


Protocolo de Mensajería Instantánea



Grupo 2
- María José González
- Diego Tarud

Descripción General

El protocolo diseñado para el intercambio de mensajes entre usuarios a través de un chat por consola. Este chat será implementado en Python mediante su librería de sockets. La aplicación, a grandes rasgos, permitirá la comunicación entre uno o varios usuarios, así como el envío de archivos entre ellos.

Para establecer esta comunicación, se usará el protocolo TCP a través de una arquitectura cliente-servidor, lo que garantizará el envío íntegro de estos mensajes así como mantener el orden que son enviados a través de la red. Los mensajes serán del tipo byte-stream, definido por el protocolo TCP. 

El usuario al entrar a la sala de chat se conectará con el servidor. Cuando se efectúe esta conexión el servidor responderá con la lista de usuarios conectados, con la ip del servidor y al puerto de cada uno. Cuando el usuario desee comunicarse con otro, podrá iniciar una conversación con él, así mismo, puede crear un grupo con varios usuarios. A este grupo también será posible agregar más usuarios, sin embargo el que creó el grupo es el que enviará las invitaciones para que otras personas puedan participar de la conversación.

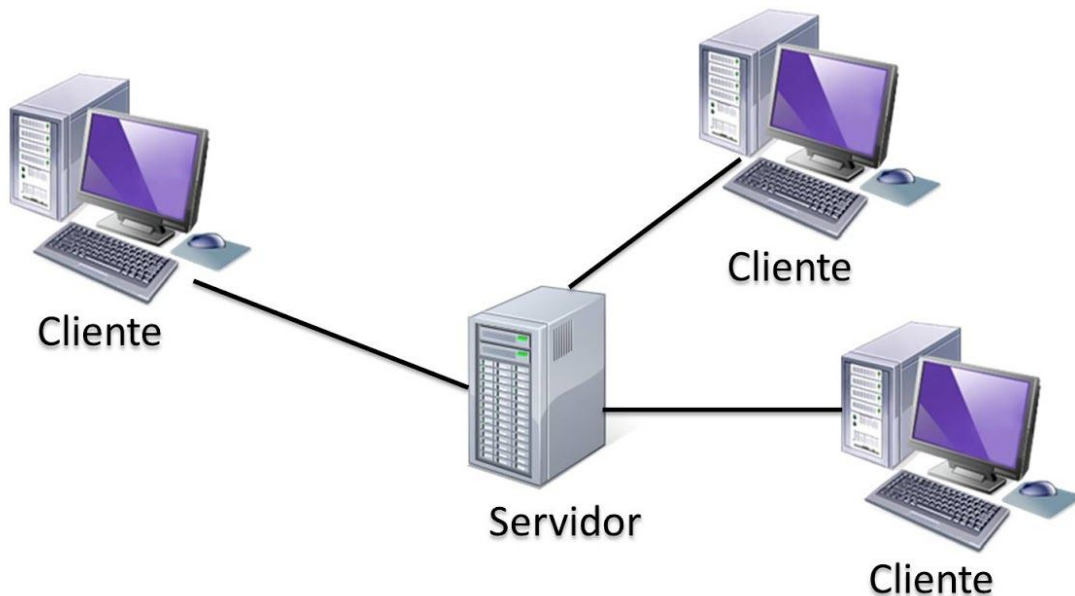
Por otro lado, para enviar archivos, esto será posible indicando la ruta del archivo y el puerto de destino. Esto implica que los archivos pueden ser enviados a uno o varios usuarios que estén participando de la interacción, en donde cada uno podrá decidir si aceptarlo o rechazarlo (explicado esto en los comandos). El envío de archivos interrumpirá una conversación para mostrar la solicitud de envío. Cuando un usuario quiera abandonar la conversación podrá hacerlo sin necesidad de solicitar el permiso del administrador en el caso de un grupo, o en el caso de una interacción uno a uno puede simplemente abandonarla. A continuación se listan los comandos de interacción con la aplicación:

- *show_comands*: muestra la lista de comandos y una breve descripción de cada uno.
- *connect_with <port>*: permite establecer una conversación con una sola persona.
- *create <nombre>*: permite crear un grupo de conversación. El usuario que creó el grupo pasará a ser su administrador. Si el nombre ya existe, se lanzará una excepción que lo destaque y pida al usuario volver a crear el grupo con otro nombre.
- *invite <port>*: permite que el administrador de un grupo invite a un usuario al grupo de chat.
- *send_file <file_path> <por_1, port_2, ..., port_n>*: permite enviar un archivo a uno o varios usuarios especificando su ruta y puerto de destino. Los usuarios que reciban el request del archivo pueden decidir si aceptarlo o rechazarlo. El archivo por defecto se guardará en la carpeta de descargas.
- *show_users*: despliega una lista con los usuarios actualmente conectados y al grupo al que pertenecen (si es que lo hacen).
- *show_groups*: muestra una lista con los grupos activos.

- *quit*: retirarse de una conversación. Solicita una confirmación por parte del usuario.

Tipo de hosts

La arquitectura elegida para nuestro protocolo será Cliente-Servidor, donde los usuarios se conectan a un servidor y este responderá con una lista de usuarios conectados. Así mediante un comando se podrá iniciar un chat de conversación con otro(s) usuario(s).



En esta arquitectura, el cliente es capaz de:

- Iniciar las solicitudes de comunicación, enviando un request al servidor.
- Recibir las respuestas del servidor.
- Conectarse a varios servidores a la vez, aunque en este contexto no es necesario.
- Interactuar con otros usuarios mediante la conexión a servidores comunes.




Mientras que el servidor tiene las siguientes atribuciones:

- Esperar a que lleguen solicitudes de los clientes.
- Recibir solicitudes de los clientes, procesar y enviar la respuesta al cliente que se lo solicitó.
- Es capaz, además, de soportar múltiples conexiones de clientes, aunque en algunos casos el número de peticiones puede estar limitado para evitar caídas de conexión.

El host corresponde a un computador conectado al servidor que recibe las conexiones de los clientes y permite la comunicación entre usuarios ejecutando la aplicación de chat.


Formato de mensajes

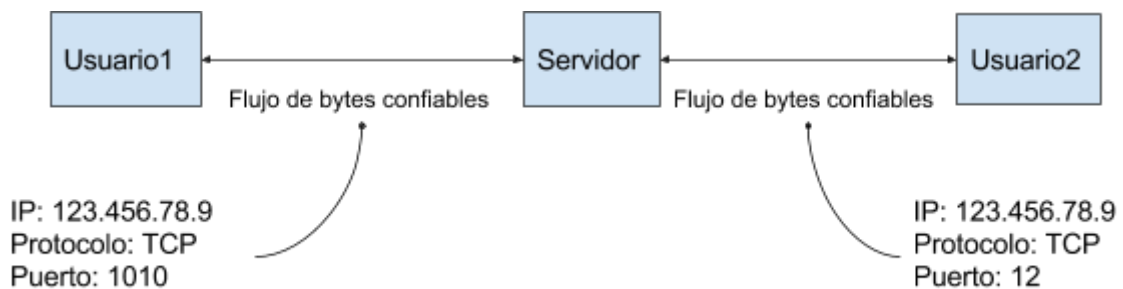
El siguiente cuadro nos muestra el formato que tendrán los mensajes transmitidos entre cliente - servidor.

