

## Team:

Jeremiah Blanchard  
Saurin Desai  
Prasad Saripalli

## Multiple Server-based Chat Protocol Specification

**Objective:** Extend project 2 to include multiple chat servers and implement distributed fault tolerance and server discovery mechanism. The deliverable is a complete and consistent protocol specification specifying message format, syntax and semantics as well as proper message sequencing and actions associated with each message received / sent by the nodes / server.

### Message Format

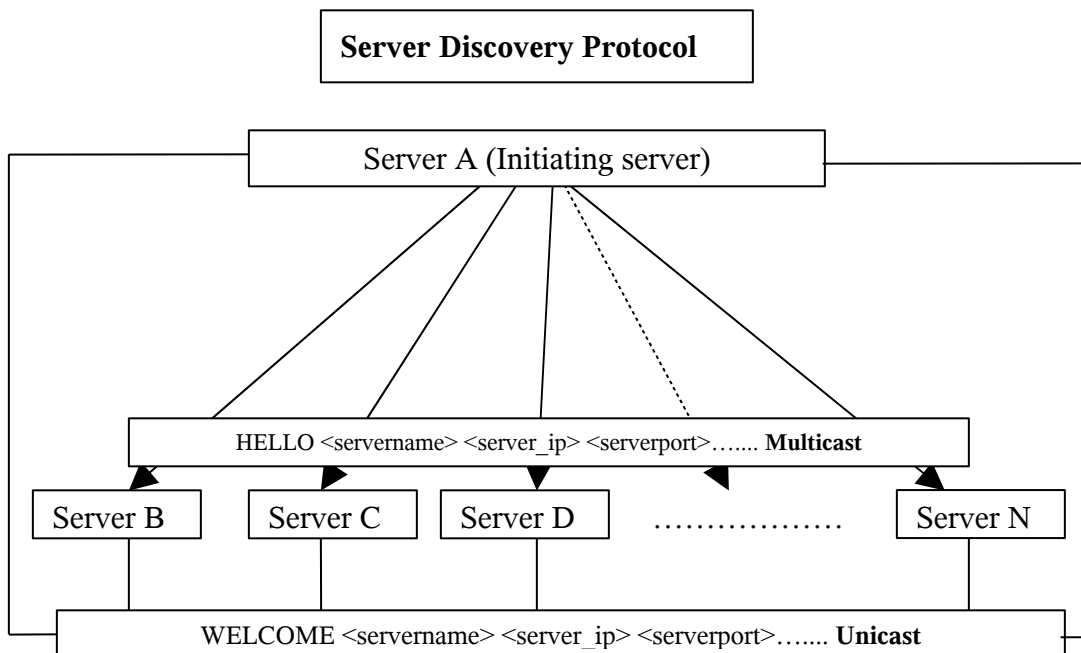
Each message is composed of two parts: the command and the argument list. All messages begin with '<' and end with '>'. The backslash character serves as an escape character so that any character following it is read as data and not a delimiter.

`<command, arg1, arg2, ... , argn>`

### Protocol for Server Discovery

This part defines the protocol used by servers to discover other servers. It includes a multicast search command (HELLO) and a unicast response command (WELCOME.) Servers are uniquely identified by their name and IP address.

**Concept:** Servers seeking peers will send out a multicast message indicating such. Servers willing to peer with such a server respond with a unicast message inviting the server to peer.



**Command:** HELLO

Parameters: <servername> <server\_ip> <serverport>

The HELLO command is used by a server to indicate that it is seeking peers. This message is sent via a predefined multicast address and port. After sending a HELLO message, the server should set a timer and wait for peers to respond to the hello message.

<servername> is the server's alias.

<server\_ip> is the server's IP address.

<serverport> is the server's peer listening port.

**Command:** WELCOME

Parameters: <servername> <server\_ip> <serverport>

The WELCOME command is used by a server to indicate that it is willing to peer with another server. This message is sent via unicast. This message does not establish a connection; it merely is a communication mechanism to establish a willingness to peer.

<servername> is the server's alias.

<server\_ip> is the server's IP address.

<serverport> is the server's peer listening port.

**Fault tolerance:** The nature of HELLO and WELCOME messages means they require no special ordering or response and are therefore fault tolerant by default.

### **Protocol for Inter-Server Communication**

This part defines the protocol used by servers to communicate with each other. It allows servers to connect to each other effectively forming a network. Servers are uniquely identified by their name and IP address.

**Establishing a server-server connection:**

The commands described here are used to register a connection with another server.

