100505349 Distributed Systems Test Log

*100505349*

# Foreword

The following conducted tests were all operated on the final release build of the distributed systems project. Undocumented testing was completed throughout the development cycle to ensure the features shown met their requirements as defined in the course specification. Tests in this document are divided into three categories; Acceptance - referring to tests that demonstrate the correct usage of the product, edge case – referring to tests that test the boundaries of acceptable data and erroneous – inputs or procedures that are specifically expected to cause an error response.

Additionally, due to limitations of equipment, all tests assume the following restrictions:

* All tests are completed on a windows device running a windows 10 OS
* All tests are conducted on devices with access to ports 50000-50010
* Tests are only conducted on ports between these port restrictions
* All nodes contain a coordinated list of node IPs and authentication data
* Tests are all conducted over a local area network – the software is only certified for use over local networks

Any additional limitations or parameters specific to a test or group of tests will be specified in the test description.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Number | Category | Description | Input | Output | Result |
| 0 | Acceptance | The program may be started on a single device with an active internet connection. This should prompt the user to input the name of their node | Start the program | The program successfully starts | Positive |
| 1 | Edge Case | The program may be started on a single device with no internet connection. This may return an error asking for an internet connection – though this is not necessary if the device is able to initialise an IP regardless | Start the program with a device with no active internet connection | The program starts and initialises on an address regardless of connection | Positive |
| 2 | Acceptance | Once the program is running, the user may input the word “Control” to initialise a new control node | Enter “Control” when prompted | The program successfully generates a control node instance | Positive |
| 3 | Acceptance | Following from the previous test, enter the string “Client” to initialise a client node while a control node is active | Create a control node. Enter “Client” when prompted | A client instance is initialised | Positive |
| 4 | Acceptance | When a control node exists, generation of new clients should start on port 50006, and each new client instance should generate on the next port in sequence | Start a control node. Create two clients | The second client instance initialised on port 50007 | Positive |
| 5 | Edge Case | The input reader is expected to be case sensitive, so inputs of “client” and “control” are expected to produce error messages. If no error message is produced but the program runs normally, this is still successful. | Enter “control” when prompted | An error message is produced | Positive |
| 6 | Edge Case | Attempt to start a control node on an IP not listed in the active IPs directory. This should run with no issues | Change the IP file to include the IP ‘192.168.0.36’ | An error message is produced | Positive |
| 7 | Edge Case | Clean the list of IPs in the \_ConnectionList directory. This should produce an error message or allow the program to run as before | Change the IP file to include no characters | The program crashes | Negative, Crash |
| 8 | Erroneous | Attempt to start an instance of a node that exists on the system but should not be usable by clients. This should return a message telling a user they may not initialise this type of node | Enter “Echo” when prompted | An error message is displayed | Positive |
| 9 | Erroneous | Attempt to initialise a client when there is no active control node on any listed IP. An error message should be displayed to show that a control node must exist on the network before this operation can be completed | Enter “Client” when prompted | An error message is displayed | Positive |
| 10 | Erroneous | Attempt to initialise a control node when one already exists on the same IP. This should not be permitted and should return an error response. | Start a control node instance. Start a second control node instance on the same device. | An error message is displayed | Positive |
| 11 | Erroneous | Attempt to initialise a node with a name that does not reflect a valid node in the system. This should produce the same error as if an existing but invalid node input was input. | Enter ‘Abc’ when prompted | An error message is displayed | Positive |
| 12 | Erroneous | Input a blank character to the node input screen. This should once again produce an error message | Enter ‘’ when prompted | An error message is displayed | Positive |
