

RFC 99,999,999

**Interconnected Multi-server Chat System (IMCS) Protocol  
Protocol Specification Version 3 (Final)**

November 2004

Prepared for

CMSC491N Network Programming  
University of Maryland, Baltimore County  
1000 Hilltop Circle  
Baltimore, Maryland 21250

by

The students of CMSC491N  
University of Maryland, Baltimore County  
1000 Hilltop Circle  
Baltimore, Maryland 21250

## **1. Introduction**

The Interconnected Multi-server Chat System (ICMS) is intended to provide an extremely simple, text-based, very reliable chat system. ICMS is a network of servers connected to each other, built on TCP to provide a fail-safe method of communication for approved network users.

### **1.1 Motivation**

Chat systems have become an increasingly large part of life on the Internet. Some such chat systems lack reliability. For example, if a server that provides certain channels goes down for any reason then that channel is lost as well as the clients in that channel. Other chat systems provide a wide range of options, channels and tools that more times than not go unused by the user.

ICMS was designed to rectify these problem areas discovered in these chat systems. ICMS through its web of interconnected servers provides channel reliability over the reliable TCP (RFC 793). Additionally, ICMS was designed to be GUI free in order to provide a simple, clutter free environment for clients.

## 2. Philosophy

### 2.1 Elements of the ICMS

The ICMS consists of a web of servers with a socket connection to each of the other servers already connected to the system. There is no master server in this environment. Once the initial server is up then follow-on servers will connect to that server and all other servers in the network. Additionally, the ICMS will consist of clients or users that can connect to any of the servers in the ICMS.

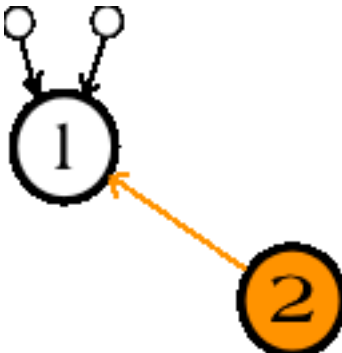
### 2.2 Use Cases



Server1 is up, tries to connect to other servers in network, none are up so it goes into a listening state.



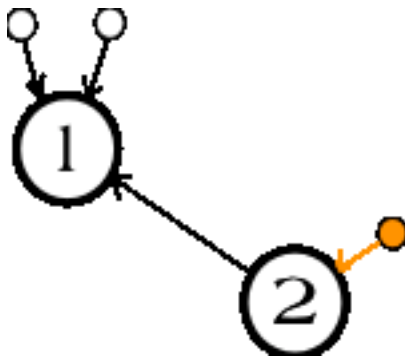
A couple of clients connect to Server1.



Server2 comes up, tries to connect to other servers in network, finds Server1 up and connects to it.

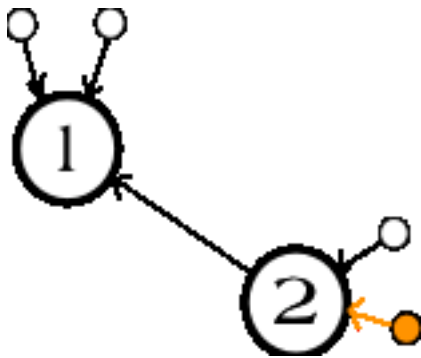
Server1 checks Server2 authorization.

Server2 synchronizes with Server1 (requests users and channels). Server1 sends list of users and channels to Server2.

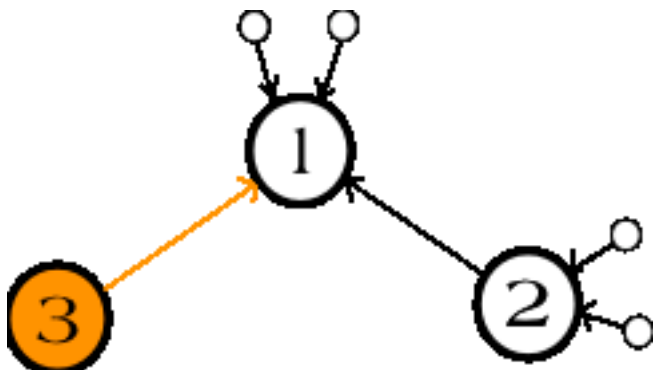


A client connects to Server2.

