THE UNIVERSITY OF MELBOURNE

COMP90015

DISTRIBUTED SYSTEMS

Project 2

Semester 1, 2018

Group 666

Shujing Xiao (Login name: shujingx Email: shujingx@student.unimelb.edu.au)

Ziyi Xiong (Login name: zxiong1 Email: zxiong1@student.unimelb.edu.au)

Ziqi Zhang (Login name: zzhang8 Email: zzhang8@student.unimelb.edu.au)

Zhengqing Zhu (Login name: zhengqingz Email: zhengqingz@student.unimelb.edu.au)

Introduction

5 methods are used to solve the Aims and the problems of failure model in this project. **Server load balance** method aims to solve the evenly distributed problem(Aim6) of this project, which is used after a new server joining the server system(Aim1). **Server Crash** method is used to solve both disconnection between servers(failure model3) problem and server may crash at any time problem(failure mode1). **T time message** method aims to solve the problem which guarantees that an activity message sent by a client reaches all clients that are connected to the network at the time that the message was sent(Aim4). **Message sequence** method aims to solve the problem where activity messages sent by a client are delivered in the same order at each receiving client(Aim5). **Availability-for clients** method aims to solve clients joining or leaving at any time problem(Aim2). When the clients crash at any time, the server can treat this situation as LOGOUT procedure(failure mode2).

Assumptions

Server Crash:

For one server, the backup_server_to_use and initial_server won't fail at same time.

No client redirect occur at first 5 seconds after a server crashed.

Message Sequence:

When one server just join in the server net, the first ACTIVITY_BROADCAST won't contain a out-of-order message.

Server Crash

Terminology and Attribution

Initial_server: the ancestry server for one server to connect via outgoingConnection(). Typically Indicated by the arguments of -rh and -rp. Every server except the root server have this field.

Backup_server to send: the server or its address that one server send to descendant. Typically the ancestor of this server. Every server have this field.

Backup_server_to_use: the server or its address that one server would redirect to after the initial server or connection crashed. Every server have this field.

Root_server: The server who doesn't have any outgoing connection. Typically the first server that started without -rh and -rp arguments. There is only one root_server in a server net.

Backup_root_server: The server that chosen by the root server as the backup server to send. There is only one backup_root_server in a server net.

Problem and Solution

Failure mode to deal with

Connections can crush at anytime,

Servers can crush at anytime.

Detailed method to handle it

First of all, our program can only catch the exception about connection lost and didn't know the reason caused this exception.

To deal with the failure mode, we divided the servers in three types, the root_server, the backup_root_server and the normal server.

For the first assumption, when one server catch the exception, try to connect to initial_server indicated by the arguments. This procedure is called re-connect.

The re-connect procedure is valid except the root_server because root_server doesn’t have any initial_server.

If the initial_server crushed which would cause exception when re-connecting, the server would try to connect to backup_server_to_use. This procedure is called re-direct. This is only valid for normal servers.

For the backup_root_server, if the exception from initial_server which is the root_server captured, the re-connect procedure was triggered. If the re-connect was failed, the backup_root_server would become the new root_server and stop to connect to others. Then it would choose the new backup_root_server and store it in the backup_server_to_send and inform other descendant to change their backup_server_to_use via the message of BACKUP_SERVER.

For the root_server, it can catch the exception caused by a descendant crash. When this occur, the root_server would choose a new backup_root_server and inform descendants to change their backup_server_to_use with message of BACKUP_SERVER.

Message and Protocol

Newly added message:

AUTHENTICATION_SUCCESS

```
{
        “command” : “AUTHENTICATION_SUCCESS”,
        “backupname” : “localhost”,
        “backupport” : “3780”
}
```

Why this message: To indicate receiver the authentication state and the backup_server_to_use to redirect to after crash.

When to send: Reply to the AUTHENTICATE message. The address if from the backup_server_to_send of the sender.

How to react: Server would store the backup address from the message as the backup_server_to_use. If the backup address is the the server's current address, which means the server was chosen as the backup_root_server, it would reset the backup_server_to_use to null

BACKUP_SERVER

```
{
    "command" : "BACKUP_SERVER",
    "backupname" : "localhost",
    "backupport" : "3780"
}
```

Why this message: Only the root_server could send this message. To indicate the new backup_root_server and inform other descendants to update their backup_server_to_use.

When to send: It would be sent when one of the descendant of root_server was crushed. Or the backup_root_server became root_server when previous root_server crushed.

How to react: Server would check if the address in the message is the address of itself. If so, this server was chosen to be the backup_root_serever and it would reset its bcakup_server_to_use to null.

Modified message:

AUTHENTICATE

```
{
    "command" : "AUTHENTICATE",
    "secret" : "group666",
    "hostname" : "localhost",    //newly modified
    "port" : "3780"    //newly modified
}
```

Why modify: add the server address to help generate re-direct info.

Effect and Conclusion

After we apply the re-connect and re-direct mechanism, the high availability was guaranteed. And the procedure of the root_server and backup_root_server guaranteed the server net won't divide into several nets after the node crashed.

Message Sequence

Terminology and Attribution

Message_buffer: A class to store all the messages from one client. Every server have several of this and each of this corresponding to one client.

Next_order_to_get: This pointer always points to next received in-order message. So when out-of-order message received, this pointer stay still.

Next_order_to_send: This pointer always points to the next message that should be sent. And this pointer will never exceed the next_order_to_get.

Flush: Every time flush the buffer would get a in-order message and the next_order_to_send would increase.

Problem and Solution

The message might be out of order due to any reason. Our job is to keep message transport in order.

The solution is to add a sequence order in ACTIVITY_BROADCAST and use message_buffer to keep out-of-order message which means the order in message is larger than next_order_to_get. Check the order when every message have been received. Then flush the buffer when no more in-order message in message_buffer which means the next_order_to_send equals next_order_to_get.

The message_buffer would be cleaned when a server received a ANNOUNCE_LOGIN. That is because receiving this message means this client is logged in and was accepted by all the servers. So we could reset the message_buffer

Message and Protocol

Modified message:

ACTIVITY_BROADCAST

```
{
    "command" : "ACTIVITY_BROADCAST",
    "activity" : activity,
    "time" : 144739821749,
    "client" : "localhost:3781",    //newly modified
    "order" : "3"    //newly modified
}
```

Why modify: the "client" field is to distinguish source of the message. The "order" is to indicate the current order of this message.

Effect and Conclusion

Every server would check every messages' order, so the out-of-order problem would be corrected once being detected. As for the concurrency, the initial order is from the first received ACTIVITY_BROADCAST so that ensures that every server have same react to out-of-order message.

T Time Message problem

Problem and Solution

Achieve the aim:

1. Guarantees that an activity message sent by a client reaches all clients that are connected to the network at the time that the message was sent.

2. On the other hand, to make sure client can still receive the message which start transmission before client logout or crash but it has not arrived to the client.

