Anonymous Peer-to-Peer Publish Subscribe System

A thesis submitted in the partial fulfilment for the requirements of the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Saumyajit Dey
Roll No : 137250
Sushil Paneru
Roll No : 137258
Aman Alam
Roll No : 137203

Under the Guidance of

Dr. R. Padmavathy
Assistant Professor, CSE Dept.
NIT Warangal

Department of Computer Science and Engineering

National Institute of Technology

Warangal

2017

Dissertation Approval for B.Tech.

This Project Work entitled

Anonymous Peer-to-Peer Publish Subscribe System

by

Saumyajit Dey

Sushil Paneru

Aman Alam

is approved for the Degree of

B. Tech in Computer Science and Engineering

Examiners

.....

.....

.....

Supervisor(s)

.....

.....

Chairman

.....

Date: .....

Place: .....

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY

WARANGAL

CERTIFICATE

This is to certify that the project titled **"Anonymous Peer-to-Peer Publish Subscribe System"** is a bonafide work carried out by **Saumyajit Dey, Sushil Paneru & Aman Alam** in partial fulfilment of the requirements for the award of the degree of **B.Tech.** in **Computer Science and Engineering** and submitted to the Department of Computer Science and Engineering, National Institute of Technology, Warangal.

Dr. R. Padmavathy
Project Guide
Department of CSE
NIT Warangal

Dr. Ch. Sudhakar
Head of the Department
Department of CSE
NIT Warangal

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, We have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea / data / fact / source in my submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

.....	.....	.....
Saumyajit Dey	**Sushil Paneru**	**Aman Alam**
137250	137258	137203

Date: .....

Place: .....

Abstract

This project aims to present an anonymous message exchange system which uses peer to peer publish-subscribe pattern. Publish-Subscribe systems allow distribution of data based on subscription rather than specific addressing i.e a message is addressed by a subject string rather than to a specific recipient. Publishsubscribe is a sibling of the message queue paradigm, and is typically one part of a larger message-oriented middleware system. This kind of messaging paradigm adds flexibility in which senders and recipients are decoupled and can operate without even knowing of each others existence.Without anonymity, it is possible to track down publishers and subscribers for a given subject. However, if anonymity is introduced into this pattern, there could be a wide range of applications in the field of file sharing or simple text messaging.

A high level example is an anonymous communication system, wherein users can contact each other and exchange data while protecting their identities from each other and from outside parties. Such a model would further allow governments to take part in covert operations or news agencies to obtain sensitive and ground breaking information secretly from a source.

The project aim to design an anonymous P2P system that covers the identity of the sender(s) and receiver(s) and brings optimizations on performance of other publish subscribe P2P systems.

Keywords: Publish-Subscribe, P2P Systems, Anonymity.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Aim

The project aims to develop an anonymous message exchange system in a Peer-to-Peer network that uses the popular Publish-Subscribe design pattern.The idea is

- To use the Publish Subscribe Design pattern in a peer to peer network.
- To introduce a certain degree of anonymity into the network.
- To maximize the use of P2P network to distribute the load of publishing messages from publisher to other nodes in the network.

Classic message exchange system based on publish subscribe pattern are all about the topic of the message rather than the identity of the sender or receiver. This project aims to assure the users that their

identities are hidden.

In order to solve this problem, the openness of the system in terms of who can join the network and how needs to be discussed. Two kinds of systems need to be defined: **open** and **closed**. They are discussed in later sections.

1.2 Anonymity in our context

The main crux of this work is introducing anonymity. To start, anonymity in our context should be defined. In this project, anonymity refers to the following three situations:

- An eavesdropper (i.e a middle man) should not be able to know about the identity of the peers and their role in the network (i.e whether they are a publisher or a subscriber).
- A forwarder peer (a peer who helps in message dissemination) should not be able to know about the content of the messages that it is forwarding to other peers.
- A publisher should not know the identity of subscribers. Similarly, subscribers should not know about the identity of the publisher.

1.3 Open and Closed Systems

Open and closed systems and the definition of anonymity were mentioned in the previous section. They will be discussed in some more detail through a couple of examples

- For an **open system**, anyone can join the network and a person can read any other person's messages. An example of this would be a civil rights activist wanting to communicate to the masses in order to protest about Civil Rights Violation during a state of emergency (e.g Arab uprising in Lebanon, Egypt, Syria, etc.), but is afraid that his/her identity will get leaked. This system will have **message authentication** but no **message confidentiality** as it is an open forum and everybody can read the message.
- On the other hand in a **closed system**, only those who have already obtained a pass phrase by some means can join the network and communicate. For example, a source wants to leak sensitive information to a group of newspapers but all their identities must be kept secret during this

exchange of information. Here a closed network is going to be used. This system will have both **message authentication** and **message confidentiality**

1.4 Things to know

Here are some of the things to know in order to tackle this problem statement

1.4.1 Peer-to-Peer Networks

Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or work loads between peers. Peers are equally privileged, equipotent participants in the application. They are said to form a peer-to-peer network of nodes. Peers are both suppliers and consumers of various resources such as network bandwidth, storage and memory, unlike in a centralized client-server architecture in which the consumers and suppliers of resources is divided. While P2P systems had previously been used in many application domains the architecture was popularized by the file sharing system Napster, originally released in 1999

1.4.2 P2P vs Centralized System for Publish Subscribe

Publish/subscribe systems are becoming very popular for building large scale distributed systems /applications. A publish/subscribe infrastructure is responsible for matching events to related subscriptions and delivering the matching events to interested consumers. Building a centralized publish/subscribe system has the advantage of having a global image of the system and thus making the matching algorithm much easier to implement. This approach suffers from scalability problems as the number of publications and subscriptions increases. Thus, the decentralized approach is more appropriate. The peer-to-peer (P2P) paradigm is appropriate for building large-scale distributed systems/applications. P2P systems are completely decentralized, scalable, and self-organizing.

1.4.3 RSA Public Key Cryptosystem

RSA[1] is a popular public key cryptosystem. In RSA, data is encrypted with a public key and decrypted using a private key. The private key is kept secret and public key is generally available to the public. The key pair has a mathematical relationship so that if data is encrypted with the public key then it can be decrypted with the private key and vice versa. RSA requires 100 to 1000 times more computation

than symmetric cryptosystems, hence it is not suitable for bulk encryption and decryption. Instead, RSA is widely used to distribute keys for symmetric cryptosystems.

