Collaborative Spreadsheet Protocol

Drew McClelland, Kameron Paulsen, Camille Rasmussen, Jessie Delacenserie

March 30, 2015

Contents

1 Rationale

The purpose of this document is to propose a protocol which may be used to create a collaborative spreadsheet application. The goal is to alter the existing spreadsheet program written fall 2014 so that multiple users can simultaneously connect to and edit a single spreadsheet. This specification was created to communicate the cause-effect relationships between server and client.

2 Overview

The proposed protocol is designed in accordance with the guidelines set forth by Peter Jensen. It assumes a server-client relationship only, where request packets are sent from the client to the server, and each client receives subsequent status updates from the server; no peer-to-peer communication is included in the protocol. Any number of clients can connect to the server simultaneously and open any spreadsheet on the server. Every client connected to a single spreadsheet may request changes and will receive updates of every approved change. Clients also have the option to save the current version of the spreadsheet (eliminating the ability to "undo" changes made before that point in time) and close the spreadsheet, with the guarantee that the next time the sheet is opened the client will receive the most updated version.

This specification describes the details of the communication between the clients and the server. The communication will exist as packets of information being sent between the two entities. Specifications on how each end handles packets is also outlined. It also discusses and provides a protocol for handling conflicts in the priority of change requests and the possibility of data loss due to accidental client-server disconnections.

3 Client Login and Spreadsheet Chooser

Once users have signed in with a valid login, they have access to any spreadsheet on the server. Users may also choose to create a new (blank) spreadsheet.

3.1 Client Login and Username

Upon opening the client, the user will be prompted to enter a username. Usernames must be unique and approved by the server against a list of active users. If the username is currently in use, the client is prompted for a different username. If the user provides a valid username, the server adds the username to a list of active users and sends a list of all spreadsheets to the client. This process is shown from the server end in Figure 1.

3.2 Creating a new spreadsheet

If the user chooses to create a new spreadsheet, they are prompted to provide a unique title for their spreadsheet and an xml file representing an empty spreadsheet is created. The new spreadsheet file is added to the list of spreadsheets on the server and the xml file is then sent to the client, providing them a blank sheet to begin editing.

3.3 Spreadsheet Selection

Upon login, the user can select any of the existing spreadsheets to open. Once selected, the requesting client is added to the list of users for the given spreadsheet and the server creates an xml file of the current spreadsheet state and sends it to the client. The scope of this xml file lasts only long enough for the requesting client to receive the spreadsheet. This ensures that changes that were enqueued before the xml file was created are propagated to the requesting client, and also ensures that the spreadsheets isn't saved without the active users requesting a save. Spreadsheet requests are diagramed in Figure 2.

4 In Spreadsheet

Once the client has an active spreadsheet they can do five things: edit the current spreadsheet, undo the last change, save the current spreadsheet, close the current spreadsheet, or request to open another spreadsheet. All packet commands are listed in section 5.

4.1 Making a change

When editing a cell in the spreadsheet, a change is not reflected in the client's sheet, nor propagated to the server, until the user commits the change. At this time, a packet is sent to the server requesting the change. The server enqueues each change request as it is received. The server then dequeues each change one at a time and checks for circular dependencies and formula errors resulting from cell references. Formula format errors are checked within the client's spreadsheet GUI. Once deemed valid, the change is propagated to each client in the active users list for the given spreadsheet. The spreadsheet name must be sent to each client along with the change because clients may have more than one spreadsheet open at a time. The use of a queue to prioritize the changes eliminates the possibility trying to make multiple edits at the same time and creating conflicting cells. Figures 3, 4 and 5 illustrate how changes will be made.

4.2 Saving a spreadsheet

The client can request to save a spreadsheet at any point in time while that spreadsheet is open. Upon request, a packet containing the message required to save the spreadsheet is sent to the server. A new xml file is created for the specified spreadsheet and replaces the one currently saved in the list of spreadsheets on the server.