Ways of solving the problem:

1. Using timestamp to compare the establish time.
2. To solve this problem, we decide to use message history which is created by each server itself to add each received broadcast message and a logout list to record the user and its logout time in itself.

Message and Protocol

Newly added message:

```
Connection(Socket socket) throws IOException {
        establishTime = (new Date()).getTime();
    }
```

LAST_LOGOUT

```
{
      “command” : “LAST_LOGOUT”,
      “username” : “abc”,
      “secret” : “123abc”,
      "Logouttime" : “1231231”
}
```

Modified message:

ACTIVITY_BROADCAST

```
{
   “command” : “ACTIVITY_BROADCAST”,
```

“activity” : activity,

“time” : 144739821749,   //newly modified

“client” : “localhost:3781”,

“order” : “3”

}

When to react: When the new connection is set up first time, this new attribute is set.

How to use: This attribute is used to record the time when the first time connection is established. The activity message can send to all right clients according to the establishTime.

If clients’ establishTime is smaller than servers’ establishTime, the activity should not send to this client. If clients’ establishTime is greater than servers’ establishTime, the activity can send to this client.

Secondly, each broadcast and logout activity(include client crash) of each server has records with time. Therefore, we assume our system with client has latency up to 5000 ms, so that client can receive the messages which are 5 seconds after its last logout time when client login again and the server which client is logining will delete the client’s last logout record(no matter which server, because it will check last logout record in whole system by sending the Last Logout command message which show above to notice each server).

Server Load Balance Problem

Problem and Solution

To balance the load of each server while system is running. The case of the new server incoming or client login/logout activity will cause the load balance problem.

Message and Protocol

To solve this problem with consideration of high availability for client and eventual consistency. Firstly, we need to keep smooth experience for user when the balance load happen. Secondly, we need a trigger mechanism to active client redirection.

Therefore, we can make our redirect mechanism trigger automatically for 5 seconds interval each time with probability theory to achieve smoothly redirect which can avoid multiple client redirect to same lowest server at the same time.
On the other hand, we will have a probability to trigger checking the load from each server's announce for each 5 seconds. If yes, it will make low servers as two candidates of server which are a set of low load server and a lowest server. Furthermore, it will randomly choose a candidate to as the redirect server and if the low server set is chosen, it will randomly choose one server in the set.

Finally, the core of this redirect mechanism is probability applying to solve high availability and eventual consistency problem.

Availability-for-clients Problem

Message and Protocol

Newly added message:

LOGIN_LOCK

```
{
        "command" : "LOGIN_LOCK",
        "username" : "abc",
        "secret" : "123abc"
}
```

LOGIN_ALLOWED

```
{
        "command" : "LOGIN_ALLOWED",
        “username” : “abc”,
        “secret” : “123abc”
}
```

LOGIN_DENIED

```
{
        “command” : “LOGIN_DENIED”,
        “username” : “abc”
}
```

ANNOUNCE_LOGIN

```
{
        “command” : “ANNOUNCE_LOGIN”,
        “username” : “abc”,
        “secret” : “123abc”
}
```

Problem and Solution

In order to let clients to have high availability of service, several issues need to be considered.

Issue 1: A given username can only be registered once over the network.

Issue 2: A client can login at anytime to any server (even it is a newly added server).

Issue 3: Only one client of one username-secret pair can be logged in at the same time.

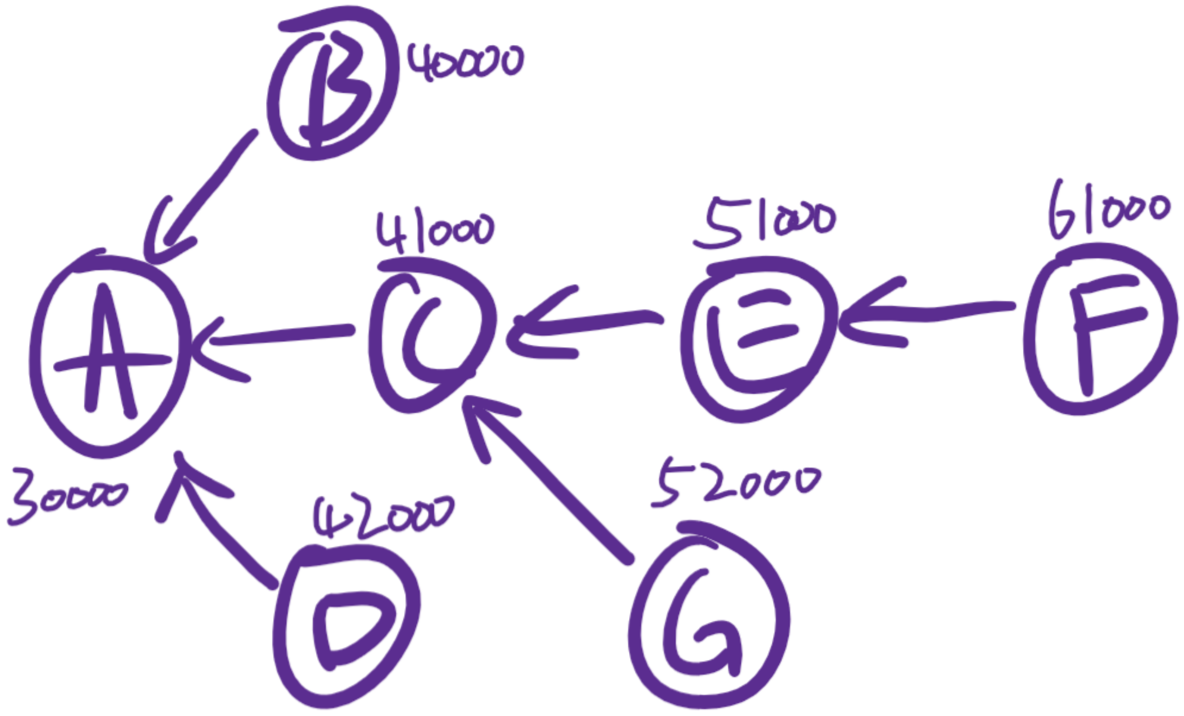
For Issue 1: This is achieved by previous protocol “LOCK_REQUEST”, “LOCK_DENIED” and “LOCK_ALLOWED”.

