LILAC: ARCHITECTURE FOR ANONYMOUS LIGHT WEIGHT COMMUNICATION

by

ANKIT UPADHYAYA

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

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2013

To my parents without whom I would not be where I am today and my sister Shreya Upadhyaya for her unconditional support.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Matthew Wright for supervising me on this thesis and helping me in every step of my work. His patience, motivation and guidance helped me immensely during my research and the writing of this thesis. I am grateful to Dr. Donggang Liu and Dr Christoph Csallner for their invaluable advice and interest in my research and for taking time to serve on my thesis committee. I would like to thank all the members of the Information Security Lab for creating a positive and enjoyable work environment and being available at any time to discuss problems. I would like to express my honest acknowledgments to Mr Irie Bito for always motivating me and mentoring me throughout this work. I would also like to thank my friends Subhasish Dutta Chowdhuri, Jaineel Mehta, Harshal Patel and Mehrdad Amirabadi for being there with me in my difficult times and supporting me throughout.

Finally, I thank my parents for their constant encouragement and unconditional support and my sister for her consistent motivation.

August 05, 2013

ABSTRACT

LILAC: ARCHITECTURE FOR ANONYMOUS LIGHT WEIGHT COMMUNICATION

Ankit Upadhyaya, M.S.

The University of Texas at Arlington, 2013

Supervising Professor: Matthew Wright

We present lilac; a circuit based low latency anonymous light weight instant messaging communication system. This browser based chat platform provides anonymity in context with unlinkability and unobservability for the secured real-time communication. The system uses a relay based anonymization mechanism where circuits are built and routed over a set of trusted nodes in the network. Unlike other typical instant messaging systems, Lilac uses user pseudonymity and ephemeral message exchange, which eliminates conventional user registration and storing the messages for the future retrieval; this leaves no footprints behind in the system in terms of user information, who talks to whom and the messages exchanged between the users. This system is deployed to be used for one-to-one communication and also an experimental group communication up to five users in a group. Lilac works on the real-world internet, requires no configuration, browser add-ons and operating system modification on the client side; thus it provides a hands-on ease of access on both desktop and mobile browsers acquiring the anonymity by reasonable tradeoff amongst the deployability, compatibility, usability and efficiency. Lilac demonstrates the primary design and workings of core modules deployed on real-world internet. We have briefly described our experience with the experimental setup of lilac platform discussing the key ways in which it challenges our current concept of anonymity and closing with the future work for the system.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Instant messaging (IM) is a real-time chat communication which offers a short message exchange between communicating partners over the internet. IM comprises of short messages since a sender can quickly get a response from a receiver. IM may address to point-to-point communication as well as multicast communications from a sender to many receivers. This type of communication can be indicative of imminent purposes or sensitive information e.g. combatant commands and military control systems, a patient's medical record or friends' and family matter in civilian locales. With the increased use of instant messaging on internet in all aspects of daily life, the realization has dawned that privacy and confidentiality are essential and important requirements for the success and wide-spread use of real-time chat communication systems. Encryption alone does not provide the level of privacy required by users. It focuses primarily on protecting the confidentiality of the transmitted data, while the identities of the communicating parties remain unprotected. Traffic analysis can easily uncover information about the participants in a distributed application. Chat privacy is an important criterion for concealing the sender-receiver identity, confidentiality and integrity of the transmitted data. In the real-time communication, it is essential to protect the identities of the correspondents such that it could relate the information exchanged between communicating partners in a chat session. These messages can contain sensitive information; the addressing, timing and volume of traffic can in some cases leak as much information as its content [16]. Despite this, there are very few committed systems which have been deployed to provide strong anonymity in real-time chat communications.

Lilac aims to provide an experimental deployment of a normative model of anonymous instant messaging. The main goal for this design is to abandon the linkabilty between the communication partners from an eavesdropper by constructing multi layered encrypted circuits between end users. To achieving the unobservability, we have adopted always on communication by adding the heartbeats in the circuits and subsequently concealing the identity of the communicating partners from an adversary, hence providing the anonymity. However it comprises of several other considerations directed towards the deployment. Figure 1.1 gives an overview of the Lilac system.

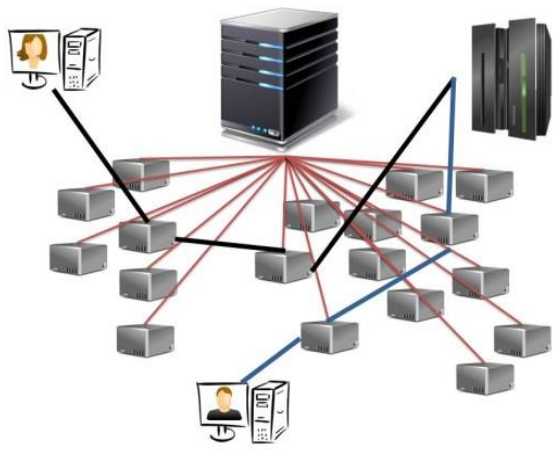


Figure 1.1 Lilac System Overview

Simple real-time communication platform: At the core of the design is to develop a real-time anonymous one-to-one chat application exchanging messages between two actively communicating users. This light-weight communication requires a low bandwidth connection; however the high bandwidth connection can be used in relaying the circuits in case of using client as proxy node and to implement the redundant noise in the communication channel. Like any other popular chat applications (Yahoo! Messenger[1], Google Talk[2]) this design comprises a chat display, input box, and a list of currently online users. One thing is to be noted here, that is – like other popular chat applications, the user status of being online and available is different in this system; an online user may not be available to chat at the same time. This distinction is derived because Lilac implements always on communication through adopting the "heartbeat" implication, which means a user remains online throughout the session to achieve the

[1] http://messenger.yahoo.com/
[2] http://www.google.com/talk

unobservability from an eavesdropper. Thus in certain situation to get to the rendezvous point, a user should send a ‘ping message’ to a desired user, if the responder decides to communicate in return she may respond a ping response, this would change the user status from online to available. In future work this ambiguity will be addressed by classifying the user status into one more category: ‘idle’, thus it will exclude the pinging step. An input message can contain maximum one hundred and forty characters including alphanumeric, punctuations and whitespaces, in a message. This limitation is set to achieve the fixed size cells archetype synchronizing with a heartbeat dummy message.

Deployability: This system is designed to achieve the most possible deployability over all major web-browsers. There should be no client side installation, configuration or maintenance required to get this system rolling. It is designed and tested to work only in the browser, since this is a TCP/IP based asynchronous and bidirectional web-application.

Usability: A complicated system gets fewer users, and the number of users is directly proportional to the anonymity measurement [6]. Thus a simple to use application is the prime goal for development. Lilac is a very simple to use chat application. It has UNIX Terminal like user interface. A very first step is to enter the user credentials in the input box and that’s it, rest of the proceedings is done asynchronously on the client-end side. This takes the user into the system. User gets a list of all the online users, she can either chat one-to-one or multicast a message (group message) in the chat window.

Cross-browser compatibility: This application is aimed to be used and compatible on all major desktop browsers (Google Chrome[3], Internet Explorer[4], Firefox[5], and Safari[6]). Moreover,

[3] http://www.google.com/chrome
[4] http://microsoft.com/ie
[5] http://mozilla.org/firefox

since JavaScript is widely supported on all major desktop and mobile browsers, and this application only requires JavaScript support in a browser it should also be accessible from any mobile-browsers supporting JavaScript. We have tested this application on major browsers with certain testing-parameters, which is discussed in testing and result section.

No footprints: This system does not store chat record for future retrieval. The communication messages are exchanged in real time sessions and are destroyed along with the session itself. No footprints have been left behind in terms of user information and content shared between the users. Each message is encrypted end-to-end with a symmetric key computed at user ends only. Moreover, in the circuit at each hope from a node-to-node is encrypted with a session key. Thus only communicating clients can read the message within the session. Once a session expires, the exchanged data between the users and the circuits asynchronously get destroyed.

This work is aimed to provide a preliminary design through deploying the prime goals of the system on real world internet. We have studied three aspects of the system through the real world deployment on internet – circuit establishment using two relay nodes, a lilac application server and a directory server; measuring the crypto overhead and latency in the system; and anonymity provided in the present system.

