Project 3 Protocol Design v3.0
The following assumptions are made:

1. At launch servers are able to connect to a well known multicast address and port.
2. All messages sent, either multicast or unicast, are serialized XML objects.
3. UA’s aliases are appended with the parent’s name (i.e. A.a., A.b, etc.).

Command Line Arguments
1. Server: serverName mcIP mcPort [-recover]
2. UA: serverIP serverPort
3. User: userAgentIP userAgentPort

Server Discovery

Initialization
When a server first launches, it multicasts a Server Hello message in order to initiate the discovery process. If the initiating server receives at least one Server Hello NAK, from a pre-existing server, then there is a name collision. The initiating server must change its name through user input; afterwards the Server Hello message is re-multicast.

Once a pre-existing server receives a Server Hello message, it first checks if there is a name collision with itself and then with any of its neighbors, by checking its local neighbor list. If there is a name collision with itself or with one of its neighbors a pre-existing server sends a Server Hello NAK to the initiating server. Note that the initiating server may receive multiple NAKs for the same name collision. The sending of multiple NAKs increases the probability that the initiating server will receive at least one NAK, in case of a lost message or link failure.

If there is no name collision, then a pre-existing server must determine if a connection offer is to be made or not. The probability of making an offer is 1 when a pre-existing server has yet to meet its minimum number of connections. As connections are made, this probability decreases and approaches but never reaches 0 when its maximum number of connections is reached. When a pre-existing server determines that a connection is to be made, a Server Hello ACK is sent to the initiated server. Note that this load balance formula will only work if the minimum number of connections is 2 or more and the maximum number of connections is always two more than the minimum number of connections (i.e. Min Num = N, Max Num = N+2).

<table>
<thead>
<tr>
<th>Message Name</th>
<th>Medium</th>
<th>Channel</th>
<th>Message Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Hello</td>
<td>Multicast</td>
<td>Server to Server</td>
<td>[Server Name] [Listening unicast address] [Listening unicast port]</td>
</tr>
<tr>
<td>Server Hello ACK</td>
<td>Unicast</td>
<td>Server to Server</td>
<td>[Server Name] [Listening unicast address] [Listening unicast port]</td>
</tr>
<tr>
<td>Server Hello NAK</td>
<td>Unicast</td>
<td>Server to Server</td>
<td>[Server Name that caused the collision]</td>
</tr>
</tbody>
</table>

Neighbor Management

When the initiating server receives a Server Hello ACK message it sends a Neighbor Connect message to the pre-existing servers. As soon as a pre-existing server receives the Neighbor Connect message it adds the initiating server to its neighbor list and replies with a Neighbor Connect ACK message. After the initiating server receives a Neighbor Connect ACK the initiating server adds the pre-existing server to its neighbor list.

After a successful connection to its neighbors the servers will backup their neighbor list and UA list to a non-volatile storage.

In the rare event that the server receives a valid but delayed Server Hello NAK message, the server will force a migration upon its UAs to one of its neighbors. The server will then shutdown.

<table>
<thead>
<tr>
<th>Message Name</th>
<th>Medium</th>
<th>Channel</th>
<th>Message Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neighbor Connect</td>
<td>Unicast</td>
<td>Server to Server</td>
<td>[Server Name] [Listening unicast IP] [Listening unicast port]</td>
</tr>
<tr>
<td>Neighbor Connect ACK</td>
<td>Unicast</td>
<td>Server to Server</td>
<td>[Server Name] [Listening unicast IP] [Listening unicast port]</td>
</tr>
</tbody>
</table>
User Agent Management

Servers ensure that their UA's maintain unique names by receiving a Set Alias message from the UA. The server verifies that the name is unique by comparing that name to its local UA list. If the name is unique the server sends a Set Alias ACK message, otherwise the server sends a Set Alias NAK message.

A UA should not send any commands until its alias has been set by the user. Additionally, if a UA is in a chatroom the user should not be able to change its alias; this check will be accomplished on the UA and not by the server.

