FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 1 — File Transfer
Grupo B11
Fecha de entrega: 03/10/23

[75.43] Introducción a los Sistemas Distribuidos
Segundo Cuatrimestre de 2023

Integrantes

Alumno	Padrón
DUCA, Francisco	106308
GOMEZ, Nahuel	106514
SCHIFFER, Brandon	107890
SCAZZOLA, Martín	106403
SANTANDER, Valentín	105637

Índice

1. Introducción

Este trabajo práctico se plantea como objetivo la comprensión y la puesta en práctica de los conceptos y herramientas necesarias para la implementación de un protocolo RDT (reliable data transfer).

Para lograr este objetivo, se deberá desarrollar una aplicación de arquitectura cliente-servidor que implemente la funcionalidad de transferencia de archivos mediante las siguientes operaciones:

- UPLOAD: Transferencia de un archivo del cliente hacia el servidor
- DOWNLOAD: Transferencia de un archivo del servidor hacia el cliente

La comunicación entre procesos será implementada a través del protocolo UDP como capa de transporte, y tendrá una transferencia confiable usando los algoritmos de protocolos Stop and Wait y Selective Repeat.

Además, el servidor será capaz de procesar múltiples clientes a la vez y deberá proveer una garantía de entrega de paquetes.

2. Hipótesis/Supuestos

- Definimos 4096 bytes como tamaño máximo de paquete.
- En el servidor se sobreescriben los archivos en caso de subirlos con mismo nombre.
- Se asume que la red es segura y que no hay amenazas importantes de ataques de seguridad.
- Se supone que el sistema de archivos del servidor tiene suficiente espacio de almacenamiento para manejar los archivos que se cargarán.
- Se supone que no hay restricciones de firewall que impidan la comunicación entre el cliente y el servidor a través del puerto especificado.
- Se supone que la red en la que se implementará la aplicación cliente-servidor es relativamente estable.
- Se asume que tanto el cliente como el servidor están utilizando la misma versión del protocolo.
- Se asume que el checksum sea validado por UDP [RFC 768].
- Se define un tamaño maximo de archivo de 100MB para la transferencia de datos.
- Se define un TIMEOUT fijo de 0.1 segundos.

3. Detalles de implementación

La aplicación cliente-servidor consta de dos componentes principales: el cliente (client.py) y el servidor (server.py). Ambos componentes están diseñados para funcionar en el lenguaje de programación Python y se basan en la arquitectura UDP (User Datagram Protocol) para la comunicación.

En el archivo README.md se encuentran las formas de levantar el servidor y el cliente con sus respectivos comandos para la transferencia de archivos.

Al momento de realizar una transferencia se debe especificar la acción que el cliente quiera hacer (download o upload) del archivo, y su respectivo protocolo de comunicación, en este caso, Stop and Wait o Selective Repeat.

El componente del cliente se encarga de las siguientes tareas:

- Descargar archivos: El cliente puede solicitar y descargar archivos del servidor.
- Subir archivos: El cliente puede cargar archivos al servidor.

El cliente utiliza dos protocolos diferentes para la comunicación: Stop and Wait (SAW) y Selective Repeat (SR), dependiendo de la configuración.

El componente del servidor maneja las solicitudes entrantes de los clientes. Está diseñado para:

- Escuchar y aceptar conexiones: El servidor escucha en un puerto específico y acepta conexiones entrantes de los clientes.
- Gestionar múltiples clientes: Puede manejar múltiples clientes simultáneamente utilizando subprocesos separados para cada cliente.
- Almacenar archivos: Almacena los archivos cargados por los clientes en una ubicación específica en el sistema de archivos del servidor.

Al inicializarse el servidor, el mismo utiliza threads para poder manejar concurrentemente a los distintos clientes que le lleguen. Cada cliente se comunica a través de un socket (interfaz para permitir la comunicación entre hosts) para el envío y recibimiento de mensajes. El servidor tiene un único socket de comunicación para varios clientes. El servidor va a estar siempre esperando mensajes de los clientes salvo que se interrumpa a través de la consola.

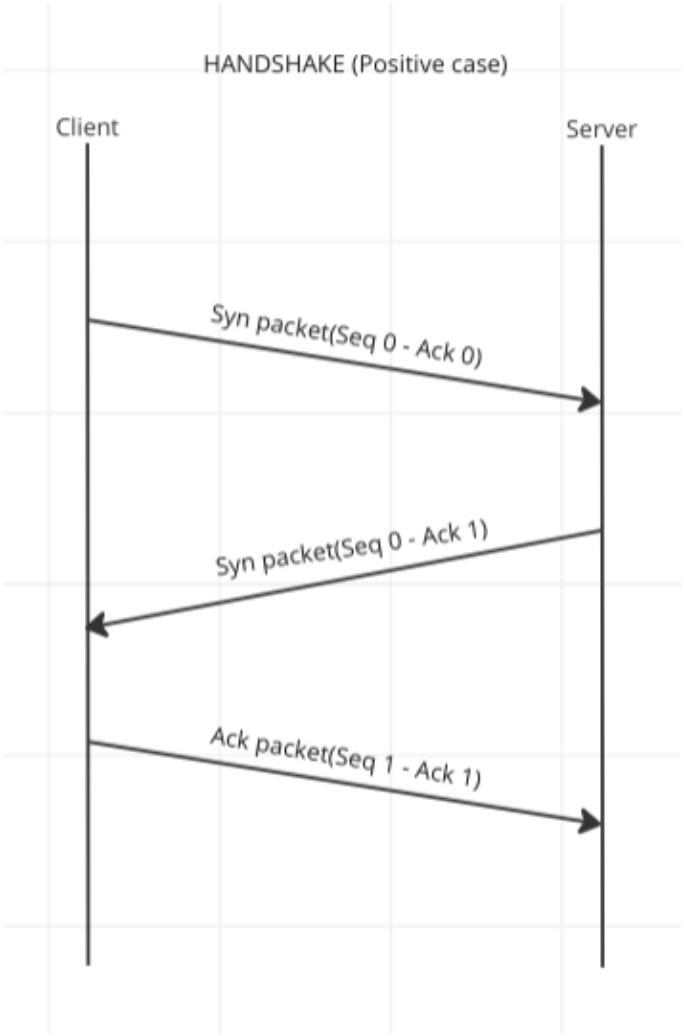

Figura 1: Conexión entre cliente y servidor antes de enviar o recibir paquetes

En la figura 1 se ve una conexión exitosa entre el cliente y el servidor.

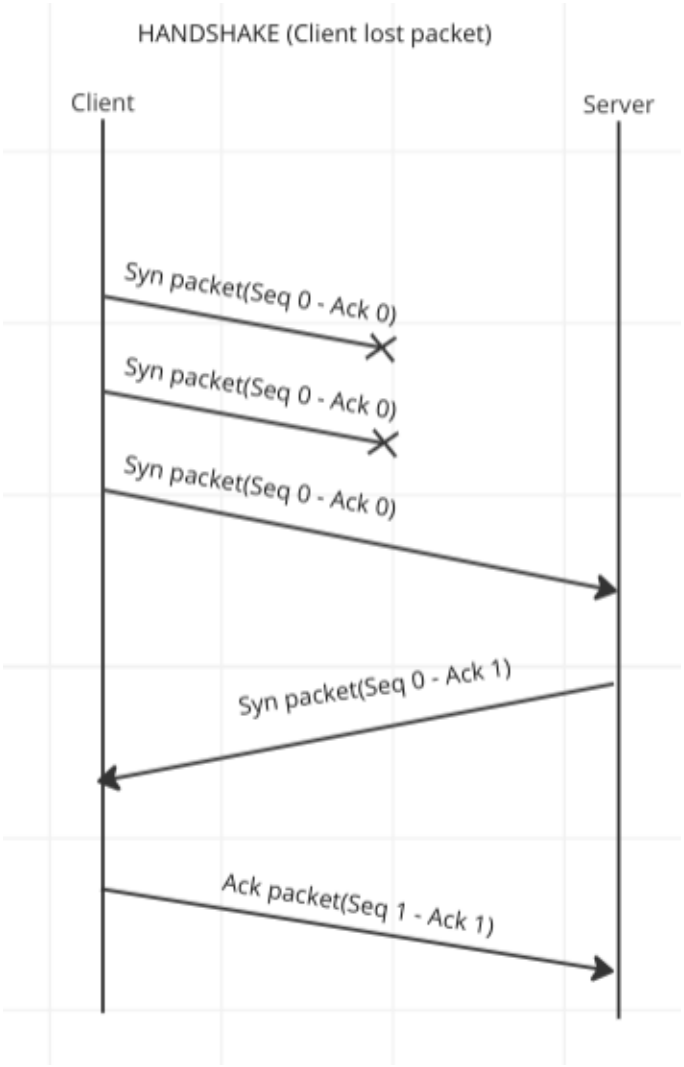

Figura 2: Perdida del paquete SYN

En caso de que se pierda el paquete de sincronización del lado del cliente, lo muestra la figura 2.