[6] http://apple.com/safari

CHAPTER 2

BACKGROUND

In this chapter we provide the insight of the related work done to design the lilac architecture. Chat privacy is the primary concern in deploying the system; we have summarized the privacy aspects in the following sub section. There are existing anonymous communication systems which are designed with distinct approaches to provide the anonymity at different levels. Anonymous communication systems were first proposed by Chaum [1] in which the message to be anonymized is relayed through a series of nodes called mix nodes. A mix can be thought of as a server which accepts incoming connections and forwards them in such a way that an eavesdropper cannot easily determine which outgoing connection corresponds to which incoming connection. Moreover, since any given mix can be compromised, traffic is usually routed through a chain of mixes. Furthermore the anonymous communication systems can be classified into two categories: message-based high latency architecture and connection-based low-latency systems. High latency applications are those which do not involve real time responses such as email systems. Low latency applications on the other hand are those that need real-time responses; instant messaging. The Onion Routing (OR) paradigm is focused on practical systems for low-latency Internet-based connections that resist traffic analysis, eavesdropping, and other attacks. OR prevents the transport medium from knowing who is communicating with whom – the network knows only that communication is taking place. In addition, the content of the communication is hidden from eavesdroppers up to the point where the traffic leaves the OR network. Tor is the second generation of the OR design which provides low latency anonymity for TCP-based applications. It is widely deployed anonymous communication system which primarily concentrated to ensure low enough latency to facilitate the use of interactive applications such as instant messaging and web browsing. Another prime objective in deployment is to adopt an appropriate light weight key exchange protocol paradigm in the application, such that it adds only a reasonable crypto overhead in the system execution.

We conclude this section with reviewing the cover traffic and data loss prevention aspects for the anonymous communication systems.

2.1 Chat Privacy

Privacy is the most important aspect of chat communication. The real time chat communication comprises of messages exchanged between communicating partners. This type of communication can be indicative of imminent purposes or sensitive information e.g. combatant commands and military control systems, a patient's medical record or friends' and family matter in civilian locales. Chat privacy is the most important concealment criteria for the sender-receiver identity, confidentiality and integrity of the transmitted data. In the real-time communication, it is very important to protect the identities of the correspondents such that it holds up the messages exchanged in a chat session. These messages can contain sensitive information; the addressing, timing and volume of traffic can in some cases leak as much information as its content [16]. Moreover malicious hackers have consistently used IM networks as vectors for delivering phishing attempts, "poison URLs", and virus-laden file attachments. This requires a mechanism to conceal the messages exchanged in chat-communication channel as well as the communicating partners' identities from the eavesdroppers and filtering the messages to prevent the malicious content sharing in the chat session. Allowing only plain text in the communication and designing a model that only displays the communication in real time, such that it does not store the commutation log in database are the likely adoptions in the lilac design to ensure the chat privacy.

2.2 Types Of Anonymous Communication Systems

According to Pfitzmann and Waidner [18], there are three types of anonymities that can be provided by anonymous communication systems: sender anonymity, receiver anonymity, and unlinkability of sender and receiver. Sender anonymity means that the identity of the information sender is hidden, and receiver anonymity means that the identity of the information receiver is hidden. Unlinkability of sender and receiver refers to the property that the sender and receiver of a communication cannot be identified even if the sender and receiver are known to be of communicating with someone. Since anonymity is the state of lacking identity, anonymous communication can only be achieved by removing all the identifying characteristics from the network flows.

Anonymous communication systems can be classified into two categories: message-based high latency architecture and connection-oriented low-latency systems. Both types of systems are built on the idea proposed by Chaum [1] whereby unlinkability is provided by using a sequence of nodes between the sender and its receiver and encryption is used to hide the message content. An intermediate node knows only its predecessor and its successor. High-latency anonymous systems provide conceivably optimal anonymity for the systems which do not require quick responses such as email. This design schedules one message per path and new path is created for a new message. Mixmaster [8] and afterward mixminion [7] are implemented to deliver remailer anonymity for sending and receiving e-mail. A message is sent in fixed-size packets to the remailer server, with embedded instructions on where to send them next. Server reorders messages, preventing anyone watching them go in and out of remailers. Although this anonymous remailer architecture has turned out to be an economical implementation with considerable efficiency against timing analysis, it lacks to serve the real-time applications such as Instant messaging, VoIP, and SSH connection where a significant delay is not at all affordable.

On the other hand low-latency systems use a connection-based path for a period of time and the data is sent as a stream of packets over the same path. The Onion Routing project including Tor [2] aims to provide low-latency communication architecture [9]. It is primarily concentrated on TCP-based concurrent and asynchronous applications like web-browsing, instant messaging and secure shell. In this distributed overlay network architecture, clients choose a path through a secure tunnel and build a circuit, in which each onion node knows only its predecessor and successor, but no other nodes in the circuit. This circuit circulates a small chunk of layered segments which are unpeeled (like an onion) by a symmetric key at each node and relayed subsequently. Another notion in low-latency systems is to adopt the mixes. The mixes are derived using a sequential set of cascading proxy servers which receive incoming messages from multiple senders, shuffle them, and send them out in random order to the next set of mix nodes or to the exiting nodes. This architecture is implemented in deployed systems such as ISDN-mixes [13] which proposed a model that allows phone conversation to be anonymized, web-mixes which pretty much followed the same paradigm for web traffic and recently the Java Anon Proxy (JAP) has been implemented and actively running at the University of Dresden [14]. Reiter and Rubin developed Crowds [19], which uses a group of nodes that serve as proxies for a given initiator from the group. A message is routed from the originator to a series of proxies, forming a path for all upcoming messages from the originator. Each receiving proxy decides based on a probability of forwarding (pf), whether to extend the path through another proxy chosen at random with uniform probability or to become the last node on the path and communicate with the responder directly. This path keeps updated at a regular interval. The time limit allows nodes that join the protocol to add their paths at the same time as all other nodes; otherwise new paths may be easily attributed to recently joined nodes. To attack Crowds, a number of attackers may simply join the crowd and wait for paths to be reformed — a periodic occurrence, usually hourly to gain the routing mechanism and analyzing the network traffic [20]. Since such systems are not suitable enough for a persistent communication. Figure 2.1 illustrates generic anonymous mix systems.

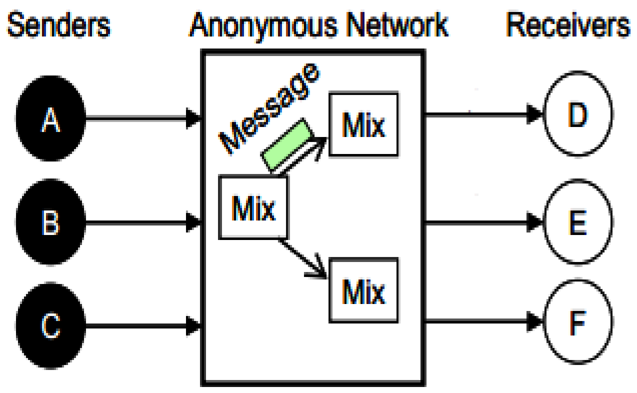


Figure 2.1 Anonymous Communication Systems

Although these deployed systems are designed to provide real-time anonymous communication, they work in synchronous fashion which is not well suitable for the widely deployed bidirectional asynchronous TCP/IP networks [15]. We focus on low-latency systems whose main purpose is to protect the privacy of interactive real time internet chat communications.

2.3 Onion Routing and Tor

Tor is the second generation of the onion routing design which provides low latency anonymity for TCP-based applications [2]. The primary design goal of Tor is to ensure low enough latency to facilitate the use of interactive applications such as instant messaging and web browsing. Tor's system architecture comprises of Tor routers, which are volunteer-operated servers, a set of trusted directory servers that advertise information about the Tor routers such as their IP addresses, public keys, exit policies, self-reported bandwidth capacities etc. and Tor proxies (or clients). Tor clients query one of the authoritative directory servers to obtain a signed list of the available Tor routers and then establish paths, or virtual circuits, through the Tor network by choosing precisely three Tor routers and establishing a shared symmetric key with each, using authenticated Diffie-Hellman and a telescoping key agreement procedure. The client encrypts their data in fixed 512 byte cells in a layered fashion with each key and sends the encrypted cell to the first router on the circuit, called the entry guard. The entry guard removes one layer of encryption using the symmetric key shared with the client, revealing the IP address of the middle router. The cell is forwarded in this manner, removing one layer of encryption at each router until the final router in the circuit, called the exit router, removes the last layer of encryption, revealing the cell's destination. The exit router finally forwards the message to the destination. The entry guard only knows the client's identity and only the exit router knows the destination's identity. More details about Tor can be found in its design document [2].