1.4.4 AES Symmetric Key Encryption

Advanced Encryption Standard (AES)[2] is the most widely used Symmetric Key Encryption Scheme. It is used in TLS (Transport Layer Security) protocol. It is a block cipher based symmetric key encryption scheme. The key must be securely exchanged between the peers. The key size can be either 128, 192 or 256 bits. In our design, this encryption scheme is used extensively. Initially, RSA is used to exchange the AES keys between the nodes and after that every communication is encrypted using AES.

1.5 Challenges

We will be tackling the following challenges during the course of this project work

- Hiding the identity of peers
- Maintaining message confidentiality and message authentication
- Maintaining an efficient P2P network i.e maintaining the P2P network when peers get disconnected unintentionally

Chapter 2

Literature Review

One of the most popular pub-sub message exchange system is **Scribe**[3]. Scribe depends on the features of **Pastry P2P network**[4] which is considered to be scalable but it cannot provide anonymity to the users. Scribe introduced the use of spanning tree consisting of subscribers for efficient data dissemination.

One of the publish-subscribe systems that aims to provide anonymity is by **Datta et al**[5]. They propose a routing system based on maintaining multiple layers of weakly connected directed acyclic graphs. In this system, one or more sink nodes, which may change over time, become dissemination points receiving all publications and forwarding them to subscribers.

However, anonymity is provided only by stating that the node a receiver gets a message from may not be the original publisher. However, an adversary would still know that that node could possibly be an orig-

inal publisher. Without probabilistic analysis of this possibility, it is difficult to say how well protected the publishers actually are. Also, no mention is made as to how difficult it is to identify subscribers in the system and this has not been implemented.

The other publish subscribe system is the work by **Binh Vo and Steven M. Bellovin**[6]. They propose the use of **Tor hidden service**[7] as an anonymous network which will handle the work of hiding the true identity of users and route messages by constructing spanning tree for each topic among the peers. The approach to use Tor hidden service is quite slow and expensive because it includes maintaining relay nodes for tor and multi-layered encryption and decryption whereas there is single layered encryption and no involvement of relay nodes in our system and also it is to be noted we use virtual address routing for hiding peers identity instead of Tor hidden service. Also, many attacks have been successfully made on Tor network and it has been broken by many systems in the past.

Chapter 3

Methodology

3.1 Virtual Address Routing

IP address cannot be used because messages can be traced easily and identities of peers can be revealed, so concept of virtual address is adopted to hide peer identity. The idea of virtual address routing is taken from **ANT Routing Protocol** [8] which is used for anonymous file transfer.

A unique hashed virtual address using the SHA256 hashing algorithm is generated for every peer based on its IP Address and the current Timestamp i.e

$$VirtualAddress = H(IPAddress||Timestamp)$$

. This virtual address would be a unique for every peer and would require atleast 2^{60} operations to find a collision and hence break the hash algorithm[9, 10]. A centralised registry is maintained that stores the mapping of all the virtual addresses to IP addresses. So, when a new peer connects to the network, it registers itself to the registry and gets address of 2 to 3 peers that have the lowest degree where we define degree as

$$Degree = number\ of\ other\ peers\ connected\ to\ a\ particular\ peer$$

. The reason behind this is to **avoid formation of articulation points** in the network.

Each peer maintains a routing table which contains information which neighbour to forward when a message is received.

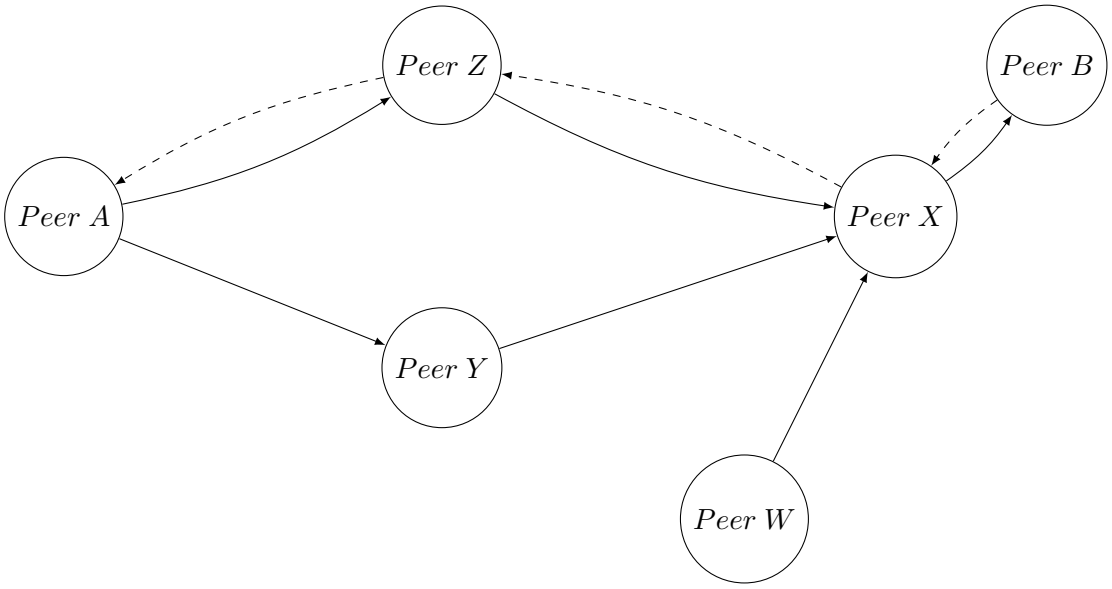

Figure 3.1: Routing of Messages among Peers

Peers	Neighbour Connections		
	$Peer\ Z$	$Peer\ Y$	$Peer\ W$
$Peer\ A's$ Address	$Forward$	$Forward$	$Don'tforward$