| 13 | Acceptance | Start a control node and client node on the same device. Attempt to authenticate the client using a valid username and password. This should spawn an authentication node which will provide the client authentication | Enter ‘LOGIN|ADMIN|Egg’ when a connection has been established | An authentication node is generated which provides a client authentication | Positive |
| 14 | Acceptance | Attempt to login using another login in the login file, to ensure all logins listed are validated. | Enter ‘LOGIN|Natost|Password’  when a connection has been established | An authentication node is generated which provides a client authentication | Positive |
| 15 | Acceptance | Attempt to login using invalid credentials. Then follow by inputting valid credentials. | Enter ‘LOGIN|EGG|EGG’ followed by ‘LOGIN|ADMIN|Egg’ | An auth node is generated and provides a denied response. When the second input occurs, it grants authentication | Positive |
| 16 | Edge Case | Input a valid login with invalid case. This is not expected to work however it is of no consequence if it does. | Enter ‘LOGIN|admin|Egg’  when a connection has been established | Authentication node is generated. Authentication is denied | Positive |
| 17 | Erroneous | Attempt to login using a valid username and a valid password that are not tied together. This should produce an authentication denied response. | Enter ‘LOGIN|ADMIN|Password’ when a connection has been established | Authentication node is generated. Authentication is denied | Positive |
| 18 | Erroneous | Enter an invalid username and password combination | Enter ‘LOGIN|EGG|EGG’ when a connection has been established | Authentication node is generated. Authentication is denied | Positive |
| 19 | Erroneous | Clear the login file of all data and attempt to login using an invalid login. This is as a response to the prior connection IP test and while it should not permit connection, it may cause a crash. | Clear the Login file. Enter ‘LOGIN|ADMIN|Egg’ when a connection has been established | Authentication node is generated. Authentication is denied | Positive |
| 20 | Acceptance | Attempt to use a command on a node with proper authentication. This should permit the command to be processed. | Login using the ADMIN account and enter ‘ECHO|AAA’ | Authentication is granted and the echo node spawned provides the ‘AAA’ response | Positive |
| 21 | Edge Case | Attempt to use the login command once login is already granted. The results of this should not matter but must not crash the system. | Login using the ADMIN account then use the ‘EGG|EGG’ login to get an auth deny response. Then attempt to use the ‘ECHO|AAA’ command | Authentication is granted but upon the second input it is not revoked. This does not cause any additional errors but does allow for the use of the echo command | Positive |
| 22 | Erroneous | Attempt to use an external command when authentication has not been attempted. This should prompt the user to login. | Enter ‘ECHO|AAA’ on connection | An error message directs the client to login to the system. This error is sent by the server and does not spawn an authentication node. | Positive |
| 23 | Erroneous | Attempt to login with an invalid login, and then proceed to attempt to use an external command. | Enter ‘LOGIN|EGG|EGG’ and then ‘ECHO|AAA’ | The authentication node is spawned, and the client is not granted authentication. The client is then prompted to login when attempting to use the echo command | Positive |
| 24 | Erroneous | Attempt to login using valid credentials on one client. Then spawn another and attempt to use a command. This should not permit the second client to use commands, despite running on the same IP. | Spawn a client and login using ADMIN, then spawn a second client and enter ‘ECHO|AAA’ | The second node is prompted to login to the system and the echo command is not processed | Positive |
| 25 | Acceptance | Login on a client before taking the authentication node that authorised the login down. This should still permit a client to use commands. | Login into the ADMIN account and then close the authentication node. Then enter ‘ECHO|AAA’ on the client who is logged in | The client is able to send the echo command despite the authentication node no longer existing after providing authentication | Positive |
| 26 | Acceptance | Start two clients and log into the same authentication node using different credentials. This should only spawn one authentication node that handles both requests | Login to the ADMIN account on one client, login to Natost on another. | Both logins are processed by the same authentication node and both clients are granted access. | Positive |
| 27 | Acceptance | Login on one client to the ADMIN login, then close the authentication node and start a second client to attempt login using the Natost login. This should spawn an authentication node to replace the missing node. | Login into the ADMIN account with one client. Start a second client, then login to the natost login. | A replacement authentication node is generated in response to the death of the first. Both clients are authenticated properly. | Positive |
