2016

A new framework of decentralized social networks

Ting Wu
Iowa State University

A new framework of decentralized social networks

by

Ting Wu

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:

Wensheng Zhang, Major Professor

Carl K Chang

Xiaoqiu Huang

Iowa State University

Ames, Iowa

2016

DEDICATION

I would like to dedicate this thesis to my fiance Wen without whose encouragement and support I would not have been able to complete this work. I would like to thank my family and friends for their love and guidance during the writing of this work.

TABLE OF CONTENTS

LIST OF FIGURES vi

LIST OF TABLES vii

ACKNOWLEDGEMENTS viii

ABSTRACT ix

CHAPTER 1. INTRODUCTION 1

CHAPTER 2. RELATED WORKS 4

CHAPTER 3. PROBLEM DEFINITION 7

3.1 System Model 7

3.2 Threat Model 7

3.3 Design Goal 8

3.3.1 Functionality goal 8

3.3.2 Security goal 8

CHAPTER 4. SYSTEM DESIGN 9

4.1 System Overview 9

4.2 Security Overview 9

4.3 System Components 10

4.3.1 Personal server 10

4.3.2 Relay server 13

4.3.3 Client agent 14

4.4 Initialization 14
4.4.1 Initialization of relay server 14
4.4.2 Initialization of personal server 14
4.4.3 Initialization of client agent 15
4.5 People Search 15
4.6 Friend Making 15
4.7 Contact Management 17
4.7.1 Adding a group 17
4.7.2 Managing a contact 17
4.8 Message sharing 17
4.8.1 Building block 17
4.8.2 Posting a message 18
4.8.3 Retrieving a directory of meta data of messages 18
4.8.4 Retrieving posts 19
4.9 Activity awareness 20
CHAPTER 5. IMPLEMENTATION 21
5.1 Communication 21
5.1.1 SSL socket 21
5.1.2 Message types 23
5.2 Security 25
5.2.1 Encryption 25
5.2.2 Verification process 27
5.2.3 Access control 27
5.3 Personal Server 29
5.3.1 Overview of initialization 29
5.3.2 Request handling 30
5.3.3 Profile definition 30

5.3.4 Data structure 31
5.4 Relay Server 32
5.4.1 Data structure 32
CHAPTER 6. EVALUATION 36
6.1 Scenario One 36
6.1.1 Settings 36
6.1.2 Result 37
6.2 Scenario Two 38
6.2.1 Settings 38
6.2.2 Result 38
6.3 Scenario Three 38
6.3.1 Settings 39
6.3.2 Result 40
6.4 Scenario Four 40
6.4.1 Settings 40
6.4.2 Result 40
6.5 Security Analysis 41
CHAPTER 7. CONCLUSION 43
BIBLIOGRAPHY 44

LIST OF FIGURES

Figure 4.1	Basic system model	10
Figure 4.2	Making Friend Procedure	16
Figure 4.3	Notification Flow From PS of owner A to user B	20
Figure 5.1	SSL Handshaking	22
Figure 5.2	Message Header	24
Figure 5.3	Verification Process	28
Figure 5.4	Flow chart of PS initialization	34
Figure 5.5	Database tables at Personal Server	35
Figure 5.6	Database tables at Relay Server	35
Figure 6.1	Response time of querying posts with different table sizes	37
Figure 6.2	Response time for queries with different post sizes	39
Figure 6.3	Response time of different query intervals	41

LIST OF TABLES

Table 5.1 Summary of requests handled by PS 30

Table 6.1 Configuration details 37

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who gave me with all aspects of help in research and writing of this thesis. First and foremost, I would like to thank Dr. Wensheng Zhang for his guidance, patience and support throughout this research. His insights and words of encouragement have often inspired me and helped me when I had difficulties in research. Also I would like to thank my lab mates Qiumao Ma, Daolin Cheng, Keji Hu for discussion and help. I would also like to thank my committee members, Dr. Carl K Chang and Dr. Xiaoqiu Huang, for their help and contribution to this work.

ABSTRACT

Due to the increasing popularity of online social networking services, more and more people have been outsourcing their personal data to these service providers. Data privacy has gradually become a big concern. Decentralized social network proposed in recent researches has been a promising alternative. But there lacks an efficient system of decentralized social network with complete social networking functions. In this thesis, we design a new framework of decentralized social network and implement the prototype. Our experiments shows that our design protects data privacy as well as achieve communication efficiency.

CHAPTER 1. INTRODUCTION

Online social networking services have been very popular during recent years. They have provided online platforms for users to maintain personal profiles and to share messages among them. As people have enjoyed the benefits brought up by these service providers, some issues have come to surface. Since users keep outsourcing their personal data to provider servers, these providers can take full control of the way data is used. For example, they may sell data to merchants to produce customized advertisements or to dictatorial governments to monitor antigovernment activities. Data to some degree no longer belongs to owners, bringing in data privacy concerns.

As an alternative approach, decentralized social network helps users protect their data security and privacy by allowing users to choose their own data servers at will. The control of data could be transferred from any central social networking server to the hands of users themselves. Diaspora [Diaspora (2010)] is a widely used decentralized social network. However its property that each server is public in the social network might make it vulnerable to be attacked. Also it lacks of ways to make users know each other. There have been research works studying on different aspects of decentralized social network. PrPl[Seong (2010)] proposed the design of personal cloud bulters over personal servers, but their performance was not promising. Vis-a-Vis[Shakimov (2011)] focused only on location-based decentralized social networking instead of providing a complete solution tousers to know each other. Works in [Jahid (2012); Nilizadeh (2012); Buchegger (2009); Cutillo and Strufe (2009)] have studied problems such as data confidentiality, data availability and server location privacy, in decentralized social networks with distributed

hash table as a backbone. However the reliance on DHT could limit the social functions, making it more blog-like wall, and also less communication efficiency.

In general some problems remain unsolved in current proposals of decentralized social network: (1) Some social functions have not been provided effectively in decentralized social network which impedes it from being applied and popularized in real life. For instance, searching people based on some keywords is an important feature of state-of-the-art social networks. This function, however, is difficult to implement in a decentralized social network due to the lack of central machine serving as a host for all users. (2) As most individual servers holding personal data are public in the social network, they could be very vulnerable to attacks. Also, it would pose a big challenge to require regular users of a social network to take care of their servers' security configuration. (3) Some works lack of efficient performance as they depend on complex network architecture such as Chord Stoica (2001).

In order to resolve the above problems and to achieve data confidentiality that protects personal data from unauthorized access, this thesis proposes a new framework of decentralized social networking system. We introduce three components in the system: personal servers, client agents, and a relay server. Each personal server where personal data is stored can run in any subnet behind a firewall such that it can be protected from some outside attacks. To enable users to access data in personal server at any time, users can use portable client agents to conduct some social actions, such as posting messages. As data is encrypted and transferred between personal servers and client agents via a relay server, the relay server plays an important role of routing messages in the system. Moreover, the relay server can be used as a central platform to store public information of users and to facilitate socialization. Under such an architecture, this project develops protocols to realize common social networking functions, including profile management, people search, contact management, and information sharing. And then it implements the prototype of the system with these essential functions. Evaluations based on the pro-

totype, using response time of message sharing as the major performance measurement, have been carried out. The results shows that this design achieves good communication efficiency.

The rest of the thesis is organized as follows:

- Chapter 2 introduces recent related work in decentralized social network;
- Chapter 3 elaborates problems to be resolved;
- Chapter 4 presents our system design and protocols;
- Chapter 5 gives some details about the system implementation;
- Chapter 6 shows the results of experiments;
- Chapter 7 concludes the thesis.

CHAPTER 2. RELATED WORKS

In the recent years, there have been various research works that have studied different architectures of decentralized social network, such as, unstructured peer-to-peer(P2P) architecture, or structured: distributed hash table (DHT) [Stoica (2001)] architecture. Some researchers have studied the trade-offs between privacy, cost and availability of different personal data server choices.

Diaspora [Diaspora (2010)] is a decentralized social network in widely use for the purpose that user data can be protected from monitoring and analyzing. It adopts and unstructured P2P architecture where each user can choose an existing server (called pod) or set up its own host server for data storage. When a post is made by one user, the post is saved into the local database and sent to contacts on different servers.Data is stored without encryption and a certain degree of access control is enforced by pods. Compared to our work, Diaspora is different in: (1) It requires storage servers to be in the public network (instead of private subnets), which needs more careful security protection. (2) Personal data is not encrypted. (3) The adoption of unstructured P2P architecture requires each server to maintain the IP addresses of other servers that host its contacts, which may pose a higher management cost. (4) it doesn't support user search based on keywords. Users can find a contact on the condition that they need to know the id of the contact.