4.3 Closing current spreadsheet

The client can request to close the current spreadsheet. The user will then be prompted to save the spreadsheet. If they choose yes, the spreadsheet will be saved as in the previous section. If they choose not to save the spreadsheet, the user's client will close and no unsaved changes will be saved. If the last active client exits without saving, the current unsaved spreadsheet changes will not be committed to the server. Instead, a backup copy is saved to the server as '*~filename*' with filename being the name of the spreadsheet. This backup copy will overwrite the autosave version described in section 6.

4.4 Opening another spreadsheet

If a user decides to open a new or existing spreadsheet, a request packet is sent to the server and the user will receive an xml file as described in sections 3.2 and 3.3.

5 Packets

Below are the packet commands used for communication between the clients and server. All commands are plain text and must end with a newline character.

<u>Client to Server</u>

- User login
```
LOGIN   username
```
- Create new spreadsheet
```
NEW   username   spreadsheet_name
```
- Open existing spreadsheet
```
OPEN   username   spreadsheet_name
```
- Update a cell
```
UPDATE   username   spreadsheet_name   cell_name   cell_contents
```
- Save spreadsheet
```
SAVE   spreadsheet_name
```
- Undo cell change
```
UNDO spreadsheet_name
```

- Close spreadsheet

  CLOSE   username   spreadsheet_name

- User logout/closing client window

  LOGOUT   username

<u>Server to Client</u>

- User login -- sends 0 if login successful, 1 if not

  LOGIN   number

- Send list of spreadsheets -- lists each spreadsheet name separated by a space

  LIST   spreadsheet_name   spreadsheet_name   …

- Joined spreadsheet session -- send 0 if joined successfully, 1 if not

  JOIN   number

- Open new or existing spreadsheet -- send xml file

  SPREADSHEET xml

- Send cell update status -- 0 = success, 1 = circular dependency, 2 = formula error

  STATUS   spreadsheet_name   number

- Send cell update -- sent individually to each connected client

  UPDATE   spreadsheet_name   cell_name   cell_contents

- Bad command received

  IGNORING   command

6 Handling Dropped Clients

When a client is dropped from the server their user name is taken off the list of active users. Any pending changes in the queue should still remain enqueued until they are dealt with. This ensures any save request or change not executed before the client dropped will still honored. The client's socket is then closed.

This implementation requires an auto-save feature designed to ensure changes are not lost if the last client is dropped. Every five minutes an xml file will be created of the current state of the spreadsheet. If the last active client is dropped, this xml file will be saved by the server as *~filename* and will act as a regular spreadsheet when being sent to a client.

7 Undo Functionality

As changes are propagated to each client, the change is also added to a stack on the server so that they can be “undone” based on the most recently completed. When ctrl+z or an undo button is pressed by a user, a packet is sent to the server requesting a change undo. The server then propagates this reverted change to each of the clients as it would a normal cell update. If the spreadsheet is saved by any user, the stack is cleared and changes prior to the save can not be undone.

8 Limitations

Although all efforts were made to account for possible scenarios there are still limitations with this protocol. It is important for users to realize that spreadsheets are public, and that anyone can access them. This was done in order to simplify the login process and allow more time for development. This implementation also requires the use of unique user names and a way to check an incoming login against a list of active users. This protocol assumes clients are using contemporary hardware, and no efforts were made to optimize use for embedded systems or legacy systems. We assume the clients have ample ability to handle 32 bit integers. It is also recommended that clients have sufficient bandwidth to the server. Although no obvious faults were apparent to us, clients using dial-up or slower connections will probably see considerable latency when using this protocol.

