Aidan Kealey - 20151256
Kailey Fejer - 20182434
Paulina Flores - 20152096

Testing Document – ELEC 477 – Lab 2

The testing strategy were designed to cover the main functionalities introduced in this lab, specifically focussing on the service directory, flat namespace integration and RPC robustness with the naming service now implemented. The test cases below detail the methods used to validate our implementation.

Test Case 1: 1 Directory, 1 Server, & 1 Client

The goal for this test case it to verify that the new implementation of the system maintains the same utility as the solution from assignment 1. This encompasses instantiating a server and client, establishing a connection between the two using a network, and facilitating the interaction between the two. The modification made to the implementation of assignment 1 include providing the server with its own nickname, making sure the client knows this nickname, and having the two interact with another server to register or look up/use the connection information of servers on the network in a directory.

The steps for this test are as follows (refer to Figure 1 for main.cpp implementations of these steps):

1. Start the service directory server.
2. Start a KV server with a service nickname and port.
    a. It will send its connection information to be stored/registered in the directory.
3. Start a KV client and provide it with the nickname of the KV server.
4. Let the Client execute.
    a. This involves putting and then getting key-value records located on the KV server.
5. After results, stop all services/servers.

If functioning correctly, the expected outcome would be the client successfully finding the service by its nickname, storing the key-value pair, and retrieving the correct value.

Aidan Kealey - 20151256
Kailey Fejer - 20182434
Paulina Flores - 20152096

Test Case 1 Code

```cpp
#include <thread>
#include <string>

#include "network.hpp"
#include "kvserver.hpp"
#include "kvclient1.hpp"

#include "sdserver.hpp"
#include "sdstub.hpp"
#include "DNS_custom.hpp"

extern std::map<std::thread::id,shared_ptr<Node>> nodes;
extern std::map<std::thread::id,string> names;

int main(int argc, char * argv[]){
    // handle command line arguments...
    int res = network_init(argc, argv);
    std::stringstream ss;

    // start all of the servers first. This will let them get up
    // and running before the client attempts to communicste
    std::cout << "Main: ***********************************" << std::endl;
    std::cout << "Main: starting servers" << std::endl;

    std::cout << "Main: starting directory server" << std::endl;

    shared_ptr<SDServer> sdServer = make_shared<SDServer>("Directory1");

    sdServer->setAddress("10.0.0.1");
    sdServer->init();
    sdServer->startServices();

    std::this_thread::sleep_for(std::chrono::milliseconds(1000));


    std::cout << "Main: starting other servers" << std::endl;
    // make shared broken?
    shared_ptr<KVServer> kvServer1 = make_shared<KVServer>("server1");
    // shared_ptr<KVServer> kvServer2 = make_shared<KVServer>("kvserver-server2");

    kvServer1->setAddress("10.0.0.12");
    kvServer1->setDBMFileName("server1");
    kvServer1->setNickname("snow");
    kvServer1->setPort(1515);
    kvServer1->init();
    kvServer1 -> startServices();

    // kvServer2->setAddress("10.0.0.13");
    // kvServer2->setDBMFileName("server2");
    // kvServer1->setDS("server2-soccer", 7272);
    // kvServer2->init();
    // kvServer2 -> startServices();
    std::this_thread::sleep_for(std::chrono::milliseconds(1000));

    std::cout << "Main: ***********************************" << std::endl;
    std::cout << "Main: init client" << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(1000));
    shared_ptr<KVClient1> kvClient = make_shared<KVClient1>("kvclient");
    kvClient->setAddress("10.0.0.14");
    kvClient->setServerName("snow");
    kvClient -> init();
    std::this_thread::sleep_for(std::chrono::milliseconds(1000));
```

```cpp
    std::this_thread::sleep_for(std::chrono::milliseconds(1000));

    std::cout << "Main: ***********************************" << std::endl;
    std::cout << "Main: init client" << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(1000));
    shared_ptr<KVClient1> kvClient = make_shared<KVClient1>("kvclient");
    kvClient->setAddress("10.0.0.14");
    kvClient->setServerName("snow");
    kvClient -> init();
    std::this_thread::sleep_for(std::chrono::milliseconds(1000));

    std::cout << "Main: ***********************************" << std::endl;
    std::cout << "Main: starting client" << std::endl;
    vector<shared_ptr<thread>> clientThreads;
    {
        // need a scope for the lock guard.
        // if this doesn't work put it in a function
    std::lock_guard<std::mutex> guard(nodes_mutex);

    shared_ptr<thread> t = make_shared<thread>([kvClient](){
        kvClient -> execute();
    });

    clientThreads.push_back(t);
    nodes.insert(make_pair(t->get_id(), kvClient));
    names.insert(make_pair(t->get_id(),"kvclient"));

    }

    // wait for clients to finish
    std::cout << "Main: ***********************************" << std::endl;
    std::cout << "Main: waiting for clients to finish" << std::endl;
    vector<shared_ptr<thread>>::iterator thit;
    for (thit = clientThreads.begin(); thit != clientThreads.end(); thit++){
        shared_ptr<thread> tmp = *thit;
        tmp->join();
    }

    std::this_thread::sleep_for(std::chrono::milliseconds(1000));

    std::cout << "Main: ***********************************" << std::endl;
    std::cout << "Main: calling stop services on servers" << std::endl;
    kvServer1 -> stopServices();

    std::cout << "Main: ***********************************" << std::endl;
    std::cout << "Main: waiting for threads to complete" << std::endl;
    // wait for all server threads
    kvServer1 -> waitForServices();

    std::cout << "Main: ***********************************" << std::endl;
    std::cout << "Main: stopping directory" << std::endl;
    sdServer -> stopServices();
    sdServer -> waitForServices();

    //std::this_thread::sleep_for(std::chrono::milliseconds(10000));

    google::protobuf::ShutdownProtobufLibrary();

    return 0;
}
```

