Project 1 Design

1. Local file update

Each node will have a version vector for each file. The version vector will be a map with the nodeID as the key and the write count as the value. That is, the version vector for a file will be a map associating a node to the number of times that the node has received a local update to the file. (Another alternative is to use the local clock instead of the counter.) When a node receives a local file update, it increments the corresponding write count number for the given fileID using the local nodeID. (In the alternative case, when node j does a local update on file n, it the version value for node j of the version vector for file n is set to the current time of j.)

2. Connection setup

After the local file update, FileSyncrhonizerThread does service discovery and connection setup. It first broadcasts messages to announce itself to the network. As it receives broadcast messages from other nodes, it keeps the connectivity information in a list.
FileSynchronizerThread spawns server (actually before sending broadcast), and spawns ClientServiceThread to connect to each peers in-range.
ClientServiceThread is also used when the server accepts a peer. The ClientServiceThread contains the socket that is used to communicate to the peer. In order to have one thread (or one socket) per peer, we keep a map to hold the one to use. We keep the one that establishes the connection first. However, two connected peer may see the different order. For example, let's consider two possible existing TCP connections A and B between a pair of nodes. One node may register A first to the map while the other goes with B. So after connection establishment, the node exchanges whether they want to keep this connection. If they agree to either keep or kill, we follow the agreement. Otherwise, we go with the node whose ID is greater.
So we ensure that a pair of node establishes one TCP connection. The associated ClientServiceThread is the worker thread we use to synchronize information.

3. Synchronization

The ClientServiceThread handles Synchronization.
At the beginning of the synchronization phase, a node will send all its version information using the socket. It then reads the peer's version information.
Once we exchange the version vectors, for each local file ID A, find the peer's version vector for A and detect one of the following cases:

1) A's local version is newer than the peer's, or the peer does not have file A.
2) A's local version is the same as peer's version.
3) A's local version is older than peer's version.
4) A's local version and the peer's version is not comparable. (conflict)

In case 1), do the followings.
    Send the new version and the file content.
In case 2), do nothing.

In case 3), do nothing.

In case 4), mark the file to be in conflict. When passing the version vector for the file, send the “conflict” bit, so that any comparison will lead to case 4, later on.

After this, now we read the information from peer and parse them. We read the file update from peer.

If there was any update from peer or there was a new conflict, repeat the above process (version vector exchange, and file update) in threads serving other nodes. (This deals with the propagation of update in chained topology.)