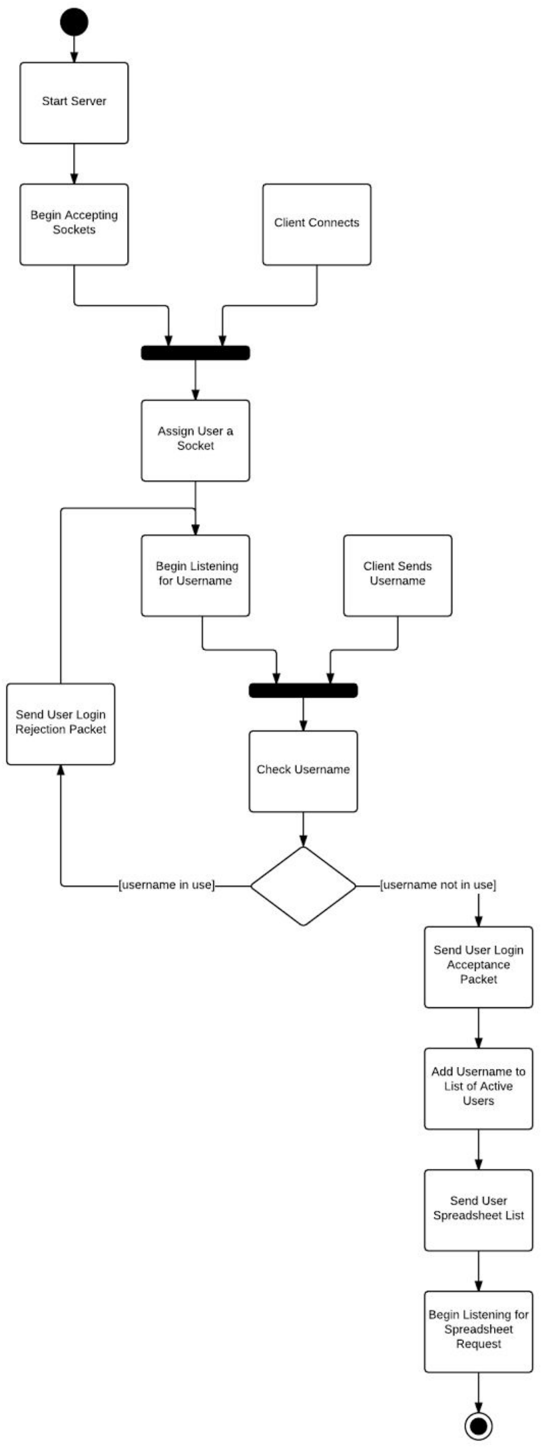
Start Server
Begin Accepting Sockets
Client Connects
Assign User a Socket
Begin Listening for Username
Client Sends Username
Send User Login Rejection Packet
Check Username
[username in use]
[username not in use]
Send User Login Acceptance Packet
Add Username to List of Active Users
Send User Spreadsheet List
Begin Listening for Spreadsheet Request

Figure 1: Client Connection Activity Diagram (server view)

