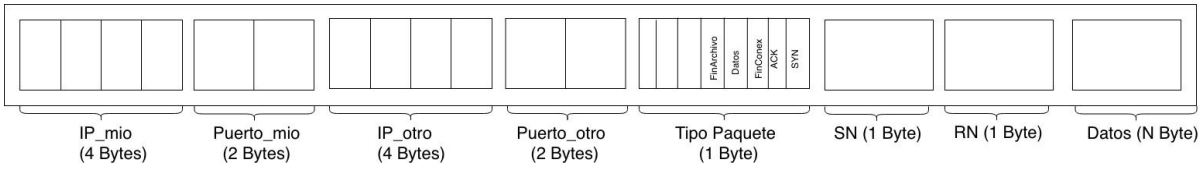
Decisiones de diseño:

3 pasos de handshake:

Para poder producir un handshake completo, se realizó en 3 pasos. Durante el handshake los nodos comparten sus SN y puertos por los cuales los nodos se van a comunicar. El handshake solo se realiza cuando es una nueva conexión, por lo tanto antes de saber si hacer un handshake se busca dentro de las conexiones para saber si hay que hacer una nueva. Durante el handshake no se transmite ningún dato entre los nodos.

Tamaño del paquete y todo su contenido:



En la imagen anterior mostramos un diagrama del esqueleto de los paquete que utilizamos para enviar y recibir datos. Al principio tenemos el Ip (4 bytes) y el puerto(2 bytes) del nodo que envía, seguido por el Ip (4 bytes) y el puerto(2 bytes) del nodo que recibe, seguido de esto viene un byte donde se especifica el tipo de paquete dependiendo del número que venga, puede tomar valores combinados para SYN, ACK, FinConex, Datos y FinArchivo. Este byte es primordial a través de todo el proceso para saber qué tipo de paquete es que se está enviando y cómo tratarlo. Seguido de este byte siguen los 2 bytes: 1 Byte SN y otro Byte para RN, estos campos son los que se utilizan para enviar y recibir los números de secuencia y así saber si el paquete que no está llegando es el correcto.

Por último tenemos la sección de datos, que en nuestro paquete consiste en 5 bytes donde pueden guardarse los datos del archivo que se desea enviar.

HiloConexionUDPSegura (Receptores-Emisores)

Estos hilos realizan la conexión full-duplex son tanto emisores como receptores, esto para que puedan enviar datos en el paquete de respuesta (en el caso de que tengas paquetes por enviar).
Se utiliza un buffer para almacenar los datos(particionados) que se tienen que enviar y en cada envio de paquete se toma uno de esos datos y se agrega el encabezado necesario, luego de esto se envía, en caso de no tener que enviar datos, se responde con un paquete sin datos.
Estos hilos solicitan paquetes de una estructura llamada Buzón, donde se almacenan los paquetes recibidos por el hilo servidor, en caso de no tener paquete, espera un tiempo y vuelve a solicitar y si de nuevo no hay paquete entonces hace un reenvío del paquete que había enviado anteriormente. Si llega a 10 reintentos se cierra la conexión.

Hilos emisores

Este es el hilo encargado de manejar toda la interacción con el usuario.
Cuando se solicita enviar un archivo a un nodo, se solicita la IP y el Puerto, si la conexión ya existe la utiliza(añade los datos al buffer que utiliza el Hilo Receptor-Emisor), si no existe la conexión entonces crea una nueva.

Hilos servidores

El hilo receptor es el encargado de recibir todos los paquetes de todas las conexiones, y los ingresa a un buzón donde los Hilos Receptores-Emisores pueden leer. En caso de que algún paquete sea de inicio de conexión, este crea un Hilo Receptor-Emisor y luego pasa el paquete al buzón para que el hilo realice el proceso de inicio de conexión.

Para simular la pérdida de paquetes se genera un numero random entre 0 y 10, si este número generado es menor o igual a 1 entonces se descarta el paquete.