PrPl [Seong (2010)] is a decentralized social networking prototype that aims at enabling users to control their own data. It allows users to store data in their own preferable devices, e.g., home server, or paid vendor, and to use social applications across different

domains while preserving data security. Upon this, a layer called personal-cloud butlers was proposed to keep meta-data such as index of personal data, and access control policies for personal data and user ID management. A database query language SocialLite is also proposed to enable access data from different Butlers, which makes it easier for a developer to develop social applications. Their experiments showed that query results can be obtained within a couple of seconds. Unlike PrPl, our work protects data servers from disclosure to the public network and due to our simpler architecture, query results can be obtained within milliseconds.

Vis-a-Vis [Shakimov (2011)] proposes a design of a decentralized social network framework based on virtual individual servers. By enabling various degrees of location sharing among different groups, VIS preserves privacy of location information. Due to the focus on location-based applications, the work lacks of solution to address other social networking functions such as the support for users to get to know each other.

There are several projects of distributed social network systems which have benefited from distributed hash table (DHT-based) [Stoica (2001)] architecture due to decentralization and high scalability of DHT. For the purpose of protecting the confidentiality and availability of user content and privacy of user relationships, DECENT [Jahid (2012)] is an architecture that uses DHT to store user data and utilizes attribute-based encryption(ABE) to realize access-control policy and protect data confidentiality. To improve the query efficiency on DECENT, Cachet [Nilizadeh (2012)] maintains continuous connections with online friends to receive updates directly instead of retrieving after the data is posted in DHT. The dependence on DHT makes systems more suitable to provide blog-like services rather than messaging-like ones.

PeerSoN [Buchegger (2009)] aims at maintaining OSN fectures as well as protecting data privacy and realizing communication even in poor Internet connectivity. It proposes an architecture of decentralized social network with two-tier system. The first tier is DHT which can provide look-up service and the other consists of peers representing users who

communicate with each other directly. They propose direct exchange in real-life physical social network without Internet connectivity, however this is not the usual case because most friends might be long distance away. Protocols on how to make friends in the social network has been omitted.

Safebook [Cutillo and Strufe (2009)] is a distributed social network scheme based on DHT as a peer-to-peer substrate and on a special structure named Matryoshka which is built on the trust relation between friends. The main objective is to protect privacy of users. The component Matryoshka helps achieve anonymity and untraceability of user communications and encryption enforces data confidentiality, but compromises communication efficiency.

Shakimov et al.[Shakimov (2009)] compare three schemes for decentralized social network on the basis of virtual individual servers: cloud computing virtual machine, home desktop machines and hybrid of desktop for normal operation and cloud virtual machine when failure of the former. Normally, a personal server could be a desktop machine that is always running or a virtual machine in the cloud. Desktop machines that can keep alive always can also meet the same level of availability of cloud virtual machines.

Richter and Koch [Richter and Koch (2008)] have identified six basic functionalities of social networking services. There are identity management where users can manage personal profiles, expert finding which provides a way to enable users to search other users who have interests in common, contact management where users can make friends and separate friends into different groups which can be enhanced by access controls, context awareness presenting the ability to be aware of other people who have a common context, network awareness being the capability to be notified of activity updates from contacts, and exchange where users can exchange information directly or indirectly.

CHAPTER 3. PROBLEM DEFINITION

3.1 System Model

We consider a system that is composed of one routing server (called relay server) in the cloud, and of multiple personal servers in different domains which can be protected by firewalls, and of client agents which can be in mobile devices or web browsers. A personal server is the place where personal data is stored. For personal servers, users can choose their home machines, rent cloud machines, or share machines with friends or family members. A user can install a client agent in a mobile device, laptop or desktop. The client agent can retrieve personal information from personal servers via the relay server. The relay server plays the role of multiplexing messages among personal servers and client agents. Each personal server keeps a long-term connection with the relay server.

3.2 Threat Model

Regarding the threat model, we make the following assumptions:

- The relay server is honest but curious. In other words, it will always obey the regulations, honestly routing requests and responses to target machines. But it can be curious about the real content in messages routed.
- A personal server is trustworthy. It always behave normally.

Besides, we also consider the following two types of adversaries: (1) outsider adversaries who want to eavesdrop communication between viewers and personal servers or even tamper messages sent over connections; (2) inside-network machines which might be corrupted by malicious ones, aiming at attempting to get data from target users who don't authorize them to access.

3.3 Design Goal

The design goal we try to achieve is made up of the goal in social networking functionality and that in data security and privacy:

3.3.1 Functionality goal

With the functionalities introduced by [(Richter and Koch, 2008)] in mind, we make efforts to realize these social functions: profile management, contact management, people searching, friend making, activity awareness(notification) and information sharing consisting of posting and retrieval of messages. The first two functions allow users to manage own profiles and friends, such as grouping friends with different labels. The next two functions provide means to know contacts with something in common. The last ones enable users to share data with whom they authorize to access in a cost-effective way.

3.3.2 Security goal

Firstly, we let owners take control of personal data and make decision they can share data with. Secondly, we should prevent personal servers from disclosing publicly and being attacked easily. And more, we need to ensure the confidentiality and integrity of personal data from unauthorized parties, including the relay server, outsiders, and malicious insiders.

CHAPTER 4. SYSTEM DESIGN

4.1 System Overview

The system consists of three major components: personal servers(PSes), client agents(CLs) and a relay server(RS). The basic system model with one PS and one CL and the one RS, is in Figure 4.1. PSes are hidden in private subnets behind firewalls and host personal data for users. Client agents can be portable such that users can access their data at any place. It's accomplished by RS that stands in between. PSes can establish long-term connections with RS, which also can be connecting with CLs. Meanwhile PS and CL can be movable without concerning about notifying others to its new address, as RS is the only one that needs to be notified. RS is the proper place to hold public profiles from users, as it's very convenient for users to search people with keywords from RS.

4.2 Security Overview

To prevent outsiders from eavesdropping data exchange between CL and RS and between PS and RS, we deploy Secure socket layer protocol(SSL) for all connections. To prevent RS from learning content of messages posted by CLs and sent by PSes, such type of messages are encrypted with secret keys shared between contacts. Public key encryption is utilized in scenarios where identity verification is in need. We assume that all the underlying encryption mechanisms are secure such as SSL, symmetric encryption and public key encryption. Access control is enforced by PS to prevent personal data from being accessed by unauthorized parties.

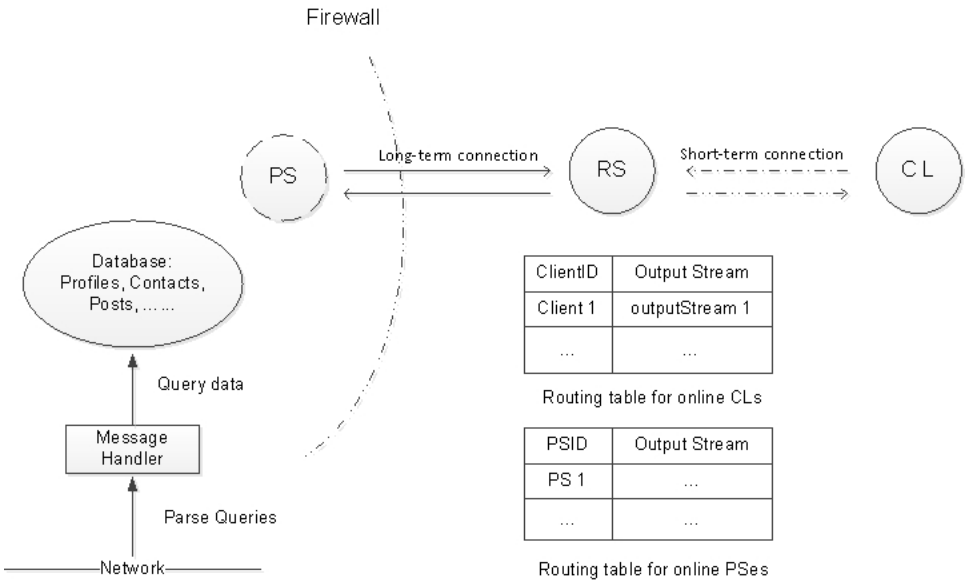

Figure 4.1: Basic system model

4.3 System Components

4.3.1 Personal server

Personal server(PS), can be a home server or a cloud machine in a subnet behind a firewall. The main functions of a Personal Server are: (1) hosting personal data for a owner or a group of owners; (2) setting up a long-term secure connection with the Relay Server; (3) accepting requests from clients or other Personal Servers via Relay Server; (4) processing requests and producing responses; (5) sending response to requester via Relay Server.

