# Redes 1: Informe Proyecto Final - Chat Asincrónico

**Grupo 4:** José Cárdenas (209163), Christian López (200613), Diego Veintimilla (206037)

## 1. Diseño Protocolo

Para este proyecto, se escogió la opción del chat asincrónico con funcionalidad de compartición de archivos. Para alcanzar este objetivo, se ha diseñado una aplicación servidor y otra cliente. Dentro del servidor existen 2 sockets escuchando en puertos distintos, los cuales corresponden al servicio de chat y al servicio de archivos respectivamente.

A continuación, se presentan los diagramas de secuencia de los protocolos implementados para cada servicio.

**Gráfico 1. Diagrama de secuencia del protocolo del chat.**

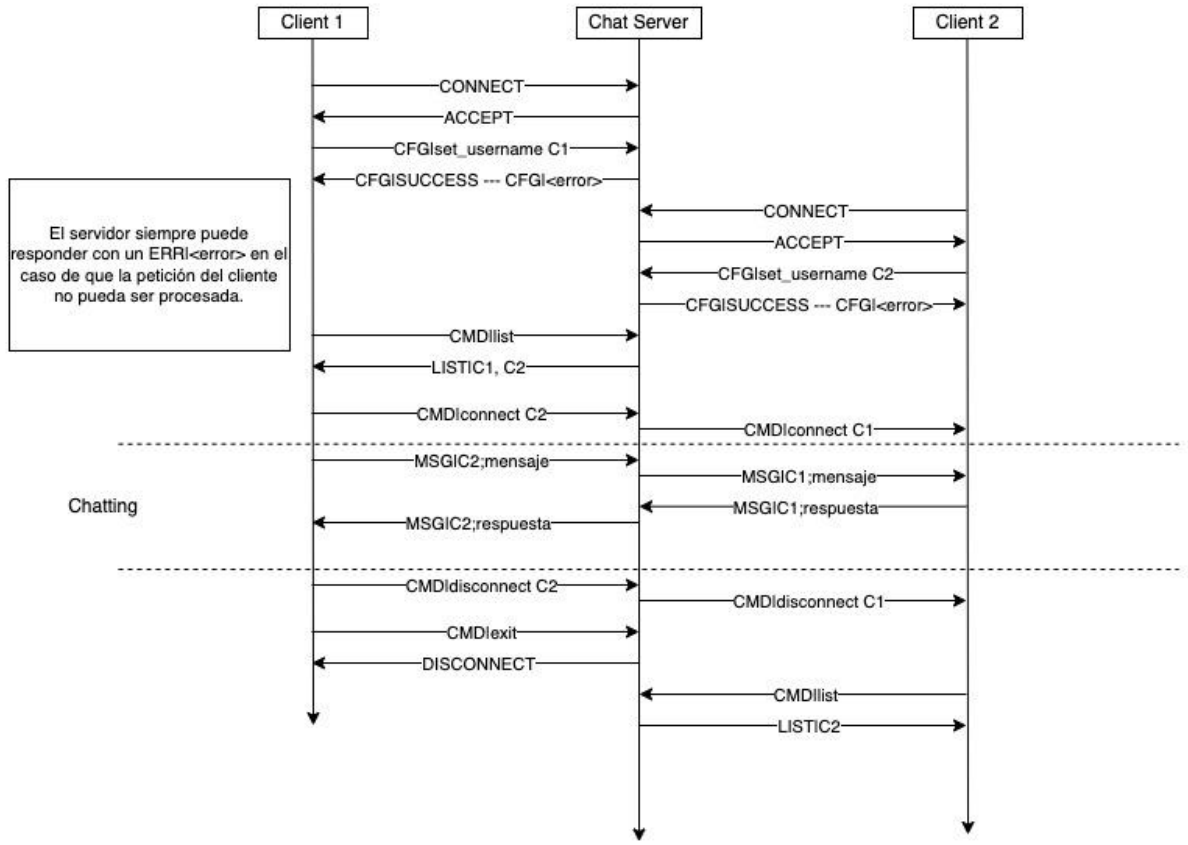
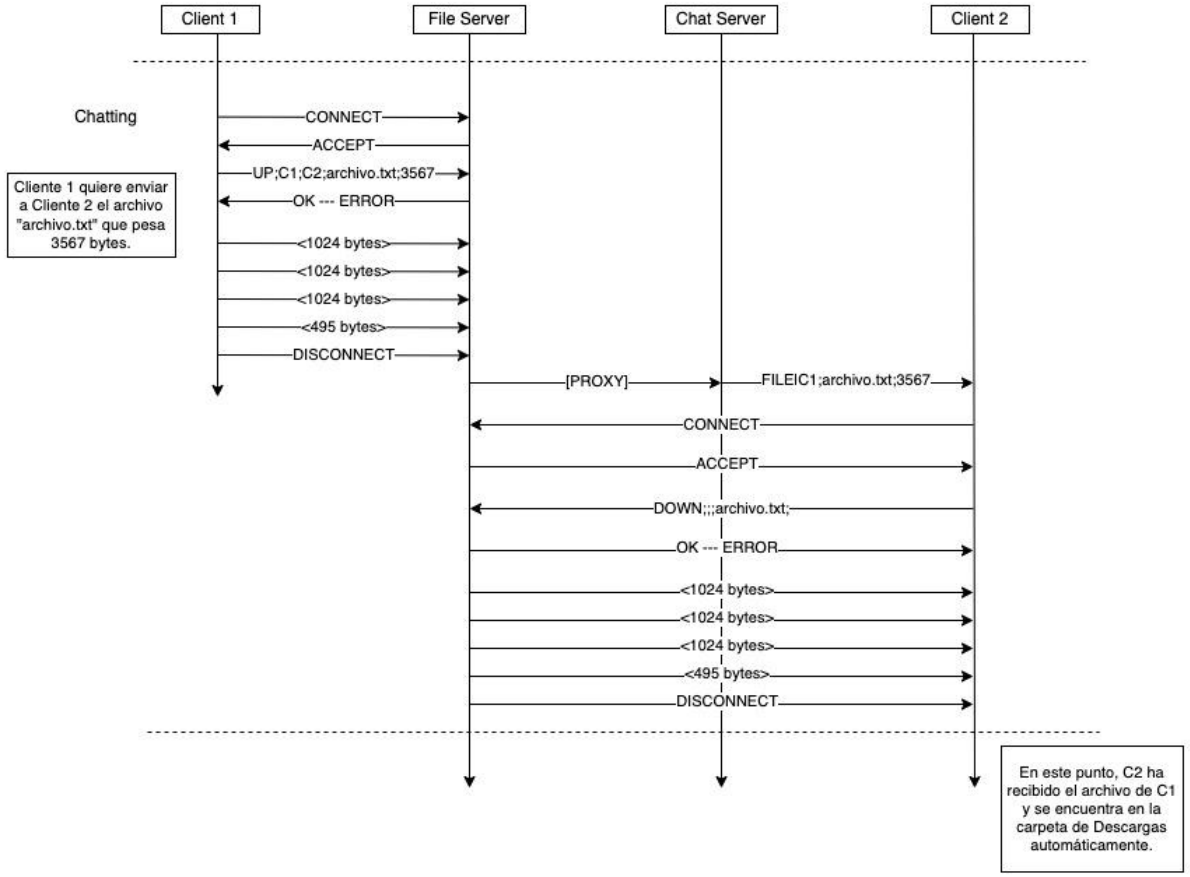


