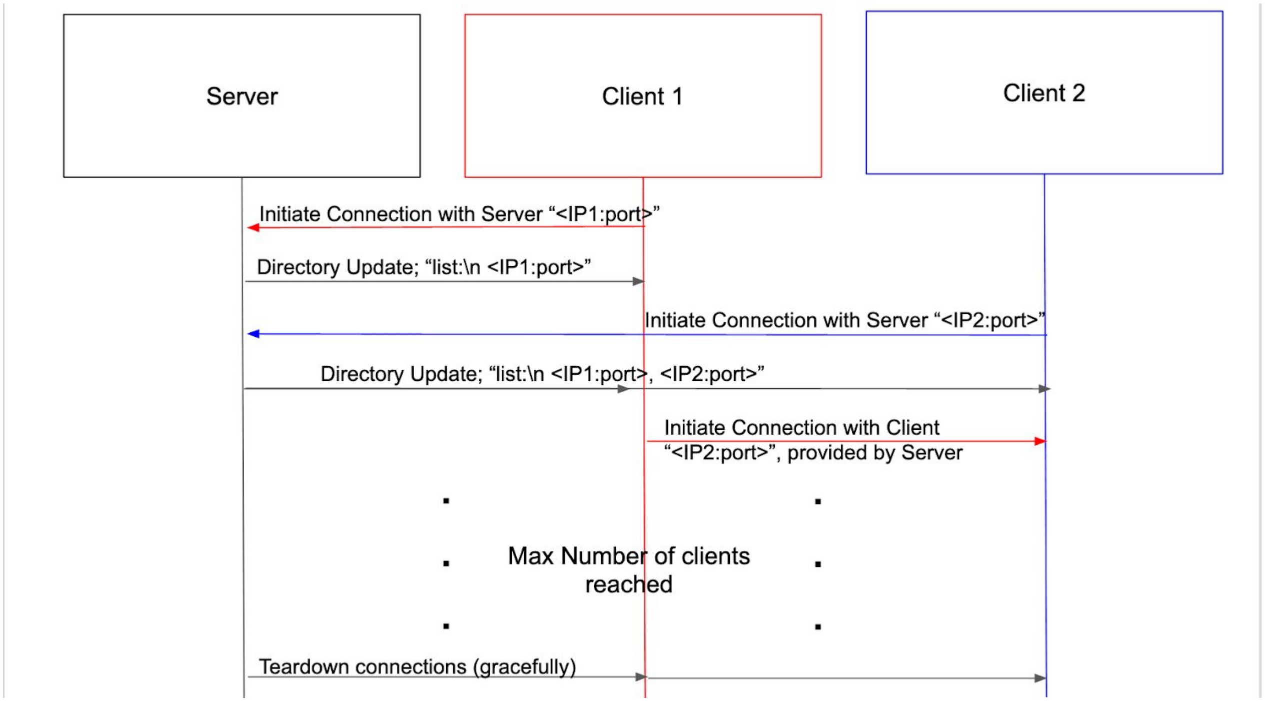
## Networking, Really

### Design

Server Startup & Directory Updates: Initially, the server is booted up and gets ready to receive incoming connections. Once a client reaches out and connects with the server, the server sends out a directory update to all connected clients that consists of a list of all connected client IPs and their respective ports in the order the nodes joined the server.

Establishing Client->Client Connections: The directory update sent by the server upon each new client's connection is received by each connected client. These clients then parse the directory update for the IP and port of all connected clients. In our implementation, each client is responsible for establishing a connection with the clients that joined the server after they did. This way, a connection between every pair of clients is guaranteed because as each new client joins all existing clients will reach out to the new client. The message sent between clients is in the vein of "Hello from IP:port"

Teardown: Once the number of clients connected to the server reaches the max number of acceptable clients, the server begins teardown by sending a teardown message to each client and closing connections


Server
Client 1
Client 2
Initiate Connection with Server "<IP1:port>"
Directory Update; "list:\n <IP1:port>"
Initiate Connection with Server "<IP2:port>"
Directory Update; "list:\n <IP1:port>, <IP2:port>"
Initiate Connection with Client "<IP2:port>", provided by Server
Max Number of clients reached
Teardown connections (gracefully)

*Server - Client Sequencing Diagram*

**Implementation**

**Server:**
`init_socket(ip, port):`
This method sets up a socket on which the server will listen for incoming connections, and binds that socket to its `ip:port` address. This socket is not in listening mode yet.

`sock_listen():`
This method initializes the socket to `listen()`. The listening socket is added to the set of active file descriptors with `FDSSET()`. The server is now ready to handle new connections from clients keep them posted on nodes in the network with directory updates whenever a new client registers.

`start():`
This method is probably the most important one. The server loops here while `num_clients` is less than `max_clients`. While this is true the server will `_wait_for_activity()`, a method that internally uses `select()` to watch for activity on any of the file descriptors in the `master fd_set` which is used to keep track of all active connections. When any new activity is detected on a file_descriptor `fd` in the `master fd_set`, the following routine kicks off based on the file descriptor on which the activity was detected using `FD_ISSET()`:

`if fd == listener:`

This implies that we have a new connection that needs to be handled. We first `accept()` this new connection, and then `_handle_client_connections()` `--` a subroutine that adds the file_descriptor for this new connection to the `master fd_set`

`else:`

Either

- the client shutdowns - in which case you close its file_descriptor, remove it from the `fd_set` with `FD_CLR()` and remove its ip-string from `client_addresses.`
- Or you're probably just receiving a registration message from the client. The registration message is just a string formatted as "`<ip>:<port>`" which gets stored in a list of ip-strings for all clients that have connected, called `client_addresses.`

**Node Startup:**
A node in the network first boots up creating a socket to talk with the rendezvous server, and a second socket that will listen for incoming connections from other nodes on the network.

The node's server-socket is initialized in the constructor of the node class and added to the `master fd_set` with FD_SET(). All other processesses are set up with the following exposed API methods:

`connect_to_server(serv_addr):`
This method simply initializes a connection to the server that's hosted at `serv_addr` using the POSIX networking API's `connect()` method and the socket created in the class constructor

`init_listenting_socket(ip, port):`
This method sets up a socket on which the client that will listen for incoming connections from other clients on the node. This second socket is integral for supporting direct communication between nodes in the network managed by the rendezvous sections.

`sock_listen():`
Same as server.

`register_with_server():`
Sends the registration message to the rendezvous server using the POSIX `send()` method. The registration message is a string formatted as `"<ip>:<port>"` referring to this nodes ip and port information.

`start():`
Main listening loop. This method functions in a similar manner to the server's `start()`, except for handling of data coming in from the server, which is presumably the `Directory-Update` from the server.
The following routine kicks off when the activity was detected on the file descriptor used to talk with the server:

- `if fd == server_sock:`
  - The three cases here are:
    - `Server_shutdown:`
      - Terminate program
    - `Teardown_msg:`
      - Terminate program
    - `Directory-Update:`
      - The directory update msg is a string of the form `"list:\n<client1-addr>\n<client2-addr>...<clientN-addr>"`
        The client will go through and find itself in the list. It will then create `sockets` and `connect()` to addresses that follow it in the list. If the `connect()`ion was successful, it adds the nodes address to the ip-string-list `client_addresses` and sends a Hello message formatted as the string `"Hello from:<ip:port>"` to each of the sockets.

        Before connecting it also checks if the client address is not already in the `client_addresses` list.

This cascading connection design where a client only connects to all clients below it in the list, ensures that there isn’t multiple connections between each pair of clients.