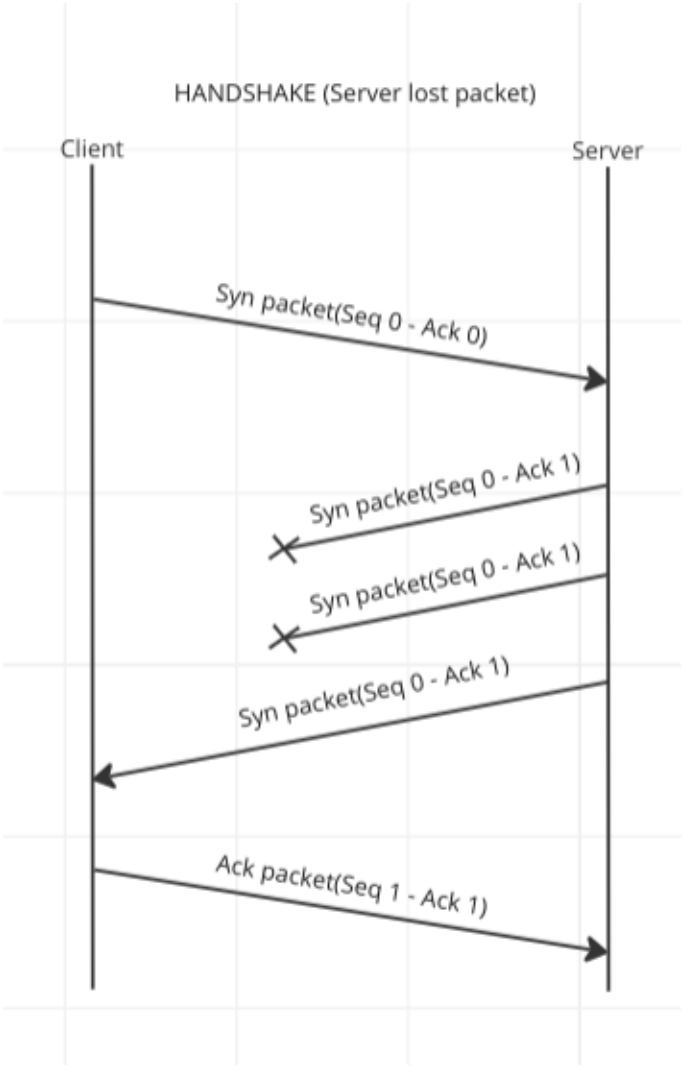

Figura 3: Perdida del paquete SYN

En caso de que se pierda el paquete de sincronización del lado del servidor, lo muestra la figura 3.

3.0.1. Paquetes

Los paquetes tienen una estructura de tamaño fijo que contiene:

- Sequence Number (Número de Secuencia): Este campo tiene una longitud de 4 bytes y se utiliza para indicar el número de secuencia del paquete. Cada paquete enviado tiene un número de secuencia único que se utiliza para ordenarlos.
- Acknowledgment Number (Número de Reconocimiento): También tiene una longitud de 4 bytes y se utiliza para indicar el número de secuencia que el receptor espera recibir a continuación. Ayuda en la confirmación de la recepción de paquetes y la detección de paquetes perdidos.
- Flags (Banderas): Este campo tiene una longitud de 1 byte y contiene bits de control que indican el propósito del paquete. Los bits de las banderas se utilizan para marcar si el paquete es un ACK (acknowledgment), un SYN (synchronize), un FIN (finish), o si se trata de una operación de carga o descarga. Cada bit de este byte se utiliza para una bandera específica.
- Payload (Carga Útil): La longitud de la carga útil puede variar según el tamaño de los datos que se están transmitiendo. En nuestra implementación, se utiliza un tamaño de carga útil de 4087 bytes. La carga útil contiene la información real que se está enviando, como los datos de un archivo.

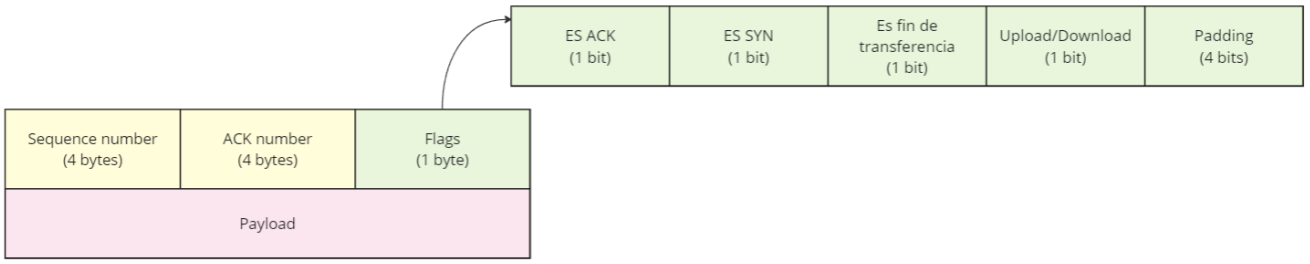

Figura 4: *Formato del paquete*

3.0.2. Protocolo Stop and Wait

El protocolo Stop and Wait es un protocolo de comunicación simple y eficiente utilizado en la capa de transporte para garantizar la entrega confiable de datos. Se basa en el principio de enviar un paquete y esperar una confirmación (acknowledgment) antes de enviar el siguiente paquete. El protocolo consta de los siguientes elementos clave:

- ACK (Acknowledgment): Un paquete de ACK es enviado por el receptor para confirmar la recepción exitosa de un paquete.
- SYN (Synchronize): Un paquete SYN se utiliza para establecer una conexión entre el cliente y el servidor.
- FIN (Finish): Un paquete FIN se utiliza para finalizar una conexión de manera ordenada.
- Retransmisiones: Si un paquete se pierde o no se recibe correctamente, se realiza una retransmisión para garantizar la entrega.

El protocolo Stop and Wait se implementa en la clase StopAndWait. A continuación, se describen las principales funciones relacionadas con el protocolo:

- initializeHandshake(): Inicia el proceso de conexión entre el cliente y el servidor mediante el intercambio de paquetes SYN.
- sendPacket(): Envía un paquete al servidor y espera un ACK como confirmación. Realiza retransmisiones en caso de timeout o pérdida de paquetes.
- recvPacket(): Recibe un paquete del servidor y confirma su recepción mediante un ACK. Maneja retransmisiones en caso de problemas de recepción.

En el servidor, el protocolo Stop and Wait se utiliza para la comunicación con cada cliente. El servidor maneja múltiples conexiones de clientes de manera concurrente utilizando threads.