**Gráfico 2. Diagrama de secuencia del protocolo de la transferencia de archivos.**




Un cliente se puede conectar al servidor de chat a través del puerto 10023. Apenas se conecta, debe enviar su nombre de usuario a través del mensaje **CFG|set_username <user>**, si recibe un **CFG|SUCCESS** entonces el cliente está listo para interactuar con el servidor, caso contrario el cliente deberá cerrar la conexión e intentar nuevamente. Los nombres de usuario deben ser únicos, por lo que, si el cliente intenta utilizar un nombre de usuario ya tomado, recibirá un **ERR|Username already in use.**

En este momento, el cliente puede enviar un **CMD|list** para adquirir una lista de los usuarios conectados (separados por , ), esta lista lo incluye a él, por lo que en su recepción se debería filtrar su nombre. Para conectarse con alguien, el cliente debe enviar un **CMD|connect <user>**, en el caso de que el cliente no haya especificado un nombre de usuario, o si el nombre de usuario no se encuentra en el servidor, entonces recibirá un **ERR|You need to specify a user to connect to.** o un **ERR|Username not found.** Si todo sale bien, el servidor responderá con un **CMD|SUCCESS** y enviará un comando al cliente destinatario tal que **CMD|connect <user>.**

Una vez conectado con alguien, el cliente puede enviar un **MSG|<user>;<message>** para enviar su mensaje a un destinatario definido, dado a que puede estar conectado con varias personas a la vez. Dentro de un chat, el cliente puede enviar un **CMD|disconnect <user>** para desconectarse del chat con el destinatario. En caso de que el usuario no haya sido especificado o si no está en un chat con el cliente ya iniciado, el servidor responderá con un

**ERR|You need to specify a user to disconnect from.** o un **ERR|You are not chatting with <user>.** En el caso de que la desconexión sea exitosa, el servidor responderá con un **CMD|SUCCESS** y enviará un mensaje a la contraparte tal que **CMD|disconnect <user>.**

El cliente también puede enviar un **CMD|exit** para desconectarse del servidor por completo, esto enviará los **CMD|disconnect <user>** necesarios para que las contrapartes que hayan estado chateando con este cliente, estén notificadas de su desconexión. En este punto, el servidor cierra el socket con el cliente.