- **Número de secuencia:**  para garantizar que el mensaje o los datos son enviados íntegramente y en el orden correcto, además de revisar las pérdidas de paquetes.
- **Acción:** Es para ver si el mensaje es por una conexión con el servidor, desconexión, envío de archivo a algún usuario(s), solicitar lista de usuarios conectados. A continuación se listan y detallan los tipos de acción:
 - Req.Connect: request de conexión. Solicitud del cliente (usuario) para conectarse al servidor.
 - Req.Disconnect: request de desconexión. Solicitud del cliente para desconectarse del servidor.
 - Res.Confirmation: response de estado de la conexión. Entrega un mensaje de si la conexión ha sido exitosa o no. Es análogo a un acknowledgement para entregar el estado del inicio de la interacción.
 - Req.Send: request de envío de mensaje.
 - Res.Send_confirmation: response de envío y recepción exitoso de mensaje.
 - File_transfer: request de transferencia de archivos. Establece una conexión para el envío de archivos y entrega respuesta si se hizo la transferencia o no.
- **Cliente/Servidor:** Entregará información si el mensaje viene del cliente o servidor.
- **IP:** Se necesita la IP del servidor para comunicarse con él.
- **Puerto origen y Puerto de destino:**  Puerto identifica el proceso(usuario en caso de ser en el mismo computador)
- **Tipo Contenido:** Texto plano o binario. 
- **Cuerpo:** Mensaje

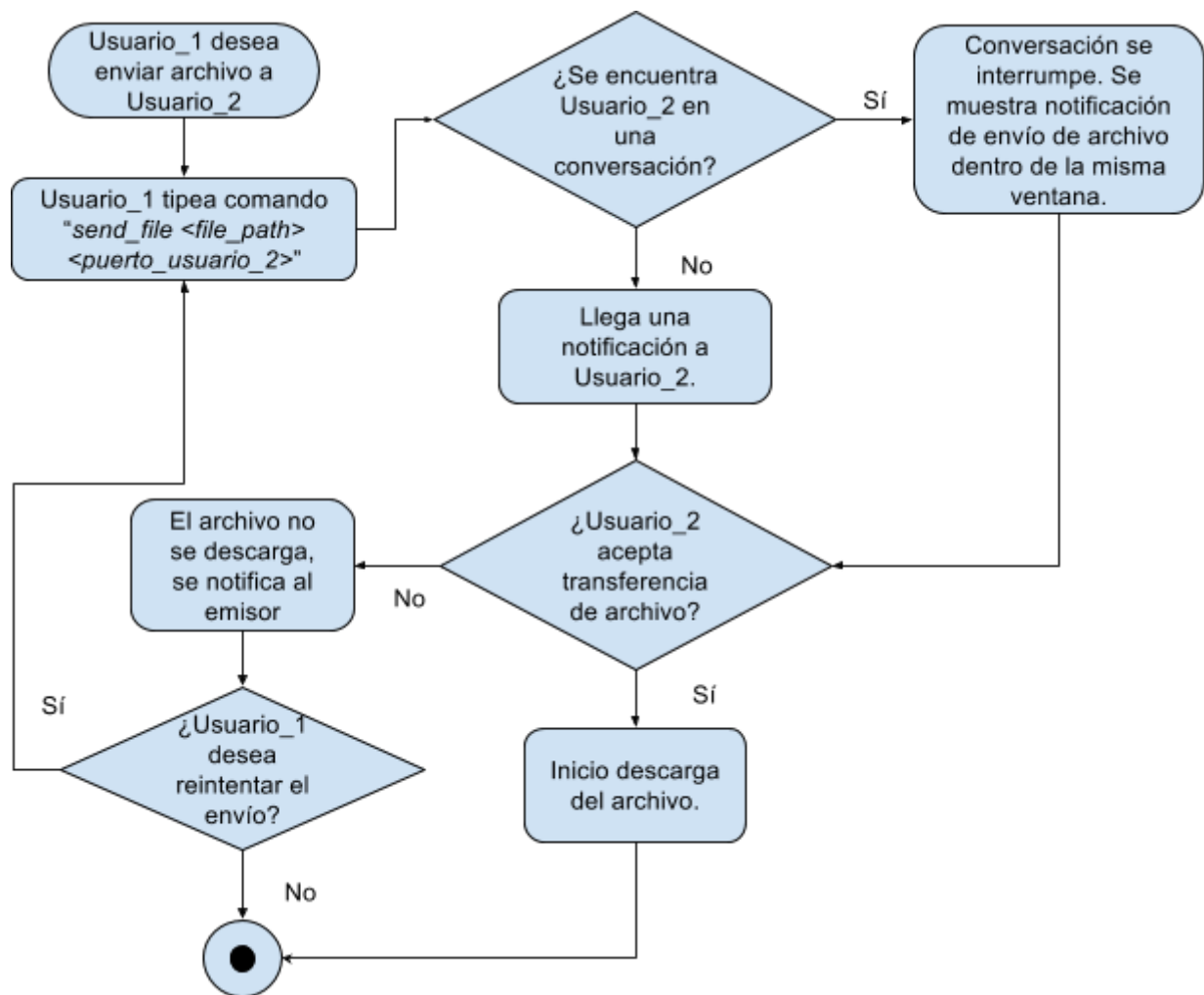
Número de secuencia	IP	
Cliente/Servidor		
Acción		
Puerto origen		Puerto destino
Tipo Contenido		
Cuerpo		