Table 3.1: Routing table of $Peer\ X$

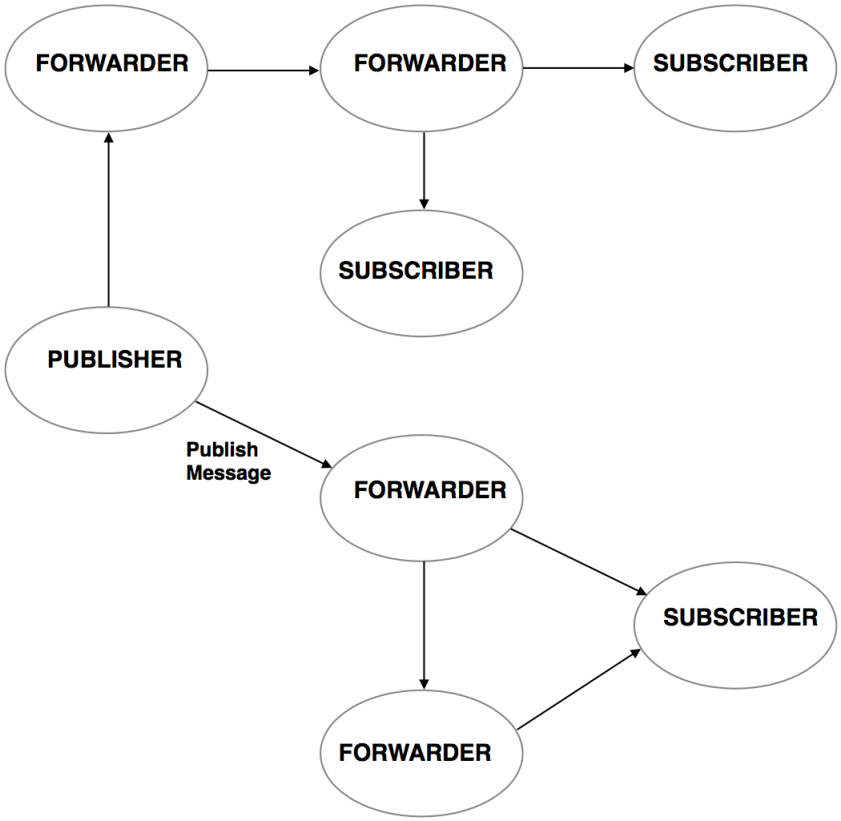

Figure 3.2: Publishing a Message

In Figure 3.1, $Peer\ X$ received messages originated from $Peer\ A$ through two of its neighbors ($Peer\ Y$ and $Peer\ Z$). $Peer\ X$ would then populate its routing table as shown in Table 3.1.

Later, when $Peer\ X$ receives a message from $Peer\ B$, it checks the destination's Virtual Address inside the message header. Because the destination address happens to be $Peer\ A$s virtual address, $Peer\ X$ then knows that it could forward the message to either $Peer\ Z$ or $Peer\ Y$. $Peer\ X$ then randomly chooses one of these neighbors and forwards the message to it. So in Virtual Address Routing, each peer maintains its routing table by learning from the packets flowing in the network.

Virtual address routing helps the message to go from the publisher node to the subscriber node through some forwarder nodes as shown in Fig 3.2

3.2 Public Key Encryption

$Public\ Key\ Encryption$ along with $Symmetric\ Key\ Encryption$ is used for maintaining message confidentiality and message authentication so that a message forwarder, who happens to be an attacker, wont be able to know content of the message and the message receiver will be assured of its source.

When a peer wants to subscribe to a topic, it broadcasts a request query along with its public key all over the network and the corresponding publisher and other subscribers of the same topic will store the virtual address and public key of the subscriber and in response, publisher will send its public key.

In a $Closed\ Network$, a hash of the passphrase will act as the key for symmetric encryption. Since authenticated subscribers have a passphrase from an earlier point in time, only they can decrypt the message.

In an $Open\ Network$, where there is no restriction on users while joining the network, message confidentiality cannot be provided because any peer can join the network and view the message no matter how we encrypt it. Message authentication can be provided in both closed and open network by signing the message with the private key of the publisher.

A common strategy used to circumvent man-in-the-middle attacks is to use certificates signed by a trusted party i.e certificate authority(CA). This strategy will be discussed in a section later on.

Alternative : **MAC**[11, 12] along with **Public Key Encryption** can also be used for the same purpose.

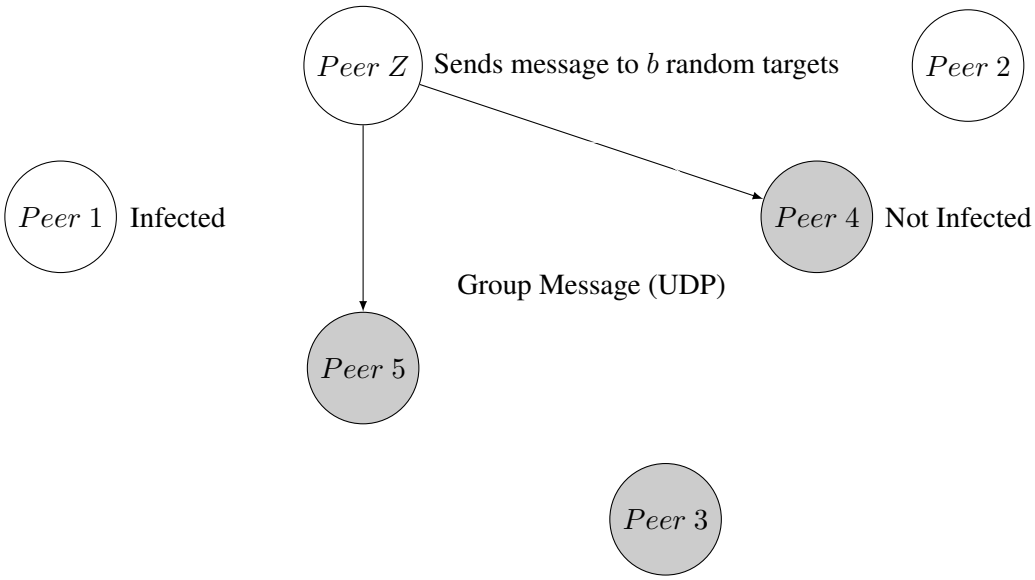

Figure 3.3: Epidemic Multicast or $GOSSIP$ Protocol

3.3 Message Dissemination

When a publisher has a message to disseminate, it needs to make sure all the subscribers receive them. Since a publisher has a list of all subscribers virtual address, one naive approach is that it can perform unicasting for each subscriber but this is very resource intensive.

A efficient way to disseminate messages to the subscribers is to use **GOSSIP**[13] protocol shown in Fig 3.3. The protocol is inspired by the form of gossip seen in social networks. A gossip spreads in a social network when it starts as a conversation between two parties. These two parties then tell two more parties each and this message spreads across the entire network. When one party hears the message for a second time, it simply identifies the message and ignores it because it has already received the message from someone else.

