CS5450 Lab 2 Report

Peer to Peer Networking

Will Krasnoff//Ishan Virk
wk285//isv3

Implementation Details

Socket Setup: Servers are configured with 2 sockets, 1 TCP that is connected to the proxy server, and another UDP socket that is bound to port 20000+pid. Both sockets are non-blocking and IO operations are selected from active sockets using FDSET and select() calls.

Internal Sequence Number Setup: Messages are assigned a sequence number by the server that receives them from the client. A counter is maintained on each server to keep track of the local sequence number for *i* incoming messages. If a server crashes, the server will update it's internal sequence number to catch up with the vector clock (this is not a perfect solution, as discussed below).

Vector Clock Implementation: An array of integers is maintained at each server. The array indices correspond to the server numbers and the value in each array represents the lowest sequence number message the server has NOT seen from the corresponding peer (as specified). The servers still receive and store messages with higher sequence numbers and updating the vector clock will jump once the gap is filled.

Message Log + Message ID implementation: Each server stores a heap allocated array containing all of the messages it has received so far. It also stores a corresponding array that stores the origin server and sequence number of the sender of the message. These two arrays correspond to one another by index. The seqnum and origin are stored to ensure no repeat messages are incorrectly propagated and for cumulative vector clock update.

Send status message on startup: A server sends status messages to all of its neighbors upon startup. This helps to catch up crashed servers when they come back online, and helps to prevent consistency and repeat sequence number issues when a server comes back online.

Vulnerabilities

No Encryption: None of the messages are encrypted when being sent, so this system is vulnerable to attacks where messages may be intercepted (such as man in the middle attacks).

Hardcoded ports: Furthermore, the system architecture is fully deterministic. In other words, the ports on which "neighbors" in the peer to peer network can be found are hardcoded. Given this architecture, it would be very easy for a malicious program to join the network to pose as a peer and steal messages, or to act to attack the network's function.

Vulnerable to intermediate node message "Hoarding": One way a malicious node could compromise performance on the network is by inserting itself as an intermediate node and hoarding the messages that come through. Since nodes in the network are not all connected to one another they depend on intermediate neighbors to transmit messages to the other side of the network. If a malicious node is in the middle and doesn't forward messages, it will split the network in half and cause inconsistency and loss of functionality.

Vulnerable to message flooding: This network is also potentially vulnerable to a DDoS attack of sorts. If a malicious party knows the port that any/all of the peers are receiving messages on, they can flood that port with UDP messages and potentially compromise the functionality of the p2p network.

Internally generated seqnums vulnerable to server failure/inconsistency: One scenario that is possible given the current structure of the network is the following: Server 1 participates in the peer to peer network, and sends several messages, after which it crashes. When the server comes back up, the local sequence number will be reset to 1. While the server does send out an initial status message upon startup, this is not guaranteed to synchronize the server with the rest of the network. If server 1 then sends a message before it's local sequence number catches up to the rest of the network, it's new sequence messages will have a sequence number that other servers have already seen, thus the messages will be ignored and lost.