| 28 | Acceptance | Start two clients which both attempt to login using the same login credentials. This should permit both authorisation. | Login to the ADMIN account on both clients using the same authentication node. | Both clients receive authorisation as expected. | Positive |
| 29 | Acceptance | Start three clients which each attempt to login. This should spawn up a second authentication node to handle the traffic | Login to the ADMIN account three times across three separate clients on the same device | A second authenticator is spawned on the device to handle the traffic | Positive |
| 30 | Acceptance | Close a client connection. This should cause no problems | Close a connected client | All nodes the client has a connection to process the connection has ceased and do not crash or cause any errors. | Positive |
| 31 | Acceptance | Close a control node while clients are connected. This should stop all further inputs into the system for the client. | Close a control node with connected clients | The client can no longer input commands. Even if a new control node is generated the client is unable to connect unless it is restarted | Positive, but could be improved. |
| 32 | Acceptance | Start an echo node using a logged in client and attempt to use the echo command with a single parameter in ascii. | Login and enter ‘ECHO|ABC’ | An echo node is spawned and responds to the client with ‘ABC’ | Positive |
| 33 | Acceptance | Spawn a second client and attempt an echo of ‘ABD’ then attempt an ‘ECHODUMP’ on both to display all echos created on this node. | Enter ‘ECHO|ABC’ on one client then ‘ECHO|ABD’ on another. Then use ‘ECHODUMP|’ on each | Each client both sees the echo requests submitted by the other to the node | Positive |
| 34 | Erroneous | Attempt to use a second client with invalid credentials to submit an echo request to a pre-existing echo node. This should prove the echo node requires authentication | Enter ‘ECHO|ABC’ | The user is prompted to login. | Positive |
| 35 | Acceptance | Enter an echo command from two clients into an echo node. Disconnect one of these clients before starting a new client instance, then attempt to connect to the first echo node and then process an echodump command. This should return the data sent by the now-dead client, demonstrating the echo node is storing the data. It also shows that the control node keeps track of the load on its child nodes. | Enter ‘ECHO|ABC’ on one node, then ‘ECHO|EGG’ on another. Close the latter client and open a new one, login and enter ‘ECHODUMP|’ | The client is connected to the pre-existing echo node as it has not reached its load capacity, then the echos of the dead node are displayed when echodump is sent. | Positive |
| 36 | Edge Case | Enter an echo command with more than one parameter. This parameter should be ignored by the node | Enter ‘ECHO|ABC|ABD’ | Only the ABC parameter is echoed. Using the ‘ECHODUMP|’ command shows the ABD input was not stored | Positive |
| 37 | Edge Case | Enter ‘ECHODUMP|’ into an echo node that has not yet received any echo requests. This should display an empty set. | Enter ‘ECHODUMP|’ | An empty set is returned. | Positive |
| 38 | Erroneous | Enter no parameters into the echo request. This should not cause any errors | Enter ‘ECHO|’ | A space character was echoed back, using ECHODUMP| displayed the space as stored. | Positive, but could be improved. |
| 39 | Erroneous | Enter an echo request without a parameter without the appended ‘|’ delimiter | Enter ‘ECHO’ | The echo node responds with a response of ‘AUTHGRANT’ – demonstrating that it has incorrectly displayed the appended authentication parameter. | Negative |
| 40 | Acceptance | Enter a valid dictionary request from a client with no connection to a dictionary node | Login and enter ‘DICT|Isak’ | A new dictionary node is spawned and responds with the definition of ‘Cool’ | Positive |
| 41 | Erroneous | Attempt to use the dictionary node commands without a valid login | Enter ‘DICT|Isak’ | The user is prompted to login to the system | Positive |
| 42 | Erroneous | Attempt to enter an invalid dictionary key | Login and enter ‘DICT|EH’ | Received an error message describing that the key was not present in the dictionary | Positive |
| 43 | Erroneous | Attempt to use a dictionary value as a parameter | Login and enter ‘DICT|Cool’ | Received the same error message as previously mentioned | Positive |
| 44 | Acceptance | Attempt to use the DICTADD command to specify a new key value pair to be inserted in the dictionary. Then use the DICT command to return the value | Login and enter ‘DICTADD|ABD|EGG’. Then enter ‘DICT|ABD’ | Returned the message ‘EGG’ | Positive |