3.3.1 Modification of Gossip Protocol for Publish Subscribe

When a new message is to be disseminated, the publisher selects b number of subscribers from its list and sends messages to all of them. Now these randomly selected subscribers send the message

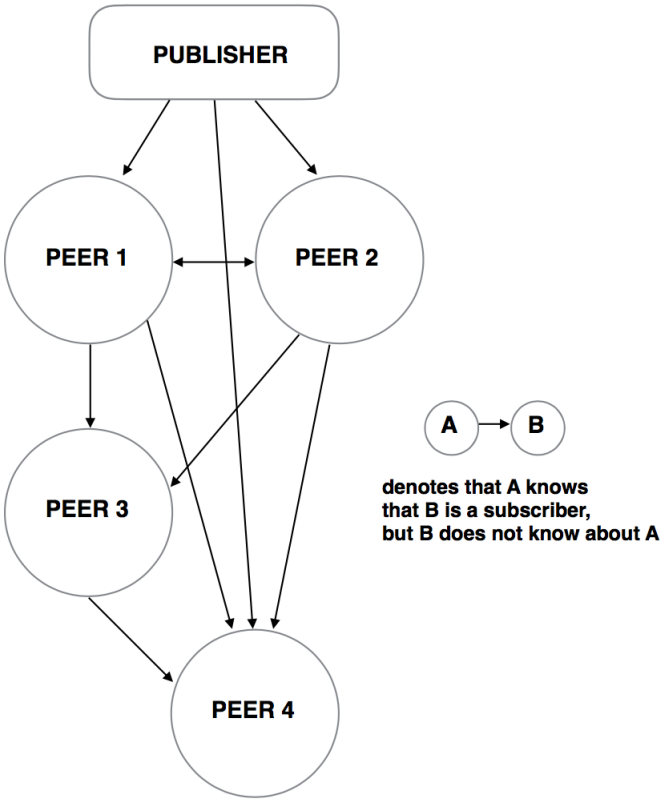

Figure 3.4: Subscriber Graph

to randomly selected subscribers since each subscriber will also have list of subscribers for its topic. When all the subscribers have received the message, the dissemination process stops. This process of dissemination will have complexity of $log_b N$, where b is the number of subscribers randomly selected. A subscriber graph is formed during this process which is shown in Fig. 3.4

3.4 Closed System Passphrase Authentication

As mentioned earlier, in a closed system, the peer needs to obtain a passphrase in order to join the network. The said passphrase needs to be added to the subscribe message. It is used to authenticate the user before joining the network. To prevent eavesdropping, the password cannot be sent as plain text or as a deterministically hashed or encrypted cipher text because such a cipher text can be used for replay attacks. So to be pevent replay&forgery attacks, the requirement for using undeterministic passphrase arises. While sending passphrase as credentials in subscribe message in the network, a passphrase as a

hash comprising of the passphrase, the source virtual address and the topic name i.e.

$$PassphraseHash = H(Passphrase||SourceVirtualAddress||TopicName)$$

is created. This generates a undeterministic passphrase field for every subscribe message which an attacker cannot easily forge. Similarly, publisher checks the credentials as

$$PassphraseHash == H(Passphrase||VirtualAddressofReceiveddatapacket||TopicName)$$

.

3.5 Key Exchange & Prevention of Man in the Middle Attack

There needs to be a mechanism to share public keys with a keyring so that we can have confidential communication amongst the users. There are various ways of key exchange between the server and the peer. One naive way is to send the peer's public key by encrypting with server's public key. This method is vulnerable to Man in the Middle attack because an attacker can intercept the message and change the keys without letting the participants know about it and this attack is identical to Man in the Middle attack in Diffie Hellman Key Exchange Protocol.

Another efficient way is to use certificates. However, there is a problem with this approach. Neither can certificates be issued to all peers nor can every peer be forced to issue a certificate. Thus, a simple approach is taken here. The project proposes the use of a **2-way Handshake Protocol** to make sure that there is no Man in the Middle Attack. The way it works is that a user sends it's public key to server along with a random number encrypted with the server's public key i.e

$$Message = PK_{server}||Enc_{PK_{server}}(r) \tag{3.1}$$

where PK_{server} is Public Key of server and **r** is a random integer. The peer then waits for a reply from the server as **r + 1** ecnrypted with the peer's public key i.e

$$Message = Enc_{PK_{peer}}(r+1) \tag{3.2}$$

, where PK_{peer} is the Public Key of the peer. This is how the project ensure that Public Keys have been exchanged with minimum vulnerability. The protocol is demonstrated in Figure 3.5

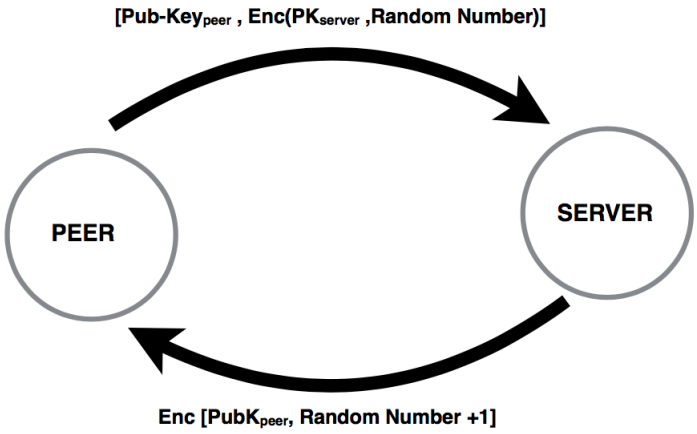

Figure 3.5: Two Way Handshake Protocol

3.6 Generation of Unique IDs

Unique virtual addresses are required for every node for it's pseudo identity in the network. The way they are generated is

$$VirtualAddress = H(physicaladdressofethernetdevice) \tag{3.3}$$

. According to international standard of issuing addresses, unique physical addresses of devices are given by device manufactures thus we can argue that when SHA256 is used, there is very less chances for virtual address collision in the network.

Similarly there is a need to generate unique id for each message that is to be circulated in the network. This is needed to prevent network flooding which is caused when a message keeps circulating in the network. A unique message id is generated as

$$MessageID = H(EhternetDevicePhysicalAddress||Timestamp||RandomInteger) \tag{3.4}$$

.The combination of random integer and timestamp ensures that the strings to hash function are unique so when SHA256 is used it can be ensured that there is very less chance for hash collison and every message ID is unique.

