

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Fakultät für Elektrotechnik, Informatik und Mathematik

Masterthesis

Systematic Evaluation of a Decentralized Social Network

Dheeraj Balabhadruni
Weifeng Peng

Student ID:
6644484
6638258

E-Mail:
dheeraj@uni-paderborn.de
pengweifeng21@gmail.com

Paderborn, December 22, 2014

supervised by
Jun.-Prof. Dr.-Ing. Kalman Graffi
Prof. Dr. Christian Scheideler

Declaration

I hereby declare that this thesis incorporates the result of a joint research undertaken in collaboration with Dheeraj Balabhadruni and Weifeng Peng under the supervision of junior professor Kalman Graffi. The collaboration and individual work is covered in Chapter 1 of this thesis. In table of contents the chapters, sections and subsections coloured in blue are written by author Dheeraj Balabhadruni and similarly dark gray coloured contents are written by Weifeng Peng. We have not used outside sources without declaration in the text. Any concepts or quotations applicable to these sources are clearly attributed to them. This thesis has not been submitted in the same or substantially similar version, not even in part, to any other authority for grading and has not been published elsewhere.

Paderborn, December 22, 2014

Dheeraj Balabhadruni
Weifeng Peng

Contents

1 Intoduction **1**
- 1.1 Motivation . . . 1
- 1.2 Goals . . . 2
  - 1.2.1 Completing Test Cases . . . 2
  - 1.2.2 Synthesis test . . . 2
  - 1.2.3 Random test . . . 2
  - 1.2.4 Random test without churn . . . 2
  - 1.2.5 Long term test . . . 2
  - 1.2.6 Evaluation of test results . . . 2
- 1.3 Thesis Structure . . . 3
- 1.4 Collaborative Work . . . 4

2 Background **5**
- 2.1 Peer-to-Peer Technology . . . 5
- 2.2 OSGi Framework . . . 6
- 2.3 LifeSocial . . . 7
  - 2.3.1 Peer-to-Peer Overlay Layer . . . 7
  - 2.3.2 Peer-to-Peer Framework layer . . . 9
  - 2.3.3 Plugins and Apps Layer . . . 9
  - 2.3.4 GUI Layer . . . 14
- 2.4 Basic Definitions . . . 14
- 2.5 Secure Shell . . . 16

3 Testing **19**
- 3.1 Test Plugin . . . 19
  - 3.1.1 Prerequisites . . . 19
  - 3.1.2 Requirements . . . 20
  - 3.1.3 Test Case Structure . . . 22
  - 3.1.4 Test Case Implementation . . . 26
- 3.2 Example Test Case . . . 28

The chapters, sections and subsections coloured in:
blue are written by Dheeraj Balabhadruni
dark gray are written by Weifeng Peng

3.2.1 Implementing a Precondition . . . 30
3.2.2 Implementing OSGi Command . . . 30
3.2.3 Defining OSGi Declarative Services . . . 31
3.2.4 Define bundle with component metadata . . . 33
3.2.5 Reference all Plugin Services . . . 33
3.3 Test Cases . . . 34
3.3.1 List of Test Cases . . . 35
3.3.2 Further Improvements . . . 36
3.4 Metrics . . . 42
3.4.1 New Metrics . . . 44
3.4.2 Further Improvements . . . 46
3.5 Setup Parameters . . . 48
3.5.1 Environment Parameters . . . 49
3.5.2 Social Parameters . . . 50
3.5.3 LifeSocial Parameters . . . 50

4 Monitoring **55**
4.1 Monitoring . . . 55
4.2 SkyEye.Kom . . . 55

5 Synthesis Test **59**
5.1 Test Environment . . . 59
5.2 Deployment . . . 59
5.3 Test Plan . . . 62
5.4 Monitoring Data Processing . . . 66
5.5 First Round Result Evaluation . . . 66
5.6 Modifications . . . 82
5.7 Second Round Result Evaluation . . . 82
5.8 Conclusion . . . 86

6 Random Test **87**
6.1 Test Environment . . . 87
6.2 Deployment . . . 87
6.3 Start Script . . . 88
6.4 Random Selection . . . 90
6.5 Monitoring Data Processing . . . 92
6.6 First Round Result Evaluation . . . 92
6.6.1 Random Test with Churn . . . 92
6.6.2 Random Test without Churn . . . 97
6.7 Modifications . . . 102
6.8 Second Round Result Evaluation . . . 102
6.8.1 Random Test with Churn . . . 103

6.8.2 Random Test without Churn 105
6.9 Conclusion 105

7 Long Term Test **107**
7.1 Test Environment 107
7.2 Deployment 107
7.3 Test Plan 114
7.4 Monitoring and Data Processing 116
7.5 Issues in Long Term Test 116
7.6 Evaluation 118
7.7 Conclusion 121

8 Conclusion **123**
8.1 Achievement 123
8.2 Future Work 124

Bibliography **135**

1 Intoduction

1.1 Motivation

Social network has become an indispensable part in many people's life, especially young people. Accessing Facebook to check friend's status, tweeting when watching a football game, and pressing `Like` button for a picture of a friend on Instagram, all these actions happen in our day to day life. Currently, majority of social networks are centralized and based on Client-Server architecture, which means the companies of these centralized social networks have to spend huge amount of money on server maintenance, and also, users of these centralized social networks have to worry about their private information on the servers.

Peer-to-peer (P2P) technology provides the possibility to solve these potential problems initially. In P2P network, there is no central server and each peer acts as both client and server, shares their CPU power, storage capacity and bandwidth. Data in P2P network is distributed on peers all over the network instead of central server. Therefore, social network company can save a great deal of money on server maintenance and users of P2P network can feel relieved about the security of their privacy. Moreover, P2P technology can keep the network from single point of failure since data in the network is decentralized and no particular peer is absolutely necessary in theory.

`LifeSocial` is a relatively new social application based on P2P technology and it contains most of the common features provided by current social applications/platforms, for instance, chat with friend, viewing friend's photo and commenting on friend's wall post. `LifeSocial` is a joint project developed by University of Paderborn and Technische Universitat Darmstadt. Apparently, for a developing application, it is necessary to test to find out its potential bugs and defects.

1.2 Goals

The main goal of this thesis is to benchmark a Peer-to-Peer framework for social networks. To achieve the main goal the following sub-goals must be accomplished.

1.2.1 Completing Test Cases

Before this thesis, there are 26 test cases in the `LifeSocial` Test Plugin but 7 of these test cases are invalid for some reason. These invalid test cases need to be modified according to the corresponding functionalities. Also, new test cases are needed to be created to test the missing features. Meanwhile, new metric for monitoring needs to be created when it is required.

1.2.2 Synthesis test

In synthesis test, all slaves in the network execute the same test plan at the same time. It can be examined if each functionality of `LifeSocial` is working correctly by evaluating monitoring data after the testing.

1.2.3 Random test

There is no fixed test plan for each slave to execute. This test aims at imitating real user behaviors that user will log out the network after a certain duration he/she logs in. The monitoring data of the test would be helpful to determine if `LifeSocial` is robust when churn exists.

1.2.4 Random test without churn

The only difference from the random test is that there is no "churn" in this test. All users in the network execute test cases and do not log out. The comparison between monitoring data of this test and previous test would be helpful to verify certain functionality.

1.2.5 Long term test

The goal of this test is to maintain the network for long time, check the performance and reliability of the `LifeSocial` application under the specified environmental conditions.

1.2.6 Evaluation of test results

After performing all the tests, the monitoring data is collected from all the instances to plot the graphs for results analyzing and evaluation.

1.3 Thesis Structure

In this section the outline of the this thesis is presented along with the individual and combined contribution of the authors work.

Chapter 1

This chapter presents the motivation, goals and structure of the thesis. This entire chapter is written by the author D.Balabhadruni.

Chapter 2

This chapter presents a theoretical knowledge on Peer-to-Peer technology, `LifeSocial` architecture, OSGi Framework, Secure Shell (SSH) and basic definitions to put the reader in right context. `LifeSocial` is based on a layered architecture which contains P2P overlay, P2P framework, plugins and app, and GUI. All these layers are explained in this chapter to give an overview to the reader. This entire chapter is written by the author D.Balabhadruni.

Chapter 3

This chapter introduces the `TestPlugin` to test the performance and reliablity of the `LifeSocial` application. The first part of this chapter explains the prerequisites and requirements of the `TestPlugin`, test case structure and test case implementation with an example. Later in second part, the list of the implemented test cases, metrics and parameters of `LifeSocial` application are explained. This chapter is written by both the authors. The first part is written by D.Balabhadruni and the second part by W.Peng.

Chapter 4

This chapter introduces the monitoring mechanism called *SkyEye.Kom* to measure the performance of `LifeSocial` application by observing the behavior of peer-to-peer network. It is a tree based monitoring mechanism that is responsible for monitoring and gathering all information of whole network. This entire chapter is written by the author D.Balabhadruni.

Chapter 5

The synthesis test is explained in this chapter. The first part of this chapter explains the test environment, test deployment, test plan and monitoring data processing is explained. Later in second part, the results of the synthesis test

with 10 and 20 instances are evaluated and explained. This entire chapter is written by the author W.Peng.

Chapter 6

The random test is explained in this chapter. The first part of this chapter explains the test environment, test deployment, test plan and monitoring data processing is explained. Later in second part, the results of the random test with 10 and 20 instances are evaluated and explained. This entire chapter is written by the author W.Peng.

Chapter 7

This chapter explains the long term test. At first the test environment, test deployment, test plan and monitoring data processing is explained. Later in second part, the results of the long term test with 10 instances are evaluated and explained. This entire chapter is written by the author D.Balabhadruni.

Chapter 8

This chapter summarizes the entire thesis along with the future work. In addition, it also presents the future work. This chapter is written by both the authors D.Balabhadruni and W.Peng.

1.4 Collaborative Work

This thesis is a combined work of authors D.Balabhadruni and W.Peng. Actually, each author got an individual thesis topic at first and the aim of both the thesis is to test the performance of LifeSocial under different conditions with synthesis test, random test and long term test. Since there are many common parts on both of the thesis like creating test cases and metrics, collaborative work is good option for achieving overall evaluation.

During the period of this master thesis, many test cases were designed by both of the authors to cover the functionalities of the LifeSocial. Next, the first round of synthesis test was designed and performed by both of the authors. After that, both authors worked on their independent tasks. Author D.Balabhadruni worked on the long term test and author W.Peng worked on further synthesis test, random test and improvement of test cases. In table of contents the chapters, sections and subsections coloured in blue are written by author D.Balabhadruni and similarly dark gray coloured contents are written by W.Peng.

2 Background

This chapter presents some backgrounds to put the reader in the right context. First, we give an idea about the Peer-to-Peer technology, then we explained about the OSGi framework, `LifeSocial` architecture, available plugins in the application and basic definitions. This entire chapter is written by the author D.Balabhadruni.

2.1 Peer-to-Peer Technology

Peer-to-Peer (P2P) networking is a distributed system in which peers are connected to each other and share the resources. Unlike to traditional Client-Server system, this methodology does not require any centralized servers or hosts for requesting a resource. Peers store data in a distributed manner. Each peer or node can communicate with another node and every node acts as both supplier and consumers of resources. The number of users use P2P and amount of storage, bandwidth are always proportional. Some of the applications of P2P are content delivery, multimedia, file sharing networks etc.

Other side, the trend of social networking is growing tremendously day by day. There are some margins exist in the current social networking systems as they use client-server approach for data sharing. The major issues are expanding with network traffic, server maintenance cost, data privacy, single point of failure etc. These limitations can be avoided using P2P approach.

There are also few other limitations may be introduced by P2P. While servers require high bandwidth to maintain the data and to stay online for longer periods, peers are not such prevailing to stay in network and may leave at any time. The data sharing, stack of data between the peers should be impartial and there should not be any data dependency between the peers. This can be overcome by data replication concept which is nothing but copying and maintaining data in multiple peers in the system and changes made to data are stored locally before sharing to remote places. This also eliminates loss of data.

2.2 OSGi Framework

OSGi framework is an open source specification which describes modules and service models for Java. The components and services can be altered dynamically. OSGi frameworks deals with the lack of support for modularity in Java by establishing SOA, service oriented programming model to differentiate interface from execution. Modularity is nothing but separating the massive structure into minor elements logically.

Java has few restrictions with modularity, few of them are low level code visibility control by combining the distinct classes into package and by using multiple packages, error level class path model such as code versions and no specific or defined version of class path to add into library or jar file, and limited support during release and maintenance. These limitations can be overcome by considering an appropriate modularity layer where OSGi comes into the scenario.

OSGi framework is a major part as the framework will be the execution location for applications. The requirements also facilitate the design of numerous implementations of the interior framework. Few of the popular open source projects available are Apache Felix, Eclipse Equinox, and Knopflerfish etc. It is clear that the program is defined according to the specifications and not with respect to vendor, i.e. unique for every vendor.

The flexibility of OSGi framework is classified by three theoretical layers called Module layer, Lifecycle layer and Service layer. Similar to classic models, each layer is dependent on other layers. Below are the characteristic details of each layer of the structural design.

- ***Module Layer:*** The module layer describes about binding and sharing the code by using an OSGi module concept called *Bundle* [4]. A Bundle in OSGi framework defines the collection of class files, resource files and their Meta data. Meta data is the one which separates the bundle from normal JAR file, and this is not wrapping up of the application into a package but uniting the logical modules to form an application. Bundles are more than the general access modifiers of Java. The bundles can be declared explicitly by which, the framework can handle and authenticate their stability. This process is called *bundle resolution*.
- ***Lifecycle Layer:*** The lifecycle layer describes how bundles are administrated in the framework. Lifecycle layer provides two functionalities i.e, both externally and internally to the application. Interior to the system,

the layer identify about accessing bundles, how the communication happen with the framework and services present while execution. Externally, the layer handles and develops the application in a definite way. The layer takes care of installation, update, start, stop and uninstallation of bundles. The application does not require restart for adding up and taking out the bundles.

- ***Service Layer:*** The service layer works on service oriented architecture or interaction model by differentiating the interface from implementation. Service layer provides communication between service providers and service clients. Providers publish services into service registry where requesters can find for the services that are available for use. Services can become visible or vanish at any time.

2.3 LifeSocial

This section presents brief overview of the `LifeSocial` application. It is a P2P-based online social network jointly developed by University of Paderborn and Technische Universitat Darmstadt. In `LifeSocial`, there are no central instances and it is based on distributed systems. It is based on the layered architecture and is developed on the top of the P2P framework. In `LifeSocial`, user-to-user communication is very secure because it provides security mechanisms like authentication, confidentiality, encryption and access control on distributedly stored data. Like other social networks, it also provides the basic functionalities like creating profiles, sending friend request, uploading and sharing photos, sending messages, creating events and many more. Eclipse plugin platform is used to implement these features, and all the plugins are based on OSGi framework. Therefore, it is easy to extend the functionality of the application in future. The layered architecture of `LifeSocial` is shown in the figure (add ref). `LifeSocial` architecture contains four layers and each layer can communicate with other through well defined interfaces [15]. The architecture is shown in the figure 2.1 and the layers are explained below.

2.3.1 Peer-to-Peer Overlay Layer

The main functionality of P2P overlay layer is to provide the routing functionality by connecting the nodes. This layer is presented at bottom level and it provides the peer-to-peer functionalities. This layer uses an open source implementation of the structured overlay called *FreePastry* which is developed by Rice University in USA. In structured overlay, the peers are connected as per the designed protocol. FreePastry provides each peer or node in the pastry

network with unique, uniform random identifier[1]. In FreePastry, the peers and objects are mapped into a 128-bit identifier space.

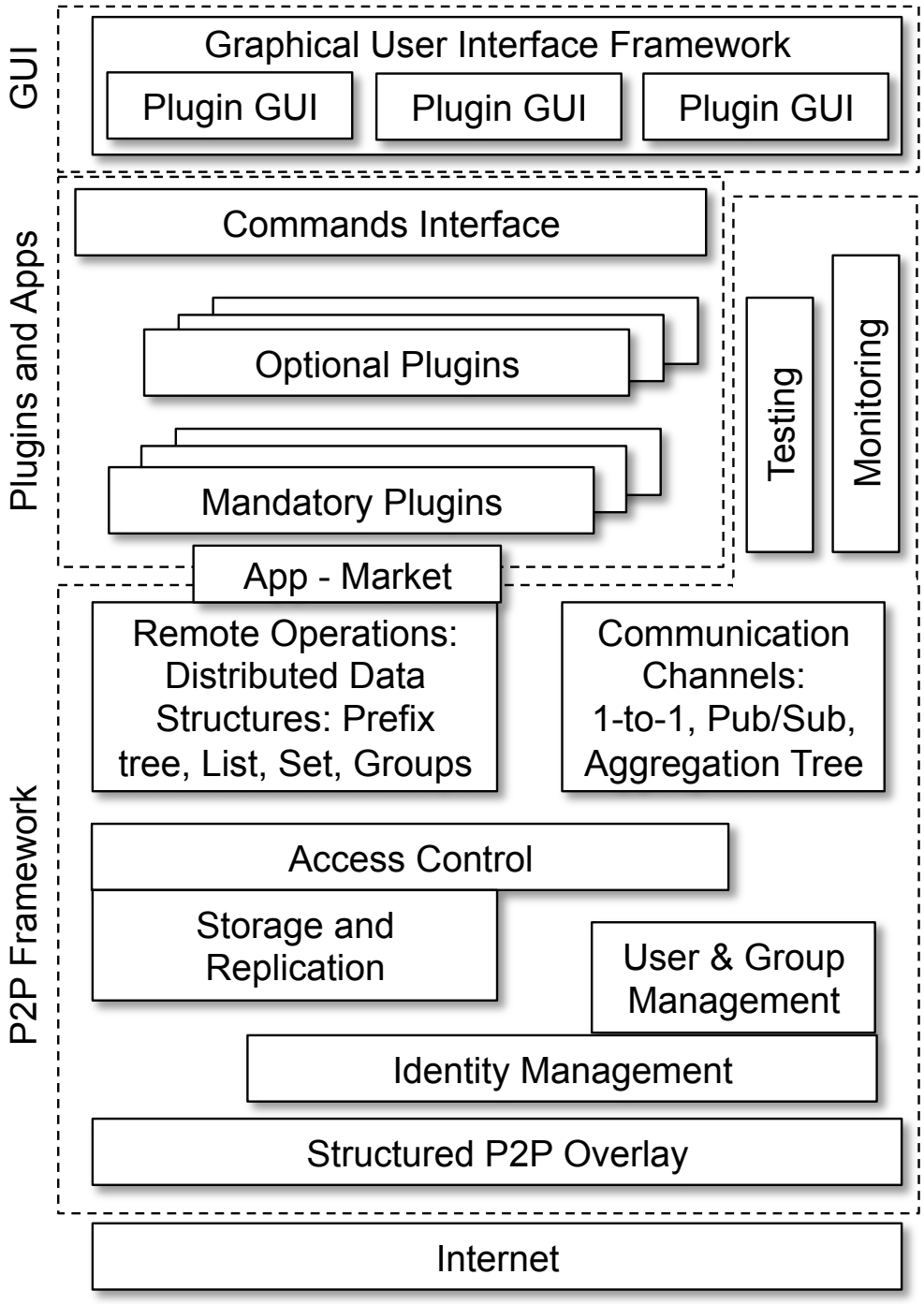


Figure 2.1: LifeSocial Architecture[15]

2.3.2 Peer-to-Peer Framework layer

P2P Framework layer is located on the top of the P2P overlay layer. This layer provides core functionalities like data structures, access control and secure storage. The access control mechanism is used to allow the authorized user to read or write the data items in the network [15]. The plugins and apps in the upper layer uses these functionalities through a well defined interface.

2.3.3 Plugins and Apps Layer

Generally, a plugin is a software component that adds the specific feature to enhance its capability. The plugins are used to increase the extensibility, simplicity and allows the parallel development to a software application. In `LifeSocial`, plugins are based on OSGi framework and are placed at the top of the P2P Framework layer of the architecture which uses the functionality provided by the framework layer. The plugins in the `LifeSocial` are managed by the App store. Since, all plugins are GUI based it requires user interactions. Each plugin provides the OSGi commands which are used by the `TestPlugin` to check the functionality of the TestCases. This subsection gives the overview of the available plugins and OSGi commands of the `LifeSocial` application.

Vote Plugin

The word Vote refers to a choice or an opinion made by the user. In `LifeSocial`, the vote plugin is used to make a user choice between the different options for a survey, scheduling events, decision making, etc. By using this plugin the user can create a public or private vote, add the number of choices for voting, set the maximum and minimum number of choices made by a user in the poll, invite the other users for voting, see the vote results and get the existing public vote by the corresponding vote ID etc. The available OSGi commands of vote plugin are shown in the table 2.1.

Command	Description
CreateVote	Creates new voting with specified topic, description, a list of options.
DeleteVote	Deletes the voting with specified id
GetMyVotings	Returns all the votings the user created, added or was invited in
GetVotingResults	Return the results of specified voting
InviteUser	Invite a user to vote
Vote	Vote for a voting

Table 2.1: Vote Plugin OSGi Commands

Messaging Plugin

Message plugin is like a email application. By using this plugin user can send and receive the messages. It provides Inbox and Outbox which contains re-

ceived and sent messages. The user can also organize his/her messages by using labels. The application notifies the user when he/she receives a new message. The available OSGi commands of message plugin are shown in the table 2.2.

Command	Description
SendMsg	Sends a message to a given user
GetInbox	Returns message Id's available in inbox
GetOutbox	Returns message Id's available in sent box
DeleteInboxMsg	Deletes an inbox message with given message key

Table 2.2: Messaging Plugin OSGi Commands

Wall Plugin
Every user in the network owns a profile space and the purpose of the wall plugin is to post messages or entries on his/her profile. Later, the user friends can comment on that post. It also provides user privacy by offering the different visibility options like only for me, public, all my friends. The available OSGi commands of wall plugin are shown in the table 2.3.

Command	Description
AddComment	Adds a comment to the wall post.
AddPost	Adds a new post shown on own wall and the walls of all friends.
RetrievePosts	Retrives all posts of the user with the specified user ID.

Table 2.3: Wall Plugin OSGi Commands

Chat Plugin
In LifeSocial, chat plugin offers real time transmission of test messages between user and his friends. It supports one-to-one and many-to-many communications. The user can send the attachments, use the emoji, and the url parser in the chat interface. The user interface provides the chat history of the past conversations. The available OSGi commands of chat plugin are shown in the table 2.4.

Command	Description
SendChatMsg	Sends a chat message in the given channel.
SendInv	Sends a chat invitation to the given users.
UnsubscribeTopic	Unsubscribe to topic channel and leaves multi chat.

Table 2.4: Chat Plugin OSGi Commands

Photos Plugin
In LifeSocial, user can upload photos to his profile space to share it with his/her friends. The user can set the visibility options of the photos. If the

visibility of the photo is set to public then the user can share the photo with his friends. The application interface provides various features like delete a photo, create a new album, delete album, edit album, edit photo etc. The available OSGi commands of photos plugin are shown in the table 2.5.

Command	Description
AddPhoto	Adds a photo to the defined album.
AddAlbum	Adds a photo album.
EditAlbum	Edits the information of the album.
LoadMyAlbums	Loads own albums.
LoadUserAlbums	Loads the albums of the given user.
DeletePhoto	Deletes a photo with given keys.
DeleteAlbum	Deletes an album with the given key.

Table 2.5: Photos Plugin OSGi Commands

Groups Plugin

The groups are used to share different things with different people. The user can create the group, join the group, leave the group at any time and add friends to the group. When a user is added, all the group members are get notified that a new user is added to the group. Groups plugin provides group drive and group forum. The groups drive is used to share the files between the group members. Unlike the owner of the group all the members in the group can create folders and upload the files. The group forum interface provides the user to create threads to have discussion on the topics. The available OSGi commands of groups plugin are shown in the table 2.6.

Command	Description
AddFiles	Add files into specific folder.
AddFolder	Create a folder with specific parameters.
AddFriendstoGroup	Add friends into group.
AddGroup	Create a group with specific parameters.
DeleteContent	Delete an item and its sub-items with specific parameters.
DeleteGroup	Delete a group with specific group id.
GetContents	Retrieve contents of a group or folder with specific parameters.
GetGroupsList	Get the group list of the current logged-in user.
LeaveGroup	Removes the current logged-in user from the group.
RenameContent	Rename a item with specific parameters.

Table 2.6: Groups Plugin OSGi Commands

Files Plugin

The registered users in the network can upload, download and delete the files. The plugin supports different types of file exetensions like .pdf, .xls, .ppt, .doc, .png, .jpeg etc. The user can also share the files with his friends via messages

and chat. The delete option in the user interface is used to permanently remove the files from the network. The available OSGi commands of files plugin are shown in the table 2.7.

Command	Description
StoreFileCommand	To store a file in the network.

Table 2.7: Files Plugin OSGi Commands

Friends Plugin

The friends plugin is used to send friend request to the other users in the network, accept and decline the friend request. The user can block and unblock friends form his friend list. The available OSGi commands of friends plugin are shown in the table 2.8.

Command	Description
SendFriendRequest	Sends a friend request to a user.
BlockFriendFromUserFriendsList	Blocks a friend from user friends list.

Table 2.8: Friends Plugin OSGi Commands

Calender Plugin

The aim of the calender plugin is used to organize the schedule and keep track of important events of users life. In `LifeSocial`, every user have a private calender and he can create, modify or delete the events. Every event contains the information like name of the event, description, start date, start time, end date and end time. He can also use the event remainder to stay on the schedule. The application reminds the user by sending notifications at the time of the event. The available OSGi commands of calender plugin are shown in the table 2.9.

Command	Description
CreateEvent	Creates an event with specific parameters.
DeleteEvent	Delete an event with specific id.
EditEvent	Edit an event with specific parameters.
GetEvents	View user's calender of specific month.

Table 2.9: Calender Plugin OSGi Commands

Search Plugin

The purpose of the search plugin is to find the user friends or other users who are registered in the network. It contains search fields like name, gender, city, occupation, country, etc., which helps the user to find his/her fried in the existing network. There are some limitations in this plugin, it is just used to

search friends and it does not provide other information like messages, groups and votes. The available OSGi commands of search plugin are shown in the table 2.10.

Command	Description
GetSearchResults	Returns a list of users retrieved with specific criteria.

Table 2.10: Search Plugin OSGi Commands

Profile Plugin

Profile plugin is used to set the user information like name, age, address, email, phone number etc. Each and every field in the user profile has a privacy feature where a user can control the privacy of their data. The user can also set profile and a cover picture to their profile page. The available OSGi commands of profile plugin are shown in the table 2.11.

Command	Description
GetMyProfile	Returns own profile.
ShowCurrentProfile	Shows the previously requested profile.
StoreMyProfile	Stores own profile.

Table 2.11: Profile Plugin OSGi Commands

Login Plugin

Login plugin is used to login and logout the user from the network. It requires the user credentials to login into the network. The available OSGi commands of login plugin are shown in the table 2.12.

Command	Description
Register	Register a user account.
Login	Logs the current user in.
Logout	Logs the current user out.

Table 2.12: Login Plugin OSGi Commands

Notification Plugin

In `LifeSocial`, Notification center is placed at the top-right of the application. The purpose of this notification center is to inform the user by providing alerts and related information of the network. The application plays beep sound to notify the user if any new notifications are available. Currently, there are no OSGi commands available for this plugin.

Audio/Video Chat Plugin

Audio/Video chat plugin helps you to communicate with your family and friends from your LifeSocial aplication. It supports both audio and video inputs and it provides one-to-one audio and one-to-one video communication. There are no OSGi commands available for this plugin.

2.3.4 GUI Layer

The GUI layer is located at the top of the LifeSocial architecture. In `LifeSocial`, the standard web technologies like HTML5, AJAX, JQuery, Bootstrap and Knockback are used to develop the graphical user interface. By using these technologies not only increases the interactivity and usability, but also gives a stunning look to the `LifeSocial` application. The GUI of `LifeSocial` application is shown in the figure 2.2.

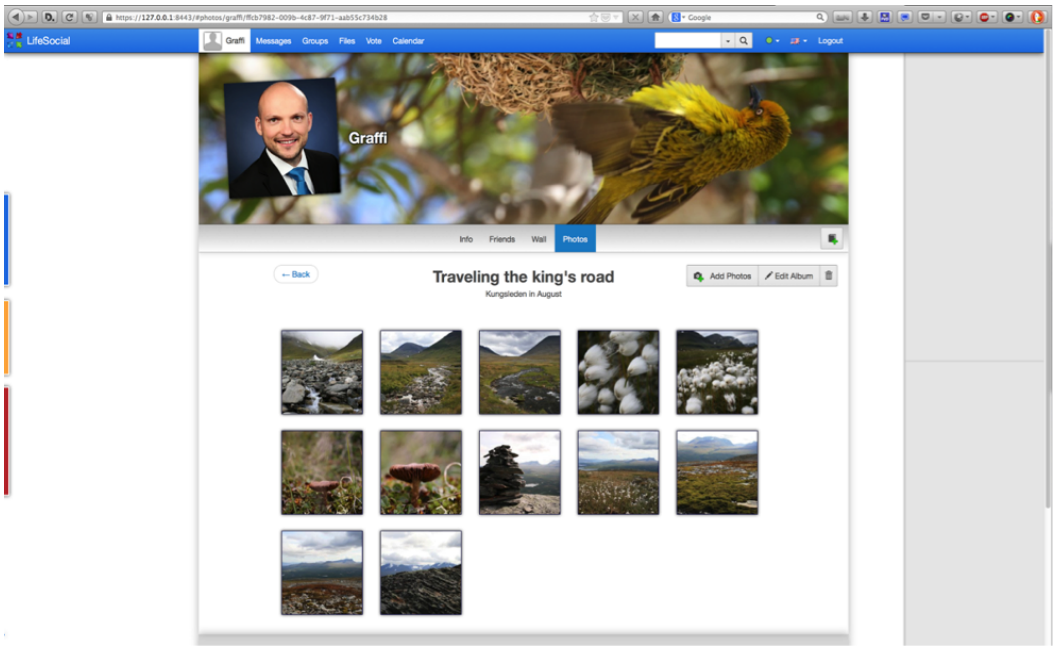


Figure 2.2: GUI of user photos page in LifeSocial application[15]

2.4 Basic Definitions

According to the International Software Testing Quality Board (ISTQB), Testing is a process which aims at planning, preparing and evaluating a program in order to find errors, faults and failures.

Unit Testing

Unit test is a method of testing the exactness and robustness of the meticulous element of source code. The main goal of this is to segregate each part of program and prove that every single part of code is true. In which the test cases are written for every non-trivial method so that every test case is unique to itself. The major profits of isolated unit testing are easy and encourage changes, make the integration process effortless, document the code, differs the interface from implementation.

Integration Testing

This type of testing is applicable to client-server or distributed systems. It is supposed to be performed after unit testing and precedes system testing. The aim of this testing is to merge each separate module and test them as a group. The tested result of unit testing is taken as inputs and combines them in larger aggregates to deliver the output; makes the system prepared for system testing. The main purpose of integration testing is to verify the performance and reliability requirements.

System Testing

System testing is performed on integration testing, to test the entire system and evaluate whether it matches with the system requirements or not.

Acceptance Testing

Acceptance testing is performed subsequent to system testing. In which, the clients are involved to validate the system as per their requirements. Even though there are no key faults in the system, we cannot make sure that the client expectations have accomplished.

Alpha Testing

Alpha test will be performed by the actual user for the first time. The goal of this testing is to find the correctness of the final product developed. It is a prior work to beta testing. The bugs noticed in this stage can be fixed before performing beta test based on the user satisfaction.

Beta Testing

Beta test is executed by end users for a longer duration to confirm the developed system/product.

Test Plan

Test plan summarizes the methodical or logical approach of the system to be tested. This consists of test scope and activities to be attained with the time estimated. The test plan should be accepted by both client and supplier before starting the process.

Test Environment

Test environment is a place where all the tests are designed, stored and executed. This can be a hardware or software. The tests from this environment can interact with other software too.

Test Case

Test case is a part of the system which describes about input to take, output to receive and expected values or conditions. It is written to test a specific functionality of the system.

2.5 Secure Shell

Secure Shell (SSH) is a program used for contributing the preeminent security between the systems when using remotely. SSH provides the secured encrypted network protocol for any kind of communication. Suppose when there is any communication between client and server; the user need to login the machine. The password (or any other secured data) can be known to the other user who is working on the same network. SSH comes in such scenario to protect the data and so nobody else can read it without session-negotiated keys.

SSH tunneling

SSH tunnel is used for relocating the unencrypted traffic all the way through encrypted channel in the network. The encrypted channel is formed using SSH protocol connection. This tunnel can be used to move the data or files between an FTP server and a client securely though the server protocol is not encrypted. The other major advantage of SSH tunnel is, it can supply a bypass firewalls that restricts definite internet services, i.e. when the organization block few specified websites using proxy filter; such things can avoid by connecting to an external SSH server and creating an SSH tunnel.

SSH Port Forwarding

SSH port forwarding is used to create a virtual private network (VPN). Port forwarding through SSH tunnelling establishes a protected and safe connection between local and remote system by which the services are communicated. Port forwarding aspect may import or export different kinds of internet traffic in/out of the network secretly. This can be used to stay away from network monitoring. There are 3 categories of port forwarding techniques.

Local port forwarding Local port forwarding connections are made from client are forwarded by server, then to a target server. It is a most common method used to connect from local machine to other server. To use this, details like source port number, destination port number and destination server should be known. In the below command the option -L specifies local port forwarding.

```
$ ssh -L <source port number>:<destination server>:
  <destination port number> <host>
```

Remote port forwarding Remote port forwarding connections are made from server are forwarded by client, then to a target server. It is less commonly used method to connect from remote SSH server to another server. In the below command the option -R specifies local port forwarding.

```
$ ssh -R <source port number>:<destination server>:
  <destination port number> <host>
```

Dynamic port forwarding Dynamic port forwarding connections are made from different programs are forwarded by client, then by server, and finally to different target servers. It is a rarely used method to connect SSH client into a proxy server.

3 Testing

Testing is one of the important phase in software development life cycle that has to be considered. It is very important because human beings made mistakes all the time in any process, some of those mistakes have less very impact, but some mistakes are very dangerous which can crash the whole software system. We need to test the whole system and find those mistakes before the software system is deployed in the production environment.

In this chapter, we explain the testing in detail. First we present the basic definitions and goals of testing. Thereafter, we explained the prerequisites and requirements details that are required to develop a test plugin in `LifeSocial` application. Next, we explained about the test case structure, test case implementation, example test case and the list of test cases. Afterwards, we explained about the metrics that are required to measure the quality, load and performance of the `LifeSocial` application. Finally, all kinds of setup parameters are introduced in the last part. This chapter is written by both the authors. The sections 3.1 and 3.2 are written by the D.Balabhadruni and the sections 3.3, 3.4, and 3.5 are written by W.Peng.

3.1 Test Plugin

The goal of the Test Plugin is to test the performance and reliability of the `LifeSocial` application. The tester can enable or disable the Test Plugin in config.txt file. The Test Plugin contains two roles namely Master and Slave. The master sends the test plan to the slaves and slaves executes it.

3.1.1 Prerequisites

Before we setup the Test Plugin following prerequisites are need to take care of.

Test Plugin component - To use the Test Plugin, the test plugin component `com.p2pframework.tester.component` must be added to the application's dependency.