Figure 1: Test Case 1 main.cpp code.

Aidan Kealey - 20151256
Kailey Fejer - 20182434
Paulina Flores - 20152096

Test Case 1 Output

```
aidankealey@Aidans-MacBook-Pro Assignment2 % bin/assign2
Main: ************************************
Main: starting servers
Main: starting directory server
Starting service: Directory1
in SDService::start
Main: starting other servers
Starting service: server1.KV_RPC
KVServiceServer registering new service
atempting to register new service
in SDStub init
successful send to socket: 1000
SDService::start() successful recieved in while
registering new server in directory
in SDService::registerServer
registered new server -- GOOD
service successfully registered in directory
Main: ************************************
Main: init client
Main: ************************************
Main: starting client
SDStub::searchForServiceMain: ************************************
Main: waiting for clients to finish

atempting to search for service
service nickname: snow
in SDStub init
successful send to socket: 1000
SDService::start() successful recieved in while
searching for server in directory
in SDService::searchForService looking for: snow
service for server (snow) found in directory
search results good, found server - server1 and port - 1515
search message good
put message requested
put result is 1
status is 1
get message requested
leaving get stub
status is 1
00 54686973 20697320 00206120 74657374 This is . a test
10 2121                                 !!
12
Main: ************************************
Main: calling stop services on servers
Main: ************************************
Main: waiting for threads to complete
Main: ************************************
Main: stopping directory
aidankealey@Aidans-MacBook-Pro Assignment2 %
```

Figure 2: Test Case 1 terminal output.

As seen in Figure 2, Test Case 1 has successfully executed. The directory server is created. Then the KV server is created and successfully registers. Next, the client is started and connects to the KV Server with the nickname snow. After that, the client then preforms the assignment 1 functionalities of put and get, resulting in a successful status of 1. Finally, the after the client finishes executing the servers and directory stop.

Test Case 2: 1 Directory, 2 Servers, & 2 Clients

Expanding upon test case 1, this case creates 1 directory server, 2 independent KV servers, and 2 independent KV clients, and simultaneously preforms key-value functionality (put and get operations) without error.

