**Project 2 Multi Server Network System User Guide & Test**

This is an advanced version of project 1, which achieves high availability and eventual consistency. Detailed user guide and test report will be presented in following sections.

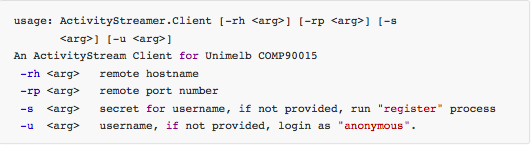
1. **System User Guide**

* System Set Up

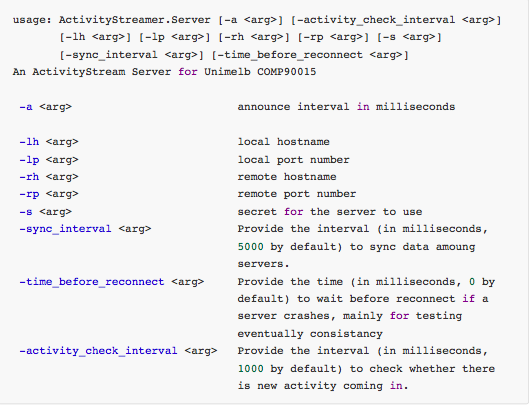
There are two JAR files in source code package, ActivityStreamerClient.jar and ActivityStreamerServer.jar.

Specific configurations for Client and Server are as below:

* Client



* Server



1. **Test Scenario**

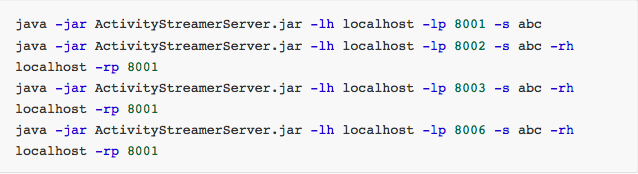
Per project specification, this system is supposed to achieve following functions:

* High Availability: system can reconnect automatically after network partition
* Clients can join (register/login) and leave (logout) the network at any time, Servers can join the network at any time
* Unique Register: a given username can only be registered once over the server network
* Message ensure: a message sent by a client can reach all clients that are connected to the network at the time
* Message order: all activity messages sent by a client are delivered in the same order at each receiving client
* Load balancing: clients are evenly distributed over the servers

Our implementation for delivering activity to clients is synchronous, so you may need to wait a period of time before you can actual receive an activity, default period is 1 second.

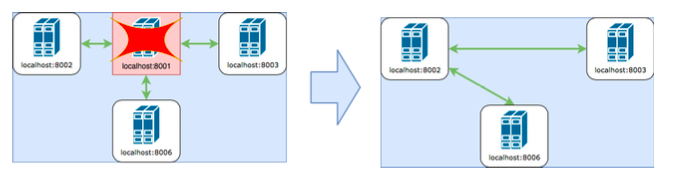
Six scenarios have been designed for test case.

* 1. **High Availability**
* Operations:
  + Strat 4 servers (commands can be copied from README.md directly in terminal)