Resource Folder - Some `txt` files are required to run the test plugin. All these files must be placed in the resources folder and described in table 3.1.

config.txt	Used to configure the IP address of Master Node, enable or disable the Test Plugin and set the paths of the test plan.
cityNames.txt	Consists of available city names.
countryNames.txt	Consists of available country names.
groupNames.txt	Consists of available group names.
interests.txt	Consists of available users interests.
loginNames.txt	Consists of available login names.
messages.txt	Consists of available messages.
onlineDuration.txt	Contains the Online Duration Distribution
startScript.txt	Defines a sequence of actions user need to execute when joining the network in random test.
subjects.txt	Consists of available message subjects.
pictures	Folder which consists of available pictures.
testCaseRegistry.txt	Contains class names of all used and implemented test cases

Table 3.1: Files in the resource folder

The files config.txt, onlineDuration.txt and testCaseRegistry.txt are used for configuration and the remaining files are used for sample content generation. For example, when the slave node registered in the network it gets the random login name which are listed in the `loginnames.txt` file.

Resource Folder Path Location - The path location of the resources folder is different from different operating systems and it is described in the table 3.2

Windows	/eclipse/resources
MacOSX	/Applications/Eclipse/Eclipse.app/Contents/MacOS/resources
OpenSuse	/eclipse/resources

Table 3.2: Resource folder location path in different operating systems

3.1.2 Requirements

This subsection explains the requirements of Test Plugin.

Master node

The master is unique in the network and it should be started first during the test. The role of this Master is to send the commands to the slave instances in

the network, every slave instance login into the network via Master. To setup a master instance the server parameter in the `config.txt` must be same as the IP address of the machine. The graphical user interface of master instance contains two types of views called `Master view` and `Delegater view` which are shown in the figures 3.1 and 3.2.

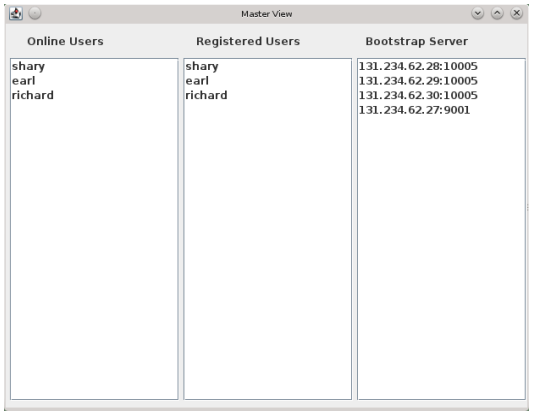


Figure 3.1: Master View of LifeSocial Instance

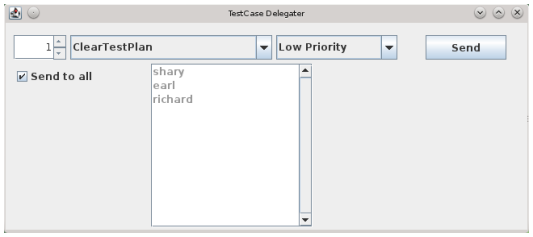


Figure 3.2: Delegater View of LifeSocial Instance

- ***Master View:*** Master view contains the list of online users, registered users and the bootstrap server. All the slave nodes need a permission from the master node to join the P2P network, and the master checks whether the user of that slave node is registered or not. If the user is registered, then it grants a permission to join the network and it allocates a bootstrap node for user to login. Otherwise, the slave node choose a new random name from the *loginnames.txt* file and repeat the process.

- ***Delegater View:*** Delegater view is used to send the command to the slaves. It contains a list of online users shown in the list box, the right and left combo boxes are used to select the command and the execution priority of the command. The check box *Send to all* is used to select all the users and the `Send` button is used to send the command to the users.

Slave node

The role of the slave node is to execute the command sent by its master. To setup a slave node the server parameter in the *config.txt* must be same as the IP address of the master node. There should be atleast one or more slaves in the network.

Test Plan

A test plan is a document which is used to test the design specifications and requirements of the software application. It contains a list of test cases which are executed in the same sequence mentioned in the document. In order to initiate test plan a signal has to be send using a command by the master instance to the slaves instances and this can be done in two ways. One way is to send the command through the test case delegater view (figure 3.2) of the master instance, and another way is to send the command through the eclipse IDE console of the master instance. If test plan is empty or no more test cases to execute then slave state is set to *Idle*.

- ***Test plan path:*** The path of the test plan may refer to local file or remote file, and can be configured in the *config.txt* file. In this thesis, the test plan is stored in *Dropbox* and its path is set in *config.txt* file.

3.1.3 Test Case Structure

This subsection explains about the structure of the test case. The `TestCase` abstract class in the Test Plugin is used to define and implement the test cases. The parameters and preconditions must be defined in every test case before its execution. Test Plugin can automatically generates the missing parameters for the test case and to comply its preconditions.
The abstract class `TestCase` contains a public method `executeTestCase()` to check automatically whether the preconditions and parameters are compiled or not. This method is called during the execution of each test case. The control flow of this method is shown in the figure 3.3and explained below.

1. Execution of executeTestCase() method is started.
2. Checks the validation of all the needed parameters.
3. If the parameters are invalid, it generates the missing parameters.
4. If the parameters are valid, then it continues to verify whether all the preconditions are compiled or not.

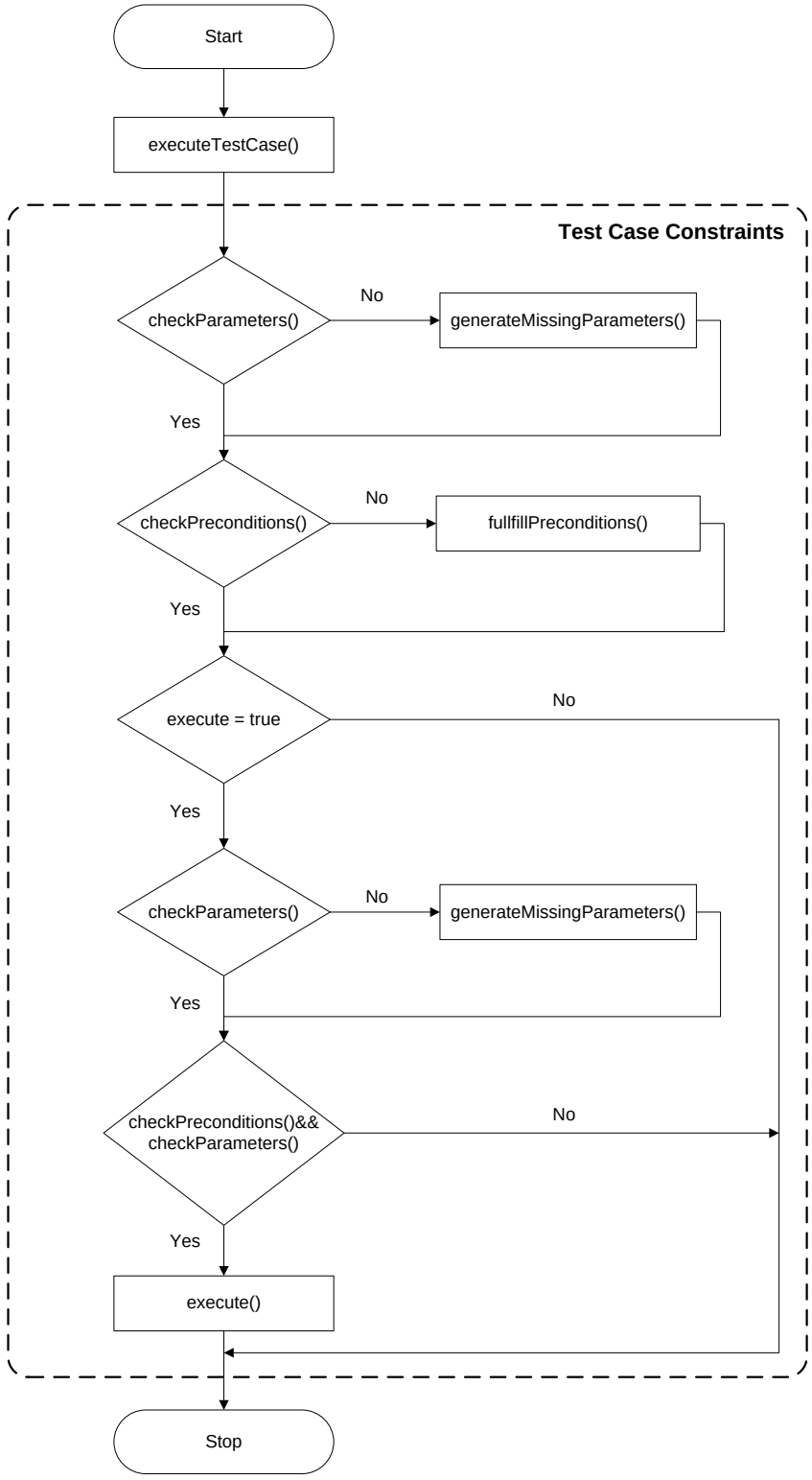


Figure 3.3: Flowchart of executeTestCase() method implemented by each Test Case

5. If the preconditions are not compiled, then it will fulfill the preconditions.

6. The test case in no need to be executed if there are any side effects in the fulfillment of the preconditions. So it checks whether the test case is still be executed or not. If not the test case execution is terminated.

7. Again check for valid parameters and generates missing parameters if necessary.

8. Now it first check the preconditions, and parameters next. If anyone of these are invalid then the test case execution is terminated.

9. If all the parameters are valid and all the preconditions are compiled then the test case is executed.

In [17] the author mentioned that there are some side effects with the fulfillment of preconditions, and this is the reason behind the execution of 6, 7 and 8 steps to overcome the problem.

Preconditions

In [17] the author explains about the concept of the Precondition. It is an abstract class which contains `isFulfilled()` and `fulfill()` methods. The method `isFulfilled()` is used to check whether the certain condition is true or false, and it returns the result. The fallback is a `TestCase`, that needs to be executed to fulfill the preconditions of the certain test case. In this abstract class, the `fulfill()` method of the is overridden to execute the fallback of the implemented test case. The implementation of the abstract class precondition is shown in the figure 3.4.

There are two helper classes `ParameterGenerator` and `Condition Verifier` are used to support the implementation of the Test Case and its preconditions. The `ParameterGenerator` class contains the methods that provides the random values to the parameters used in the test cases and the `Condition Verifier` contains the methods to check the multiple conditions. The implemented preconditions in the `LifeSocial` are shown in the table 3.3.

PreconditionAlbumExists	Is fulfilled, if the user already has a photo album with the specified name.
PreconditionAlbumExistsNot	Is fulfilled, if a photo album with the specified name does not exist.
PreconditionAlbumsLoaded	Is fulfilled, if the users or a friends photo albums are loaded.
PreconditionCalendarLoaded	Is fulfilled, if the users latest calendar list are loaded.
PreconditionFriendlistLoaded	Is fulfilled, if the users latest friend list are loaded.
PreconditionFriendsHavePhoto	Is fulfilled, if the users friends have a photo.
PreconditionGroupNotExists	Is fulfilled, if a group with specified name is not exists.
PreconditionGroupsLoaded	Is fulfilled, if the groups are loaded.
PreconditionHasAlbum	Is fulfilled, if the user has photo album.
PreconditionHasFolder	Is fulfilled, if the group contains a folder.
PreconditionHasForumThread	s fulfilled, if the group contains a forum thread.
PreconditionHasFriendship	Is fulfilled, if the user has friendship.
PreconditionHasGroup	Is fulfilled, if the user has group.
PreconditionHasInboxMessage	Is fulfilled, if the user has inbox message.
PreconditionHasPhoto	Is fulfilled, if the user has photo.
PreconditionHasTopicChannel	Is fulfilled, if the user has topic channel.
PreconditionHasVote	Is fulfilled, if there is a vote.
PreconditionLoggedInUser	Is fulfilled, if a user is logged in and connected to the network.
PreconditionProfileLoaded	Is fulfilled, if a certain profile is loaded.
PreconditionUserExists	Is fulfilled, if a given username is already registered.
PreconditionUserExistsNot	Is fulfilled, if a given username is not already registered.
PreconditionUserNameNotLoggedIn	Is fulfilled, if a given username is not already logged in to the network.
PreconditionVoteNotExist	Is fulfilled, if the votes with the specific name does not exist.
PreconditionVotesLoaded	Is fulfilled, if the users latest votes list are loaded.
PreconditionWallPostExists	Is fulfilled, if a wall post exists for a specified user.

Table 3.3: Implemented Preconditions

```
public abstract class Precondition
{
  protected TestCase fallback = null;
  protected TestCase dependingTestCase = null;

  public abstract boolean isFulfilled();

  public void fulfill()
  {
    System.out.println("Precondition " + this + " is going to be fulfilled");
    if (fallback != null)
    {
      System.out.println("Executing fallback " + fallback);
      fallback.matchCommonParameters(dependingTestCase);
      fallback.executeTestCase();
    }
  }
}
```

Figure 3.4: Implementation of abstract class Precondition

3.1.4 Test Case Implementation

This subsection explains about the implementation of the test case. The control flow of the test case implementation is shown in the figure 3.5 and explained below .

1. Create a new test case which extends the TestCase abstract class.
2. Override the TestCase abstract methodsin the new test case.
3. Check the corresponding OSGi command to check the functionality of the new test case is exists or not. If the command is presented then it will execute step 7.
4. If the OSGi command is not presented, then create a new OSGi command for the new test case.
5. Define OSGi declarative services for the OSGi command via component.xml file.
6. Add XML component files to the `Service-Component` header in `MANIFEST.MF` file.
7. Reference a service in *TestComponent.xml* to implement the plug-in interface of the command.
8. Execute the corresponding OSGi command in the test case with the required parameters as inputs.

9. Add preconditions in the default constructor class to check.

10. Now, the execution of the test case is same as the *Test Case Constraints* block in the 3.3.

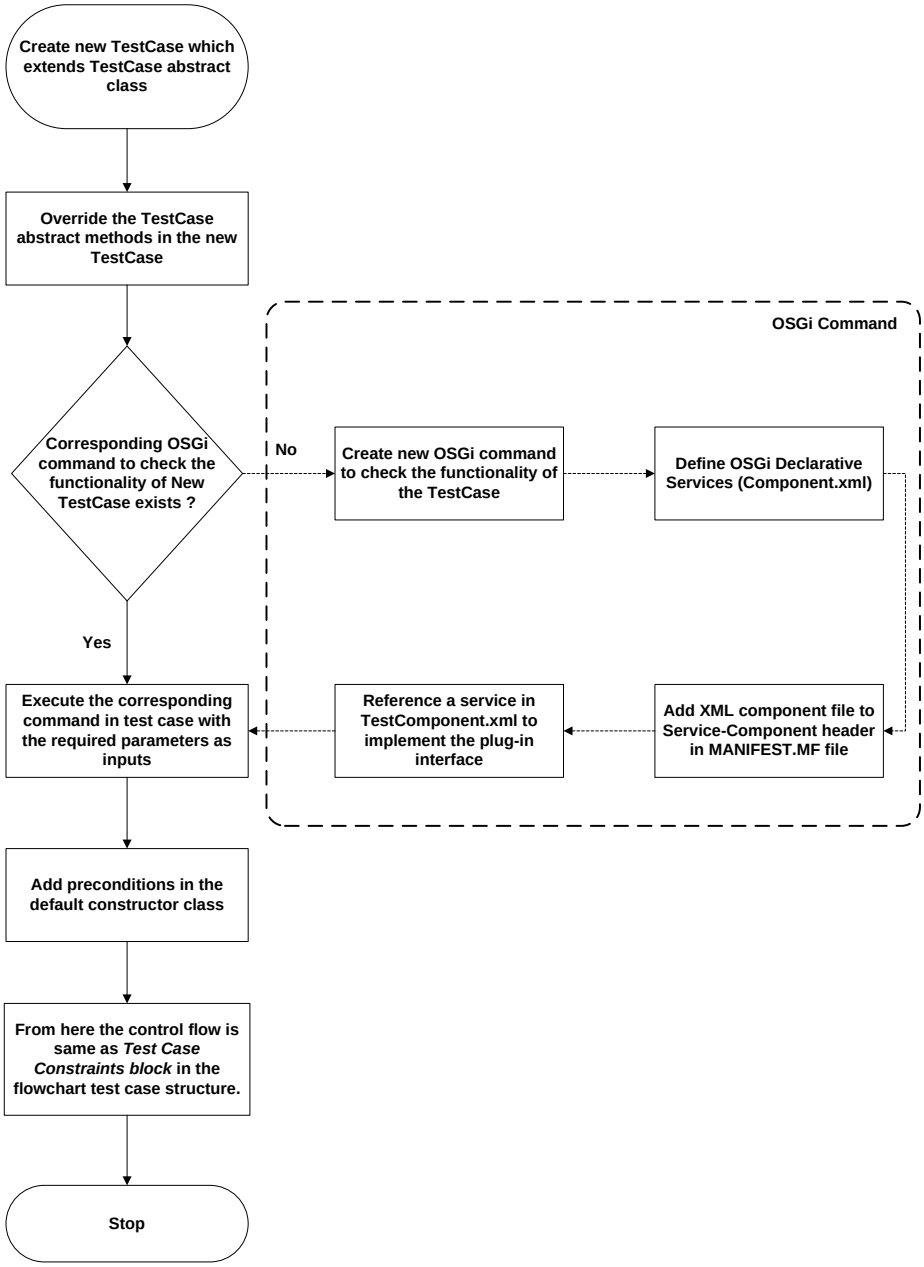


Figure 3.5: TestCaseImplementation

3.2 Example Test Case

This section explains the implementation of test case `TestCaseDeleteCalendarEvent` as an example, and is shown in the figure 3.6. The class `TestCaseDeleteCalendarEvent` is a new test case, and it is inherited from the abstract class `Test Case`. So the abstract methods(lines 22-46) of the class `Test Case` must be implemented. The aim of this test case is to delete the calender event, and its functionality is provided by the `DeleteEvent` OSGi command (figure 3.8) in the calendar plugin. This commands needs a parameter called `eventid` to delete the event in a calender. For this reason, we are defining the variable `eventid` in this class. The preconditions are defined in the constructors (lines 11-13) of this class to verify that the user is logged in, the calendar list is loaded and is contains an event.

This test case contains two constructors the parameterized constructor and parameter free constructor. To assign default values to the object of the class `TestCaseDeleteCalendarEvent` the parameter free constructor is called, which in turn calls the parameterized constructor with null as a parameter. This parameterized constructor now assigns the name variable with the test case name, and assigns parameters to the class variables (lines 8 and 9).

The lines 29 to 34 explains the method `checkParameters()` to check the validity of the parameter. This method returns `true` if the assigned variable is not equal to null. Otherwise, it returns `false`. If these parameters are invalid then it generate the missing parameters (lines 37 to 41). The helper class `ParameterGenerator` contains the method `getCalendarEventID()` to get the random event id from the list of the calendar events.

The method `execute()` is still remained to be executed. This method contains the OSGi command which provides the functionality of this test case. When this method is executed it adds the generated parameters to the OSGi command and executes it.

Finally, all the implemented test cases must be registered to the Test Plugin. This can be done by adding the class name of the implemented test case to the `testCaseRegistry.txt` file in the resource folder (see 3.1.1). The test case category can be assigned by adding the name after a colon. For example, the registration of the test case `TestCaseDeleteCalendarEvent` would be looks like this in `testCaseRegistry.txt` file.

```
TestCaseDeleteCalendarEvent :  Calendar
```

```
public class TestCaseDeleteCalendarEvent extends TestCase
{
  private String eventid;

  public TestCaseDeleteCalendarEvent(String eventid)
  {
    name = "TestCaseDeleteCalendarEvent";
    this.eventid = eventid;

    preconditions.add(new PreconditionLoggedInUser(this));
    preconditions.add(new PreconditionCalendarLoaded(this));
    preconditions.add(new PreconditionHasCalendarEvent(this));
  }

  public TestCaseDeleteCalendarEvent()
  {
    this(null);
  }

  @Override
  protected void execute()
  {
    String command = "CalendarPlugin.DeleteEvent " + eventid;
    commands.executeToString(command);
  }

  @Override
  protected boolean checkParameters()
  {
    if (eventid != null)
      return true;
    return false;
  }

  @Override
  protected void generateMissingParameters()
  {
    ParameterGenerator.getParameterGenerator().loadCalendar();
    eventid = ParameterGenerator.getParameterGenerator().getCalendarEventID();
  }

  @Override
  public void matchCommonParameters(TestCase dependingTestCase)
  {
  }

}
```

Figure 3.6: Implementation of test case TestCaseDeleteCalendarEvent

3.2.1 Implementing a Precondition

In the above section we explained the implementation of the `TestCaseDeleteCalendarEvent` test case. Here we will explain the implementation of one of its precondition `PreconditionHasCalendarEvent`. The class `PreconditionHasCalendarEvent` is inherited from the `Precondition` abstract class. The constructor assigns the fallback to the `TestCaseCreateCalendarEvent()` and the `dependingTestCase` (line 5 and 6). If the precondition i.e, the calendar has event is failed to fulfill then the `fallback` is executed. For this reason, it executes the `TestCaseCreateCalendarEvent` to have a calendar event, and if the precondition is fulfilled i.e, if a calendar event exists then it returns true else false.

```
public class PreconditionHasCalendarEvent extends Precondition
{
  public PreconditionHasCalendarEvent(TestCase dependingTestCase)
  {
    fallback = new TestCaseCreateCalendarEvent();
    this.dependingTestCase = dependingTestCase;
  }

  @Override
  public boolean isFulfilled()
  {
    return ConditionVerifier.getConditionVerifier().hasCalendarEvent();
  }

  public String toString()
  {
    return "Precondition: Has Calendar Event";
  }

}
```

Figure 3.7: Implementation of class PreconditionCalendarLoaded

3.2.2 Implementing OSGi Command

This subsection explains the implementation of the OSGi command that we used in the `TestCaseDeleteCalendarEvent` test case. Every plugin contains the abstract command handler class and all the commands of that plugin are inherited from it. These commands are used by the test case to remotely initiate the plugin actions. In figure 3.8 the `CommandDeleteEvent` class is inherited from the `AbstractCalendarCommandHandler` abstract class. So the abstract methods (lines 12 - 28) of the `AbstractCalendarCommandHandler` class must be implemented. In the `CommandDeleteEvent()` constructor the value is assigned

to the command from the PluginPropertiesContainer. The main functionality of this class is based on the method `executeCommand(String arg)`, it calls the deleteEvent of the calender plugin to delete the calendar event. On successful deletion it returns the "Event has been deleted" message, else it returns "parameter is missing" message. These messages are displayed in the eclipse console to notify the tester.

```
public class CommandDeleteEvent extends AbstractCalendarCommandHandler
{
  public CommandDeleteEvent()
  {
    this.command = PluginPropertiesContainer.CommandDeleteEvent;
        this.help = "<eventid> - delete a event with specific id.";
  }

  @Override
  public Object executeCommand(String arg)
  {
    if (this.checkValidArg(arg))
    {
      String eventid = arg;
      this.calendarPlugin.deleteEvent(eventid);
      return PluginPropertiesContainer.EventDeletedMsg;
    }
    return PluginPropertiesContainer.ParameterMissingMsg;
  }

  @Override
  public String executeCommandToString(String arg)
  {
    return this.executeCommand(arg).toString();
  }

}
```

Figure 3.8: Implementation of class CommandDeleteEvent

3.2.3 Defining OSGi Declarative Services

After the OSGi Command is defined, the next step is to define the OSGi declarative services. We use these declarative services to define the OSGi services without extending or implementing the OSGi classes [5]. These declarative services allows the tester to define and use the services via XML (metadata). The component definition files must be placed in the corresponding plugin bundle's OSGI-INF/ directory and can be added via `New > Other > Plug-in development > Component Definition`. The `CommandDeleteEvent` command component definition is shown in the figure 3.9 and its metadata is explained here.

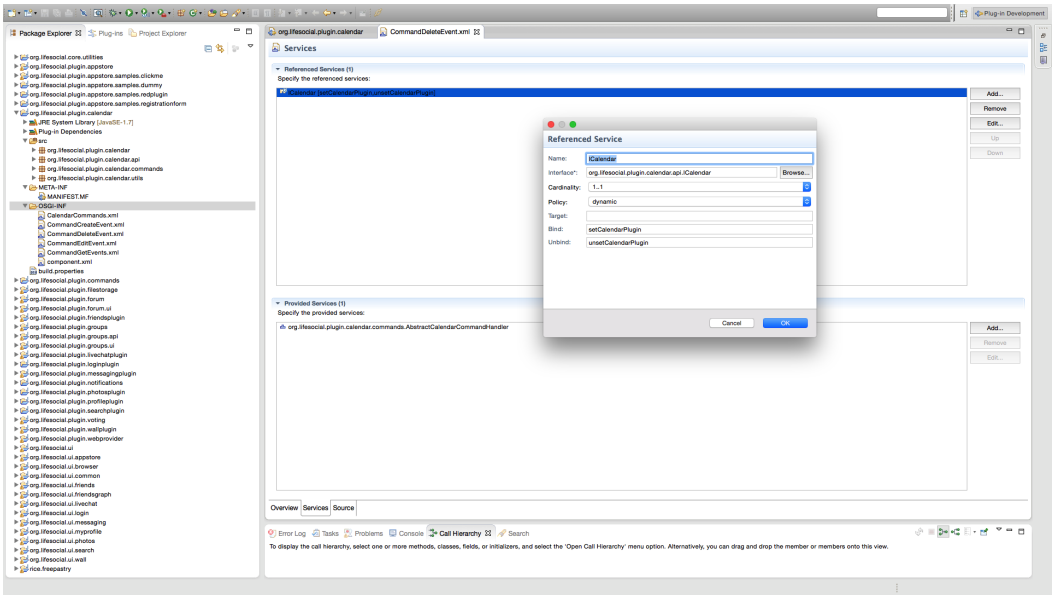


Figure 3.9: Component definition of CommandDeleteEvent.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="CommandDeleteEvent">
   <implementation class="org.lifesocial.plugin.calendar.commands.CommandDeleteEvent"/>
   <reference bind="setCalendarPlugin" cardinality="1..1"
   interface="org.lifesocial.plugin.calendar.api.ICalendar"
   name="ICalendar" policy="dynamic" unbind="unsetCalendarPlugin"/>
   <service>
      <provide interface=
      "org.lifesocial.plugin.calendar.commands.AbstractCalendarCommandHandler"/>
   </service>
</scr:component>
```

Figure 3.10: Component definition of CommandDeleteEvent.xml file

The definition of the service component shown in the figure 3.10 is explained here.

- In line 2, service name is defined.
- In line 3, class name of the service class is defined here to implement.
- In line 4, the service references bind, unbind, interface and cardinality are defined. The defined methods are called to bind and unbind. Cardinality defines the number of service to be bound and interface defines the name of the service to be bound.
- In line 9, the `AbstractCalendarCommandHandler` service is declared here.

3.2.4 Define bundle with component metadata

The `MANIFEST.MF` file defines the OSGi bundle. This file contains the name, version, bundle activator, exported, service-component, exported packages, imported packages and all other information of the bundle[4]. The XML component files must be added to the `MANIFEST.MF` file by using *Service-Component* header. If you miss this statement or XML files are not listed then the declarative services cannot find the corresponding service. For example, in the figure 3.12 the `CommandDeleteEvent.xml` is added to the *Service-Component* header in line 18.

3.2.5 Reference all Plugin Services

`LifeSocial` framework contains many plugins, the plugin that contains the OSGi commands must be added to the Test Plugin. This can be done by adding the plugin service reference to `com.sign.plugin.testplugin/OSGI-INF/TestComponent.xml` file. If you miss this statement, then the declarative services cannot find the plugin service and it cannot execute the OSGi commands of that plugin. For example, in figure 3.11 the calendar plugin service is added in line 14.

```
<?xml version="1.0"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" immediate="false"
name="com.p2pframework.tester.component.TestComponent">
  <implementation class="com.p2pframework.tester.component.TestComponent"/>
 <reference bind="setCommandInterpreter" cardinality="1..1"
 interface="org.lifesocial.commands.api.ICommandsComponent" name="CommandInterpreter"
 policy="dynamic" unbind="unsetCommandInterpreter"/>
 <service>
    <provide interface="org.eclipse.osgi.framework.console.CommandProvider"/>
 </service>
 <reference bind="setLiveChatPlugin" cardinality="1..1"
 interface="org.lifesocial.livechatplugin.api.ILiveChatPlugin"
 name="ILiveChatPlugin" policy="dynamic" unbind="unsetLiveChatPlugin"/>
 <reference bind="setCalendarPlugin" cardinality="1..1"
 interface="org.lifesocial.plugin.calendar.api.ICalendar"
 name="ICalendar" policy="dynamic" unbind="unsetCalendarPlugin"/>
 <reference bind="setMessagingPlugin" cardinality="1..1"
 interface="org.lifesocial.messsagingplugin.api.IMessagingPlugin"
 name="IMessagingPlugin" policy="dynamic" unbind="unsetMessagingPlugin"/>
 <reference bind="setVotePlugin" cardinality="1..1"
 interface="org.lifesocial.plugin.voting.IVotePlugin"
 name="IVotePlugin" policy="dynamic" unbind="unsetVotePlugin"/>
</scr:component>
```

Figure 3.11: TestComponent.xml in Test Plugin

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Calendar
Bundle-SymbolicName: org.lifesocial.plugin.calendar
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: org.lifesocial.plugin.calendar.Activator
Require-Bundle: org.eclipse.ui,
 org.eclipse.core.runtime,
 org.lifesocial.core;bundle-version="1.0.0",
 org.lifesocial.loginplugin.component;bundle-version="1.0.0",
 rice.freepastry;bundle-version="1.0.0",
 com.p2pframework;bundle-version="0.0.1",
 org.lifesocial.plugin.notificationplugin
Bundle-ActivationPolicy: lazy
Bundle-RequiredExecutionEnvironment: JavaSE-1.7
Service-Component: OSGI-INF/component.xml,OSGI-INF/CalendarCommands.xml,
OSGI-INF/CommandCreateEvent.xml,
OSGI-INF/CommandEditEvent.xml,OSGI-INF/CommandDeleteEvent.xml,
 OSGI-INF/CommandGetEvents.xml
Import-Package: com.p2pframework,
 com.p2pframework.DDS,
 com.p2pframework.network,
 com.p2pframework.storage,
 com.p2pframework.users,
 com.p2pframework.utils,
 org.lifesocial.core.data.impl,
 org.lifesocial.core.data.interfaces,
 org.lifesocial.core.data.utils,
 org.lifesocial.localstorage.api,
 org.lifesocial.loginplugin.api,
 org.lifesocial.messagedispatcher.api,
 org.lifesocial.messsagingplugin.api,
 org.lifesocial.middlewarestorage.informationcache.caches,
 org.lifesocial.middlewarestorage.informationcache.main,
 org.lifesocial.middlewarestorage.storagedispatcher.api,
 org.lifesocial.utilities,
 org.lifesocial.logger.api,
 rice.p2p.commonapi
Export-Package: org.lifesocial.plugin.calendar,
 org.lifesocial.plugin.calendar.api
```

Figure 3.12: Calendar plugin MANIFEST.MF file

3.3 Test Cases

Previously, there were 26 test cases for `LifeSocial`. However, most of them might no longer work correctly on the new version of `LifeSocial` because it has been largely modified by P2PFramework Project Group 2013 in last year. A lot of features were changed and added. Furthermore, several plugins were

Test Case	Status
TestCaseChangeProfileInformation	valid
TestCaseChangeProfilePicture	valid
TestCaseViewOwnProfile	invalid
TestCaseViewFriendsProfile	invalid
TestCaseRegister	valid
TestCaseLogin	valid
TestCaseLogout	valid
TestCaseLogoutLogin	invalid
TestCaseSendWallPost	invalid
TestCaseCommentWallPost	invalid
TestCaseViewOwnWall	invalid
TestCaseViewFriendsWall	invalid
TestCaseCreatePhotoAlbum	valid
TestCaseLoadPhoto	invalid
TestCaseUploadPhoto	invalid
TestCaseViewFriendsPhoto	invalid
TestCaseSendFriendRequest	valid
TestCaseSearchCity	invalid
TestCaseSearchCountry	invalid
TestCaseSearchUser	invalid
TestCaseSendChatMessage	invalid
TestCaseViewChatHistory	invalid
TestCaseSendMessage	invalid
TestCaseForum	invalid
TestCaseStoreFile	invalid
TestCaseViewFriendsGroups	invalid

Table 3.4: Status of old test cases

completely redesigned.
Therefore, it is necessary to check and verify these old test cases ahead of creating new test cases for those new features. The tests on the old test cases are performed by W.Peng. And Table 3.4 shows the status of old test cases before this master thesis. As can be seen, only 7 of 26 test cases are still suitable for the new `LifeSocial`. And the other 19 test cases cannot work properly due to various reasons. Apparently, these invalid test cases need to be fixed in some way. Also, the new features of `LifeSocial` should be covered as much as possible by adding corresponding test cases.

3.3.1 List of Test Cases

During the period of thesis, some of the invalid test cases have been deleted because the related functionalities were abandoned or totally reconstructed. And the other invalid test cases have been modified based on the newest version of `LifeSocial`. Meanwhile, a number of new test cases have been created for the new plugins and functionalities of `LifeSocial`.

The modifications on the old test cases are done by W.Peng. These modified test cases are shown in Table 3.5. The new test cases are designed by two authors together. Those test cases created by D.Balabhadruni are listed in Table 3.6 and 3.7. The other test cases are created by W.Peng, which are shown in Table 3.8, 3.9 and 3.10.

3.3.2 Further Improvements

All the improvements in this subsection are made by W.Peng, after the first round of synthesis and random test.
During the evaluation on the first round of synthesis test result (Section 5.5) and random test result (Section 6.6), it is found that there are much more various exceptions in either type of random test than synthesis test. Briefly, this difference on the test results is mainly caused by the their distinct characteristics that in synthesis test, test cases are executed based on the timed test plan, which is not used in random test.
As introduced in Section 3.1.3, the *ParameterGenerator* is used to generate the parameters of a test case before it is executed. Actually, in the *ParameterGenerator*, there exist a number of global variables to simplify the parameter invocation for several relatively complicated plugins, such as Groups Plugin and Photos Plugin. Apparently, it could be problematic if these global variables are used before initialization, or their values are not updated in time. Fortunately, with the help of timed test plan, these potential issues are well avoided or controlled in synthesis test, because the execution sequence of test cases is consciously designed in the test plan. However, that is not the case for random test. In the random test either with or without churn, the test cases are always selected randomly, so that the execution order cannot be determined manually, which would lead to a lot of null pointer exceptions and other exceptions.
In order to make the execution sequence no longer matter, the independence of each test case needs to be enhanced. For this purpose, some improvements have been done before the second round of test.