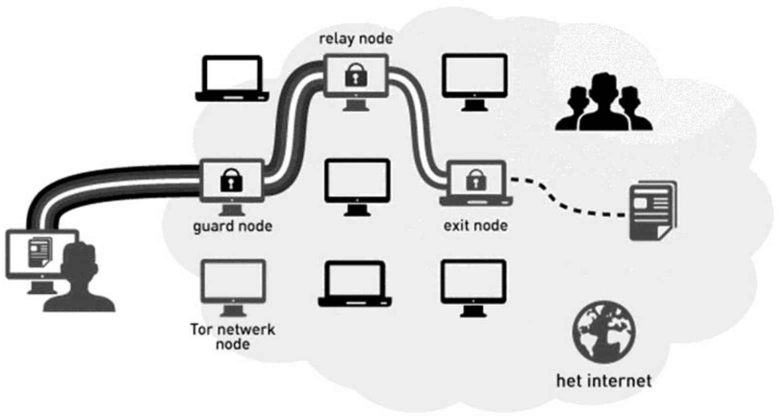


Figure 2.2 Tor's System Architecture

Entry guards are Tor routers that are used as the first node in a client's circuit. To mitigate the threat from adversaries setting up Tor routers and profiling a large number of clients over time, only those routers are chosen as entry guards which have high uptime and high bandwidth. Clients choose a fixed number of entry guards (three by default) to use on their circuits. By default, clients choose precisely three entry guards to use for their circuits. Exit routers are the Tor routers that allow connections to leave the Tor network. Each router's entry guard status and exit policy are advertised by the trusted directory servers. Tor clients query one of the authoritative directory servers to obtain a signed list of the available Tor routers, their public keys, bandwidth advertisements, exit policies, uptime, and other flags indicating their entry guard status and other information. Routers for each position of the circuit are chosen in proportion to their self-advertised bandwidth. Fig. 2.2 provides an illustration of Tor’s system architecture.

2.4 Tor Circuits

The initiator chooses the available nodes from the directory server. These chosen nodes are ordered to provide a circuit through which the messages would be transmitted. No node within the circuit, except for the exit node, can infer where in the chain it is located, and no node can tell whether the node before it is the originator or how many nodes are in the circuit. Using the asymmetric key encryption, the initiator encrypts the create cell message with the public key of the first node. This cell contains the circuit ID and a request for the receiving node and the initiator’s half of the Diffie-Hellman handshake to compute the shared secret point. Then the entry node responses to the initiator with its half of the Diffie-Hellman handshake and a hash of the computed shared secret such that the initiator can verify the shared secret between them. Now the initiator and the entry node have their respective half of the Diffie-Hellman handshake, they can derive a shared secret and subsequently would use this shared secret key in symmetric encryption. This shared secret key is called a session key, since it is used with the session between these nodes. A relay cell, as opposed to a command cell like the create cell, is not interpreted by the receiving node, but relayed to another node. Using the already established encrypted link, the originator sends the entry node a relay extend cell, which is like any relay cell, only that it contains a create cell intended for the next node (known as the relay node) in the chain, encrypted using the relay node's public key and relayed to it by the entry node, containing the circuit ID, a request from the entry node to the relay node to extend the circuit, and the half of a Diffie-Hellman handshake of the initiator. The relay node subsequently follows the similar mechanism for extending the circuit to the exit node. This generates a telescopic circuit between the initiator and the destination server. When the chain is complete, the originator can send data over the Internet anonymously.

CHAPTER 3

THE LILAC SYSTEM

In this chapter we provide an insight of the Lilac system design. Lilac comprises of four entities – a client, a directory server, a lilac server and the relay nodes. The lilac server is an event driven multi-threaded web server which extends the communication channels from a sender to the receiver distinguishing them with a flagged ID – a display name provided by the users. All the system notifications and connection-disconnection event messages on the client side are generated by the application server. The directory server keeps track of changes in the network topology and node state and provides a set of available relay nodes to the new users. The relay nodes are a set of proxy servers which relay the circuit between a user and the application server. Figure 3.1 illustrates bi directional circuits in this system.

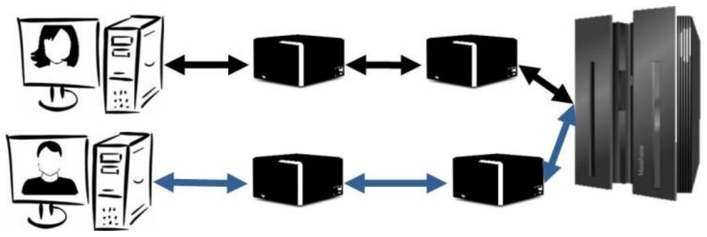


Figure 3.1 Lilac Circuits

A client first connects to the directory server to get the list of available relay nodes. From the given list of relay nodes, client randomly chooses a set of two nodes which will create a two hop circuits between the client and the lilac server. After getting the node information, the very first step is to build a circuit between the client and the first node. Then client initiates the circuit to be extended to the next hop, subsequently the circuit is extended to the lilac server. This creates a virtual bidirectional communication path between a client and the application server. Once a bidirectional circuit is built, a user can enter into the system by providing a display name. The lilac server gets the details of newly joined user – a display name and the public point, and checks the exclusivity of the display name. If the display name is exclusive within the chat room then the server responses with a set of public points and display names of entire online users. Now, a user can initiate communication with a desired online user.

3.1 Connecting the Directory Server

The user sends an HTTP GET request to the directory server to get into the system. The directory comprises a list of active nodes and the network status. Each user needs to create a bi directional path between the lilac server and the user itself. This path is created using a set of relay nodes. Each relay node carries a public point and the unique node ID. Directory server keeps track of all these nodes along with their respective distinct node information. The nodes are connected with the directory server through a bi directional TCP socket.

The client connects with the directory server from a web browser sending an HTTP request. This request contains a set of flagged information, which is molded into a packet. Section 3.1.1 describes this packet format. Once the directory server gets this packet, it generates a similar response packet which contains the details of entire online nodes to relay the circuit. Using this information, a user can initiate a two hop circuit to the lilac server.

3.1.1 The Packet Format

Each client and nodes exchanges a predefined set of packets containing the directive flagged values. Figure 3.2 illustrates a visualization of a packet format. Beginning from a client end, each message is end to end encrypted using a shared secret key between the receivers. A receiver can be a communicating partner or a relaying node in the circuit. Moreover in building the circuit each message is encrypted with layers of encryption using derived session keys of each participating node in the circuit. Along with the layers of encryption, these packets contain directive flag values for the server. A display name of a user is flagged and embedded along with the message such that the lilac server can redirect that message to the respective receiver end. Moreover each packet is flagged with message type bits on the top of a text message, zero is addressed for a fake message and a one is addresses an actual message. With this flagged values the lilac server can identify the incoming message and can take necessary action on it.

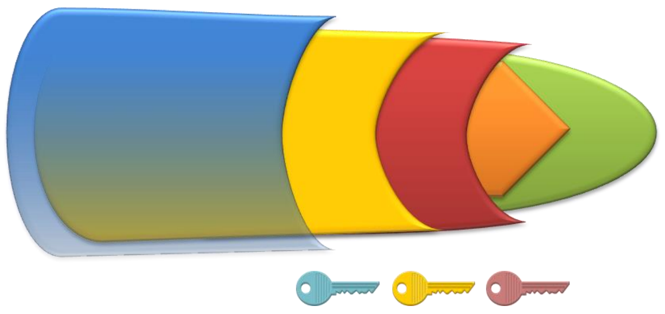


Figure 3.2 Packet Visualization

3.2 Lilac Server

Lilac system design requires a scalable and efficient application specific HTTP server which concurrently handles bidirectional and asynchronous I/O requests. In the traditional thread per request approach, a thread is forked for each I/O request. In case of constructing and extending several circuits simultaneously the conventional thread per request design serves each request one-by-one in flow. This means the web server remains idle until the circuit is built from the server to the extended nodes, which may create a thread block for the further incoming requests. To avoid this, additional threads can be added, which requires rich hardware resources. Moreover building circuits for each user and extending those to another end of communicating partners simultaneously would increase chances of thread block latency in the system. The primary objective is to construct a light weight real time communication platform, which reasonably avoids the network and the application level overheads from the system. We have deployed a light weight application specific HTTP web server which adopts event driven paradigm in terms of creating and utilizing the thread pool[7]. Thus a thread-pool is forked to serve the events instead of thread-per-I/O. This improves the performance of HTTP server as the thread block latency could be avoided with this approach. It handles each event with a thread, and when a thread-pool is full then only it forks another thread pool (or worker in web-server terminology) for subsequently scheduled event. We have used node.js[8] to create a scalable, efficient and light weight HTTP server.