The steps for this test case are as follows (refer to Figure 3 for main_t2.cpp implementations of these steps):

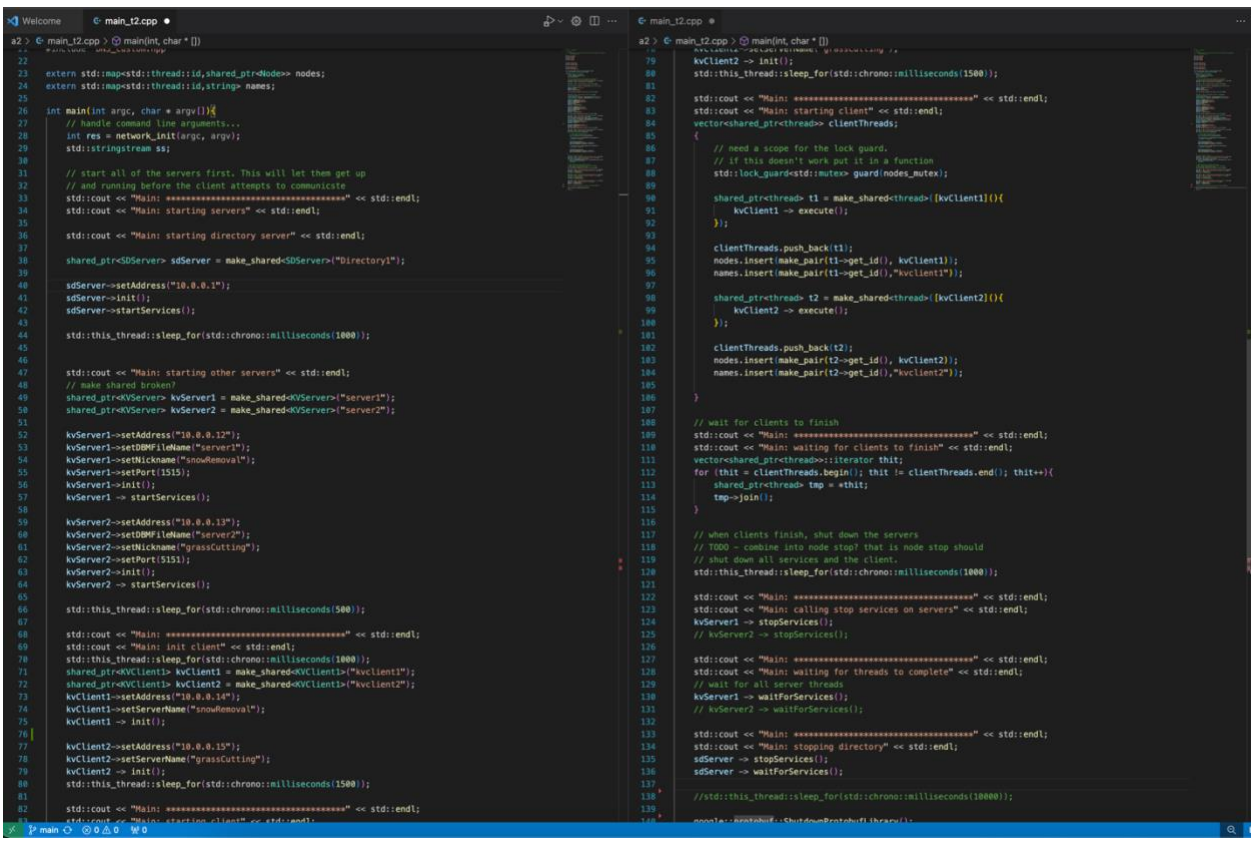
1. Start the service directory server.

Aidan Kealey - 20151256
Kailey Fejer - 20182434
Paulina Flores - 20152096

2. Start two separate KV servers with different service nicknames and ports.
    a. These servers will send their connection information to be stored/registered in the directory.
3. Start two separate KV clients and provide one with the first server's nickname and it with the nickname of the KV server.
4. Let the Client execute.
    a. This involves putting and then getting key-value records located on the KV server.
5. After results, stop all services/servers.