Comunicación

Para la transmisión de mensajes en texto plano y archivos se usará comunicación de flujo de bytes confiable ya que se debe asegurar una correcta recepción del mensaje y mantener el orden de emisión y recepción.  quiere decir que el servidor se encargará que dicho mensaje llegue en el orden correcto, revisando que el número de secuencia de cada bit del mensaje esté en orden. Además habrá manejos de errores, interrupciones para el caso de transferencia de archivos , mensajes de confirmación al crear grupos y al establecer conexión.

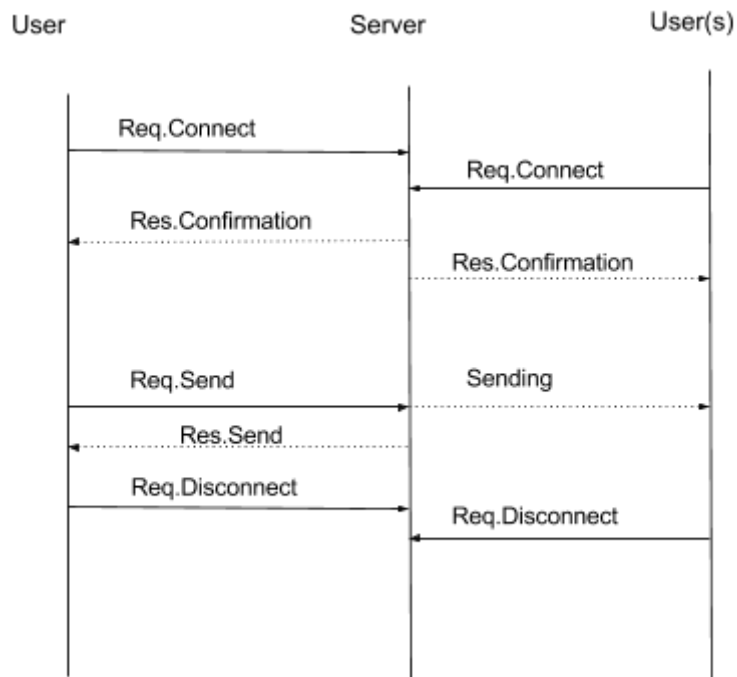


Para el envío de archivos, esto se explicará en el siguiente diagrama de flujo (esta transferencia también se hace a través de un flujo de bytes confiable):