| 45 | Acceptance | Spawn a second client and attempt to use the DICT command to define the previously written key | Login and enter ‘DICTADD|ABD|EGG’. Login on a new client and enter ‘DICT|ABD’ | The second client receives the message ‘EGG’ – showing that the dictionary node stores the data | Positive |
| 46 | Erroneous | Attempt to spawn a new client and enter the DICT|ABD command while there are two connections to the dictionary storing the ABD key. This should not return a definition as the newly spawned dictionary node should not have this data. | Login and enter ‘DICTADD|ABD|EGG’. Login on a new client and enter ‘DICT|ABD. Login on a new client and enter ‘DICT|ABD’ | A new dictionary node was spawned to handle the third clients request, this node did not carry the definition for ABD and therefore did not report the information. | Positive |
| 47 | Acceptance | Connect a client and login, before using the PLAYLIST command to display all music connected. For this test, there is only one device. Later tests will demonstrate the playlist command across a network | Login and enter ‘PLAYLIST|’ | The applicable songs, “Moonsetter” and “Test” were listed. | Positive |
| 48 | Erroneous | Attempt to use the PLAYLIST command on a client that has not logged into the system | Enter ‘PLAYLIST|’ | The client is not shown the song list | Positive |
| 49 | Acceptance | Attempt to play the song “Moonsetter” using the “!PLAY” command on a logged in client | Login and enter “!PLAY:Moonsetter” | A file distributor node is spawned, and the client begins to download the music. When complete, the music begins to play from the client node. When the client is closed the music stops. | Positive |
| 50 | Acceptance | Attempt to play a song that was not specified in the PLAYLIST command. | Login and enter “!PLAY:Egg” | An error message is returned. | Positive |
| 51 | Acceptance | Use the client volume command to change the volume | Enter “VOLUME|0” when music is playing | The music lowers in volume without stopping the music | Positive |
| 52 | Edge Case | Attempt to change the system volume while nothing is playing. This should be accepted and any subsequently played music should follow the set volume. | Enter “VOLUME|5” before music is played. Begin playing music. | The volume setting was applied despite not being connected to a file node. When music was played the volume was quieter than previous tests | Positive |
| 53 | Erroneous | Enter a volume value greater than 100 | Enter “VOLUME|120” | The volume setting is automatically set to 100, the maximum volume value | Positive |
| 54 | Erroneous | Enter a volume value less than 0 | Enter “VOLUME|-1” | An error message is displayed dictating that the value is invalid | Positive, but could be improved |
| 55 | Erroneous | Enter a non-integer volume value | Enter “VOLUME|a” | The previously seen error message is displayed | Positive |
| 56 | Erroneous | Enter no volume value | Enter “VOLUME” | The same error message is seen again | Positive |
| 57 | Acceptance | Use the pause command to stop playback | Enter “PAUSE|” while music is playing | The music stops playing | Positive |
| 58 | Edge Case | Enter the pause command while music is not playing. Regardless of the result, if there are no errors then this test is positive | Enter “PAUSE” while no music is playing | An error message saying that no music is playing is shown. | Positive |
| 59 | Edge Case | Enter the pause command after already pausing playing music. | Enter “PAUSE” twice while music is playing | No errors occur | Positive |
| 60 | Acceptance | Use the resume command to resume playback | Use the aforementioned pause command then enter “RESUME|” | The music resumes playing at the time it was paused at | Positive |
| 61 | Edge Case | Use the resume command when music is already playing | Enter “RESUME|” while music is playing | No errors occur | Positive |
| 62 | Erroneous | Attempt to use the resume function while no music is playing | Enter “RESUME|” while no music is playing | An error message says that no music is playing | Positive |
| 63 | Acceptance | Use the stop command to cease playback. A following resume call should not allow the music to continue. | Enter ‘STOP|” followed by “RESUME|” | The music stops and attempts to resume the music return an error message. | Positive |
| 64 | Erroneous | Attempt to use the stop command while no music is playing | Enter “STOP|” without music playing | The error message mentioned is shown again | Positive |
| 65 | Edge Case | Attempt to download and play more music while music is already playing. | Enter “!PLAY:Test” followed by “!PLAY:Moonsetter” | The music playing is stopped, and the second input begins playing. There is no sound overlap. Any changes to volume/pause/play all apply as expected | Positive |