When a new user tries to join in the system, she is asked to enter a unique display name within the current session. This display name is used to distinguish the users in the system. Each message is flagged with its sender and receiver display name, thus the application server can distinguish the messages using theses flagged IDs and delivers the messages to the respective receivers. A random secret key is generated for each newly joined user on the client side. Using the secret key and standard elliptic curve parameters, a public point is computed for each communicating user. Lilac server only receives this derived public point from a user. Thus a secret key is never exchanged in the communication channel. Once lilac server receives this public point and the display name from a user it makes sure exclusivity of the display name with current online users and if it is true then it responses with a set of public points of all the online users within the session. This prototype accords the lilac server to know only the directives of the message – who-is-to-whom. Once a user gets the public points from the lilac server, she can begin the chat over the communication channel. This communication is end-to-end encrypted between a sender and receiver, thus the lilac server does not know what is being exchanged between the users. It only addresses the incoming messages flagged with a sender and a receiver details and transmits it to the other end. Each user is connected with the server with a distinct circuit built on trusted relay nodes, thus lilac server identifies each client with its middle node. This is further discussed in building a circuit

[7] http://www.ibm.com/developerworks/java/library/j-jtp0730/index.html
[8] Node.js http://nodejs.org/

section. Lilac server generates a link between two distinct client circuits, by extending one end of a sender towards the receiver. This provides unlinkability for an eavesdropper between end-to-end communicating partners. Each online user asynchronously generates redundant fake messages, which look identical to an actual encrypted chat message, and is sent it to the server periodically like heartbeats when there is no communication is being done between users. This prototype requires a mechanism at the server side to distinguish an actual message with a fake heartbeat message, since they both are flagged with a similar user and only sever can distinguish it with the flagged value.

3.3 Building a Two Hop Circuit

A circuit is constructed for each newly joined user to the server end. A set of HTTP requests from an initiator are sent asynchronously to the directory server to retrieve available relay nodes. A response from the directory server carries two randomly selected relay nodes N^1 and N^2. These chosen nodes are ordered to provide a circuit through which the messages would be transmitted. The lilac circuits, unlikely to the Tor circuits, require only two randomly chosen relay nodes (entry node N^1, middle node N^2) between a client and the application server. The application server receives a request from a relay node and returns the response to the same node, thus each connected user in the system is addressed by the corresponding relay node adjoining to the server.

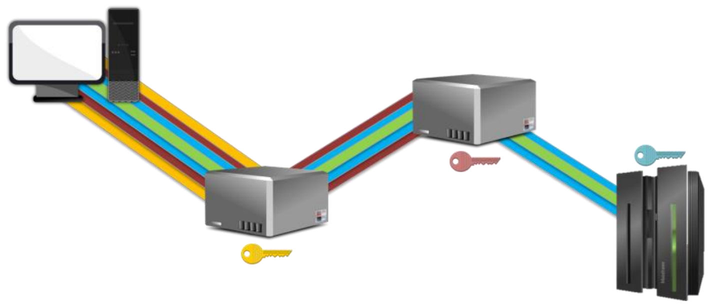


Figure 3.3 Layers of Encryption In the Circuit

Each relay node carries a unique public point. The very first step in constructing the circuit is to exchange the public points between the initiator and the entry node. Both entities – an initiator and the relay node randomly generates a secret key. Using the respective secret key and the exchanged public point, a node-to-node shared secret key is derived between the initiator and the entry node. Within the session an initiator Alice randomly generates a secret key A_K= (a) and the entry node N^1 generates $N^1{}_K$= (p). Subsequently the public point of Alice $A_{pub} = (ag^{xy})$ and of the node $N^1{}_{pub} = (pg^{xy})$ are exchanged with each other. Using these public points and respective secret keys, a Diffie-Hellman shared secret is computed. Once the node-to-node key is established a flag is sent back to the initiator to build the circuit. A session-key S_{AN^1} between the initiator and the entry node is then derived.

$$R_{AN^1} = \left(A_K N^1{}_{pub}\right) = (a(pg^{xy}) \qquad (1)$$

$$S_{AN^1} = \left(A_K N^1{}_{pub}\right) = (a(pg^{xy})) \qquad (2)$$

Now using this key R_{AN^1} a request from an initiator to the entry node is sent to extend the circuit to the middle node N^2. Subsequently the entry node N^1 and successor node N^2 follows the similar key exchange mechanism to derive a shared secret key $R_{N^1N^2}$ between these two nodes. Then a subsequent session key S_{AN^2} between the initiator and the extended successor node is derived.

$$R_{N^1N^2} = \left(N^1{}_K N^2{}_{pub}\right) = (p(qg^{xy})) \qquad (3)$$

$$S_{AN^2} = \left(A_K N^2{}_{pub}\right) = (a(qg^{xy})) \qquad (4)$$

Once a circuit is extended using all the relay nodes provided by the directory server, then it is extended and linked to the lilac application server. Lilac server generates a random secret key $Z_K = (r)$ for each incoming user add request. Thus a Diffie-Hellman shared secret is derived between the middle node N^2 and the Lilac server Z_K, and the circuit is completed between the initiator and the lilac server.

$$R_{N^1Z} = \left({N^2}_K Z_{pub}\right) = (q(rg^{xy})) \qquad (5)$$

$$S_{AN^2} = \left(A_K {N^2}_{pub}\right) = (a(qg^{xy})) \qquad (6)$$

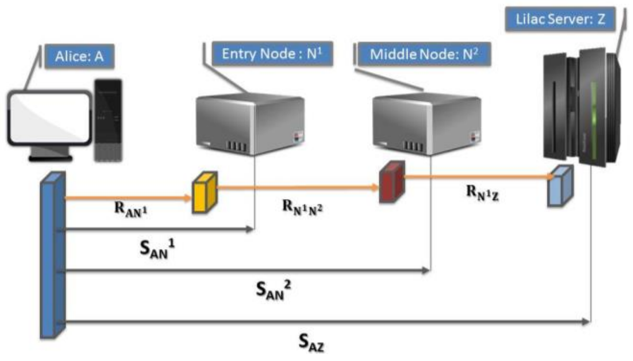


Figure 3.4 Session Keys and Node-to-node Keys

Once the circuit is complete between the initiator and the lilac server, a loopback path is generated using the reverse key exchange mechanism. Thus this virtual path constructs a bidirectional circuit between a user and the lilac server.

3.4 Key Exchange And End to End Message Encryption

A user is prompted to get started in the application by providing the display name from the input box. It follows typical internet relay chat[9] (IRC) pseudonymity. The display name represents the distinct user entity in the current session. In the experimental demonstration of this system, the password or the private key is not asked from the user; rather a 32-bit hash salt is computed on the given username. This hash is used as private key (a) for the Diffie–Hellman computation. The public point of sender Alice (A = aG) (X, Y) is derived using the private key and the Elliptic Curve parameters. Similarly for the recipient Bob the public point (B = bG) (X,Y) is computed. Now this public point is broadcasted to all the online users, in this case Alice and Bob mutually exchanges their public points with each other. Now Alice computes the shared secret key (S = aB = abG) (X,Y) using her private key and Bob's public point, in similar way Bob derives the shared secret key (S = bA = baG) (X,Y). So for computing one's shared secret key, one must need the private key to compute it from the public point. The private keys of users are never exchanged over the network, thus this approach makes the key exchange mechanism robust in terms of concealing and not sending the private key along with the cipher text. From this point, now a user can send and receive the messages. Each message is encrypted using a simple RSA encryption algorithm. Thus the message is always exchanged in cipher text, so an eavesdropper cannot know what is being shared amongst the users unless one finds out the private key of a user. Figure 3.2 illustrates the light weight key exchange.

[9] http://en.wikipedia.org/wiki/Internet_Relay_Chat

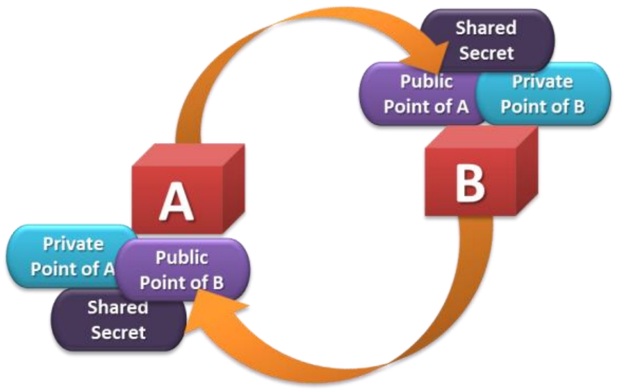


Figure 3.5 Light Weight Key Exchange

This key exchange approach follows an ephemeral key generation paradigm; hence a private key or the password of a user is never saved in the system. Each time a user has to provide a new private key for the newly created client session and each time the shared secret key computation is done when a user broadcasts or unicasts a message. So in a group chat of five users, if Alice wants to broadcast a message, the shared secret key computation for all four other users has to be done simultaneously and with the derived four shared secret key the message is encrypted and sent over the network. By testing this application with different numbers of users in a group chat, we found that the latency added in the message is trivial until the number of user is five in the chat room, but then it increases linearly with the number of users added in the system. Thus for the better performance and to achieve nearly zero latency in the message exchange, we have limited the number of users in group communication up to five. For one-to-one communication, the performance of this system is expectedly well; there is nearly zero computational latency in communication of five users exchanging messages simultaneously. The result section demonstrates the numerical enlightenments.