3.7 Message Flooding

Message flooding is the phenomenon of too many messages circulating in the network. It would simulate a Denial Of Service Attack during which resources will be over used and peers will not be able to communicate with each other during this period. This disables the system and slowly crashes it. In this project Message flooding can happen during two scenarios

3.7.1 Subsriber Message Flooding

Whenever a Subscriber joins the network, it sends a subscribe message to the network. This message goes to every peer. While traveling across the network, the message can continuously keep coming to one or more than one peers. Such a message will keep circulating in the network for an infinite amount of time and unnecessarily waste resources.

3.7.2 Publisher Message Flooding

Whenever a publisher publishes a message, it sends it to a random set of forwarders. The concerned subscriber will receive the message from all the forwarding peers that are immediately connected to itself. Thus an unnecessary amount of messages will be received by the subscriber each time one of the publishers publishes any message. The subscriber must be able to find out that the messages are same and discard other messages once it receives one of them.

3.8 Prevention of Message Flooding

Every node maintains a special data structure called a **set** which is fundamentally an array which stores unique values. This set stores message IDs. If a message ID already exists in the set it ignores the message ID, otherwise it adds the message ID to the set. The complexity of adding a single Message ID to the set is $\mathcal{O}(\log K)$where K is the size of the set. As there can be many number of messages and the number of messages cannot be predetermined, the size of the set (K) is kept constant. If the set is already full, one Message ID is removed from the set in **FIFO** manner,i.e. the first inserted message ID is removed followed by the second inserted message ID, etc. While this method doesn't eliminate message flooding completely as the size of the set is not infinite, the size is kept sufficiently large so that the probability of message flooding decreases because the messages that are repeating generally repeat

in very small time intervals.

3.9 Peer Churning

The sudden disconnection of peers from the network is called Peer Churning[14]. Sudden arrival and departure of peers is one of the major characteristics of P2P systems. The participation of a peer is very dynamic. Due to many unknown reasons, network partition can occur and nodes might get disconnected. Thus handling peer related information is a challenging job and that is why handling peer churning is a major issue in P2P architecture.

The availability of peers is determined through a mechanism known as heartbeat mechanism where a peer sends periodic heartbeat to the neighboring peers. Here, a heartbeat signifies a UDP packet with some payload. If a peer doesn't receive the heartbeat from it's neighbor, then it marks that neighbor as a dead peer. We implement this by using an Open Source tool called Apache Zookeeper which is used for coordination of distributed applications.

Zookeeper creates a special kind of node to detect peer availability called an ephemeral node. These nodes exist as long as the session that created the nodes is active. When the session ends the node is deleted. Thus, checking whether an ephemeral node is active or not can give us information about the availability of the peer. An absence of an ephemeral node signifies that the peer that had created it has disconnected itself from the network. However, how do the other peers automatically come to know when an ephemeral node has been terminated? Zookeeper also provides event listeners called watches. These watches can be made to listen to various events. We can register a watch to listen on the activities of an ephemeral node. This watch will tell the system when the ephemeral node is down and can necessary actions can be taken during the callback of the event listener. Thus, the project has used two features of Zookeeper, ephemeral nodes and watches to detect peer churning in the network.

3.10 UDP Packet Spoofing

Since the project is using Virtual Address, TCP connections cannot be made because without IP address, the famous three way handshake protocol will not work. Therefore, the project uses UDP as the transport protocol. UDP provides an even more helpful feature for providing more anonymity i.e. spoofing the UDP packets with **false source IP addresses** so when a publisher publishes a message, it can change the source IP address of the packet and hence it becomes even more difficult for an attacker

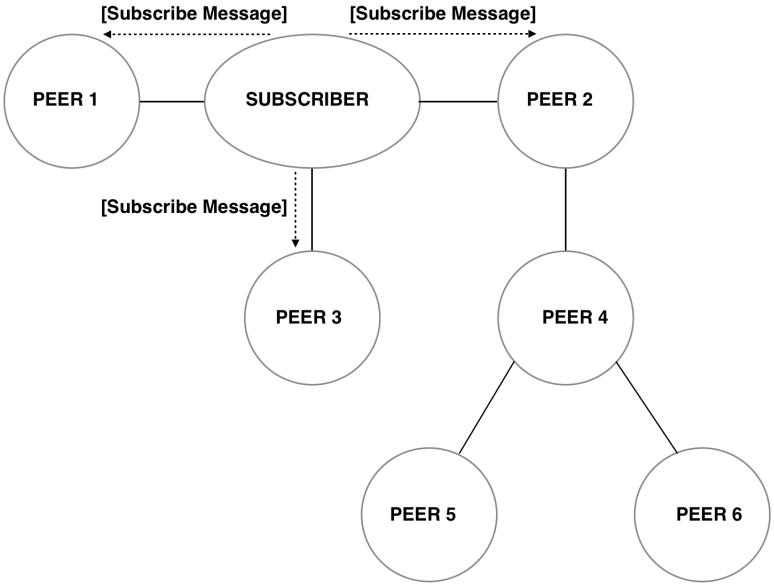

Figure 3.6: Subscription Message Broadcast

to find the identity of publisher.

3.11 Protocol Messages

1. **Connect Request Query**

   This is the message sent by a peer to the registry when it wants to connect to the network. This Message includes a message type (M_Type)the Virtual Address (V_Addr) generated by the peer and its Public Address (Pu_Addr). Pu_Addr will be used by the registry to construct a public keyring.

$$\textbf{Message format} : M\_Type \,||\, V\_Addr || \, Pu\_Addr \tag{3.5}$$

2. **Reply from registry**

   This message is sent by the registry as a reply to the Connect Request Query. This message

includes mapping of V_Addr to IP_Addr of peers who are already connected to the network.

$$\textbf{Message format} : M\_Type \,||\, f : V\_Addr \rightarrow IP\_Addr \tag{3.6}$$

3. **Subscription Request**

   This message is sent when a peer wants to subscribe to a topic. It includes the hashed topic name, hashed passphrase($\mathcal{P}$) if it wants to connect to a closed network, the source Virtual Address and a Message Id. Each message will have a unique Message Id(M_ID), which will be $\mathcal{H}(V\_Address \,||\, Timestamp(T))$. Message id is used to prevent same message from circulating in the network twice. When the subscriber sends this message, it is routed across the network in a manner shown in Fig 3.6

$$\textbf{Message format} : M\_Type \,||\, M\_ID \,||\, Source\, V\_Addr \,||\, \mathcal{H}(Topic) \,||\mathcal{H}(\mathcal{P}) \tag{3.7}$$