For Issue 2: The main problem is that, a client need to be able to login to a newly added server. It is the “Availability” and “Consistency” problem of the “CAP”: the whole system must ensure that its service is available for clients at anytime and the register-information of clients must be consistent within servers so that this registered client can login to the new server. In our project, we use a strategy that when a client using a username and a secret to login to a server, this server first check its local register-information. If it has matched username-secret pair, it will let this client to login; if it does not has matched information, this server will broadcast to other servers a message "LOGIN_LOCK" which contains the username and secret of that client who is trying to login and this server will add this client to a container "loggingClients". Other servers who received this message will check their register-information on their local memory. It they have the matched information, they will broadcast a "LOGIN_ALLOWED", which contains the matched username and server, to all other servers; if they do not have the matched information, they will broadcast a "LOGIN_DENIED" to all other servers. Servers who receive a "LOGIN_ALLOWED" will check its local memory, and if they don't contains matched information, they will save this username-secret pair in their register-information list (It is an important mechanism for achieving "Eventually Consistency"). Server who receives a "LOGIN_DENIED" will check the "loggingClients" list. If it has matched username-secret pair (which means this server is the server to whom the client login), this server will increment the count number and when the count number matches the number of servers in the network(which means all other server do not have the username-secret pair registered, in other words, this client has not registered yet), it will deny the login request of the client.

Issue 3: We use a strategy that, whenever a client successfully login on a server, this server will broadcast a message to all other servers to indicate that this client (with username XXX and secret XXX) has just logged in on me. And other servers who receive this message will check the login list on their own. If they have matched information, they will kick out them to ensure that only one client of a username-secret pair is allowed logged in at the same time (and this client must be the latest one).

Server Crash Example

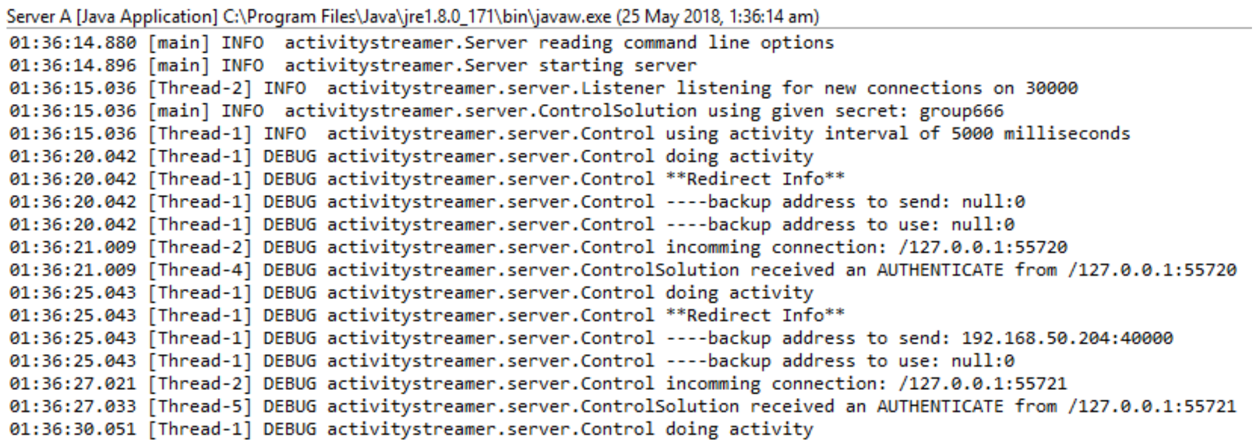
Scenario:



In this case, Server A is the root_server and the startup order is A-B-C-D-E-F-G, The G will connect to C after several servers crashed. So the server B is the backup_root_server.

Startup procedure:

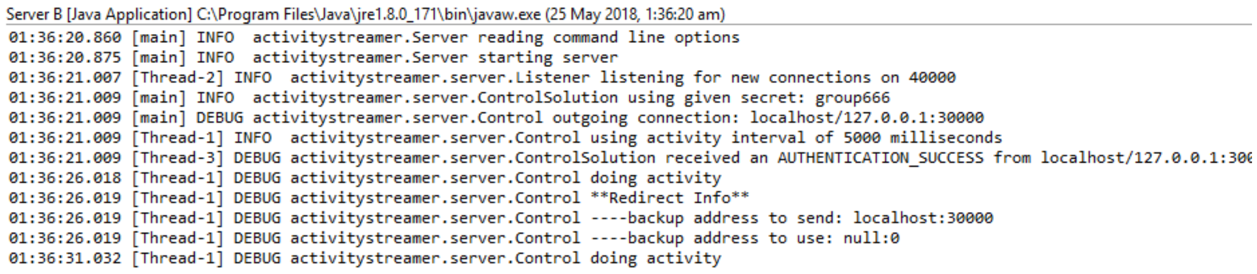
Console of server A:



```
Server A [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (25 May 2018, 1:36:14 am)
01:36:14.880 [main] INFO  activitystreamer.Server reading command line options
01:36:14.896 [main] INFO  activitystreamer.Server starting server
01:36:15.036 [Thread-2] INFO  activitystreamer.server.Listener listening for new connections on 30000
01:36:15.036 [main] INFO  activitystreamer.server.ControlSolution using given secret: group666
01:36:15.036 [Thread-1] INFO  activitystreamer.server.Control using activity interval of 5000 milliseconds
01:36:20.042 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:36:20.042 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:36:20.042 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: null:0
01:36:20.042 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
01:36:21.009 [Thread-2] DEBUG activitystreamer.server.Control incomming connection: /127.0.0.1:55720
01:36:21.009 [Thread-4] DEBUG activitystreamer.server.ControlSolution received an AUTHENTICATE from /127.0.0.1:55720
01:36:25.043 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:36:25.043 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:36:25.043 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: 192.168.50.204:40000
01:36:25.043 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
01:36:27.021 [Thread-2] DEBUG activitystreamer.server.Control incomming connection: /127.0.0.1:55721
01:36:27.033 [Thread-5] DEBUG activitystreamer.server.ControlSolution received an AUTHENTICATE from /127.0.0.1:55721
01:36:30.051 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

From the console of server A, we could see the backup address to send and backup address to use are both null until the first connection from server B. Then, B became the backup_server_to_send of A.

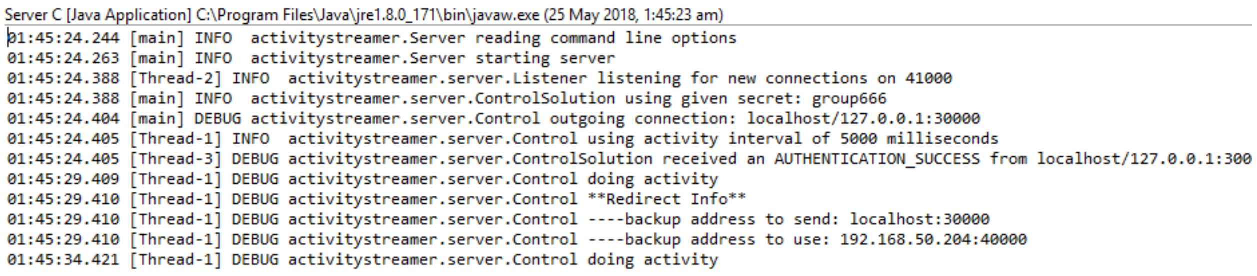
Console of B:



```
Server B [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (25 May 2018, 1:36:20 am)
01:36:20.860 [main] INFO  activitystreamer.Server reading command line options
01:36:20.875 [main] INFO  activitystreamer.Server starting server
01:36:21.007 [Thread-2] INFO  activitystreamer.server.Listener listening for new connections on 40000
01:36:21.009 [main] INFO  activitystreamer.server.ControlSolution using given secret: group666
01:36:21.009 [main] DEBUG activitystreamer.server.Control outgoing connection: localhost/127.0.0.1:30000
01:36:21.009 [Thread-1] INFO  activitystreamer.server.Control using activity interval of 5000 milliseconds
01:36:21.009 [Thread-3] DEBUG activitystreamer.server.ControlSolution received an AUTHENTICATION_SUCCESS from localhost/127.0.0.1:300
01:36:26.018 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:36:26.019 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:36:26.019 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:30000
01:36:26.019 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
01:36:31.032 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

From the AUTHENTICATION_SUCCESS message, B knows that it has been chosen as backup_root_server for the address in message is the address of B.