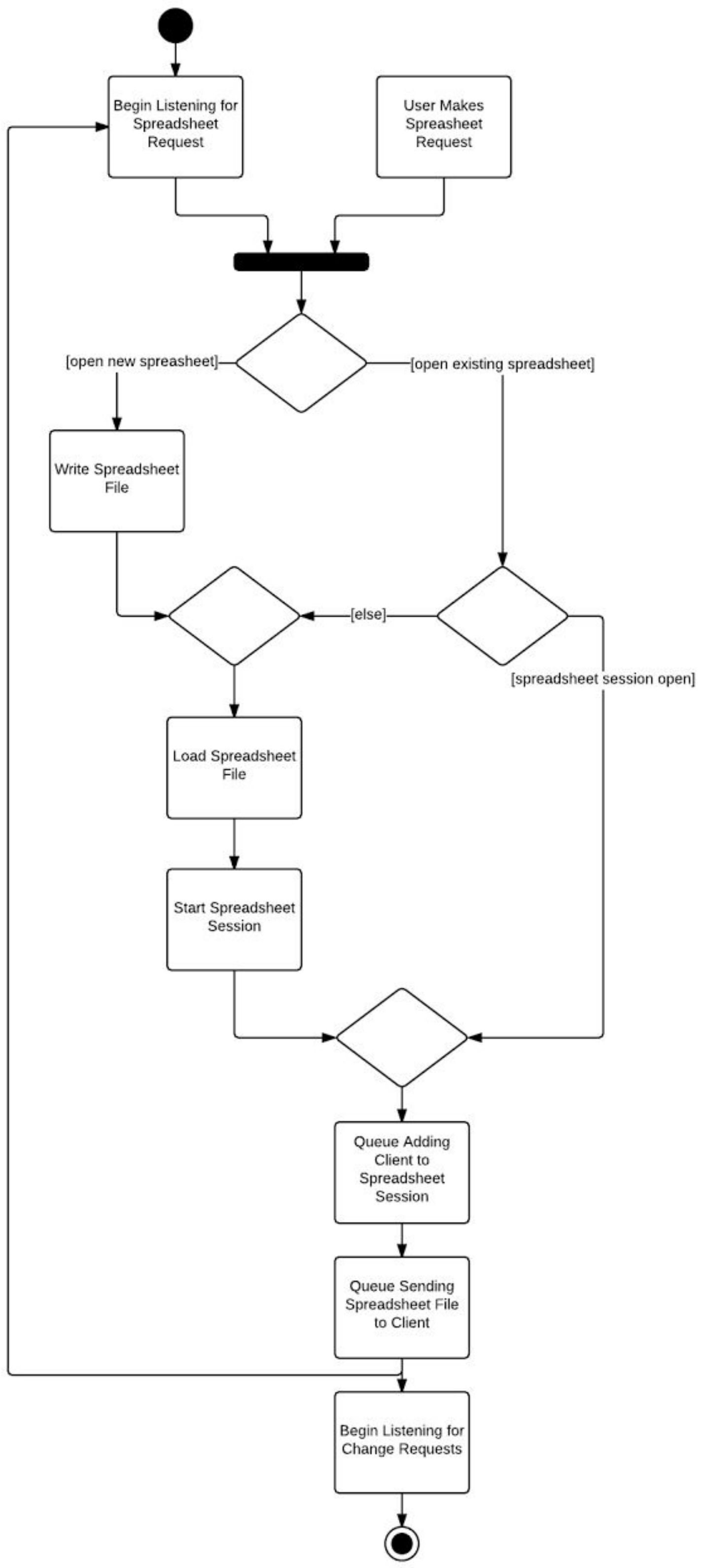
Begin Listening for Spreadsheet Request
User Makes Spreasheet Request
[open new spreasheet]
[open existing spreadsheet]
Write Spreadsheet File
[else]
[spreadsheet session open]
Load Spreadsheet File
Start Spreadsheet Session
Queue Adding Client to Spreadsheet Session
Queue Sending Spreadsheet File to Client
Begin Listening for Change Requests

Figure 2: Spreadsheet Request Activity Diagram (server view)