| 66 | Edge Case | Attempt to play music through two clients connected to the same device | Enter “!PLAY:Moonsetter” on one connected client and “!PLAY:Moonsetter” on another connected client | The playing music overlaps, demonstrating that each client has its own music playing. Further tests show that commands like “PAUSE” only impact the client they are input into | Positive |
| 67 | Acceptance | Test the availability of other commands to a client while music is playing | Play moonsetter and then attempt an echo request | The echo request is processed as usual despite the music playing | Positive |
| 68 | Acceptance | Attempt to initialise a node on a device with more than one applicable IP addresses. The program should select the most appropriate IP to connect through | Start a control instance on a multiple-IP device | The device connected to the correct IP automatically | Positive |
| 69 | Acceptance | Attempt to connect to a control node using a foreign client using an IP present in the IP file | Add two IPs to the connection list – start a control on one and a client on another. | The nodes were able to create an active connection | Positive |
| 70 | Acceptance | Attempt to connect to a control node using a foreign client using an address not in the IP list. This should still be accepted | Add only one IP to the connections list – that of the control – and then attempt a connection through the client | The nodes made a connection | Positive |
| 71 | Acceptance | Attempt to use the authentication system from a remote client | Start a control on one device and a client on another. Send an ADMIN login request from the client. | The control node spawned an auth node on its own device and then authenticated the login data | Positive |
| 72 | Acceptance | Attempt to use the echo node from an authenticated foreign client | Using the same setup as the prior test – enter “ECHO|BBB” | The control spawned an echo node. The echo node sent an echo of “BBB” to the client | Positive |
| 73 | Acceptance | Attempt to use the dictionary node from an authenticated foreign client | Again, using the same setup enter “DICT|Isak” | A dictionary node was spawned and responded with “Cool” to the client | Positive |
| 74 | Acceptance | Attempt to use the playing functionality to download a song onto the authenticated foreign client from the control node. | Remove the music file from the client nodes running directory. Send a request of “!PLAY:Moonsetter” to the control node | The control node generated a file distributor node. The music was downloaded to the client at a slower rate than would be expected on a one-device connection, but still played the music with no errors. | Positive |
| 75 | Acceptance | Attempt the running of a second control node on the network, on a device using an IP in the connections list | Add two IPs to the connections list, start a control on each device. | Both control nodes started without issue, and then communicated with eachother immediately. Connected clients routed to the first control node in the list regardless of their own device | Positive |
| 76 | Edge Case | Connect a number of clients to a control node that should exceed its capacity. New connections should then be routed towards the next living control node in the network | Create two control nodes with IPs in the connections list. Attempt to connect four clients to the first listed IP. | The fourth client was routed to the second control node in the list, balancing the load across the network. | Positive |
| 77 | Edge Case | Attempt to spawn a number of clients that exceeds the maximum number of clients a control can handle without another control existing. This should force the control node to accept the connection regardless of exceeding its limit. | Create four clients that all connect to the same control node, the only listed node in the \_ConnectionsList file | The control node accepts the connection despite exceeding its capacity, as it cannot find a valid redirect for the client | Positive |
| 78 | Edge Case | Attempt to spawn a number of clients to exceed the collective capacity of the control nodes set. This should force the first connected control node to accept the connection | Spawned two control nodes and then attempted to connect eight nodes across two devices (to not exceed the port capacity) | The first control node crashed upon attempting to find a connection. | Negative, Crash |
| 79 | Follow-up Test | Attempt the previous operation but using a third control node, to ensure the problem is solely with reaching the end of the list of connections rather than redirection. | Appended a third IP into the connections list set and ran a control node on each device. Then attempted to connect 12 clients | Connections correctly routed to the third control up to its limit, after which the first control node crashed. This shows that the issue lies with the connections list rather than the redirection | See Prior |