Console of C:



```
Server C [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (25 May 2018, 1:45:23 am)
01:45:24.244 [main] INFO  activitystreamer.Server reading command line options
01:45:24.263 [main] INFO  activitystreamer.Server starting server
01:45:24.388 [Thread-2] INFO  activitystreamer.server.Listener listening for new connections on 41000
01:45:24.388 [main] INFO  activitystreamer.server.ControlSolution using given secret: group666
01:45:24.404 [main] DEBUG activitystreamer.server.Control outgoing connection: localhost/127.0.0.1:30000
01:45:24.405 [Thread-1] INFO  activitystreamer.server.Control using activity interval of 5000 milliseconds
01:45:24.405 [Thread-3] DEBUG activitystreamer.server.ControlSolution received an AUTHENTICATION_SUCCESS from localhost/127.0.0.1:300
01:45:29.409 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:45:29.410 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:45:29.410 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:30000
01:45:29.410 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: 192.168.50.204:40000
01:45:34.421 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

Different from B, C got the backup_server_to_use from the AUTHENTICATION_SUCCESS. So C is so called normal server. C's backup_server_to_send is A and backup_server_to_use is B. Also we could find out that a server's backup_server_to_use is its backup_server_to_send's backup_server_to_send. Typically, backup_server_to_send is the ancestor and the backup_server_to_use is the ancestor's ancestor.

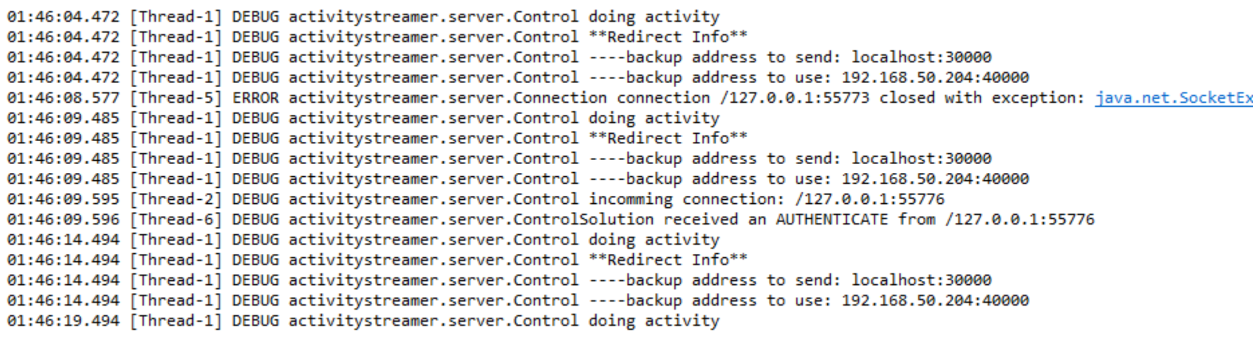
Same as D,E,F,G.

Crash test

1. E crashed (normal server crashed)

C will do nothing since C isn't a root_server.

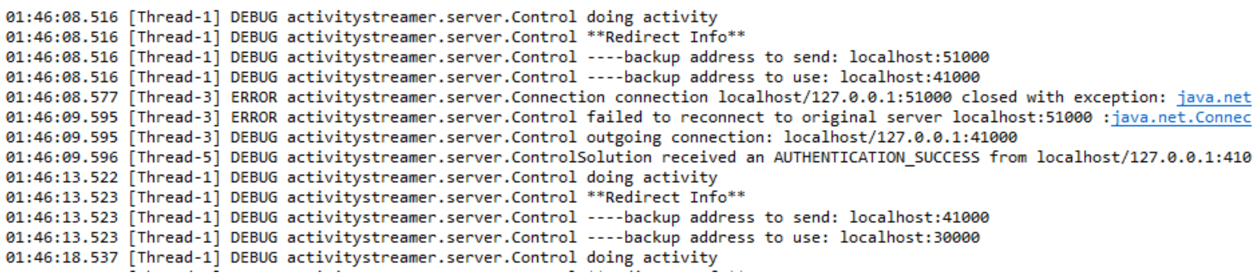
Console of C:



```
01:46:04.472 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:46:04.472 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:46:04.472 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:30000
01:46:04.472 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: 192.168.50.204:40000
01:46:08.577 [Thread-5] ERROR activitystreamer.server.Connection connection /127.0.0.1:55773 closed with exception: java.net.SocketEx
01:46:09.485 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:46:09.485 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:46:09.485 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:30000
01:46:09.485 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: 192.168.50.204:40000
01:46:09.595 [Thread-2] DEBUG activitystreamer.server.Control incomming connection: /127.0.0.1:55776
01:46:09.596 [Thread-6] DEBUG activitystreamer.server.ControlSolution received an AUTHENTICATE from /127.0.0.1:55776
01:46:14.494 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:46:14.494 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:46:14.494 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:30000
01:46:14.494 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: 192.168.50.204:40000
01:46:19.494 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

Since E is a normal server, F will try to re-connect and re-direct. From the console, we could see that the re-connect was failed and re-direct was successful. The corresponding backup_server_to_use was updated via receiving AUTHENTICATION_SUCCESS and backup_server_to_use became backup_server_to_send.