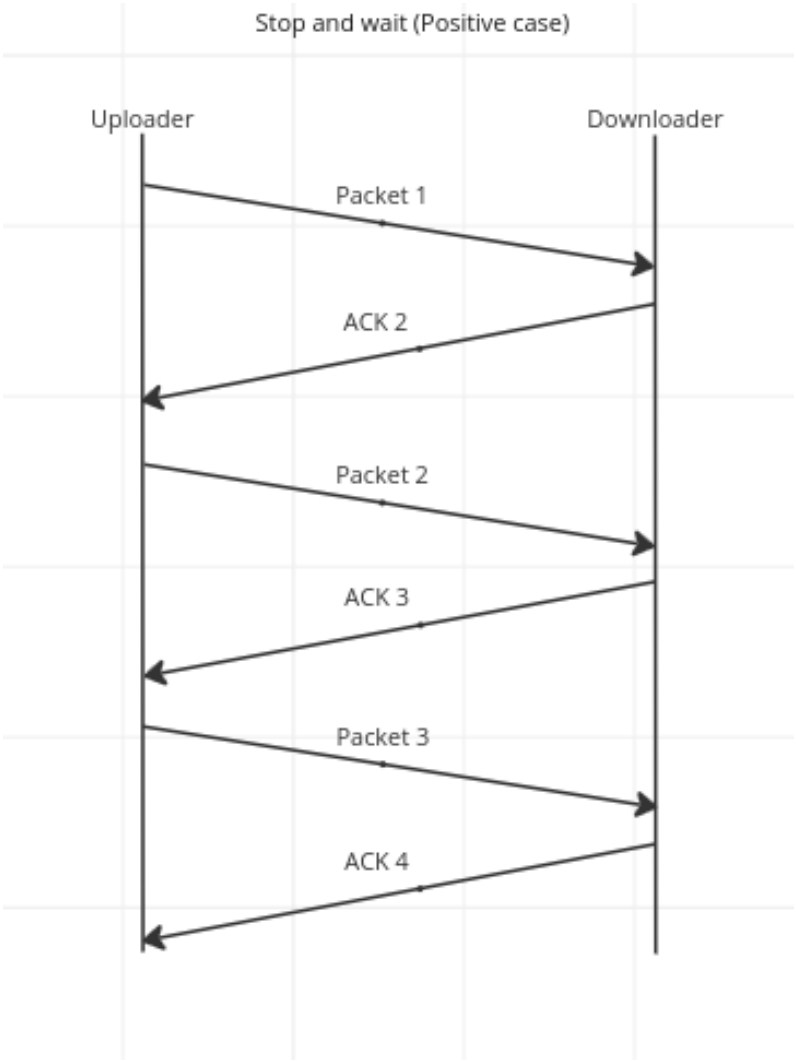

Figura 5: Caso en el que no se pierde ningun paquete y llegan los ack correctamente

En la figura 4 se muestra el funcionamiento de la comunicación del protocolo Stop and Wait.

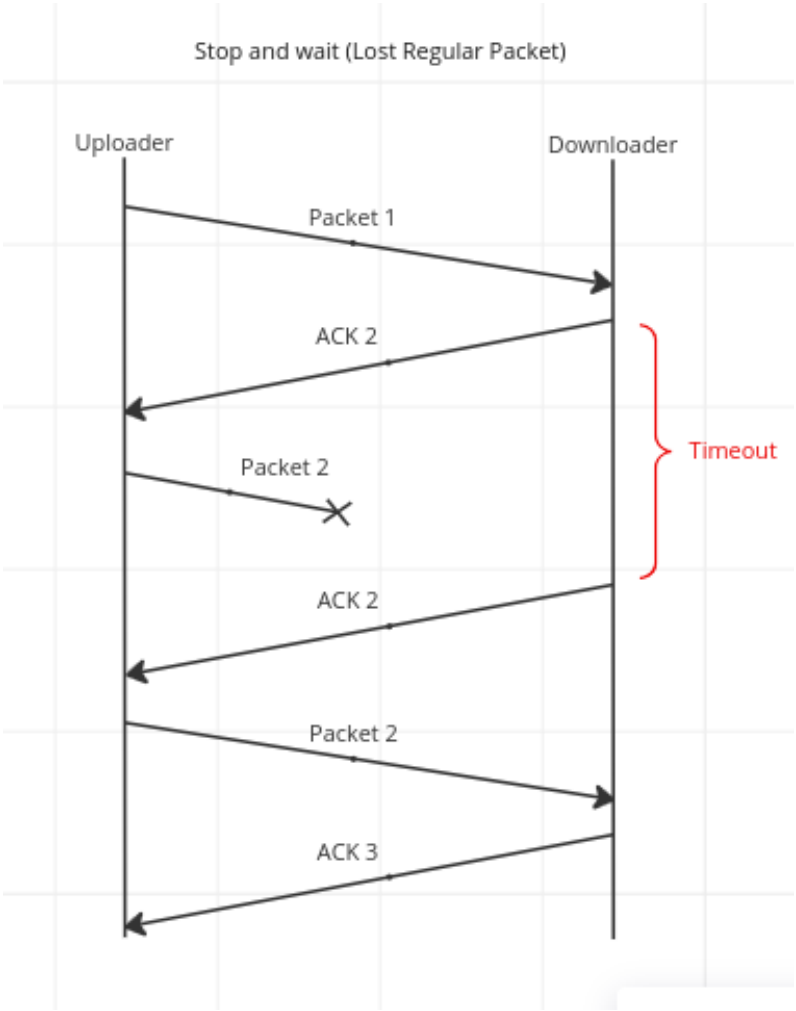

Figura 6: Caso en el que se pierde un paquete regular

En la figura 5 se muestra como al fallar el envio de un paquete, el protocolo espera un tiempo de timeout para reenviar el ack correspondiente.

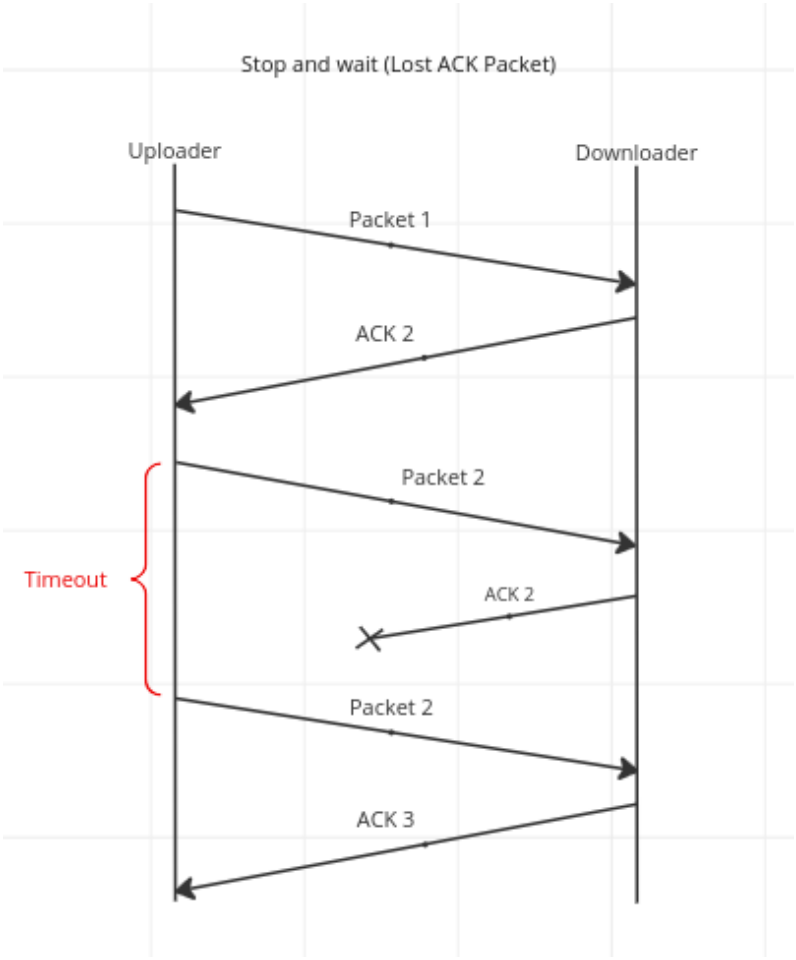

Figura 7: Caso en el que se pierde un paquete ACK

En la figura 6 muestra cuando se pierde un paquete ACK en el protocolo Stop and Wait.