If functioning correctly, the expected outcome would be that the clients are successfully able to use the directory and communicate with the servers based on the given nicknames, without the other messages impacting the results.

Test Case 2 Code



```
extern std::map<std::thread::id,shared_ptr<Node>> nodes;
extern std::map<std::thread::id,string> names;

int main(int argc, char * argv[]){
    // handle command line arguments...
    int res = network_init(argc, argv);
    std::stringstream ss;

    // start all of the servers first. This will let them get up
    // and running before the client attempts to communicste
    std::cout << "Main: ********************************" << std::endl;
    std::cout << "Main: starting servers" << std::endl;

    std::cout << "Main: starting directory server" << std::endl;

    shared_ptr<SDServer> sdServer = make_shared<SDServer>("Directory1");

    sdServer->setAddress("10.0.0.1");
    sdServer->init();
    sdServer->startServices();

    std::this_thread::sleep_for(std::chrono::milliseconds(1000));


    std::cout << "Main: starting other servers" << std::endl;
    // make shared broken?
    shared_ptr<KVServer> kvServer1 = make_shared<KVServer>("server1");
    shared_ptr<KVServer> kvServer2 = make_shared<KVServer>("server2");

    kvServer1->setAddress("10.0.0.12");
    kvServer1->setDBMFileName("server1");
    kvServer1->setNickname("snowRemoval");
    kvServer1->setPort(1515);
    kvServer1->init();
    kvServer1 -> startServices();

    kvServer2->setAddress("10.0.0.13");
    kvServer2->setDBMFileName("server2");
    kvServer2->setNickname("grassCutting");
    kvServer2->setPort(5151);
    kvServer2->init();
    kvServer2 -> startServices();

    std::this_thread::sleep_for(std::chrono::milliseconds(500));

    std::cout << "Main: ********************************" << std::endl;
    std::cout << "Main: init client" << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(1000));
    shared_ptr<KVClient1> kvClient1 = make_shared<KVClient1>("kvclient1");
    shared_ptr<KVClient1> kvClient2 = make_shared<KVClient1>("kvclient2");
    kvClient1->setAddress("10.0.0.14");
    kvClient1->setServerName("snowRemoval");
    kvClient1 -> init();

    kvClient2->setAddress("10.0.0.15");
    kvClient2->setServerName("grassCutting");
    kvClient2 -> init();
    std::this_thread::sleep_for(std::chrono::milliseconds(1500));

    std::cout << "Main: ********************************" << std::endl;
```

```
    kvClient2 -> init();
    std::this_thread::sleep_for(std::chrono::milliseconds(1500));

    std::cout << "Main: ********************************" << std::endl;
    std::cout << "Main: starting client" << std::endl;
    vector<shared_ptr<thread>> clientThreads;
    {
        // need a scope for the lock guard.
        // if this doesn't work put it in a function
        std::lock_guard<std::mutex> guard(nodes_mutex);

        shared_ptr<thread> t1 = make_shared<thread>([kvClient1](){
            kvClient1 -> execute();
        });

        clientThreads.push_back(t1);
        nodes.insert(make_pair(t1->get_id(), kvClient1));
        names.insert(make_pair(t1->get_id(),"kvclient1"));

        shared_ptr<thread> t2 = make_shared<thread>([kvClient2](){
            kvClient2 -> execute();
        });

        clientThreads.push_back(t2);
        nodes.insert(make_pair(t2->get_id(), kvClient2));
        names.insert(make_pair(t2->get_id(),"kvclient2"));

    }

    // wait for clients to finish
    std::cout << "Main: ********************************" << std::endl;
    std::cout << "Main: waiting for clients to finish" << std::endl;
    vector<shared_ptr<thread>>::iterator thit;
    for (thit = clientThreads.begin(); thit != clientThreads.end(); thit++){
        shared_ptr<thread> tmp = *thit;
        tmp->join();
    }

    // when clients finish, shut down the servers
    // TODO - combine into node stop? that is node stop should
    // shut down all services and the client.
    std::this_thread::sleep_for(std::chrono::milliseconds(1000));

    std::cout << "Main: ********************************" << std::endl;
    std::cout << "Main: calling stop services on servers" << std::endl;
    kvServer1 -> stopServices();
    // kvServer2 -> stopServices();

    std::cout << "Main: ********************************" << std::endl;
    std::cout << "Main: waiting for threads to complete" << std::endl;
    // wait for all server threads
    kvServer1 -> waitForServices();
    // kvServer2 -> waitForServices();

    std::cout << "Main: ********************************" << std::endl;
    std::cout << "Main: stopping directory" << std::endl;
    sdServer -> stopServices();
    sdServer -> waitForServices();

    //std::this_thread::sleep_for(std::chrono::milliseconds(10000));

    google::protobuf::ShutdownProtobufLibrary();
```