Console of F:

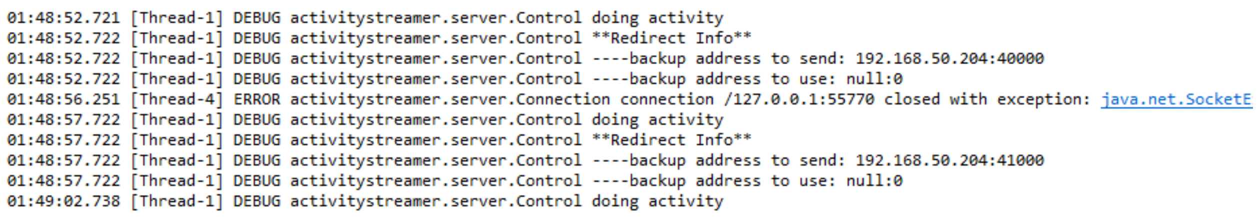


```
01:46:08.516 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:46:08.516 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:46:08.516 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:51000
01:46:08.516 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: localhost:41000
01:46:08.577 [Thread-3] ERROR activitystreamer.server.Connection connection localhost/127.0.0.1:51000 closed with exception: java.net
01:46:09.595 [Thread-3] ERROR activitystreamer.server.Control failed to reconnect to original server localhost:51000 :java.net.Connec
01:46:09.595 [Thread-3] DEBUG activitystreamer.server.Control outgoing connection: localhost/127.0.0.1:41000
01:46:09.596 [Thread-5] DEBUG activitystreamer.server.ControlSolution received an AUTHENTICATION_SUCCESS from localhost/127.0.0.1:410
01:46:13.522 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:46:13.523 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:46:13.523 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:41000
01:46:13.523 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: localhost:30000
01:46:18.537 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

2. B crash (backup_root_server crashed)

Since B is the backup_root_server, the crash of B would cause A to change backup_root_server to avoid partition.

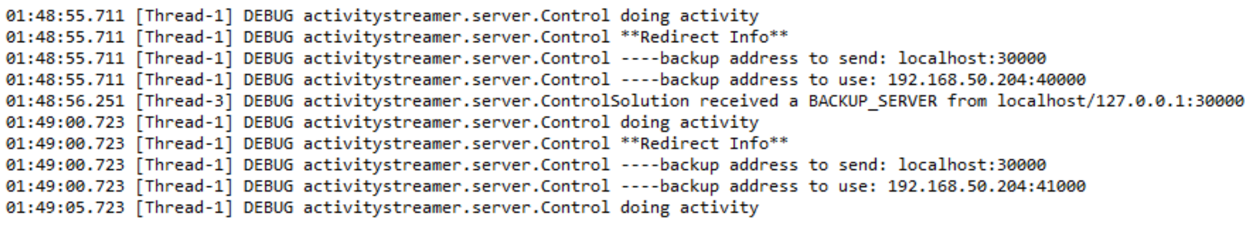
Console of A:



```
01:48:52.721 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:48:52.722 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:48:52.722 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: 192.168.50.204:40000
01:48:52.722 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
01:48:56.251 [Thread-4] ERROR activitystreamer.server.Connection connection /127.0.0.1:55770 closed with exception: java.net.SocketE
01:48:57.722 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:48:57.722 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:48:57.722 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: 192.168.50.204:41000
01:48:57.722 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
01:49:02.738 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

After A changed the backup_root_server, A would issue a BACKUP_SERVER to inform other descendants with new backup_root_server. D was not the new backup_root_server, so only the backup_server_to_use was changed.

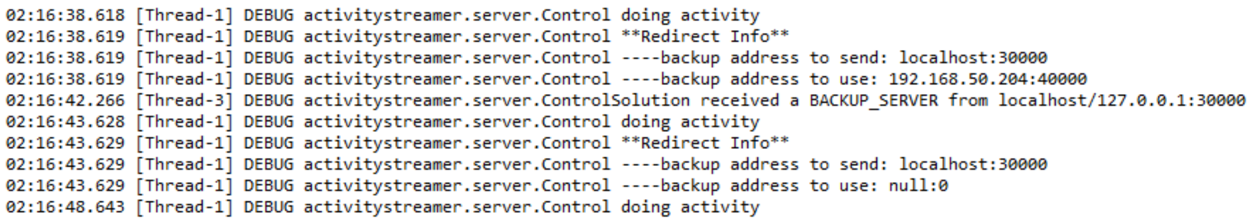
Console of D:



```
01:48:55.711 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:48:55.711 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:48:55.711 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:30000
01:48:55.711 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: 192.168.50.204:40000
01:48:56.251 [Thread-3] DEBUG activitystreamer.server.ControlSolution received a BACKUP_SERVER from localhost/127.0.0.1:30000
01:49:00.723 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:49:00.723 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:49:00.723 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:30000
01:49:00.723 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: 192.168.50.204:41000
01:49:05.723 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

C has been chosen as backup_root_server. So C reset its backup_server_to_use.

Console of C:

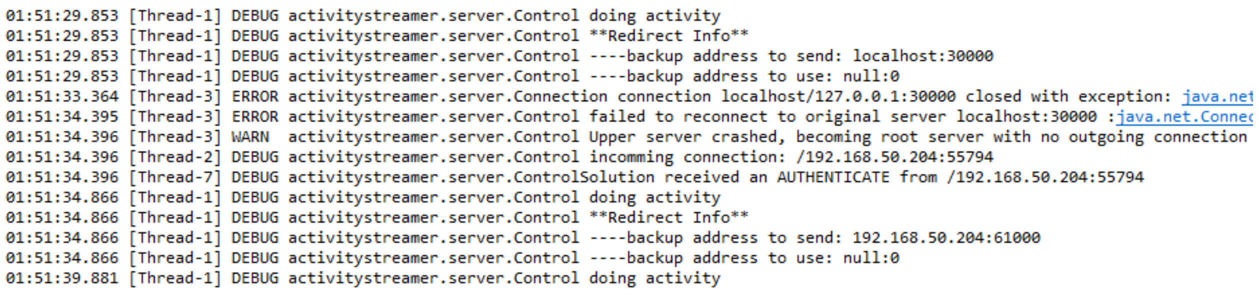


```
02:16:38.618 [Thread-1] DEBUG activitystreamer.server.Control doing activity
02:16:38.619 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
02:16:38.619 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:30000
02:16:38.619 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: 192.168.50.204:40000
02:16:42.266 [Thread-3] DEBUG activitystreamer.server.ControlSolution received a BACKUP_SERVER from localhost/127.0.0.1:30000
02:16:43.628 [Thread-1] DEBUG activitystreamer.server.Control doing activity
02:16:43.629 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
02:16:43.629 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:30000
02:16:43.629 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
02:16:48.643 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

3. A crash (root_server crashed)

C became the new root_server and chose F as new backup_root_server.