3.0.3. Protocolo Selective Repeat

El protocolo Selective Repeat es un protocolo de control de flujo en la capa de transporte que permite la retransmisión de paquetes individuales que se pierden o se corrompen en una red de comunicación

- El constructor init inicializa las variables necesarias para el protocolo, como la dirección, números de secuencia y ack, y las estructuras de datos para el seguimiento de paquetes.
- initializeHandshake y responseHandshake se utilizan para establecer una conexión inicial entre el cliente y el servidor. Esto implica el intercambio de paquetes SYN y ACK para sincronizar los números de secuencia.
- send packet se utilizan para enviar paquetes desde el servidor y el cliente, respectivamente.
- La función send packet verifica si el tamaño de la ventana de envío (WINDOW SIZE) se ha superado. Si es así, espera la recepción de los ACK correspondientes para liberar espacio en la ventana antes de enviar más paquetes.
- El método wait ack se utiliza para esperar ACKs y manejar la retransmisión si no se recibe un ACK válido dentro de un límite establecido.
- recv packet se utiliza para recibir paquetes en el servidor y el cliente, respectivamente.
- Estas funciones manejan la espera de paquetes y envían ACKs correspondientes.
- Si se reciben paquetes fuera de orden, se almacenan en recv buffer hasta que llegue el paquete esperado.
- Si se supera el límite de reintentos (RETRIES), se maneja la retransmisión o se lanza una excepción.
- wait last packets se utilizan para esperar la confirmación de los últimos paquetes en el servidor y el cliente antes de finalizar la conexión. Esto asegura que todos los paquetes se entreguen correctamente antes de cerrar la conexión

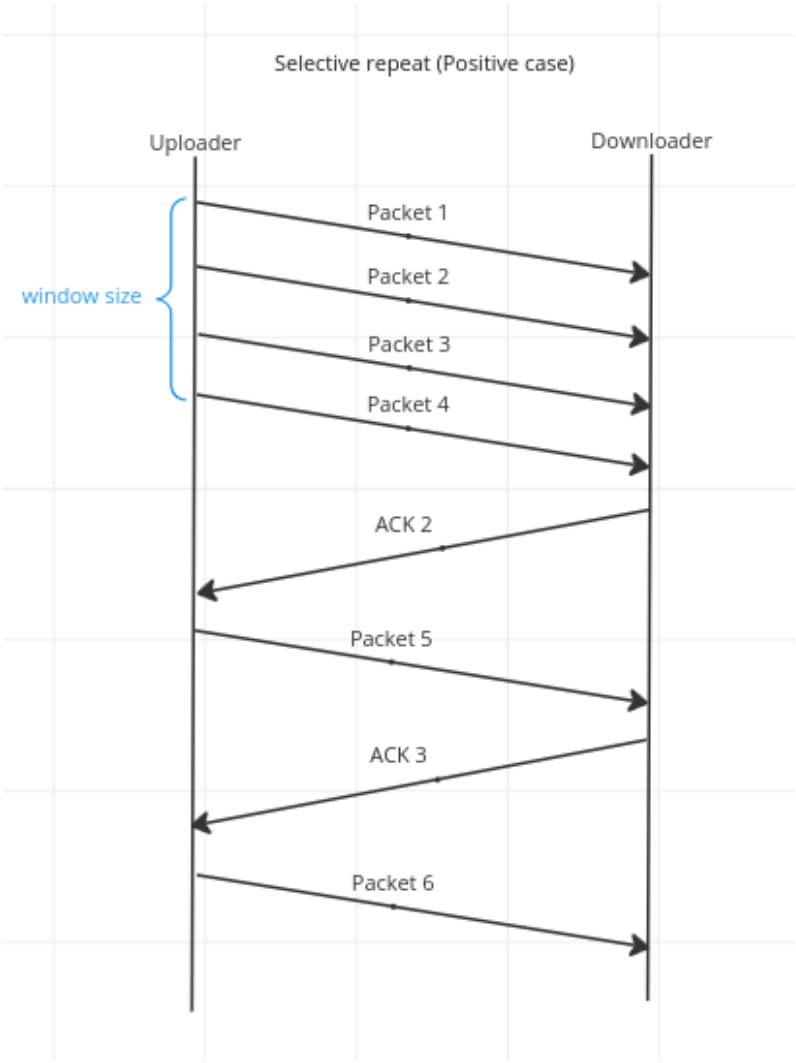

Figura 8: Comunicación de protocolo Selective Repeat

En la figura 7 se muestra un diagrama de la comunicación utilizando el protocolo Selective Repeat, sin que se pierdan paquetes.

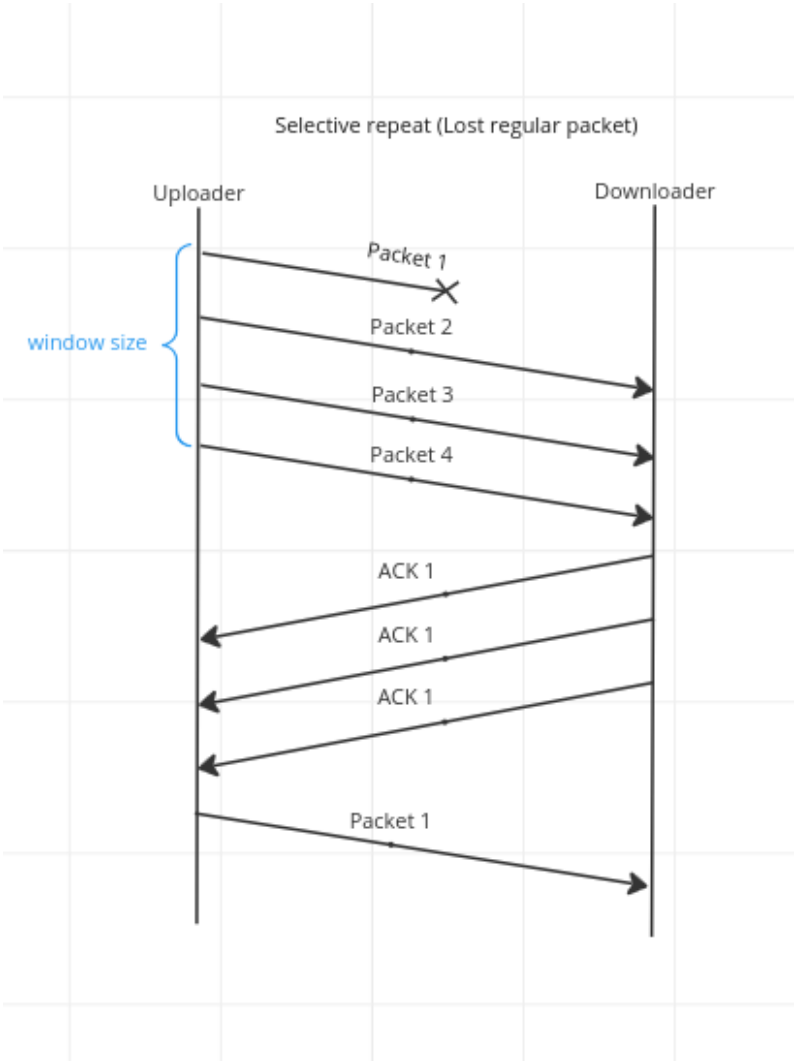

Figura 9: Caso en el que se pierde un paquete regular usando Selective Repeat

En la figura 8 se ve como se envian N paquetes, siendo N el tamaño de la ventana de recepción, con la pérdida de un paquete regular.