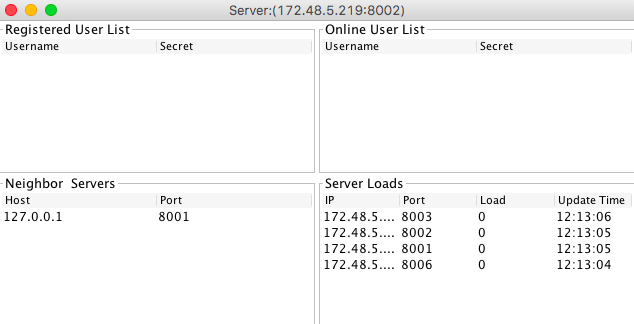
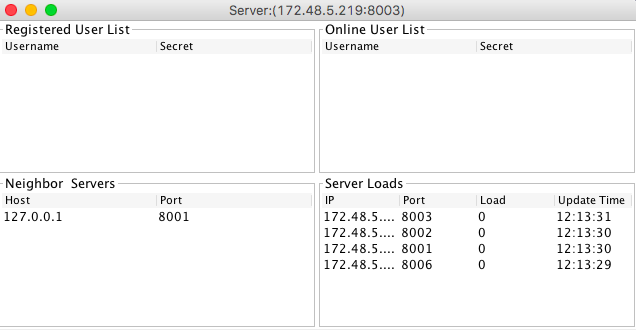
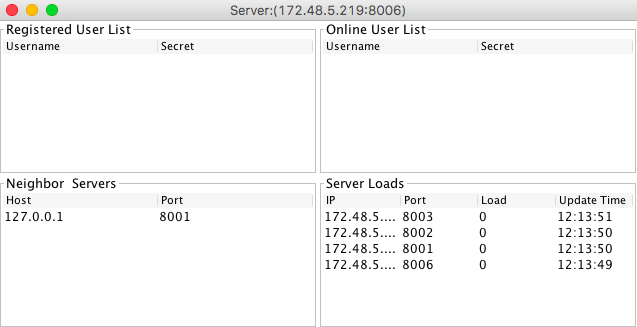
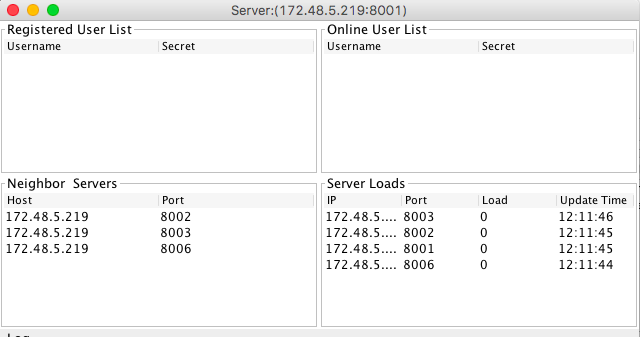
* + Force quit server 8001 (close server GUI or press CTRL+C in terminal)
* Expected Result

Server 8002, 8003, 8006 will automatically connect as below picture shows:



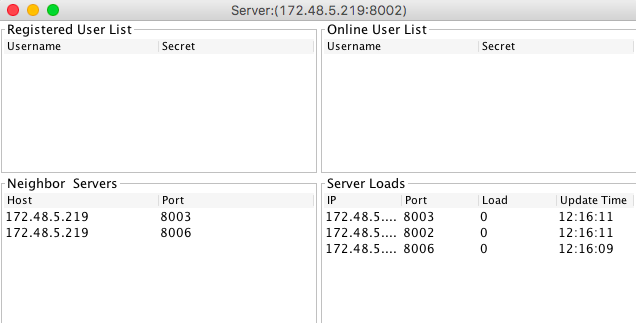
* Actual Result

Snapshot of starting 4 servers:



Snapshot of reconnection:

8002 is the first server in 8001’s backup list, after 8001 quit, 8003 and 8006 will automatically connect to 8002.



* 1. **Client/Server join and leave the network**
* Operations
  + Start the very first server



* Register a user at 8001 and remember its secret



* + Quit client of last step (close GUI or press CTRL+C in terminal)
  + Start a new server and connect it to 8001



* Login user1 at new server 8002 (replace $secret by actual secret)