**Command:** SERVER

Parameters: <servername> <hopcount> <ID> <info>

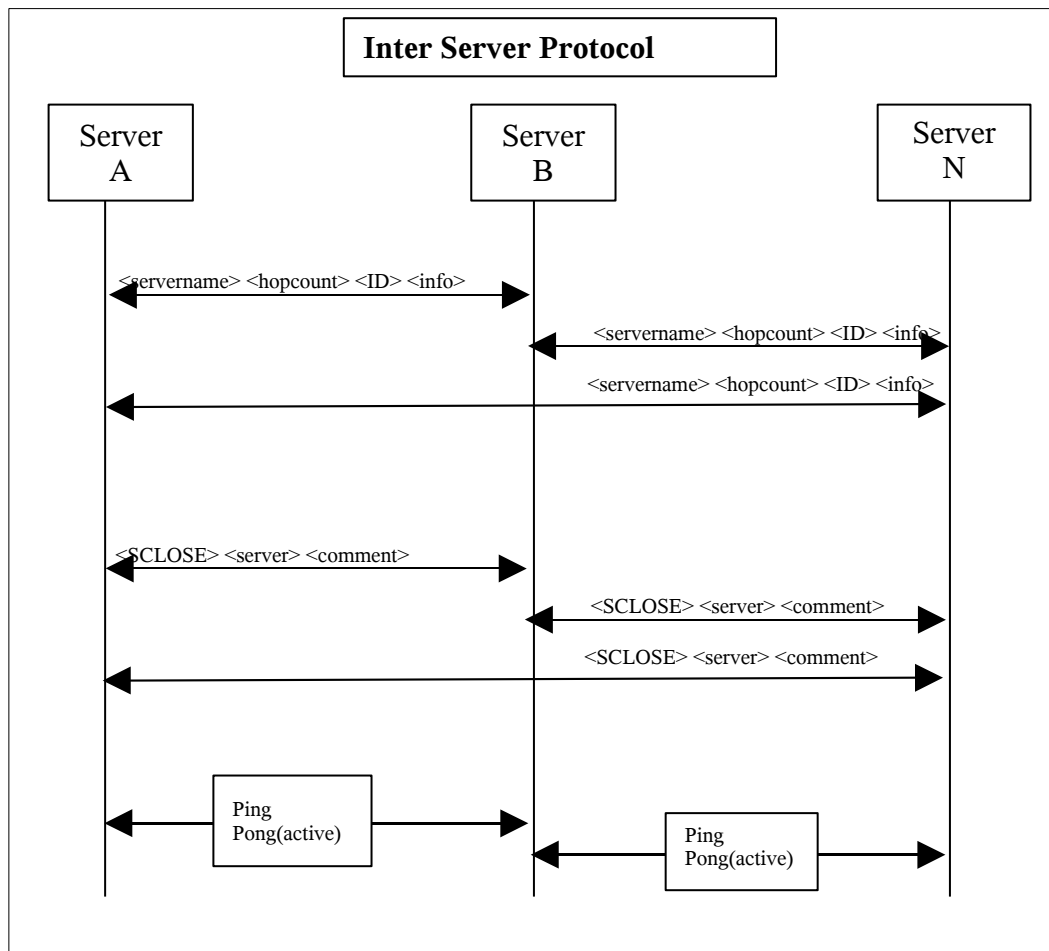
The SERVER command is used to register a new server. A new connection introduces itself as a server to its peer. This message is also used to pass server data over whole network. When a new server is connected to the network, information about it must be broadcasted to the whole network. The SERVER message should only be accepted from either a connection which is yet to be registered and is attempting to register as a server, or an existing connection to another server, in which case the SERVER message is introducing a new server behind that server. In this command:

The <info> parameter may contain space characters.

<hopcount> is used to give all servers some internal information on how far away each server is. Local peers have a value of 0, and each passed server increments the value. With a full server list, it would be possible to construct a map of the entire server tree.

The <ID> parameter is an unsigned number used by servers as an identifier. This identifier is subsequently used to reference a server in the messages sent between servers.

After a server has received a connection using a SERVER command, the server should reply with its own SERVER information for that connection as well as all of the other state information it knows about other neighboring servers, and the clients attached to each of those servers. By passing the state information about servers first, any collisions with servers that already exist occur before nickname collisions caused by a second server introducing a particular nickname.



**Terminating server-server connections:** Server connections are terminated using a server exit command specified as below:

Command: SCLOSE Parameters: <server> <comment>

The SCLOSE message allows operators to break a local or remote server link. It is a graceful way for a server to exit. If a server-server connection is closed, either via a SCLOSE command or other causes, the rest of the connected network must have its

information updated by the server which detected the closure. The terminating server then sends a list of SQUITs (one for each server behind that connection).