<table>
<thead>
<tr>
<th>Message Name</th>
<th>Medium</th>
<th>Channel</th>
<th>Message Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Alias</td>
<td>Unicast</td>
<td>UA to Server</td>
<td>[Alias Name]</td>
</tr>
<tr>
<td>Set Alias ACK</td>
<td>Unicast</td>
<td>Server to UA</td>
<td>[Alias Name]</td>
</tr>
<tr>
<td>Set Alias NAK</td>
<td>Unicast</td>
<td>Server to UA</td>
<td>[Alias Name] [Message]</td>
</tr>
</tbody>
</table>

Failure Recovery

Detecting Failures

A server should periodically send a Heartbeat message to its neighbors. Every time a server receives a Heartbeat message it will update that neighbor's timestamp. The server will check its own local neighbor list every N seconds, if an entry hasn't been updated in the last M checks that server is assumed to be dead, then it removes the server from the local neighbor list and notifies its UAs by sending an updated Backup Server List message.

In the case that the server removes its dead neighbor and the server is now orphaned the server will reinitiate the server discovery protocol.

A UA should also periodically ping its server to determine if it is alive by sending a Server Ping message. If the UA doesn't receive a certain number of Server Pong messages in the allotted time it determines that its server is dead and switches over to one of its backup servers.

The server tracks the last time it received pings from its UAs. If the server does not receive a ping from its UA in the allotted time the server assumes that the UA is dead and removes it from its UA list.

<table>
<thead>
<tr>
<th>Message Name</th>
<th>Medium</th>
<th>Channel</th>
<th>Message Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heartbeat</td>
<td>Unicast</td>
<td>Server to Server</td>
<td>[Server Name]</td>
</tr>
<tr>
<td>Server Ping</td>
<td>Unicast</td>
<td>UA to Server</td>
<td>[UA Name]</td>
</tr>
<tr>
<td>Server Pong</td>
<td>Unicast</td>
<td>Server to UA</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Switching Servers

When a server initially connects to its neighbors or when its neighbor's list is updated it sends an updated Backup Server List message to each of its connected UAs.

If a UA detects that its server has failed the UA sends a Migrate message to the first backup server in its list. The backup server then responds with a Migrate ACK upon receipt of a Migrate message. If the UA does not receive a Migrate ACK message after an allotted time the UA attempts to connect to the next backup server in its list. When the UA receives the first Migrate ACK message it ignores any following Migrate ACK messages. If a UA is not able to connect to any of its back up server the UA must die. Once a UA has migrated, their backup server list is updated by their new parent. Note that all UAs attempt to migrate to the same backup server. Therefore, chatroom services are not interrupted as long as all the UAs migrate to the same server. In the extreme case of chatroom fragmentation (i.e. UAs belonging in the same chatroom but connected to different servers) user transparency is shattered.

When the failed server recovers it reads its neighbor list and UA list from the non-volatile storage, if it exists. The server then reconnects to its neighbors, by issuing a Neighbor Connect message and in the case that the server is unable to connect to any of its neighbors the server will reinitiate the server discovery protocol. After connecting to at least one of its neighbors the server sends a Reconnect message directly to each one of its UAs in its list. When a UA
receives a Reconnect message it sends a Migrate message to its resurrected parent, awaits a Migrate ACK and then sends Reconnect ACK message informing its original parent about its current parent. When the original parent receives the Reconnect ACK message the parent sends a Recovered UA message to the UA’s former parent. Upon receipt of a Recovered UA message the former parent removes the UA from its local list.

<table>
<thead>
<tr>
<th>Message Name</th>
<th>Medium</th>
<th>Channel</th>
<th>Message Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backup Server List</td>
<td>Unicast</td>
<td>Server to UA</td>
<td>[Backup Server Collection]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The backup server collection is composed of: [Neighbor Name] [Listening unicast IP] [Listening unicast port]</td>
</tr>
<tr>
<td>Migrate</td>
<td>Unicast</td>
<td>UA to Server</td>
<td>The migration info is composed of: [Server Name] [Chat room name] [Chat room multicast IP] [Chat room multicast port] [Chat room seed] [UA Name] [UA listening unicast IP] [UA listening unicast port]</td>
</tr>
<tr>
<td>Migrate ACK</td>
<td>Unicast</td>
<td>Server to UA</td>
<td>N/A</td>
</tr>
<tr>
<td>Reconnect</td>
<td>Unicast</td>
<td>Server to UA</td>
<td>[Server Name] [Listening unicast IP] [Listening unicast port]</td>
</tr>
<tr>
<td>Reconnect ACK</td>
<td>Unicast</td>
<td>UA to Server</td>
<td>[Current Server Name] [Current Server Listening unicast IP] [Current Server Listening unicast port] [Migration Info]</td>
</tr>
<tr>
<td>Recovered UA</td>
<td>Unicast</td>
<td>Server to Server</td>
<td>[UA Name]</td>
</tr>
</tbody>
</table>