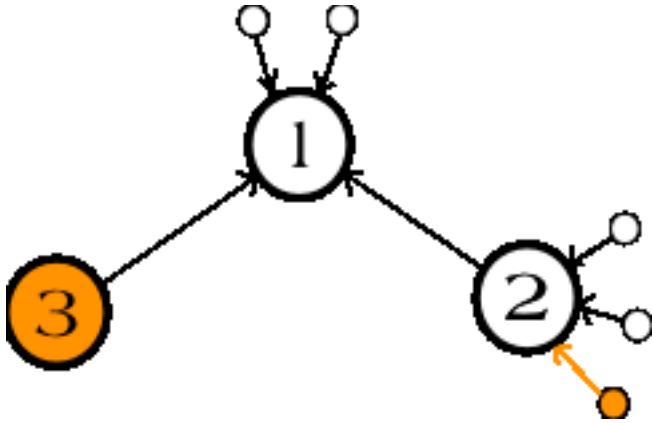
[Servers periodically resynchronize with each other after connection is established.]



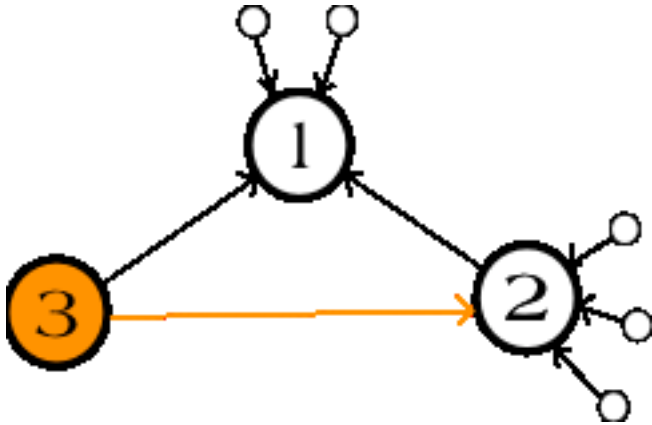
Another client connects to Server2.



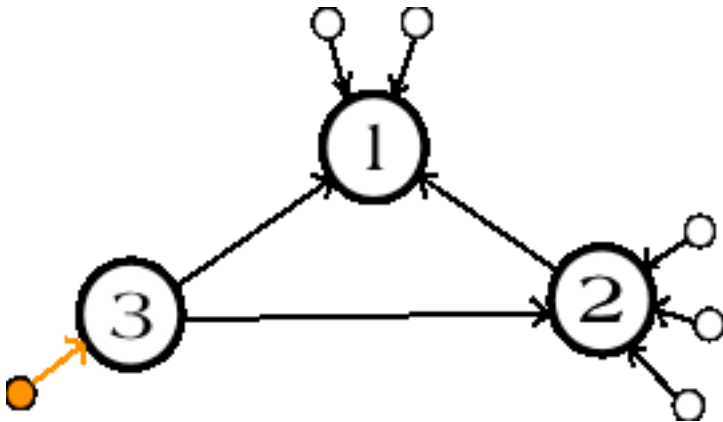
Server3 comes up, finds Server1 up and connects to it and synchronizes.



Before Server3 connects to Server2, a new client connects to Server2.