**Name Resolution:** If a SERVER message is parsed and it attempts to introduce a server which is already known to the receiving server, the connection, from which that message arrived, must be closed. The server should send an Already Registered error message. For a reasonable history, a server should be able to keep previous nickname for every client it knows about if they all decided to change. This size is limited by other factors (such as memory, etc).

### **Fault Tolerance:**

The following commands are used to detect and gracefully deal with faults. Servers should request a response from peers that have not been updated recently by either an incoming request or response. If no response is received from a peer, then the peer is assumed to be disconnected.

#### **Command:** PING

Parameters: <servername> <ID>

The PING command is used by a server to request a PONG message from a peer in order to determine whether a connection has been lost. When a server receives a PING command, it should first check to see if the requesting server is connected as a peer. If not, it should not respond. If it is, the receiving server should send a PONG message and should update the server's entry to indicate the communication time.

<servername> is the requesting server's hostname.

<ID> is the requesting server's alias.

#### **Command:** PONG

Parameters: <servername> <ID>

The PONG command is used by a server to indicate that it is a live peer. When a PONG message is received, the receiving server should update the peer's entry to indicate the communication time.

<servername> is the requesting server's hostname.

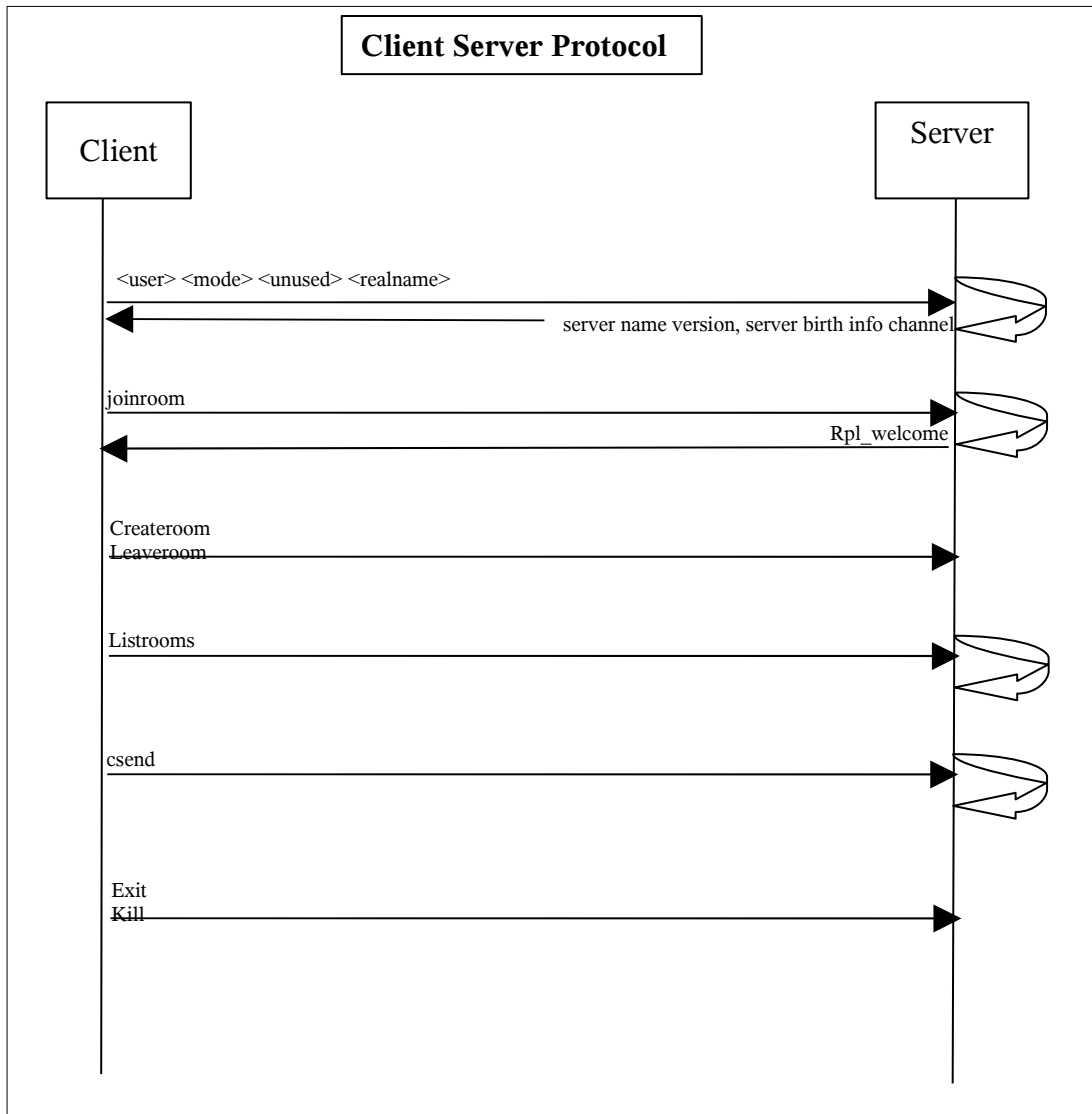
<ID> is the requesting server's alias.