- First, in *ParameterGenerator*, the functions for all the plugins involved with global variable have been standardized and classified into three categories. Previously, since the *ParameterGenerator* was implemented by several persons, some global variables were exception-prone and depended largely on the execution sequence of test cases. Now, after modification, there are three kinds of functions for each plugin which is related to global variable. One type (*load()*) is used to load the desired resources and initialize the relevant global variables. One type (*get()*) is for returning the values of requested global variables. And the last one (*detection()*) is used to check if the global variables are initialized. In this way, it is

Test Case :	**Logout Login**
Description :	To logout current user account, then register a new user and login
Preconditions :	User logged in
Parameters :	-
Test Case :	**Send Wall Post**
Description :	To send a post on own wall
Preconditions :	User logged in
Parameters :	Content of post
Test Case :	**Comment Wall Post**
Description :	To add a comment on friend's post
Preconditions :	User logged in, at least one friend, chosen friend has wall entry
Parameters :	Friend name, ID of post, content of comment
Test Case :	**View Own Wall**
Description :	To browse user's own wall
Preconditions :	User logged in
Parameters :	ID of current user
Test Case :	**View Friend's Wall**
Description :	To browse a friend's wall
Preconditions :	User logged in, at least one friend
Parameters :	ID of friend
Test Case :	**Create Photo Album**
Description :	To create a photo album for current user
Preconditions :	User logged in, user's albums loaded, album name available
Parameters :	Album name, description
Test Case :	**Upload Photo**
Description :	To add a photo into an album of user
Preconditions :	User logged in, user's albums loaded
Parameters :	Album name, photo name, path to photo, description
Test Case :	**View Friend's Photo**
Description :	To browse a photo of friend
Preconditions :	User logged in, at least one friend, albums of chosen friend loaded
Parameters :	Friend name, ID of album, ID of photo
Test Case :	**Send Message**
Description :	To send a message to another user
Preconditions :	User logged in
Parameters :	User name, subject of message, content of message

Table 3.5: Table of modified test cases

Test Case :	**Block User**
Description :	To block a friend from user's friends list
Preconditions :	User logged in, user's friends list loaded, at least one friend
Parameters :	Friend name
Test Case :	**View Inbox Messages**
Description :	To browse user's inbox messages
Preconditions :	User logged in
Parameters :	-
Test Case :	**View Outbox Messages**
Description :	To browse user's outbox messages
Preconditions :	User logged in
Parameters :	-
Test Case :	**Delete Inbox Message**
Description :	To remove a messages from user's inbox
Preconditions :	User logged in
Parameters :	ID of message
Test Case :	**Send Multi Chat Message**
Description :	To send a chat message
Preconditions :	User logged in, at least one chat channel
Parameters :	ID of chat channel, content of message
Test Case :	**Send Multi Chat Invitation**
Description :	To invite all friends into a chat channel
Preconditions :	User logged in, at least one friend
Parameters :	Friend names
Test Case :	**Leave Multi Chat**
Description :	To quit from a chat channel
Preconditions :	User logged in, at least one chat channel
Parameters :	ID of chat channel
Test Case :	**Get My Votings**
Description :	To retrieve user's voting list
Preconditions :	User logged in
Parameters :	-
Test Case :	**Delete Vote**
Description :	To remove a vote
Preconditions :	User logged in, at least one vote
Parameters :	ID of vote
Test Case :	**Get Voting Result**
Description :	To obtain result of vote
Preconditions :	User logged in, at least one vote
Parameters :	ID of vote

Table 3.6: Table of new test cases (1)

Test Case :	**Create Vote**
Description :	To create a vote
Preconditions :	User logged in
Parameters :	Topic, description, options, private or not, minimum choices, maximum choices

Test Case :	**Vote**
Description :	To do a voting
Preconditions :	User logged in, at least one vote
Parameters :	ID of vote, index of choice

Table 3.7: Table of new test cases (2)

Test Case :	**Edit Album**
Description :	To edit album's information
Preconditions :	User logged in, user's albums loaded, at least one album
Parameters :	ID of album, album name, description

Test Case :	**Delete Album**
Description :	To remove a user's album
Preconditions :	User logged in, user's albums loaded, at least one album
Parameters :	ID of album

Test Case :	**Delete Photo**
Description :	To remove a user's photo
Preconditions :	User logged in, user's albums loaded, at least one photo
Parameters :	ID of album, ID of photo

Test Case :	**View Own Albums**
Description :	To browse own albums list
Preconditions :	User logged in
Parameters :	-

Test Case :	**View Friend's Albums**
Description :	To browse albums list of a friend
Preconditions :	User logged in, at least one friend
Parameters :	Friend name

Test Case :	**View Groups List**
Description :	To show user's groups list
Preconditions :	User logged in
Parameters :	-

Test Case :	**Create Group**
Description :	To create a group
Preconditions :	User logged in
Parameters :	Group name

Table 3.8: Table of new test cases (3)

Test Case :	**Invite Friend to Group**
Description :	To invite a friend into a group
Preconditions :	User logged in, at least one group, at least one friend
Parameters :	ID of group, friend name
Test Case :	**View Group**
Description :	To browse a group
Preconditions :	User logged in, at least one group
Parameters :	ID of group, ID of root folder
Test Case :	**Leave Group**
Description :	To quit from a group
Preconditions :	User logged in, at least one group
Parameters :	ID of group
Test Case :	**Create Folder**
Description :	To add a folder in a group
Preconditions :	User logged in, at least one group
Parameters :	Folder name, ID of group, ID of root folder
Test Case :	**Rename Folder**
Description :	To rename a folder
Preconditions :	User logged in, at least one group, at least one folder
Parameters :	Folder name, ID of group, ID of root folder, ID of folder
Test Case :	**View Folder**
Description :	To browse a folder
Preconditions :	User logged in, at least one group, at least one folder
Parameters :	ID of group, ID of folder
Test Case :	**Delete Folder**
Description :	To remove a folder
Preconditions :	User logged in, at least one group, at least one folder
Parameters :	ID of group, ID of root folder, ID of folder
Test Case :	**Upload File in Folder**
Description :	To add a file into a folder
Preconditions :	User logged in, at least one group, at least one folder
Parameters :	ID of group, ID of folder, path to file
Test Case :	**Delete File from Folder**
Description :	To remove a file from a folder
Preconditions :	User logged in, at least one group, at least one folder, at least one file
Parameters :	ID of group, ID of folder, ID of file
Test Case :	**View Forum**
Description :	To browse forum in a group
Preconditions :	User logged in, at least one group
Parameters :	ID of group

Table 3.9: Table of new test cases (4)

Test Case :	**Create Forum Thread**
Description :	To create a thread in forum
Preconditions :	User logged in, at least one group
Parameters :	ID of group, thread name
Test Case :	**Comment Forum Thread**
Description :	To add a comment on a thread
Preconditions :	User logged in, at least one group, at least on thread
Parameters :	ID of group, ID of thread, content of comment
Test Case :	**View Forum Thread**
Description :	To browse a thread
Preconditions :	User logged in, at least one group, at least on thread
Parameters :	ID of group, ID of thread
Test Case :	**View Calendar**
Description :	To browse user's calendar
Preconditions :	User logged in
Parameters :	Current year, current month
Test Case :	**Create Calendar Event**
Description :	To add an event on calendar
Preconditions :	User logged in
Parameters :	Event name, start date, end date, description
Test Case :	**Edit Calendar Event**
Description :	To edit a calendar event
Preconditions :	User logged in, at least on calendar event
Parameters :	ID of event, event name, start date, end date, description
Test Case :	**Delete Calendar Event**
Description :	To remove an event from user's calendar
Preconditions :	User logged in, at least on calendar event
Parameters :	ID of event
Test Case :	**Search**
Description :	To look for users
Preconditions :	User logged in
Parameters :	Criterion for searching

Table 3.10: Table of new test cases (5)

Precondition	Condition
PreconditionCalendarLoaded	the calendar of user is loaded
PreconditionFriendsHavePhoto	at least one friend has photo
PreconditionGroupNotExist	the imported group does not exist
PreconditionGroupsLoaded	the groups list of user is loaded
PreconditionHasInboxMessage	user has at least one inbox message
PreconditionVoteNotExist	the imported vote does not exist
PreconditionVotesLoaded	the votings list of user is loaded

Table 3.11: New preconditions

much easier to create a new test case and ensure the independence of the test case at the same time.

- Second, the logics of generating missing parameters for all the test cases, which are related to those global variable involved plugins, have been uniformed. For every test case, the first phase of parameter generation is always to call *load()* so as to load the newest resources. Then, it chooses proper resources as the demanded parameters, with the help of *get()*. In this case, null pointer exceptions and other unexpected exceptions can be avoided efficiently in the period of parameter generation.
- Third, lots of missing preconditions have been added with the assistance from *detection()*. Previously, the preconditions of many test cases were not complete, since there were not required preconditions. It was also a reason that led to unexpected exceptions. Table 3.11 lists all the new added preconditions. And these preconditions further improves the independence of the test case.

3.4 Metrics

A metric is a measurement scale which represents a certain attribute of the `LifeSocial` network during test. The data of the attribute is periodically collected by a monitoring tool in the runtime. Afterwards, the statistics of the metric can be plotted into a graph which reflexes the quality of a specific aspect of `LifeSocial`. The monitoring mechanism is called SkyEye.KOM, which is described in Chapter 4.
Basically, the most important properties of a metric are its name and value, which are explicitly explained below.

Metric name Normally, the name of a metric consists of four components: *Statistic scope*, *Collection location*, *Measured object* and *Aggregation strategy* respectively. For instance, here is a metric's name:

GLOBAL_STORAGEDISPATCHER_NUM_STORED_REPLICATIONS_COUNT .

- *Statistic scope* indicates the current metric is generated from global data or data of one specific node in the network. Since the monitoring mechanism enables the capacity of aggregating single data of each peer to global data, it is feasible to collect the global statistics as well as local statistics of a node. If a metric's name starts from *GLOBAL_*, like the example above, it is generated from the global statistics; Otherwise, it generated from the statistics of a node locally.

- *Collection location* reveals where exactly the metric is collected. In `LifeSocial` project, the different kinds of metrics are mainly collected in three files during test. They are /com.p2pframework.message.scribe/PastryMessageDispatcher.java, /com.p2pframework.network.freepastry/PastryNetwork.java and /com.p2pframework.storage.past/PastryStorageDispatcher.java. These metrics are prefixed by *MESSAGEDISPATCHER_*, *NETWORK_* and *STORAGEDISPATCHER_* respectively. Evidently, the example metric is collected in *PastryStorageDispatcher*.

- *Measured object* tells what kind of data is measured by the metric. It could be the number of deleted item, the size of used memory or something else. For instance, the object that the example metric measures is the number of stored replications in the `LifeSocial` network. Besides, *Measured object* can be the rate of some special data, such as sent and received messages. Figure 3.13 and 3.14 illustrate how the mechanism of calculating data rate works. As can be seen in Figure 3.13, a newRate is generated in every period of data collection, by calculating with the data value of current period and last period. Then, this newRate will be stored in a rateBuffer. Figure 3.14 shows how the data rate displayed in the final statistics is calculated. Actually it is the average value of those rates in the rateBuffer. The size of rateBuffer is customized and it is set to 2 as default.

- *Aggregation strategy* indicates in which way the monitored data is aggregated. The monitoring mechanism implements a number of aggregation strategies. And in `LifeSocial`, there are four types are used: sum, greatest, mean and smallestNonNegative. The latter three types are represented by *MAX_*, *MEAN_* and *MIN_* respectively. If a metric's name does not contain any of these three, it means the aggregation strategy of this metric is sum, like the example metric.

Thus, the example metric represents the total number of stored replications in

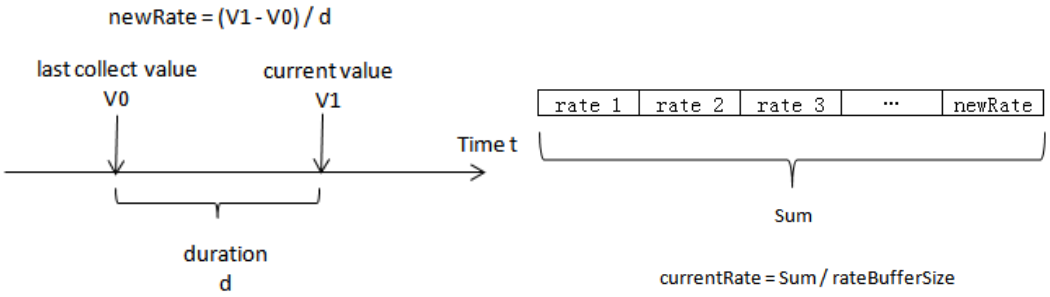


Figure 3.13: Value of newrate [18]

Figure 3.14: Value of currentrate [18]

the `LifeSocial` network. And it is collected in *PastryStorageDispatcher*.

Metric value Metric value is determined by the value of measured object and the time stamp together. Usually, it is displayed in a graph in order to make itself more readable and the evaluation of test result more convenient. Figure 3.15 illustrates the total number of stored replications in a network over time. It can be seen from the graph that the number of stored replications is shown on the y axis and the time stamp is on the x axis. With the graph, it is easy to find out a desired metric value. For example, it is clear in Figure 3.15 that 80 minutes after the test is started, there are more than 20000 stored replications in the network.

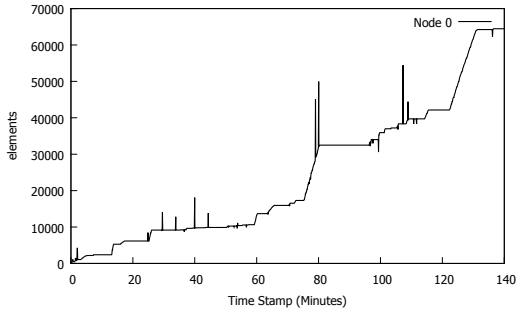


Figure 3.15: Global number of stored replications

3.4.1 New Metrics

Before this master thesis, the `LifeSocial` project has already included various metrics that covered many aspects of `LifeSocial` network, such as the data flow, workload and performance of it. However, the social metrics were not well developed. Only about half of `LifeSocial` plugins had some social metrics for monitoring performed user behaviours. Table 3.12 illustrates the original

Plugin	Social Metrics
FileStorage	stored files
Forum	created folders; created posts; created public groups; created private groups; invite user to group; join group; leave group
Friend	accepted friendship request; sent friendship request; friends list size
LiveChat	send chat messages
Login	registration ; login; logout
Profile	created basic profiles; get own profile; request user profile
Search	search
Wall	view friends wall; wall comments; wall entries;

Table 3.12: Previous social metrics for LifeSocial plugins

status of social metrics for each plugin. In addition, many of these social metrics were not accurate because the corresponding plugins have been changed or redesigned in last year. For example, the metric for measuring search action was inserted in several wrong functions that have been completely abandoned by the *Search Plugin* and replaced by a powerful search engine.

Actually, social metrics record all sorts of actions performed by users in the `LifeSocial` network. Through these metrics, the statistics of each action can be collected so as to analyse user behaviours in the network. Also, social metrics are useful for evaluating the test result by comparing metrics' values with repetitions of the corresponding test cases in test plan.

Therefore, it is necessary to add missing social metrics for both the previously existing plugins and the new plugins built last year, as well as to correct the inaccurate social metrics.

Table 3.13 lists all the current social metrics of `LifeSocial` after enhancement.

Additionally, each of measured object by these social metrics has four types of statistics that are collected with full aggregation strategies, namely sum, greatest, mean and smallestNonNegative. In other words, for any user action, the number of the action executed either in the whole network, or on one certain peer, is monitored. For instance, *GLOBAL_PLUGIN_VOTING_VOTEACTION_COUNT* reveals the total number of vote action in the network over time, and *GLOBAL_PLUGIN_VOTING_VOTEACTION_MIN_COUNT* shows the smallest number of vote action done by one user at each time point.

All the modifications and creations on social metrics are made by W.Peng alone.

Plugin	Social Metrics
Calendar	created events; deleted events; modified events; view calendar
FileStorage	stored files
Forum	created posts; created threads; view forum; view forum thread
Friend	accepted friendship request; sent friendship request; friends list size
Group	added files; created folders; created private groups; created public groups; deleted items; invite user to group; join group; leave group; view folder; view group
LiveChat	chat channels; send chat messages; leave chat channel
Login	registration ; login; logout
Messaging	deleted inbox messages; deleted outbox messages; sent messages ; view inbox; view outbox
Photos	created albums ; deleted albums; uploaded photos; view friend album list; view own album list
Profile	created basic profiles; get own profile; request user profile
Search	search
Voting	created votings; deleted votings; get voting result; view my voting lists; vote action
Wall	view wall; wall comments; wall entries;

Table 3.13: Current social metrics for LifeSocial plugins

3.4.2 Further Improvements

All the improvements in this subsection are designed by W.Peng, after the first round of synthesis and random test.
As described in [18], there are plenty of metrics covering the data flow inside of `LifeSocial` network, which is shown in Figure 3.16.
It can be seen from the figure that the data is transferred among four layers, namely plugin layer, framework layer, overlay layer and the Internet at the outermost. Obviously, the framework and overlay are two most significant layers for monitoring the data flow of `LifeSocial`. To be more exact, the whole data flow can be well measured as long as the data going through framework and overlay is collected by all kinds of relevant metrics. Since the storage dispatcher and message dispatcher are two mainly responsible modules for data transfer in framework, the data flows through these two modules and overlay need to be monitored accurately.
The author of [18] added various corresponding metrics for these three parts and gathered them in *PastryStorageDispatcher*, *PastryMessageDispatcher* and *PastryNetwork* respectively. However, a number of metrics' names did not match the actually measured objects of these metrics, which was really confusing and might adversely affect the evaluation of test result. Besides, a few metrics that monitored abandoned features of the newest `LifeSocial` were still collected. It costed senseless memory consumption and storage space of testing machines.

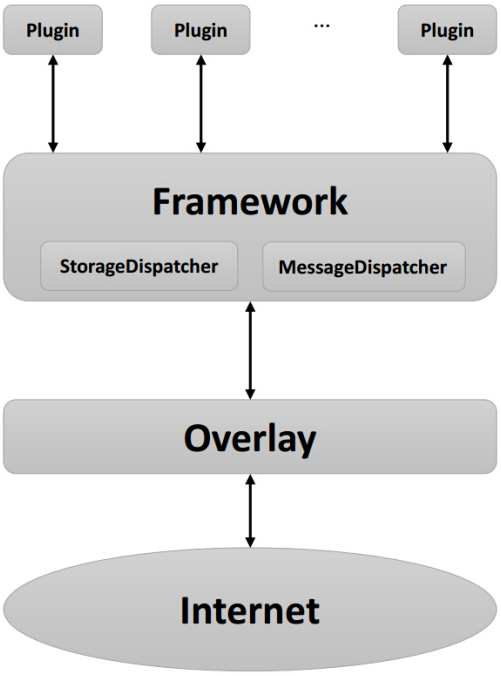


Figure 3.16: Data flow through LifeSocial

Hence, several modifications have been done during the period of this thesis for enhancing these mentioned disadvantages.

- First, all the metrics for information cache have been removed because this feature is no longer used in current `LifeSocial`. Furthermore, some metrics which measure the number of messages sent/received by message dispatcher have been added.

- Additionally, the previous metrics for stored/retrieved data of storage dispatcher in framework layer were named as *STORAGE_DATA_SENT/RECEIVED*, which has been changed to *STORAGE_DATA_STORED/RETRIEVED*. In fact, the size of stored data in the network barely matters how much data is retrieved. In contrast, the global data sent/received by storage dispatchers of all peers in the network is equal, because each data item sent into the network will be received by one or several peers in theory. As a result, the value of total data sent/received by all the frameworks has been modified from the sum of data stored/retrieved by storage dispatcher and data sent/received by message dispatcher into the sum of data sent/received by storage dispatcher and message dispatcher.

- Last but not least, in order to keep the consistency of all metrics and make them easier to understand, the prefix of all the metrics that measure data flow through plugins has been renamed from *FRAMEWORK_* to *STORAGEDISPATCHER_STORAGE_*.

Table 3.14 lists the added and removed metrics. And all the changes of renamed metrics are shown in Table 3.15 and 3.16. Table 3.15 lists old names of these metrics. And the corresponding new names are listed in Table 3.16.

Action	**Metrics**
Add	*STORAGEDISPATCHER_MESSAGE_NUM_SENT*
Add	*STORAGEDISPATCHER_MESSAGE_NUM_RECEIVED*
Delete	*STORAGEDISPATCHER_CACHE_USED_SIZE*
Delete	*STORAGEDISPATCHER_CACHE_TOTAL_SIZE*
Delete	*STORAGEDISPATCHER_CACHE_NUM_ELEMENTS*

Table 3.14: Added and removed metrics

Former name
STORAGEDISPATCHER_STORAGE_DATA_SENT
STORAGEDISPATCHER_STORAGE_DATA_RECEIVED
FRAMEWORK_DATA_SENT_PLUGINNAME
FRAMEWORK_DATA_RECEIVED_PLUGINNAME

Table 3.15: Former names of renamed metrics

Current name
STORAGEDISPATCHER_STORAGE_DATA_STORED
STORAGEDISPATCHER_STORAGE_DATA_RETRIEVED
STORAGEDISPATCHER_STORAGE_DATA_STORED_PLUGINNAME
STORAGEDISPATCHER_STORAGE_DATA_RETRIEVED_PLUGINNAME

Table 3.16: Current names of renamed metrics

3.5 Setup Parameters

For a test, there are various parameters which can more or less influence on the result. By modifying these parameters, therefore, different test environments and configurations can be set up for specific purpose. For example, for a small scale test aiming at verifying some certain functionalities of `LifeSocial`, the number of testing node is not necessary to be as many as that of a formal test, and monitoring function is not needed to use as well.

According to the previous master thesis [18] about testing LifeSocial, there are three types of parameters can be considered for configuring a test, namely *Environment Parameters*, *Social Parameters* and *LifeSocial Parameters*.

3.5.1 Environment Parameters

Usually, *Environment Parameters* are basics for a test. So it should be considered in the first place. *Environment Parameters* contain:

Number of peer This parameter means how many LifeSocial instances are used to join in the test, including master and slave. Theoretically, the more instances participate in the test, the collected testing result is more realistic. In the test pool of Uni-Paderborn, however, there are a total of only 20 machines available for this thesis. And unlike the testing setup in [18], in which several instances were running on every machine, in all following tests, one machine is just allowed to launch one instance, in order to ensure that the performance of each instance is independent. In brief, in the context of this master thesis, one machine is a peer and the maximum number of peer is 20.

Peer performance To a large extent, this parameter depends on the ability of device in which LifeSocial instance is launched. CPU power, memory size and hard disk capacity, any of these factors could influence on the performance of peer. In the real world, a peer can be a powerful server, or a desktop computer, or a personal laptop , or even a portable tablet. In other words, the performance of peers in a real network is heterogeneous. But in this thesis, since every testing machine has the same hardware ability for each test, peer performance is always homogeneous.

Peer bandwidth Similar to peer performance, peer bandwidth is heterogeneous in real networks as well. On the one hand, the quality of Internet from different Internet Service Provider is hardly equal. On the other hand, peer bandwidth could be affected by plenty of factors, such as location, time, and even weather. However, the bandwidth of peers in this thesis is homogeneous because all the testing machines are connected to the same router in Uni-Paderborn via Gigabit Ethernet.

Operating system LifeSocial supports most common operating systems, including Windows, Linux, and Mac OS. And instances on different OS are able to communicate with each other in a same LifeSocial network. In the thesis, Linux is the only OS been used because it is the only OS installed on all the testing machines.

3.5.2 Social Parameters

Social Parameters are related to the status of a social network. *Social Parameters* includes:

Churn Churn indicates that there are users joining and leaving the social network during test, which means churn rate can determine online duration of each user in test. This parameter can be configured by changing the value of *enableRandomLogin* in /eclipse/resources/config.txt. In this thesis, churn is used only in random test.

Application load Application load mainly depends on how many users are active in social network and what behaviours are exactly done by every user. As mentioned in 3.3, most functionalities of `LifeSocial` have corresponding test cases now, which means almost all the common behaviours are covered by these test cases. Thus, application load can be configured in *TestPlan* by assigning a variety of test cases and their repetitions.

3.5.3 LifeSocial Parameters

LifeSocial Parameters affects test result directly because they are inserted right in the code. These parameters can be configured in /org.lifesocial.core.utilities/config/LifeSocialConfig.txt. The current configuration of *LifeSocial Parameters* are shown in Figure 3.17 and 3.18.

autoDeleteStorage This parameter determines whether the old storage of a user should be removed. Last year, the P2PFramework Project Group 2013 implemented a function to delete the storage of a user every time when the user is logging into the `LifeSocial` network, unless the user is the only peer in the network. In normal cases, this parameter should be enabled. But in random test, autoDeleteStorage is disabled in order to avoid unexpected data lost due to users' frequent login and logout in the test.

enableHeterogeneityAwareness This parameter controls the heterogeneity of the `LifeSocial` network. If it is enabled, the network will assign distinct workload to different peers based on their capacity and bandwidth information. In this way, the resources of the network can be optimized. Nevertheless, this parameter is disabled all the time throughout the thesis because either capacity or bandwidth of these testing machines is homogeneous.

heterogeneityCacheSize This parameter specifies the size of cache assigned

to the heterogeneity function. In this thesis, since the heterogeneity function is not used, heterogeneityCacheSize does not make any impact on test result .

CanYouProvideTimeOut This parameter defines the time limit for a *CanYouProvideMessage* related to the heterogeneity function. Same like the previous one, this parameter has no effect on test result as well.

providerListLength This parameter determines the number of providers stored in every responsible node, which is also related to the heterogeneity function. Therefore, providerListLength does not matter to this thesis because enableHeterogeneityAwareness is disabled.

replicationFactor This parameter specifies the number of replicas for every data item in the `LifeSocial` network. Ideally, increasing replicationFactor can enhance the safety of a data item in the network, and also improve the efficiency to retrieve the data item. However, the maintenance cost of periodically updating the data information to every replicas node cannot be ignored. Considering the maximum number of peer is 20 in this thesis, this parameter is set to 3.

objectSizeCalculationEnabled This parameter determines whether the `LifeSocial` network calculates the size of object passing through PAST layer or not. Based on the experience of [18], enabling this parameter will waste a lot of resources and cause delays in the network. So objectSizeCalculationEnabled is always disabled in this thesis.

randomSelectionModeEnabled This parameter determines in which way a provider is chosen from the list, by capacity or just randomly. As the enableHeterogeneityAwareness is disabled, this parameter does not impact on test result.

measuringData This parameter is used for monitoring the data passing through Plugin layer and Framework layer, and calculating the size of the data. In order to collect more comprehensive monitoring statistics to evaluate test result, this parameter is enabled throughout this thesis.

socialMetricCollection This parameter is for collecting social metrics which are distributed in all the plugins of `LifeSocial`. During the period of this thesis, there are numerous new social metrics added to measure the amount of every type of behaviour happened in the network. These social metrics can be found in Table 3.13 in last section. Same like the parameter measuringData, socialMetricCollection is always enabled to get more monitoring data during test.

autoStartMonitoring This parameter determines if the monitoring function is automatically started when a peer is launched. In common cases, it is not necessary to enable this parameter because it will increase memory consumption of `LifeSocial`. Also, user can start the monitoring manually when the monitoring data is needed. But it is obviously not the case for this thesis. All the monitoring statistics from the beginning to the end of test is required for a complete evaluation. So autoStartMonitoring is enabled for all the tests in this thesis.

combinedParameter This parameter specifies the number of rounds which are combined for reducing the fluctuation of monitoring data. The detail of this mechanism is described in [18]. And the value of this parameter is kept to 3.

autoFriendRequestAccept This parameter determines whether a friend request is automatically accepted in the `LifeSocial` network. Normally, it should be disabled. In this thesis, however, it is impossible to accept friend request for every peer manually during test. Moreover, friendship is a precondition of most test cases. So this parameter needs to be enabled for all the tests.

autoLiveChatInvitationAccept This parameter determines whether a live chat invitation is automatically accepted in the `LifeSocial` network. It should be disabled in common cases. But, considering live chat invitation cannot be accepted manually during test, this parameter is enabled in this thesis.

autoVotingInvitationAccept This parameter determines whether a voting invitation is automatically accepted in the `LifeSocial` network. Same reason with autoLiveChatInvitationAccept, this parameter is always enabled in this thesis.

onlyGnuplot This parameter controls the collection of monitoring data. If it is enabled, `LifeSocial` will only generate one type of monitoring statistic file that is the only type useful to plotting the monitoring data. Otherwise, `LifeSocial` will generate other two kinds of monitoring statistic files as well. Since these two types of files are unnecessary for the thesis, this parameter is enabled for saving storage space of testing machines.

monitorDataVisual This parameter determines if the monitoring statistic can be visualized at runtime. As enabling this parameter will lead to large memory consumption and affect the stability of `LifeSocial`, it is always disabled in this thesis.

bulletinURL This parameter specifies a url where the content of `LifeSocial's` bulletin board can be customized.

```
# auto storage deletion before accessing into network (1 = enabled,
# 0 = disabled)
autoDeleteStorage = 0

# enables the heterogeneity awareness (1 = enabled, 0 = disabled)
enableHeterogeneityAwareness = 0

# size of the heterogeneitycache in MB
heterogeneityCacheSize = 1

# timeout for a canyouprovidemessage in ms
CanYouProvideTimeOut = 5000

# maximal number of providers for each data object
providerListLength = 1

# number of replications of a data object in the system
replicationFactor = 3

# enables the object size calculation for the monitoring
# (1 = enabled, 0 = disabled)
objectSizeCalculationEnabled = 0

# enables the random selection mode (1 = enabled, 0 = disabled)
randomSelectionModeEnabled = 0

# enables the measuring data function, the data measured located in Plugin,
# Framework levels (1 = enabled, 0 = disabled)
measuringData = 1

# enables the collecting of social metrics, such as friend requests
# (1 = enabled, 0 = disabled)
socialMetricCollection = 1

# auto start monitoring (1 = enabled, 0 = disabled)
autoStartMonitoring = 1

# in monitoring, how many rounds you want to combine
combineParameter = 3
```

Figure 3.17: LifeSocial parameters (1)

```
# enables the auto friend request accepted (1 = enabled, 0 = disabled)
autoFriendRequestAccept = 0

# enables the auto live chat invitation accepted (1 = enabled, 0 = disabled)
autoLiveChatInvitationAccept = 1

# enables the auto voting user invitation accepted (1 = enabled, 0 = disabled)
autoVotingInvitationAccept = 1

# only create GNUPlot monitoring data file (1 = enabled, 0 = disabled)
onlyGnuplot = 1

# visualization of monitored data in runtime, this function will keep the
# memory usage increasing! (1 = enabled, 0 = disabled)
monitorDataVisual = 0

# url to the bulletin board
bulletinURL = a url
```

Figure 3.18: LifeSocial parameters (2)

4 Monitoring

This chapter presents overview of monitoring mechanism on structured P2P systems. Section 4.1 explains the monitoring challenges involved in P2P systems and section 4.2 explains the SkyEye.Kom monitoring mechanism. This whole chapter is written by author D.Balabhadruni.

4.1 Monitoring

P2P systems need to meet the following challenges in order to provide the constant quality in the system. First challenge is peers heterogeneity, the peers connection and capacity are different to each other. The second challenge consists of scenarios like distributed storage, content delivery, and discovery and contacting of users which can not be controlled in a centralized way. Third challenge is, there are many uncontrollable elements in P2P system for example churn, network size. The goal of the monitoring is not only to gather the quality metrics but also provide the constant quality in P2P system. An information architecture needs to be established on top of the structured P2P overlay to gather information from the peers and aggregate the values to retrieve the global view of the metrics.

4.2 SkyEye.Kom

SkyEye.KOM is a monitoring mechanism used to monitor the whole P2P network. In SkyEye.KOM, a tree topology is formed on the top of the structured P2P overlay to gather all information of the network. At first, a *Domain ID* is assigned to a peer when it joins into the network, next based on the *Domain ID* each peer calculates its position and send its own metrics to the parent node. The parent node aggregates its own metrics and the metrics received from the child nodes and forwards it to the upper node, this process is continued on all the child and parents nodes in the network, and finally overall data from the parents nodes are aggregated to the root node. Then the root node aggregates its own metrics and received data and finally sends the global metrics to the descendants step-by-step. For consistent and reliable routing in the tree

structure, a peer must detect the dead or failure nodes when it is sending the metrics.

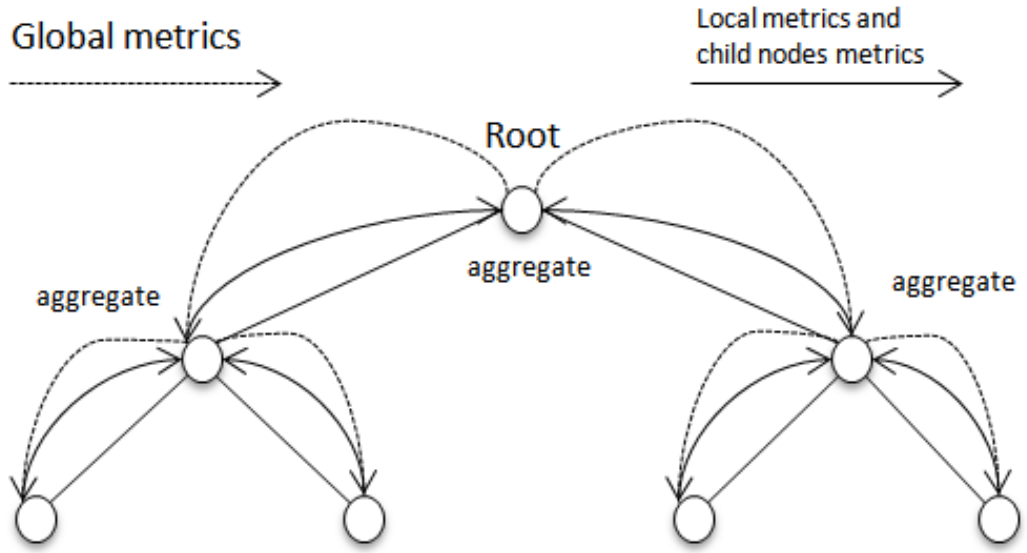


Figure 4.1: Message routing in SkyEye.KOM layer

In figure 4.1 we can see that there are three layers in the tree structure. The global metrics are shown in the dotted lines and local as well as child node metrics are shown in solid line. The local and child metric values are sent to the root node, the root node aggregates the received values and its own value and send the global metrics value to all child nodes in the network. The aggregation strategy is shown in the table 4.1. In P2P system, the system statistics means the statistics that are generated periodically from the multiple metrics. To get the system statistics all the nodes must continuously send the metric updates and these metric updates are aggregated and forwarded later. Every node checks its position and its parents position before sending the aggregated data. All the metrics updates are acknowledged in the monitoring mechanism.

Function	Description
SUM	Get the sum of all values of a metric
MEAN	Get the mean value of all values of a metric
MAX	Get the maximum value of all values of a metric
MIN	Get the minimum value of all values of a metric
SUMOFSQ	Get the sum of the squares of a metric
COUNT	Counts the number of metric values
AVG	Gets the average value of a metric
STDDEV	Gets the standard deviation of a metric

Table 4.1: Aggregation strategies

In figure 4.2 we can see the structure of the tree. There are 7 nodes in the tree structure, each node is assigned with a domain ID and the maximum domain interval is [0,1]. The domain ID of the each node is the middle value in the domain interval. The node 0.75 is responsible two domain ID's 0.625 and 0.875 in the tree. If the messages are sent up the parent node 0.75 receives the data from the 0.625 and 0.875 nodes. When data is received the parent node 0.75 will send only one message to the highest node in the tree i.e, root node 0.5. Here, we can notice that the communication is only between the child node and its parent node.