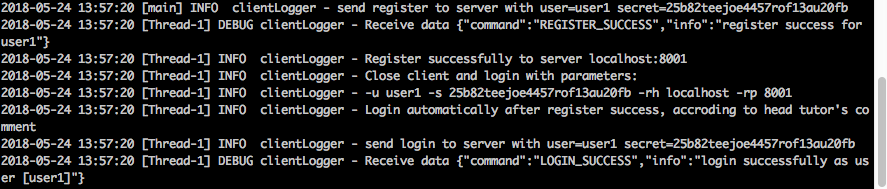


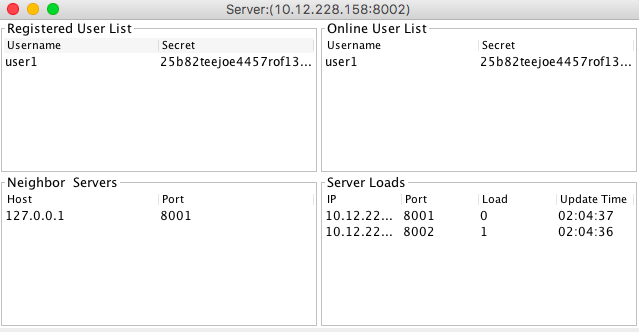
* Expected Result

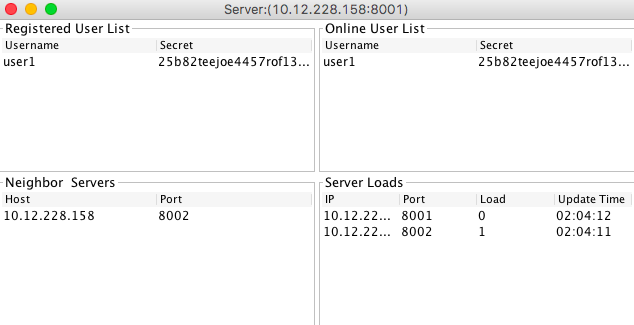
User1 should login on new server 8002 successfully, and all data of 8002 should be consistent with 8001.

* Actual Result

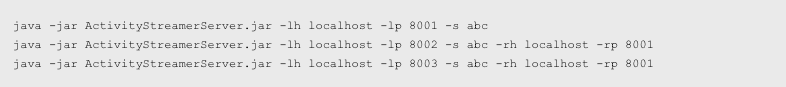
Snapshot of register success and auto login with given secret



Snapshot of user1 relogin on 8002 (user1 login successfully, 8001 and 8002 is consistent)



* 1. **Unique Register**
* Operations
  + Start 3 servers



* + Register user1 at server 8001



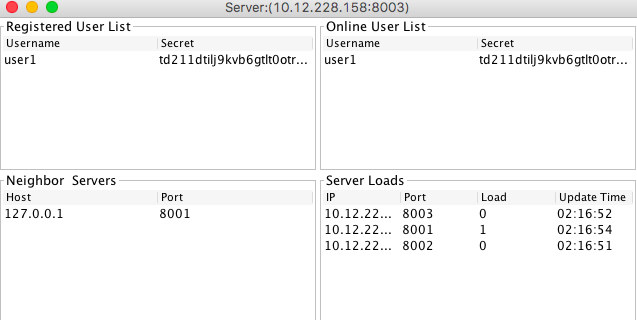
* + Try to register user1 at server 8002

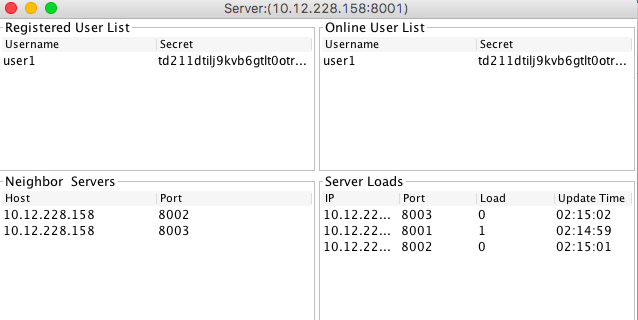


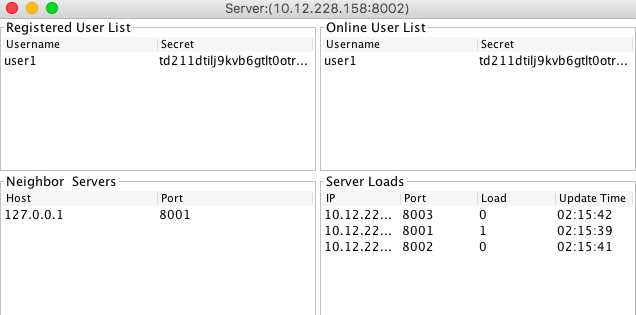
* Expected Result

Register at server 8002 should fail with error message.