4. **Subscription Request Reply**

   This message will include the Unique M_ID, Source(S) and Destination(D) V_Addr, $\mathcal{H}(Topic)$ and Signature of the Publisher (Sign_Pub).

$$\textbf{Message format} : M\_Type \,||\, M\_ID \,||\, S\, V\_Addr \,||\, D\, V\_Addr \,||\, \mathcal{H}(Topic) \,||\, Sign\_Pub \tag{3.8}$$

5. **Publish Message**

   This message is sent by the Publisher when a new message is to be published. This message contains S and D V_Addr, $\mathcal{H}(Topic)$, Encrypted Message (E(K,$\mathcal{M}$)) and M_ID.

$$Message\ format : M\_Type \,||\, M\_ID \,||\, S\, V\_Addr \,||\, D\, V\_Addr \,||\, \mathcal{H}(Topic) \,||\, E(K, \mathcal{M}) \tag{3.9}$$

3.12 Environment Setup

The distributed nature of the application adds difficulty to the development of the project. It is infeasible to write the program and continuously setup a real distributed environment to test our code. So the project needed a tool that allowed the simulation of a distributed environment. Virtual machines would have a solution to our problem but it consumes too much of the host machine's resources and slows down the performance of the system which is being used for development. Linux containers are a better option compared to virtual machines because they provide same set of features like isolation and resource control but are very lightweight compared to Virtual Machines. The project has used a software called Docker [15] to use containers. A Python library pycrypto has also been used to use the ready made Cryptographic and Hashing functionalities. We use Apache Zookeeper to realize when the peer has exited the network, an event called peer churning. These three softwares and libraries are now discussed in some detail.

3.12.1 Containers

A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Available for both Linux and Windows based apps, containerized software will always run the same, regardless of the environment. Containers isolate software from its surroundings, for example differences between development and staging environments and help reduce conflicts between teams running different software on the same infrastructure.[16]. Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware, containers are more portable and efficient.

Containers Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), and start almost instantly.

Virtual Machines On the other hand, Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, one or more apps, necessary binaries and libraries - taking up tens of GBs. VMs can also be slow to boot. The difference between the two is illustrated in the figures 3.7 and 3.8

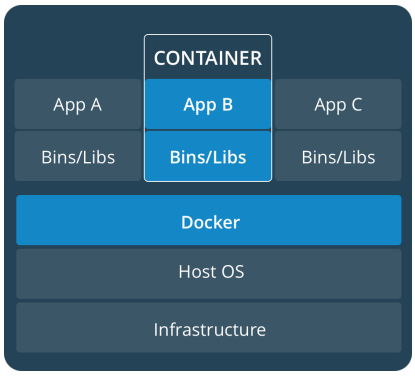

Figure 3.7: Containers

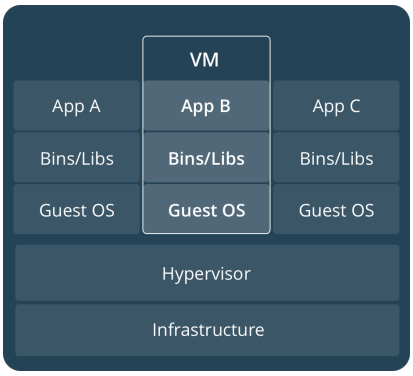

Figure 3.8: Virtual Machines

3.12.2 Docker

Docker is an open-source project that automates the deployment of applications inside software containers. From Docker's website : "Docker automates the repetitive tasks of setting up and configuring development environments so that developers can focus on what matters: building great software." [17] Docker provides an additional layer of abstraction and automation of operating-system-level virtualization on Windows and Linux. Docker uses the resource isolation features of the Linux kernel such as cgroups and kernel namespaces, and a union-capable file system such as OverlayFS and others to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines. Docker is a tool that can package an application and its dependencies in a virtual container that can run on any Linux server. This helps enable flexibility and portability on where the application can run, whether on premises, public cloud, private cloud, bare metal, etc. Docker implements a high-level API to provide lightweight containers that run processes in isolation. Docker is secure by default and it helps us in building scalable apps. Docker delivers apps by flexibly integrating with existing processes and enterprise systems for storage, networking, logging, and authentication.

3.12.3 Zookeeper

Apache ZooKeeper[18] is a software project of the Apache Software Foundation. It is essentially a distributed hierarchical key-value store, which is used to provide a distributed configuration service, synchronization service, and naming registry for large distributed systems. We have used Zookeeper to detect session expiration. Expirations happens when the Zookeeper cluster does not hear from the peer within the specified session timeout period (i.e. no heartbeat). Zookeeper also has a feature called watches or basically event listeners that can take necessary actions when there is any unusual activity in the peers. This project has registered a watch on each peer that notifies Zookeeper instance when the session of the peer expires, an event called peer churning. On being notified, the Zookeeper instance instructs the peer that was previously a neighbour of the disconnected node to get connected to two or more other nodes. The way in which Zookeeper is used in our project is demonstrated by Fig 3.9

3.12.4 Python

Python is an extremely widely used high-level programming language for general purpose programming. It is an object oriented and interpreted language. Python has a design philosophy that helps programmers write more readable code with indentations and whitespaces rather than curly braces in other programming languages such as JAVA or C++. It also has a syntax that allows programmers to express concepts in fewer lines of code. Due to all these features, a myriad of Python libraries exist that make the job of programmers very easy, providing them with ready made code that allows programmers to develop applications very easily. This project has used Python as the primary programming language for all the above reasons.

3.12.5 PyCrypto Library

PyCrypto Library [19] is a toolkit written in Python. This is a collection of both secure hash functions (such as SHA256 and RIPEMD160), and various encryption algorithms (AES, DES, RSA, ElGamal, etc.). The project has used the PyCrypto library for many purposes such as Encrypting the messages using AES, hashing the passphrase and topics using SHA256 and Public Key Encryption in the Two Way Handshake Protocol using RSA, etc.