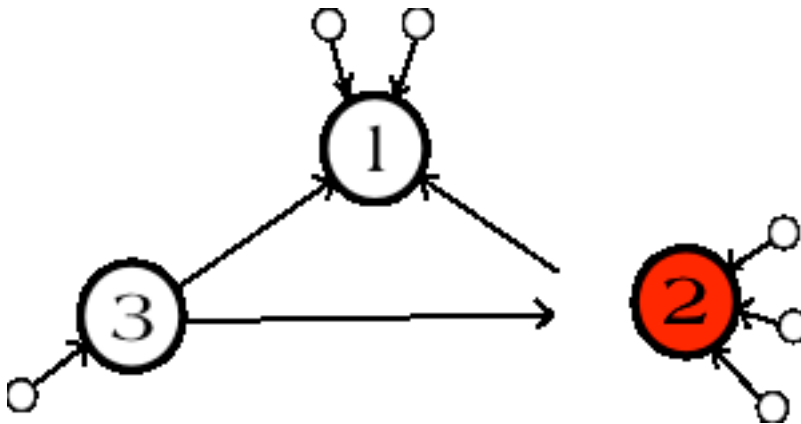


Server3 now connects to Server2.  
 [Servers periodically resynchronize with each other after connection is established.]

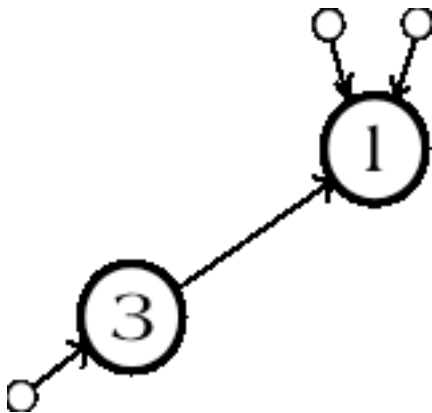


A client connects to Server3.

[Clients continue to connect, send messages, quit, start new channels, change nicknames, and leave channels.]



Server2 disconnects from network.



After deciding Server2 no longer exists, Server1 and Server3 need to resynchronize with each other, deleting users which were connected to Server2 from their list of users and any channels that no longer contain any clients.

### 3. Functional Specification

#### 3.1 Synopsis

This section outlines the commands available to both clients and servers as well as the responses to those commands. Additionally, this section provides the syntax for commands, channel names, and client names.

### **3.2 Maximum Capacity**

In the ICMS the total number of servers connected to the system at one time has been set to 10. Each server will maintain a maximum of 50 channels of its own as well as a maximum of 5 clients at one time. The limit of 5 clients has been imposed for testing purposes. Future releases will allow considerably more clients to connect to one server.

In total, the ICMS has the capability of supporting 10 servers, 50 channels and 300 clients.

### **3.2 Role of the Server**

Servers will communicate with other servers over TCP. Servers will maintain information for the following items:

1. List of Servers currently on the network
2. List of locally logged-in users
3. List of channels available on the network.
4. List of users connected to other servers in the network through periodic synchronization.

Note: Item 4, the list of other users, should not be relied upon as being accurate. When servers need up to the minute information about other servers on the network it should query the other servers for that data.

Communication will be done in plain-text ASCII mode over TCP. Additionally, lines of communication will begin with one of the commands specified in section **3.7 "Commands."** There are separate commands for client-to-server and server-to-server communication.

Servers will read from a configuration file that lists known servers to connect to. Servers will navigate the configuration file attempting to connect to each server in that list. If the server fails to connect to any server listed in the configuration file, it can be assumed that this server is the first one connected to the system and will be called the "initial" server. There is no master server. Once at least one server is in the system all other servers will have a connection point. From that point, servers will build their list of

information as they connect to all servers in the system at that time, i.e. users logged in, channels available.

Servers allow clients to connect to the system with a login (NICK) name (or nickname) of the client's choice. Server will verify that the chosen name is not currently in use and then respond to the client accordingly. Once the client has received verification allowing the chosen nickname, the client is considered logged in and can begin sending traffic. Servers will relay all incoming traffic from clients to all the other servers in the system. This traffic can consist of a multitude of requests that are outlined in section 3.7.

Servers periodically synchronize (LCHN/LUSR) their stored data with the other servers on the network. This is to continually maintain an accurate list of users currently logged in and a list of available channels. Additionally, servers will give warning to the other servers in the system when it plans to shut down. This will allow the system to purge channels and users as that server begins termination. However, if a server has not communicated with a client or another server in the last 60 seconds the server will send a heartbeat (PING) to that machine. If the server receives no response within 10 seconds, the server will purge (PRGE) its data and send a KILL command to all other servers.