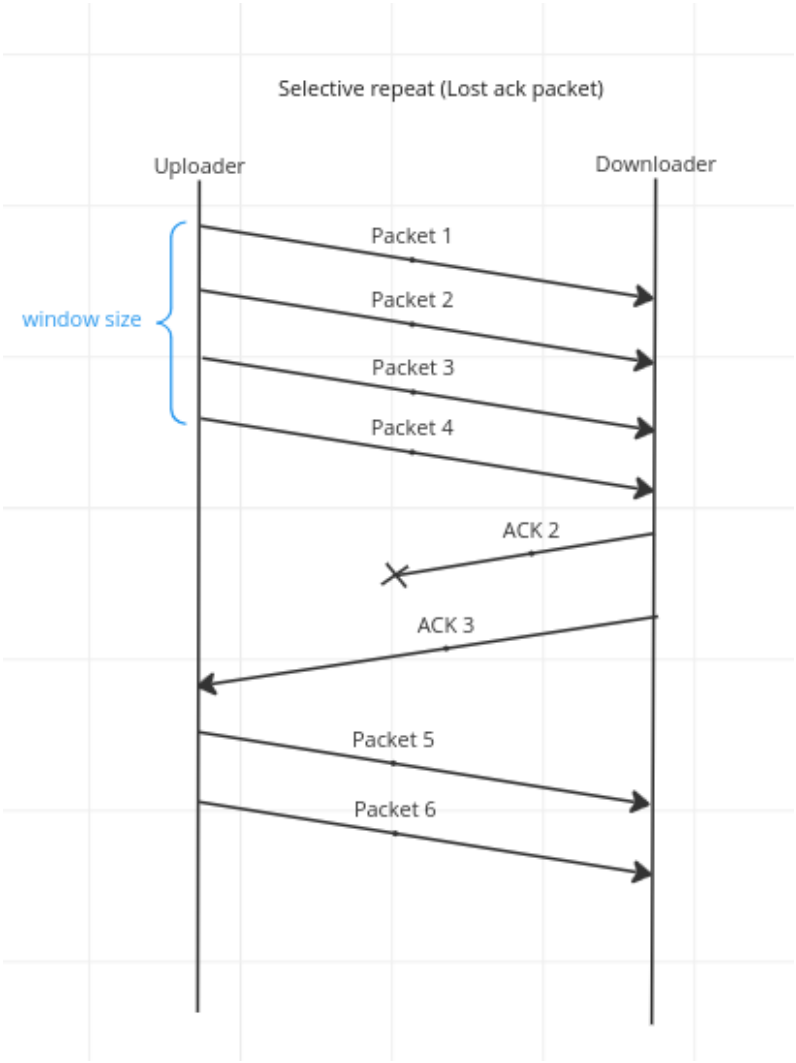

Figura 10: Caso en el que no llega un ACK correctamente usando Selective Repeat

En la figura 9 se ve como se envian N paquetes, siendo N el tamaño de la ventana de recepción, con la pérdida de un ACK.

4. Pruebas

4.1. Subir un archivo usando el protocolo Stop and Wait

Para poder subir un archivo usando nuestra aplicación con el protocolo Stop and Wait, primero se levanta el servidor con un flag de su respectivo protocolo de esta manera:

```
franvm@ubuntu:~/Desktop/Redes/TP1-File-Transfer/src$ python3 start_server.py -v -H 127.0.0.1
 -p 65432 -pr saw -s /home/franvm/Desktop/Redes/TP1-File-Transfer/server_files/
[2023/10/02 15:32:23] - [server DEBUG] - Setting DEBUG log level
[2023/10/02 15:32:23] - [server INFO] - Server starting up on: ('127.0.0.1', 65432)
```

Figura 11: *Levantar servidor usando Stop and Wait*

Desde el lado del cliente, se utiliza este comando para subir un archivo usando el protocolo Stop and Wait:

```
franvm@ubuntu:~/Desktop/Redes/TP1-File-Transfer/src$ python3 upload.py -v -H 127.0.0.1 -p 654
32 -n roman.png -s "/home/franvm/Desktop/Redes/TP1-File-Transfer/client_files/" -pr "saw"
[2023/10/02 15:33:04] - [upload DEBUG] - Setting DEBUG log level
[2023/10/02 15:33:04] - [upload INFO] - Client up
[2023/10/02 15:33:04] - [upload INFO] - Uploading /home/franvm/Desktop/Redes/TP1-File-Transfe
r/client_files/ to Server with file name roman.png
[2023/10/02 15:33:04] - [upload INFO] - Handshake initiated
[2023/10/02 15:33:04] - [upload INFO] - Sending first syn with sequence number: 0
sequence_number: 0,  is_ack: False, payload_len: 0, ack_number : 1
[2023/10/02 15:33:04] - [upload INFO] - Received syn response with sequence number: 0 and ack
: 1
[2023/10/02 15:33:04] - [upload INFO] - Connection established
[2023/10/02 15:33:04] - [upload INFO] - File size to be sent: 4096 bytes
[2023/10/02 15:33:04] - [upload DEBUG] - Preparing packet to ('127.0.0.1', 65432)
[2023/10/02 15:33:04] - [upload DEBUG] - Packet sent as (sequence_number: 1,  is_ack: False,
payload_len: 4087, ack_number : 1)
[2023/10/02 15:33:04] - [upload DEBUG] - Received packet as sequence_number: 1,  is_ack: True
, payload_len: 0, ack_number : 2
[2023/10/02 15:33:04] - [upload DEBUG] - Preparing packet to ('127.0.0.1', 65432)
[2023/10/02 15:33:04] - [upload DEBUG] - Packet sent as (sequence_number: 2,  is_ack: False,
payload_len: 4087, ack_number : 2)
[2023/10/02 15:33:04] - [upload DEBUG] - Timeout event occurred on send. Retrying...
[2023/10/02 15:33:04] - [upload DEBUG] - Packet sent as (sequence_number: 2,  is_ack: False,
payload_len: 4087, ack_number : 2)
[2023/10/02 15:33:04] - [upload DEBUG] - Received packet as sequence_number: 3,  is_ack: True
, payload_len: 0, ack_number : 3
```

Figura 12: *Conectar cliente con el servidor*

Ahora desde el lado del servidor podemos observar el handshake (conexión entre el cliente y servidor):

```
franvm@ubuntu:~/Desktop/Redes/TP1-File-Transfer/src$ python3 start_server.py -v -H 127.0.0.1
 -p 65432 -pr saw -s /home/franvm/Desktop/Redes/TP1-File-Transfer/server_files/
[2023/10/02 15:32:23] - [server DEBUG] - Setting DEBUG log level
[2023/10/02 15:32:23] - [server INFO] - Server starting up on: ('127.0.0.1', 65432)
[2023/10/02 15:32:41] - [server INFO] - Received first syn with sequence number: 0
[2023/10/02 15:32:41] - [server INFO] - Sending second syn with sequence number: 0 and ack:
1
[2023/10/02 15:32:41] - [server INFO] - Created new handler with key ('127.0.0.1', 49694)
[2023/10/02 15:32:41] - [server DEBUG] - Received packet as sequence_number: 1,  is_ack: Fal
se, payload_len: 4087, ack_number : 1
[2023/10/02 15:32:41] - [server DEBUG] - Sent ack as sequence_number: 1,  is_ack: True, payl
oad_len: 0, ack_number : 2
```

Figura 13: *Conectar servidor con el cliente*

Finalmente vemos como se envian y se reciben los paquetes del archivo, con sus respectivos timeouts y retransmisiones:

```
 payload_len: 4087, ack_number : 34)
[2023/10/02 15:33:05] - [upload DEBUG] - Received packet as sequence_number: 35,  is_ack: Tru
e, payload_len: 0, ack_number : 35
[2023/10/02 15:33:05] - [upload DEBUG] - Preparing packet to ('127.0.0.1', 65432)
[2023/10/02 15:33:05] - [upload DEBUG] - Packet sent as (sequence_number: 35,  is_ack: False,
 payload_len: 4087, ack_number : 36)
[2023/10/02 15:33:05] - [upload DEBUG] - Received packet as sequence_number: 35,  is_ack: Tru
e, payload_len: 0, ack_number : 36
[2023/10/02 15:33:05] - [upload DEBUG] - Preparing packet to ('127.0.0.1', 65432)
[2023/10/02 15:33:05] - [upload DEBUG] - Packet sent as (sequence_number: 36,  is_ack: False,
 payload_len: 4087, ack_number : 36)
[2023/10/02 15:33:05] - [upload DEBUG] - Timeout event occurred on send. Retrying...
[2023/10/02 15:33:05] - [upload DEBUG] - Packet sent as (sequence_number: 36,  is_ack: False,
 payload_len: 4087, ack_number : 36)
[2023/10/02 15:33:05] - [upload DEBUG] - Received packet as sequence_number: 36,  is_ack: Tru
e, payload_len: 0, ack_number : 37
[2023/10/02 15:33:05] - [upload DEBUG] - Preparing packet to ('127.0.0.1', 65432)
[2023/10/02 15:33:05] - [upload DEBUG] - Packet sent as (sequence_number: 37,  is_ack: False,
 payload_len: 2128, ack_number : 37)
[2023/10/02 15:33:05] - [upload DEBUG] - Received packet as sequence_number: 37,  is_ack: Tru
e, payload_len: 0, ack_number : 38
[2023/10/02 15:33:05] - [upload DEBUG] - Preparing packet to ('127.0.0.1', 65432)
[2023/10/02 15:33:05] - [upload DEBUG] - Packet sent as (sequence_number: 38,  is_ack: False,
 payload_len: 0, ack_number : 38)
[2023/10/02 15:33:05] - [upload DEBUG] - Received packet as sequence_number: 38,  is_ack: Tru
e, payload_len: 0, ack_number : 39
[2023/10/02 15:33:05] - [upload INFO] - Upload completed for file: roman.png
[2023/10/02 15:33:05] - [upload INFO] - Total transfer time:  1.58 seconds.
[2023/10/02 15:33:05] - [upload INFO] - Average transfer speed: 92.32 KB/s
[2023/10/02 15:33:05] - [upload INFO] - Total number of packets sent: 37
```