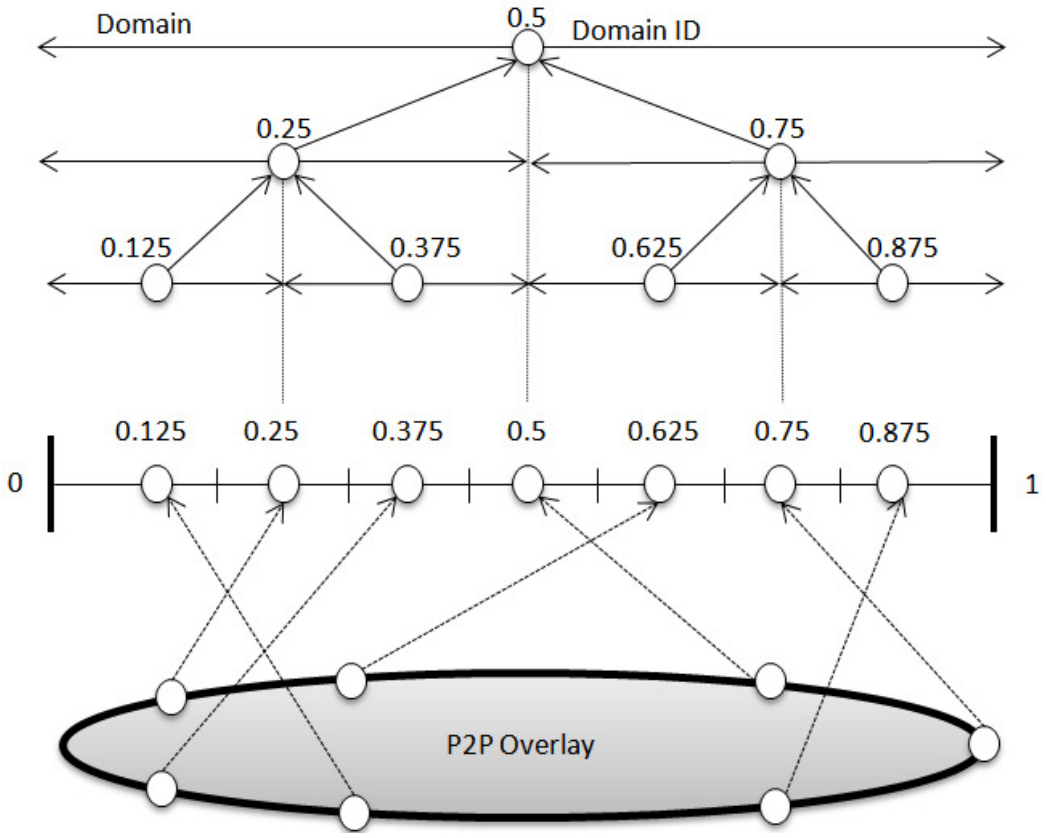


Figure 4.2: SkyEye.KOM ID Domain

5 Synthesis Test

In this chapter, the hardware and software environments of synthesis test, all kinds of deployments for the test, and the test plan of it are introduced in section 5.1, 5.2 and 5.3 respectively. Also, the collection and processing of monitoring data after synthesis test are described in section 5.4. In section 5.5, the result of the first round of synthesis test is analysed and evaluated. Based on the test result, some modifications are made and introduced in section 5.6. In the end, the result of the second round of synthesis test is evaluated and compared in section 5.7.
This whole chapter is written by W.Peng. The test plan of synthesis test is designed by two authors together. The two synthesis tests in the first round are also performed by both of authors and the two tests in the second round are performed by W.Peng alone. All the evaluations on the test results are contributed by W.Peng.

5.1 Test Environment

The test pool for synthesis test is in Heinz-Nixdorf-Institute(HNI) of Uni-Paderborn. There are 20 testing machines in the area and the hardware and software information of these machines is below:

- Processor : Intel i7-2600
- RAM : 8 GB
- Operating System : openSUSE 8.32.5-x86_64
- Network Interface : Gigabit Ethernet
- Java VM : 1.7.0_51 build 24.45-b08

5.2 Deployment

This section describes the basic requirements for running a synthesis test in the first place. After that, it presents how to set up configurations of synthesis test, and what is required to do before the test as well.

Step 1: Fulfill requirements

To obtain and synchronize the code of `LifeSocial` project, a git tool needs to be installed on every testing machine. *Smartgit* is the chosen one in this master thesis. And *Eclipse Kepler* for RCP and RAP developers is installed on each testing machine as well, in order to launch the `LifeSocial` application. Additionally, the *resources* folder under com.sign.plugin.testplugin in git repository needs to be copied into eclipse directory.

Step 2: Configure parameters

As introduced in Section 3.5, there are three types of parameters involved in synthesis test.

- *Environment Parameters* are controlled by performer of the test. Considering the test environment, however, number of peer is the only configurable environment parameter for synthesis test, because the other parameters cannot be changed, such as peer bandwidth. In this thesis, the number of peer is set to two different values, 10 and 20, so that to evaluate the performance of `LifeSocial` in different network scales.
- *Social Parameters* are determined by /eclipse/resources/Config.txt. First, the test plugin needs to be enabled in this file. Also, the IP address of the master node needs to be specified there. And the path to *timedTestplan* can be a local path or a URL. In this thesis, it is set to a URL that points to a file in *Dropbox* of author. With the URL, it is much more convenient to synchronize the *timedTestplan* on all the testing machines. Figure 5.1 presents the contents of Config.txt for synthesis test.
- *LifeSocial Parameters* largely impact on the performance of `LifeSocial` in synthesis test. These parameters are configured in /org.lifesocial.core.utilities/config/LifeSocialConfig.txt. For collecting required monitoring data automatically, *measuringData*, *socialMetricCollection* and *autoStartMonitoring* need to be enabled. Besides, *autoFriendRequestAccept*, *autoLiveChatInvitationAccept* and *autoVotingInvitationAccept* are enabled because these invitations cannot be accepted manually during the test. The exact values of these parameter can be found in Figure 3.17 and 3.18.

Step 3: Delete old data

Normally, once `LifeSocial` is launched, there are two types of files generated on testing machine, including storage files and monitoring statistics. The storage files manages the information about stored items in the network. And

```
# IP address of master node (also used as bootstrap server)
server=131.234.62.27

# Enables the TestPlugin. Values: true | false
enableTestPlugin=true

# Enables the random slave login mode: true | false
enableRandomLogin=true

# If the random slave login mode is not enabled, you can login
# the specific user you want: <username,password>
user=tester,123

# Slaves can logout according to an online duration distribution
# and log in again with another username
# In random mode only. Values: true | false
enableAutomaticLogout=true

# Path to testplan. Either starting with http:// or file:/
testplan=https://dl.dropboxusercontent.com/.../testplan.txt

# Path to a testplan with time slots for execution.
# Either starting with http:// or file:/
timedTestplan=https://dl.dropboxusercontent.com/.../timedTestplan.txt

# Path to a start script, that is executed when a new user registers
# in random mode.
startScript=https://dl.dropboxusercontent.com/.../startScript.txt

# Maximum time to wait after TestCase has been executed (in ms)
maxSleepTime=5000

# Minimum time to wait after TestCase has been executed (in ms)
minSleepTime=2000

# Time for Wait command (in ms)
waitTime=6000

# Time for startup delay (in ms)
startupDelay=5000
```

Figure 5.1: Parameters in Config.txt

the monitoring statistics affects evaluation of the test result directly. Therefore, it is necessary to remove the storage and monitoring data of previous tests before running a new synthesis test. The deletion of old storage and monitoring data is implemented by two shell scripts, which can be found in /com.p2pframework.tester/Shell_Scripts.

5.3 Test Plan

As mentioned in Section 3.1.2, the master node initiates a timed test plan and sends it to the salves in same network. The timed test plan contains series of test cases arranged by the authors, as well as the repetition and time constrain of each test case. For example, a line of timed test plan is shown below:

5x TestCaseCreateCalendarEvent 2m

It means that *TestCaseCreateCalendarEvent* is required to be executed for 5 times in 2 minutes. If 5 times of the test case are finished in 2 minutes, the responsible slave will wait until the 2 minutes is up; If the 5 times are not accomplished when the 2 minutes is up, the remaining repetitions will be skipped. In this case, the progresses of all salves executing the test plan are kept synchronized.

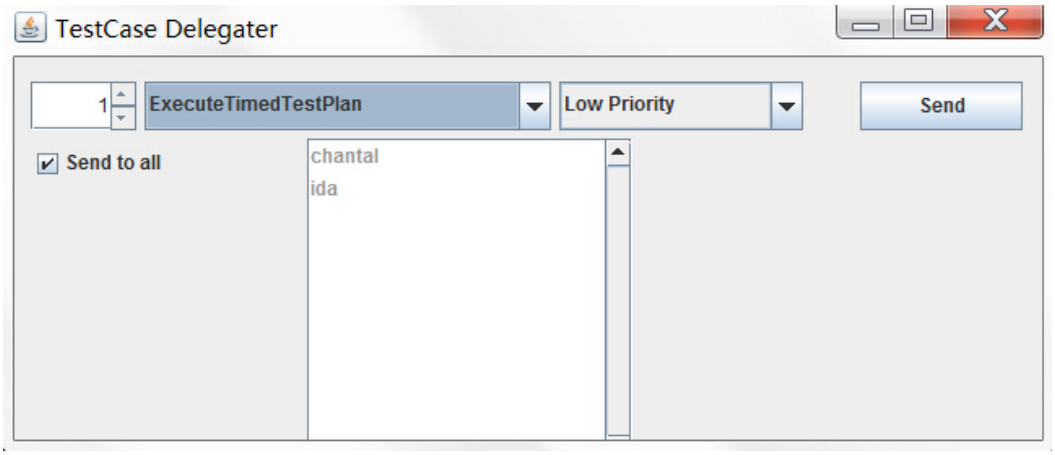


Figure 5.2: TestCase Delegater

And there are two ways to send the command of executing timed test plan to slaves. First, the command *ExecuteTimedTestPlan* can be sent via the TestCase Delegater, which is presented in Figure 5.2. The delegater is only available on master node, and it enables the master to send a variety of commands to chosen slaves easily. In addition, *ExecuteTimedTestPlan* also can be send to all slaves

through the eclipse console of master node, in case the delegater is unavailable under some unexpected conditions.
In [18], the author mentioned a problem caused by *TestCaseSendFriendRequest*. When this test case was executed by a number of slaves at the same time, some of the slaves were probably stuck. The reason is that each slave will ask for a list of online users from the master when the slave starts executing *TestCaseSendFriendRequest*, but the master cannot deal with so many requests in a short time. So, in order to avoid from this potential issue, a command *SendFriendRequestInOrder* has been created, which enables the master to send a *TestCaseSendFriendRequest* to all the slaves separately with a time interval. This command should be called in front of *ExecuteTimedTestPlan* in synthesis test because friendship is a precondition of many test cases.
The goal of synthesis test is to examine various functionalities of `LifeSocial` and evaluate the performance of the network. Based on it, the timed test plan of synthesis test is designed to contain majority of available test cases, which is shown in Figure 5.3 and 5.4.
As can be seen, the whole test plan are supposed to last for more than 130 minutes. For testing as many features of `LifeSocial` as possible, most of available test cases mentioned in Section 3.3 are arranged in the test plan. And there is always a *Wait* after each functional test case so as to release unused memory.
TestCaseBlockUser is not included in the test plan in case that the executing instance has only one friend. If a user has no friend, *TestCaseSendFriendRequest* will be called by the precondition of certain test case, which means the possibility that some instances get stuck during the test will increase. Since the functionality of blocking user is not that vital, it is not worth the risk.
Besides, some of test cases related to Group Plugin are not used in synthesis test. *TestCaseInviteFriendtoGroup* might lead to access issue, which is difficult to control in the test. *TestCaseDeleteFilefromFolder* and *TestCaseDeleteFolder* will probably destroy several instances during the test, due to some unclear reasons. So, these three test cases are not arranged in the test plan as well.

```
1x TestCaseChangeProfilePicture 1m
1x Wait 1m
1x TestCaseChangeProfileInformation 1m
1x Wait 1m
1x TestCaseViewOwnProfile 1m
1x Wait 1m
1x TestCaseViewFriendsProfile 1m
1x Wait 1m
5x TestCaseSendWallPost 2m
1x Wait 1m
5x TestCaseCommentWallPost 2m
1x Wait 1m
5x TestCaseViewOwnWall 2m
1x Wait 1m
5x TestCaseViewFriendsWall 2m
1x Wait 1m
10x TestCaseSendMessage 3m
1x Wait 1m
5x TestCaseViewOutboxMessage 2m
1x Wait 1m
10x 5x TestCaseViewInboxMessage 2m
1x Wait 1m
2x TestCaseDeleteInboxMessage 1m
1x Wait 1m
1x TestCaseSendMultiChatInvitation 1m
1x Wait 1m
10x TestCaseSendMultiChatMessage 3m
1x Wait 1m
1x TestCaseLeaveMultiChatMessage 1m
1x Wait 1m
5x TestCaseSearch 2m
1x Wait 1m
1x TestCaseViewCalendar 1m
1x Wait 1m
5x TestCaseCreateCalendarEvent 2m
1x Wait 1m
2x TestCaseEditCalendarEvent 1m
1x Wait 1m
2x TestCaseDeleteCalendarEvent 1m
1x Wait 1m
1x TestCaseGetMyVotings 1m
1x Wait 1m
10x TestCaseCreateVote 3m
1x Wait 1m
10x TestCaseVote 3m
1x Wait 1m
5x TestCaseGetVotingResults 2m
1x Wait 1m
2x TestCaseDeleteVote 1m
1x Wait 1m
```

Figure 5.3: Timed test plan (1)

```
5x TestCaseCreatePhotoAlbum 2m
1x Wait 1m
50x TestCaseUploadPhoto 10m
1x Wait 1m
5x TestCaseViewOwnAlbums 2m
1x Wait 1m
5x TestCaseViewFriendsAlbums 2m
1x Wait 1m
10x TestCaseViewFriendsPhoto 3m
1x Wait 1m
5x TestCaseDeletePhoto 2m
1x Wait 1m
1x TestCaseViewForum 1m
1x Wait 1m
5x TestCaseCreateForumThread 2m
1x Wait 1m
5x TestCaseCommentForumThread 2m
1x Wait 1m
5x TestCaseViewForumThread 2m
1x Wait 1m
10x TestCaseCreateFolder 3m
1x Wait 1m
2x TestCaseViewGroup 1m
1x Wait 1m
2x TestCaseRenameFolder 1m
1x Wait 1m
50x TestCaseUploadFileinFolder 10m
1x Wait 1m
5x TestCaseViewFolder 2m
1x Wait 1m
1x TestCaseLeaveGroup 1m
1x Wait 1m
```

Figure 5.4: Timed test plan (2)

5.4 Monitoring Data Processing

During synthesis test, monitoring statistics consisted of all sorts of metrics are periodically collected and stored on testing machines locally. When the test is finished, the monitoring data on all testing machines needs to be gathered together at first. There is a shell script for gathering all the monitoring data files into a folder on master node, and it can be found in /com.p2pframework.tester/Shell_Scripts.
Furthermore, there are three java files provided to process the monitoring data.

- OnlyPlotOneNodeGlobal.java just chooses the largest one among all the monitoring data files, and generates a .plt file based on the selected data, for plotting graphs of all global metrics. Apparently, the plotted graphs only displays the statistics collected by the node that has the biggest monitoring data.
- getAll.java generates a .plt file according to all the monitoring statistics, in order to plot graphs of both local and global metrics measured on every node. And every plotted graph contains the data of certain metric collected by each node. As a result, the spent time on plotting all the graphs is much longer than the first option.
- getGlobalDat.java does not generate a .plt file, instead, it generates the global statistics based on the local monitored statisitcs of each instance, since the function for aggregating statistics in the network is not stable sometimes. This file provides an alternative way to obtain the global data manually. But normally, it is not used because it costs a lot of memory consumption.

Usually, either of OnlyPlotOneNodeGlobal.java and getAll.java is needed to deal with the monitoring data of the test. In this master thesis, OnlyPlotOneNodeGlobal.java is the chosen one, since it is enough for result evaluation and it spends less time than getAll.java.
Finally, the generated .plt needs to be executed. The only requirement for executing .plt file is GNUPlot runtime environment is installed. If it is met, the .plt file can be simply executed by a double click. Afterwards, the graphs plotting will take a long time.

5.5 First Round Result Evaluation

As mentioned in 5.2, there are two tests, with 10 and 20 instances respectively, performed as the first round of synthesis test. In this section, the results of the first round synthesis tests are evaluated.

Figure 5.5 and 5.6 show the total number of nodes in the network during these two tests. It can be seen clearly that in either test, after all nodes gradually join the network at the very beginning, the number of nodes is quite stable at 10 or 20 until the end of test. It means there is no node crashed or lost connection to the network during the whole test.

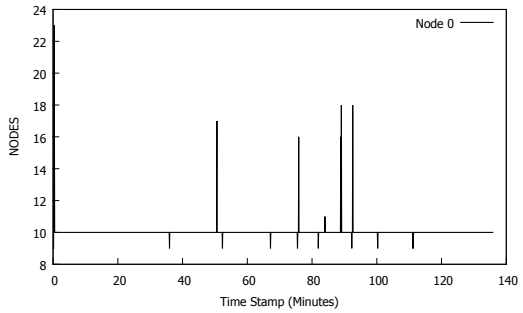


Figure 5.5: Global Nodes - 10 instances

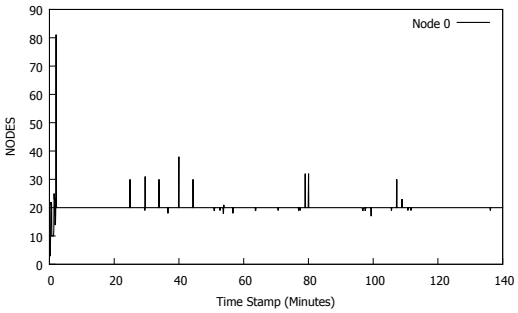


Figure 5.6: Global Nodes - 20 instances

Figure 5.7 presents the average time that a user joins the network and 5.8 tells the average time spent on registration in the 10-instance synthesis test. And Figure 5.9 and 5.10 show these two average time in the 20-instances test. As we can see, in both tests, the join network time is less than 0.3 second, and the registration time is much more than the join network time. The reason is that the registration time includes the join network time, which is used for connecting to the network, and the time to register a user account and log into the network. In other words, the time used on registration and login is more than 10 seconds in total, which is too long for a social network. Therefore, these functionalities should be well examined and improved in future.

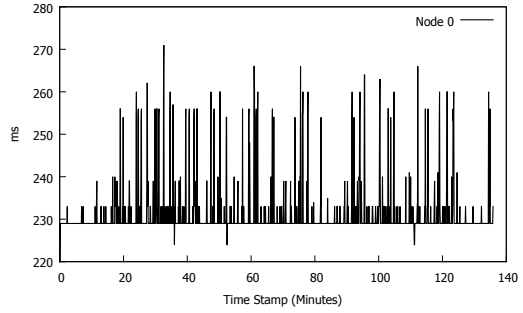


Figure 5.7: Join Network Time - 10 instances

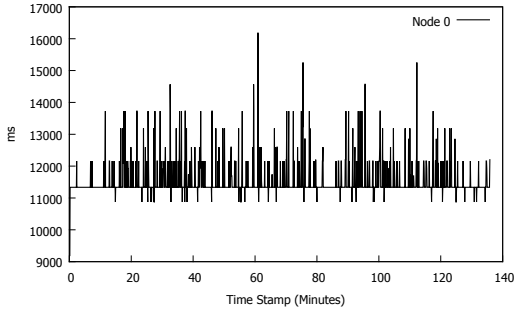


Figure 5.8: Registration Time - 10 instances

Figure 5.11 and 5.12 show the total numbers of messages that are sent and received by MessageDispatcher in 10-instance synthesis test separately. And

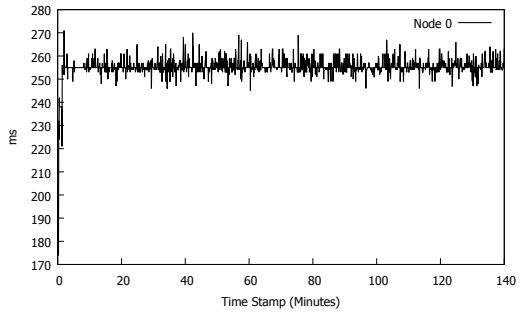


Figure 5.9: Join Network Time - 20 instances

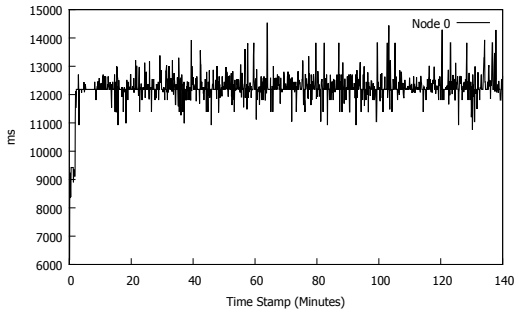


Figure 5.10: Registration Time - 20 instances

the sent and received messages by MessageDispatcher in 20-instance test are shown in Figure 5.13 and 5.14. It is obvious that, for each test, the sent and received messages have the same amount and curve, which means there is no lost message and network delay during the synthesis test. In every figure, although the number of messages is growing all the time, there are three dramatical ascends at around 10, 60 and 120 minutes respectively. The three ascends are caused by the test cases related to Wall Plugin, Voting Plugin and Groups Plugin, since some functionalities of these three plugins needs a lot of message exchanging. For instance, *TestCaseViewOwnWall* requires to retrieve all the wall entries and attached comments of the user. If the number of wall entries or comments is relatively big, there will be a considerable amount of messages generated.

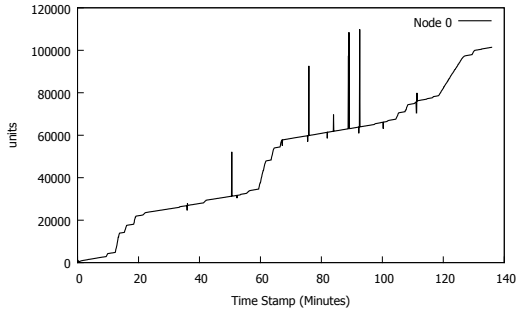


Figure 5.11: Global MessageDispatcher Sent Messages - 10 instances

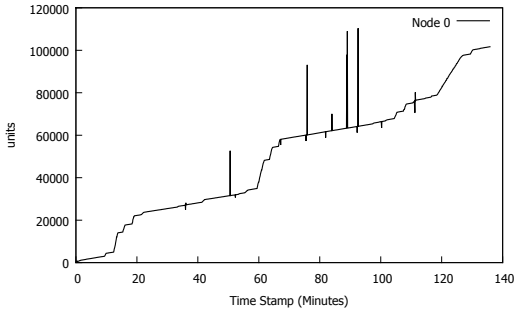


Figure 5.12: Global MessageDispatcher Received Messages - 10 instances

In addition, Figure 5.15, 5.16 and Figure 5.17, 5.18 illustrate the size of messages that are sent and received by MessageDispatcher in 10-instance and 20-instance synthesis test respectively. Actually, these metrics monitor the mes-

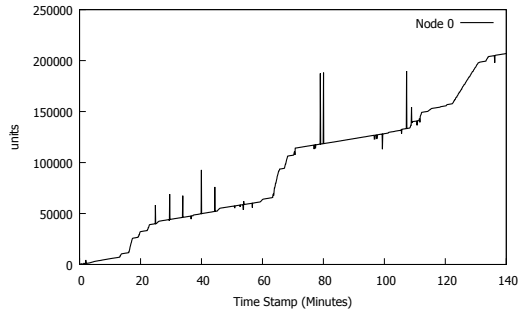


Figure 5.13: Global MessageDispatcher Sent Messages - 20 instances

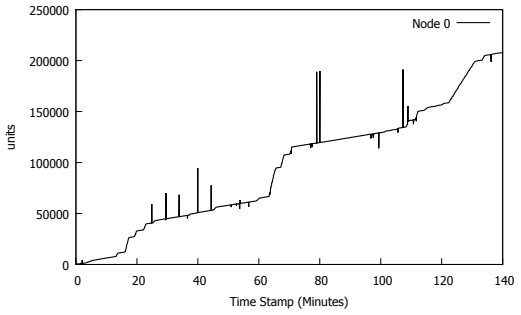


Figure 5.14: Global MessageDispatcher Received Messages - 20 instances

sages sent and received by MessageDispatcher as well. The only difference is that the measured value here is not the number, but the size of messages. It can be found that the curve in any of these four figures is growing gradually and mildly. And there is no apparent increase during the whole test, which means those dramatically increased messages occupy only a very small part of the total size of sent and received messages. In practice, the biggest part contributed to the total size is monitoring messages, which are generated by every node in every 5 seconds. Since a monitoring message contains the values of all the metrics, the size of a monitoring message is about 28 kbytes, and it is much larger than a normal message.

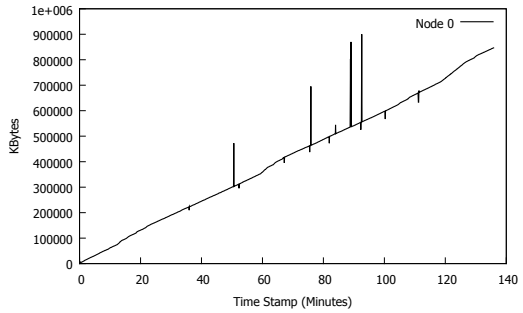


Figure 5.15: Global MessageDispatcher Sent Data - 10 instances

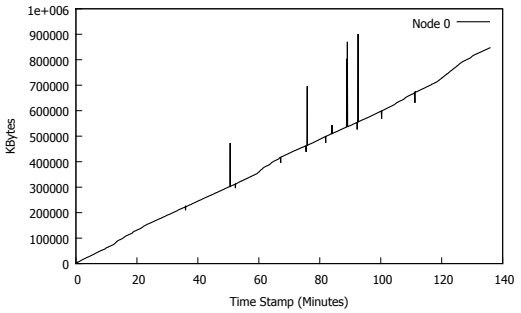


Figure 5.16: Global MessageDispatcher Received Data - 10 instances

Figure 5.19 and 5.20 show the global sent messages and received messages through the `LifeSocial` network in 10-instance synthesis test, and Figure 5.21 and 5.22 show global sent and received messages in 20-instance test. Same with the MessageDispatcher sent and received messages, the values of network sent and received messages are totally equal. Also, the curves of network sent

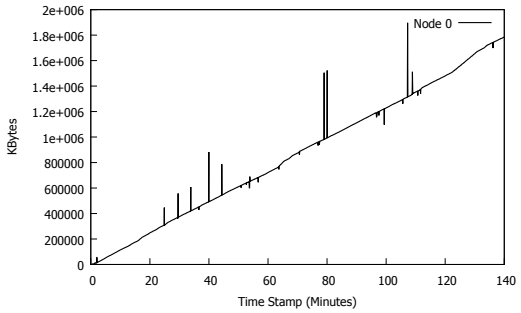


Figure 5.17: Global MessageDispatcher Sent Data - 20 instances

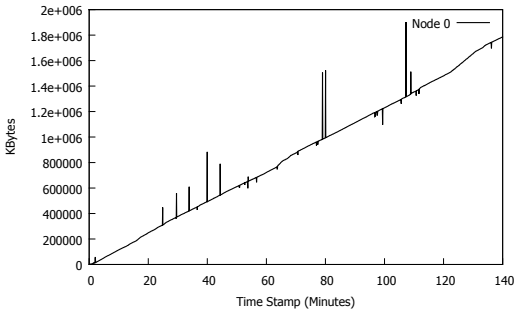


Figure 5.18: Global MessageDispatcher Received Data - 20 instances

and received messages are very similar to the curves of MessageDispatcher sent and received messages, namely there are three rapid ascends at the time point of 10, 60 and 120 minutes. Nevertheless, the exact numbers of messages that pass through network and MessageDispatcher are different. In fact, the network messages are mainly contributed by two types of messages. One type is the messages controlled by the MesssageDispatcher in framework layer, for exchanging information between various plugins and measuring all sorts of metrics. The other type is the messages generated by overlay layer, for looking up stored item, verifying nodes' activity, and periodically updating the routing data among all the nodes. As can be seen, there are 110000 messages received by MessageDispatcher, and 690000 messages received by the network at the end of 10-instance synthesis test. In other words, there are just about 15% messages passing through the network are contributed by MessageDispatcher. And in the 20-instance test, this percentage is close, around 14%, which means the network scale is not a significant factor that affects this message percentage. Actually, the major factor influences the percentage between messages through MessageDispatcher and network is the intensity of test plan. If the repetitions of test cases in the timed test plan are increased and the time bounds are kept same, the messages through MessageDispatcher will take more percentage on the network messages.

Figure 5.23 and 5.24 show the sent and received data through StorageDispatcher in the 10-instance synthesis test. And Figure 5.25 and 5.26 show the data in the 20-instance test. Unlike the previous figures, in either 10-instance or 20-instance test, the curves of Storage sent and received data are a little bit similar but different. Moreover, the sizes of sent and received data are not equal either. In theory, if there is no item lost in the network, the data sent and received by StorageDispatcher should be equal. So, there are two possibilities regarding the different sent and received data. One is there exists data lost because the quality of the network is not stable during synthesis test; The other

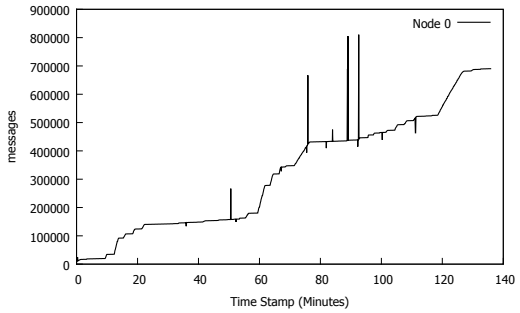


Figure 5.19: Global Network Sent Messages - 10 instances

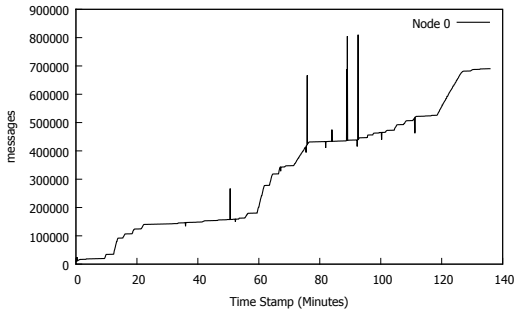


Figure 5.20: Global Network Received Messages - 10 instances

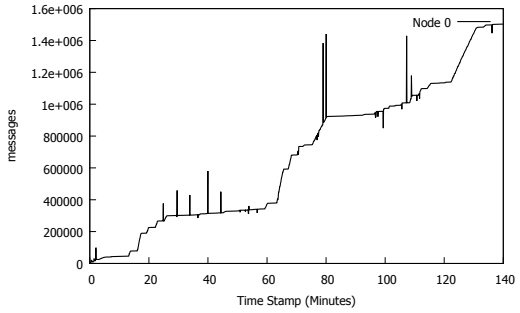


Figure 5.21: Global Network Sent Messages - 20 instances

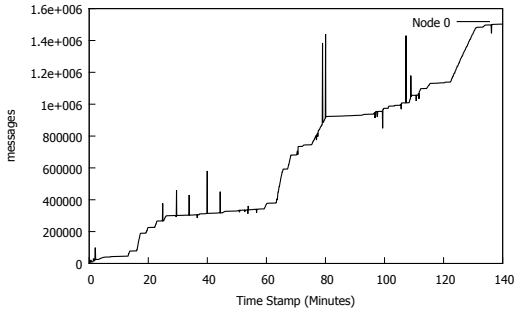


Figure 5.22: Global Network Received Messages - 20 instances

one is the metrics are monitoring wrong objects that are not actual sent and received data through StorageDispatcher, due to the mistakes when the metrics were created. By careful examination, the second possibility is confirmed. These two metrics are not measuring the sizes of sent and received data, but the sizes of stored and retrieved data through StorageDispatcher instead. In this case, the difference between these figures can be well understood.
Actually, Figure 5.23 and 5.25 show the sizes of data stored by StorageDispatcher in the 10-instance and 20-instance tests. It can be seen that there are one slight and two sharp increases in both figures. These increases happens at around 20, 70 and 120 minutes, which are caused by *TestCaseSentMessage*, *TestCaseUploadPhoto* and *TestCaseUploadFileinFolder* respectively. As shown in the test plan of synthesis test(Figure 5.3 and 5.4), at the time point of 21 minutes, 10 *TestCaseSentMessage* start to be executed. However, they do not generate too much stored data because the size of every single sent message is very small. On the contrary, 50 times of *TestCaseUploadPhoto* are executed from 71 to 80 minutes, and contribute lots of sent data, since the size of a photo is much bigger than a normal message. Besides, the repetition of *TestCaseUploadPhoto* is much larger than other test cases as well.
Similarly, Figure 5.24 and 5.26 show the sizes of data retrieved by StorageDispatcher in 10-instance and 20-instance tests in fact. As can be seen, from 12 to 19 minutes, there are three slight ascends in the figures. These slight ascends are caused by *TestCaseCommentWallPost*, *TestCaseViewOwnWall* and *TestCaseViewFriendsWall* respectively, since each of these three test cases requires to retrieve existing wall entries and relevant comments of certain user. Same to some test cases related to Voting Plugin, which are executed from 58 to 66 minutes, they also need salve to retrieve the information of certain vote at first, then slave can perform further actions.

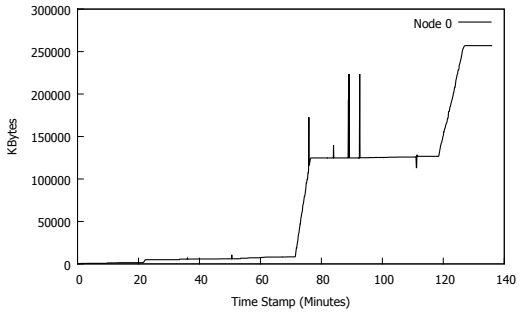


Figure 5.23: Global StorageDispatcher Sent Data - 10 instances

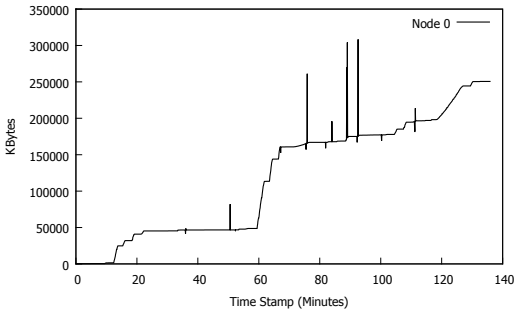


Figure 5.24: Global StorageDispatcher Received Data - 10 instances

Figure 5.27 shows the total used storage size of the network in the 10-instance synthesis test. And Figure 5.28 shows the used size in the 20-instance test. It

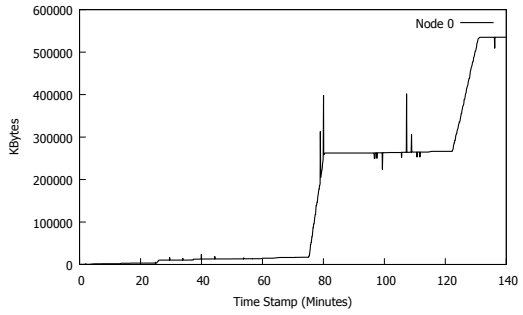


Figure 5.25: Global StorageDispatcher Sent Data - 20 instances

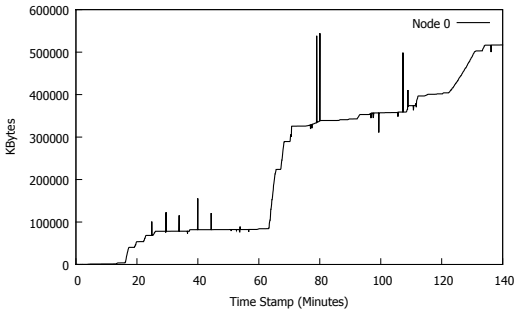


Figure 5.26: Global StorageDispatcher Received Data - 20 instances

can be found that the curves in both figures are almost same, and the used storage size the in 10-instance test is about half of the size in the 20-instance test. The reason is that both synthesis tests use the same time test plan. Therefore, the only difference between the two tests is the number of peers in 20-instance is doubled. As a result, the used storage size in 20-instance test is also doubled. In these two figures, similar to Figure 5.23 and 5.25, there are two dramatical ascends that are caused by *TestCaseUploadPhoto* and *TestCaseUploadFileinFolder*. In addition, there is a obvious decrease from 92 to 95 minutes in either of Figure 5.27 or 5.28. According to the test plan, in this period, *TestCaseDeletePhoto* and *TestCaseDeleteAlbum* are executed. And when a photo album is deleted, all the photos in the album will be removed as well. So, there are a number of photos deleted by the two series of test cases in the time slot.