Data Model As PS offers a platform to store data content, it needs to keep various data. We categorize into the following groups:

- *PS's own data.* This includes its unique ID and a public key pair that are needed to set up connection with relay server. In the setup, the ID is determined and returned by Relay Server and then can be used for future connections when disconnected.

- *Basic information of PS's owners.* For each registered owner, it stores the network name, the email address and password which can be used by the owner to authenticate and log in locally. Moreover it needs to have the information for client agent to make connection with Relay Server. It contains the unique user identifier determined by Relay Server(during the first registration), public key pair for authentication with relay server in the future, and a symmetric key for encrypting requests from the agent to its own PS.

- *Social contacts of owners.* This involves three aspects of information: contacts, which includes the user and PS identifier of each contact, public key pair and a secret key for communication, groups which are labels that contacts can be assigned to, and membership where the mappings of contact and group is stored.

- *Data content of owners.* There are two types of data content: owner's profiles and owner's posts.

    1. *owner's profiles* Users can use some predefined attributes to describe themselves. Thus we use a pair of key and value(K, V) to define the attributes. For instance, there is a user named Alice, then the pair is (network name, Alice). The profile can have multiple records of the pairs. For each record, the user can specify which group of users can access. And group '0' means that the information is public to anyone. Each user can publish its public profile (containing records with group '0') onto open social platform which is introduced in Relay Server such that he/she can be searched by others.

    2. *owner's posts* Posts have two types: one is root message that is posted by the owner, and the other is comment posted by contacts of the owner whom have the access right to make comment on the root message. The common properties of these posts are that they have title, text content and media, the time when the post is made and the time when it expires. The difference

relies in the privilege format of the post. The owner of the root message can indicate the group number as the privilege information such that users of that group can access the root message. However, the group number doesn't work in the comments as the comment owner has no idea of the group number of the root owner. One neat turn-around is that the comment owner can attach to the comment with a bloom filter where efficiently stores information of users whom he/she authorizes to access. As a result, contacts who are both in the group of the root message and in the bloom filter can access the comment.

3. *owner's notifications* Notifications are meta data of contacts' updates, like whose update it is and when it happens. PS can help owners store notifications from their contacts when their client agent are offline.

Security Design To prevent outsiders from eavesdropping the communication between PS and relay server, each PS maintains SSL socket with RS. To prevent relay server from learning the data content from posts, posts are encrypted with secret keys when published and retrieved. Moreover to efficiently protect PS from replay attacks of posts retrieval and publishing by relay server(as these two types of messages are main ones in social networking), we propose that each initialization vector of the symmetric encryption(256-bits) over the request of posts(either publishing or retrieval) consists of two parts: the current timestamp of long type and 8 bytes of random data. When the PS retrieves such a kind of request from a specific client at first time (via RS), it records the timestamp in the request. The next time it again gets the request from the client, it checks if the timestamp in the current request is larger than the one in record, if yes, then updates the new timestamp with the old one in record and proceeds.

Normal Status In order to respond requests from client agents at any time, PS needs to keep running and keep connecting to Relay Server. When the connection interrupts unexpectedly, it needs to be reconnected.

4.3.2 Relay server

Intuitively, Relay Server(RS) plays an important role of bridge that connects client agents to PSes and PS to PS, as in most cases PSes are in private subnets behind firewalls. RS can help deliver requests from client agents to PSes and the other way back, and messages from one PS to another PS. To realize the goal, RS needs to keep track of the currently alive and secure connections from client agents and from PSes such that it can route messages correctly.

Data Model RS needs to store each PS's information that includes PS id and its public key and all registered users of each PS including user id and the public key and the user's PS.

Again we consider that SSL connections from PS keeps alive for a long time and those from client agents might be torn down intermittently. Each time the client agent or a registered PS tries to establish a SSL connection, RS uses the public key of the user to verify the identity. Besides routing messages, RS also provides an platform for strange users to get chance to know each other.

Open socialization platform (OSP) OSP can hold public profiles of users so that other users can retrieve them in order to make friends. When a user create profiles onto his/her own PS, the PS chooses the public items and post them to OSP. When a user wants to retrieve some profiles back with some preferences, OSP can help filter these profiles out and return to the user. More details are introduced in Section 4.3.

4.3.3 Client agent

By design, Client agent is supposed to work as a local mini-web server for browser. Due to time and work limit, we simplify it as an remote end that can send requests to and receive response from PS via RS. In order to fulfill the functionality, it should store some credentials locally and do some encryption over requests and decryption over responses. It always uses SSL socket to connect with RS.

4.4 Initialization

4.4.1 Initialization of relay server

Relay server starts running by listening on connections from client agents or personal servers.

4.4.2 Initialization of personal server

The initialization can be divided into two parts:

- *Owners(Users) registration on PS.* Before PS registers to RS, owners of PS can be enrolled locally into PS, by generating a public/private key pair which is used to verify owners to RS during connection to RS, and a secret key used to encrypt request/response during communication with PS.

- *PS registration on RS.* PS generates its own public/private key pair, and uses such key pair to be authenticated by RS, such that PS start to establish a long-time connection to RS. After that, PS registers all owners that have been enrolled on itself to RS. RS responds with a global unique user id for each owner.

4.4.3 Initialization of client agent

First of all, basic information including user id, public key pair and secret key needs to be extracted from PS and transferred via a secret channel onto the client agent. Then the client agent can use the user id and the public key pair to authenticate with RS that the user of the client agent is registered in order to make secure connection with RS and make requests to PS.

4.5 People Search

The design of profile in the form of key and value can facilitate fine grained people search. Users who want to search people with some specific value for an attribute can express the search query in the formula: "key = ? AND value = ?", and then client agent can submit the query to OSP. OSP can search the result based on the specification. This can be easily extended into complex logic expressions. For example, querying profiles one of which should satisfy several pairs of key and value restriction can be done by: first search out the profiles that satisfy the first requirement, and then for each profile from the result, iteratively check if it satisfies all the other requirements.

4.6 Friend Making

This work emphases on mutual friend relationship. The Procedure of making a friend can be illustrated in Figure 4.2. The setting is: user B likes to add user A as his contact after he reviews A's profile. All messages are first forwarded to RS and then forwarded by RS to the destination.

- B's client agent will send a request of adding A as a contact to PS B, the request contains user id and PS id of A;

- PS B will locally record the request for B and then prepare a formal friend request to PS A. The request has B's user id, PS id, public key, and a signature over user id and PS id.;
- Once PS A receive the request, first it verifies the signature, If the request is new and correct, PS A saves the request to wait for A's client agent to retrieve;
- Later on when A's client agent gets online, and retrieves the recent friend request from PS A, user A can make decision to accept or deny the request. If accepts, A's client agent signals PS A of the decision;
- After PS A receives the signal from client agent A, PS A creates a secret key for the new contact for future communication, and encrypts it with the public key of the new contact, and then delivers it to PS B;
- PS B first checks previous records to verify that client B has sent such a friend request, and use B's private key to decrypt the message, and then restore A's information alongside the secret key into contact list.

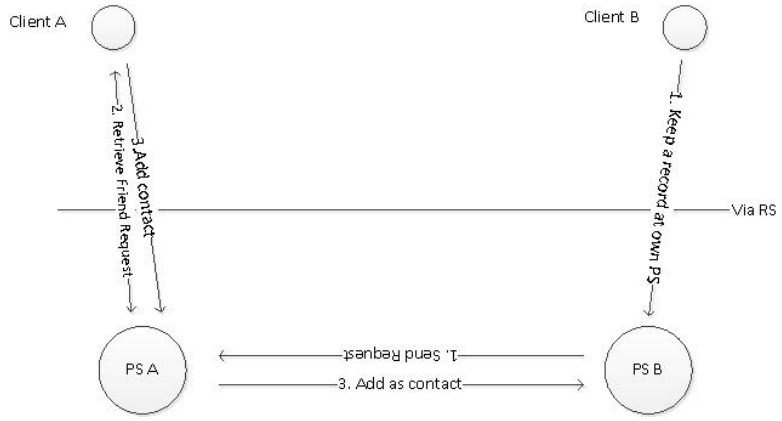

Figure 4.2: Making Friend Procedure

4.7 Contact Management

A user can create groups at will to manage their contacts. By default, each contact forms a group just with one member. A contact can be added into a group defined by the user and also can be removed from the group. A contact can belong to multiple groups. Access control of a message by the user is enforced on one group or multiple groups.

4.7.1 Adding a group

With the input of group name from the user, the client agent encrypts the request of adding group with the secret key shared with the user's PS and with the initialization vector mechanism, and sends the encrypted request to the PS via RS. PS first verifies the initialization vector and then deciphers the request, stores the new group information for the user, and returns the new generated group id (encrypted with the secret key) to the user via RS.