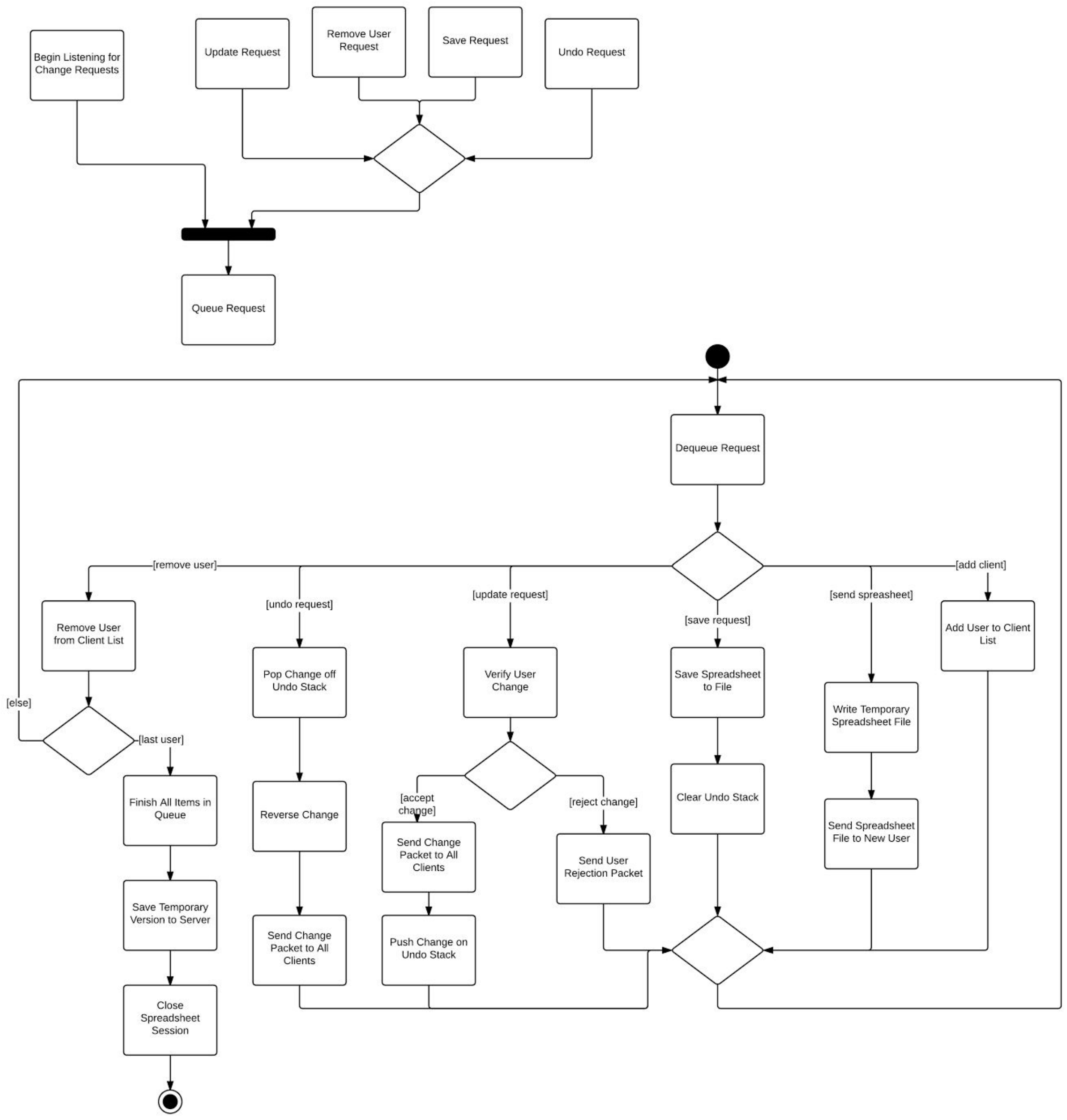
Begin Listening for Change Requests
Update Request
Remove User Request
Save Request
Undo Request
Queue Request
Dequeue Request
[remove user]
[undo request]
[update request]
[save request]
[send spreasheet]
[add client]
Remove User from Client List
Pop Change off Undo Stack
Verify User Change
Save Spreadsheet to File
Write Temporary Spreadsheet File
Add User to Client List
[else]
[last user]
Finish All Items in Queue
Reverse Change
[accept change]
[reject change]
Clear Undo Stack
Send Spreadsheet File to New User
Send Change Packet to All Clients
Send User Rejection Packet
Save Temporary Version to Server
Send Change Packet to All Clients
Push Change on Undo Stack
Close Spreadsheet Session

Figure 3: Change Request Activity Diagram (server view)