### **Protocol for Client-Server Communication**

Clients: For each client, all servers will have the following information: a unique identifier and the server to which the client is connected. In addition, each user is distinguished from other users by a unique nickname. In addition to the nickname, all servers must have the following information about all users: the name of the host that the user is running on, the username of the user on that host, and the server to which the client is connected.

**Terminating server-client connections:** When a client connection unexpectedly closes, a QUIT message is generated on behalf of the client by the server to which the client

was connected. No other message is to be generated or used.



**Command:** EXIT Parameters: [<Exit Message>]

A client session ends with a quit message. The server must close the connection to a client which sends a EXIT message.

**Command:** USER

Parameters: <user> <mode> <unused> <realname>

The USER command is used at the beginning of connection to specify the username, hostname and real name of a new user.

The following commands are sent from Client to the server and are supported by all instances of the server.

**Command:** JOINROOM

Parameters: <chatroom>

JOINROOM allows particular user to join a chat room. This command must be broadcast to all servers so that each server knows where to find the users who are on the channel. Upon success, the client will receive an RPL\_WELCOME message indicating that the connection is now registered and known to the entire IRC network. The reply message must contain the full client identifier upon which it was registered.

**Command:** CREATEROOM

Parameters: <chatroom>

CREATEROOM creates a chat room and names that chat room.

**Command:** LEAVEROOM

Parameters: <chatroom>

LEAVEROOM allows the user to leave a chat room.

**Command:** GETSEQUENCE

Parameters: <chatroom>

GETSEQUENCE allows client to request a sequence number so that chat messages may be ordered. The server will respond with a NEWSEQUENCE message.

**Command:** ULIST

Parameters: <chatroom>

ULIST lists all the users that are in a chat room

**Command:** LISTROOMS

Parameters: none

LISTROOMS lists all existing chat room names.

**Command:** CSEND

Parameters: <message>

CSEND sends message to current chat room multicast address.

**Command:** QUIT

Parameters: none

A client session is terminated with a quit message. The server acknowledges this by sending an ERROR message to the client.

**Command:** KILL

Parameters: <nickname> <comment>

The KILL command is used to cause a client-server connection to be closed by the server which has the actual connection. Servers generate KILL messages on nickname collisions.

### **Accepting a client to server connection**

When a server successfully registers a new user connection, it sends to the user messages stating: the user identifiers upon which it was registered, the server name and version, the server birth information, available user and channel modes.

**Client Name Resolution:** This is accomplished by Tracking recently used nicknames and Tracking nickname changes. All servers should keep a history of recent nickname changes. This is important to allow the server to have a chance of keeping in touch of things when nick-change race conditions occur with commands manipulating them. The server is required to first check for the existence of the nickname, then check its history to see who that nick now belongs to. This reduces the chances of race conditions but they can still occur with the server ending up affecting the wrong client.

### **Client Locator**

To be able to exchange messages, two clients must be able to locate each other. Upon connecting to a server, a client registers using a label which is then used by other servers and clients to know where the client is located. Servers are responsible for keeping track of all the labels being used.

### **Message To A List**

The least efficient style of one-to-many conversation is through clients talking to a list of clients. Here the client gives a list of destinations to which the message is to be delivered and the server breaks it up and dispatches a separate copy of the message to each given destination.

### **Broadcast Message (One-to-all)**

The one-to-all type of message is better described as a broadcast message, sent to all clients or servers or both. On a large network of users and servers, a single message can result in a lot of traffic being sent over the network in an effort to reach all of the desired destinations.

### **Message Ordering**

The ordering of messages that clients send to a chatroom is established by the server in this case. The server receives GETSEQUENCE requests and responds with a NEWSEQUENCE message dictating the client's new sequence number. All messages will be displayed by clients in order of sequence number. If a sequence number is missing, the client may issue a FINDSEQUENCE message to the server which will in turn establish whether or not the client assigned to the sequence is still connected. If not, the server issues a SKIPSEQUENCE message to clients so they may skip that sequence number.