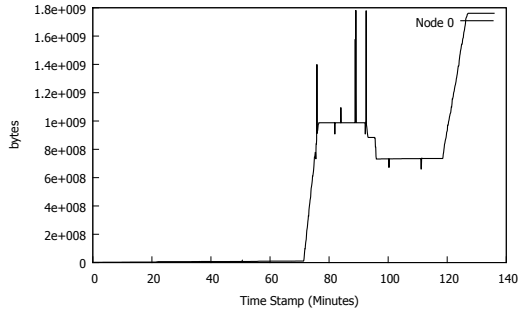


Figure 5.27: Global Storage Used Size - 10 instances

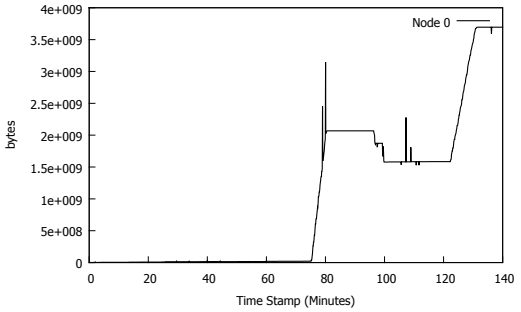


Figure 5.28: Global Storage Used Size - 20 instances

Figure 5.29 shows the data sent by framework layer in the 10-instance synthesis test, and 5.30 shows the framework received data in the test. At first glance, the two figures look a little bit similar, both of the values and the curves. But it still can be found that the curve in 5.30 is more smooth than the curve in

5.29. And the sizes of sent and received data are not exactly equal at certain time point, especially from 60 to 80 minutes. The same situation occurs in Figure 5.31 and 5.32, which present the framework sent and received data in the 20-instance test respectively. As described in Section 3.4, StorageDispatcher and MessageDispatcher contribute the majority of data flow passing through the framework layer. So, the framework sent and received data is close to the sum of StorageDispathcer data and MessageDispathcer data. However, as explained earlier, the measured StorageDispatcher sent data(Figure 5.23 and 5.25) and received data(Figure 5.24 and 5.26) are not actual data that is sent and received by StorageDispatcher, the framework sent and received data is slightly affected by the wrong measurements. Even so, it can be seen that the data contributed by MessageDispatcher is the most important part of the data going through the framework. In Figure 5.15 and 5.16, the data sent and received by MessageDispatcher at the end of the 10-instacne test is same, around 850000 kbytes. It means MessageDispatcher contributes about 77% data flow that passes through framework layer in the 10-instance test. And in the 20-instance test, this percentage remains the same.

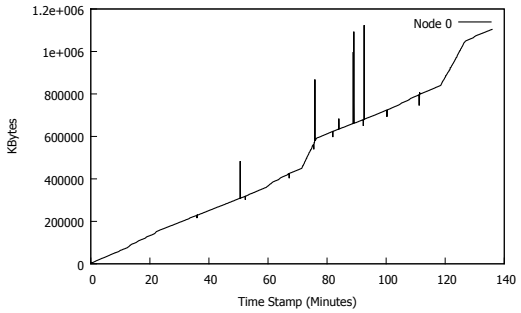


Figure 5.29: Global Framework Sent Data - 10 instances

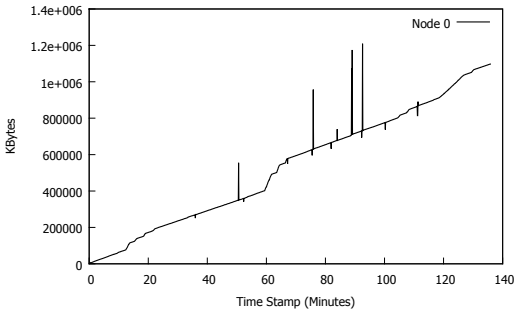


Figure 5.30: Global Framework Received Data - 10 instances

Figure 5.33 and 5.34 show the numbers of missed calculation in the 10-instance and 20-instance tests respectively. For the purpose of monitoring, the size of every data item is calculated before it is sent and received. Additionally, the size of each message is also measured before being sent out. These calculations could consume a large amount of memory. Therefore, for every node, when the free memory is less than $1/4$ of the total memory, the size of currently transferred data item or message will not be calculated. Comparing the test plan with these two figures, it can be found the missed calculations mainly happen in the durations of *TestCaseCommentWallPost*, *TestCaseSearch*, *TestCaseVote* *TestCaseUploadPhoto* and *TestCaseUploadFileinFolder*, since these test cases need a lot of either data item or message exchanges. However, it also can be seen that the number of missed calculations is very small in both figures. Even

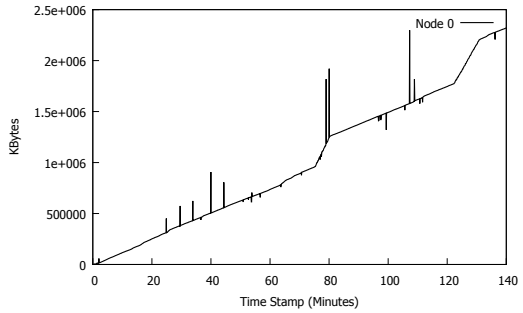


Figure 5.31: Global Framework Sent Data - 20 instances

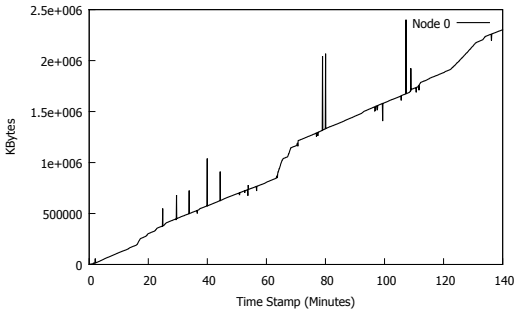


Figure 5.32: Global Framework Received Data - 20 instances

at the end of the 20-instance test, there are only less than 40 missed calculations. So, the missed calculations hardly impact on the test results.

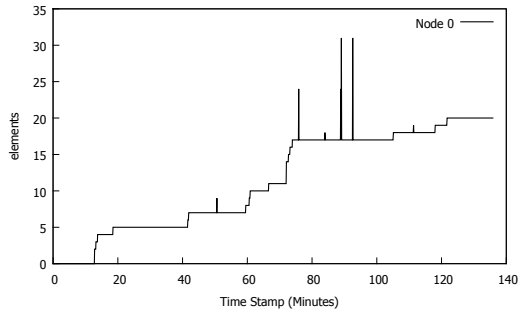


Figure 5.33: Global Missed Calculation - 10 instances

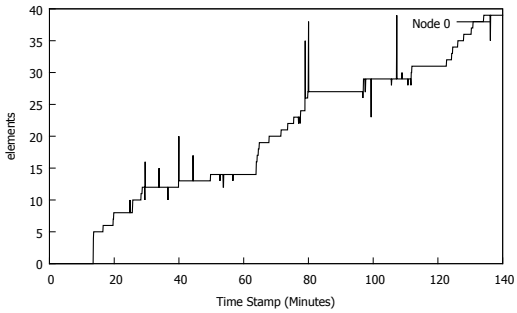


Figure 5.34: Global Missed Calculation - 20 instances

Figure 5.35 and 5.36 show the total numbers of null pointer exceptions and other exceptions occurred in the 10-instance synthesis test. And Figure 5.37 and 5.38 show the numbers of these exceptions in the 20-instance test. By comparing the figures and the timed test plan, it can be found that most of the exceptions are related to *TestCaseCommentWallPost* and *TestCaseUploadFileinFolder*. So, these two test cases as well as the two corresponding functionalities of `LifeSocial` need to be well examined. Also, it can be seen from Figure 5.35 and 5.37 that the amounts of null pointer exceptions are not regular, even these two synthesis tests share same configurations and test plan. And the only difference, 2 times of network scale, is not clearly displayed in the figures. The numbers of other exceptions, which are shown in Figure 5.36 and 5.38, are irregular. The reason of the exceptions is required to find out in the future.
Figure 5.39 and 5.40 show the average lookup time in the 10-instance and 20-instance synthesis tests. As the lookup function is always called in Stor-

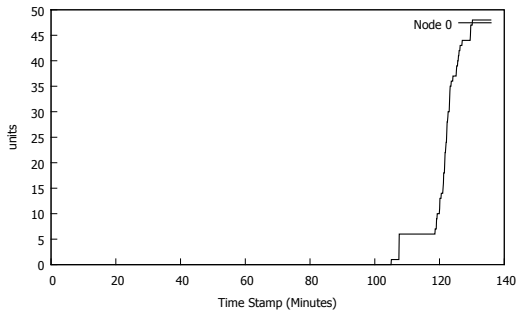


Figure 5.35: Global Null Pointer Exception - 10 instances

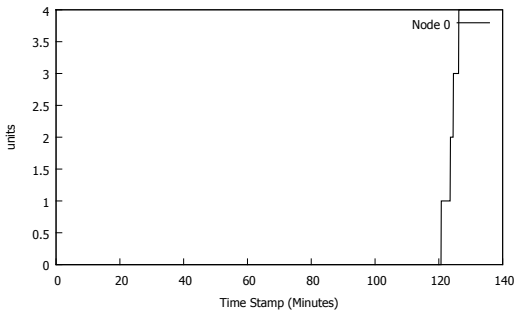


Figure 5.36: Global Other Exception - 10 instances

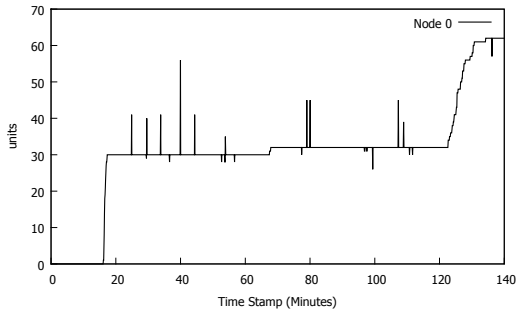


Figure 5.37: Global Null Pointer Exception - 20 instances

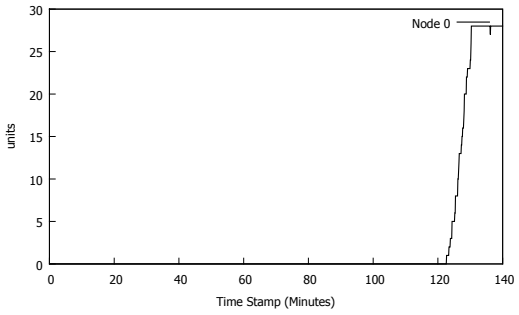


Figure 5.38: Global Other Exception - 20 instances

ageDispatcher when storing and retrieving data items, it is directly related to user experience. It can be seen from the figures that the average lookup time does not change based on the network scale. In most time of both tests, the average time is less than 5 milliseconds, which is quite fast and good for user experience.

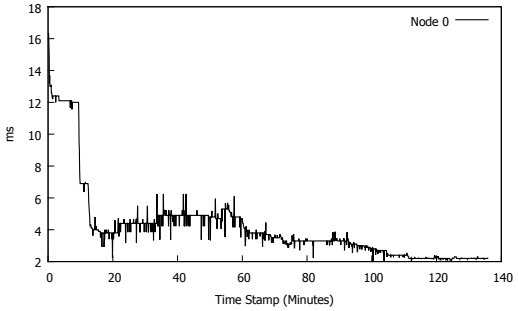


Figure 5.39: Global Average Lookup Time - 20 instances

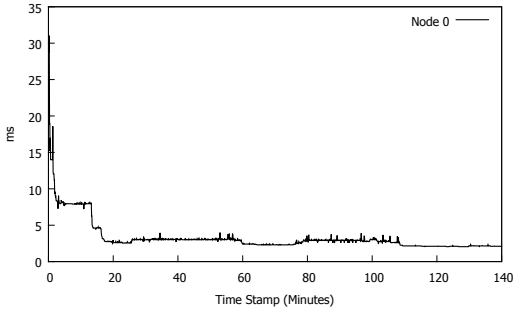


Figure 5.40: Global Average Lookup Time - 20 instances

Figure 5.41 and 5.42 show the total numbers of created wall entries in the 10-instance and 20-instance synthesis tests. At around 10 minutes, there are 45 created in 10-instance test, and 95 wall entries created in 20-instance test. According to the test plan, each salve should execute 5 times of *TestCaseSendWallPost*. Since there are 9 salves in 10-instance test and 19 salves in 20-instance test, the test results match the expectation of test plan. It means that all the *TestCaseSendWallPost* are executed correctly, and the functionality of sending wall entry has no problem.

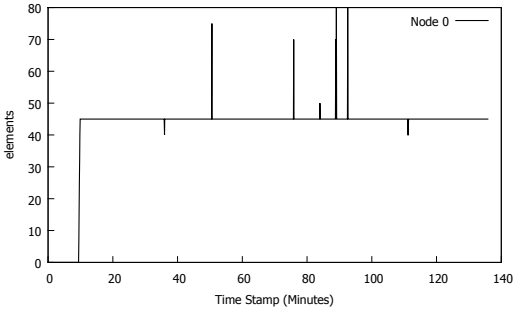


Figure 5.41: Global Wall Entry - 10 instances

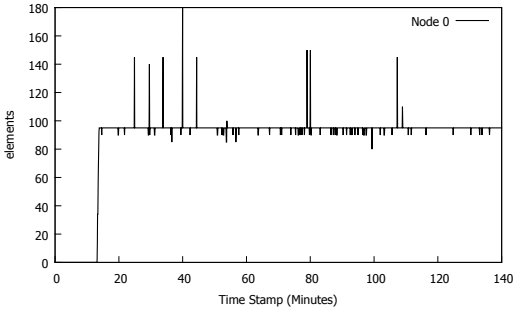


Figure 5.42: Global Wall Entry - 20 instances

Figure 5.43 and 5.44 present the total numbers of wall comments added in the 10-instance and 20-instance tests respectively. Based on the test plan, the repetition of *TestCaseCommentWallPost* is 5. It can be seen from the figures that

in 10-instance, there are 45 wall comments created in total, which means the test cases are well performed. However, the number of created wall comments in 20-instance test is only 65, which means there are totally 30 times of *TestCaseCommentWallPost* are failed during the test. It is obvious that the failed test cases are closely related to the null point exception shown in Figure 5.37. Therefore, both of *TestCaseCommentWallPost* and the functionality of adding wall comment need to be carefully examined.

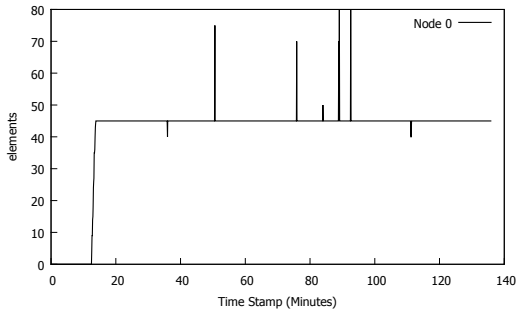


Figure 5.43: Global Wall Comment - 10 instances

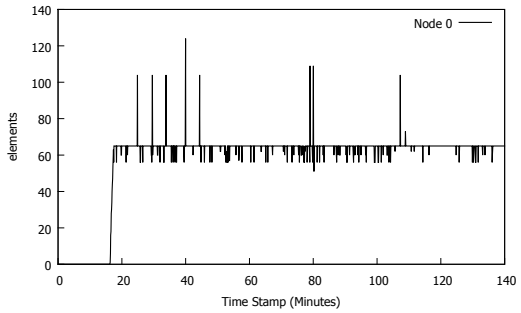


Figure 5.44: Global Wall Comment - 20 instances

Figure 5.45 and 5.46 show the amounts of viewing outbox messages in the 10-instance and 20-instance synthesis tests. According to the test plan, each salve is supposed to execute 5 times of *TestCaseViewOutboxMessage*. If so, there should be 45 times of viewing outbox action in the 10-instance test, and 95 in 20-instance test. As can be seen from the figures, the test results completely match the expectations. It means that both of *TestCaseViewOutboxMessage* and the functionality of viewing outbox messages are working well.

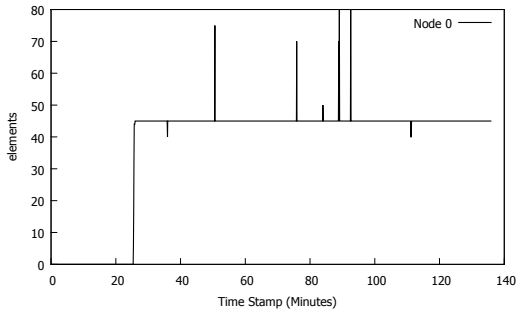


Figure 5.45: Global View Outbox - 10 instances

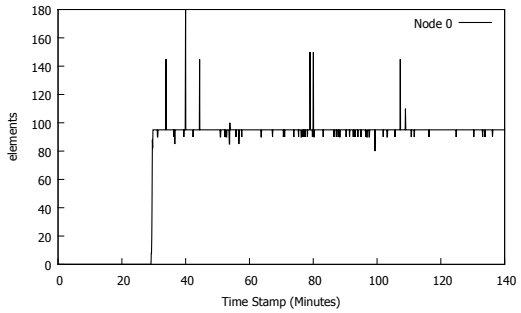


Figure 5.46: Global View Outbox - 20 instances

Figure 5.47 and 5.48 show the total numbers of sent chat messages in the 10-instance and 20-instance synthesis test. Based on the test plan, each slave

should execute 10 times of *TestCaseSendMultiChatMessage*, which means there should be 90 chat messages in the 10-instance test, and 190 chat messages in the 20-instance test. As can be seen from Figure 5.47, the sent chat messages in the 10-instance test are just as many as the test plan expects. On the contrary, the number of chat messages in the 20-instance test, which is shown in Figure 5.48, just reaches about 180. In other words, there are 10 times of *TestCaseSendMultiChatMessage* failed in the 20-instance test due to some unclear reason.

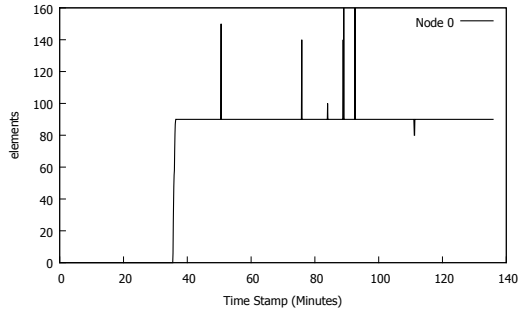


Figure 5.47: Global Sent Chat Message - 10 instances

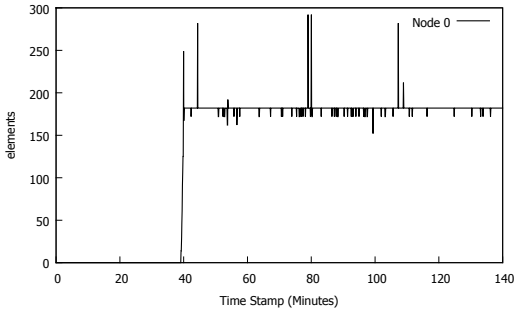


Figure 5.48: Global Sent Chat Message - 20 instances

Figure 5.49 and 5.50 show the amounts of searching a user in the 10-instance and 20-instance synthesis tests. According to the test plan, each salve is supposed to execute 5 times of *TestCaseSearch*. Ideally, there should be 45 times of search action in the 10-instance test, and 95 in 20-instance test. As can be seen from the figures, the expectations are completely matched by the test results. Therefore, both of *TestCaseSearch* and the functionality of Searching a user are working fine.

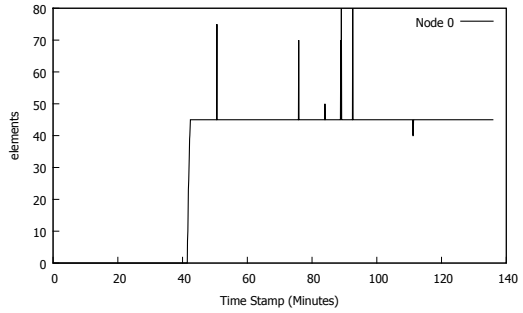


Figure 5.49: Global Search - 10 instances

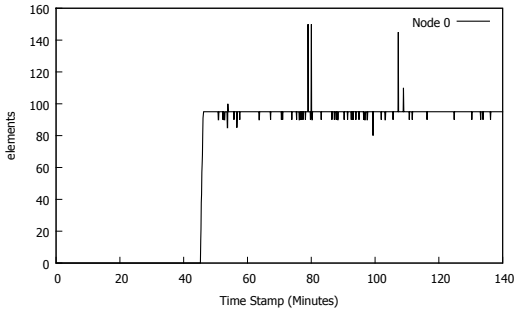


Figure 5.50: Global Search - 20 instances

As can be seen from Figure 5.51 and 5.52, the numbers of created calendar events in the 10-instance and 20-instance synthesis test are 45 and 95 respec-

tively, which totally matches the repetition of *TestCaseCreateCalendarEvent* in the test plan. So, *TestCaseCreateCalendarEvent* and the corresponding functionality of `LifeSocial` are satisfactory according to the test results.

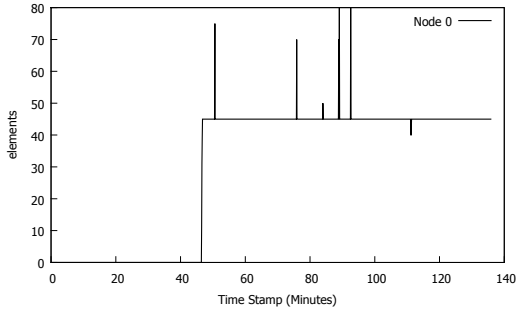


Figure 5.51: Global Calendar Event - 10 instances

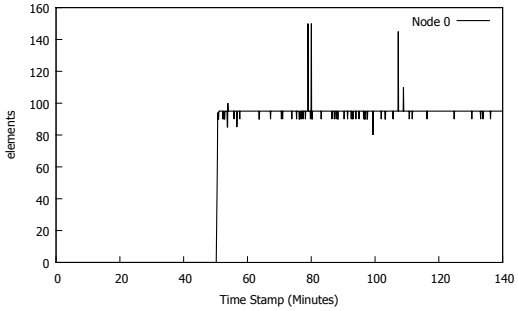


Figure 5.52: Global Calendar Event - 20 instances

Figure 5.53 shows the amount of deleted votings in the 10-instance synthesis test, and Figure 5.54 shows the deleted votings in the 20-instance test. According to the test plan, each slave is supposed to execute 2 *TestCaseDeleteVote* to delete 2 votings from user voting list. As can be seen from the figures, there are a total of 18 votings deleted in the 10-instance test, and 38 votings deleted in the 20-instance test, which means the test results completely match the expectations of test plan. In short, either *TestCaseDeleteVote* or the functionality of removing a voting from user voting list is working properly.

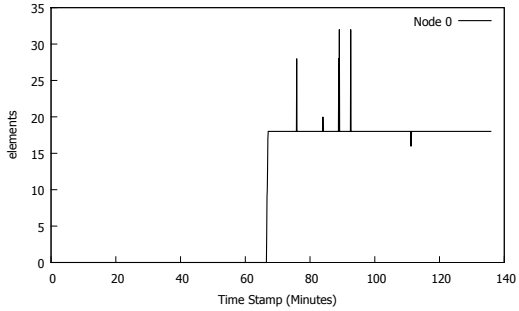


Figure 5.53: Global Deleted Voting - 10 instances

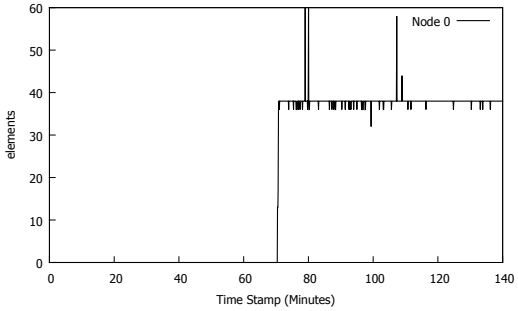


Figure 5.54: Global Deleted Voting - 20 instances

Figure 5.55 and 5.56 present the amounts of uploaded photos in the 10-instance and 20-instance synthesis tests respectively. According to the test plan, each slave should start to execute 50 times of *TestCaseUploadPhoto* from about 70 minutes. In this case, there should be 450 photos uploaded in the 10-instance test, and 950 photos in the 20-instance test in the end. It can be seen from

the figures that the expectations of test plan are totally met by the test results. Since there is not event one failure among so many times execution of *TestCaseUploadPhoto*, the functionality of uploading photo is considered very robust.

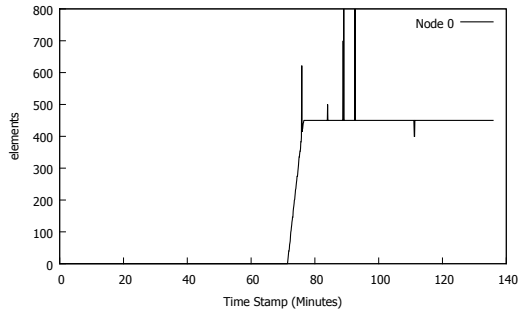


Figure 5.55: Global Uploaded Photo - 10 instances

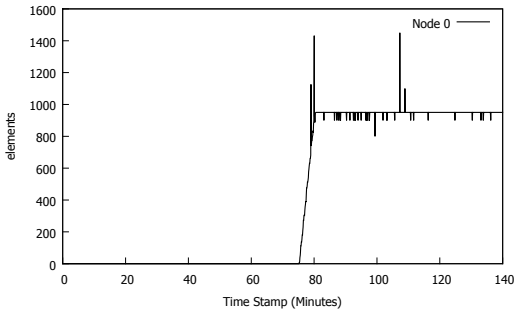


Figure 5.56: Global Uploaded Photo - 20 instances

Figure 5.57 shows the total number of files that are uploaded into group folders in the 10-instance synthesis test, and 5.58 shows the number of uploaded files in the 20-instance test. Based on the test plan, each salve should execute 50 times of *TestCaseUploadFileinFolder* from around 120 minutes. If so, at the end, there should be 450 uploaded files in the 10-instance test and 950 files in the 20-instance test. As can be seen from the figures, however, the results of both tests are less than the expectations. Similar to the wall comments shown in Figure 5.44, the failed test cases here, which aim at uploading a file into certain group folder, are probably relevant to the null pointer exceptions that are shown in Figure 5.35 and 5.37. Consequently, both *TestCaseUploadFileinFolder* and the corresponding functionality of `LifeSocial` need to be carefully examined.

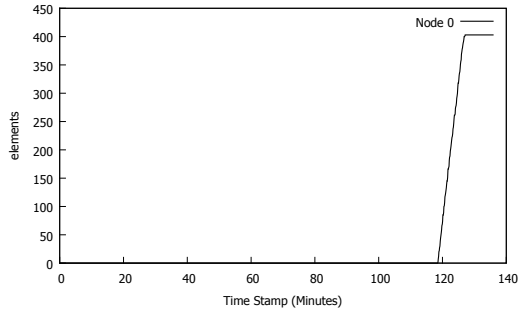


Figure 5.57: Global Uploaded File - 10 instances

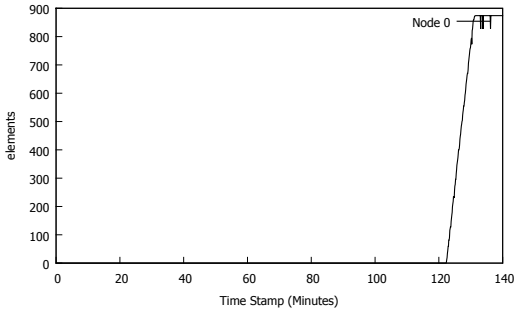


Figure 5.58: Global Uploaded File - 20 instances

In general, the first round of synthesis tests are successfully accomplished. Through the evaluation on the test results, it can be found that the monitoring function is working well. Most of metrics collect and measure the data of monitored objects accurately. Also, most of created test cases and corresponding `LifeSocial` functionalities are working correctly. Besides, the quality of network is quite good. There is no lost message during the tests, and the time to join the network and to look up items in the network is very short. However, there still exist some issues that have been mentioned during the evaluation and need to be fixed.

5.6 Modifications

After the first round of synthesis and random test, the independence of each test case has been improved in order to reduce the amount of various exceptions. The details about improvements of test cases are described in Section 3.3.2. Also, a number of metrics have been modified to enhance the accuracy of monitoring statistics, and to make their names more understandable. The improvements of metrics are introduced in 3.4.2.

5.7 Second Round Result Evaluation

After several improvements, the second round of synthesis test has been preformed. Since its test environment, deployment and test plan are still same with the first round, only a part of the test results related to the improvements are evaluated in this section.
Figure 5.59 and 5.60 show the sizes of data stored and retrieved by StorageDispatcher in the 10-instance synthesis test respectively. And Figure 5.61 and 5.62 show the StorageDispatcher stored and retrieved data in the 20-instance test. As can be seen, the names of these metrics have been corrected. Previously, these metrics were named as StorageDispatcher sent and received data, which was improper and quite misleading. After modifications, the names are much more accurate and understandable.
Figure 5.63 and 5.64 show the data sent and received by framework layer in the 10-instance test respectively. And Figure 5.65 and 5.66 show the framework sent and received data in the 20-instance test. Previously, the framework sent and received data was not equal because the data through StorageDispatcher was monitored in a wrong way. As can be seen, after modifications, the data sent and received by framework is totally equal in either test, which means the data flows through both StorageDispatcher and MessageDispatcher are well measured now.

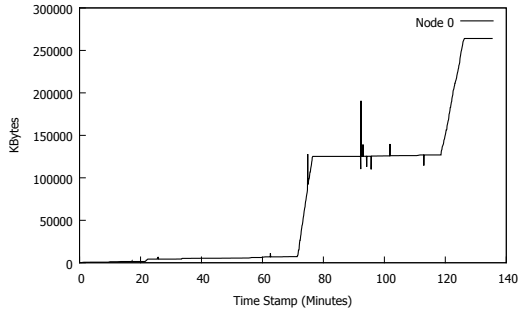


Figure 5.59: Global StorageDispatcher Stored Data - 10 instances

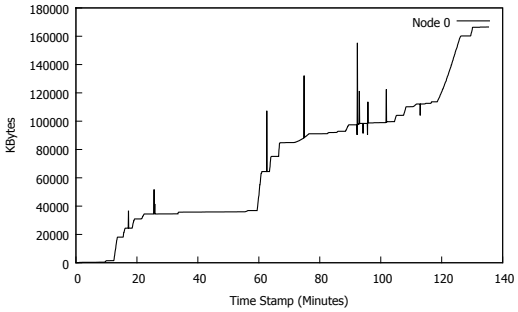


Figure 5.60: Global StorageDispatcher Retrieved Data - 10 instances

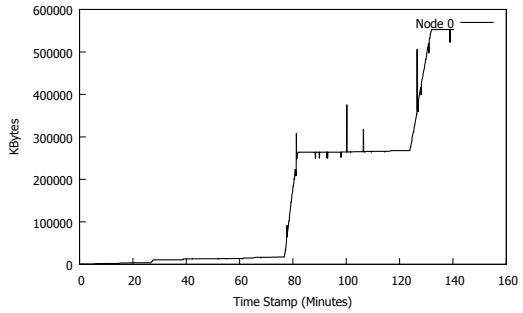


Figure 5.61: Global StorageDispatcher Stored Data - 20 instances

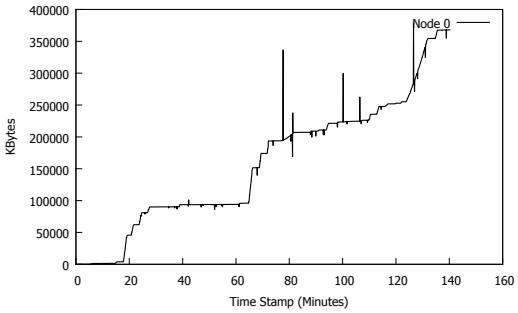


Figure 5.62: Global StorageDispatcher Retrieved Data - 20 instances

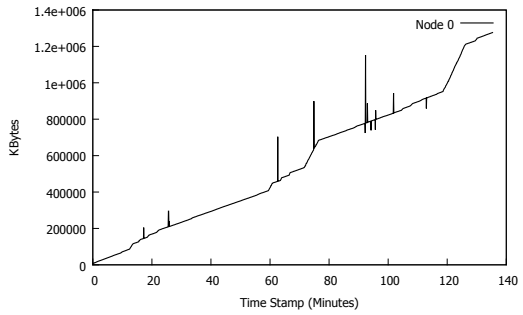


Figure 5.63: Global Framework Sent Data - 10 instances

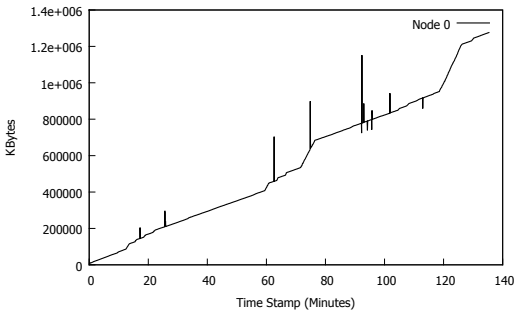


Figure 5.64: Global Framework Received Data - 10 instances

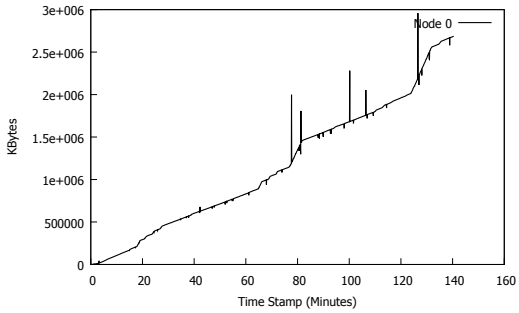


Figure 5.65: Global Framework Sent Data - 20 instances

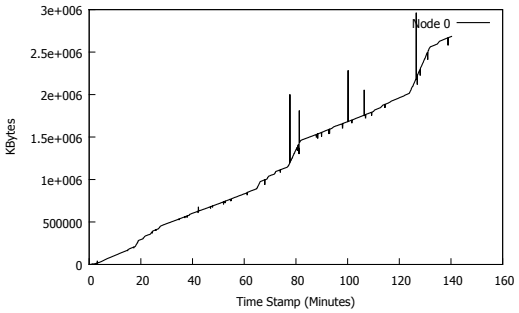


Figure 5.66: Global Framework Received Data - 20 instances

Figure 5.67 and 5.68 show the amounts of null pointer exceptions and other exceptions happened in the 10-instance test. And Figure 5.69 and 5.70 show the amounts of the exceptions in the 20-instance test. As can be seen, there are a total of 40 exceptions in the 10-instance test, and 75 exceptions in the 20-instance test. By comparing with the result of first round synthesis test, it can be found that the total numbers of exceptions have just reduced a little. It is because timed test plan is used in every synthesis test, and the test plan is well designed to avoid from potential exceptions. In other words, the impacts of test case independence on the overall result of synthesis test is limited.