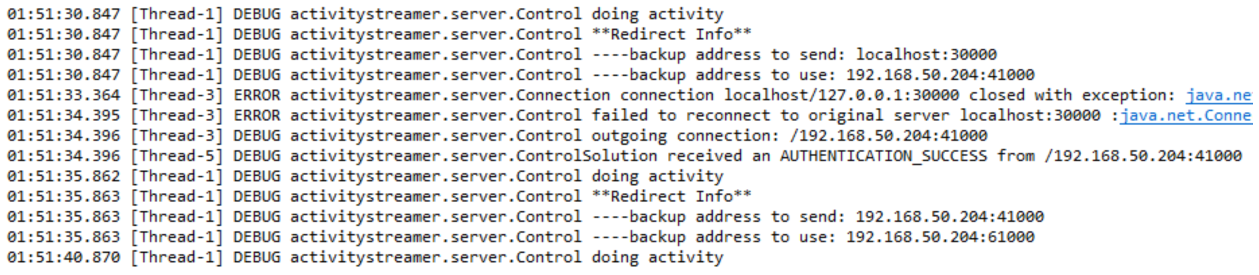
Console of C



```
01:51:29.853 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:51:29.853 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:51:29.853 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:30000
01:51:29.853 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
01:51:33.364 [Thread-3] ERROR activitystreamer.server.Connection connection localhost/127.0.0.1:30000 closed with exception: java.net
01:51:34.395 [Thread-3] ERROR activitystreamer.server.Control failed to reconnect to original server localhost:30000 :java.net.Conne
01:51:34.396 [Thread-3] WARN  activitystreamer.server.Control Upper server crashed, becoming root server with no outgoing connection
01:51:34.396 [Thread-2] DEBUG activitystreamer.server.Control incomming connection: /192.168.50.204:55794
01:51:34.396 [Thread-7] DEBUG activitystreamer.server.ControlSolution received an AUTHENTICATE from /192.168.50.204:55794
01:51:34.866 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:51:34.866 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:51:34.866 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: 192.168.50.204:61000
01:51:34.866 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
01:51:39.881 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

D is a normal server connected to A, so it would re-direct to C and update the backup_server information via AUTHENTICATION_SUCCESS.

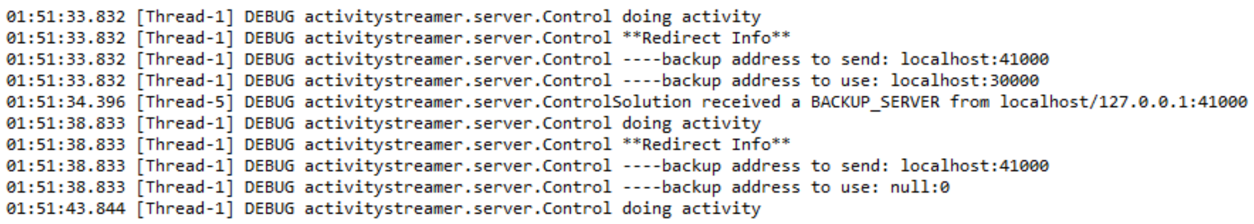
Console of D:



```
01:51:30.847 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:51:30.847 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:51:30.847 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:30000
01:51:30.847 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: 192.168.50.204:41000
01:51:33.364 [Thread-3] ERROR activitystreamer.server.Connection connection localhost/127.0.0.1:30000 closed with exception: java.ne
01:51:34.395 [Thread-3] ERROR activitystreamer.server.Control failed to reconnect to original server localhost:30000 :java.net.Conne
01:51:34.396 [Thread-3] DEBUG activitystreamer.server.Control outgoing connection: /192.168.50.204:41000
01:51:34.396 [Thread-5] DEBUG activitystreamer.server.ControlSolution received an AUTHENTICATION_SUCCESS from /192.168.50.204:41000
01:51:35.862 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:51:35.863 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:51:35.863 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: 192.168.50.204:41000
01:51:35.863 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: 192.168.50.204:61000
01:51:40.870 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

F has been chosen as backup_root_server. So F reset its backup_server_to_use.

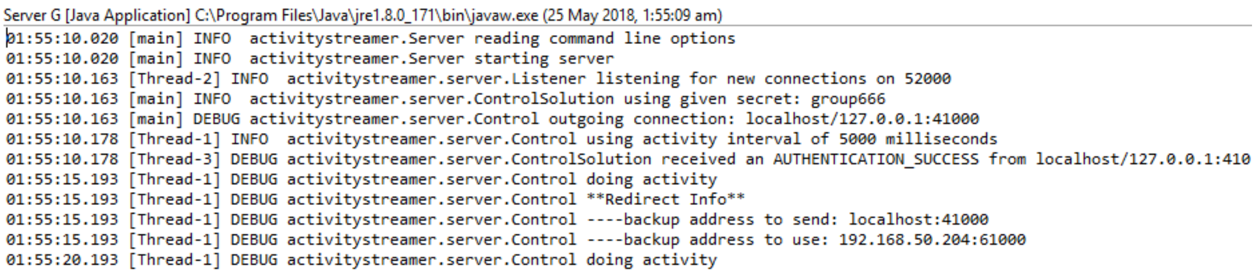
Console of F:


```
01:51:33.832 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:51:33.832 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:51:33.832 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:41000
01:51:33.832 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: localhost:30000
01:51:34.396 [Thread-5] DEBUG activitystreamer.server.ControlSolution received a BACKUP_SERVER from localhost/127.0.0.1:41000
01:51:38.833 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:51:38.833 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:51:38.833 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:41000
01:51:38.833 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
01:51:43.844 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

4. Incoming of G

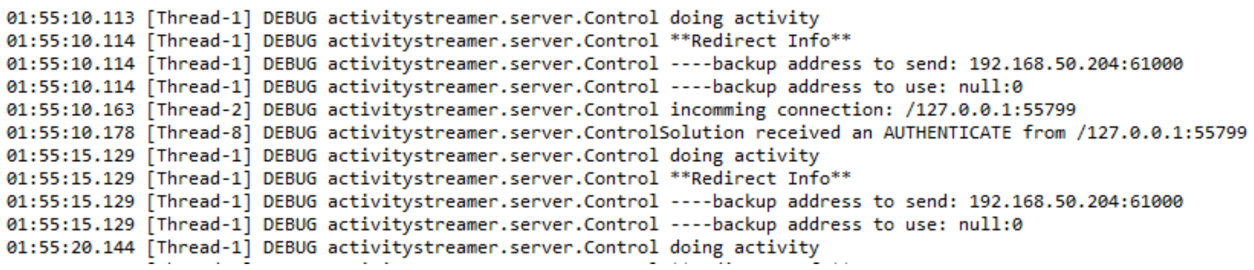
In this case, G is a normal server and it would update the backup_server information via AUTHENTICATION_SUCCESS.

Console of G:


```
Server G [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (25 May 2018, 1:55:09 am)
01:55:10.020 [main] INFO  activitystreamer.Server reading command line options
01:55:10.020 [main] INFO  activitystreamer.Server starting server
01:55:10.163 [Thread-2] INFO  activitystreamer.server.Listener listening for new connections on 52000
01:55:10.163 [main] INFO  activitystreamer.server.ControlSolution using given secret: group666
01:55:10.163 [main] DEBUG activitystreamer.server.Control outgoing connection: localhost/127.0.0.1:41000
01:55:10.178 [Thread-1] INFO  activitystreamer.server.Control using activity interval of 5000 milliseconds
01:55:10.178 [Thread-3] DEBUG activitystreamer.server.ControlSolution received an AUTHENTICATION_SUCCESS from localhost/127.0.0.1:410
01:55:15.193 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:55:15.193 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:55:15.193 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:41000
01:55:15.193 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: 192.168.50.204:61000
01:55:20.193 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

C have backup_root_server. So G is only a normal server and C would not change anything.