Each message is encrypted using RSA 6b bit encryption algorithm. The shared secret key is used to encrypt the message. From the derived ECDH shared secret key of Alice (S = aB = abG) (X, Y) and of Bob (S = bA = baG) (X, Y) the ‘X’ coordinate value is used as the shared secret key for the encryption of the message.

We have used the recommended Elliptic Curve Domain Parameters of secp160r1 based on Standards for Efficient Cryptography (SEC) [22]. These parameters are specified by the sextuple T = (Q; A; B; G; N). The first stage of the key exchange mechanism is to set uniform elliptic curve domain parameters. Since ECDH allows for elliptic curves of arbitrary sizes and thus arbitrary security strength, it is important that the size of elliptic curve be chosen to match the security strength of other elements of the symmetric encryption of the messages. These parameters define a finite field GF(p), a field of integers modulo a large prime p, an elliptic curve over that finite field, a point on that curve, and the order and cofactor of the subgroup generated by scalar-point multiplication at that point. These Elliptic Curve parameters are initialized at the page load event on client side browser each time a client loads the application. The mouse movement entropy is used for generating the stronger and distinct seed value, for the random number generators (RNG), used in the crypto algorithm parameters. This conceivably stronger entropy ensures that an attacker could not reproduce these parameters. The second stage is to compute the public point of a user and that is to be exchanged between users. A public point is derived by computing the private key with an ephemeral public key coordinates and the Elliptic Curve Domain Parameters. Then this public point along with the user display name is exchanged with the communicating partner. This public point is also broadcasted to the entire online users for group communication. Every time a new session is created, this public point is computed and exchanged amongst the online users. Now, a user has exchanged the public point, the shared secret key is generated on the client side using the public point of communicating partner with the private key of a user itself. This shared secret key is used for symmetric encryption of each message sent by a user. On the other hand, the communicating partner receives the encrypted message and decrypts it with the shared secret key she has derived. This archetype avoids sharing the private key along with the message and makes it pragmatically very difficult to decrypt the message by an intruder, as an intruder must need a private key any of the communicating partners, which is never exchanged and generated only on the client side. We have used “jsbn[10]” crypto library; hence it is deployed in pure JavaScript, it does not require any other dependencies. It is fast and portable crypto implementation using JavaScript.

3.5 Heartbeats In the Circuits

Each user entity in idle state periodically generates a random cipher text and asynchronously broadcasts it to the entire online users through extended circuits and on the loopback circuit to her too. This random noise is generated only when a user is in idle state – not communicating. In the communication channel the actual messages are synchronized with the random noise. From the deployment point of view, the distinction of idle state and available state is derived through mouse pointer movement and key press event of the input box. When both of these actions are not being performed on the user window, the user state is then set to idle. The user state is derived by periodically running a loop which detects both the factors – pointer movement and key press event in the input box.

Consider a stack of five blocks. Each user generated message and an asynchronous fake message first get pushed into the stack. A cyclic event checks periodically whether there is any available entry on the top of the stack, if it is not null then the item is popped out and sent over the network as HTTP GET Request. Both a fake and a legitimate message are flagged during the stack operation. A legitimate message is always prioritized and pushed on the top of

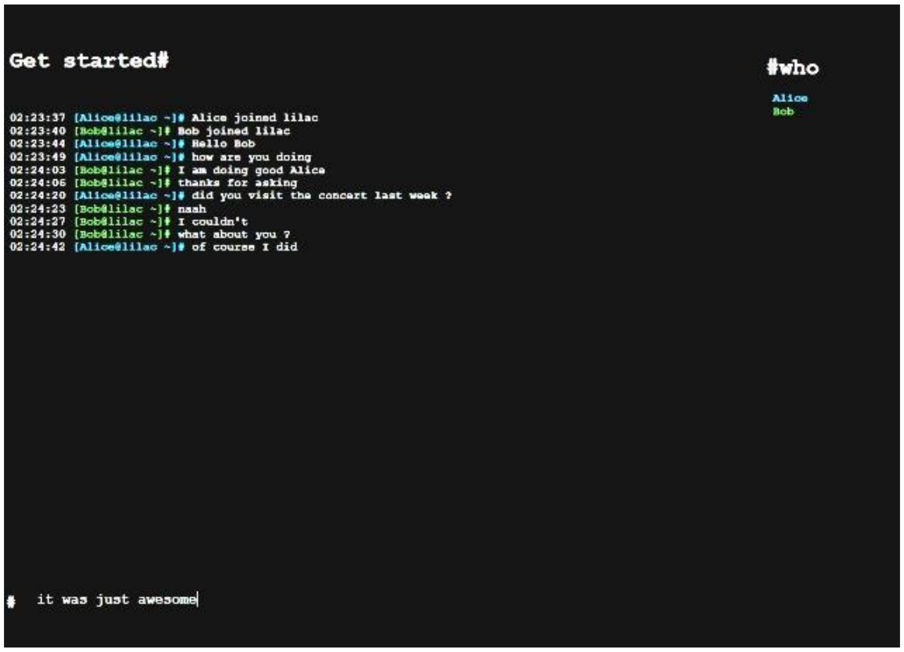
[10] http://www-cs-students.stanford.edu/~tjw/jsbn/

the stack; hence there is no added latency due to the stack operation in sending a message. When fake messages are pushed into the stack and then if a legitimate message is pushed; the entire fake messages drops out simultaneously and the actual message is pushed on the top of the stack to be popped out. This flow of always prioritizing the legitimate message pushing over a fake message prevents any executional delay in the system. This whole flow of stacking the messages and sending them periodically generates a heartbeat like prototype in the system. Due to this paradigm for an eavesdropper, a user is always online and communicating with others. This provides a reasonable unobservability in the system.

3.6 User Interface

The chat user interface is simple and straight forward to use. A user only requires providing a display name to enter into the system and subsequently she gets a list of current online users within the scope. A scope is derived as a chat room based on IRC norms; hence a user is bound to communicate within a current chat room only. The chat module comprises of three parts – the chat display, the multipurpose input area and the list of online users within the chat room. The chat display outputs the messages from both the entities – user and lilac server. The standard format of display window is similar to a UNIX like terminal. It shows timestamp of each event, the username, current chat room display name and the exchanged messages. The multipurpose input serves four distinct filters over the textual inputs: entering username, entering an individual message starting with ‘at sign’ (@username), a broadcast message, and the system commands starting with the ‘dollar sign’ ($connect). At this moment the system commands are enabled only in the debugging mode. The third part is the list of online users on the right side panel. This list shows the active users in the current chat room. So from this list, a user can send a message to a specific recipient, prefixing it with an ‘at sign’ following the username. The very first input is taken as the username, and all other inputs except special prefix characters broadcasted within the chat room.

This deployment is aimed to demonstrate the primary functionality and execution of core functions of the design; hence we are not overly concerned with profiling a user and establishing a user-to-user relationship distinction. Instead for an experiment purpose, we have implemented an analogous way of communication through a full-duplex two-way radio communication prototype, for authorizing the communicating partner in the system, in which a transceiver allows the operator to have a conversation with other similar radios operating on the same radio frequency channel. A sender exchanges a shared secret key with a receiver which is analogous to setting similar frequency between two communicating transceivers, and any other user having a the similar shared secret key can receive and transmit a message within a specific communication channel. As long as the session is over, the communication channel is asynchronously destroyed. Although this provides hands on ease-of-access to demonstrate the core functionality of the system, it is not intended to be the primary mechanism of authenticating the users. Figure 3.6 illustrates a sample user interaction between Alice and bob.



```
Get started#                                                    #who

                                                                Alice
                                                                Bob
02:23:37 [Alice@lilac ~]# Alice joined lilac
02:23:40 [Bob@lilac ~]# Bob joined lilac
02:23:44 [Alice@lilac ~]# Hello Bob
02:23:49 [Alice@lilac ~]# how are you doing
02:24:03 [Bob@lilac ~]# I am doing good Alice
02:24:06 [Bob@lilac ~]# thanks for asking
02:24:20 [Alice@lilac ~]# did you visit the concert last week ?
02:24:23 [Bob@lilac ~]# naah
02:24:27 [Bob@lilac ~]# I couldn't
02:24:30 [Bob@lilac ~]# what about you ?
02:24:42 [Alice@lilac ~]# of course I did

#  it was just awesome
```

Figure 3.6 User Interface

CHAPTER 4

EXPERIMENTS AND RESULTS

To demonstrate the efficacy of our proposed improvements, we design and execute a set of experiments. These experiments are performed using a real world internet deployment setup and used Webserver Stress Tool[11]. Webserver Stress Tool is a geographically distributed platform for evaluating and accessing planetary-scale services simulating the HTTP requests generated by up to 10.000 simultaneous requests and serves as a test bed for computer networking and distributed systems.