4.7.2 Managing a contact

The user specifies the contact id and group id along with the action(adding or removing), the client agent encrypts the request as above and sends to PS via RS. PS deciphers the request as above and proceeds correspondingly.

4.8 Message sharing

4.8.1 Building block

Bloom Filter Bloom filter [Bloom (1970)] is a memory-efficient data structure that stores a set S and supports membership queries. Let S be a set of size N and T be a bit set of size M ($N >> M$), then pick k hash functions uniformly at random, $h_1, h_2, ..., h_k$. To create the bloom filter in T for S, for each $x \in S$, all $T[h_1(x)]$, $T[h_2(x)]$, ..., $T[h_k(x)]$ are set to true; To check if y belongs to S or not, then check if $T[h_1(x)]$, $T[h_2(x)]$, ...,

$T[h_k(x)]$ are all true. If no, it's for sure that y is not in S. If yes, y is considered in S with high probability. The false positive can be proven to be $(1 - e^{-kN/M})^k$.

We utilize bloom filter in access control of comments because: (1) as comment owners have no idea of group settings, one naive way for comment owners is to specify a list of users that can access in comments, which is expensive in aspects of communication and storage overhead; (2) By choosing parameters M and k properly, we can make the false positive lower that fits in our application.

4.8.2 Posting a message

Message could be a root message by a owner or a comment message associated to a root message by a contact of the owner. When routed via RS, the message is enciphered with secret key (the one shared between the owner and the owner's PS or the one shared between two contacts). A message might contain a title, a text content and a media file(e.g, image or video), and the access right information. It could be any existing group id for a root message and a bloom filter which stores a set of user identifiers for a comment. Comment message should have reference of root message. After receiving and deciphering a post request, PS stores the post for the requester.

4.8.3 Retrieving a directory of meta data of messages

Considering that there are many messages that users are not interested but have large size of data, it is a very beneficial function for users to retrieve a directory of meta data of messages first. The meta data includes the title, the author, the time when the message created.

The requester at client agent needs to specify whose meta data of messages are queried. There are two cases. The requester can be the data owner or the contact of the data owner. If the user is the data owner, additional parameters are needed, query time range($start\_time$, end_time). The request is encrypted with the secret key shared with

the own PS and sent to the PS. PS discovers the directory query with the time range from its one of owners, and it searches the owner's posts (root messages by the owner and comments by contacts) whose creation times meet the requirement and encrypts them to return.

If it's the second case, as for the parameters to be transferred, it's not of necessity to indicate *end_time*, as the target PS regards timestamp of the retrieval of the request as *end_time*. *start_time* is optional as PS can use the last directory retrieval time of the contact if it's not indicated. The request is encrypted with the contact secret key and sent over to the target PS. The PS reveals the request, and figures out the query time range. PS filters out posts based on the following procedure:

- For each root message of the target owner, check if it satisfies the time range of the request and if its access group includes the requester identifier. If both are true, PS add the meta data of the message into result set;

- For each comment for the target owner, first check if it satisfies the time range and if the requester is included in the access group of the root message of the comment as well as in the bloom filter associated to the comment. If all are true,PS add the meta data of the message into result set.

4.8.4 Retrieving posts

Retrieving messages is similar to retrieving a directory of meta data, except for:

- The time range (starting time and end time) of the query creation time is set by the requester at the client agent;

- Content of posts are retrieved, including text content and media data, along with meta data.

4.9 Activity awareness

A notification contains three basic pieces of data: receiver id, sender id and notification creation time. A basic model of notification flow is depicted in Figure 4.3. When there is a notification from PS of owner A for user B to read, the notification is transferred to RS. If there is live connection from user B's client agent, RS forwards it to the client agent. Otherwise, RS sends it to PS of owner B to save it for user B. One question arises, i.e, when is notification generated? The easy solution is that each time the new post comes, for each member in the access group, PS generates the notification and sends out. But this would cause much network traffic. We improve it by recording the last notification time delivered to the contact along with last retrieval time of meta data directory of the contact. Before sending out notifications for each contact in the access group, PS compares the last notification time to the last retrieval time. If the retrieval time is older than the notification time, which means the contact didn't respond to the last notification, PS holds back the notification for the contact. Otherwise, PS sends out the newest notification and updates the last notification time in record.

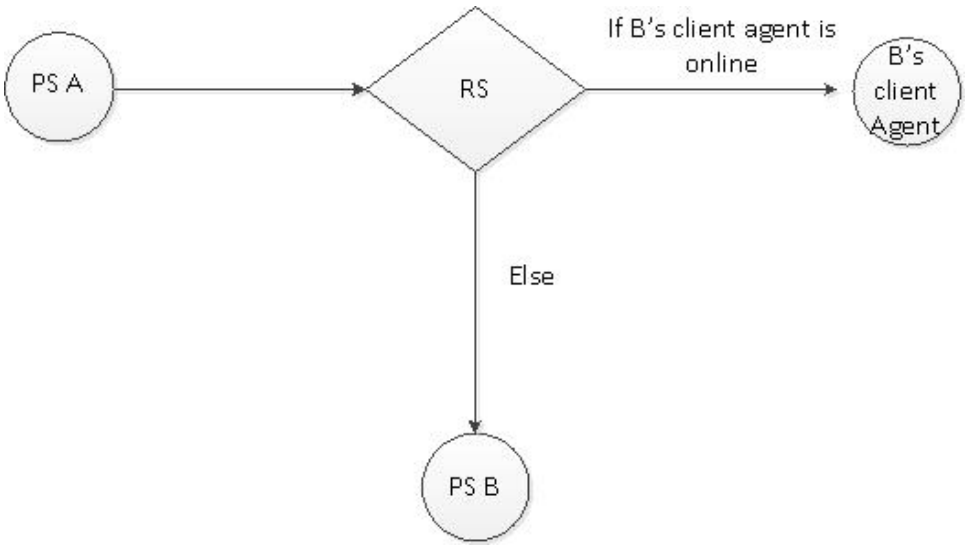

Figure 4.3: Notification Flow From PS of owner A to user B

CHAPTER 5. IMPLEMENTATION

5.1 Communication

5.1.1 SSL socket

SSL(Secure socket layer) is a cryptographic protocol for providing communication security over TCP connections between a server and a client. Not only can it protect data confidentiality and ensure data integrity, but also it provides peer authentication. This is done due to handshaking.

Overview The handshaking can be depicted in Figure 5.1. The handshake starts when client requests a secure connection and presents a list of cipher suites; After receiving the request, the server chooses a cipher and hash function that it supports to return to client alongside with server's certificate; Then the client verifies server's certificate and exchanges session key with server. The session key is usually generated by server and client separately with the same random number that is chosen by client and transferred to the server.

In our case, we regard RS as the server and PSes and CLes as clients in the client/server model. To be authenticated by clients, RS needs to maintain a certificate that can be uploaded and verified during the handshaking; we provide public key cryptography to RS to authenticate clients as it would be expensive for each client to apply a certificate. The implementation of creating a SSLSocket at client side in Java is as follows:

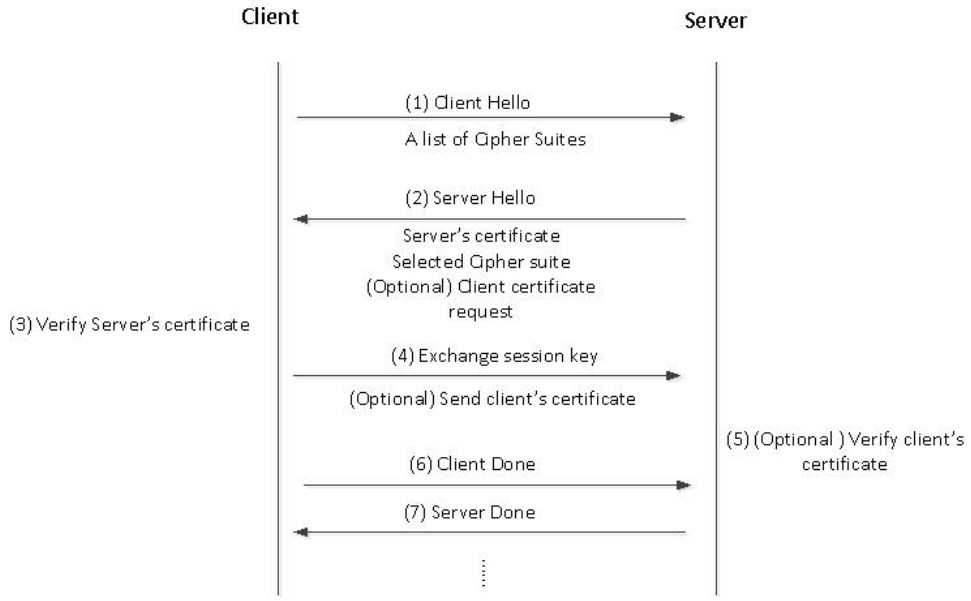

Figure 5.1: SSL Handshaking

- Create a TrustManager object $trustManager$ that can authenticate the remote server for the client;
- Create a SSLContext object $sslContext$ by calling $SSLContext.getInstance(\text{“}TLS\text{”})$ and initialize it with $trustManager$;
- Use $sslContext$ to create a SSLSocketFactory object $sslSF$ that is used to create a socket with the server by taking the server IP address and port number, and returns a SSLSocket object $sslSocket$;
- The handshaking is initialized by using $sslSocket.startHandShake()$. In this call, server authentication is done and cipher suites and session key are determined.
- Data exchanged via $sslSocket$ is encrypted and secure after handshaking.