### Querying for Users

When a user initiates a user search request, the User sends a Search message to its UA. The UA forwards this message to its server, appending its listening unicast IP and port. The server first checks its own UA list for the desired user, and if found, returns a User Found message to the requesting UA. Regardless if a user is found locally or not, the server then sends a Search message to its neighbors. Therefore, the requesting user may receive delayed responses depending on the size of the network.

Each server will maintain two lists of searches. The first list maintains searches that originated from the server itself and is awaiting a response from its neighbors. The list also contains the information of the UA that requested the search. The second list maintains searches that originated from neighboring servers and are awaiting a response from the remaining neighboring servers.

To prevent runaway searches each server checks the unique ID of the search against its local search lists. If a server receives a duplicate search request, it checks for the desired user locally and if a match is found a Search Result ACK message is sent back to the requesting server; otherwise a Search Result NAK message is sent back. Note, that in either case the search is not propagated to its other neighbors.

If a Search message reaches the end of the network (i.e. a server with only one neighbor) and the server does not contain the desired user, the server then sends a Search Result NAK message to the neighboring server that requested the search.

Now once a match is found, a Search Result ACK message is sent back to the requesting server. When this message reaches the parent of the requesting UA, a User Found message is sent to the UA.

Additionally, each server will have a timeout period of N, after which the server will respond to its awaiting neighbors with a Search Result NAK, if no users were found.
XML Message Format

Using a XML format to represent the message structure decouples the need to specify data types for the messages. In order to sustain interoperability of different implementations all the teams must use the same XML message formats when sending messages. Each message is sent as a string that contains the XML structure. Teams can either create their own XML parsers or use the following package [http://xstream.codehaus.org/](http://xstream.codehaus.org/) to serialize/de-serialize their objects to XML and back again.

User Messages

Alias – Used to defined named alias (Project 1)

```xml
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>UserAlias</Command>
  <Data>
    <Alias>
      <AliasName>sand</AliasName>
      <IP>sand.cise.ufl.edu</IP>
      <Port>30000</Port>
    </Alias>
  </Data>
</Message>
```

Create Room – Creates a chatroom and names that chatroom (Project 2)

```xml
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>UserCreateRoom</Command>
  <Data>
    <Room>
      <Alias>
        <AliasName>COP5615</AliasName>
      </Alias>
    </Room>
  </Data>
</Message>
```

Join Room – Allows user to join a chatroom (Project 2)

```xml
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>UserJoinRoom</Command>
  <Data>
    <Room>
      <Alias>
        <AliasName>COP5615</AliasName>
      </Alias>
    </Room>
  </Data>
</Message>
```
Leave Room – Allows user to leave a chatroom (Project 2)
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>UserLeaveRoom</Command>
  <Data>
    <Room>
      <Alias>
        <AliasName>COP5615</AliasName>
      </Alias>
    </Room>
  </Data>
</Message>

List Users – (Formerly ‘ulist’) Lists all the users that are in the chatroom (Project 2)
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>UserListUsers</Command>
  <Data>
    <Room>
      <Alias>
        <AliasName>COP5615</AliasName>
      </Alias>
    </Room>
  </Data>
</Message>

Set Alias – Used to set one’s alias name (Project 1)
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>UserSetAlias</Command>
  <Data>
    <Alias>
      <AliasName>sandy</AliasName>
    </Alias>
  </Data>
</Message>

Remove Alias – Used to remove an existing alias (Project 1)
<Message>
  <Sender>
<Alias>aliasName</Alias>
<IP>127.0.0.1</IP>
<Port>30000</Port>
</Sender>
<Command>UserRemoveAlias</Command>
<Data>
  <Alias>
    <AliasName>sandy</AliasName>
  </Alias>
</Data>
</Message>