### **3.3 Role of the Client**

Clients can connect to any known server in the ICMS. Upon connection clients will use the NICK command to send their desired user name for that session. At that point clients can use a range of commands to communicate in the channel, to the server or to other clients. Private chats are supported in the ICMS. So, communication can be done directly with other clients, but tunneled through the servers to which communicating clients are connected.

Clients will also receive traffic from servers and other clients. Traffic from servers will generally be responses to requests made by the client. Traffic from other clients will be considered to be messaging and not requests for state information. Requests of this manner will be made to servers only.

Clients will give warning of their leaving a channel or their leaving the ICMS all together. This allows for proper maintenance of the system. However, servers can ping clients to

test for availability. If there is now response clients can be removed.

All commands and communication details are outlined below in section 3.7.

### **3.4 Security**

There is security functionality, but it is implementation dependent. The functionality is comprised within the SERV and DENY commands outlined in section 3.7. The commands provide for communicating login/password information. Any further functionality is outside the scope of this protocol and is implementation-dependent. Implementation can be local or off-the-shelf.

### **3.5 Format**

The size of any message sent across the network will be the smaller of the maximum receive buffer or 1024 bytes.

All commands will be 4 characters in length and in upper case. Command will be followed by one or more parameters. The entire command line is space delimited. Example below:

```
JOIN <channel> <name>
```

### **3.6 Syntax**

The following rules concerning syntax pertain to user names and channel names.

Channel names will begin with the # symbol. They will be no more than a total of 32 characters in length including the beginning # symbol. Channel names can be alphanumeric and may consist of a combination of 0-9, a-z, A-Z, and the underscore.

User names or nicknames also will be alphanumeric and may consist of a combination of 0-9, a-z, A-Z and the underscore. Nicknames do not have a special starting symbol but they will also be no more than 32 characters in length.

### 3.7 Commands

The formats for some of these commands are slightly different than for the Server-Server commands. Optional arguments are given to simplify passing commands from Client-Server on to Server-Server.

Parameters listed in []'s are optional to send, but can be sent if desired.

Commands are listed in pairs where applicable and correspond to a command and its respective response or to command opposites.

Examples of command syntax are also given.

#### 3.7.1 Client-Server Commands

JOIN <channel> [<user>] - Enters <user> into <channel>

PART <channel> [<user>] - Exits <user> from <channel>

These two commands are issued from both C-S and S-S.

ex. JOIN #myChan

    PART #myChan

ex. JOIN #yourChan myName

    PART #yourChan myName

QUIT [<user>] - Causes a user to log off entirely, whereas the command is sent to other servers by the <user>'s local server to completely exit the user from the system.

ex. QUIT

ex. QUIT myName

LUSR [<channel> [<user>]] - Requests lists all of the users in a channel. It is a Client-Server command. Sends result back to user.

RUSR [<channel>] <user 1> ... <user N> - This returns the list of users, in <channel> if the argument is given. It is the response the server sends to the <user> that called LUSR.

ex. LUSR #myChan myName

    RUSR #myChan yourMom UglyBob CrazyLouann myName

ex. LUSR #myChan  
RUSR #myChan yourMom UglyBob CrzyLouann myName  
ex. LUSR  
RUSR yourMom UglyBob CrzyLouann myName uncleJoe myGirlFriend  
myGFsGirlFriend

LCHN [<user>] - Requests list of all channels, responds to  
<user>

RCHN <chan 1>...<chan K> - The response sent from Server-Client  
to the LCHN command, where client is <user>

ex. LCHN  
RCHN #myChan #yourMOMsChan #KeiChan  
ex. LCHN myName  
RCHN #myChan #yourMOMsChan #KeiChan

NICK <name> - this command is issued upon connection to register  
a nickname with the network. It can be responded to with NCLD  
or OKAY.