The authentication of client is explained in the security section as it involves encryption. Initially the server should obtain a certificate from trusted authorities. For convenience, in our experiment we generate the certificate by using a key and certificate

management tool *keytool* in Java, which can manage certificate and public key pair into file *keystore.jks*. The steps involved in creating a SSLServerSocket at the server side in Java are as follows:

- Load *keystore.jks* file that contains private key and public key and certificate information into a KeyStore object *keyStore*;
- Create a KeyManagerFactory object *keyFac* initialized with *keyStore*.
- Use *keyFac* created the above step to initialize a SSLContext object *sslContext* of TLS;
- *sslContext* is used to create a SSLServerSocketFactory object that calls method createServerSocket() with parameter the port number to generate a SSLServerSocket object *sslServerSocket*;
- *sslServerSocket* can listen on the port and accept connection requests from clients.
- *sslContext* helps the server handshake with client in the backend.

5.1.2 Message types

As there are various types of messages that RS needs to route around, we describe the header format of messages as in Figure 5.2. We generalize five types of messages: Request, Response, FriendRequest, FrienConfirm and Notification.

1. Request is sent from client agent to personal server. The data part is encrypted parameters that is transparent to relay server. There are 12 types of requests: AddGroup, AddProfiles, AddContact, RetrieveFriendRequest, AcceptFriendRequest, ManageContact, PostMessage, RetrieveOwnMessage, RetrieveContactMsgDirectory, RetrieveContactMessage, CommentContactMessage and RetrieveNotifications. Personal server discovers the parameters in the encrypted data and proceeds correspondingly.

2. Response is sent from personal server to client agent. The data part is encrypted result that corresponds to a query from the client agent.

3. FriendRequest is sent from a personal server to another personal server. Based on our model of Friend making, the first personal server sends message FriendRequest on behalf of its owner. The second personal server is the one that the future contact owns. The data part contains a secret key encrypted with the public key of the contact, owner's profile encrypted with the secret key, and a signature over the encrypted profile. The signature can be used to verify that the owner of the profile is as declared. The FriendRequest is saved by the second personal server so that later it can be retrieved by the owner on the client agent.

4. FriendConfirm is sent in the reverse direction compared to FriendRequest. The data part in the message includes a secret key encrypted with the public key of the friend request initializer, that is the communication key between the pair of contacts, and a signature of the former part. The receiving personal server first verifies the signature and then decipher the secret key and keep it for the owner.

5. Notification is sent from the personal server when receiving a PostMessage request to a client agent (if it's online) or a personal server. The relay server makes the decision of whom to deliver based on the connection status of the client agent. The data part is simple just containing the timestamp of the message posted.

Figure 5.2: Message Header

5.2 Security

5.2.1 Encryption

In this thesis we employ two cryptographic methods symmetric encryption and public key encryption. Symmetric encryption is used in situations where data exchange between PS and client agents of its one owner or contact of one owner and where long messages need encryption, and public key encryption in situations where verification is in need and where communication occurs before two users become contacts, e.g, a user wants to add another as a contact.

Symmetric Encryption We use Advanced Encryption Standard (AES) encryption method and choose Cipher Block Chaining (CBC) as the mode of operation. The encryption needs a initialization vector (IV) besides a secret key. Both IV and secret key have 16 bytes. Before encryption takes place each time, IV is initialized to the concatenation of 8 bytes of current timestamp and 8 bytes of random data. This guarantees that IV has randomness as well as timestamp. The timestamp can protect attacks of replay efficiently as it can be compared to the last timestamp. The IV can be transferred with the encrypted data as it's secure and necessary for the other side to do the decryption.

The implementation of AES algorithm is based on package *javax.crypto* provided on Java. It implements various classes that supports cryptographic functions. These are important and useful classes:

- SecretKeySpec is the class that defines a secret key. It has a constructor that takes the bytes of secret key and the encryption method name as inputs.

- IVParameterSpec is the one that defines an IV. The constructor needs the input of bytes of an IV.

- Cipher is the class that defines the encryption method and mode. It can be initialized with an object of SecretKeySpec, one of IVParameterSpec and a static variable of Cipher that indicates if it's encryption or decryption.

- SealedObject can encapsulate the content of a serialized object with encryption algorithm. It can also retrieve the original object from encrypted object for decryption.

Public key encryption The public key encryption method has two keys, private key and public key. Private key is merely held by the key owner, and public key can be public to others. Both keys can be used upon encryption and decryption. When private key is applied in encryption, it's often called signature generation, and in this case public key is used to verify the signature. In the other case, when one party needs to verify that a user is the one that holds the public key, it can use public key to encrypt and make the user apply the corresponding private key to decipher.

We use RSA as the public key cryptosystem. Like symmetric encryption, the implementation of RSA algorithm relies on package *javax.crypto*.

- PrivateKey and PublicKey are classes that define private key and public key.

- Class Signature provides the functionality of a digital signature. Three phases are involved in signing a Signature object or verifying: Firstly, initialize a Signature Object with a PrivateKey object by calling $initSign()$ or with a PublicKey object by calling $initVerify()$; Then method of the signature $update()$ can be used to update bytes of data to be signed or verified; Last $sign()$ and $verify()$ can sign and verify the updated bytes respectively.

- Cipher class defines the encryption method. In RSA, a Cipher object can be initialized with a PublicKey (or PrivateKey) object and the static variable ENCRYPT_MODE or DECRYPT_MODE. When calling $doFinal()$ with the target bytes, it can encipher the bytes with the public key or decipher the bytes with the private key.

5.2.2 Verification process

This thesis considers verification process in cases where PS (or Client Agent) needs to reestablish a connection with RS. It can be explained in Figure 5.3. As RS has the public key of PS (or Client Agent), it can encrypts some data with the public key. The encrypted data can only be decrypted with the correspondingly right private key. In such way it can verify the requesting side in order to prevent unauthorized parties from connecting.

When the client side receives verification data, it utilizes the private key of the user to reveal the plain bytes. Then it generates 10 bytes of random data, concatenated with the revealed 10 bytes, and then encipher them all with the private key. RS use the public key to decipher the new combined data and compares the first 10 bytes data with the 10 bytes generated. If they match, RS can realize that the client is authenticated.

5.2.3 Access control

We have two mechanisms to enforce access control over messages. A root message have a specific group of whose members can access and a comment have a bloom filter that efficiently stores the authorized users information.

Group Membership The membership of a user who belongs to a group is saved into a MySQL database table named Membership during request ManageContact. When the contact comes to request posts, PS makes a JOIN SQL query of table posts and membership and contacts to check if access groups of posts include the contact.