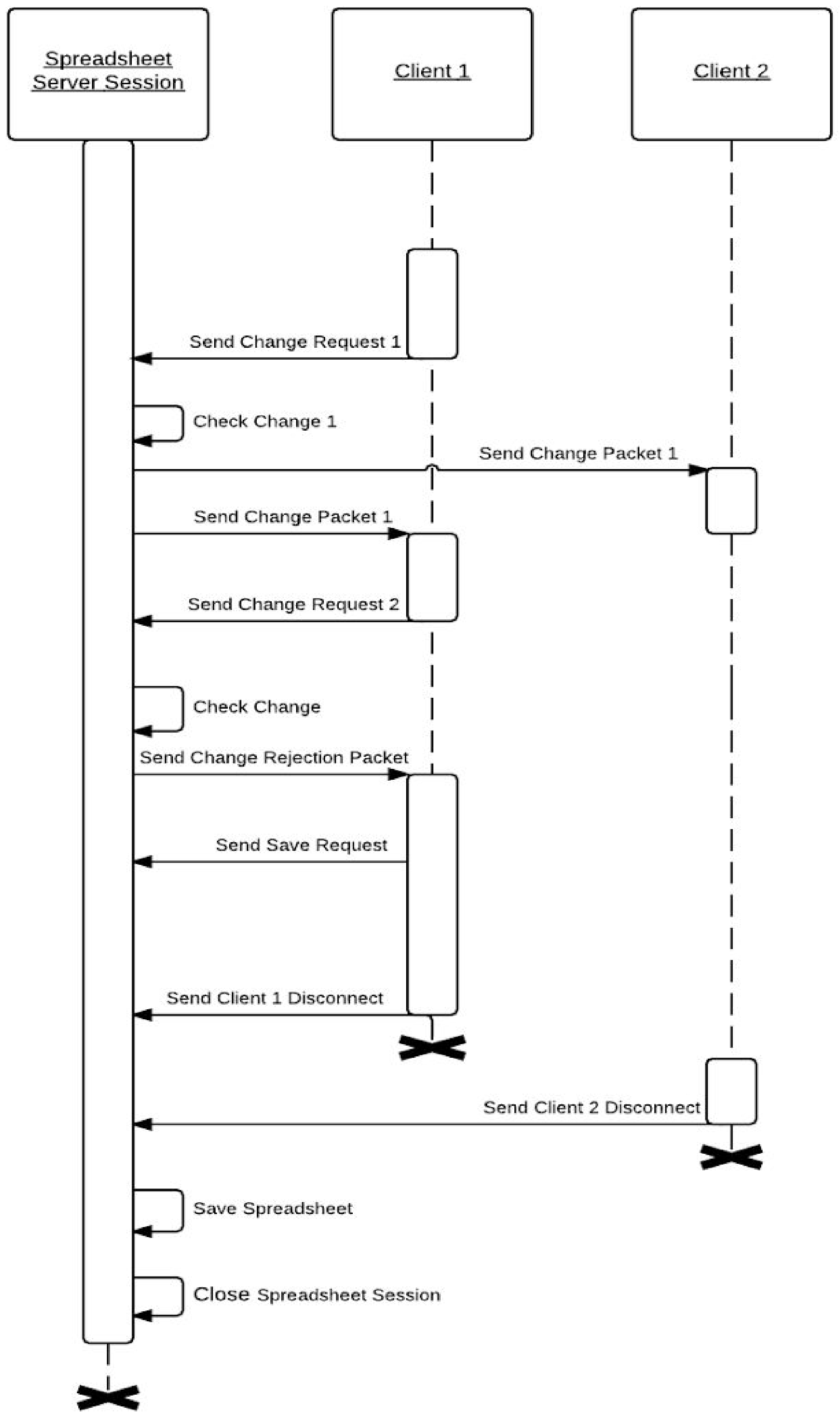

Figure 4: Multiple Change Request Sequence Diagram