Figura 14: *Envio de paquetes, retransmisiones y upload completado del cliente*

Al terminar de subir el archivo, se observan las estadisticas que tuvo la transferencia de datos.

Por el lado del servidor se ve:

```
[2023/10/02 15:33:05] - [server DEBUG] - Sent ack as sequence_number: 34,  is_ack: True, pay
load_len: 0, ack_number : 35
[2023/10/02 15:33:05] - [server DEBUG] - Timeout event occurred on receive. Retrying...
[2023/10/02 15:33:05] - [server DEBUG] - Received packet as sequence_number: 34,  is_ack: Fa
lse, payload_len: 4087, ack_number : 34
[2023/10/02 15:33:05] - [server ERROR] - Ack number is not correct
[2023/10/02 15:33:05] - [server DEBUG] - Sent ack as sequence_number: 35,  is_ack: True, pay
load_len: 0, ack_number : 35
[2023/10/02 15:33:05] - [server DEBUG] - Received packet as sequence_number: 35,  is_ack: Fa
lse, payload_len: 4087, ack_number : 36
[2023/10/02 15:33:05] - [server DEBUG] - Sent ack as sequence_number: 35,  is_ack: True, pay
load_len: 0, ack_number : 36
[2023/10/02 15:33:05] - [server DEBUG] - Timeout event occurred on receive. Retrying...
[2023/10/02 15:33:05] - [server DEBUG] - Received packet as sequence_number: 36,  is_ack: Fa
lse, payload_len: 4087, ack_number : 36
[2023/10/02 15:33:05] - [server DEBUG] - Sent ack as sequence_number: 36,  is_ack: True, pay
load_len: 0, ack_number : 37
[2023/10/02 15:33:05] - [server DEBUG] - Received packet as sequence_number: 37,  is_ack: Fa
lse, payload_len: 2128, ack_number : 37
[2023/10/02 15:33:05] - [server DEBUG] - Sent ack as sequence_number: 37,  is_ack: True, pay
load_len: 0, ack_number : 38
[2023/10/02 15:33:05] - [server DEBUG] - Received packet as sequence_number: 38,  is_ack: Fa
lse, payload_len: 0, ack_number : 38
[2023/10/02 15:33:05] - [server DEBUG] - Sent ack as sequence_number: 38,  is_ack: True, pay
load_len: 0, ack_number : 39
[2023/10/02 15:33:05] - [server INFO] - Upload completed
```

Figura 15: *Upload completo desde el servidor*

4.2. Descargar un archivo usando el protocolo Stop and Wait

Ahora para descargar un archivo desde el servidor usando el protocolo Stop and Wait, para levantar el servidor se hace exactamente de la misma manera que para el upload.

Desde el lado del cliente usamos el siguiente comando:

```
franvm@ubuntu:~/Desktop/Redes/TP1-File-Transfer/src$ python3 download.py -v -H 127.0.0.1 -p 65
432 -n roman.png -d "/home/franvm/Desktop/Redes/TP1-File-Transfer/client_files/sr" -pr "saw"
[2023/10/01 19:54:08] - [download DEBUG] - Setting DEBUG log level
[2023/10/01 19:54:08] - [download INFO] - Downloading roman.png from Server to /home/franvm/De
sktop/Redes/TP1-File-Transfer/client_files/srroman.png
[2023/10/01 19:54:08] - [download INFO] - Handshake initiated
[2023/10/01 19:54:08] - [download INFO] - Sending first syn with sequence number: 0
sequence_number: 0,  is_ack: False, payload_len: 0, ackNumber : 1
[2023/10/01 19:54:08] - [download INFO] - Received syn response with sequence number: 0 and ac
k: 1
[2023/10/01 19:54:08] - [download INFO] - Client up
[2023/10/01 19:54:08] - [download INFO] - Connection established
[2023/10/01 19:54:08] - [download DEBUG] - Received packet as sequence_number: 1,  is_ack: Fal
se, payload_len: 4087, ackNumber : 1
[2023/10/01 19:54:08] - [download DEBUG] - Sent ack as sequence_number: 1,  is_ack: True, payl
oad_len: 0, ackNumber : 2
```

Figura 16: *Conexión cliente con el servidor para descarga*

Luego vemos como se completa la descarga del archivo:

```
[2023/10/01 19:54:09] - [download DEBUG] - Sent ack as sequence_number: 35,  is_ack: True, pay
load_len: 0, ackNumber : 36
[2023/10/01 19:54:09] - [download DEBUG] - Timeout event occurred on receive. Retrying...
[2023/10/01 19:54:09] - [download DEBUG] - Received packet as sequence_number: 35,  is_ack: Fa
lse, payload_len: 4087, ackNumber : 36
[2023/10/01 19:54:09] - [download ERROR] - Ack number is not correct
[2023/10/01 19:54:09] - [download DEBUG] - Sent ack as sequence_number: 36,  is_ack: True, pay
load_len: 0, ackNumber : 36
[2023/10/01 19:54:09] - [download DEBUG] - Received packet as sequence_number: 36,  is_ack: Fa
lse, payload_len: 4087, ackNumber : 37
[2023/10/01 19:54:09] - [download DEBUG] - Sent ack as sequence_number: 36,  is_ack: True, pay
load_len: 0, ackNumber : 37
[2023/10/01 19:54:09] - [download DEBUG] - Received packet as sequence_number: 37,  is_ack: Fa
lse, payload_len: 2128, ackNumber : 37
[2023/10/01 19:54:09] - [download DEBUG] - Sent ack as sequence_number: 37,  is_ack: True, pay
load_len: 0, ackNumber : 38
[2023/10/01 19:54:09] - [download DEBUG] - Timeout event occurred on receive. Retrying...
[2023/10/01 19:54:09] - [download DEBUG] - Timeout event occurred on receive. Retrying...
[2023/10/01 19:54:09] - [download DEBUG] - Received packet as sequence_number: 38,  is_ack: Fa
lse, payload_len: 0, ackNumber : 38
[2023/10/01 19:54:09] - [download DEBUG] - Sent ack as sequence_number: 38,  is_ack: True, pay
load_len: 0, ackNumber : 39
[2023/10/01 19:54:09] - [download INFO] - Download completed for file: roman.png. It was saved
 in /home/franvm/Desktop/Redes/TP1-File-Transfer/client_files/sr
[2023/10/01 19:54:09] - [download INFO] - Total transfer time:  1.13 seconds.
[2023/10/01 19:54:09] - [download INFO] - Average transfer speed: 128.66 KB/s
[2023/10/01 19:54:09] - [download INFO] - Total number of packets sent: 37
```