NICK <new name> <old name> - This is the command to change your  
nickname.

ex. NICK myName  
ex. NICK myName myOldName

NCLD <name> - Nickname collision. This response is sent from S->C,  
and/Or S->S, depending on the source of the collision.  
Additionally, it can be sent from servers that don't maintain a  
complete network user list at any time.

ex. NCLD myName  
ex. NICK myName  
NCLD

MESG <dest> <from> <text ... > - A message either to a channel,  
or to another user. Differentiated by channel's names starting  
w/ a '#'.  
'#channel'

ex. MESG #channel myName This is a message to a channel  
'#channel'

STAT [<user>] - Here <user> is requesting generic info from the server. What it returns is implementation dependent, but it would be most helpful to include IP:port, # users connected, # servers connected, # channels, ect.

RSTT <text .... > - This is the actual message sent from the Server-Client in response to 'STAT'

```
ex STAT
  RSTT my ip is 127.0.21.234:6666
ex STAT myName
  RSTT my ip is 12.123.12.31:6666
```

FULL <ip:port> - Server has too many users connected, so try to connect to <ip:port>

OKAY - Acknowledgement command

WTF0 - Response to malformed/screwed-up command. We could put as a parameter the actual command received that caused the error.

### 3.7.2 Server-Server Commands

JOIN <channel> [<user>] - Enters <user> into <channel>

PART <channel> [<user>] - Exits <user> from <channel>  
These two commands are issued from both C-S and S-S.

```
ex. JOIN #myChan
  PART #myChan
ex. JOIN #yourChan myName
  PART #yourChan myName
```

LUSR [<channel>] - This lists all of the users, in a channel if an argument is given. It is a synchronization command.

RUSR [<channel>] <user 1> ... <user N> - This returns the list of users, in <channel> if the argument is given.

```
ex. LUSR #myChan
  RUSR #myChan yourMom UglyBob myName
ex. LUSR
  RUSR yourMom UglyBob myName uncleJoe myGirlFriend
myGFsGirlFriend
```

LCHN - Lists all channels.

RCHN <chan 1>...<chan K> - The response sent back from LCHN.

ex. LCHN  
RCHN #myChan #yourMOMsChan

NICK <nickname> [<old name>] - Command to register a nickname, and logon. If <old name> is present, it's used to rename a user.

NCLD <name> - Nickname collision. This response/command is sent from Server-Server, depending on the source of the collision.

ex. NICK myName  
NCLD myName  
ex. NCLD myName  
ex. NICK myNewName myName  
NCLD myNewName

MESG <dest> <from> <text ... > - A message either to a channel, or to another user. Differentiated by channel's names starting w/ a '#'. This is passed on from client to server to server to client.

KILL <user/server> - This command is a Server-Server command. When a user issues a quit, its server will issue the KILL command to other servers to destroy all instances of <user> in their lists. In the case of servers, if a server is sending bad commands constantly, or won't respond to a PING, KILL is used.

ex. KILL myName

SBYE - server issues this to other servers to tell them that it's leaving.

PRGE - under certain instances, all tables will need to be reformed: When a server crashes, or when it connects. In the case of the crash, the first server to detect it sends this message.

SERV - This is the command issued first from a connecting server to other servers. In cases of authentication, the details can follow the command.

DENY - If authentication fails (if security is included) this command is sent to let the server attempting to connect know it's auth. failed.

ex. SERV

ex. SERV ServerAuthInfo Password

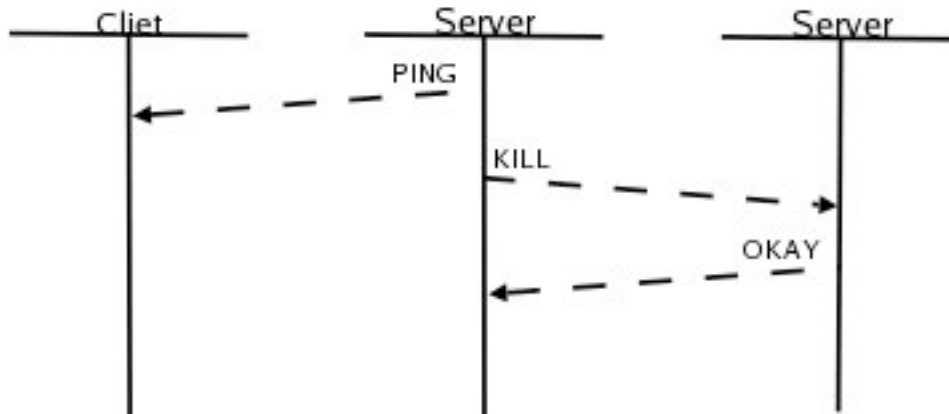
DENY Bad Server Name/ Bad Password

OKAY - Acknowledgement command.