Our experimental setup is comprised of an application server, directory server and two relay nodes on real world internet which builds the lilac architecture for the users. We have deployed geographically distributed set of client nodes on the PlanetLab to exhibit the performance and stress measurements on both the server and client entities.

4.1 Implementation Analysis

Our experimental setup comprised of 10 geographically distributed client nodes, a directory server, lilac application server and two relay nodes. Since Lilac is aimed for low-latency communication, the evaluations that we do focus on the performance metrics that is particularly important to the quality of service of an end-user from that perspective. We measure the time-to-first-byte (TTFB) which is the time a user must wait from the time they issue a request for data until they receive the first byte (duration from the web client making a HTTP request to the first byte of the page being received by the browser). The time-to-first-byte is two end-to-end circuit RTTs: one RTT to connect to the lilac HTTP server, and a second RTT to issue a request for data (e.g., HTTP GET) and receive the first byte of data in response. We also measure the download time which is how long a user must wait for a web page to load. To evaluate the time-to-first-byte, we measure the latency between the 10 geographically

[11] http://www.paessler.com/webstress

distributed clients constructing the loop back circuits simultaneously. We measured the latency between each pair of communicating partners to receive the first message on both ends.

We evaluate the Round Trip Time (RTT) taken for constructing a loopback circuit from the client side. We gathered these RTTs concurrently for 3 days, and derived a median page load time on the client side.

We have measured the latency in our system comparing the Lilac communication with and without circuit construction. We have setup the lilac experimental deployment without circuits on URL - isec.uta.edu:8060 and with extended circuits on URL - ankit-pc.uta.edu:9000.

Table 4.1 Lilac with and without circuit construction

URL#	URL
1	isec.uta.edu:8060
2	ankit-pc.uta.edu:9000

Table 4.2 Average time taken for each setup

URL No.	Name	Clicks	Errors	Errors [%]	Time Spent [ms]	Avg. Click Time [ms]
1		100	0	0.00	461,578	4,616
2		121	0	0.00	262,164	2,167

First we measured the time taken for the first byte on lilac version without the circuit construction, and then the similar measurement is derived after the circuit construction. Table 4.2 shows average time taken in each case.

Table 4.3 System configuration in case of without constructing the circuit

Project and Scenario – Without Circuit	
Test Setup	
Test Type:	TIME (run test for 5 minutes)
User Simulation:	10 simultaneous users - 35 seconds between clicks (Random)
Logging Period:	Log every 5 seconds
Browser Settings	
Browser	User Agent: Mozilla/5.0 (compatible; Webserver Stress Tool 7; Windows)
Browser Simulation:	HTTP Request Timeout: 120 s
Options	
Advanced Settings:	Save all HTML files to "[user][click][request].htm"
Local IPs: Automatic	URL#1: GET isec.uta.edu:8060 POSTDATA= Click Delay=
Client System	
System	Windows 7/2008 R2 V6.1 (Build 7601) Service Pack 1, CPU Proc. Lev. 686 (Rev. 5898) at 2992 MHz,
Memory	572 MB available RAM of 3487 MB total physical RAM, 2989 MB available page file

Table 4.4 System configuration in case of constructing the circuit

Project and Scenario – With Circuit	
Test Setup	
Test Type:	TIME (run test for 5 minutes)
User Simulation:	10 simultaneous users - 10 seconds between clicks
Logging Period:	Log every 5 seconds
Browser Settings	
Browser Simulation:	User Agent: Mozilla/5.0 (compatible; Webserver Stress Tool; Windows)
Browser Simulation:	HTTP Request Timeout: 120 s
Browser Simulation:	Simulate Maximum Data Rate: 128 kbit/s
Local IPs: Automatic	URL#2: GET ankit-pc.uta.edu:9000 POSTDATA= Click Delay=
Client System	
System	Windows 7/2008 R2 V6.1 (Build 7601) Service Pack 1, CPU Proc. Lev. 686 (Rev. 5898) at 2992 MHz,
Memory	586 MB available RAM of 3487 MB total physical RAM, 2857 MB available page file