Figura 17: *Descarga completada desde el lado del cliente*

Y desde el lado del servidor se observa:

```
ue, payload_len: 0, ackNumber : 36
[2023/10/01 19:54:09] - [server DEBUG] - Preparing packet to ('127.0.0.1', 49361)
[2023/10/01 19:54:09] - [server DEBUG] - Packet sent as (sequence_number: 36,  is_ack: False
, payload_len: 4087, ackNumber : 37)
[2023/10/01 19:54:09] - [server DEBUG] - Received packet as sequence_number: 36,  is_ack: Tr
ue, payload_len: 0, ackNumber : 37
[2023/10/01 19:54:09] - [server DEBUG] - Preparing packet to ('127.0.0.1', 49361)
[2023/10/01 19:54:09] - [server DEBUG] - Packet sent as (sequence_number: 37,  is_ack: False
, payload_len: 2128, ackNumber : 37)
[2023/10/01 19:54:09] - [server DEBUG] - Received packet as sequence_number: 37,  is_ack: Tr
ue, payload_len: 0, ackNumber : 38
[2023/10/01 19:54:09] - [server DEBUG] - Preparing packet to ('127.0.0.1', 49361)
[2023/10/01 19:54:09] - [server DEBUG] - Packet sent as (sequence_number: 38,  is_ack: False
, payload_len: 0, ackNumber : 38)
[2023/10/01 19:54:09] - [server DEBUG] - Timeout event occurred on send. Retrying...
[2023/10/01 19:54:09] - [server DEBUG] - Packet sent as (sequence_number: 38,  is_ack: False
, payload_len: 0, ackNumber : 38)
[2023/10/01 19:54:09] - [server DEBUG] - Timeout event occurred on send. Retrying...
[2023/10/01 19:54:09] - [server DEBUG] - Packet sent as (sequence_number: 38,  is_ack: False
, payload_len: 0, ackNumber : 38)
[2023/10/01 19:54:09] - [server DEBUG] - Received packet as sequence_number: 38,  is_ack: Tr
ue, payload_len: 0, ackNumber : 39
[2023/10/01 19:54:09] - [server INFO] - Download completed
```

Figura 18: *Descarga completada desde el lado del servidor*

4.3. Subir archivo usando el protocolo Selective Repeat

Siguiendo los mismos pasos, para levantar un servidor usando el protocolo Selective Repeat:

```
franvm@ubuntu:~/Desktop/Redes/TP1-File-Transfer/src$ python3 start_server.py -v -H 127.0.0.1
 -p 65432 -pr "sr" -s "/home/franvm/Desktop/Redes/TP1-File-Transfer/server_files/"
[2023/10/01 20:11:51] - [server DEBUG] - Setting DEBUG log level
[2023/10/01 20:11:51] - [server INFO] - Server starting up on: ('127.0.0.1', 65432)
[2023/10/01 20:11:51] - [server INFO] - Server using sr protocol
```

Figura 19: *Servidor levantado con Selective Repeat*

Luego con el mismo comando que antes, pero usando el flag "sr", se sube un archivo al servidor, desde el lado del cliente:

```
franvm@ubuntu:~/Desktop/Redes/TP1-File-Transfer/src$ python3 upload.py -v -H 127.0.0.1 -p 6543
2 -n roman.png -s "/home/franvm/Desktop/Redes/TP1-File-Transfer/client_files/" -pr "sr"
[2023/10/01 20:14:22] - [upload DEBUG] - Setting DEBUG log level
[2023/10/01 20:14:22] - [upload INFO] - Client up
[2023/10/01 20:14:22] - [upload INFO] - Uploading /home/franvm/Desktop/Redes/TP1-File-Transfer
/client_files/ to Server with file name roman.png
[2023/10/01 20:14:22] - [upload INFO] - Handshake initiated
[2023/10/01 20:14:22] - [upload INFO] - Sending first syn with sequence number: 0
sequence_number: 0,  is_ack: False, payload_len: 0, ackNumber : 1
[2023/10/01 20:14:22] - [upload INFO] - Received syn response with sequence number: 0 and ack:
 1
[2023/10/01 20:14:22] - [upload INFO] - Connection established
[2023/10/01 20:14:22] - [upload INFO] - File size to be sent: 4096 bytes
[2023/10/01 20:14:22] - [upload DEBUG] - Packet sent as (sequence_number: 1,  is_ack: False, p
ayload_len: 4087, ackNumber : 1)
[2023/10/01 20:14:22] - [upload DEBUG] - Packet sent as (sequence_number: 2,  is_ack: False, p
ayload_len: 4087, ackNumber : 1)
```

Figura 20: *Subida de archivo usando Selective Repeat*

Finalmente desde el lado del cliente se ve como finaliza la subida del archivo:

```
[2023/10/01 20:15:24] - [upload DEBUG] - Packet sent as (sequence_number: 38,  is_ack: False,
payload_len: 0, ackNumber : 1)
river
[2023/10/01 20:15:24] - [upload DEBUG] - Received packet as sequence_number: 1,  is_ack: True,
 payload_len: 0, ackNumber : 36
dict_keys([35, 36, 37, 38])
[2023/10/01 20:15:24] - [upload DEBUG] - Received packet as sequence_number: 1,  is_ack: True,
 payload_len: 0, ackNumber : 37
dict_keys([36, 37, 38])
[2023/10/01 20:15:24] - [upload DEBUG] - Received packet as sequence_number: 1,  is_ack: True,
 payload_len: 0, ackNumber : 37
dict_keys([37, 38])
[2023/10/01 20:15:24] - [upload DEBUG] - Timeout event occurred on send. Retrying...
[2023/10/01 20:15:24] - [upload DEBUG] - Timeout event occurred on send. Retrying...
[2023/10/01 20:15:24] - [upload DEBUG] - Retransmitted packet sent as (sequence_number: 37,  i
s_ack: False, payload_len: 2128, ackNumber : 1)
[2023/10/01 20:15:24] - [upload DEBUG] - Received packet as sequence_number: 1,  is_ack: True,
 payload_len: 0, ackNumber : 38
dict_keys([37, 38])
[2023/10/01 20:15:24] - [upload DEBUG] - Received packet as sequence_number: 1,  is_ack: True,
 payload_len: 0, ackNumber : 39
dict_keys([38])
[2023/10/01 20:15:24] - [upload INFO] - Upload completed for file: roman.png
[2023/10/01 20:15:24] - [upload INFO] - Total transfer time:  0.21 seconds.
[2023/10/01 20:15:24] - [upload INFO] - Average transfer speed: 693.95 KB/s
[2023/10/01 20:15:24] - [upload INFO] - Total number of packets sent: 37
```

Figura 21: *Subida de archivo completada usando Selective Repeat*

4.4. Descargar un archivo usando el protocolo Selective Repeat

De igual manera, para subir un archivo usando Selective Repeat, se especifica con el flag "sr":

```
franvm@ubuntu:~/Desktop/Redes/TP1-File-Transfer/src$ python3 download.py -v -H 127.0.0.1 -p 65
432 -n roman.png -d "/home/franvm/Desktop/Redes/TP1-File-Transfer/client_files/sr" -pr "sr"
[2023/10/01 20:18:27] - [download DEBUG] - Setting DEBUG log level
[2023/10/01 20:18:27] - [download INFO] - Downloading roman.png from Server to /home/franvm/De
sktop/Redes/TP1-File-Transfer/client_files/srroman.png
[2023/10/01 20:18:27] - [download INFO] - Handshake initiated
[2023/10/01 20:18:27] - [download INFO] - Sending first syn with sequence number: 0
sequence_number: 0,  is_ack: False, payload_len: 0, ackNumber : 1
[2023/10/01 20:18:27] - [download INFO] - Received syn response with sequence number: 0 and ac
k: 1
[2023/10/01 20:18:27] - [download INFO] - Client up
[2023/10/01 20:18:27] - [download INFO] - Connection established
[2023/10/01 20:18:27] - [download DEBUG] - Received packet as sequence_number: 1,  is_ack: Fal
se, payload_len: 4087, ackNumber : 1
[2023/10/01 20:18:27] - [download DEBUG] - Sent ack as (sequence_number: 1,  is_ack: True, pay
load_len: 0, ackNumber : 2)
[2023/10/01 20:18:27] - [download DEBUG] - Received packet as sequence_number: 2,  is_ack: Fal
se, payload_len: 4087, ackNumber : 1
```

Figura 22: *Descarga de archivo usando Selective Repeat*

Se ve la conexión establecida con el servidor y como empieza a mandar y recibir paquetes.