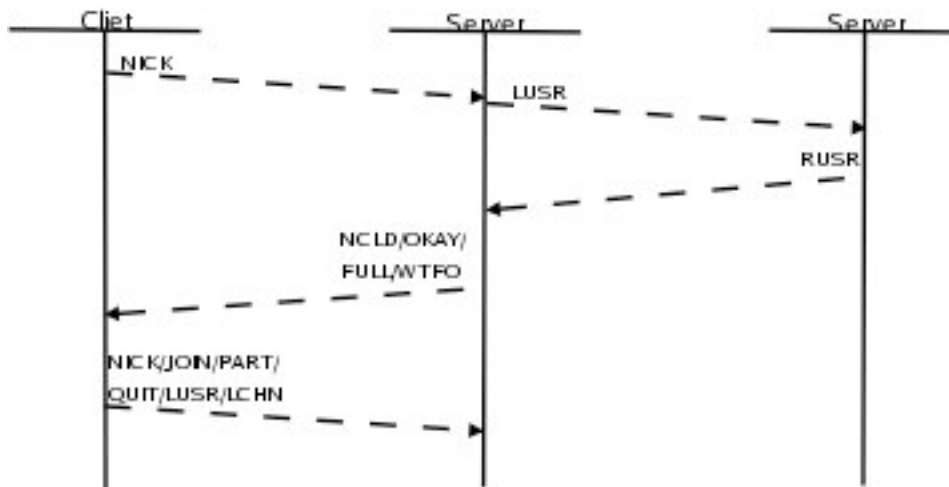
WTFO - basically a malformed/screwed-up command. We could put as a parameter the actual command received that caused the error.

#### 4. State Diagrams & Commands

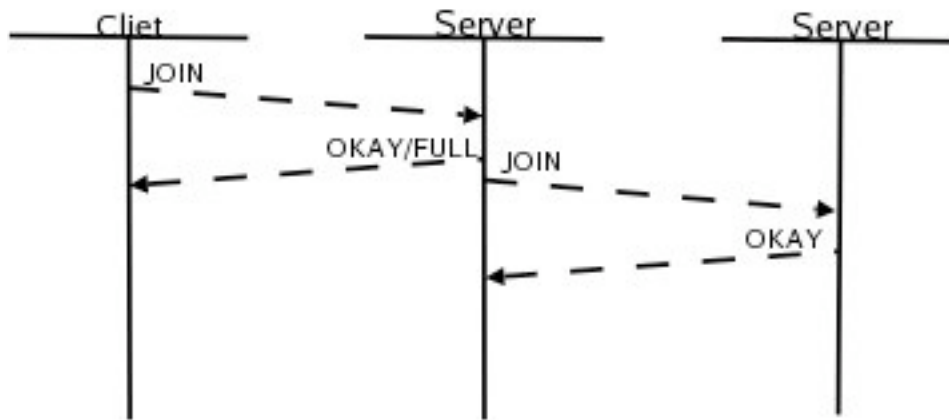
Packet Exchange for Client - Server PING  
With No Response