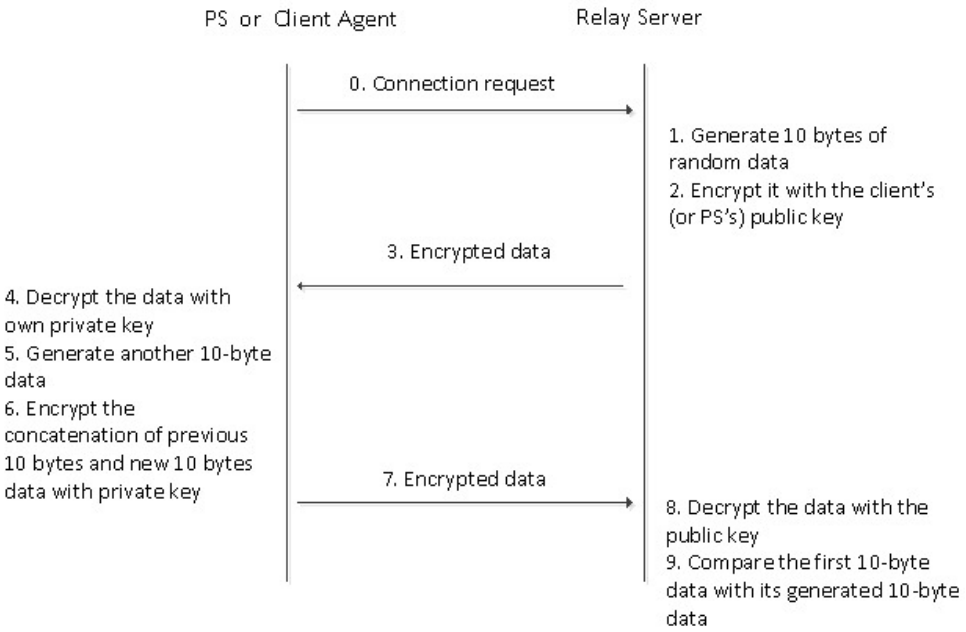

Figure 5.3: Verification Process

Bloom Filter In our implementation, we use BitSet to represent the data stored of the bloom filter. The false positive rate is $(1 - e^{-kN/M})^k$, where k is the number of hash functions, and N is the number of integers added into the bloom filter and M is the number of bits in the bloom filter . It can be shown that the probability is minimized when we take $e^{-kN/M} = 1/2$, then k, the number of hash functions, can be set up to be $\frac{M}{N}ln2$. Now the false positive rate is $\frac{1}{2^{\frac{M}{N}ln2}}$. According to Dunbar's number, that is, the number of social friends that a person can maintain is around 150. If we choose 10 bits for a number, the size of the BitSet is around 187 bytes.

Algorithm 1 getFNV64

```
Input: A string s;
Constant FNVPRIME = 1099511628211;
Constant FNVINT = 14695981039346656037;
digest = FNVINT.
for each i from 0 to s.size() − 1 do
    digest = digest XOR s[i];
    digest = (digest × FNVPRIME)  mod  2^64
end for
Output digest.
```

To implement k random hash functions, we use deterministic function FNV-64 hash function described in Pseudocode 1 and add some randomness in the hash function to get k different hash values. In Java, class Object has a method $hashCode()$. The formula(5.1) is to compute ith hash function. Bloom filter supports two methods.

$$h_i(s) = (getFNV64(s) + i \times s.hashCode()) >>> 1 \tag{5.1}$$

- *add(String s)*: This method is to add a string s into the bloom filter. For each i $(1 < i < k)$, it computes its the ith hash function $h_i(s)$based on the above formula, and then set the index $h_i(s)$ of BitSet to true.

- *appear(String s)*: This method is to check if the input string s is contained in the bloom filter or not. Like method $add(Strings)$, it computes $h_i(s)$ and check if the $h_i(s)$th index of the BitSet is true or false. That all $h_1(s)$, $h_2(s), ..., h_k(s)$ indexes of BitSet are true concludes that s is in the set with large probability.

5.3 Personal Server

5.3.1 Overview of initialization

When PS starts, it makes a SSL connection request to RS and proceeds handshaking with RS. If it passes, PS checks if this is the first time connecting to RS , if yes, PS

registers itself and all its owners onto RS. Otherwise, it's its turn to be authenticated by RS via verification process. After registration or successful verification, PS is ready to wait on the established socket for requests from others. When there comes a request, PS main thread spawns a new thread to handle it and then comes back to wait on the socket again. The steps can be illustrated in Figure 5.4.

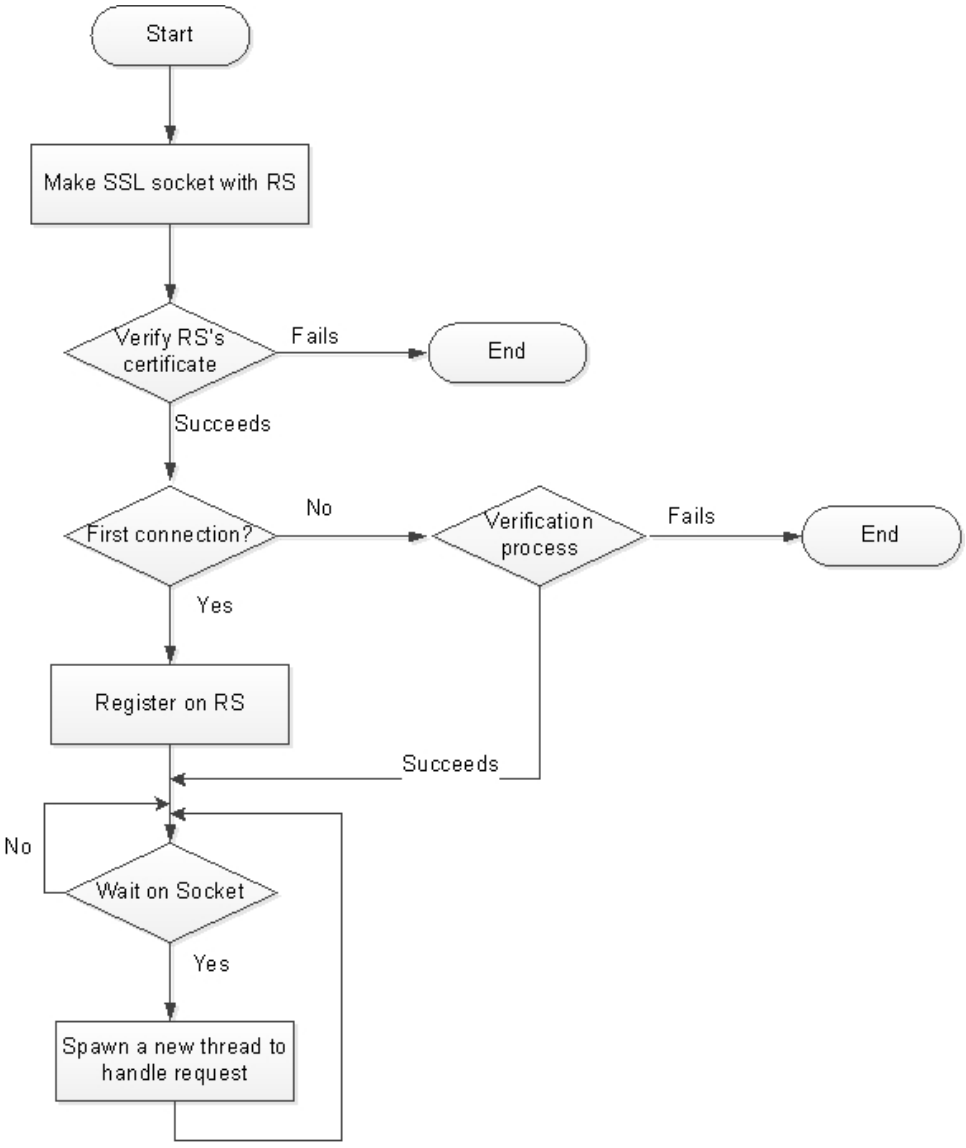

Figure 5.4: Flow chart of PS initialization

5.3.2 Request handling

As indicated in Figure 5.4, a thread of handling request is created when the request comes. In Table 5.1, requests that can be handled by PS are categorized based on request sent by whom: owner client agent, or contact client agent or contact personal server. PS can recognize request because each request has different request type id.

Table 5.1: Summary of requests handled by PS

Requester	Actions
Owner client agents	EditGroup, EditProfile, AddContact, RetrieveContactRequest, AcceptContact, ManageContact, PostMessage, CommentMessage
Contact client agents	RetrieveMessageDirectory, RetrieveMessage, CommentMessage
Contact PS	SendContactRequest, SendContactConfirm, SendNotification

5.3.3 Profile definition

Profile is itemized into (key, value) pairs that can facilitate users to search and come to know people. This thesis currently allows the following keys: 0 (client name), 1 (gender), 2 (birth year), 3 (birth month), 4 (birth day), 5 (hobby), 6 (profession), 7 (college). Keys can be extended in future and kept known to all clients.

In order to enforce fine-grained access control over profile, each item is associated with an access group id so that members of that group can access the key and value. Id 0 is reserved for public sharing.

5.3.4 Data structure

We use MySQL database to store data. We group data in PS into three categories: basic information about owners, contact management and posts of owners. All tables are shown in Figure 5.5.

- Basic information includes information of owners and owners' profiles.
- Contact management includes contact information and groups defined by owners and membership.
- Posts contains root messages of owners and comments from contacts, and their access privileges respectively. Media from root messages and from comments are stored together in table Media.