Figure 3: Test Case 2 main_t2.cpp code.

Aidan Kealey - 20151256
Kailey Fejer - 20182434
Paulina Flores - 20152096

Test Case 2 Output

```
aidankealey@Aidans-MacBook-Pro Assignment2 % bin/assign2
Main: ************************************
Main: starting servers
Main: starting directory server
Starting service: Directory1
in SDService::start
Main: starting other servers
Starting service: server1.KV_RPC
KVServiceServer registering new serviceStarting service: server2.KV_RPC

KVServiceServer registering new service
atempting to register new service
in SDStub init
atempting to register new service
in SDStub init
successful send to socket: SDService::start() successful recieved in while
1000successful send to socket:
registering new server in directory1000
in SDService::registerServer

registered new server -- GOOD
SDService::start() successful recieved in while
registering new server in directory
in SDService::registerServer
service successfully registered in directory
registered new server -- GOOD
service successfully registered in directory
Main: ************************************
Main: init client
Main: ************************************
Main: starting client
Main: ************************************
Main: waiting for clients to finish
SDStub::searchForService
atempting to search for service
service nickname: snowRemoval
SDStub::searchForService
atempting to search for service
service nickname: grassCutting
in SDStub init
in SDStub init
successful send to socket: 1000
SDService::start() successful recieved in while
successful send to socket: searching for server in directory
in SDService::searchForService looking for: grassCutting
service for server (grassCutting) found in directory
search results good, found server - server2 and port - 5151
```

Figure 4: Test Case 2 terminal output 1.

```
service for server (grassCutting) found in directory
search results good, found server - server2 and port - 5151
1000
SDService::start() successful recieved in while
searching for server in directory
in SDService::searchForService looking for: snowRemoval
service for server (snowRemoval) found in directory
search results good, found server - server1 and port - 1515
search message good
search message good
put message requested
put message requested
put result is 1
put result is 1
status is 1
status is 1
get message requested
get message requested
leaving get stub
status is 1
00 54686973 20697320 00leaving get stub
status is 1
00 546869206120 74657374 This is . a test
10 2121                                  !!
12
73 20697320 00206120 74657374 This is . a test
10 2121                                  !!
12
Main: ************************************
Main: calling stop services on servers
Main: ************************************
Main: waiting for threads to complete
Main: ************************************
Main: stopping directory
aidankealey@Aidans-MacBook-Pro Assignment2 %
```

Figure 5: Test Case 2 terminal output 2.

As seen in Figure 4 and Figure 5, Test Case 2 has executed successfully. The directory server gets created. Then the KV servers are created and successfully registers with unique connection information. Next, the clients are started and individually connect to one of the two KV servers with their respected nicknames (snowRemoval and grassCutting). After that, the client then preforms the assignment 1

Aidan Kealey - 20151256
Kailey Fejer - 20182434
Paulina Flores - 20152096

functionalities of put and get on their respected servers, resulting in successful statuses of 1. Finally, the after the clients finish executing the servers and directory stop.