4.2 Results

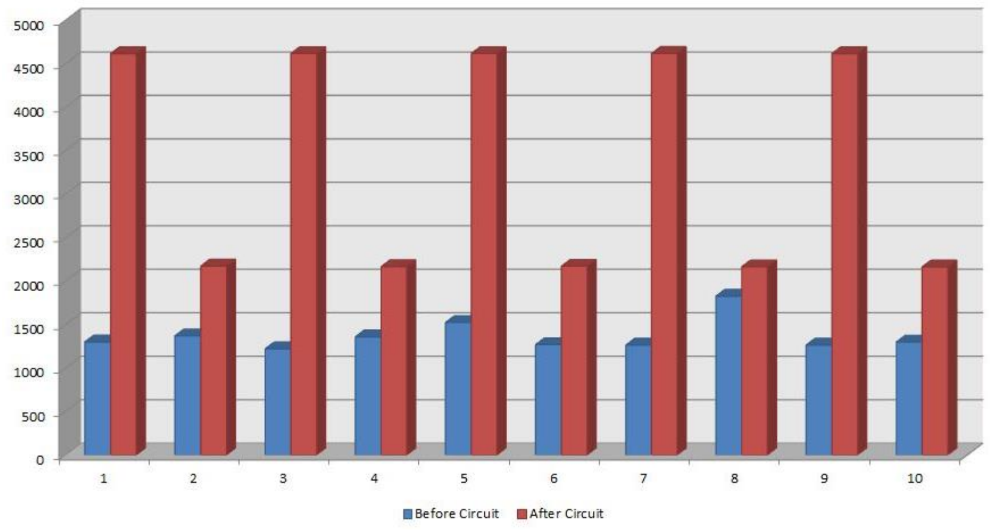


Figure 4.1 Latency Measurement

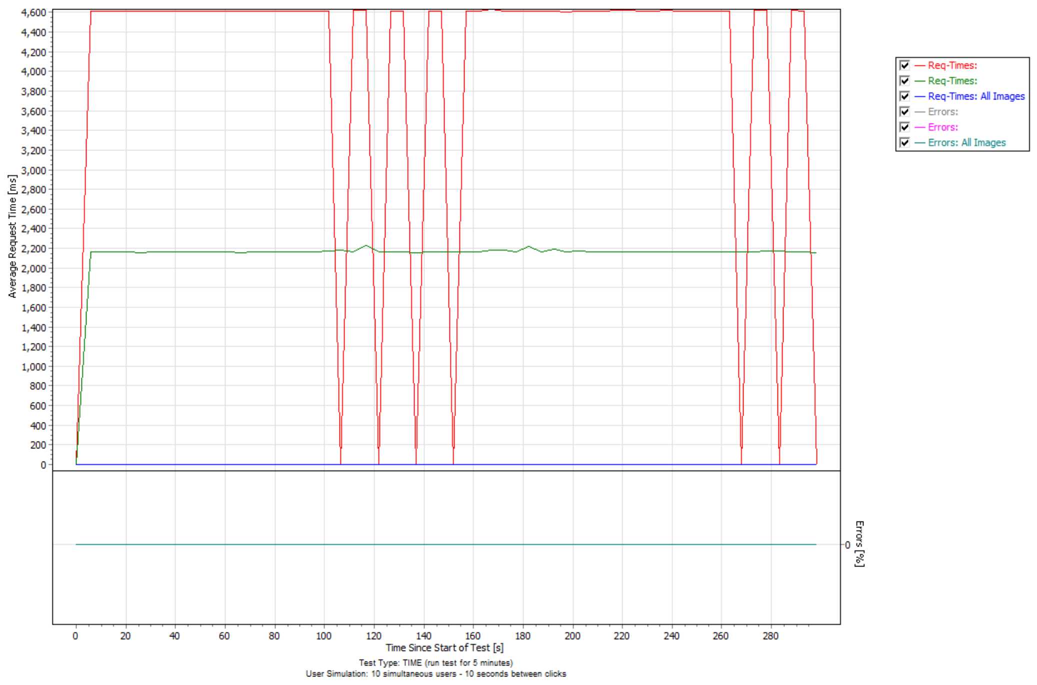


Figure 4.2 RTT Comparison

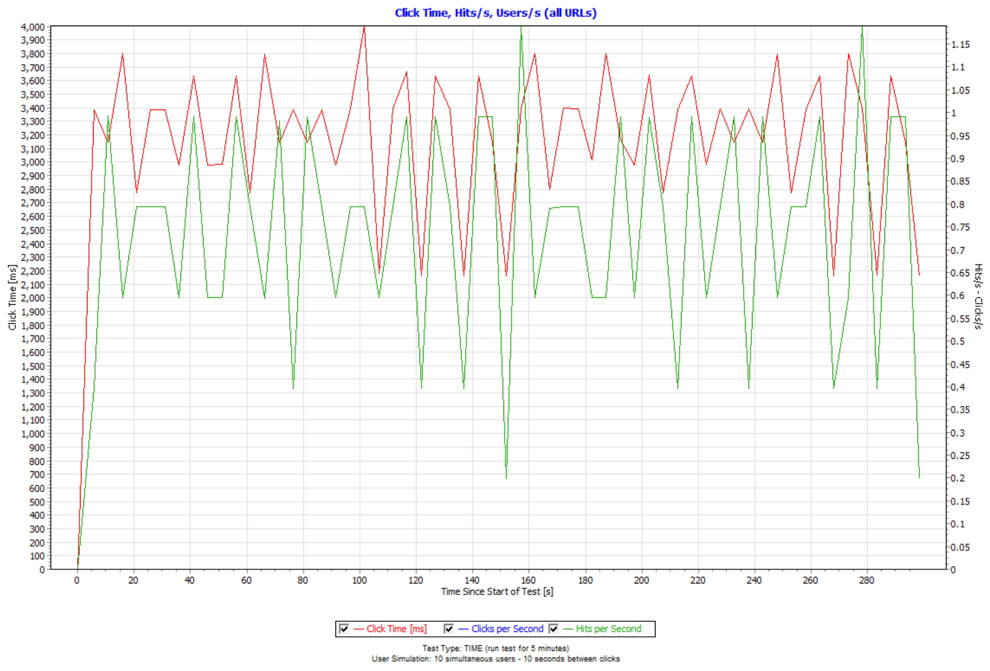


Figure 4.3 Bandwidth Throughput Comparisons

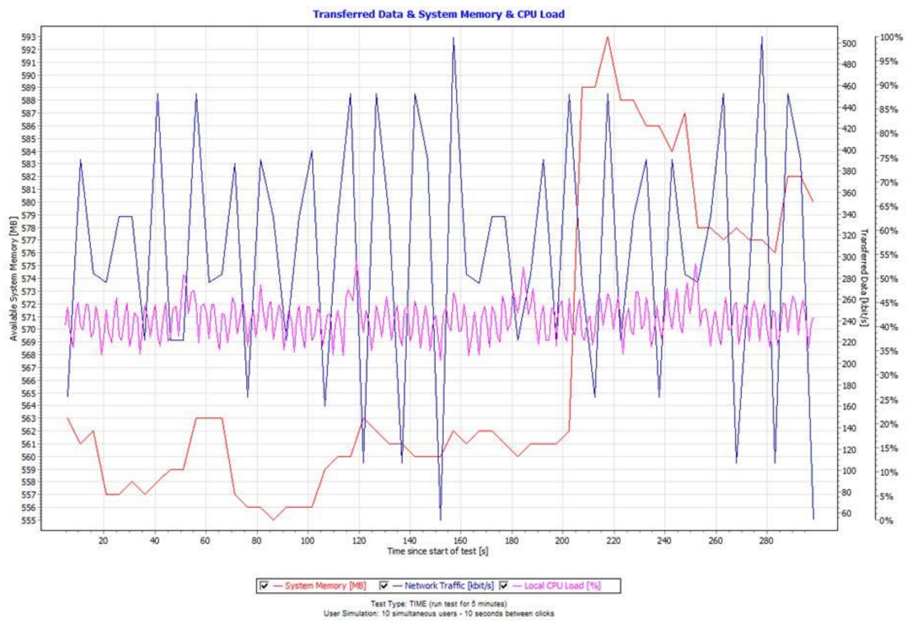


Figure 4.4 Time to First Byte

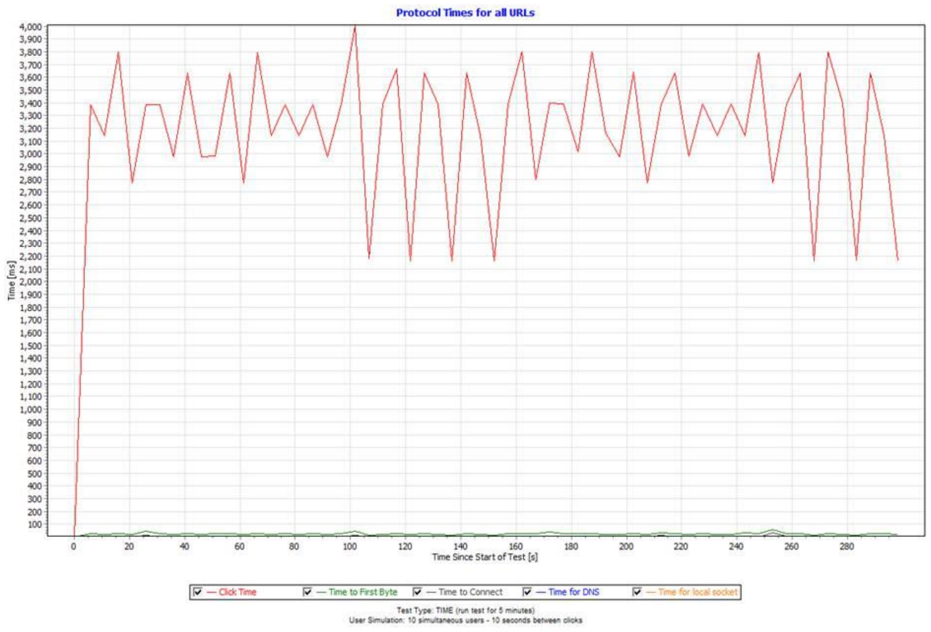


Figure 4.5 HTTP Protocol Time

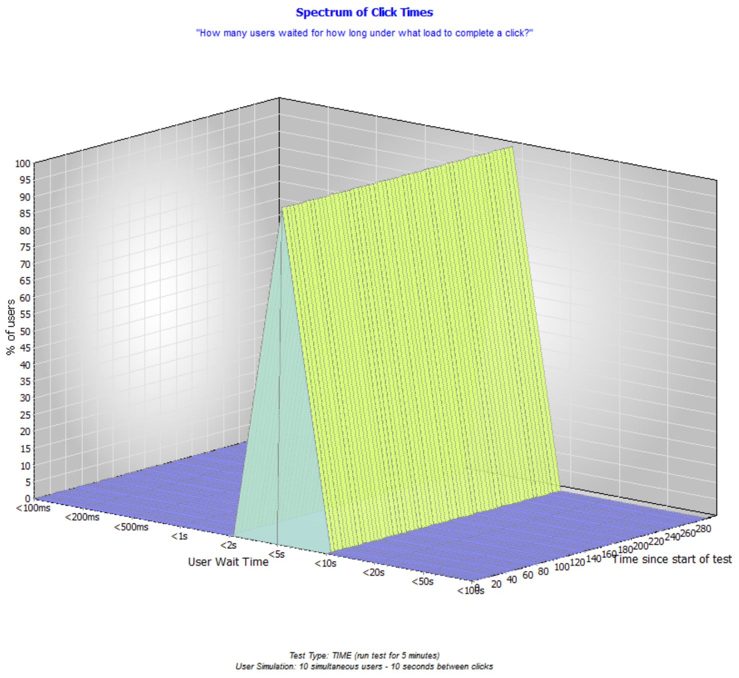


Figure 4.6 Time to First Loopback Request

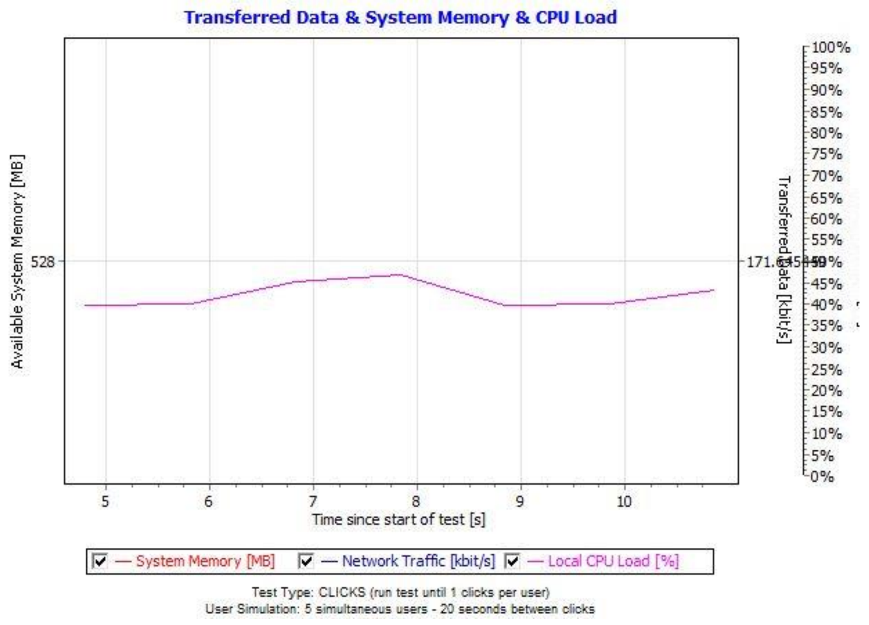


Figure 4.7 all ten nodes simultaneously connected to the server

We first evaluate the time take for constructing a circuit between a newly joined user and the lilac server itself. It can be perceived from the figure 4.1 and 4.2, while creating a circuit for newly joined user it adds a negligible amount of latency in the HTTP response compare to the circuit less system. Once the circuit is established and both the communicating partners begin the communication, it can be seen from figure 4.3 the time attribute remains in the similar region, thus it does not add any additional latency while the users are communicating.