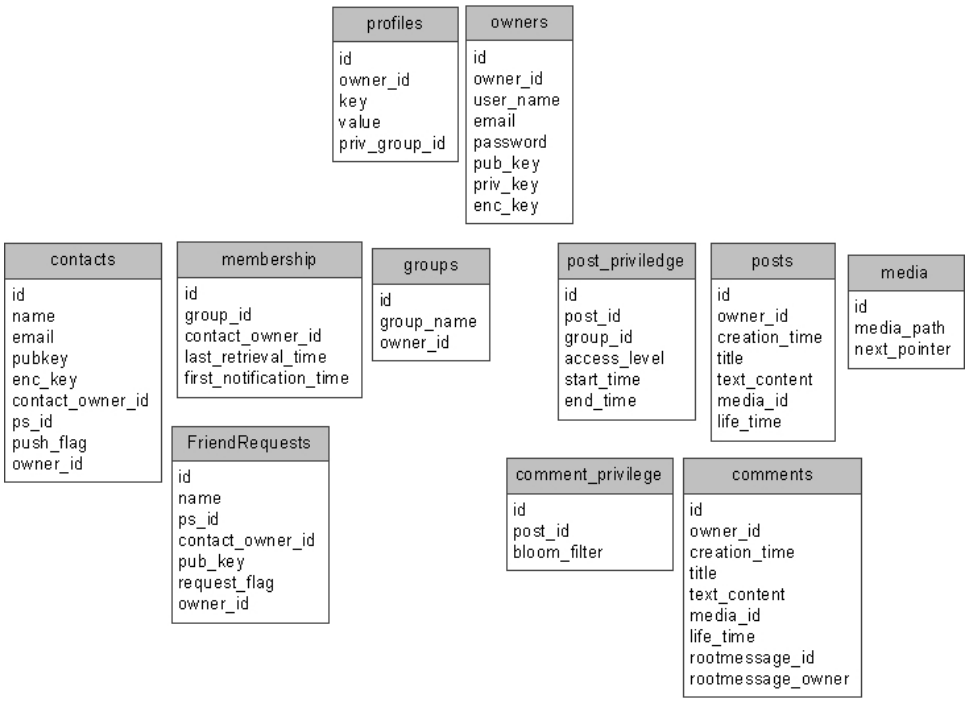

Figure 5.5: Database tables at Personal Server

We use SQL to create tables in MySQL database. Below is an example of creating table *Owners*.

```
CREATE TABLE ‘profiles‘ (
  ‘id‘ INT NOT NULL AUTO_INCREMENT,
  ‘owner_id‘ INT NOT NULL,
  ‘key_id‘ INT NOT NULL,
  ‘value‘ VARCHAR(60) NOT NULL,
  ‘group_id‘ INT NOT NULL,
  PRIMARY KEY (‘id‘)
 )
```

Query PS connects to MySQL database when storing/retrieving data is required. We use PreparedStatement in Java to represent SQL statements. A SQL statement with unknown parameters can be precompiled and stored in a PreparedStatement object. Later values of parameters can be set up into the statement by using setter methods of PreparedStatement on the condition that setters should specify the compatible types with ones defined in the statement. Below is an example of using PreparedStatement to execute SQL query of selecting private key and encryption key from owners with a specific *owner_id*.

```
String query = ‘‘SELECT privkey, enc_key FROM owners WHERE user_id = ?";
/* UserInfo is a class to define a owner  */
UserInfo usr = (UserInfo) input;
/* Connection dbConn is an active connection */
PreparedStatement preStatement = dbConn.prepareStatement(query);
preStatement.setInt(1, usr.getClientID());
ResultSet result = preStatement.executeQuery();
```

5.4 Relay Server

When Relay Server starts running, the main process creates four threads, each of which listens on a port and waits for requests from others. The first thread listens

on port 9780 for connections from client agents to transmit messages; the second on port 9781 for connections from personal servers to registration and deliver messages; the third on port 9782 for connections from personal servers to publish profiles on open socialization platform and the last one on port 9783 for request of profiles from client agents.

5.4.1 Data structure

Hash table Because of the role of routing messages between client agents and personal servers, relay server needs to memorize somehow output stream of each socket connection with client agent or personal server in order to search quickly. We use two hash tables to store them respectively, one for storing key (client id) and value (its output stream) , the other for key (personal server id) and value (its output stream). As Hashtable in Java is thread-safe, it allows two hash tables to be shared by the first two threads. When a connection with port 9780 from a client agent is established, the first thread stores client id and the output stream of the socket connection into the first hash table; when the connection gets lost, the pair of the key and value is removed from the hash table.

When there is a request message from a client agent to a personal server, the relay server can efficiently find out the output stream of the personal server in the hash table.

Database Tables To allow personal servers and client agents to make connections again after connection failure, the relay server should store information of registered personal servers and of clients (owners of personal servers) so that it can retrieve information in database to verify the requester, e.g, public key of the requester. On the other hand it needs to store public profiles from clients as it's open socialization platform. All tables in relay server are defined in Figure 5.6.

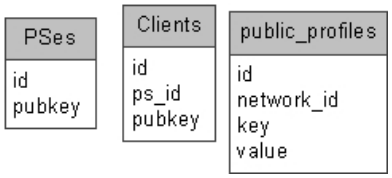

Figure 5.6: Database tables at Relay Server

CHAPTER 6. EVALUATION

In this section, we report some evaluations of the implemented social network prototype. We focus on evaluating the response time elapsed from when a client agent sends out a request to when it receives the response in four different scenarios, as detailed below. The results show that the prototype has good performance in terms of social functions.

6.1 Scenario One

It measures the response time of querying messages by a contact in different cases where the owner's "posts" table size varies. This scenario involves two personal servers, two client agents and one relay server.

6.1.1 Settings

Environment setting The relay server runs at an EC2 machine. We put client agent into the same machine of the personal server. A MacBook Pro laptop plays one personal server and its client agent, and a virtual machine created by VirtualBox in the same MacBook Pro, plays the other pair of personal server and client agent. The specific configurations of these three machines in terms of Operating System, CPU, Memory and Storage Space, are shown in Table 6.1.

Parameter setting The number of posts in database varies: 200, 400, 800 and 1600. There are 100 posts that satisfy a query from the client agent, and each post

Table 6.1: Configuration details

EC2 machine	MacBook Pro	Virtual Machine
Ubuntu-14.04-LTS	OS X 10.11.1	Ubuntu-14.04-LTS
2.5 GHz Intel Xeon	2.5 GHz Intel i5 with 1 processor	2.5 GHz Intel i5 with 1 processor
1GB	16 GB	2 GB
10 GB	240 GB	10 GB

with 200 bytes of text. For each number setting, the client agent conducts 100 times of queries. The response time is calculated as the average value of all query time intervals.

6.1.2 Result

The result is shown in Figure 6.1. The X-axis represents the changes in the table size in the units of number of records, and the Y-axis shows the response time in units of millisecond. The values from 200, 400, 800, 1600 forms are around 160 ms, forming the trend that the change of database size doesn't affect the response time very much.

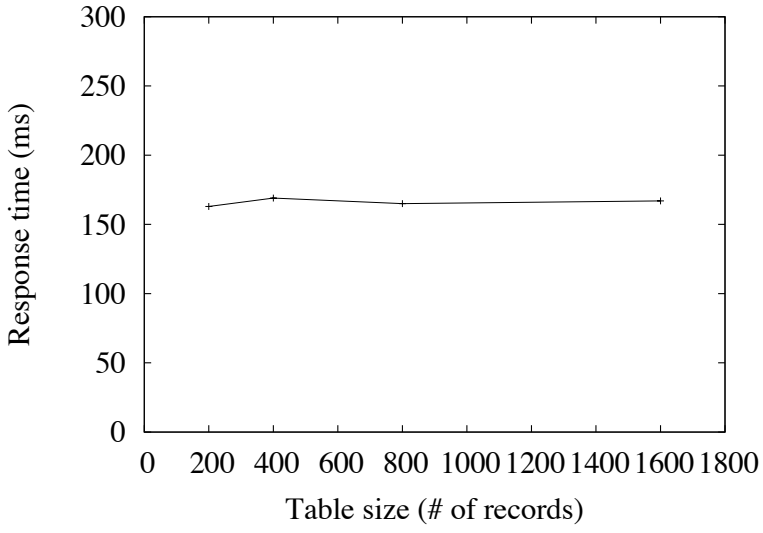

Figure 6.1: Response time of querying posts with different table sizes

6.2 Scenario Two

It measures the response time of querying messages by a contact in different situations where these messages have different sizes at owner's personal server. As in the Scenario One, it involves two personal servers and two client agents and one relay server.

6.2.1 Settings

Environment setting It's the same in the Environment setting 6.1.1.

Parameter setting The size of each message retrieved varies from 1KB, 10KB, 100KB and 1MB. The size of table "posts" at owner's PS side is fixed with 400 records. There are 10 messages satisfying a query from client agent of the contact. For each situation, the client agent conducts 100 times of queries sequentially after it gets the result from the previous query. The response time is equal to the average time of all query time intervals.