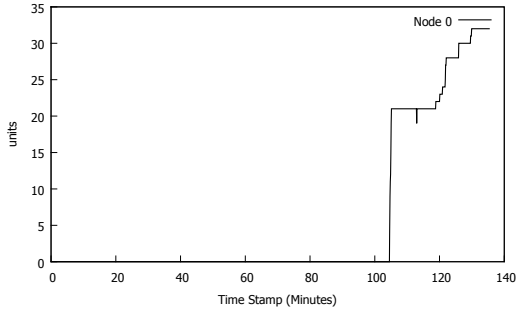


Figure 5.67: Global Null Pointer Exception - 10 instances

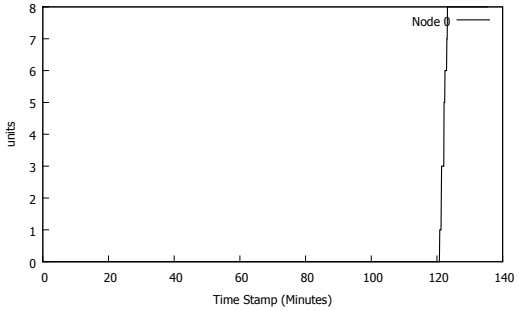


Figure 5.68: Global Other Exception - 10 instances

Figure 5.71 presents the total number of wall comments created in the 10-instance test, and Figure 5.72 presents the wall comments number in the 20-instance test. As can be seen, there are total 45 wall comments added in the 10-instance test, and 95 comments in the 20-instance test. Comparing with the test plan, it can be found that all the *TestCaseCommentWallPost* are successfully executed in both synthesis tests. In other words, the reliability of *TestCaseCommentWallPost* has largely improved due to the improvements of

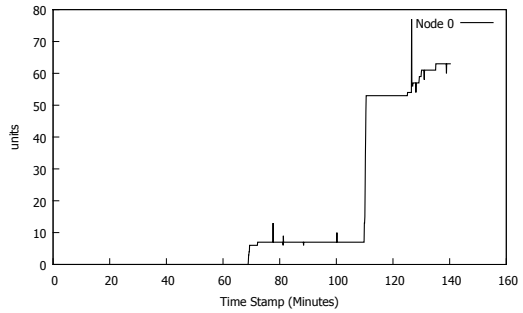


Figure 5.69: Global Null Pointer Exception - 20 instances

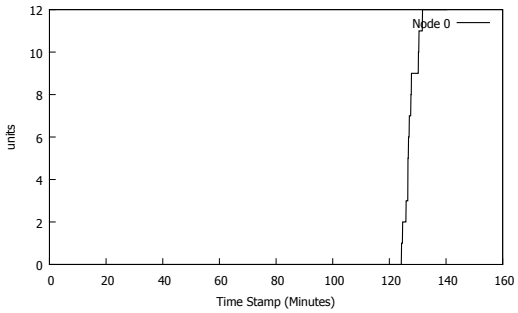


Figure 5.70: Global Other Exception - 20 instances

test case independence.

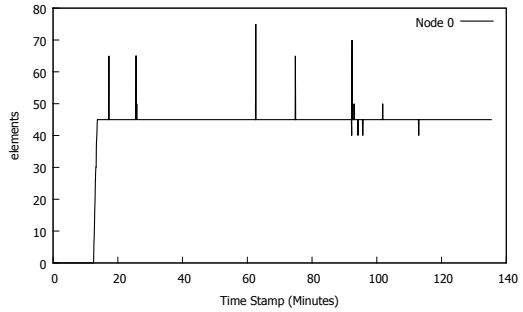


Figure 5.71: Global Wall Comment - 10 instances

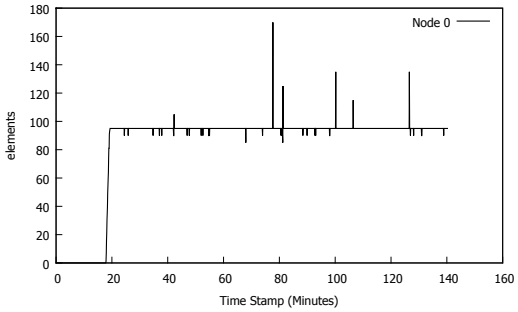


Figure 5.72: Global Wall Comment - 20 instances

Figure 5.73 and 5.74 show the amounts of sent chat messages in the 10-instance and 20-instance test respectively. It can be seen that, in the 10-instance test, the amount of chat messages is 90, and the amount in the 20-instance test is 190, which completely matches the repetition of *TestCaseSentMultiChatMessage* in the test plan. Therefore, it can be found that the small problem which previously existed in *TestCaseSentMultiChatMessage* has been fixed by the improvements of test case independence.

Figure 5.75 shows the total number of files that are uploaded into group folders in the 10-instance test, and Figure 5.76 shows the number of uploaded files in the 20-instance test. As can be seen, there are around 430 uploaded files in the 10-instance test, and about 910 files in the 20-instance test. Comparing to the result of the first round synthesis test, this result has already improved at a certain extent after modifications. However, the amount of uploaded files in either of new tests is still a little bit less than the expected number, which means there are still some *TestCaseUploadFileinFolder* failed during the test. Since

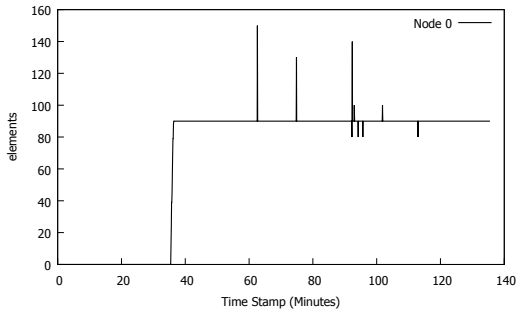


Figure 5.73: Global Sent Chat Message - 10 instances

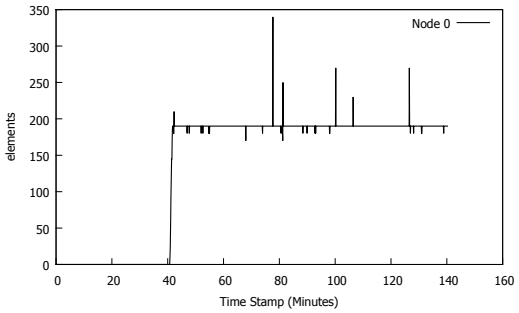


Figure 5.74: Global Sent Chat Message - 20 instances

the problem in *TestCaseUploadFileinFolder* is not totally repaired, and the responsible plugin, namely Groups Plugin, is a relatively complicated plugin, the *TestCaseUploadFileinFolder* and the corresponding functionality if `LifeSocial` need further examination.

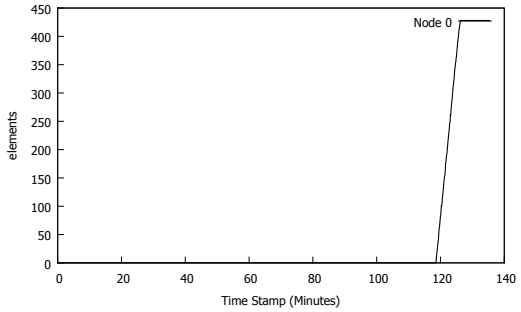


Figure 5.75: Global Uploaded File - 10 instances

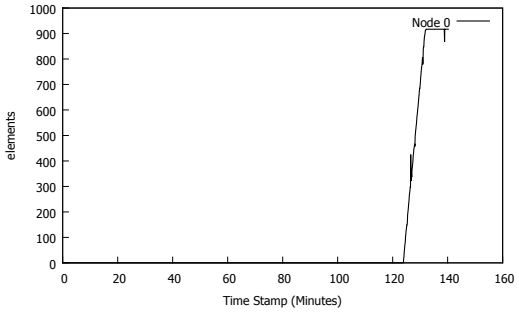


Figure 5.76: Global Uploaded File - 20 instances

5.8 Conclusion

Overall, the performance of `LifeSocial` in synthesis test is good. Except a few of small issues related to Groups Plugin, most functionalities and the corresponding test cases are working properly after the improvements of test case independence. Besides, the quality of `LifeSocial` network is stable throughout all the tests. There is no messages lost in the tests, and the join network time and lookup time in each test is quite short. Furthermore, the monitoring function is working well, and the collected statistics is more accurate and understandable after the modifications.

6 Random Test

This chapter describes all the information about random test, and the results of random test. In the beginning, the test environment is introduced in section 6.1. Next, all the deployments for random test are presented in Section 6.2. Afterwards, the basic mechanism and the exact start script of random test are introduced in section 6.3. Section 6.4 describes the random selection of user behaviours in random test and section 6.5 shows the method how to collect, choose and process the test result of random test. At the end, the evaluation on the first round of random test, added modifications and the evaluation on the second round of random test are presented in section 6.6, 6.7 and 6.8 respectively.
This whole chapter is written by W.Peng alone. All the related work of random test is made by W.Peng as well, including designing new test case registry and the start script of random test, performing two rounds of random tests, and evaluating the results of all random tests.

6.1 Test Environment

Since the testing machines in random test are still the machines used in synthesis test, the test environment of random test is as same as that of synthesis test, which is introduced in section 5.1.

6.2 Deployment

This section describes the different deployments for two kinds of random test, namely the random test with churn and the random test without churn.
Actually, the *Step 1* and *Step 3* of deployment for random test are exactly same with the corresponding steps of that for synthesis test. In *Step 1*, the code of `LifeSocial` needs to be cloned via *Smartgit* and imported into *Eclipse*, and the *resources* folder needs to be copied into eclipse directory. And in *Step 3*, the storage and monitoring files of previous tests require to be removed before new random test.
In terms of configurations, the only difference between the deployment of synthesis test and random test is two *Social Parameters*:

- *StartScript* path could be a local path or a URL. In the thesis, it is set to a URL that links to a file in author's *Dropbox* so that to make it easier to update the content of the start script on every testing machine at any time. The path of *StartScript* can be found in Figure 5.1.

- *EnableAutomaticLogout* should be set to true in the random test with churn. In fact, churn means that there are users joining and leaving the network during the test. And the online duration of user is determined by the *Online Time Distribution*, which is designed in [17] and shown in Figure 6.1. In this case, slave can automatically log out of the the network after a certain assigned duration, then log in as another user. As for the random test without churn, *enableAutomaticLogout* should be set to false so that the users logged in at the beginning will keep online until the end of the test without logout.

Except for these two *Social Parameters*, other parameters for random test are configured as same as synthesis test, which are described in Section 5.2.

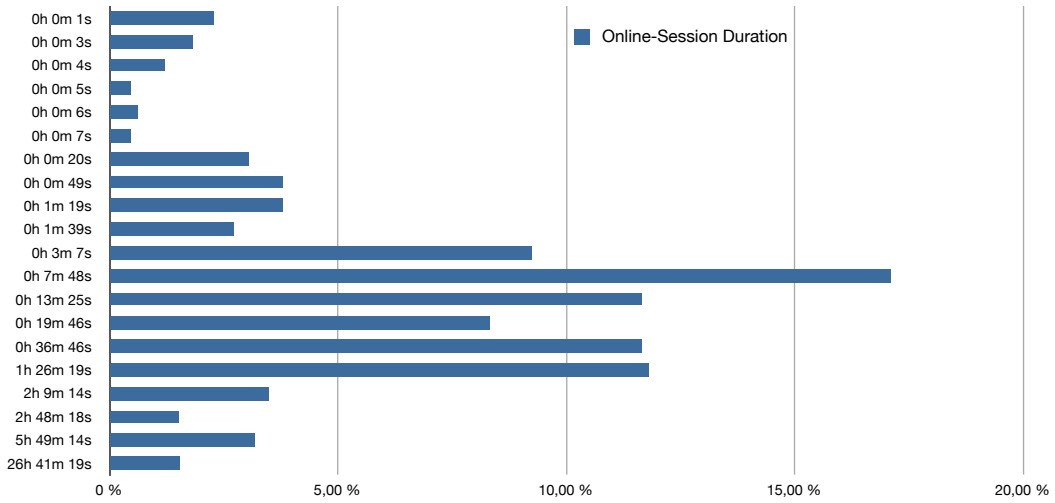


Figure 6.1: Online Time Distribution [17]

6.3 Start Script

Unlike synthesis test, there is no timed test plan in random test, since all the testing machines are not required to execute test cases synchronously. Instead, a start script is used in the random test either with or without churn. In this section, the basic mechanism of random test and the content of applied start script in the test are described.

```
1x TestCaseChangeProfilePicture
1x TestCaseChangeProfileInformation
1x TestCaseSendFriendRequest
1x TestCaseCreatePhotoAlbum
5x TestCaseUploadPhoto
2x TestCaseSendWallPost
1x TestCaseCommentWallPost
1x TestCaseSendMultiChatInvitation
1x TestCaseCreateCalendarEvent
2x TestCaseCreateVote
1x TestCaseCreateGroup
1x TestCaseCreateFolder
2x TestCaseUploadFileinFolder
1x TestCaseViewForum
1x TestCaseCreateForumThread
1x TestCaseCommentForumThread
```

Figure 6.2: Start Script

Normally, a slave is supposed to be idle after logging in the network. In order to start random test, the random mode of `LifeSocial` needs to be turned on. The only way to enable the random mode for all slaves is to send the command *EnableRandomMode* via TestCase Delegater, which is shown in Figure 5.2. After the random mode is enabled, every slave executes the start script firstly. Then, the next phase is distinct for different types of random test. In the random test with churn, each salve is assigned an online duration, and performs a sequence of test cases randomly until the online time is up. When the assigned online time is over, a slave is going to log out of the network and log in as another user; In contrast, slaves in the random test without churn, are not assigned online durations, but execute various test cases randomly until the end of test. Additionally, since there is no timed test plan, both kinds of random test have no exact time bound for salves to end the test. Consequently, a command *DisableRandomMode* is used at the end of random test, which is sent through TestCase Delegater manually as well.

In essence, a start script consists of series of test cases and corresponding repetitions. There is no time constraint for each test case to keep the progresses on all the testing machines are synchronous. Therefore, the execution of the same start script on each testing machine is completely independent. The currently applied start script of random test is shown in Figure 6.2.

As can be seen, the test cases in the start script almost cover all the plugins of `LifeSocial`. In this way, the preconditions of most existing test cases that are introduced in Section 3.3 are fulfilled, so that to eliminate null pointer exceptions and other unexpected exceptions as far as possible during the test.

6.4 Random Selection

This section describes how the testing slave choose test cases randomly to execute during random test.
Based on [14], the author of [17] classified all the social behaviours in `LifeSocial` into 6 categories, namely *Profile&Friends*, *Wall*, *Communities*, *Photos*, *Search* and *Messages*, and designed the *Category Transition Probability* for `LifeSocial`, which is shown in Figure 6.3.

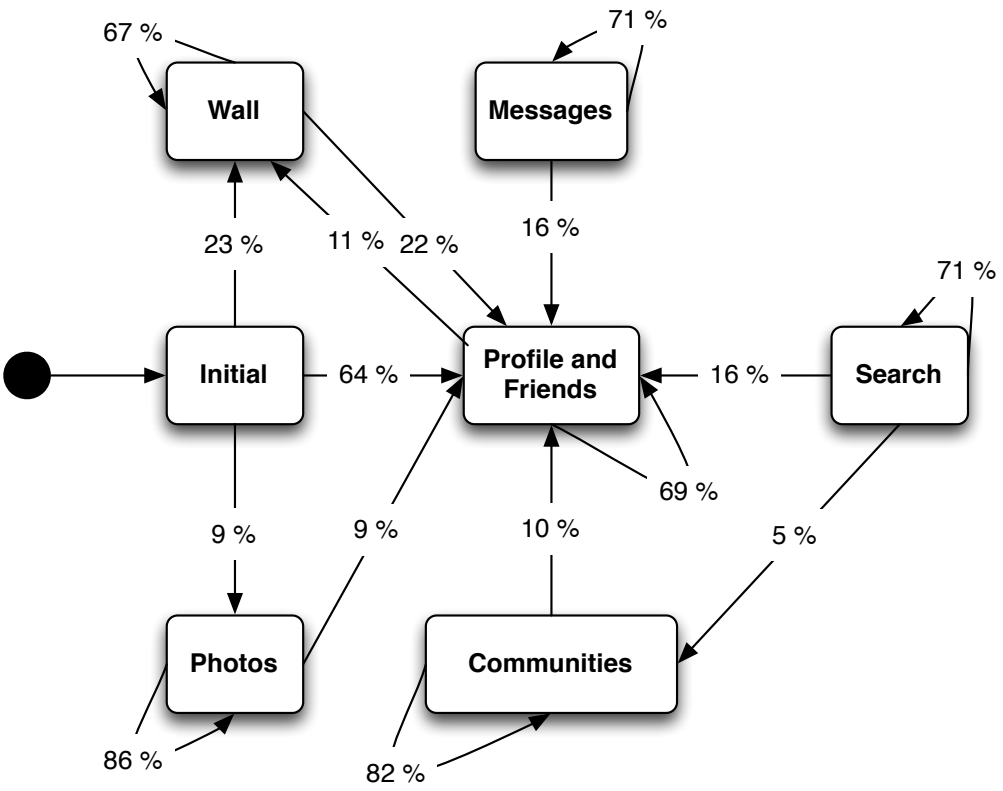


Figure 6.3: Category Transition Probability [17]

As can be seen from the figure, the *Category Transition Probability* specifies the transition probabilities among all 6 categories. For example, after executing a behaviour in *Communities*, there are just 10% users going to perform a behaviour in *Profile&Friends*, and 82% users will continue to execute a behaviour in *Communities*. According to the *Category Transition Probability*, author W.Peng redesigned the *TestCaseRegistry* by adding most test cases of new `LifeSocial` into proper categories. The new *TestCaseRegistry* is presented by Table 6.1, and it can be found in *resources* folder under *eclipse* directory.

Category	Test Case
Search	TestCaseSearch
Wall	TestCaseSendWallPost TestCaseCommentWallPost TestCaseViewFriendsWall TestCaseViewOwnWall
Messages	TestCaseSendMessage TestCaseViewInboxMessage TestCaseViewOutboxMessage TestCaseSendMultiChatInvitation TestCaseSendMultiChatMessage
Photos	TestCaseCreatePhotoAlbum TestCaseEditAlbum TestCaseUploadPhoto TestCaseViewFriendsPhoto TestCaseViewFriendsAlbums TestCaseViewOwnAlbums
Communities	TestCaseCreateGroup TestCaseCreateFolder TestCaseViewGroup TestCaseRenameFolder TestCaseViewFolder TestCaseUploadFileinFolder TestCaseViewForum TestCaseCreateForumThread TestCaseViewForumThread TestCaseCommentForumThread
Profile & Friends	TestCaseChangeProfilePicture TestCaseChangeProfileInformation TestCaseViewOwnProfile TestCaseViewFriendsProfile TestCaseCommentWallPost TestCaseViewFriendsWall TestCaseViewFriendsPhoto TestCaseViewFriendsAlbums TestCaseCreateVote TestCaseVote TestCaseGetMyVotings TestCaseGetVotingResults TestCaseViewCalendar TestCaseCreateCalendarEvent TestCaseEditCalendarEvent

Table 6.1: Test Case Registry

6.5 Monitoring Data Processing

For the random test without churn, the method to collect the monitoring data after the test is same with that of synthesis test, which is introduced in Section 5.4. All the generated monitoring statistics of testing machines are gathered onto the master node by a simple shell script. Then, the monitoring file that has the largest size is selected to generate a .plt file with the help of certain java file.
In contrast, after the random test with churn, it is not needed to gather all the monitoring files from testing machines. By several times of experiments, it is confirmed that there always are some testing machines disconnected from the network during the test, however, the monitoring functions of these lost machines are still working. It means that the monitoring statistics of these disconnected machines cannot accurately present the random test in the end. So, for the random test with churn, the only used monitoring file is the one of the master node, since the master is never disconnected from the network.

6.6 First Round Result Evaluation

In this section, the results of the 10-instance and 20-instance random tests with churn are analysed at first. Then, the results of two random tests without churn are evaluated as well.

6.6.1 Random Test with Churn

Figure 6.4 shows the amount of testing nodes in the network during the 10-instance random test with churn. And Figure 6.5 shows the amount of nodes in the 20-instance test over time. It is obvious that, in both tests, the numbers of global nodes are not stable. As can be seen, in the 10-instance test, the minimum amount of nodes is 6, at around 10 minutes, and the period from 40 to 50 minutes. And in the end of the 10-instance test, this value stays at 7. Similarly, there are just 13 active testing nodes monitored at the end of the 20-instance test. It means there are some testing nodes disconnected from the network during both churned random tests. As introduced in Section 6.3, a user in random test with churn, is supposed to live only a certain time, after that, it will log out and then log in as another user. In fact, by observation, a testing node might get stuck or crashed during the logout and login actions when the assigned time of the currently active user on the node is over. And this problem could get worse when not one but several nodes relog into the network in the same time slot. The reason of this problem is unclear yet. Therefore, the mechanism of random mode and related functions need to be well examined.

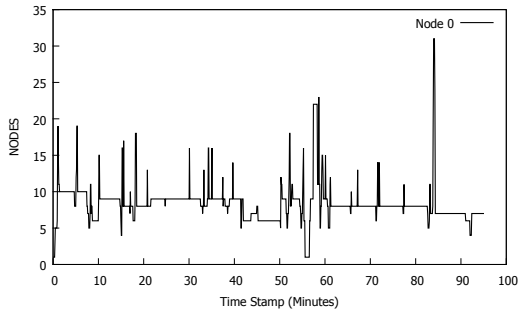


Figure 6.4: Global Nodes - 10 instances

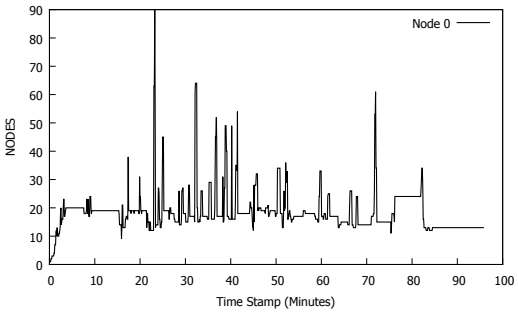


Figure 6.5: Global Nodes - 20 instances

Figure 6.6 shows the average time that a user joins the network in the 10-instance random test, and Figure 6.7 shows the average join time in the 20-instance test. Unlike synthesis test, the average join time in both random tests is not stable. In synthesis test, all the participations of user only happen at the beginning of test, therefore, the average time is fixed after that. However, in churned random test, there always are users leaving and joining the network. So, the average join network time is changing all the time, according to current network traffic. Usually, this average time in churned random test is longer than 2 seconds, which is not as good as synthesis test.

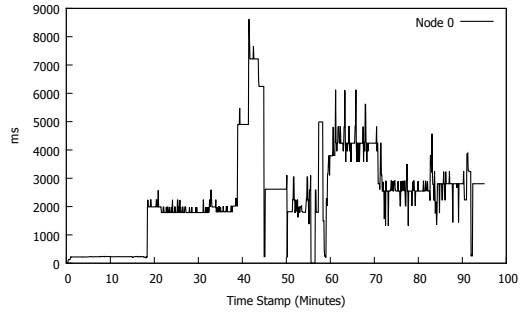


Figure 6.6: Join Network Time - 10 instances

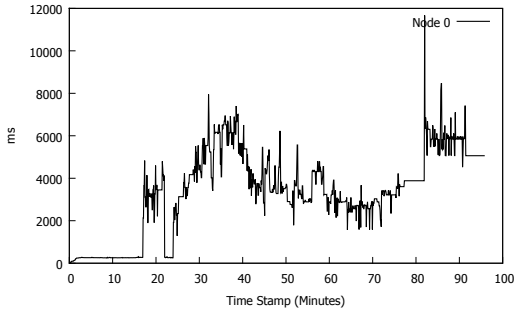


Figure 6.7: Join Network Time - 20 instances

Figure 6.8 and 6.9 present the total numbers of lost messages in the 10-instance and 20-instance churned random test respectively. It can be found from the figures that there are more than 600 messages lost in the 10-instance test, and more than 2000 messages lost in the 20-instance test. Actually, the real amounts of lost messages would be bigger than current amounts, since the monitoring statistics of the disconnected nodes are dropped. This inaccuracy is also reflected in the figures. For instance, in Figure 6.8, there is a slight decrease at around 40 minutes, which just matches the corresponding decrease

in Figure 6.4.

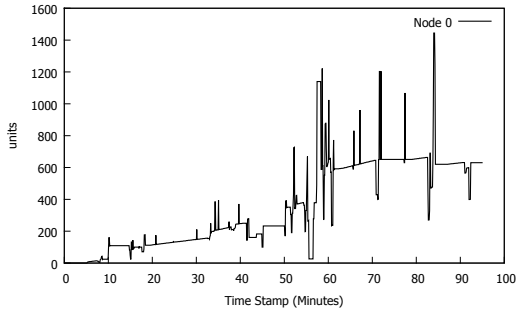


Figure 6.8: Global Lost Messages - 10 instances

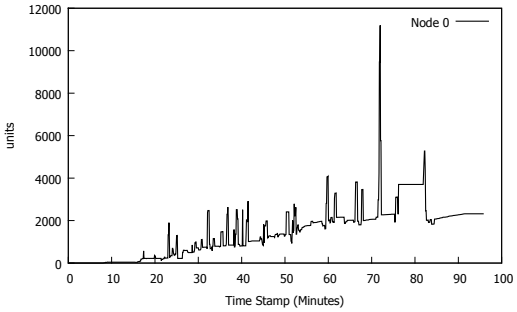


Figure 6.9: Global Lost Messages - 20 instances

Figure 6.10 and 6.11 show the total numbers of messages that are sent and received by MessageDispatcher in the 10-instance churned random test. And the sent and received messages through MessageDispatcher in the 20-instance test are shown in Figure 6.12 and 6.13. AS can be seen from the figures, because of the lost messages, the sent messages and received messages through MessageDispatcher are just close but not totally equal in both churned random tests. By comparing Figure 6.8 and 6.9, it can be found that there just are about 0.5% messages lost at the end of the 10-instance test, and 1.7% messages lost at the end of the 20-instance test. In other words, although there exist some lost messages in both churned random tests, the amounts of lost messages are relatively very small, and they can hardly affect the test results.

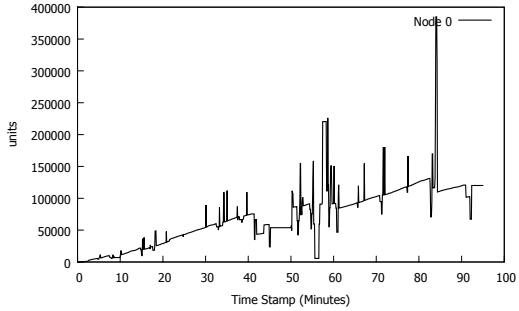


Figure 6.10: Global MessageDispatcher Sent Messages - 10 instances

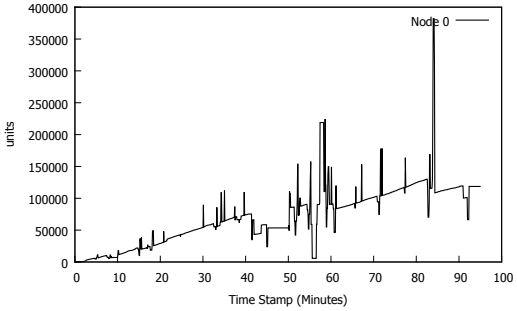


Figure 6.11: Global MessageDispatcher Received Messages - 10 instances

Figure 6.14 and 6.15 show the total numbers of sent and received messages through the network in the 10-instance churned test, and Figure 6.16 and 6.17 show the global network sent and received messages in the 20-instance test.

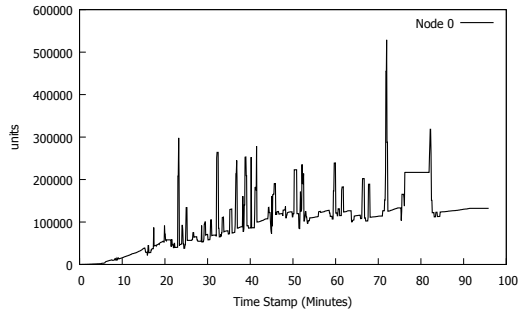


Figure 6.12: Global MessageDispatcher Sent Messages - 20 instances

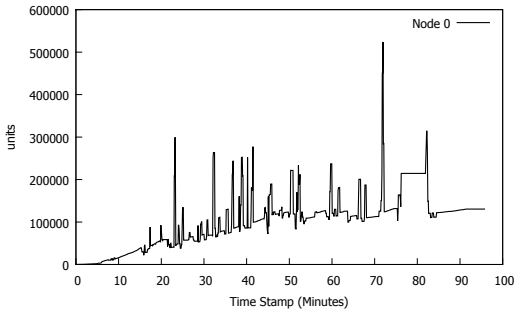


Figure 6.13: Global MessageDispatcher Received Messages - 20 instances

In these figures, the effect of lost messages can barely found. The values of network sent and received messages are almost equal in both tests. And it can be seen that at the end of the 10-instance test, there are around 660000 messages sent/received by the network. It also can be seen from Figure 6.10 and 6.11, there are 123000 messages passed through MessageDispatcher. It means about 19% messages passing through the network are contributed by MessagesDipsatcher in the 10-instance random test. And in the 20-instance test, the percentage becomes 14%. As mentioned in Section 5.5, this percentage in synthesis test is also around 15%, and is mainly determined by the intensity of user behaviours during the test. So, it can be found that the intensity of user behaviours in random test is close to synthesis test. Since the selection of user behaviour is randomly in random test, the percentage in the 10-instance test is a little bit different with that in the 20-instance test.

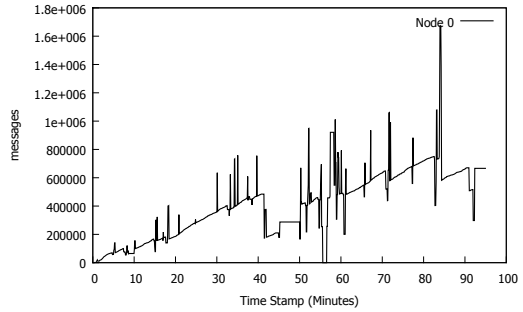


Figure 6.14: Global Network Sent Messages - 10 instances

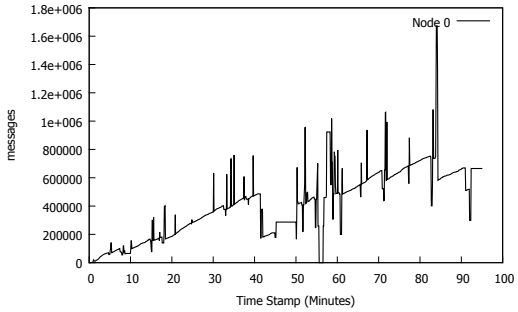


Figure 6.15: Global Network Received Messages - 10 instances

Figure 6.18 and 6.19 present the amounts of null pointer exceptions and other exceptions in the 10-instance random test respectively. And Figure 6.20 and

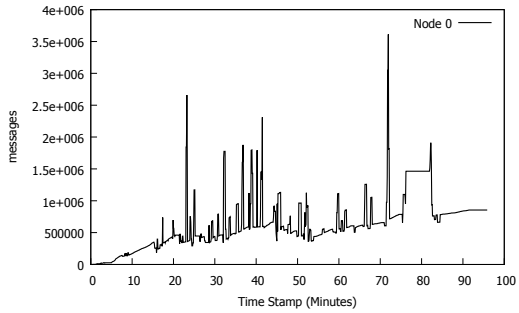


Figure 6.16: Global Network Sent Messages - 20 instances

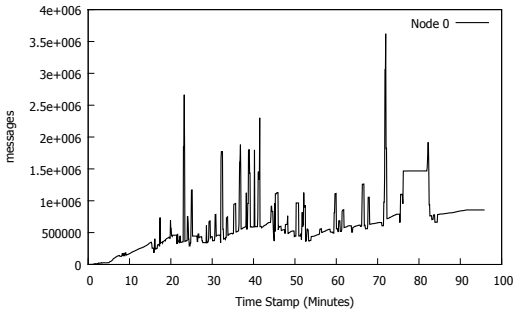


Figure 6.17: Global Network Received Messages - 20 instances

6.21 present the amounts of these exceptions in the 20-instance test. Generally, the amount of exceptions in random test is much bigger than synthesis test at the same network scale. As can be seen, there are around 80 null pointer exceptions and 120 other exceptions occurred in the 10-instance test, and 200 null pointer exceptions and 100 other exceptions in the 20-instance test. There probably are two major reasons for explaining these exceptions. First, the test plan of synthesis test is well designed and the sequence of test cases are elaborately arranged. However, in random test, the executed test cases are randomly selected, which could lead to unexpected exceptions if the preconditions of test case are not fulfilled, or required variables in *ParameterGenerator* are not initialized. Second, some needed information get lost due to the lost messages, which could result in exceptions as well. In order to fix the exception issue as far as possible, the independence of test case needs to be improved in next phase.

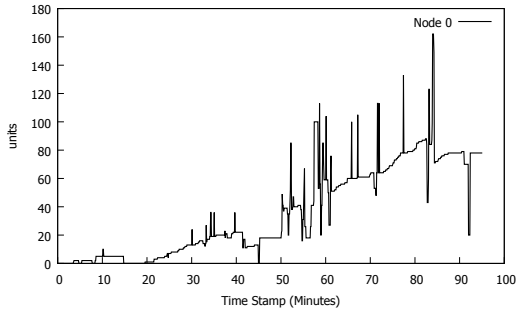


Figure 6.18: Global Null Pointer Exception - 10 instances

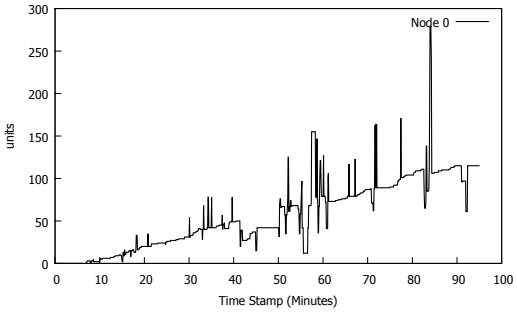


Figure 6.19: Global Other Exception - 10 instances

Figure 6.22 and Figure 6.23 show the average time spent on looking up a data item in the 10-instance and 20-instance random test respectively. As can be

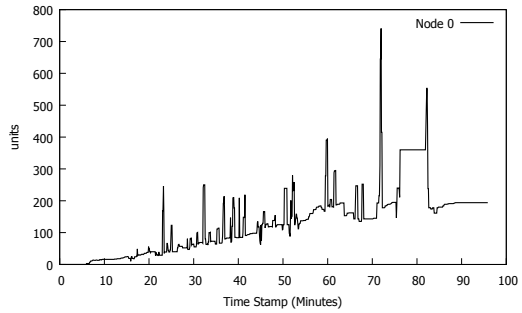


Figure 6.20: Global Null Pointer Exception - 20 instances

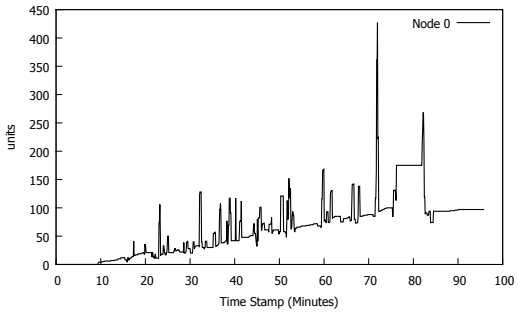


Figure 6.21: Global Other Exception - 20 instances

seen, in general, the average lookup time is still very short. However, it fluctuates quite a lot during the test, because the executed lookup needs to redirect to another replica node when the originally responsible node is offline, which costs a little extra time.

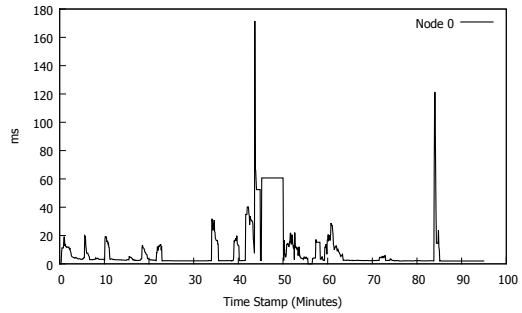


Figure 6.22: Global Average Lookup Time - 10 instances

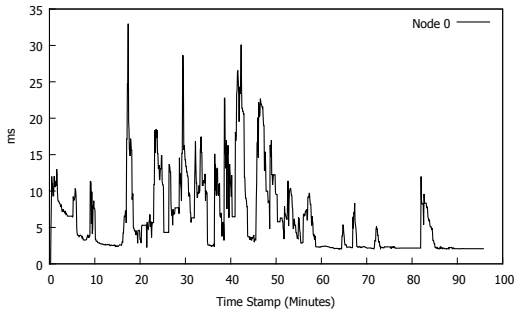


Figure 6.23: Global Average Lookup Time - 20 instances

Since the monitored data flow of StorageDispatcher is not accurate (explained in Section 5.5), the sent and received data through StorageDispathcer and framework are not evaluated in this section. To summarize, except for several obvious problems, such as crashed nodes, lost messages and many exceptions, the general performance of the network in random test with churn is acceptable. The sent and received messages through MessageDispatcher/network are almost equal, and the average lookup time is also short.