Al final se muestran las estadisticas de la descarga y la cantidad de paquetes totales que se enviaron.

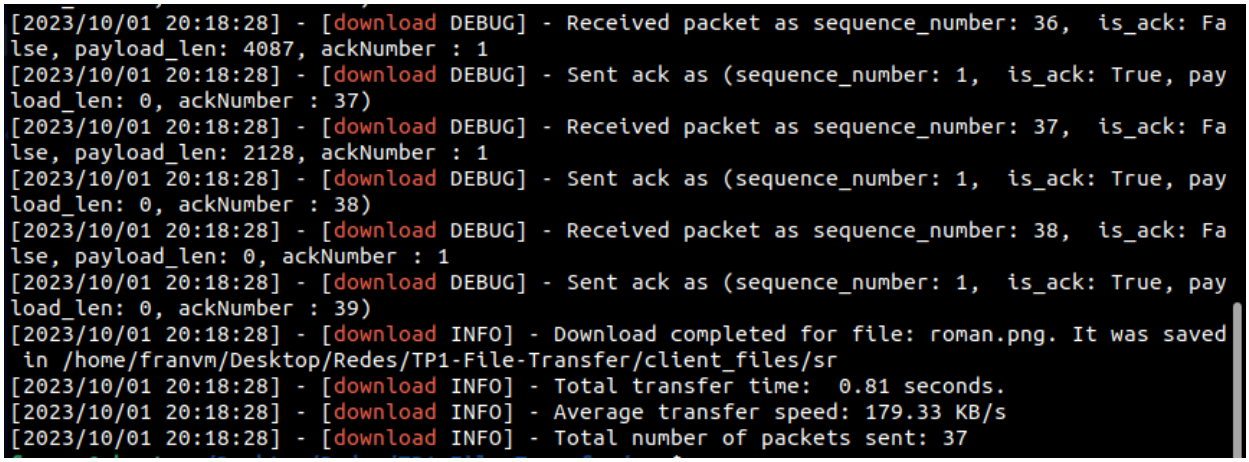

Figura 23: *Descarga de archivo completada usando Selective Repeat*

5. Análisis

Packet Loss	Operación	Tiempo SR [s]	Tiempo SAW [s]
0	Upload	0.01	0.01
10	Upload	0.42	0.93
0	Download	0.01	0.01
10	Download	0.32	0.42

Cuadro 1: *Archivo liviano 150kb*

Packet Loss	Operación	Tiempo SR [s]	Tiempo SAW [s]
0	Upload	0.03	0.05
10	Upload	5.63	9.76
0	Download	0.03	0.05
10	Download	5.08	9.44

Cuadro 2: *Archivo mediano 1.5Mb*

Packet Loss	Operación	Tiempo SR [s]	Tiempo SAW [s]
0	Upload	0.57	0.72
10	Upload	22.82	53.09
0	Download	0.59	0.67
10	Download	19.11	50.25

Cuadro 3: *Archivo pesado 8Mb*

6. Preguntas

6.0.1. Describa la arquitectura Cliente-Servidor

La arquitectura Cliente-Servidor es un modelo computacional en el que un host activo (servidor) ofrece su servicio a muchos hosts (clientes) para que utilicen sus recursos.

Los clientes y servidores interactúan a través de una red de redes, como Internet. En nuestra aplicación, los clientes envían solicitudes (mensajes) al servidor y esperan su respuesta. El servidor procesa los mensajes y responde.

6.0.2. ¿Cuál es la función de un protocolo de capa de aplicación?

Un protocolo de capa de aplicación es un conjunto de reglas y convenciones que permiten a las aplicaciones en diferentes dispositivos (sistemas terminales) comunicarse entre sí a través de una red. Sus funciones principales son:

- Definición de Formato de Datos: Define el formato en el que los datos deben ser empaquetados y transmitidos.
- Control de Sesiones: Administra la creación, el mantenimiento y la terminación de sesiones de comunicación.
- Gestión de Errores y Seguridad: Proporciona mecanismos para la detección y corrección de errores, así como medidas de seguridad como autenticación y cifrado.
- Establecimiento de Conexiones: Permite la creación de conexiones entre aplicaciones en dispositivos remotos.

6.0.3. Detalle el protocolo de aplicación desarrollado en este trabajo.

En nuestro trabajo desarrollamos un protocolo de aplicación personalizado que se utiliza para la transferencia de archivos entre el cliente y el servidor.

Tiene como base dos operaciones: 'upload' para la subida de archivos desde un cliente al servidor y 'download' para la descarga de archivos de un cliente desde el servidor.

Además se implementaron dos protocolos para garantizar la confiabilidad de la transferencia de datos, 'Stop and Wait' y 'Selective Repeat', en donde se utilizan procesos de handshake para establecer una conexión entre el cliente y el servidor, se usan paquetes con campos específicos (numero de secuencia, ack, etc) explicado en la sección 3.0.1 'Paquetes', y se implementaron reintentos de envío de paquetes en caso de haber pérdida o problemas de timeout para garantizar la integridad de los datos.

6.0.4. La capa de transporte del stack TCP/IP ofrece dos protocolos: TCP y UDP. ¿Qué servicios proveen dichos protocolos? ¿Cuáles son sus características? ¿Cuando es apropiado utilizar cada uno?

TCP (Protocolo de Control de Transmisión):

- Servicios: TCP ofrece una comunicación orientada a la conexión y con una transferencia de datos confiable. Asegura que los datos se entreguen en orden y sin errores. Se encarga de la detección y retransmisión de paquetes perdidos.
- Características: Orientado a la conexión, confiable, control de flujo, control de congestión, entrega en orden.
- Cuándo Utilizarlo: Se utiliza cuando la confiabilidad y la integridad de los datos son críticas, como en transferencias de archivos, comunicaciones de correo electrónico y aplicaciones web.

UDP (Protocolo de Datagrama de Usuario):

- Servicios: UDP ofrece una comunicación no orientada a la conexión y sin garantía de entrega de datos. No realiza corrección de errores ni retransmisiones.
- Características: No orientado a la conexión, sin garantía de entrega ni orden de los datos, más rápido en ciertos casos.
- Cuándo Utilizarlo: Se utiliza cuando la velocidad y la eficiencia son más importantes que la confiabilidad. Es adecuado para aplicaciones de transmisión en tiempo real, como videoconferencias, llamadas y transmisiones en vivo (streaming).

7. Dificultades

- Planificación: Cómo empezamos a encarar los requerimentos del tp, empezando con una pequeña aplicación cliente-servidor y luego modificandola en base al enunciado.
- Manejo de Errores y Retransmisiones: Los protocolos requieren asegurarse de que los paquetes se entreguen correctamente y en el orden correcto. Esto incluye el manejo de paquetes perdidos, duplicados o fuera de secuencia, así como la gestión de reintentos en caso de que ocurran problemas de comunicación.
- Asegurarnos de que el control del tiempo y temporizadores esten correctamente configurados para detectar paquetes perdidos y realizar las retransmisiones en el momento adecuado.
- Múltiples clientes: Tuvimos que gestionar cuidadosamente los threads para cada cliente.
- Sincronización entre Cliente y Servidor: Mantener correctamente sincronizados los números de secuencia y de ack entre el cliente y el servidor para asegurar la transferencia de datos.

8. Conclusión

El desarrollo de esta aplicación cliente-servidor para la transferencia de archivos a través de la red ha sido un proyecto desafiante pero que nos ha inquerido nuevos conocimientos sobre redes.

Se han aprendido las principales características de una transferencia de archivos confiable entre un cliente y un servidor, en donde el protocolo 'Stop and Wait' se luce por su simplicidad mientras que el protocolo 'Selective Repeat' ofrece una comunicación más eficiente pero más difícil de implementar.

Durante el desarrollo de la aplicación tuvimos varios desafíos que hemos explicado en la sección número 7 'Dificultades', pero que han servido para la comprensión de los temas vistos en clase.

De este trabajo práctico nos llevamos una base bastante sólida para futuras aplicaciones que puedan desarrollarse sobre el Internet.