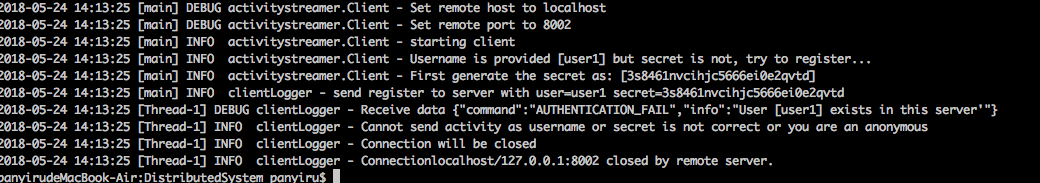
* Actual Result

Snapshot of 3 servers’ GUI





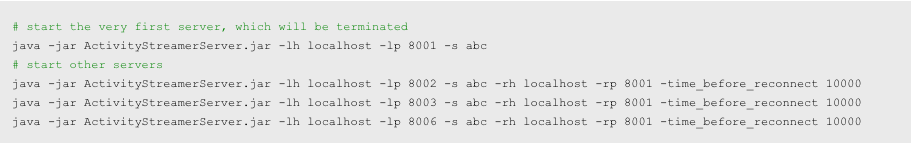
Snapshot of error message (user1 already exists in server)



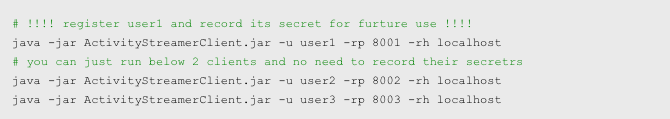
* 1. **Message Ensure**

In order to simulate message loss case, start server with a parameter to delay the reconnection function.

* Operations
  + Start 4 servers with time\_before\_reconnect=10000 (10s)



* + Connect 3 clients to 3 different servers



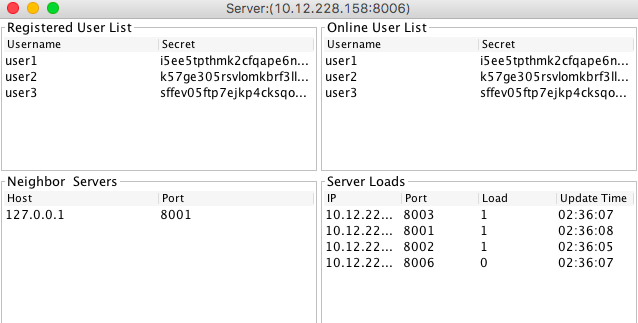
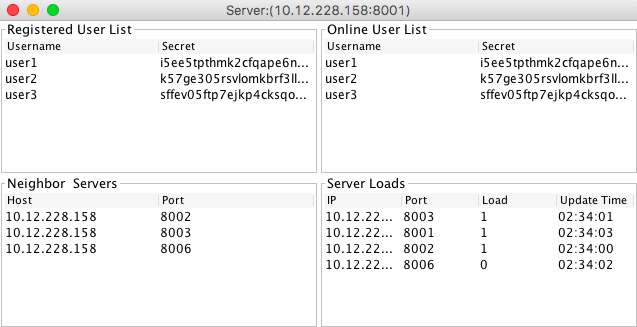
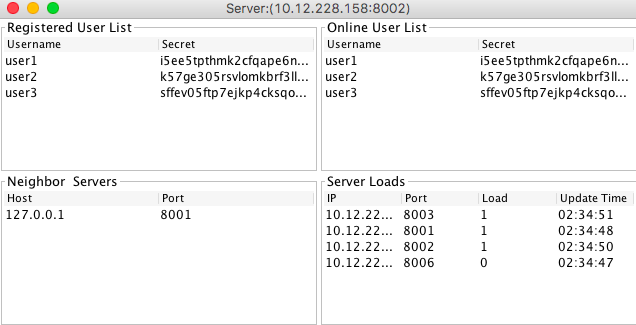
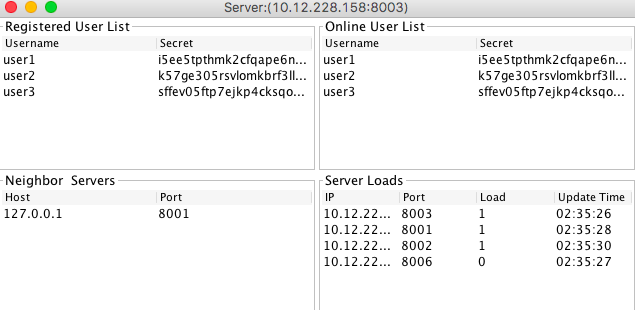
* + Terminate 8001 and send message from user2 within 10s. (close 8001 GUI, send message {“a”:1} from user2 in GUI, then wait for reconnection happens in 10s)
  + Reconnect user1 to any working server, for example 8006