**CSend – Sends a message to current chatroom multicast address (Project 2)**

<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>UserCSend</Command>
  <Data>
    <ChatMessage>Hello</ChatMessage>
  </Data>
</Message>

**Search – Refer to Project 3**

<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>UserSearch</Command>
  <Data>
    <Alias>
      <AliasName>sandy</AliasName>
    </Alias>
  </Data>
</Message>

**List Rooms – List all existing chatroom names (Project 2)**

<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>UserListRooms</Command>
  <Data/>
</Message>

**Exit – Used to terminate the system (Project 1)**

<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
</Message>
List – List all the current existing aliases (Project 1)
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>UserList</Command>
  <Data/>
</Message>

Send – Used to send a message to named alias (Project 1)
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>UserSend</Command>
  <Data>
    <Alias>
      <AliasName>sandy</AliasName>
    </Alias>
    <ChatMessage>Hi</ChatMessage>
  </Data>
</Message>

User Agent Messages
Send – Used to send a message to named alias (Project 1)
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>UASend</Command>
  <Data>
    <ChatMessage>Hi</ChatMessage>
  </Data>
</Message>

CSend – Sends a message to current chatroom multicast address (Project 2)
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>UACSend</Command>
  <Data>
    <Seed>seed</Seed>
  </Data>
</Message>
<ChatMessage>Hi</ChatMessage>
</Data>
</Message>

**List Rooms – List all existing chatroom names (Project 2)**

<Message>
 <Sender>
  <Alias>aliasName</Alias>
  <IP>127.0.0.1</IP>
  <Port>30000</Port>
 </Sender>
 <Command>UAListRooms</Command>
 <Data/>
</Message>

**Create Room – Creates a chatroom and names that chatroom (Project 2)**

<Message>
 <Sender>
  <Alias>aliasName</Alias>
  <IP>127.0.0.1</IP>
  <Port>30000</Port>
 </Sender>
 <Command>UACreateRoom</Command>
 <Data>
  <Room>
   <Alias>
    <AliasName>COP5615</AliasName>
   </Alias>
  </Room>
 </Data>
</Message>

**Join Room – Allows user to join a chatroom (Project 2)**

<Message>
 <Sender>
  <Alias>aliasName</Alias>
  <IP>127.0.0.1</IP>
  <Port>30000</Port>
 </Sender>
 <Command>UAJoinRoom</Command>
 <Data>
  <Room>
   <Alias>
    <AliasName>COP5615</AliasName>
   </Alias>
  </Room>
 </Data>
</Message>

**Leave Room – Allows user to leave a chatroom (Project 2)**

<Message>
 <Sender>
  <Alias>aliasName</Alias>
  <IP>127.0.0.1</IP>
  <Port>30000</Port>
 </Sender>
<Command>UALeaveRoom</Command>
<Data>
  <Room>
    <Alias>
      <AliasName>COP5615</AliasName>
    </Alias>
  </Room>
</Data>
</Message>

Search – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>UASearch</Command>
  <Data>
    <Alias>
      <AliasName>sandy</AliasName>
    </Alias>
  </Data>
</Message>

List Users – (Formerly ‘ulist’) Lists all the users that are in the chatroom (Project 2)
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>UAListUsers</Command>
  <Data>
    <Room>
      <Alias>
        <AliasName>COP5615</AliasName>
      </Alias>
    </Room>
  </Data>
</Message>

Server Ping – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>UAServerPing</Command>
  <Data/>
</Message>

Migrate – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
  </Sender>
  <Command>UAMigrate</Command>
  <Data>
    <Room>
      <Alias>
        <AliasName>COP5615</AliasName>
      </Alias>
    </Room>
  </Data>
</Message>
<IP>127.0.0.1</IP>
<Port>30000</Port>
</Sender>
<Command>UAMigrate</Command>
<Data>
<MigrationInfo>
<Server>
<AliasName>server</AliasName>
</Server>
<Room>
<Alias>
<AliasName>COP5615</AliasName>
<IP>255.0.0.1</IP>
<Port>30000</Port>
</Alias>
<Seed>seed</Seed>
</Room>
</MigrationInfo>
</Data>
</Message>

