SysLogFile	String	-	System log file name
25		SysLogSize	Integer	0	
26		LogLvl	Integer	4	Log level: 0 – 7, 0 – Least details,7 – Most details

REFERENCES

1. D. Chaum. “Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms.” Communications of the ACM, 24(2):84{88, Feb 1981.
2. B. N. Levine, M. K. Reiter, C. Wang, & M. K. Wright (2004), “Timing attacks in low-latency mix-based systems,” in *Proceedings of Financial Cryptography (FC'04).*
3. G. Danezis (2004), “The traffic analysis of continuous-time mixes”, in *Proceedings of Privacy Enhancing Technologies workshop (PET2004)*, Vol. 3424 of LNCS.
4. Y. Zhu, X. Fu, B. Graham, R. Bettati, W. Zhao, “On flow correlation attacks and countermeasures in mix networks,” in *Proceedings of Privacy Enhancing Technologies workshop (PET 2004),* 2004.
5. R. Dingledine, N. Mathewson, P. Syverson, “Tor: The second-generation onion router,” in *Proceedings of the 13th USENIX Security Symposium.* 2004.
6. A. Pfitzmann and M. K¨ohntopp, “Anonymity, unobservability, and pseudonymity: A proposal for terminology,” Draft, version 0.14, July 2000.

7. C. Diaz, S. Seys, J. Claessens, and B. Preneel, “Towards measuring anonymity,” in *Privacy Enhancing Technologies - PET '02,* P. Syverson and R. Dingledine, Eds., Springer Verlag, 2482, 2002.
8. A. Serjantov and G. Danezis. “Towards an information theoretic metric for anonymity, in *Privacy Enhancing Technologies workshop (PET 2002)*, R. Dingledine and P. Syverson, Eds.,volume 2482 of LNCS, pages 41{53, San Francisco, CA, USA, 14-15 April 2002. Springer-Verlag
9. C. Díaz and B. Preneel. “Taxonomy of Mixes and Dummy Traffic,” in *International Information Security Workshops,* pages 215–230, 2004
10. U. Moeller and L. Cottrell, “Mixmaster Protocol Version 3, 2000”. [Online]. Available: http://www.eskimo. com/~rowdenw/crypt/Mix/draft-moeller-v3-01.txt
11. G. Danezis, R. Dingledine, and N. Mathewson. “Mixminion: Design of a Type III Anonymous Remailer Protocol.” in *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003
12. D. Kesdogan, J. Egner, and R. Buschkes. “Stop-and-Go MIXes: Providing probabilistic anonymity in an open system,” in *Information Hiding workshop (IH 1998),* volume 1525 of LNCS, pages 83{98, D. Aucsmith, Ed., Springer-Verlag, Portland, Oregon, USA, 14-17 April 1998..
13. “JAP, Anonymity and Privacy.” [Online]. Available: http ://anon.inf.tu-dresden.de

14. R. Dingledine, N. Mathewson, and P. Syverson. “Challenges in deploying low-latency anonymity,” NRL CHACS Report 5540-265, 2005.

15. M. Freedman and R. Morris. “Tarzan: A Peer-to-Peer Anonymizing Network Layer,” in *Proc. ACM Conference on Computer and Communications Security*, Nov 2002.

16. A. Back, I. Goldberg, and A. Shostack. “Freedom 2.0 Security Issues and Analysis. Zero-Knowledge Systems, Inc.” white paper, Nov 2000.

17. W. Dei. “Pipenet 1.1”, August 1996. [Online]. Available: http://www.eskimo.com/ weidai/pipenet.txt.

18. A. Pfitzmann, B. Pfitzmann, and M. Waidner. “ISDN Mixes: Untraceable Communication with Very Small Bandwidth Overhead,” in *Proc. GI/ITG Communication in Distributed Systems*, Feb 1991.

19. P. Syverson, G. Tsudik, M. Reed, and C. Landwehr, “Towards an Analysis of Onion Routing Security,” in *Workshop on Design Issues in Anonymity and Unobservability,* July 2000.

20. Shmatikov, V. and Wang, M. H., “Timing analysis in low latency mix networks: Attacks and defenses," in *11th ESORICS*, 2006.

21. PlanetLab, [Online]. Available: http://www.planet-lab.org

BIOGRAPHICAL INFORMATION

Hatim A. Daginawala graduated with a degree Master of Science in Computer Science and Engineering from University of Texas at Arlington in the fall of 2007. Hatim has been part of iSec@UTA, an information security lab since its inception and was involved with the design of network and other lab infrastructure. His research interests include network/information security, anonymity and privacy on internet, operating systems and cryptography.