6.6.2 Random Test without Churn

Figure 6.24 and 6.25 show the total numbers of node in the network during the 10-instance and 20-instance random test without churn. As can be seen,

unlike the churned random test, the amounts of global nodes in these two tests are stable until the end, which means there is no user lost during the tests. As mentioned before, all the testing users in the random test without churn do not log out from the network, and they keep online and execute various test cases during the test.

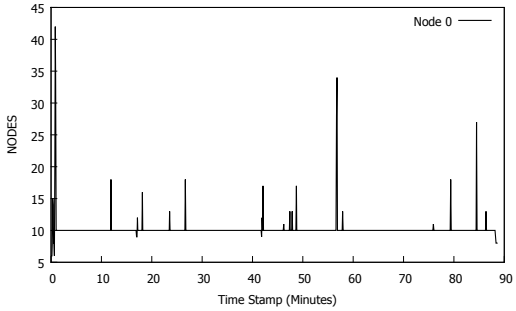


Figure 6.24: Global Nodes - 10 instances

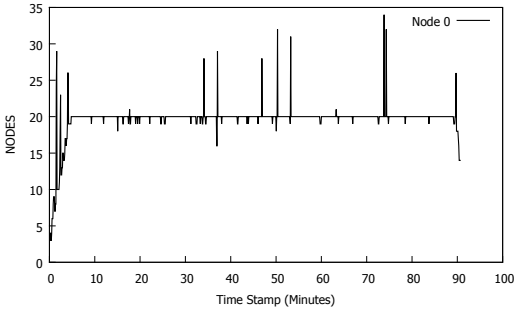


Figure 6.25: Global Nodes - 20 instances

Figure 6.26 shows the average time that a user spends on joining the network in the 10-instance random test, and Figure 6.27 shows the average time in the 20-instance test. It can be seen that, in either test, the join network time is kept very stable all the time, since there is no user leaving and joining the network during the test. And similar to synthesis test, the average join time is less than 0.3 second in both tests, which is very short.

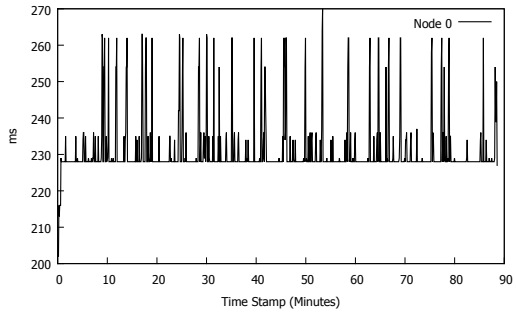


Figure 6.26: Join Network Time - 10 instances

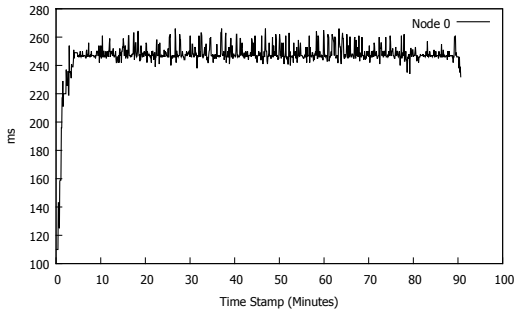


Figure 6.27: Join Network Time - 20 instances

Figure 6.28 and 6.29 present the total numbers of messages that are sent and received by MessageDispatcher in the 10-instance random test. And Figure 6.30 and 6.31 present the sent and received messages through MessageDispatcher in the 20-instance test. It can be found that, in either test, the amounts of the sent and received messages through MessageDispatcher are always same, which means there is no lost message during the random test without

churn.

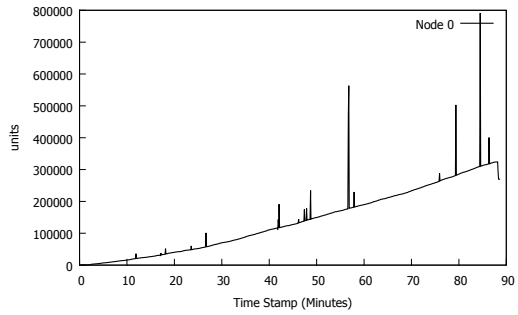


Figure 6.28: Global MessageDispatcher Sent Messages - 10 instances

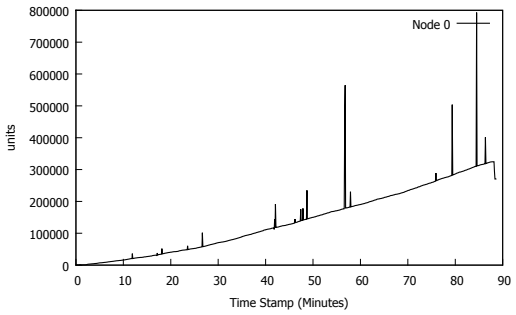


Figure 6.29: Global MessageDispatcher Received Messages - 10 instances

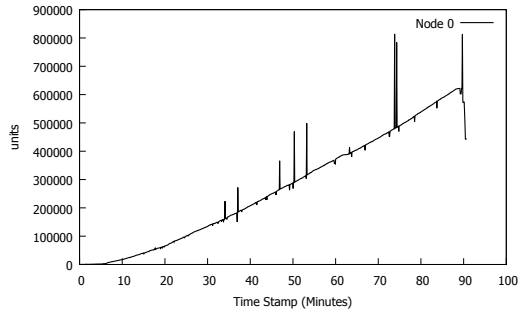


Figure 6.30: Global MessageDispatcher Sent Messages - 20 instances

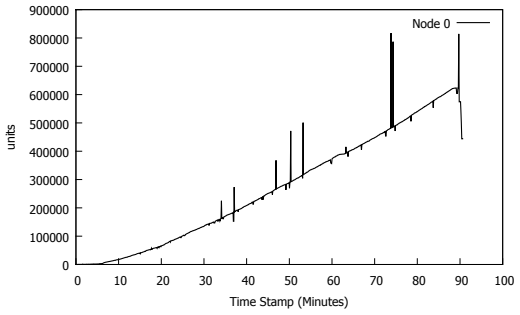


Figure 6.31: Global MessageDispatcher Received Messages - 20 instances

Figure 6.32 and 6.33 show the amounts of sent and received messages through the network in the 10-instance random test. And Figure 6.34 and 6.35 show the global network sent and received messages in the 20-instance test. Similar to the MessageDispatcher messages, the amounts of sent and received messages through the network in either test are completely equal. And comparing with the MessageDispatcher messages, it can be found that there are around 12% messages passing through the network contributed by MessageDispatcher at the end of the 10-instance random test. As for the 20-instance test, the MessageDispatcher messages occupy about 9% of the network messages in the end, which reflects the randomness of user behaviour selection in random test again.

Figure 6.36 and 6.37 show the amounts of null pointer exceptions and other exceptions occurred in the 10-instance random test respectively. And Figure

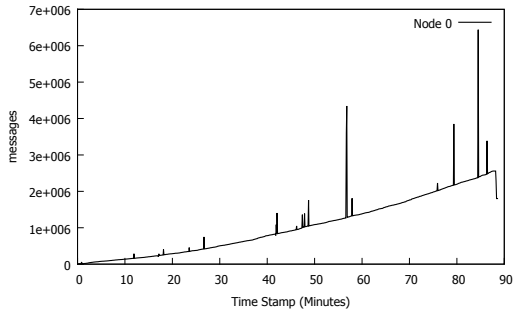


Figure 6.32: Global Network Sent Messages - 10 instances

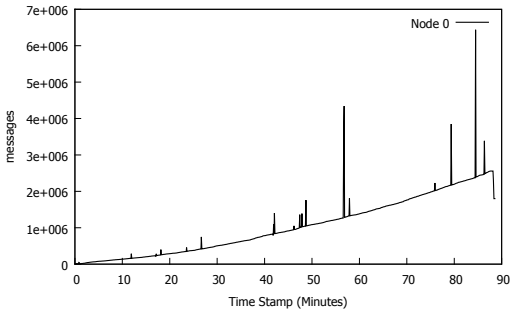


Figure 6.33: Global Network Received Messages - 10 instances

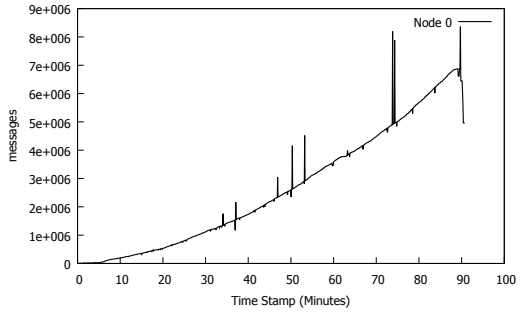


Figure 6.34: Global Network Sent Messages - 20 instances

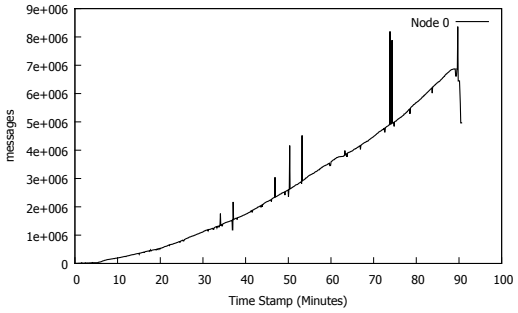


Figure 6.35: Global Network Received Messages - 20 instances

6.38 and 6.39 show the numbers of null pointer exceptions and other exceptions in the 20-instance test. Similar to the churned random test, there still exist plenty of exceptions in both random tests without churn. As can be seen that there are more than 300 all kinds of exceptions in the 10-instance test, and about 400 various exceptions in the 20-instance test. Comparing to the result of churned random test, it can be found that churn is not a significant factor which causes the exceptions. Therefore, the independence of test case probably is the most important factor responsible for the exceptions.

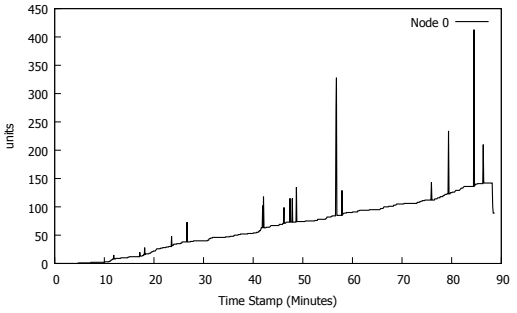


Figure 6.36: Global Null Pointer Exceptions - 10 instances

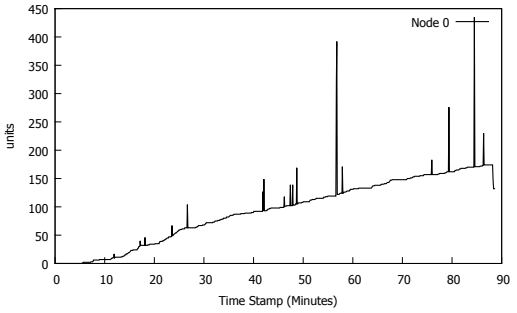


Figure 6.37: Global Other Exceptions - 10 instances

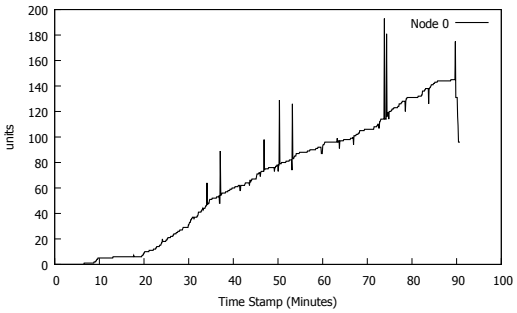


Figure 6.38: Global Null Pointer Exceptions - 20 instances

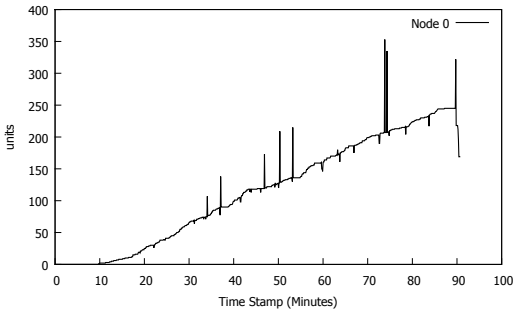


Figure 6.39: Global Other Exceptions - 20 instances

Figure 6.40 and 6.41 show the global average lookup time in the 10-instance and 20-instance random test respectively. As can be seen, in most time of both tests, the average lookup time is less than 3 milliseconds, which is very short. Besides, the lookup time is quite stable in both tests.
To conclude, the network performance in random test without churn is as good as synthesis test. There is not any node crashed, or message lost during the whole test. Also, the actions of joining the network and looking up a data item in the network are quick and stable, just like synthesis test. The only deficiency

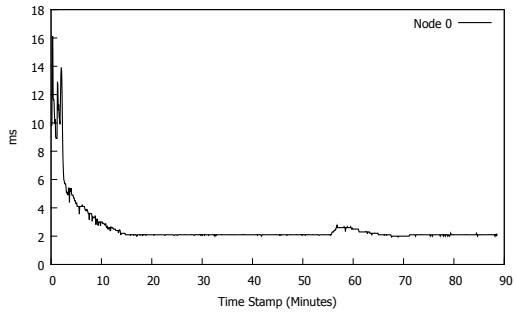


Figure 6.40: Global Average Lookup Time - 10 instances

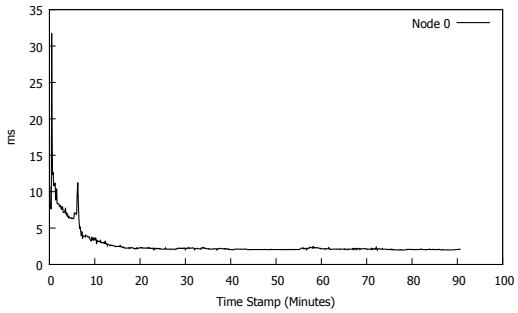


Figure 6.41: Global Average Lookup Time - 20 instances

found in random test without churn is that there still are a lot of exceptions happened during the test, which needs to be well examined and enhanced in next phase.

6.7 Modifications

All the improvements, which are introduced in Section 5.6 and used in the second round of synthesis test, are applied for the second round of random test.
Besides, another two modifications have been made for improving the second round of random test.

- First, the values of global variables in *ParameterGenerator* have been forced to clean up after a user logs out from the network. Previously, some of the global variables remained the values after the user logged out, which directly affected the accuracy of the test result. So, a function has been added in the *ParameterGenerator* in order to clean the global variables after every logout.
- Second, the sleep time after each re-login has been removed. Previously, every time a new user re-logged into the network, the currently executing thread slept for 5 seconds. It might be a reason of the lost messages and crashed nodes. So, the sleep time is no longer used in the second round of random test.

6.8 Second Round Result Evaluation

After the modifications, the second round random test has been performed. It still consists of two types, namely with or without churn. And the test environ-

ment, deployment and startscript are same with the first round as well. In this section, therefore, only the metrics related to the modifications are analysed for evaluating the second round of random test.

6.8.1 Random Test with Churn

Figure 6.42 shows the amount of testing nodes in the network during the 10-instance churned random test, and Figure 6.43 shows the amount of global nodes in the 20-instance test. As can be seen from the figures, in the 10-instance test, the number gradually decreases to 5, which means other 5 nodes crashed during the test. And in the 20-instance test, there are 5 nodes crashed, and other 15 nodes keep alive at the end. In other words, the original problem that happens during logout and login actions is still not fixed, and it is irrelevant to the test case independence and the sleep time after login. So, the mechanism of random mode and involved functions require to be well checked in the future.

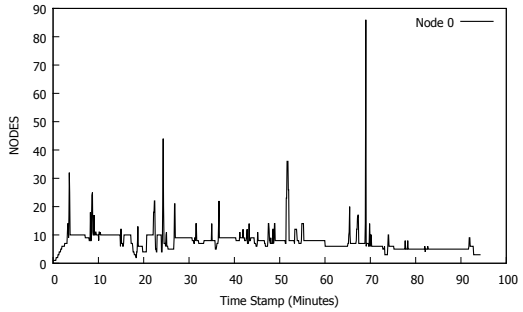


Figure 6.42: Global Nodes - 10 instances

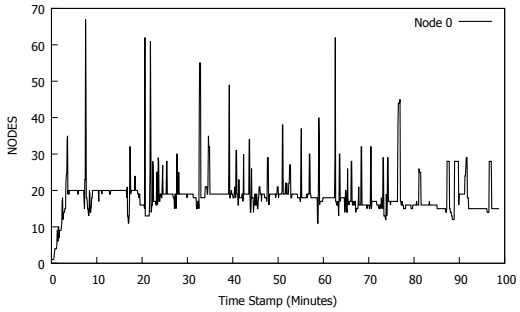


Figure 6.43: Global Nodes - 20 instances

Figure 6.44 and 6.45 present the total numbers of lost messages in the 10-instance and 20-instance churned random test respectively. It can be seen from the figures that the amounts of lost messages in both tests are still very big. There are more than 600 messages lost at the end of the 10-instance test, and 1900 messages lost at the end of the 20-instance test. It means that the lost messages are also irrelevant to the test case independence and the sleep time after login. And the exact reason causing the lost messages needs to be found out and improved in the future.
Figure 6.46 and 6.47 show the amounts of null pointer exceptions and other exceptions occurred in the 10-instance random test. And Figure 6.48 and 6.49 show the amounts of these exceptions in the 20-instance test. Comparing the figures with the corresponding results of first round test, it can be found that the total number of exceptions has significantly decreased. It means the modifications truly improved the independence of test case and enhanced the quality of churned random test.

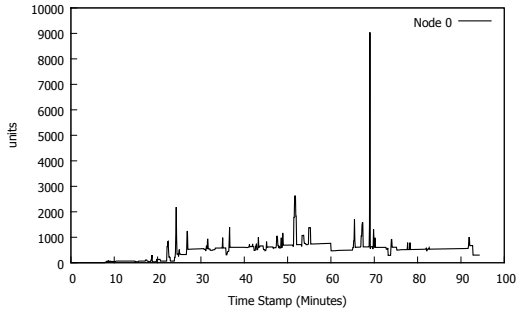


Figure 6.44: Global Lost Messages - 10 instances

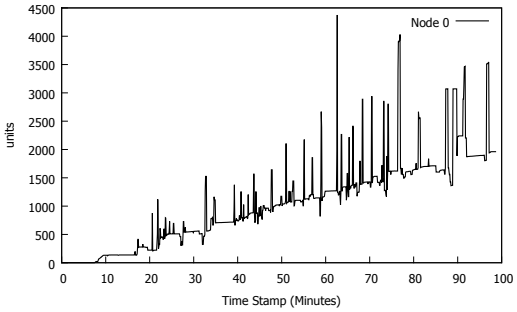


Figure 6.45: Global Lost Messages - 20 instances

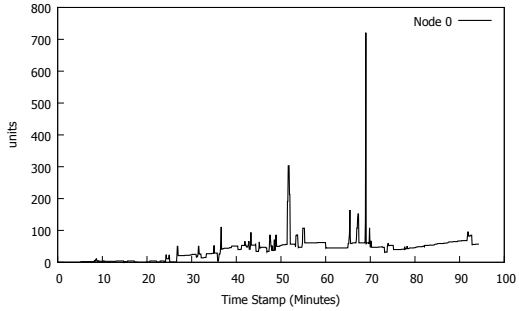


Figure 6.46: Global Null Pointer Exceptions - 10 instances

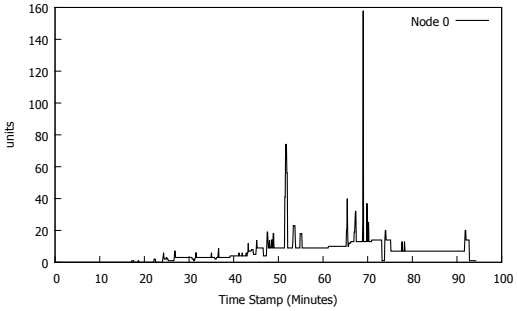


Figure 6.47: Global Other Exceptions - 10 instances

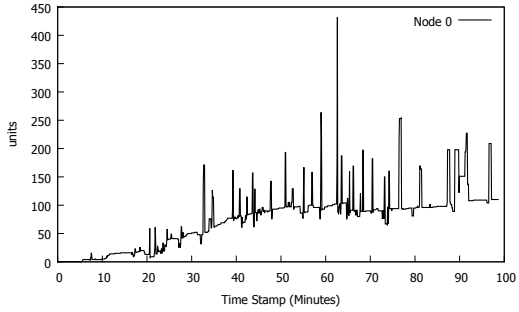


Figure 6.48: Global Null Pointer Exceptions - 20 instances

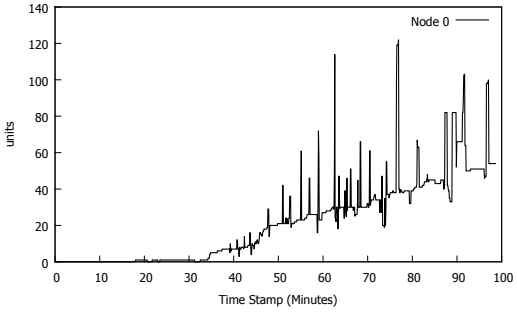


Figure 6.49: Global Other Exceptions - 20 instances

6.8.2 Random Test without Churn

Figure 6.50 and 6.51 present the amounts of null pointer exceptions and other exceptions happened in the 10-instance random test without churn. And Figure 6.52 and 6.53 present the amounts of these exceptions in the 20-instance test. As can be seen, the amount of total exceptions in either test is much smaller than the result of first round random test. It reflects the impact of the improvements made for test case independence again.

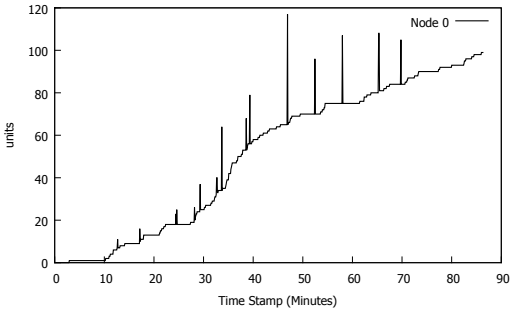


Figure 6.50: Global Null Pointer Exceptions - 10 instances

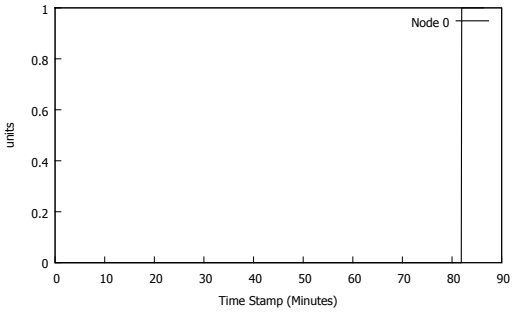


Figure 6.51: Global Other Exceptions - 10 instances

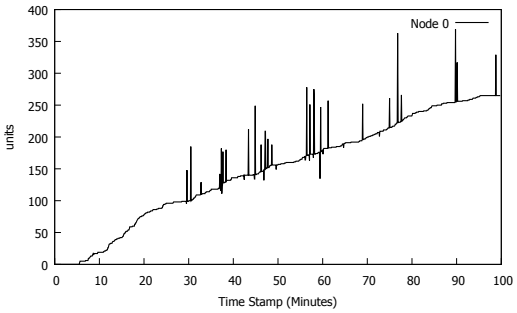


Figure 6.52: Global Null Pointer Exceptions - 20 instances

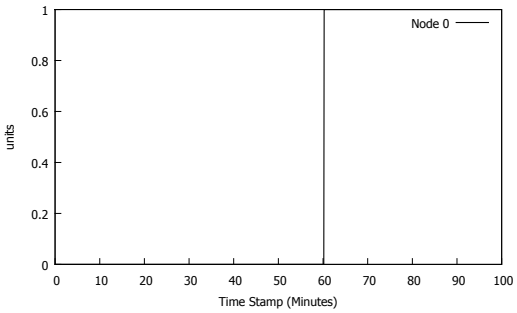


Figure 6.53: Global Other Exceptions - 20 instances

6.9 Conclusion

All in all, the performance of `LifeSocial` in the random test with churn is acceptable. Although there are still a few unsolved issues, such as crashed nodes and lost messages, almost all the test cases are executed well during the tests. Besides, the exception problem has already been improved largely. Also,

the average time for looking up an item in the network is very small all the time.
As for the random test without churn, it performs as well as synthesis test in general. There is no crashed node or lost message throughout two rounds of the tests. And both of join network time and lookup time are very short and stable. Moreover, the only deficiency, various exceptions, has been enhanced greatly after modifications.

7 Long Term Test

In chapter 5 we explained synthesis test to check the functionality of the `LifeSocial` application. This chapter explains about the long term test, which is an extension of synthesis test. The goal of this test is to maintain the network for long time, check the performance and reliability of the `LifeSocial` application under the specified environmental conditions.
This whole chapter is written by D.Balabhadruni alone. All the related work of long term test is made by D.Balabhadruni which includes designing new test plan, performing the test on 10 instances, and evaluating the results of long term test.

7.1 Test Environment

This section explains the software and hardware requirements of the systems that are used to perform the long term test. We used 10 virtual machines located in the university to initiate this test. The specifications of the system are explained below:

- Processor : Intel i7-2600
- RAM : 8 GB
- Operating System : openSUSE
- Network Interface : Gigabit Ethernet
- Java VM : 1.7

In addition to these machines, one machine in the University of Paderborn machine pool is used to initiate and controls the tests remotely on all the servers. The specifications of this system is same as the specifications explained above.

7.2 Deployment

This section explains about how to deploy long term test in a system which agrees the specifications mentioned in the section 7.1. Since the long term

test have to run for a long time we used 10 servers which are located in the University of Paderborn. The deployment process contains 6 important steps which will be described in the following subsections:

Step 1: Requirements

We have used *Ecllipse Kepler* integrated development environment to run this test and *SmartGit* repository to synchronize the code. These two tools must be installed in all the servers. The path of the test plan must be configured in the *config.txt* file and the resources folder explained in the section 3.1.1 must be copied manually on all the servers.

Step 2: Test Setup

After all the requirements are installed and copied, the next step is to setup the test environment which is explained in the section 3.1.2. In this testing, we initiated 10 instances in 10 servers which are connected in the same LAN network. Out of these 10 instances, one instance is made as a Master instance and the remaining instances are slave instances. All the slave instances are logged into the network via master instance.

1. ***Kerberos Ticket:*** Every pool machines in the University of Paderborn have a *Kerberos Ticket*. This ticket is used as an authorization service to authorize users, computers or services to other users, hosts or services [6]. The ticket is generated at the first logon, and have a lifetime of only 10 hours. If this ticket is expired, then the machine will restart automatically, and the all the user running processes will be terminated.

   In long term test, one real machine is used to initiate the test and to control remotely on the servers, and therefore it is very important to extend this ticket. So that the kerberos session will be extended and tester need not to use the vncviewer command everytime on his/her next login to display the views of all the 10 instances. The following commands are very helpful to extend the *Kerberos Ticket*.

   - `klist` command is used to view the currently valid tickets[6].
   - `kinit` command is used to request a new ticket granting ticket to extends its service life for next 10 hours[6]. Once this command is entered, it asks the tester to enter the password of the system.

2. ***LifeSocial Parameters:*** The following LifeSocial parameters which are explained in the section 3.5.3 are need to be changed in long term test and these parameters are explained below:

- ***onlyGnuplot:*** There are three different file formats .csv, .dat and GNU Plot.dat that are generated by LifeSocial monitoring function. Out of these 3 files, only one file called GNU Plot.dat is used to plot the graphs. As the long term test runs for several days, the size of all these 3 files will increase gradually and it may leads to disk quota exceeded problem. If this parameter is enabled then the monitoring function only generates the GNU Plot.dat file and ignores the other two files. So, by using this parameter each instance can save storage space.
- ***monitorDataVisual:*** This parameter consumes more memory and it is very important to disable it during the long term test.

Step 3: Permissions

You need require permissions to access and install the tools in all the servers.

Step 4: Secure Connection

Local Port forwarding via SSH tunneling is used to create a secure connection between the local computer and a remote computer [9]. The following command is used to make secure connection:

```
$ ssh -g -L <source port number>:<destination server>:
  <destination port number> <host>
```

Here, the option `-g` is used to allow the remote hosts to connect the local forwarded ports, and option `-L` represents local port forwarding [9].
`X11Forwarding` must be enabled on both the server and client to use the port forwarding in SSH tunneling, and to display the GUI applications. To do this, you need the root permissions of the system.

Step 5: Graphical User Interface

In real machines, monitor is used to render the graphical user interface. As the `LifeSocial` requires a monitor to initiate the test and the servers are placed some where and doesn't have access to the monitors, we used the X display server to work as a monitor in the real machine. This can be done by using XVFB display server and VNC which are explained in detail below.

1. ***XVFB:*** According to *xorg foundation* [11] `XVFB` is virtual frame buffer X server for X version 11 that runs on the machines which doesn't have display hardware and no input devices. To run `Xvfb` display server the `XVFB` display component must be installed in the system. The following Linux command is used to start `Xvfb`

```
$ Xvfb :1 -ac -screen 0 1024x768x8 &
```

In the above command the server will listen for connections as server 1, option `-ac` represents without access control restrictions, and screen 0 with depth 8 1024x768 [11]. Now, the X session is exported by using the following Linux command:

```
$ export DISPLAY=localhost:10.0
```

Here `localhost` represents *hostname*, `10` represents the *display number* and `0` represents the *screen number*. Each host can have one or more displays, and similarly each display can have one or more screens. To make it more easier, the export DISPLAY value must be set to the value of DISPLAY environment variable of your system and the following command is used to print all the environment variables.

```
$ printenv
```

In the first attempt of the long term test XVFB is used to render the graphical user interface of the server, and noticed that when the network disruptions are occurred the screen sessions used in the XVFB are failed to reattach the GUI of the application and the reason for this problem was unclear. For this reason, TightVNC is used in this test.

2. ***VNC:*** VNC or Virtual Network Connection is a remote access software that allows to view and control a graphical desktop environment on a remote machine[10]. In long term test, we used a VNC compatible remote desktop software called TightVNC to remotely access and control the system. This software comes default with openSUSE 12.3 operating system [7]. There are two different components in VNC namely VNC Server and VNC Viewer, and these components allows the tester to view and control the server remotely. This section explain the linux commands that are used to start the VNC server, VNC server configuration and to run the VNC viewer.

   a. **VNC Server:** VNC server component is used to start and stop the VNC session by using the following command.

   ```
   $ vncserver
   ```

   When this command is executed a new 'X' desktop session is created, it shows the host name with the display number, and the log file that is created for the session. This display number is required to kill the session.

```
$ vncserver
New 'X' desktop is graffi-1:2
Starting applications specified in ~/.vnc/xstartup
log file is ~/.vnc/graffi-1:2.log
```

VNC session can be killed by using the following command.

```
$ vncserver -kill :1
```

Here, 1 represents the display number.

b. **VNC Server Configuration:**The VNC server configuration includes the xstrartup scripts and VNC passwords. These files are placed at the VNC directory /.vnc

—**xstartup:** xstartup is a shell script file which specify X applications to run when VNC desktop is starte [10].

```
#!/bin/sh
# Uncomment the following two lines for normal desktop:
# unset SESSION_MANAGER
# exec /etc/X11/xinit/xinitrc
[ -x /etc/vnc/xstartup ] && exec /etc/vnc/xstartup
[ -r $HOME/.Xresources ] && xrdb $HOME/.Xresources
xsetroot -solid grey
vncconfig -iconic &
xterm -geometry 80x24+10+10 -ls -title "$VNCDESKTOP Desktop" &
twm &
```

If you uncomment the lines shown in the above code then the VNC Viewer displays the normal desktop view of the system when VNC desktop is started.

—**psswd:** This file contains the VNC password and the following command is used to change the password of VNC.

```
$ vncpasswd
Password:
Verify:
```

When this command is executed, it asks the user to enter the new password and verify it. The VNC server is need to be restart to apply the changes.

c. **VNC Viewer:** VNC Viewer is used to view the display of the server. The following command is used to display the view of the server.

```
$ vncviewer [Host IP address]:[ListeningPortNumber]
```

After executing the above command, it will ask the user to enter the password of the VNC server. After entering the correct password the session will start and the view is shown in the figure 7.1. It contains a *xterm* terminal emulator to put the window up and interact.

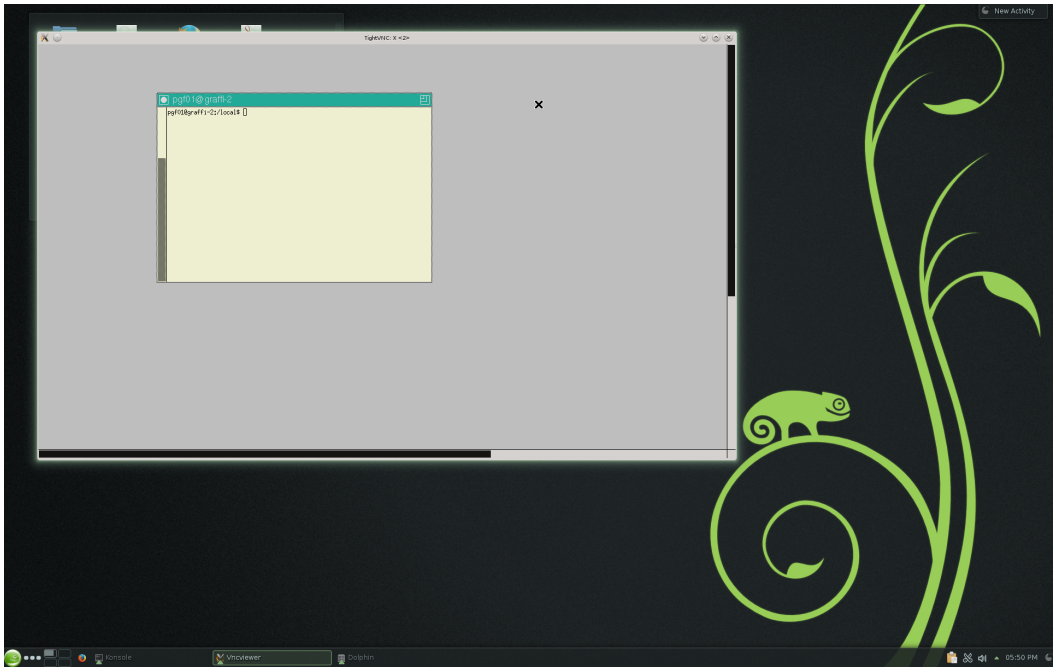


Figure 7.1: Server display in VNC Viewer

Once the test is initiated, the test will start executing in all the servers and the tester can logout the real system which is used to control the servers remotely. On his/her next login, the step 4 must be repeated to connect to the servers and then the VNC command is used to view the display the server.

There are few limitations in VNC Viewer.

—If the vncviewer console window is closed then the server display will close automatically.

—Some times the VNC viewer is failed to export graphical user interface if the VNC session is running for very long time. This can be solved by killing the VNC session and start the VNC Server again.

Step 5: Screens

Generally, in long term test once the test is initiated the user won't use the eclipse or command terminal for several days. If there is any network disruptions then all the running programs of the user were terminated. For this reason, the screen command to store the session. Before initiating the test screen command is used in the *xterm* terminal emulator to start the screen session, and once the test is initiated the tester can safely leave the session by detaching the screen. On next login, the tester can resume the screen by reattaching the screen session.
The following things are very helpful when you are using screens.

