Carson Rohan and Lucas Steffens

Dr. Jaiswal

CS 470

21 March 2021

# Project Report

For this project, we were given the task of creating, developing, and implementing a high-availability cluster (HAC) protocol in Java. Created over the well-known user datagram protocol (UDP), our protocol provides a list of available nodes within a network, and keeps track of nodes that are considered “alive” and “dead”. Implemented in both client-server and peer-to-peer (P2P) architectures, our HAC protocol offers many features and measures that make it reliable.

When developing a protocol, it is important to ensure that data is only exchanged between nodes that are available and active. That is why we have implemented a thirty second time to live (TTL), which labels nodes as “dead” if no communication has come from them within the TTL window. To do this, we created an object for every node that holds its IP address, port number, and the last time data has been sent from it. While it is important to ensure that dead nodes are not being communicated with, it is also important that previously dead nodes can reconnect and reenter the cluster. This is why our HAC protocol also has the capability to revive dead nodes, under the condition that the dead node resends a request to join the HAC once again.

Server failure is also an important occurrence to keep in mind while developing a protocol. Knowing this, we implemented a failover feature. Essentially, if the server were to fail or go offline, a client would be able to run as a server, and continue the cluster with the other clients in the network. To detect a server failure, we created an adjustable timeout counter (with

a default value of 5 seconds) that throws an exception if a client does not receive a response from the server within the given time. If the exception is thrown, then the first client to catch the exception becomes the new server. Additionally, if the server were to later come back to life, then it has the capability of continuing as a client. This makes our HAC dependable and reliable.

To demonstrate our communications inside the network, each client pings the server (or each other, in peer-to-peer) at varying random intervals, ranging from zero to thirty seconds. For that reason, clients are unlikely to be considered "dead" at any time. If one desires to see a client die and come back, one could either increase the upper bound of the random number generator, or decrease the TTL value.

To run and setup the network protocol, simply run the main function in the UDPServer.java file to start the server up, and run UDPClient.java to run a client. Proceed by following the directions outputted in the terminal. Note that every machine involved in the network needs both files to operate correctly. If peer-to-peer is desired, first enter the desired IPs into the config file newline separated, then run the UDPPeer.java file for each client.

Our team member's contributions were divided up equally, and can be seen as follows:

1. Carson Rohan
    a. Developed the ability to detect new nodes in the network as well as node failure
    b. Created the majority of the peer-to-peer architecture
    c. Wrote the README file
2. Lucas Steffens
    a. Developed the revival of dead nodes and the ability to send node statuses to all other nodes in the network
    b. Wrote the project report
    c. Implemented the failover capability in the client-server architecture