Reconnect Acknowledgement – Refer to Project 3
<Message>
<Sender>
<AliasName>aliasName</AliasName>
<IP>127.0.0.1</IP>
<Port>30000</Port>
</Sender>
<Command>UAReconnectAck</Command>
<Data>
<Server>
<AliasName>sand</AliasName>
<IP>sand.cise.ufl.edu</IP>
<Port>30000</Port>
</Server>
<MigrationInfo>
<Server>
<AliasName>server</AliasName>
</Server>
<Room>
<Alias>
<AliasName>COP5615</AliasName>
<IP>255.0.0.1</IP>
<Port>30000</Port>
</Alias>
<Seed>seed</Seed>
</Room>
</MigrationInfo>
</Data>
</Message>

Set Alias – Refer to Project 3
<Message>
<Sender>
<AliasName>aliasName</AliasName>
<IP>127.0.0.1</IP>
</Sender>
<Port>30000</Port>
</Sender>
<Command>UASetAlias</Command>
<Data>
<Alias>
<AliasName>sandy</AliasName>
</Alias>
</Data>
</Message>

Server Messages

Server Create Room Acknowledgement – ACK sent to UA (Project 2)
<Message>
<Sender>
<Alias>aliasName</Alias>
<IP>127.0.0.1</IP>
<Port>30000</Port>
</Sender>
<Command>ServerCreateRoomAck</Command>
<Data>
<Room>
<Alias>
<AliasName>bobsRoom</AliasName>
<IP>255.0.0.1</IP>
<Port>30000</Port>
</Alias>
<Seed>seed</Seed>
</Room>
</Data>
</Message>

Server Create Room Negative Acknowledgement– NAK sent to UA (Project 2)
<Message>
<Sender>
<Alias>aliasName</Alias>
<IP>127.0.0.1</IP>
<Port>30000</Port>
</Sender>
<Command>ServerCreateRoomNak</Command>
<Data>
<Room>
<Alias>
<AliasName>bobsroom</AliasName>
</Alias>
</Room>
<StatusMessage>Room could not be created.</StatusMessage>
</Data>
</Message>

Server Join Room Acknowledgement – ACK sent to UA (Project 2)
<Message>
<Sender>
<Alias>aliasName</Alias>
<IP>127.0.0.1</IP>
<Port>30000</Port>
</Sender>
<Command>ServerJoinRoomAck</Command>
<Data>
<Room>
<Alias>
<AliasName>bobsRoom</AliasName>
<IP>255.0.0.1</IP>
<Port>30000</Port>
</Alias>
</Room>
<Seed>seed</Seed>
</Data>
</Message>

Server Join Room Negative Acknowledgement – NAK sent to UA (Project 2)
<Message>
<Sender>
<Alias>aliasName</Alias>
<IP>127.0.0.1</IP>
<Port>30000</Port>
</Sender>
<Command>ServerJoinRoomNak</Command>
<Data>
<Room>
<Alias>
<AliasName>bobsroom</AliasName>
</Alias>
</Room>
>StatusMessage>Could not join room.</StatusMessage>
</Data>
</Message>

Server Leave Room Acknowledgement – ACK sent to UA (Project 2)
<Message>
<Sender>
<Alias>aliasName</Alias>
<IP>127.0.0.1</IP>
<Port>30000</Port>
</Sender>
<Command>ServerLeaveRoomAck</Command>
<Data>
>StatusMessage>Left room successfully.</StatusMessage>
<Room>
<Alias>
<AliasName>bobsroom</AliasName>
</Alias>
</Room>
</Data>
</Message>

Server Leave Room Negative Acknowledgement – NAK sent to UA (Project 2)
<Message>
<Sender>
<Alias>aliasName</Alias>
<IP>127.0.0.1</IP>
<Port>30000</Port>
</Sender>
<Command>ServerLeaveRoomNak</Command>
<Data>
<StatusMessage>Could not leave room.</StatusMessage>
<Room>
<Alias>
<AliasName>bobsRoom</AliasName>
<IP>255.0.0.1</IP>
<Port>30000</Port>
</Alias>
<Seed>seed</Seed>
</Room>
</Data>
</Message>