Cierre de Conexión

Se decidió tener 2 formas de poder cerrar conexiones, una es que el nodo cierra todas sus conexiones arbitrariamente o solo cerrar una conexión en específico. El cierre de conexión se da de la siguiente manera, el nodo que decide cerrar la conexión envía un fin de conexión y queda esperando una respuesta y si no vuelve a enviar el mensaje y lo hace con un máximo de 10 intentos antes de que se cierre porque no le respondieron.

TimeOuts(0.5 segundos)

Se decidió implementar un timeout de 0.5 seg ya que este puede variar según la velocidad de conexión entre los procesos, esto se vio reflejado cuando se utilizó la herramienta de Haguichi para simular que estábamos en la misma red y esto nos ocasionó un delay más grande de 0.3 seg.

Cantidad de reintentos de envío(10)

Se decidió que la cantidad de reintentos que un nodo puede hacer para contactar a otro nodo es de 10 intentos de mensaje, si el nodo al cual le mandó los mensajes no responde con algún tipo de mensaje antes de 10 mensajes entonces la conexión entre ambos nodos terminará. Esto nos asegura que no se envíen infinitos mensajes esperando respuesta alguna. Se decidió hacer al menos 10 intentos ya que en varias ocasiones hubieron 5 descartes continuos de paquetes.

Mantener viva la conexión hasta que se indique el cierre de las conexiones

Decidimos mantener la conexión viva hasta que el usuario desee cerrarla para que no se tenga que estar creando nuevas conexiones una vez que se envía un archivo, por lo tanto se pueden enviar N cantidad de archivos mediante una misma conexión y no hará falta volver a crear la conexión. En este periodo de tiempo entre los envíos de archivos los nodos se quedan comunicando para informar que siguen vivos.

No se envían datos el último handshake.

El handshake solo se encarga de realizar la conexión e intercambiar datos primordiales para poder hacer una transferencia de datos segura, por lo tanto hasta que el handshake termine se permite el envío de datos entre los nodos. De esta forma nos aseguramos que se envían los datos de manera controlada y segura o que no se envíe ningún dato si falló el handshake.

Nueva Conexión se crea por debajo

Cuando se quiere enviar un archivo el usuario simplemente ingresa el IP y el puerto destino, además de la dirección del archivo que desea enviar. El programa por debajo buscará si existe una conexión activa hacia esa dirección y de ser así enviará el archivo por la conexión existente, contrario a esto se realizará el handshake correspondiente y se enviarán los datos de manera confiable.

Requerimientos adicionales

Hilo es tanto emisor como receptor para poder hacer lo de full duplex y que se envíen datos en el paquete de “ack” de datos recibidos.

A mitad de diseño del código nos dimos cuenta que al principio estábamos implementando un half-duplex y que no estábamos siguiendo todos los requerimientos para que fuera full-duplex. Ya que al principio existían un socket para cada hilo que creaba. Posteriormente se entendió que solo debía de existir un solo socket que realizará todo el trabajo. Además nos dimos cuenta que en el mismo mensaje de respuesta a un mensaje recibido se tenía que enviar datos en caso de tener que enviar.

Hacer que el emisor espere a que se termine de enviar el archivo para poder realizar otra operación

Este requerimiento surgió para simular que se está esperando a que se envíe el archivo completo, porque si no lo que iba a pasar es que el usuario iba a poder encolar archivos a una conexión e incluso cerrarla aunque no se haya enviado el archivo (cosa que no permite un send en un socket TCP).

Se envia un tipo de mensaje distinto para el último paquete del archivo.

Al momento de que se termina de enviar un archivo (se envía el último tramo del archivo), se enviará este paquete indicando que es el último paquete del del archivo, esto para informar al nodo receptor que ya debe de cerrar el archivo donde estaba escribiendo lo que le llegaba, de esta manera el nodo receptor podía generar diferentes archivos.