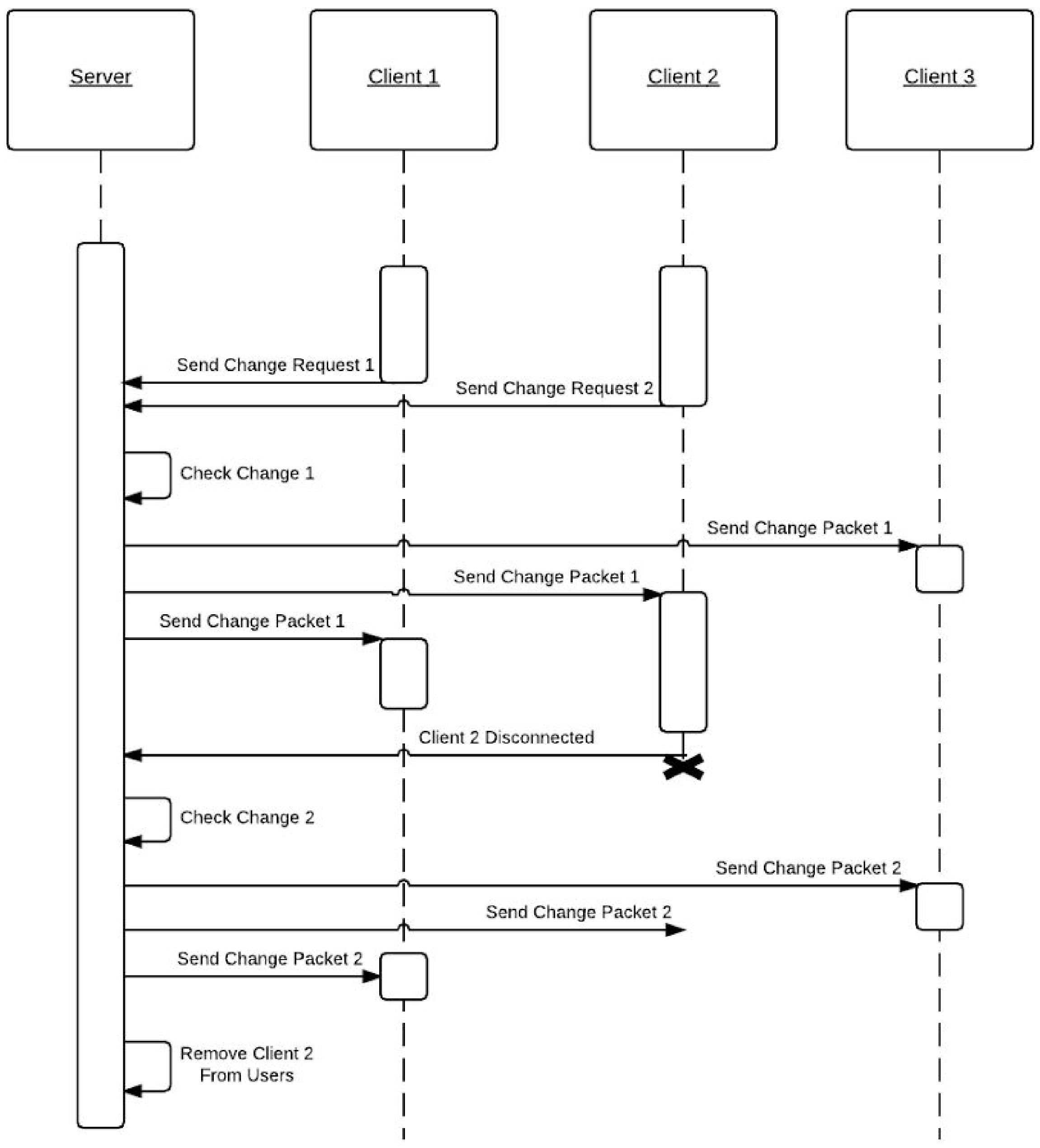

Figure 5: Change Request and Rejection Sequence Diagram