**Server List Rooms Acknowledgement – ACK sent to UA (Project 2)**

<Message>
<Sender>
<Alias>aliasName</Alias>
<IP>127.0.0.1</IP>
<Port>30000</Port>
</Sender>
<Command>ServerListRoomsAck</Command>
<Data>
<RoomList>
<Room>
<Alias>
<AliasName>bobsroom</AliasName>
</Alias>
</Room>
<Room>
<Alias>
<AliasName>janesroom</AliasName>
</Alias>
</Room>
</RoomList>
</Data>
</Message>

**Server List Rooms Negative Acknowledgement – NAK sent to UA (Project 2)**

<Message>
<Sender>
<Alias>aliasName</Alias>
<IP>127.0.0.1</IP>
<Port>30000</Port>
</Sender>
<Command>ServerListRoomsNak</Command>
<Data>
<StatusMessage>Could not list rooms or no rooms present.</StatusMessage>
</Data>
</Message>

**Server List Users Acknowledgement – ACK sent to UA (Project 2)**

<Message>
<Sender>
<Alias>aliasName</Alias>
<IP>127.0.0.1</IP>
<Port>30000</Port>
</Sender>


Server List Users Negative Acknowledgement – NAK sent to UA (Project 2)

<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerListUsersNak</Command>
  <Data>
    <StatusMessage>Could not list users. Room does not exist.</StatusMessage>
  </Data>
</Message>

Server Update Room Acknowledgement – ACK sent to UA (Project 2)

<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerUpdateRoomAck</Command>
  <Data>
    <Room>
      <Alias>
        <AliasName>bobsRoom</AliasName>
        <IP>255.0.0.1</IP>
        <Port>30000</Port>
      </Alias>
    </Room>
    <Seed>seed</Seed>
  </Data>
</Message>

Server Hello – Refer to Project 3
Server Hello Acknowledgement – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerHelloAck</Command>
  <Data/>
</Message>

Server Hello Negative Acknowledgement – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerHelloNak</Command>
  <Data>
    <StatusMessage>Error message</StatusMessage>
  </Data>
</Message>

Server Neighbor Connect – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerNeighborConnect</Command>
  <Data/>
</Message>

Server Neighbor Connect Ack – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerNeighborConnectAck</Command>
  <Data/>
</Message>

Server Heartbeat – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerHeartbeat</Command>
  <Data/>
</Message>

Server Pong (to UA) – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerServerPong</Command>
  <Data/>
</Message>

Server Backup Server List (to UA) – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerBackupServerList</Command>
  <Data>
    <AliasList>
      <Alias>
        <AliasName>sand</AliasName>
        <IP>sand.cise.ufl.edu</IP>
        <Port>30000</Port>
      </Alias>
      <Alias>
        <AliasName>rain</AliasName>
        <IP>rain.cise.ufl.edu</IP>
        <Port>30000</Port>
      </Alias>
    </AliasList>
  </Data>
</Message>

Server Migrate Acknowledgement – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerMigrateAck</Command>
  <Data/>
</Message>

Server Reconnect – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerReconnect</Command>
  <Data/>
</Message>

Recovered UA – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerRecoveredUA</Command>
  <Data>
    <Alias>
      <AliasName>sandy</AliasName>
      <IP>sand.cise.ufl.edu</IP>
      <Port>30000</Port>
    </Alias>
  </Data>
</Message>

Server Search – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerSearch</Command>
  <Data>
    <Alias>
      <AliasName>sandy</AliasName>
    </Alias>
    <QueryID>1</QueryID>
  </Data>
</Message>

Server Search Results Acknowledgement – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerSearchResultsAck</Command>
  <Data>
    <QueryID>1</QueryID>
    <Alias>
      <AliasName>sandy</AliasName>
    </Alias>
  </Data>
</Message>
Server Search Results Negative Acknowledgement – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerSearchResultsNak</Command>
  <Data>
    <QueryID>1</QueryID>
    <Alias>
      <AliasName>sandy</AliasName>
    </Alias>
  </Data>
</Message>