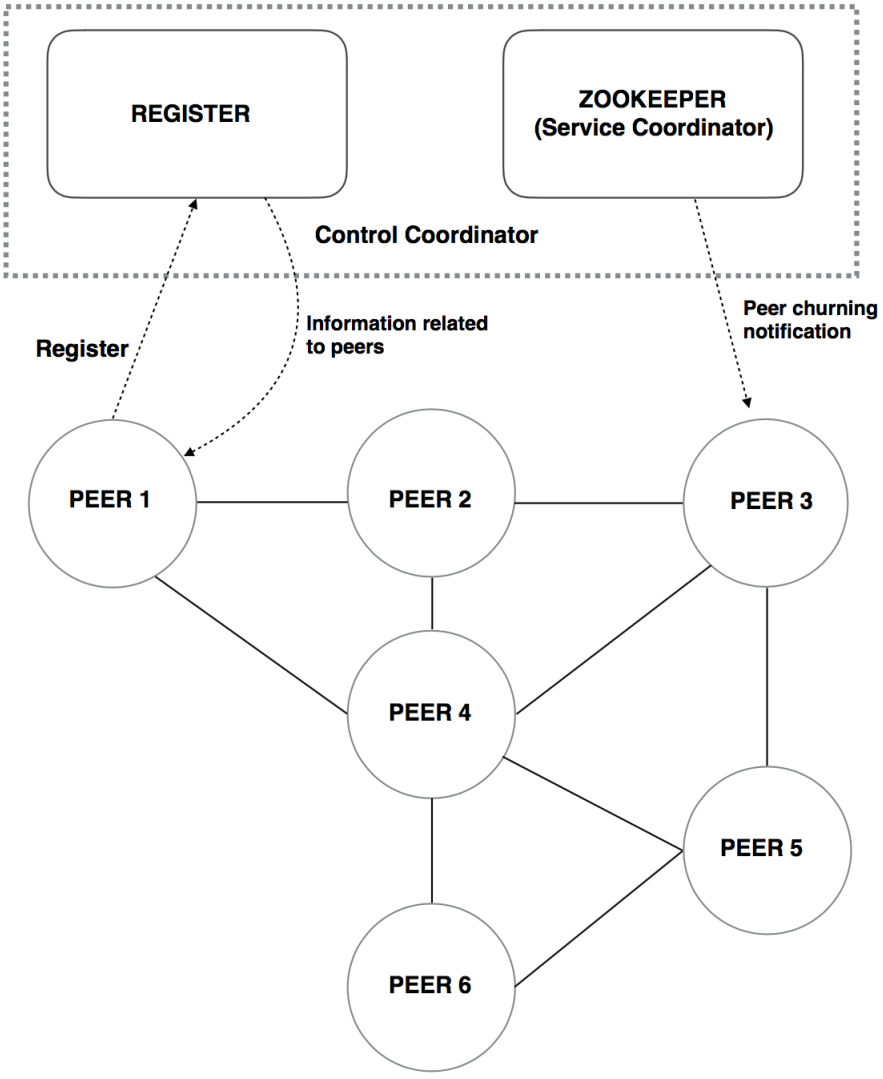

Figure 3.9: Using Zookeeper to handle Peer Churning

Chapter 4

Results and Discussion

4.1 Performance Analysis

4.1.1 Latency of Message Transfer

This work is defining latency as the number of peers that are involved in passing one message across the network. This latency is of the order of $\mathcal{O}(\log_b S * D)$where S is the number of subscribers and D is the diameter of the subscriber graph. A sample subscriber graph is shown in Figure 3.4. The length of the ”longest shortest path” between any two graph vertices of a graph.

4.1.2 Load on peer due to message transfer

There are multiple message dissemination points due to Gossip Protocol as shown in Fig 3.3. Thus, load is distributed. It is precisely $\mathcal{O}(1)$ for each peer.

4.1.3 Processing a Packet

To process a packet, four things are done at each node :

- Check Routing Table for Forwarder Peers. This occurs with a time complexity of $\mathcal{O}(T)$, where T is the size of the routing table. Here a Hash Table Data Structure has been used. More information regarding Routing Table is explained in Subsection 3.1. [1]
- Checking Message Set for repeating Messages and inserting them into the Set if the message is new. This operation has a time complexity of $\mathcal{O}(\log K)$, where K is the size of the Message Set. [2]
- Encryption of Message using RSA. This operation has an optimized time complexity of $\mathcal{O}(\log N^{1.585})$, where N is the size of the field used in RSA algorithm. This complexity is lesser than $\mathcal{O}(\log N^2)$ because of a faster multiplication called Karatsuba's algorithm. [3]
- Decryption of Message using RSA. Similarly, this operation has an optimised time complexity of $\mathcal{O}(\log N^{2.585})$, where N is the size of the field used in RSA algorithm. The complexity is more than that of encryption because here d is proportional to N. [20] [4]

Combining all the above time complexities, the total time complexity for processing a packet at one node is $\mathcal{O}(T + \log K + \log N^{1.585} + \log N^{2.585})$.
As the latency of message transfer is $\mathcal{O}(\log_b S * D)$, the total time taken to process a packet while sending it across the network is $\mathcal{O}(\log_b S * D * (T + \log K + \log N^{1.585} + \log N^{2.585}))$.

4.1.4 Memory Complexity

Each node stores the information of it's neighbours and the subscribers. As there can be a total of $\mathcal{O}(N)$subscribers in the network and the number of neighbours of a node is always constant, the memory complexity at each node is $\mathcal{O}(N)$.

4.2 Security Analysis

4.2.1 Message Confidentiality

One of the primary aims of this work was to achieve message confidentiality. The project achieved full message confidentiality via RSA public key encryption. Assuming that RSA is computationally secure [1], messages are fully confidential and the message encryption scheme is secure.

4.2.2 Message Authentication

As mentioned in Section 3.4, the project is using the hash of the passphrase along with the topic name and the source virtual address as the key to HMAC algorithm. This ensures that the key is unique as the virtual address comprises of timestamp. The scheme of our project has message authentication as an attacker cannot produce replicate the key used in HMAC and as HMAC is secure in itself [11], the system supports message authenticity.

4.2.3 Anonymity of Peers

The next main objective of the project was to provide anonymity of the publisher and to not let the publishers and subscribers to get to know each other's identities. The project has achieved this through Gossip Protocol and Virtual Address Routing. In a fairly large network, as there are a lot of forwarder peers, the subscriber will not come to know the address of the publisher as the message will come through a lot of nodes in the middle that are used for message dissemination under Gossip Protocol. Thus, the identity of peers is hidden.