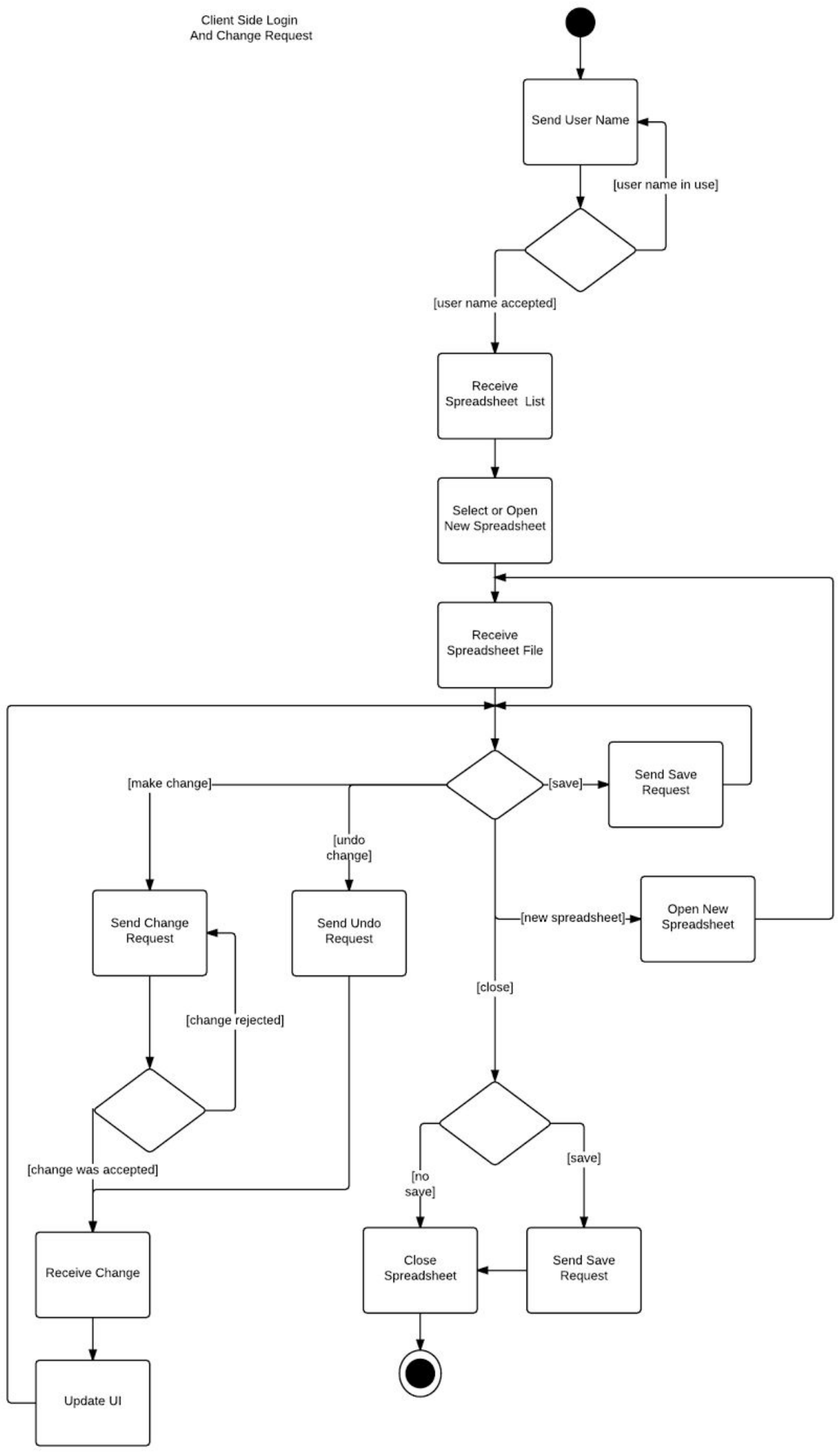

Figure 6: Client Side Communications