User Found – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerUserFound</Command>
  <Data>
    <Alias>
      <AliasName>sandy</AliasName>
      <IP>sand.cise.ufl.edu</IP>
      <Port>30000</Port>
    </Alias>
  </Data>
</Message>

Set Alias Acknowledgement – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
    <Port>30000</Port>
  </Sender>
  <Command>ServerSetAliasAck</Command>
  <Data>
    <Alias>
      <AliasName>sandy</AliasName>
    </Alias>
  </Data>
</Message>

Set Alias Negative Acknowledgement – Refer to Project 3
<Message>
  <Sender>
    <Alias>aliasName</Alias>
    <IP>127.0.0.1</IP>
  </Sender>
</Message>
<Port>30000</Port>
</Sender>
<Command>ServerSetAliasNak</Command>
<Data>
  <Alias>
    <AliasName>sandy</AliasName>
  </Alias>
  <StatusMessage>Unable to set alias name.</StatusMessage>
</Data>
</Message>
Revisions

• V3.0  
  o Added missing Alias tag in ServerSearchResultsACK and ServerSearchResultsNAK  
  o Updated ServerSetAliasNAK: moved StatusMessage field out from Alias

• v2.9  
  o Added missing information in UAReconnectAck.

• v2.8  
  o Added new section: User Agent Management  
  o Added the following messages: UASetAlias, ServerSetAliasAck, ServerSetAliasNak  
  o Modified the following messages: ServerSearch

• v2.7  
  o Made XML more compliant with W3 XML specification.  
  o Revised UserListUsers, UserCSend, UserCreateRoom, UserLeaveRoom, and UserListUsers commands.

• v2.6  
  o Completely restructured XML

• v2.5  
  o Added migration information to Reconnect ACK

• v2.4  
  o Addition of a "-recover" command line option as an argument to the server.  
  o Complete overhaul of search functionality  
  o More granularity within csend messages  
  o Elimination of one message from the server recovery phase  
  o Elimination of the neighbor list message  
  o Elimination of the neighbor pong message

• v2.3  
  o Added the following XML message format in the UA section: Send and CSend

• v2.2  
  o Renamed the following messages: ‘Search Results' to ‘Search Results Ack’  
  o Modified the following messages: Search (Server to Server)  
  o Added the following messages: Search Results Nak  
  o Added XML ‘Message Format’ section.

• v2.1  
  o Specified a range for the number of neighbor connections  
  o Removed the following messages: List Neighbors, Neighbor Disconnect, Neighbor Disconnect ACK, Neighbor Dead  
  o Changed the name of the Migrate Request to Reconnect  
  o Added the following messages: Current Server, Reconnect ACK, Remove UA

• v2.0  
  o Initial protocol
FAQ

Note: This FAQ was generated based on peer review.

1. **User should not control network topology.**
   As proposed in class users may have the option to disconnect from its existing neighbor(s). After careful consideration allowing users this ability violates user transparency. Therefore, commands allowing the user to control network topology have been removed.

2. **Neighbor’s user list is maintained at each server which is unnecessary as discussed in class.**
   Maintaining a neighbor’s user list is necessary when searching for a user as it reduces network traffic.

3. **Server recovery is not described and children recovery is not discussed.**
   Read from the protocol: When the failed server recovers it reads its neighbor list and UA list from the non-volatile storage, if it exists. The server then reconnects to its neighbors, by issuing Neighbor Connect message and then sends a Migrate Request message to each one of its UAs.

4. **Chatroom ownership not described.**
   Refer to the Switching Servers section.

5. **No mention of data types and semantics.**
   From Wiki: A protocol can be defined as the rules governing the syntax, semantics, and synchronization of communication.

6. **How are different messages differentiated on different threads on a server?**
   Defining what threads spawn each message is beyond the scope of this document.

7. **How often to check whether a server is alive? If not often, then the two message passes are good. If periodically, too many messages are passed.**
   This is left to each group to determine.

8. **Necessary to determine if your chatroom is alive? Chatrooms stay alive even if servers fail.**
   Refer to the Switching Servers section.

9. **“Neighbor Disconnect” message is not required as status of neighboring servers is taken care by periodic server ping.**
   Refer to question #1.