Then we evaluated the time-to-first byte from accessing the 10 clients. We compared the performance of normal circuit less system with the extended path communication. Fig 4.1 shows the average of the time-to-first-byte for both the systems.

Figure 4.5, 4.6 and 4.7 illustrates the load balancing on the server side, with generating 4000 requests from each client concurrently. Even with this forty thousand plus simultaneous requests on the server, it remains in a stable state and does not breakdown during this process.

CHAPTER 5

DISCUSSION

In our work we demonstrated a system which is designed to provide a platform for anonymous instant messaging. The deployed system is a browser based application, which eliminates the client side system changes and network configuration to run the system. In this section we compare our work with the existing anonymous chat application [22] which uses the Tor based hidden services as its underlying network. TorChat is decentralized anonymous application which comes in a portable version too such that storing it on a flash drive, it can be used without any installation, configuration and account creation. We have proposed a similar approach for our system, such that it can be used straight from the browser without configuring any perquisites. Over the time TorChat has been redesigned and constructed using different technologies. The older windows versions of TorChat were built with py2exe (since 0.9.9.292 replaced with pyinstaller); a rewrite of the TorChat protocol was created in the beginning of 2012, called jTorChat[12] on Google Code[13]. As of February 5th, 2013, developer Prof7bit[14] moved TorChat to GitHub[15]. The major downside of this existing system from a user prospective is not providing a stable version which ensures the anonymity. Moreover the portable version comes bundled with a copy of the Tor onion router readily configured, which means it is completely relied on pre-configured set of relay nodes which does not ensure a complete circuit randomization.

Our system demonstrates a novel approach towards anonymous light weight communication architecture which is dedicatedly designed for instant messaging, thus it does not rely on any other existing anonymous architecture. On the other hand, TorChat is overlaid on Tor hidden services and using the existing Tor relay nodes, statistically it entails more

[12] https://github.com/jtorchat/jtorchat
[13] https://code.google.com/
[14] https://github.com/prof7bit
[15] https://github.com/prof7bit/TorChat

compound relaying chains. Thus expanding the complexity of relay nodes is required to adopt the stronger entropy in the circuit generation for Lilac.

CHAPTER 6

CONCLUSION AND FUTURE WORK

This work aims to provide a simple, efficient and dedicated anonymous architecture for instant messaging. We have deployed an experimental browser based application to demonstrate the primary features. The implementation analysis and results show a reasonable tradeoff amongst the performance, efficiency and usability of the system. However prior to deploying this system for public domain, it needs to be tested and stressed over a stronger set of threat models and test cases.

6.1 Conclusion

We proposed a design for light weight anonymous communication architecture, and deployed a platform to communicate on the real world internet. This system comprises a development of application specific HTTP server, a simple and easy to deploy mechanism of circuit building and a simple cross browser compatible user interface for the instant messaging communication. The deployed system does not require any client side configuration, installation and account creation to use this application. However in the future work we have proposed to overlay a mechanism for authenticating the users by registering into the system. To the end, we provided the implementation analysis and performance results comparing with an existing deployed system.

6.2 Future Work

Our proposal creates a simple system for light weight anonymous communication which can be used straight from the browser. As a result it ensures the desired primary goals of deploying a system which is easy to use and comparably delivers a low latency system for short message exchanges. However this system needs to be tested and stressed over a stronger set of threat models and test cases such as global adversary attacks, security measurement against the timing analysis attacks and sampling the network traffic.

Another important area of interest is the static authentication mechanism, which could provide a simple and efficient way of registering the communicating users. A browser based plugin can be developed to provide this static authentication mechanism, since our primary goal is not to leave any user correspondences on the server entities, a browser based plugin would assimilate the consistent communicating partners identity for a period of time on each respective users. We also would like to adapt the distributed server entities in the design, such that a single point of failure can be addressed efficiently.

REFERENCES

[1] D. Chaum, “Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms,” in Communications of the ACM, 1981.

[2] R. Dingledine, N. Mathewson, and P. Syverson, “Tor: The second generation onion router,” in Proceedings of the 13th USENIX Security Symposium, 2004.

[3] George Danezis, Claudia Diaz, Carmela Troncoso, and Ben Laurie, “Drac: An Architecture for Anonymous Low-Volume Communications”, PETS'10.

[4] Steven J. Murdoch and George Danezis, “Low-Cost Traffic Analysis of Tor”, Proceedings of the 2005 IEEE Symposium on Security and Privacy.

[5] Damon McCoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker, “Shining Light in Dark Places: Understanding the Tor Network”, PETS 2008.

[6] Roger Dingledine and Nick Mathewson, “Anonymity Loves Company: Usability and the Network Effect”, In Proceedings of the Fifth Workshop on the Economics of Information Security.

[7] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In 2003 IEEE Symposium on Security and Privacy, 2003

[8] U. M¨oller, L. Cottrell, P. Palfrader, and L. Sassaman. Mixmaster Protocol 2003.

[9] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. IEEE Journal on Selected Areas in Communications, 1998.

[10] Steven J. Murdoch and Piotr Zielinski. Sampled traffic analysis by internet exchange-level adversaries. In Borisov and Golle.

[11] Steven J. Murdoch and George Danezis. Low-cost traffic analysis of tor. In IEEE Symposium on Security and Privacy, IEEE Computer Society, 2005

[12] Nick Feamster and Roger Dingledine. Location diversity in anonymity networks. In Vijay Atluri, Paul F. Syverson, and Sabrina De Capitani di Vimercati, editors, WPES, ACM, 2004.

[13] A. P_tzmann, B. P_tzmann, and M. Waidner. ISDN-MIXes: Untraceable Communication with Small Bandwidth Overhead. Informatik-Fachberichte, 1991.

[14] O. Berthold, H. Federrath, and S. K¨opsell. Web MIXes: A system for anonymous and unobservable Internet access.

[15] R. Bohme, G. Danezis, C. Diaz, S. Kopsell, and A. Pfitzmann. “Mix cascades vs. peer-to-peer: Is one concept superior?” In *Privacy Enhancing Technologies (PET 2004)*, Toronto, Canada, 2004

[16] Charles V. Wright, Lucas Ballard, Scott E. Coull, Fabian Monrose, and Gerald M. Masson. Spot me if you can: Uncovering spoken phrases in encrypted voip conversations.

[17] Philipp Bachmann, “Deferred cancellation: a behavioral pattern”, Proceedings of the 15th Conference on Pattern Languages of Programs 2008

[18] A. Pfitzmann and M. Waidner, “Networks without user observability - design options,” Computers and Security, 1987.

[19] Michael K. Reiter and Aviel D., “RubinCrowds: Anonymity for Web Transactions”, ACM Transactions on Information and System Security (TISSEC) 1998.

[20] Matthew K. Wright , Micah Adler , Brian Neil Levine , Clay Shields, “The Predecessor Attack: An Analysis of a Threat to Anonymous Communications Systems”, 2004.

[21] STANDARDS FOR EFFICIENT CRYPTOGRAPHY, “SEC 2: Recommended Elliptic Curve Domain Parameters” Certicom Research, 2000

[22] TorChat: Available at https://github.com/prof7bit/TorChat

BIOGRAPHICAL INFORMATION

Ankit Upadhyaya was born in Vadodara, India in 1989. He received his Bachelor's degree from the Maharaja Sayajirao University of Baroda, Vadodara India in 2011. He has been a part of iSec, the Information Security Lab, from 2012. From August 2013, he will join LinkedIn Corporation as a Site Reliability Network Operations Engineer.