Console of C:


```
01:55:10.113 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:55:10.114 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:55:10.114 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: 192.168.50.204:61000
01:55:10.114 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
01:55:10.163 [Thread-2] DEBUG activitystreamer.server.Control incomming connection: /127.0.0.1:55799
01:55:10.178 [Thread-8] DEBUG activitystreamer.server.ControlSolution received an AUTHENTICATE from /127.0.0.1:55799
01:55:15.129 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:55:15.129 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:55:15.129 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: 192.168.50.204:61000
01:55:15.129 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
01:55:20.144 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

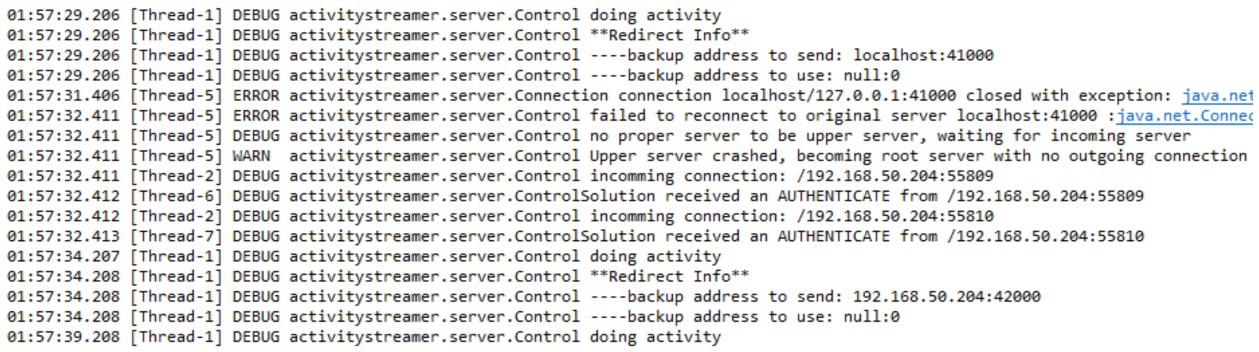
5. C crash

C is the root_server chosen by the program. At the beginning, it is a normal server, with the crash of B, it became backup_root_server. Then A crashed, C became root_server.

From the consoles below, we could see the net going well when c crashed. Which means any server could be root_server or backup_root_server and this won’t affect the system.

F became root_server since the previous root_server crashed.

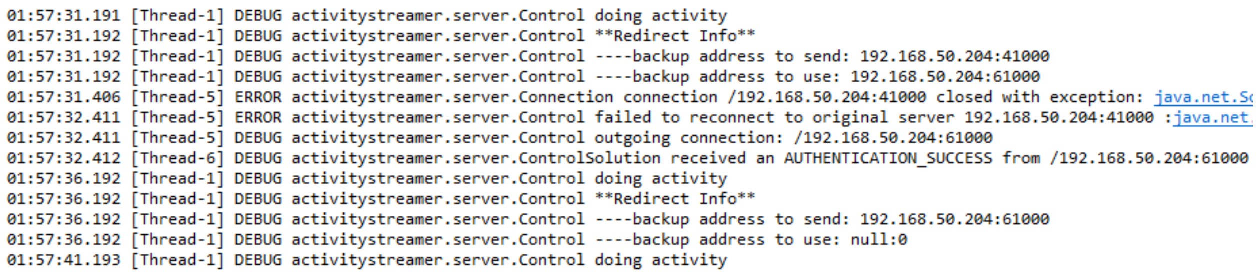
Console of F:



```
01:57:29.206 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:57:29.206 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:57:29.206 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:41000
01:57:29.206 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
01:57:31.406 [Thread-5] ERROR activitystreamer.server.Connection connection localhost/127.0.0.1:41000 closed with exception: java.net
01:57:32.411 [Thread-5] ERROR activitystreamer.server.Control failed to reconnect to original server localhost:41000 :java.net.Connec
01:57:32.411 [Thread-5] DEBUG activitystreamer.server.Control no proper server to be upper server, waiting for incoming server
01:57:32.411 [Thread-5] WARN  activitystreamer.server.Control Upper server crashed, becoming root server with no outgoing connection
01:57:32.411 [Thread-2] DEBUG activitystreamer.server.Control incomming connection: /192.168.50.204:55809
01:57:32.412 [Thread-6] DEBUG activitystreamer.server.ControlSolution received an AUTHENTICATE from /192.168.50.204:55809
01:57:32.412 [Thread-2] DEBUG activitystreamer.server.Control incomming connection: /192.168.50.204:55810
01:57:32.413 [Thread-7] DEBUG activitystreamer.server.ControlSolution received an AUTHENTICATE from /192.168.50.204:55810
01:57:34.207 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:57:34.208 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:57:34.208 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: 192.168.50.204:42000
01:57:34.208 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
01:57:39.208 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

D became backup_root_server

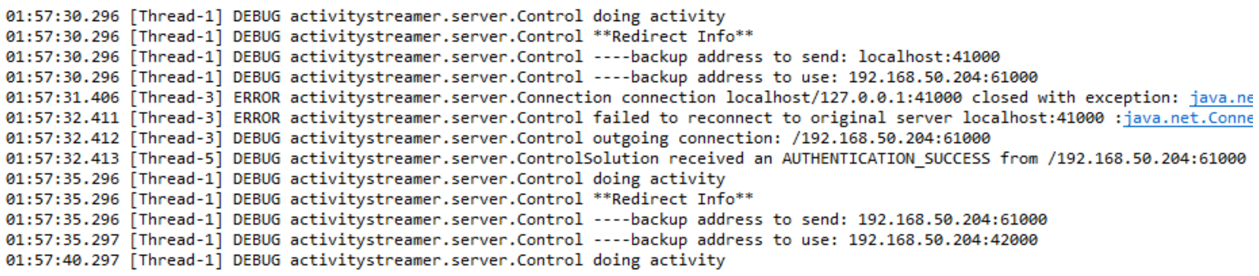
Console of D:



```
01:57:31.191 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:57:31.192 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:57:31.192 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: 192.168.50.204:41000
01:57:31.192 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: 192.168.50.204:61000
01:57:31.406 [Thread-5] ERROR activitystreamer.server.Connection connection /192.168.50.204:41000 closed with exception: java.net.Sc
01:57:32.411 [Thread-5] ERROR activitystreamer.server.Control failed to reconnect to original server 192.168.50.204:41000 :java.net.
01:57:32.411 [Thread-5] DEBUG activitystreamer.server.Control outgoing connection: /192.168.50.204:61000
01:57:32.412 [Thread-6] DEBUG activitystreamer.server.ControlSolution received an AUTHENTICATION_SUCCESS from /192.168.50.204:61000
01:57:36.192 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:57:36.192 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:57:36.192 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: 192.168.50.204:61000
01:57:36.192 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
01:57:41.193 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

The normal server G re-direct to F and stay normal.

Console of G:

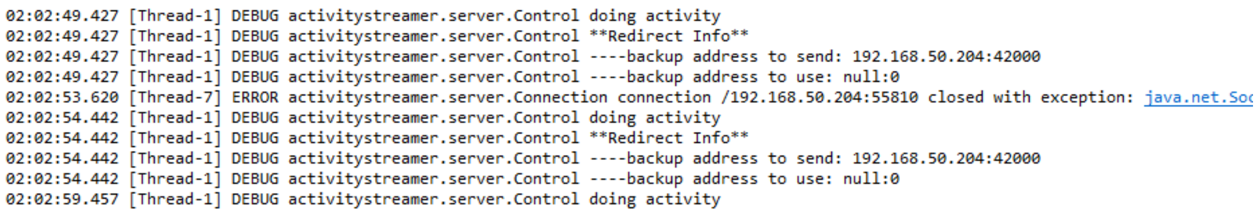