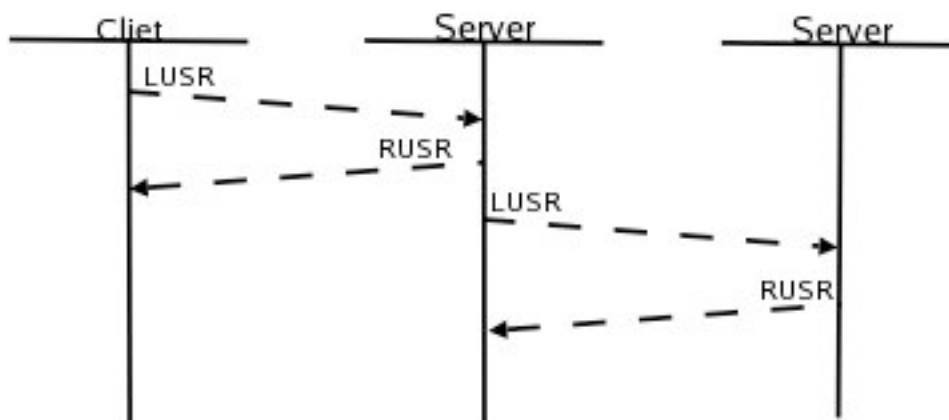
Packet Exchange for Client-Server Login  
Note: Process if server does not maintain a list of all users.



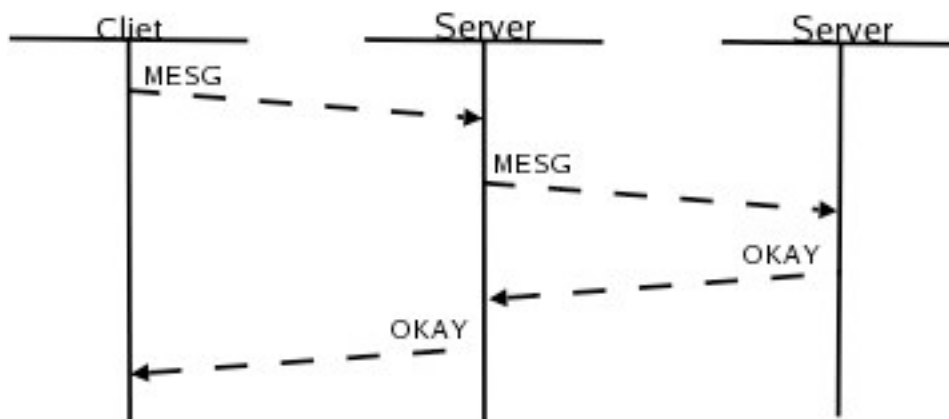
### Packet Exchange for Client - Server JOIN



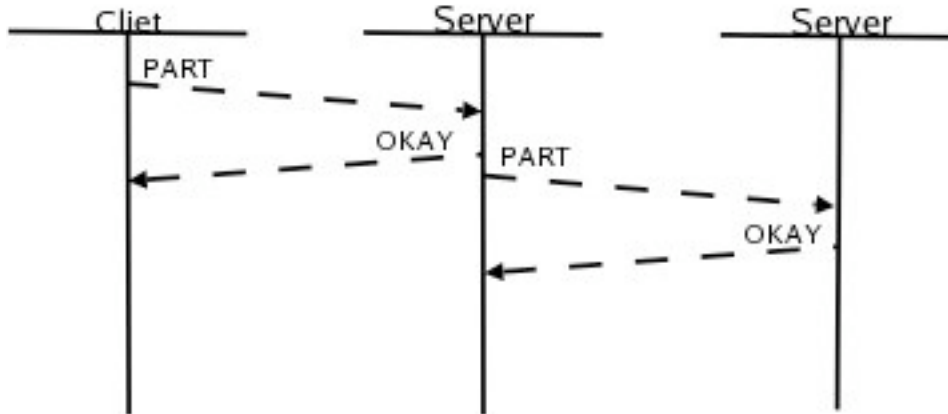
### Packet Exchange for Client - Server LUSR