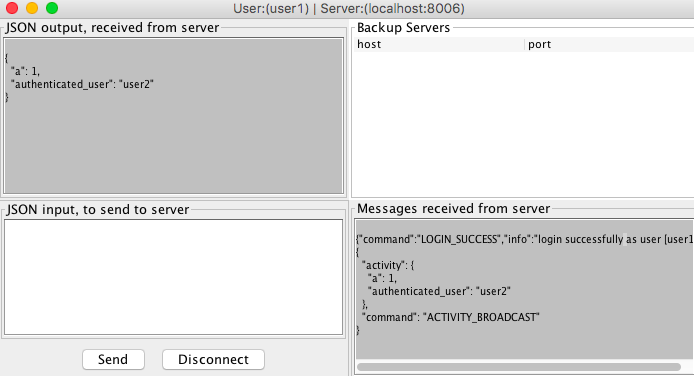
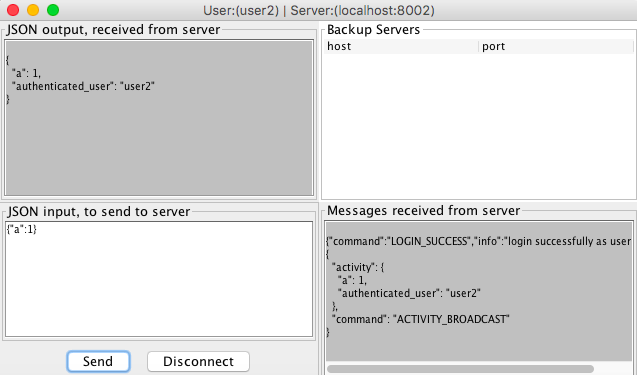


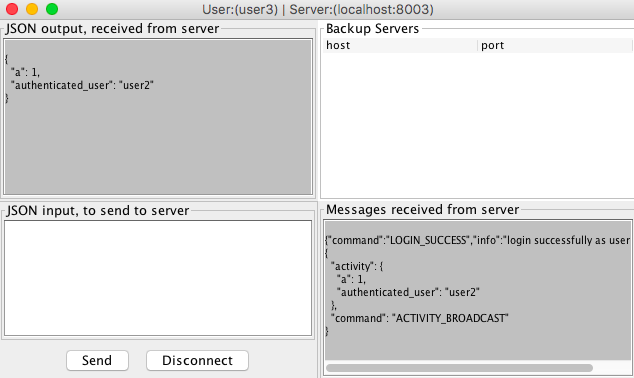
* Expected Result

User3 will receive the activity of user2 after reconnection is done (about 10s after disconnection), user1 will receive the activity of user2 after relogin on server 8006.

* Actual Result

Snapshot of 3 clients login on 8001, 8002, 8003 respectively

Snapshot of clients after reconnection(user1, user2, user3 all received activity from user2)



* 1. **Message Order**

In order to simulate message disorder case, use a telnet session to simulate a server and make the order checking period a little longer with activity\_check\_interval=10000. Fake message will be broadcast by telnet server with a hooker “timeBack” , which can set messages to be sent per configuration of “timeBack”.

* Operations
  + Start 1 server with activity\_check\_interval=10000 (10s)



* + Start a normal client connecting to 8001



* + Start a terminal and using telnet to simulate a server that can send activity



* + Authenticate telnet server on 8001



* + Send two fake activities in telnet session to simulate disorder message

Message\_num 2 is sent 0 second ago, message\_num 1 is sent 10s ago





* Expected Result

After waiting 10-20s, user1 will receive 2 activities in order, message\_num 1 first and message\_num 2 later.