Para el envío de archivos, el cliente debe estar en un chat en ese momento. De ser así, el cliente se conecta con el servidor de archivos, y envía un mensaje tal que **UP;<source_user>;<destination_user>;<filename>;<file_size>**. El servidor responderá con un **OK** si el usuario que envía el archivo y si el destinatario existen en el servidor de chat, también verifica que la primera palabra corresponda a **UP** para subida de archivos y **DOWN** para descarga, caso contrario envía un **ERROR** y cierra la conexión.

Una vez que el servidor haya enviado el **OK**, el cliente empezará a subir los bytes del archivo hasta terminar. Una vez acabado el proceso, el cliente solo cerrará la conexión sin despedirse. Cuando el servidor haya adquirido todas las partes del archivo, lo guardará en una carpeta en el servidor, y enviará (a través del socket del chat) un mensaje al usuario destino tal que **FILE|<source_user>;<filename>;<file_size>**. Al recibir esto, el cliente sabe que deberá descargar el archivo que le fue enviado. Para esto, este cliente deberá conectarse con el servidor de archivos, y envía un mensaje tal que **DOWN;;;<filename>;** (Nótese que el número de ; es el mismo que en el mensaje **UP** que fue enviado anteriormente). De manera análoga al mensaje **UP**, el servidor puede responder con un **OK** o un **ERROR** dado las mismas condiciones anteriores incluyendo si es que el nombre del archivo no ha sido encontrado en el servidor. Dado el **OK**, el servidor empezará a enviar los bytes del archivo directamente al cliente y cerrará la conexión una vez terminado el proceso.

## 2. Diseño Aplicación

Para el diseño de la aplicación, se siguió un patrón de diseño orientado a eventos. Para esto, se definieron una clase **EventEmitter** y **ClientEventEmitter** los cuales permiten registrar funciones que manejan eventos recibidos.

Se decidió utilizar este patrón de diseño dada la naturaleza asincrónica del servidor dado a que en muchos casos el cliente no necesita recibir la respuesta del servidor en el mismo lugar donde hizo la petición. Esto simplifica de cierta manera la comunicación entre el cliente y la interfaz gráfica, ya que nada más se registran los eventos ‘list’ para actualizar la lista de usuarios, el evento ‘message’ para mostrar los mensajes en la pantalla que corresponde o el evento ‘file’ el cual desencadena la descarga del archivo compartido.

Cabe mencionar que esto sirve bastante bien por la manera en que se han estructurado los mensajes, ya que cada uno tiene un tag que identifica la naturaleza del mensaje, si se trata de un comando, de una configuración o de un mensaje de alguien.

Para reducir el número de pantallas a diseñar, se decidió que el nombre de usuario del cliente debe pasarse como argumento en la ejecución del mismo y no lanzará la interfaz gráfica de haber algún problema con el nombre de usuario. Se realizó esto dado a que el comando **CFG|set_username** es el único que se identificó que sí requiere de obtener una respuesta en el mismo lugar que de donde se realizó la petición.

En el caso de que un cliente empiece un chat con alguien más, las pantallas del chat se desplegarán en ambos clientes automáticamente, así como solían funcionar aplicaciones como MSN o AIM.

Otro punto que vale la pena mencionar, es que el cliente a quien se le ha enviado un archivo descargará el mismo automáticamente en la carpeta de descargas del sistema y notificará a través de una alerta al usuario que el archivo se ha descargado. Se realizó esto de esta manera también para simplificar el desarrollo de la interfaz gráfica.

## 3. Conclusión

Finalmente, a través de este proyecto, se exploró de manera más extensa el desarrollo de aplicaciones con protocolos definidos por el desarrollador utilizando sockets de bajo nivel. Un punto importante en este caso es el envío de archivos, ya que usualmente tienden a ser de mayor tamaño que el buffer del socket, lo cual representa un desafío para que este proceso se dé adecuadamente.
Otro punto interesante es que, el desarrollo de esta aplicación nos forzó a investigar sobre patrones de diseño que se adhieran apropiadamente con los requisitos de la aplicación. El patrón de diseño orientado a eventos resulta ser bastante bueno, especialmente por cómo funciona Python y por lo que funciones pueden ser asignadas a variables y parámetros, de esta forma delegando el manejo de un recurso recibido de algún lado a un delegado que necesita de este recurso.