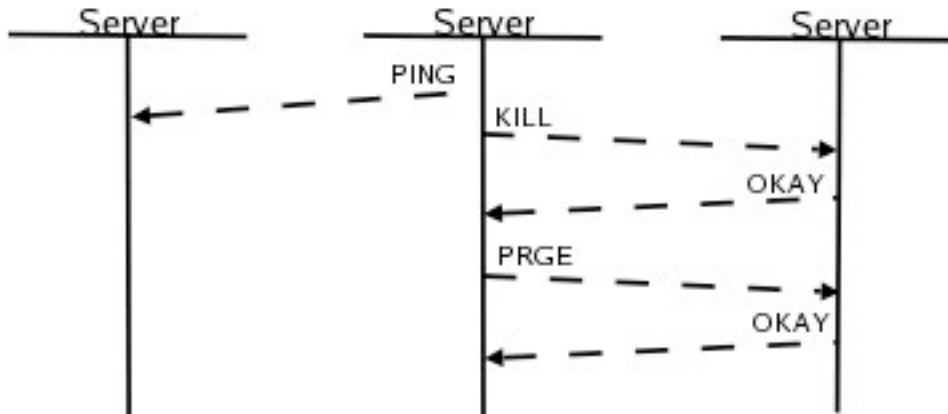
### Packet Exchange for Client - Server MESG



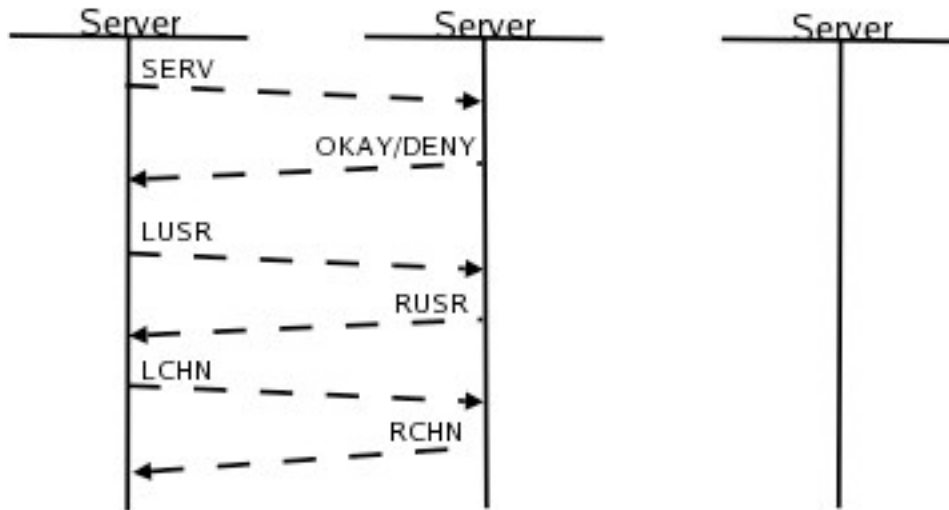
Packet Exchange for Client - Server PART



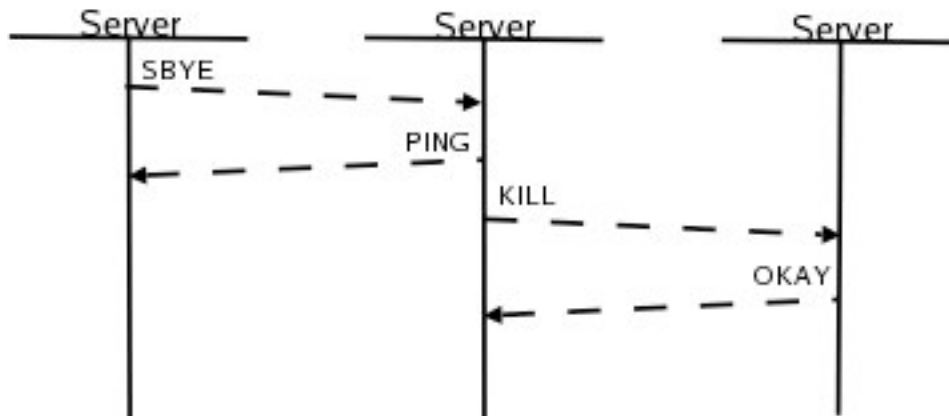
Packet Exchange for Server - Server PING With No Response



### Packet Exchange for Server - Server SERV



### Packet Exchange for Server - Server SBYE



### Packet Exchange for Client - Server STAT