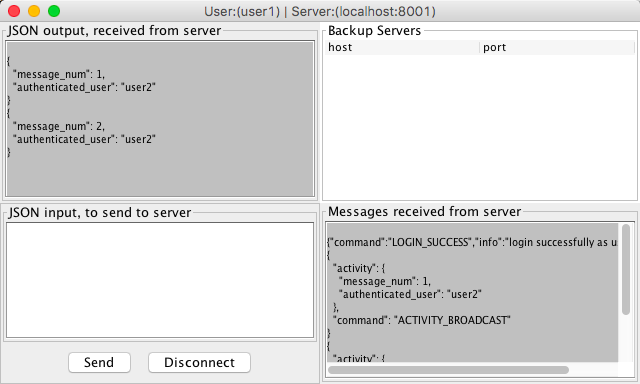
* Actual Result

Snapshot of “ACTIVITY\_SYNC” in telnet session

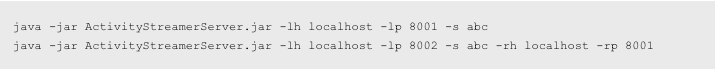


Message\_num 1 sendTime is 1527153486491, message\_num 2 sendTime is 1527153489382, program orders message per sendTime, thus message\_num 1 should be inserted in front of message\_num 2 in user1 GUI.

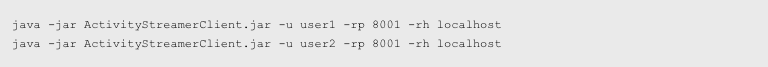
Snapshot of user1 (message\_num1 is before message\_num 2)



* 1. **Load Balancing**
* Operations
  + Start 2 servers



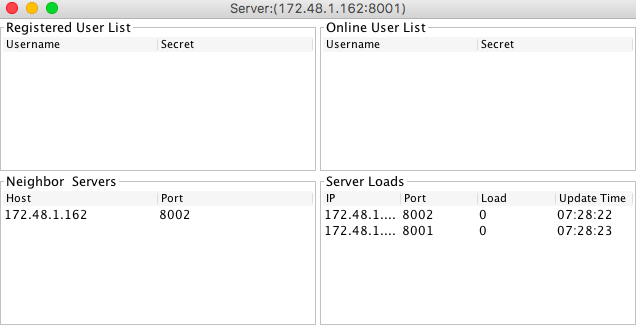
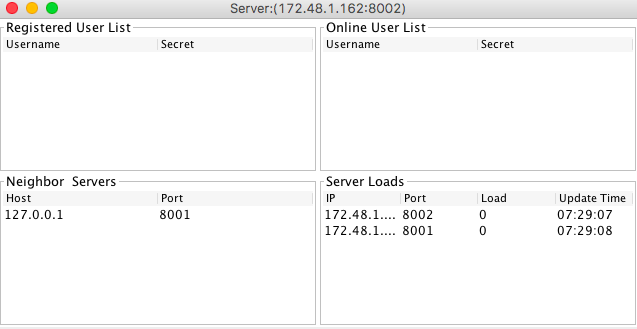
* + Register and login 2 clients on 8001

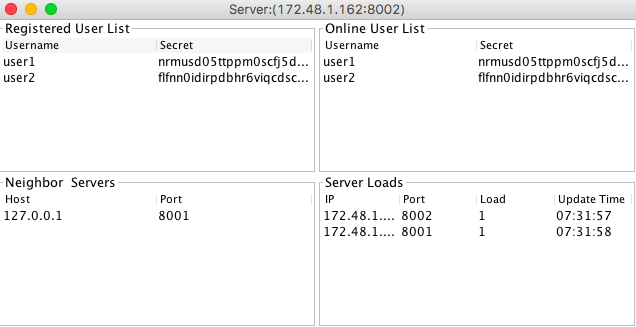


* Expect Result

User2 will be redirected to 8002 per load balancing mechanism.

* Actual Result

Snapshot of starting 2 servers

Snapshot after two clients login (load of each server has been changed to 1)