6.2.2 Result

The result is shown in Figure 6.2. The X-axis means the increasing values of post size in unit of KB and Y-axis the response time in unit of millisecond in the log scale. It shows that when post size is relatively small, like less than 200 KB, the response time is less than 300 milliseconds, but when the size arises to 1MB, it needs several seconds to go through the network traffic.

6.3 Scenario Three

This scenario takes the notification scheme into consideration. The basic model is that an owner client publishes a message onto the personal server, and the nearby online contact, who satisfies the access control policy, tries to fetch it after receiving

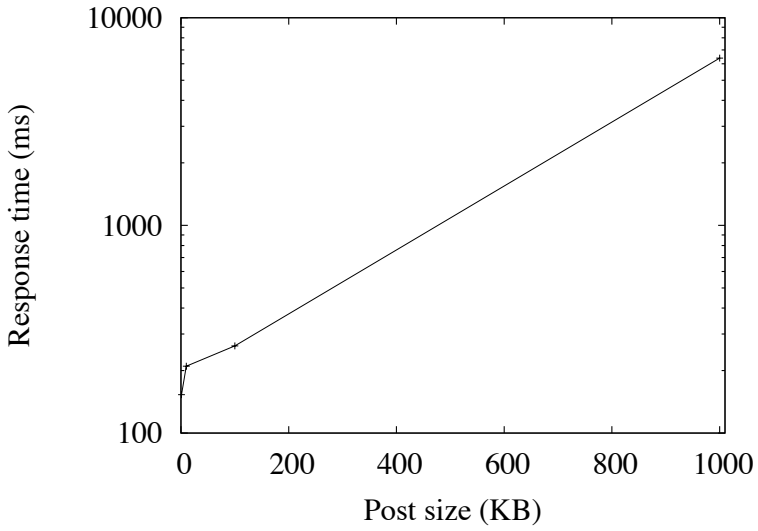

Figure 6.2: Response time for queries with different post sizes

the notification of the new message. The time interval is measured from the time when the owner acts the posting in the client agent to the time when the contact receives the message. As in the Scenario One, it involves two personal servers and two client agents and one relay server.

6.3.1 Settings

Environment setting This is the same as the setting 6.1.1.

Parameter setting The owner client posts a 10KB message onto the personal server with access setting that contains the contact nearby, which keeps an alive connection with relay server. The personal server sends out a notification to the contact. Contact receives it and then sends a message query to the owner's personal server and retrieve the message. This experiment is done 100 times.

6.3.2 Result

The average time interval for the whole process is around 700 milliseconds. This result is positive to make instant chat message possible.

6.4 Scenario Four

We measure the response time in the situations where three contacts simultaneously query messages of a owner at one personal server. These three users are continuously sending queries with a certain query frequency. In this scenario, four personal servers and four client agents and one relay server are involved.

6.4.1 Settings

Environment setting In this experiment, we add two more virtual machines with the same configuration of the virtual machine in 6.1.1, and other settings remain the same. One personal server plays where messages of the owner are stored, and three other client agents play where these of contacts send queries.

Parameter setting Each contact client has 100 queries for the target personal server. The table "posts" has 400 records, and each post is 200 bytes. For each query, there are 10 posts satisfied and returned. We change the time interval of these queries at each contact client agent while time intervals at these agents are equal. The time intervals are 1 second, 2 seconds, 4 seconds and 8 seconds. The response time for each client agent is calculated as the average value among all query response times.

6.4.2 Result

The result is shown in Figure 6.3. The X-axis represents the query intervals in unit of seconds, and Y-axis represents response time in unit of milliseconds with log scale. From

this output, we can find that when the personal server serves many client agents, the response time is approximately 130 milliseconds for each client. And the query interval starting from 1 second doesn't affect the response time very much.

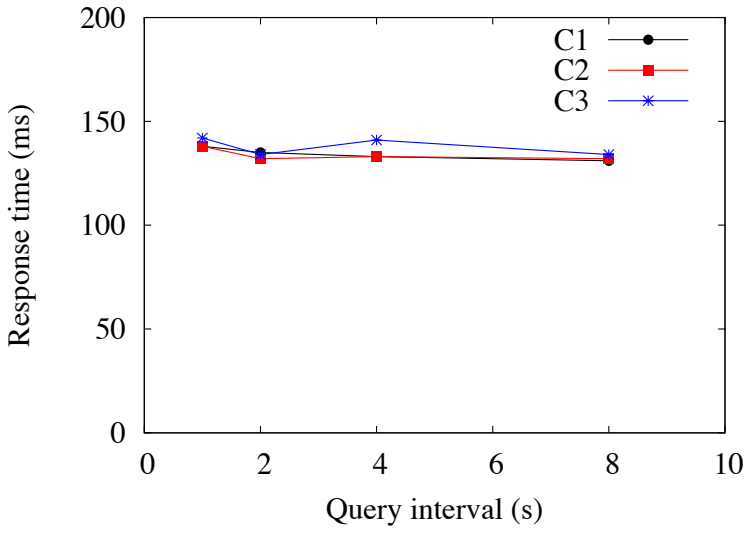

Figure 6.3: Response time of different query intervals

6.5 Security Analysis

We can conclude that our system is secure against threats in Threat Model if the underlying encryption mechanisms SSL and secret encryption are secure.

Let's first look at outside adversaries who want to eavesdrop communication between PSes or tamper communication data. They don't have capabilities to do so as all of communication channel between PS and RS are deployed with SSL. If SSL is secure against these attacks, then outside adversaries can't eavesdrop communication, not even tamper communication data.

Bad insiders might contain two parties, one is the honest but curious relay server and the other is any other user. In our system the relay server can't learn any information about messages posted by users and shared among friends of users, yet it can know the types of messages which it multiplexes, as we can see that messages transmitted in the procedures of posting and retrieving are encrypted under symmetric encryption. If the encryption is secure, then messages won't be revealed to the relay server.

Now consider a bad user *Alice* who wishes to access data of user *Bob*. When *Alice* is a friend of *Bob*, it's inevitable for *Alice* to access friends' data which she is able to. But when *Alice* is not a friend, she can't access any *Bob*'s messages which are enforced by access control policy. Neither does she learn any communication between *Bob* and the relay server, as SSL is deployed.

CHAPTER 7. CONCLUSION

Online social networking services have been very popular in recent years. More and more people spend a lot of time on posting data onto these service providers and receiving information from them. Data privacy has become a serious concern. Decentralized social network is a promising solution to the online central social networking systems. However, many existing works employ heavy encryption operations or construct systems without basic social networking functions. These would impede practical application in real life.

In this thesis we propose a new framework of decentralized social network. In order to protect servers where personal data is stored from outside attacks to some degree, and to make data accessible to users anywhere, we design three components, personal server hiding behind a firewall, client agent and relay server. The relay server is the place where client agent contacts in order to access data in personal server. Each personal server keeps a long-term connection with the relay server. Data transmitted is encrypted to protect the relay server from learning.

We design protocols to realize basic social networking function, which includes, making friends in the system, managing friends, posting messages, retrieving messages, and so on. We implement a prototype system. Evaluations have been conducted to test the performance of social networking functions.

BIBLIOGRAPHY

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. In *Communications of the ACM, 13(7)*, pages 422–426. ACM.

Buchegger, Sonja, e. a. (2009). Peerson: P2p social networking: early experiences and insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, pages 46–52. ACM.

Cutillo, Leucio Antonio, R. M. and Strufe, T. (2009). Safebook: Feasibility of transitive cooperation for privacy on a decentralized social network. In *World of Wireless, Mobile and Multimedia Networks & Workshops*, pages 1–6. IEEE International Symposium on a. IEEE.

Diaspora (2010). https://diasporafoundation.org/. In *Diaspora*.

Jahid, Sonia, e. a. (2012). Decent: A decentralized architecture for enforcing privacy in online social networks. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference*, pages 326–332. IEEE.

Nilizadeh, Shirin, e. a. (2012). Cachet: a decentralized architecture for privacy preserving social networking with caching. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 337–348. ACM.

Richter, A. and Koch, M. (2008). Functions of social networking services. In *Proc. Intl. Conf. on the Design of Cooperative Systems*, pages 87–98.

Seong, Seok-Won, e. a. (2010). Prpl: a decentralized social networking infrastructure. In *In Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, page 8. ACM.

Shakimov, Amre, e. a. (2009). Privacy, cost, and availability tradeoffs in decentralized osns. In *Proceedings of the 2nd ACM workshop on Online social networks*, pages 13–18. ACM.

Shakimov, Amre, e. a. (2011). Vis-a-vis: Privacy-preserving online social networking via virtual individual servers. In *In Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, pages 1–10. 2011 Third International Conference on. IEEE.

Stoica, Ion, e. a. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM Computer Communication Review 31.4*, pages 149–160. ACM.