4.3 Assurance of Connectivity of Network

The diameter of a disconnected graph is infinite as the longest path between any node and the node(s) disconnected from the network is of infinite length.
Since the message latency of our system is $\mathcal{O}(\log_b S * D)$where S is the number of subscribers and D is the diameter of the subscriber graph as mentioned in Subsection 4.1.1, it must be assured that the subscriber graph doesn't become disconnected. The danger of the graph getting disconnected will only arise when one or more of the peers leave the network for intensional or accidental reasons. In all other

reasons, the graph is connected as each peer is connected to at least two other peers. In the case of a peer quitting the network, Zookeeper comes to the rescue. This scenario has been explained in detail in Subsection 3.12.3.

4.4 Comparison with existing system

Here a comparison between our work and the work by Binh Vo and Steven Bellovin [6] is shown.

4.4.1 Usage of Tor

The paper by Binh Vo and Steven Bellovin used Tor network [7] to ensure anonymity of peers. However in many occasions, Tor has been broken and proven to be insecure via various attacks [21] [22]. Our system doesn't use Tor for anonymity. Instead it used GOSSIP Protocol and Virtual Address Routing.

4.4.2 Space used by each peer

In Binh and Steven's system, the space used by each peer is $\theta(N)$, where N is the total number of nodes in the network. As far as our system concerned, the space used by each node is $\theta(S)$, where S is the number of subscribers in the network. As S$\leq$N, this complexity can be better than the previous complexity of Binh and Steven, if the number of subscribers is lesser than the total number of nodes in the network. In other words our system has a space complexity of $\mathcal{O}(N)$, where N is the total number of nodes in the network. Thus, in many networks, our system will use lesser space than Binh and Steven's system.

4.4.3 Skewed Complexity Analysis by Binh Vo and Steven Bellovin

Tor uses layered encryption, i.e. Tor encrypts and decrypts the message at each layer of it's network while passing it down from one network to the other. The paper has by Binh Vo and Steven Bellovin not taken into the consideration the time complexity that arises due to the routing of messages through these layers in the form of encryption and decryption. Let this neglected time complexity term be denoted as T. Thus the paper is skewed in terms of their analysis of latency of the message. In the paper, it is mentioned that the total latency of their system for a message to reach from a publisher to a subscriber is $\mathcal{O}(S)$, where S is the number of subscribers in the network. Whereas the time complexity should have actually

been $\mathcal{O}(S * T)$. On the other hand the message latency of our system is $\mathcal{O}(\log_b S * D)$where D is the diameter of the subscriber graph as shown in Subsection 4.1.1

Chapter 5

Summary and Conclusion

As a result of this project, various ideas from different sources were combined and an approach to make a secure system that can successfully hide the identity of the participants in message exchange in a peer to peer network has been achieved. This system also provides full message confidentiality and message authentication. There has been very little research on this topic. Reference has been taken from the paper by Binh Vo and Steven Bellovin [6]. A different approach, unlike the usage of Tor done by the paper cited was introduced by this project. The idea of using Virtual Address Routing and GOSSIP Protocol to provide anonymity to the peers that are joining the network was introduced by this project.
Since there is very little research that has been done on this topic, there weren't many sources to compare our results. However, the comparison with Binh Vo and Steven Bellovin's system provided in this report, shows that this system is arguably better than theirs.
Also due to the lack of resources to study from, significant setbacks were faced while developing the system. Various ideas and components like GOSSIP Protocol, Apache Zookeeper, PyCrypto Library,

a cryptographic toolkit written in Python and Docker for simulation of a distributed environment were combined in this project. Combining all these components and simulating a distributed environment for running and testing our application was considerably difficult.

As of now, there has been a realization that this system has a minor drawback. The message dissemination process is slow as the message follows a random path from publisher to subscriber. This step can be optimized to increase the speed of message dissemination.

Bibliography

[1] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[2] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[3] Miguel Castro, Peter Druschel, A-M Kermarrec, and Antony IT Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications*, 20(8):1489–1499, 2002.

[4] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 329–350. Springer, 2001.

[5] Ajoy Kumar Datta, Maria Gradinariu, Michel Raynal, and Gwendal Simon. Anonymous publish/subscribe in p2p networks. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 8–pp. IEEE, 2003.

[6] Binh Vo and Steven Bellovin. Anonymous publish-subscribe systems. In *International Conference on Security and Privacy in Communication Systems*, pages 195–211. Springer, 2014.

[7] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[8] Mesut Gunes, Udo Sorges, and Imed Bouazizi. Ara-the ant-colony based routing algorithm for manets. In *Parallel Processing Workshops, 2002. Proceedings. International Conference on*, pages 79–85. IEEE, 2002.

[9] Marc Martinus Jacobus Stevens et al. *Attacks on hash functions and applications*. Mathematical Institute, Faculty of Science, Leiden University, 2012.

[10] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full sha-1. In *Annual International Cryptology Conference*, pages 17–36. Springer, 2005.

[11] Hugo Krawczyk, Ran Canetti, and Mihir Bellare. Hmac: Keyed-hashing for message authentication. 1997.

[12] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, 2000.

[13] André Allavena, Alan Demers, and John E Hopcroft. Correctness of a gossip based membership protocol. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 292–301. ACM, 2005.

[14] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202. ACM, 2006.

[15] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[16] Docker Containers. What is a container. `https://www.docker.com/what-container`. [Online; accessed 17-April-2017].

[17] Docker. What is docker. `https://www.docker.com/what-docker`. [Online; accessed 17-April-2017].

[18] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association.

[19] Dwayne Litzenberger. Pycrypto. `https://github.com/dlitz/pycrypto`. [Online; accessed 17-April-2017].

[20] Mark Shand and Jean Vuillemin. Fast implementations of rsa cryptography. In *Computer Arithmetic, 1993. Proceedings., 11th Symposium on*, pages 252–259. IEEE, 1993.

[21] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against tor. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 11–20. ACM, 2007.

[22] Timothy G Abbott, Katherine J Lai, Michael R Lieberman, and Eric C Price. Browser-based attacks on tor. In *International Workshop on Privacy Enhancing Technologies*, pages 184–199. Springer, 2007.

ACKNOWLEDGMENTS

This thesis would not have been possible without the help and cooperation of many. We would like to the thank following people who helped me directly and indirectly in these four years of our B.Tech. program.

First and foremost, we would like to express our gratitude to our project guide **Dr. R. Padmavathy**, Assistant Prof., Dept. of CSE. As a project guide, she has inspired us by her research intuition and attitude in doing research work. We are extremely grateful for her guidance and high standards of her technical paper review made our B. Tech. program a productive one. Her extensive knowledge helped us gain speed in our research work.

(Saumyajit Dey)

(Sushil Paneru)

(Aman Alam)