| 80 | Acceptance | Attempt to use a playlist command to discover music that is available on the network but not on each individual control node. | Remove “Test.wav” from the running directory of all control nodes but one. Start a client to connect to a control node that does not have the file. Authenticate and enter “PLAYLIST” | The playlist returned the file, demonstrating the playlisting is able to communicate across the network. | Positive |
| 81 | Acceptance | Attempt to play the music from a client connected to a node that does not have the music in question. | Start a client to connect to the first node, one without “Test.wav”, and then submit a request of “!PLAY:Test” | The control node routed the command to the other control node, which spawned up a file distributor which connected to the initial client. Notably, the client made no connection to the second control node. | Positive |
| 82 | Acceptance | Following on from the first test, attempt to play a song not present on the newly spawned file distributor. This should spawn up a new file distributor on a device that has the music in question. | Remove “Moonsetter.wav” from the second client. Start up the two control nodes and connect a client. Authenticate and request test, then request moonsetter | Each control node spawned up a file distribution node to handle the music request, with only one control node having a client connection. | Positive |
| 83 | Acceptance | Start a control node not in the first slot of the connections list, then start a client. The client should then route to the first available control node. | Start a control on the device with the second IP in the set. Attempt to start a client on any device | The client routed through to the second control despite the first control being inactive | Positive |
| 84 | Edge Case | Spawn two control nodes, one on the first IP and one on the last (in a set of three). Spawn enough clients to cause the first client to attempt to balance its load. This should then mean the new clients are connected to the third IP in the set, as the second IP is inactive. | Start two control nodes on the first and last IP. Start 4 clients | The last client was redirected towards the other active control node | Positive |
| 85 | Edge Case | Fill the capacity of one control node, then begin adding to the next node. Remove an active client from the first control node and then attempt to create a new client. The new client should route to the first control node, as it has now been freed up. | Start two control nodes. Start 4 clients to connect to the first control node. Close the first spawned client and then spawn a new client | The newly spawned client connected to the first control node, demonstrating the control node recognised it had free space. | Positive |
| 86 | Erroneous | Close a request handler node (Echo, dictionary etc.) while a client has an active connection to it, to ensure that no errors occur when a client attempts to use the command in the future. | Spawn a new control node. Spawn a client, authenticate and enter “ECHO|BBB”. Close the echo node and then attempt a “ECHO|BBBE” | A new echo node was spawned to replace the first, using the “ECHODUMP” command it showed that this new node was infact a new instance, as it only listed “BBBE” | Positive |

# Summary

With the 86 conducted tests, it is apparent that the software can demonstrate all its required functionality, while also including elements of load balancing and fault tolerance. The primary functionality of the system, file downloading, presented no errors upon testing – and even allowed for speed improvements by creating a persistent connection to the file distributor node able to handle the requested file. This allows the potential to reduce overall network load, as if devices are set up to include files of a similar type, clients will have access to music of the same genre as previous requests without the need for further routing through control nodes.

Only 3.5% of tests resulted in failure (when excluding a follow up test, test 79, which demonstrated the same failure as the previous test), and only two of these tests that resulted in a failure ended up crashing a node. These errors, while higher in number than anticipated, do not impact any of the core functionalities of the program – only crashing a node when the software has been incorrectly set up or has reached its network client capacity (a principle that belongs to load balancing rather than distribution).

It should also be noted the network is unfortunately prone to decreased control node communication speeds proportional to the number of control nodes in the IP list, as a control node must attempt to generate a connection to each in sequence. However, this procedure is threaded to ensure client experience is not impacted in any way, but high IP loads may cause instability regardless.