```
01:57:30.296 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:57:30.296 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:57:30.296 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: localhost:41000
01:57:30.296 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: 192.168.50.204:61000
01:57:31.406 [Thread-3] ERROR activitystreamer.server.Connection connection localhost/127.0.0.1:41000 closed with exception: java.ne
01:57:32.411 [Thread-3] ERROR activitystreamer.server.Control failed to reconnect to original server localhost:41000 :java.net.Conne
01:57:32.412 [Thread-3] DEBUG activitystreamer.server.Control outgoing connection: /192.168.50.204:61000
01:57:32.413 [Thread-5] DEBUG activitystreamer.server.ControlSolution received an AUTHENTICATION_SUCCESS from /192.168.50.204:61000
01:57:35.296 [Thread-1] DEBUG activitystreamer.server.Control doing activity
01:57:35.296 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
01:57:35.296 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: 192.168.50.204:61000
01:57:35.297 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: 192.168.50.204:42000
01:57:40.297 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

6. G Crash

G is a normal server with no descendant. So the crash of G won't affect anything.

Console of F:

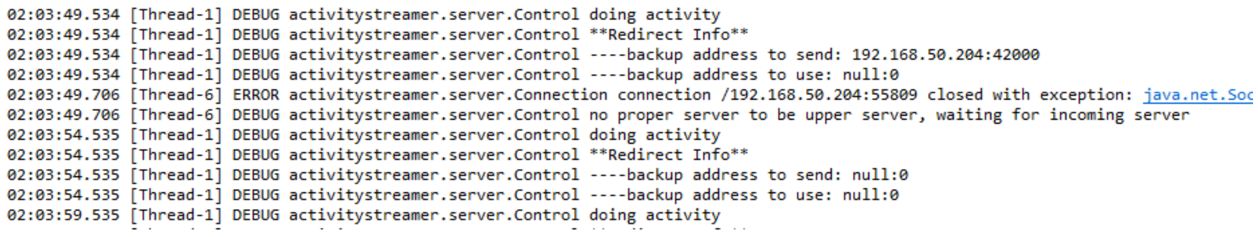


```
02:02:49.427 [Thread-1] DEBUG activitystreamer.server.Control doing activity
02:02:49.427 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
02:02:49.427 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: 192.168.50.204:42000
02:02:49.427 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
02:02:53.620 [Thread-7] ERROR activitystreamer.server.Connection connection /192.168.50.204:55810 closed with exception: java.net.Soc
02:02:54.442 [Thread-1] DEBUG activitystreamer.server.Control doing activity
02:02:54.442 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
02:02:54.442 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: 192.168.50.204:42000
02:02:54.442 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
02:02:59.457 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

7. D Crash

After D crashed, there was only one server alive which is F. As we can see, The state of F look like the very start of this test which is the startup of A. Becoming root_server and waiting for the server to connect.

Console of F:



```
02:03:49.534 [Thread-1] DEBUG activitystreamer.server.Control doing activity
02:03:49.534 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
02:03:49.534 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: 192.168.50.204:42000
02:03:49.534 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
02:03:49.706 [Thread-6] ERROR activitystreamer.server.Connection connection /192.168.50.204:55809 closed with exception: java.net.Soc
02:03:49.706 [Thread-6] DEBUG activitystreamer.server.Control no proper server to be upper server, waiting for incoming server
02:03:54.535 [Thread-1] DEBUG activitystreamer.server.Control doing activity
02:03:54.535 [Thread-1] DEBUG activitystreamer.server.Control **Redirect Info**
02:03:54.535 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to send: null:0
02:03:54.535 [Thread-1] DEBUG activitystreamer.server.Control ----backup address to use: null:0
02:03:59.535 [Thread-1] DEBUG activitystreamer.server.Control doing activity
```

Conclusion

From the test above, we can see that all the state of a server was closed. In another word, server in one state would change to another state when something like incoming connection or connection lost happened. However, the kind of state that a server might stay was finite and able to transfer to other states, which indicate that system mechanism to deal with failures is a finite-state machine. So, under our assumption which is for one server the backup_server_to_use and backup_server_to_send won't fail in same time, we could say the availability was high enough to deal with the failure mode.

Meeting Minutes

Date: May4, 2018

Location: Eastern Resource Centre

Attendance: Shujing Xiao, Ziyi Xiong, Ziqi Zhang, Zhengqing Zhu

Duration: 2 hours

Agenda:

- Analyze the requirements of project2
- Assign tasks and requirements

Actions and Decisions:

- Requirements for preliminary analysis tasks
- Analysis of the means to achieve the goal
- Preliminary settlement of foreseeable problems
- Write the schedule of this project

Date: May11, 2018

Location: Eastern Resource Centre

Attendance: Shujing Xiao, Ziyi Xiong, Ziqi Zhang, Zhengqing Zhu

Duration: 5 hours

Agenda:

- Solve the problems of the project together
- Mission completion degree

Actions and Decisions:

- Working together to solve the problems with own parts
- Plan the next meeting time
- Allocate the completion progress of each time of the next meeting

---

Date: May16, 2018

Location: Eastern Resource Centre

Attendance: Shujing Xiao, Ziyi Xiong, Ziqi Zhang, Zhengqing Zhu

Duration: 5 hours

Agenda:

- Integration of completed codes
- Unify the format of the codes
- Update the schedules and tasks

Actions and Decisions:

- Report on the tasks which they have already completed
- Combine individual works
- Handle errors after the integration of individual works

Date: May22, 2018

Location: Eastern Resource Centre

Attendance: Shujing Xiao, Ziyi Xiong, Ziqi Zhang, Zhengqing Zhu

Duration: 5 hours

Agenda:

- Final integration of the code
- Test the functions
- Write the report

Actions and Decisions:

- Combine individual works
- Test the functions based on the project2 guidelines
- Write own parts of report and integrate them