- ***Start Screen:*** To start new screen session [8] the following command is used.

  ```
  $ screen
  ```

- ***Screens List:*** Screen list is used to display the list of available screens.

  ```
  $ screen -ls
  ```

- ***Screen Detaching:*** After the test is initiated in the instance, press ctrl+a, d to detach your screen session.

- ***Screen Reattaching:*** To attach the screen the following command is used.

  ```
  $ screen -r
  ```

- ***Kill Screen:*** To kill the screen session, press ctrl+a, k then it will ask for user confirmation, and press 'y' to confirm it.

Step 6: Implementation

This subsection explains the implementation details of the long term test.

- ***Delete old data before running the test***

  As mentioned in section 5.2 Step 3, `LifeSocial` generates the storage and monitoring statistics files when it is launched, these files must be deleted before running the long term test. The shell scripts present in the `/com.p2pframework.tester/Shell_Scripts/LongTermTest` is used to remove these files.

- ***Initiate timed test plan***

  As mentioned in Section 3.1.2, there are two ways to initiate the timed test plan. In this test, the eclipse console of the master instance is used

to send the timed test plan to all the slave instances through *Execute-TimedTestPlan* OSGi command. There are two reason behind using this way. First we are not using the normal desktop view in VNC, therefore the *TestCase Delegater* shown in the figure 5.2 is unavilable to send the command. Second, if there are any errors during the execution of the test the tester can easily notice it in the console window of the eclipse. This way helps the tester during the results evaluation.

- ***Stop monitoring function after running the test***

  After performing the test, the monitoring function must be manually stopped on all the instances. This can can done by executing the OSGi command *monitor stop* in the eclipse console of the instance. If this step is missed then the monitoring data might be empty or causes some data loss.

7.3 Test Plan

The goal of this test plan is to generate the data traffic and monitor the behaviour of the P2P network in long run. Timed test plan is used to run the test for long time by adding the fixed time bound to the test cases. It is very important that the the test plan should be designed in a way that keeps the instances away from the `Idle` state. As same as in synthesis test, the command *SendfriendrequestInorder* must be send to all the slaves instances before the *ExecuteTimedTestPlan* command is send. The test plan shown in the figure 7.2 and 7.3 is one period of test plan which is capable of run the test for 970 minutes. This test plan will be repeated for tens of times to make the test run for 7 days. Anyhow, these instances are crashed after running for several days due to queue overflow exception caused by `PriorityTransportLayer` in FreePastry.

In this test plan, at first *Wait* test case is executed for 20 minutes, after that the functional test cases are executed in order as defined in the test plan. There is a 20 minutes *Wait* time between the two functional test cases to release the unused memory. The test cases of *photo plugin* and *vote plugin* mentioned in the table 3.5 and 3.6 are not included in this test plan. The *photo plugin* test cases are normally working in synthesis test but during the long term test it is causing some problems with *X display*. The reason for this problem is explained in the section 7.6. Since *TestCaseUploadPhoto* is the precondition for other test cases like *TestCaseDeletePhoto*, these test cases are also not included in the test plan. When *vote plugin* test cases are running after a day or 2 days run of test plan, some instances are get struck and the reason was unclear. Therefore,

these test cases are also not included in the test plan.

```
1x Wait 20m
1x TestCaseChangeProfilePicture 10m
1x Wait 20m
1x TestCaseChangeProfileInformation 10m
1x Wait 20m
1x TestCaseViewOwnProfile 10m
1x Wait 20m
1x TestCaseViewFriendsProfile 10m
1x Wait 20m
5x TestCaseSendWallPost 10m
1x Wait 20m
5x TestCaseCommentWallPost 10m
1x Wait 20m
5x TestCaseViewOwnWall 10m
1x Wait 20m
5x TestCaseViewFriendsWall 10m
1x Wait 20m
5x TestCaseSendMessage 10m
1x Wait 20m
5x TestCaseViewOutboxMessage 10m
1x Wait 20m
5x TestCaseViewInboxMessage 10m
1x Wait 20m
2x TestCaseDeleteInboxMessage 10m
1x Wait 20m
2x TestCaseCreateGroup 10m
1x Wait 20m
1x TestCaseViewForum 10m
1x Wait 20m
5x TestCaseCreateForumThread 10m
1x Wait 20m
5x TestCaseCommentForumThread 10m
1x Wait 20m
5x TestCaseViewForumThread 10m
1x Wait 20m
5x TestCaseCreateFolder 10m
1x Wait 20m
2x TestCaseViewGroup 10m
1x Wait 20m
2x TestCaseRenameFolder 10m
1x Wait 20m
5x TestCaseUploadFileinFolder 10m
1x Wait 20m
```

Figure 7.2: Long term test timed test plan (1)

```
5x TestCaseViewFolder 10m
1x Wait 20m
1x TestCaseLeaveGroup 10m
1x Wait 20m
1x TestCaseSendMultiChatInvitation 10m
1x Wait 20m
5x TestCaseSendMultiChatMessage 10m
1x Wait 20m
1x TestCaseLeaveMultiChatMessage 10m
1x Wait 20m
5x TestCaseSearch 10m
1x Wait 20m
1x TestCaseViewCalendar 10m
1x Wait 20m
5x TestCaseCreateCalendarEvent 10m
1x Wait 20m
2x TestCaseEditCalendarEvent 10m
1x Wait 20m
2x TestCaseDeleteCalendarEvent 10m
1x Wait 60m
```

Figure 7.3: Long term test timed test plan (2)

7.4 Monitoring and Data Processing

The monitoring and data processing in long term test is same as synthesis test explained in the section 5.4. The shell scripts for gathering the monitoring are present in */com.p2pframework.tester/Shell_Scripts/LongTermTest*. The target storage folder path of the real system to store the monitoring data must be configured manually in these shell scripts. Finally, from the gathered data using *OnlyPlotOneNodeGlobal.java* file the .plt scripts are generated and the graphs are produced by GNUPlot.

7.5 Issues in Long Term Test

This section explains about the issues in the long term test.

- ***Memory consumption:*** During the long term test, a memory leak in *MonitoringPluginHandler* is detected. A memory analyzer tool called *Jprofiler* is used to find the cause of the memory leak. It was caused by the *historysize* in *MonitoringPluginHandler*. Before this thesis, the *historysize* value was set to 200 and the size of the each monitoring snapshot is increasing as all the nodes keeps the metrics in the memory for last 200 seconds. Because of this problem the *OutOfMemoryError* exception

is thrown. After changing this value to least size i.e, 1, all the instances are consuming very less memory after executing the functional test case as shown in the figure 7.4. However, some times during the execution of test cases the memory consumption is highly fluctuating in sudden on all the instances and the reason was unclear. Also, the memory usage is slowly increasing all the time during the test and by analyzing we found that it is caused by the *FreePastry* overlay.

```
Executed test case: TestCaseViewGroup
Executed test case TestCaseViewGroup
Finished repetitions, waiting for timeslot to be finished.
Repetitions: 1, testCase: Wait, time: 1200000.0
Executing testcase Wait
Waiting for 0.1 minutes.
Total Memory: 1979.75 MB
Free Memory: 1894.5607147216797 MB
Used Memory: 85.18928527832031 MB
Total Memory: 1791.36328125 MB
Free Memory: 1775.7193908691406 MB
Used Memory: 15.643890380859375 MB
Finished repetitions, waiting for timeslot to be finished.
```

Figure 7.4: Memory consumption in slave instance

- ***Queue Overflow Exception:*** This exception is also discovered in previous thesis, after the long term test run for 3 days the messages are lost on all the nodes because the message queue was overflowed. This exception is caused by *PriorityTransportLayer* in free pastry. In [2] the reason for this exception is explained. According to them, when more or larger messages are sending than the available bandwidth then the FreePastry queues the messages. To reduce the message latency the FreePastry uses the limited queue length and when the queue overflows it drops the least priority message in the queue. This can be solved by increasing the length of the queue via *pastry_socket_writer_max_queue_length* Freepastry parameter. However, this is not the ideal solution because it increases the message latency.
- ***Exception in executing TestCaseUploadPhoto:*** A *java.lang.InternalError* exception is discovered during the execution of test case *TestCaseUploadPhoto*. When this test case is executing all the instances are not able to connect to the X11 window server with display variable's value, and the test process get struck. In Linux and Unix operating system, to run the graphic related methods in X server the java products needs access to the *Java.AWT* library and this is reason behind

this problem. This can be done by setting the *java.awt.headless* system property to *true* [3]. Headless system property is used to configure the system when display is lacking.

7.6 Evaluation

The long term test is performed in 10 instances and in this section, the results of long term test are evaluated.

Figure 7.5 shows the total number of nodes in the network during the 10-instance long term test. As we can see that the average number nodes is 9, at around 1200 minutes, and from the period 1200 to 1700 minutes the nodes count falls to 7. Till the time around 3600 minutes the node count is 7 and in the end all the nodes are active. This means that some nodes are disconnected from the network during the 1200 to 3600 minutes period and in the end there are no loss of nodes in the network. Figure 7.6 shows the average time that the user joins in the network during the 10-instance long term test. The average join network time is less than 0.4 second.

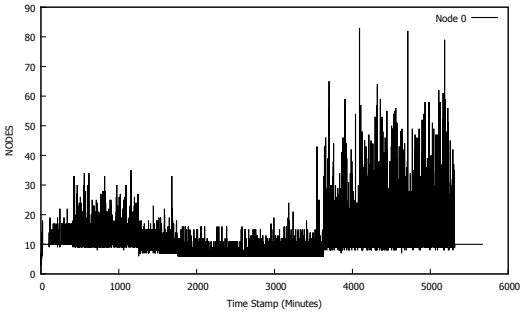


Figure 7.5: Global Network Nodes - 10 instances

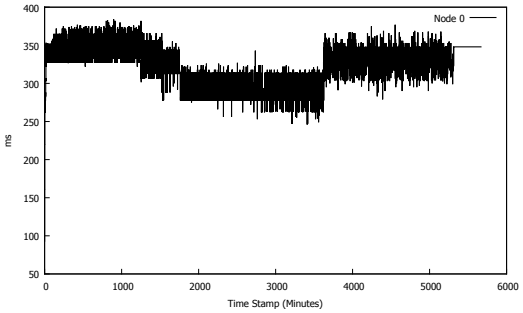


Figure 7.6: Global Join Network Time - 10 instances

Figure 7.7 shows the total number of messages that are sent by the message dispatcher and the figure 7.8 shows the total number of messages that are received by the message dispatcher. By observing both of these graphs we can see that both are looking almost same and this because of the lost messages.

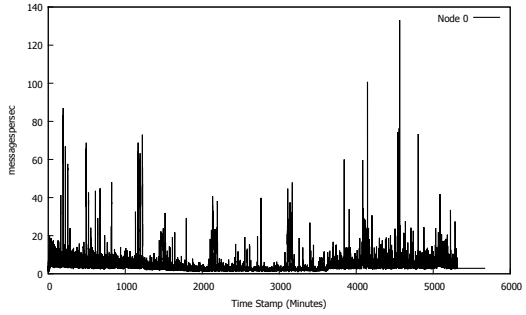


Figure 7.7: Global Message Dispatcher Sent Messages - 10 instances

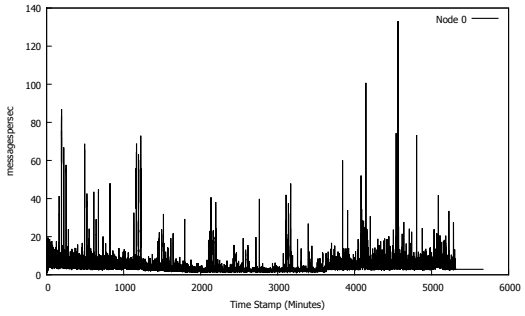


Figure 7.8: Global Message Dispatcher Received Messages - 10 instances

Figure 7.9 shows the total number of lost messages in the long term test. By observing the figure we can see that there are arounf 100 to 150 messages are lost during the test. The message are lost because of the message queue overflow problem explained in the section 7.5. Figure 7.10 shows the global network memory usage in long term test. By observing the graph we can see that the memory usage is increasing slowly all the time during the long term test. Memory usage size is reached to 1200 MB before the test is crashed. This is caused by FreePastry overlay as explained in the section 7.5.

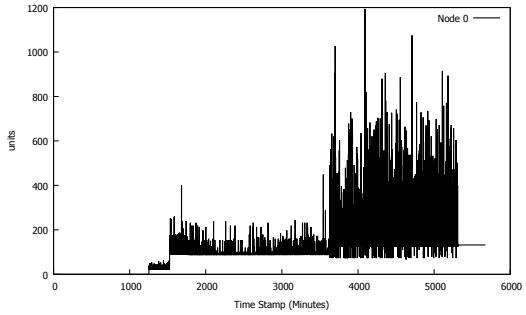


Figure 7.9: Global Lost Messages - 10 instances

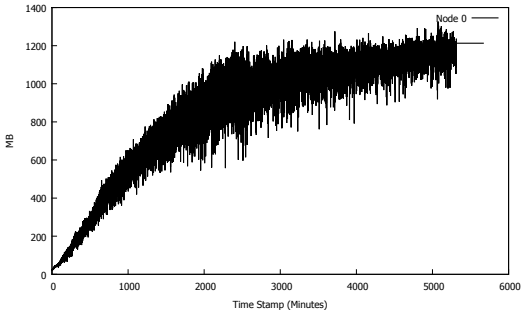


Figure 7.10: Global Network Memory Usage Size - 10 instances

Figure 7.11 and Figure 7.12 shows the other exceptions and null pointer exceptions in long term term. As we can see that there are around 250 other exceptions and around 300 null pointer exceptions. By observing the graphs we can notice that the exceptions are increasing till the end of the execution because in long term test the test plan is executing for several times and exceptions are caused in every period of test plan.

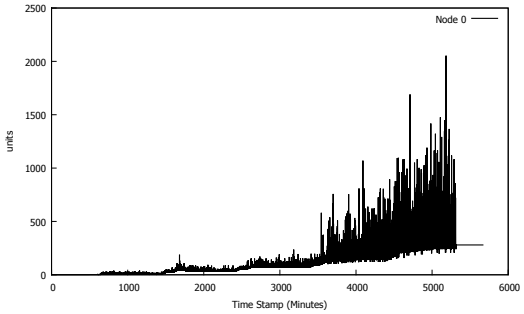


Figure 7.11: Global Other Exception - 10 instances

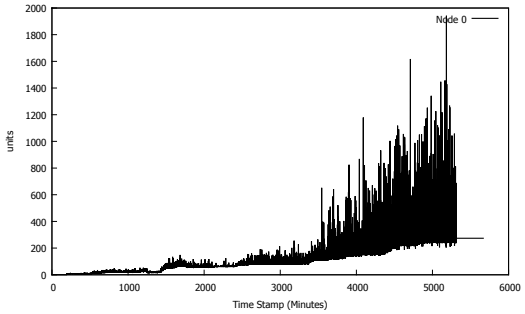


Figure 7.12: Global Null Pointer Exceptions - 10 instances

Figure 7.13 and Figure 7.14 shows the network pastry sent and receive rate. The values are almost equal in both the tests, and the data rate is very low. Because of the loose test plan both sent and received data rate is managed to keep around 600 kbytes.

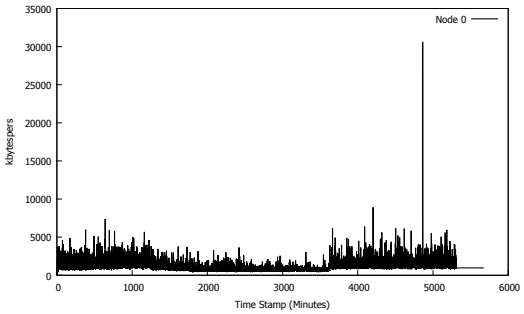


Figure 7.13: Global Network Pastry Data Sent Rate- 10 instances

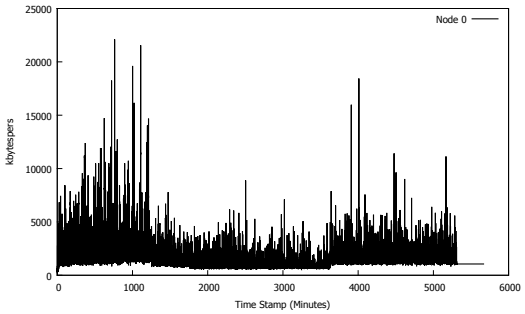


Figure 7.14: Global Network Pastry Data Received Rate - 10 instances

Figure 7.15 and Figure 7.16 shows the size of data stored and received by the storage dispatcher in long term test. By observing the graphs we can see that both the graphs are not equal and the size sent and received is also incorrect. The reason behind this problem is that one of the metrics is monitoring wrong objects and it need to be fixed.

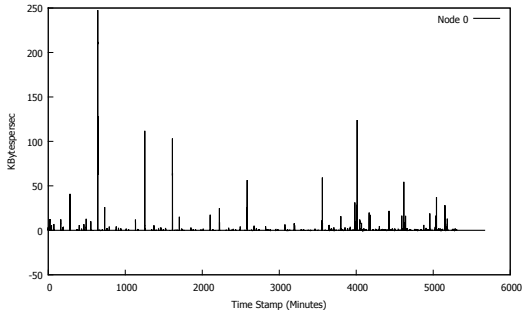


Figure 7.15: Global Storage Dispatcher Sent Data - 10 instances

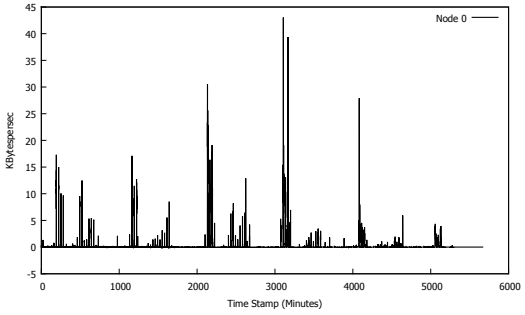


Figure 7.16: Global Storage Dispatcher Received Data - 10 instances

7.7 Conclusion

Overall performance of long term test is satisfiable as the current version of `LifeSocial` is able to run for *5300 minutes = 88.3 hours = 3.67 days*. After 5300 minutes, the message queue was overflowed on all the nodes and terminating the test execution. In previous master thesis, the long term test is only managed to run for 25 hrs. The common issue in both the thesis is the queue overflowed exception caused by `PriorityTransportLayer` in FreePastry. The memory consumption is also slowly increasing during the test and it is also caused by the FreePastry overlay.

8 Conclusion

This chapter is written by W.Peng. It sums up the achievements of this master thesis, and presents the remaining issues that need further enhancement in the future.

8.1 Achievement

The overall goal of this master thesis is to systematically evaluate a decentralized social network, `LifeSocial`. For reaching this goal, four kinds of tests are considered in the original plan, namely *synthesis test*, *random test*, *long term test* and *real user test*. *Synthesis test* is used to examine if each functionality of `LifeSocial` is working correctly. *Random test* is designed to examine the robustness of `LifeSocial` when users join and leave the network frequently and execute actions randomly. *Long term test* is for verifying the stability and reliability of `LifeSocial` after it runs for a long time. If `LifeSocial` performs well in these three tests, *real user test* is executed at last, in order to get feedback from real users and improve the social network further. But, before all of these tests, lots of test cases need to be handled. Since `LifeSocial` was changed greatly by P2P Framework Project Group 2013 from Uni-Paderborn, most of original test cases become invalid. These invalid test cases need to be modified according to the corresponding functionalities of current `LifeSocial`. Besides, a series of test cases require to be created for new features.
So, all of the previously existing test cases have been examined in the first place, and it was found that only 7 of 26 test cases were still working properly. Other 19 test cases were invalid due to several reasons, such as the invoked function was replaced by a new one, or the related plugin was totally reconstructed. As a result, these invalid test cases have been deleted or modified based on the new `LifeSocial`. Also, plenty of new test cases have been created for majority of fresh features and plugins, so as to cover as many functionalities of current `LifeSocial` as possible. Meanwhile, a number of social metrics have been added to monitor all sorts of actions made by user in the network. It is useful for analysing user behaviours after test.
Then, the first round of *synthesis test* and *random test* have been performed separately. The result of synthesis test is quite good. Every testing machine

finishes the test plan successfully, and there is no lost message during the test. Besides, either to join the network or to look up a item in the network is very quick, which provides a good user experience. The only problem of synthesis test is the exceptions occurred during the test, which lead some test cases to failure. And the exception issue also exists in random test. In addition, there are two more severe problem in the random test with churn. First, some of testing nodes get stuck or crashed during the test. Second, hundreds of messages get lost during the test.

Next, based on the problems happened in these tests, two main improvements have been made. The independence of most test cases have been improved in order to avoid various unexpected exceptions during test. And a few of existing metrics have been modified for collecting more accurate monitoring statistics and being more understandable.

Afterwards, the second round of *synthesis test* and *random test* have been done. According to the results, it is found the modifications of metrics are pretty successful. All the metrics measure the correct objects now, and the names of them are much more precise. Also, the improvements of test case have reduced the amounts of exceptions effectively in both synthesis test and random test.

Additionally, *Long term test* has been executed in parallel with synthesis and random test. The result of *Long term test* is satisfiable, since it is able to run for more than 88 hours, which is much longer than the test in the previous thesis.

As there still are some problems not fixed in the tests that are mentioned above, `LifeSocial` is regarded not stable enough. Consequently, *real user test* is not performed in this thesis.

8.2 Future Work

First, although the exception issue has been largely enhanced, there still remain some null pointer exceptions and other exceptions in all kinds of test. And the exceptions seem most frequently occur in the test cases related to Groups Plugin and Wall Plugin. These test cases and involved functions need to be well examined.

Second, the problems of crashed node and lost message in churned random test are not fixed yet. In this thesis, we tried cleaning the global variables in *ParameterGenerator* after every logout, and deleting the sleep time after every re-login, but the problems still remain. Therefore, the mechanism of random mode needs to be reconsidered.

Third, the `LifeSocial` network is still not able to last for a long time due to the queue overflowed exception caused by *PriorityTransportLayer* in FreePastry,

as well as the increasing memory consumption of FreePastry overlay. So, these issues also require to be improved in the future.

List of Figures

2.1 LifeSocial Architecture[15] 8
2.2 GUI of user photos page in LifeSocial application[15] 14

3.1 Master View of LifeSocial Instance 21
3.2 Delegater View of LifeSocial Instance 21
3.3 Flowchart of executeTestCase() method implemented by each Test Case 23
3.4 Implementation of abstract class Precondition 26
3.5 TestCaseImplementation 27
3.6 Implementation of test case TestCaseDeleteCalendarEvent 29
3.7 Implementation of class PreconditionCalendarLoaded 30
3.8 Implementation of class CommandDeleteEvent 31
3.9 Component definition of CommandDeleteEvent.xml 32
3.10 Component definition of CommandDeleteEvent.xml file 32
3.11 TestComponent.xml in Test Plugin 33
3.12 Calendar plugin MANIFEST.MF file 34
3.13 Value of newrate [18] 44
3.14 Value of currentrate [18] 44
3.15 Global number of stored replications 44
3.16 Data flow through LifeSocial 47
3.17 LifeSocial parameters (1) 53
3.18 LifeSocial parameters (2) 54

4.1 Message routing in SkyEye.KOM layer 56
4.2 SkyEye.KOM ID Domain 57

5.1 Parameters in Config.txt 61
5.2 TestCase Delegater 62
5.3 Timed test plan (1) 64
5.4 Timed test plan (2) 65
5.5 Global Nodes - 10 instances 67
5.6 Global Nodes - 20 instances 67
5.7 Join Network Time - 10 instances 67
5.8 Registration Time - 10 instances 67

5.9 Join Network Time - 20 instances 68
5.10 Registration Time - 20 instances 68
5.11 Global MessageDispatcher Sent Messages - 10 instances 68
5.12 Global MessageDispatcher Received Messages - 10 instances . . . 68
5.13 Global MessageDispatcher Sent Messages - 20 instances 69
5.14 Global MessageDispatcher Received Messages - 20 instances . . . 69
5.15 Global MessageDispatcher Sent Data - 10 instances 69
5.16 Global MessageDispatcher Received Data - 10 instances 69
5.17 Global MessageDispatcher Sent Data - 20 instances 70
5.18 Global MessageDispatcher Received Data - 20 instances 70
5.19 Global Network Sent Messages - 10 instances 71
5.20 Global Network Received Messages - 10 instances 71
5.21 Global Network Sent Messages - 20 instances 71
5.22 Global Network Received Messages - 20 instances 71
5.23 Global StorageDispatcher Sent Data - 10 instances 72
5.24 Global StorageDispatcher Received Data - 10 instances 72
5.25 Global StorageDispatcher Sent Data - 20 instances 73
5.26 Global StorageDispatcher Received Data - 20 instances 73
5.27 Global Storage Used Size - 10 instances 73
5.28 Global Storage Used Size - 20 instances 73
5.29 Global Framework Sent Data - 10 instances 74
5.30 Global Framework Received Data - 10 instances 74
5.31 Global Framework Sent Data - 20 instances 75
5.32 Global Framework Received Data - 20 instances 75
5.33 Global Missed Calculation - 10 instances 75
5.34 Global Missed Calculation - 20 instances 75
5.35 Global Null Pointer Exception - 10 instances 76
5.36 Global Other Exception - 10 instances 76
5.37 Global Null Pointer Exception - 20 instances 76
5.38 Global Other Exception - 20 instances 76
5.39 Global Average Lookup Time - 20 instances 77
5.40 Global Average Lookup Time - 20 instances 77
5.41 Global Wall Entry - 10 instances 77
5.42 Global Wall Entry - 20 instances 77
5.43 Global Wall Comment - 10 instances 78
5.44 Global Wall Comment - 20 instances 78
5.45 Global View Outbox - 10 instances 78
5.46 Global View Outbox - 20 instances 78
5.47 Global Sent Chat Message - 10 instances 79
5.48 Global Sent Chat Message - 20 instances 79
5.49 Global Search - 10 instances 79
5.50 Global Search - 20 instances 79

5.51 Global Calendar Event - 10 instances . . . 80
5.52 Global Calendar Event - 20 instances . . . 80
5.53 Global Deleted Voting - 10 instances . . . 80
5.54 Global Deleted Voting - 20 instances . . . 80
5.55 Global Uploaded Photo - 10 instances . . . 81
5.56 Global Uploaded Photo - 20 instances . . . 81
5.57 Global Uploaded File - 10 instances . . . 81
5.58 Global Uploaded File - 20 instances . . . 81
5.59 Global StorageDispatcher Stored Data - 10 instances . . . 83
5.60 Global StorageDispatcher Retrieved Data - 10 instances . . . 83
5.61 Global StorageDispatcher Stored Data - 20 instances . . . 83
5.62 Global StorageDispatcher Retrieved Data - 20 instances . . . 83
5.63 Global Framework Sent Data - 10 instances . . . 83
5.64 Global Framework Received Data - 10 instances . . . 83
5.65 Global Framework Sent Data - 20 instances . . . 84
5.66 Global Framework Received Data - 20 instances . . . 84
5.67 Global Null Pointer Exception - 10 instances . . . 84
5.68 Global Other Exception - 10 instances . . . 84
5.69 Global Null Pointer Exception - 20 instances . . . 85
5.70 Global Other Exception - 20 instances . . . 85
5.71 Global Wall Comment - 10 instances . . . 85
5.72 Global Wall Comment - 20 instances . . . 85
5.73 Global Sent Chat Message - 10 instances . . . 86
5.74 Global Sent Chat Message - 20 instances . . . 86
5.75 Global Uploaded File - 10 instances . . . 86
5.76 Global Uploaded File - 20 instances . . . 86

6.1 Online Time Distribution [17] . . . 88
6.2 Start Script . . . 89
6.3 Category Transition Probability [17] . . . 90
6.4 Global Nodes - 10 instances . . . 93
6.5 Global Nodes - 20 instances . . . 93
6.6 Join Network Time - 10 instances . . . 93
6.7 Join Network Time - 20 instances . . . 93
6.8 Global Lost Messages - 10 instances . . . 94
6.9 Global Lost Messages - 20 instances . . . 94
6.10 Global MessageDispatcher Sent Messages - 10 instances . . . 94
6.11 Global MessageDispatcher Received Messages - 10 instances . . . 94
6.12 Global MessageDispatcher Sent Messages - 20 instances . . . 95
6.13 Global MessageDispatcher Received Messages - 20 instances . . . 95
6.14 Global Network Sent Messages - 10 instances . . . 95
6.15 Global Network Received Messages - 10 instances . . . 95

6.16 Global Network Sent Messages - 20 instances 96
6.17 Global Network Received Messages - 20 instances 96
6.18 Global Null Pointer Exception - 10 instances 96
6.19 Global Other Exception - 10 instances 96
6.20 Global Null Pointer Exception - 20 instances 97
6.21 Global Other Exception - 20 instances 97
6.22 Global Average Lookup Time - 10 instances 97
6.23 Global Average Lookup Time - 20 instances 97
6.24 Global Nodes - 10 instances 98
6.25 Global Nodes - 20 instances 98
6.26 Join Network Time - 10 instances 98
6.27 Join Network Time - 20 instances 98
6.28 Global MessageDispatcher Sent Messages - 10 instances 99
6.29 Global MessageDispatcher Received Messages - 10 instances . . . 99
6.30 Global MessageDispatcher Sent Messages - 20 instances 99
6.31 Global MessageDispatcher Received Messages - 20 instances . . . 99
6.32 Global Network Sent Messages - 10 instances 100
6.33 Global Network Received Messages - 10 instances 100
6.34 Global Network Sent Messages - 20 instances 100
6.35 Global Network Received Messages - 20 instances 100
6.36 Global Null Pointer Exceptions - 10 instances 101
6.37 Global Other Exceptions - 10 instances 101
6.38 Global Null Pointer Exceptions - 20 instances 101
6.39 Global Other Exceptions - 20 instances 101
6.40 Global Average Lookup Time - 10 instances 102
6.41 Global Average Lookup Time - 20 instances 102
6.42 Global Nodes - 10 instances 103
6.43 Global Nodes - 20 instances 103
6.44 Global Lost Messages - 10 instances 104
6.45 Global Lost Messages - 20 instances 104
6.46 Global Null Pointer Exceptions - 10 instances 104
6.47 Global Other Exceptions - 10 instances 104
6.48 Global Null Pointer Exceptions - 20 instances 104
6.49 Global Other Exceptions - 20 instances 104
6.50 Global Null Pointer Exceptions - 10 instances 105
6.51 Global Other Exceptions - 10 instances 105
6.52 Global Null Pointer Exceptions - 20 instances 105
6.53 Global Other Exceptions - 20 instances 105

7.1 Server display in VNC Viewer 112
7.2 Long term test timed test plan (1) 115
7.3 Long term test timed test plan (2) 116

7.4 Memory consumption in slave instance 117
7.5 Global Network Nodes - 10 instances 118
7.6 Global Join Network Time - 10 instances 118
7.7 Global Message Dispatcher Sent Messages - 10 instances 119
7.8 Global Message Dispatcher Received Messages - 10 instances . . .119
7.9 Global Lost Messages - 10 instances 119
7.10 Global Network Memory Usage Size - 10 instances 119
7.11 Global Other Exception - 10 instances 120
7.12 Global Null Pointer Exceptions - 10 instances 120
7.13 Global Network Pastry Data Sent Rate- 10 instances 120
7.14 Global Network Pastry Data Received Rate - 10 instances 120
7.15 Global Storage Dispatcher Sent Data - 10 instances 121
7.16 Global Storage Dispatcher Received Data - 10 instances 121

List of Tables

2.1 Vote Plugin OSGi Commands . . . 9
2.2 Messaging Plugin OSGi Commands . . . 10
2.3 Wall Plugin OSGi Commands . . . 10
2.4 Chat Plugin OSGi Commands . . . 10
2.5 Photos Plugin OSGi Commands . . . 11
2.6 Groups Plugin OSGi Commands . . . 11
2.7 Files Plugin OSGi Commands . . . 12
2.8 Friends Plugin OSGi Commands . . . 12
2.9 Calender Plugin OSGi Commands . . . 12
2.10 Search Plugin OSGi Commands . . . 13
2.11 Profile Plugin OSGi Commands . . . 13
2.12 Login Plugin OSGi Commands . . . 13

3.1 Files in the resource folder . . . 20
3.2 Resource folder location path in different operating systems . . . 20
3.3 Implemented Preconditions . . . 25
3.4 Status of old test cases . . . 35
3.5 Table of modified test cases . . . 37
3.6 Table of new test cases (1) . . . 38
3.7 Table of new test cases (2) . . . 39
3.8 Table of new test cases (3) . . . 39
3.9 Table of new test cases (4) . . . 40
3.10 Table of new test cases (5) . . . 41
3.11 New preconditions . . . 42
3.12 Previous social metrics for LifeSocial plugins . . . 45
3.13 Current social metrics for LifeSocial plugins . . . 46
3.14 Added and removed metrics . . . 48
3.15 Former names of renamed metrics . . . 48
3.16 Current names of renamed metrics . . . 48

4.1 Aggregation strategies . . . 56

6.1 Test Case Registry . . . 91

Bibliography

[1] Freepastry. http://www.freepastry.org, 2009.

[2] Freepastryfaq. https://trac.freepastry.org/wiki/FreePastryFAQ, 2009.

[3] Using headless mode in the java se platform. http://www.oracle.com/technetwork/articles/javase/headless-136834.html, 2009.

[4] Osgi in action. RichardS.Hall\andKarlPauls\andStuartMcCulloch\andDavidSavage, 2011.

[5] Declarative osgi services. http://www.vogella.com/tutorials/OSGiServices/article.html, 2014.

[6] Kerberos ticket. https://www2.math.uni-paderborn.de/rechnerbetrieb/informationen-a-z/sicherheit/passwoerter-und-authorisierung.html, 2014.

[7] opensuse operating system. https://www.opensuse.org/en/, 2014.

[8] Screen user's manual. http://www.gnu.org/software/screen/manual/screen.html, 2014.

[9] Ssh portforwarding. https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding, 2014.

[10] Virtual network computing. http://en.wikipedia.org/wiki/Virtual_Network_Computing, 2014.

[11] xorg foundation. http://www.x.org/archive/current/doc/man/man1/Xvfb.1.xhtml, 2014.

[12] Kalman Graffi Aleksandra Kovacevic Song Xiao Ralf Steinmetz. Skyeye.kom: An information management over-overlay for getting the oracle view on structured p2p systems. Technische Universität Darmstadt, 2008.

[13] Kalman Graffi Dominik Stingl Julius Ru Ìˆckert Aleksandra Kovacevic Ralf Steinmetz. Monitoring and management of structured peer-to-peer systems. Technische Universität Darmstadt, 2009.

[14] Benevenuto Fabrício, Rodrigues Tiago, Cha Meeyoung, and Almeida Virgílio. Characterizing user behavior in online social networks. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 49–62, New York, NY, USA, 2009. ACM.

[15] Christy Jovita Bibiana Arockiaraj Kailaash Balachandran Navendu Barua Bing Gao Eshant Tulsidas Gour Christopher Holomek Thomas Jacobs Siddharth Kumar Bhagvanjibhai Kaneriya Maitri Modasia Weifeng Peng Chander Prakash Manasa Veena Rapolu Florian Schulte Rohan Sukumaran. Final report - a peer-to-peer framework for social networks, 2013.

[16] Andreas Eberhart and Stefan Fischer. *Web Services - Grundlagen und praktische Umsetzung mit J2EE und .NET*. Carl Hanser Verlag Mn̈chen, 2003.

[17] Pascal Kallien. Systematic evaluation of a peer-to-peer framework in the use case of a social network. Master thesis, Uni-Paderborn, 2013.

[18] Kangping Liu. Benchmarking a distributed framework for social networks. Master thesis, Uni-Paderborn, 2013.