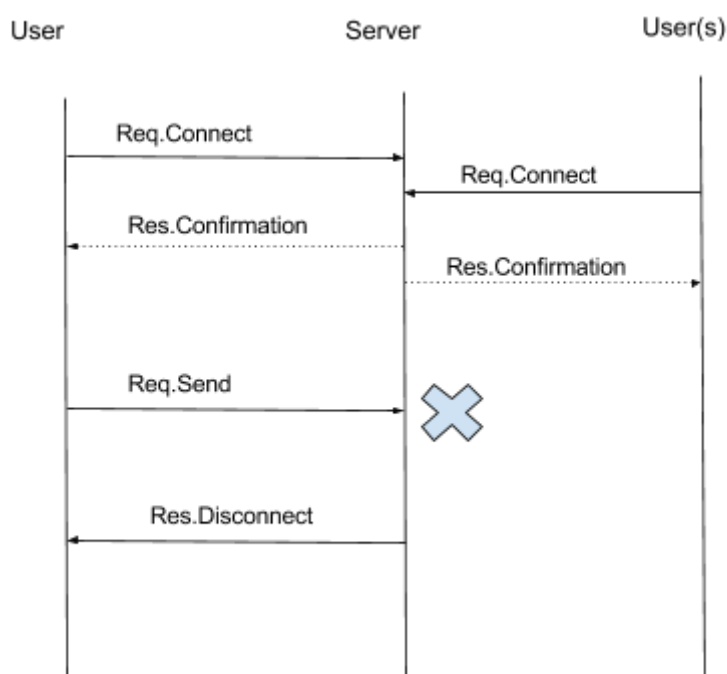


Interacciones

El siguiente diagrama muestra la interacción normal, iniciando una conexión, envío de mensajes y desconexión entre usuarios-servidor.

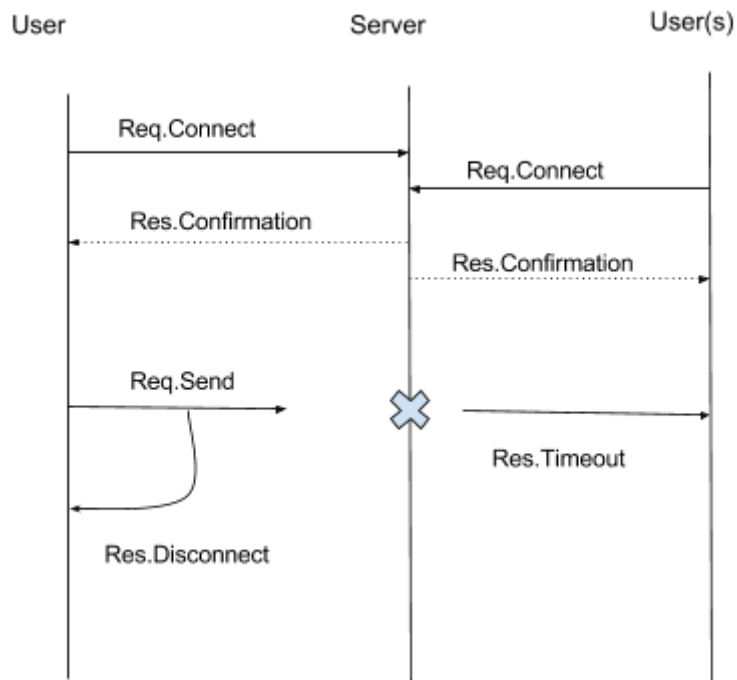


La desconexión en el diagrama anterior se lleva a cabo mediante una solicitud de alguno de los dos clientes. En el caso de que la desconexión sea de forma abrupta, por ejemplo una caída del host de uno de los usuarios, el diagrama a continuación describe la forma de manejo de dicho error:



Para esta situación, el servidor notificará al usuario que envió el mensaje que su contraparte se ha desconectado.

Otro caso de desconexión posible es una caída del servidor encargado de establecer la conexión entre usuarios, detallado a continuación:



Aquí, la aplicación del usuario que envía el mensaje se encargará de notificarle que no se ha podido establecer una conexión con el servidor. Para el usuario que recibe esta notificación será desplegada cuando el tiempo de recepción del mensaje sea excedido. Así, su aplicación le notificará que no se ha podido establecer conexión.

Estados

El siguiente diagrama muestra los estados que pueden tener un Usuario y Servidor.

El primero indica cuando el usuario pasa de estar desconectado a conectado con el servidor, luego si desea escribir un mensaje a otro usuario, ingresa el comando definido y pasa a un estado de chat, donde al escribir los mensajes serán enviados al destinatario.

El segundo muestra los estados que puede tener el servidor, el estado IDLE vendría siendo donde el servidor está escuchando cuando algún usuario desee conectarse y cuando dicho usuario desee enviar un mensaje hará llegar ese mensaje al receptor.

