



75.43/75.33/95.60 - INTRODUCCIÓN A LOS SISTEMAS DISTRIBUIDOS

FACULTAD DE INGENIERÍA, UNIVERSIDAD DE BUENOS AIRES

TRABAJO PRÁCTICO 1

File-Transfer usando protocolo RDT

Padrón	Alumno	Dirección de correo
100972	Pérez Leiras, Agustín	aperezl@f i.uba.ar
99139	Sinisi, Fernando	fsinisi@fi.uba.ar
103488	Díaz, Juan Ignacio	jidiaz@f i.uba.ar
89341	Verón, Lucas Maximiliano	lveron@fi.uba.ar
94986	Inoriza, Pablo	pinoriza@f i.uba.ar

Índice

1. Introducción	2
2. Hipótesis y suposiciones realizadas	3
3. Desarrollo e Implementación	4
4. Pruebas	9
5. Análisis	20
6. Preguntas a responder	29
7. Dificultades encontradas	35
8. Conclusión	36

1. Introducción

El presente práctico tiene por objetivo la implementación de file transfer mediante un protocolo RDT. Un protocolo RDT es un protocolo de datos confiable, es decir, un protocolo que provee un servicio que asegura que ninguno de los bits de datos transferidos sea corrompido (que algún bit cambia de 0 a 1 o viceversa), ni se pierda y todos los bits llegan en el orden en que fueron enviados.

Se llevaron a cabo 2 implementaciones de tipo de protocolo stop & wait (SAW) y Go-Back-N (GBN). Para esto último se llevo a cabo la programación de un Cliente y Servidor que mediante sus funciones UPLOAD y DOWNLOAD permite subir un archivo al Servidor y recuperarlo, usando el Cliente, respectivamente.

2. Hipótesis y suposiciones realizadas

Algunas de los supuestos e Hipótesis utilizadas son las siguientes:

1. No se contempla el caso en donde el cliente ingresa una ruta de archivo no válida o pide del servidor una archivo inexistente.
2. No se utilizo una suma de comprobación o checksum ya que la misma se encuentra implementada en la capa de transporte utilizada, a saber, UDP[RFC 768]. Se considera de suficiente el chequeo de errores proporcionado por UDP, mediante el checksum de complemento a 1.
3. Un cliente en caso de subir o descargar un archivo con un nombre ya usado pisa los ya existentes. Por lo tanto el usuario debe evitarlo.

3. Desarrollo e Implementación

Para la Implementación del File Transfer se utilizaron varias estructuras de datos, pero se eligió hacer un uso preponderante de colas y colas con prioridad. Se implementó una clase UDPServer que contiene información sobre el host, port, storage_path (path donde se van a guardar los archivos), el protocolo RDT a usar(stop & wait o GBN). Además el Server mantiene un socket de conexión, una lista de conexiones y una cola de prioridad de acciones de las conexiones.

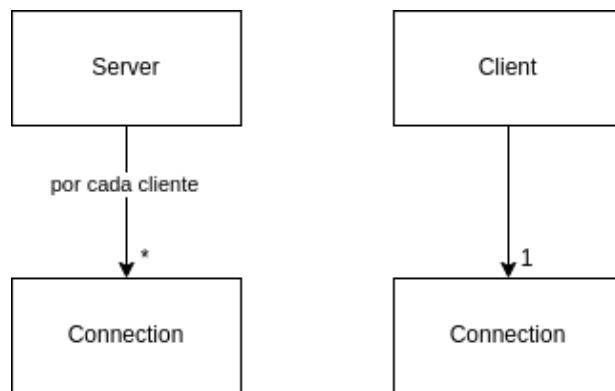


Figura 1: Marco general - Server/Client

El servidor se compone con la siguiente estructura:

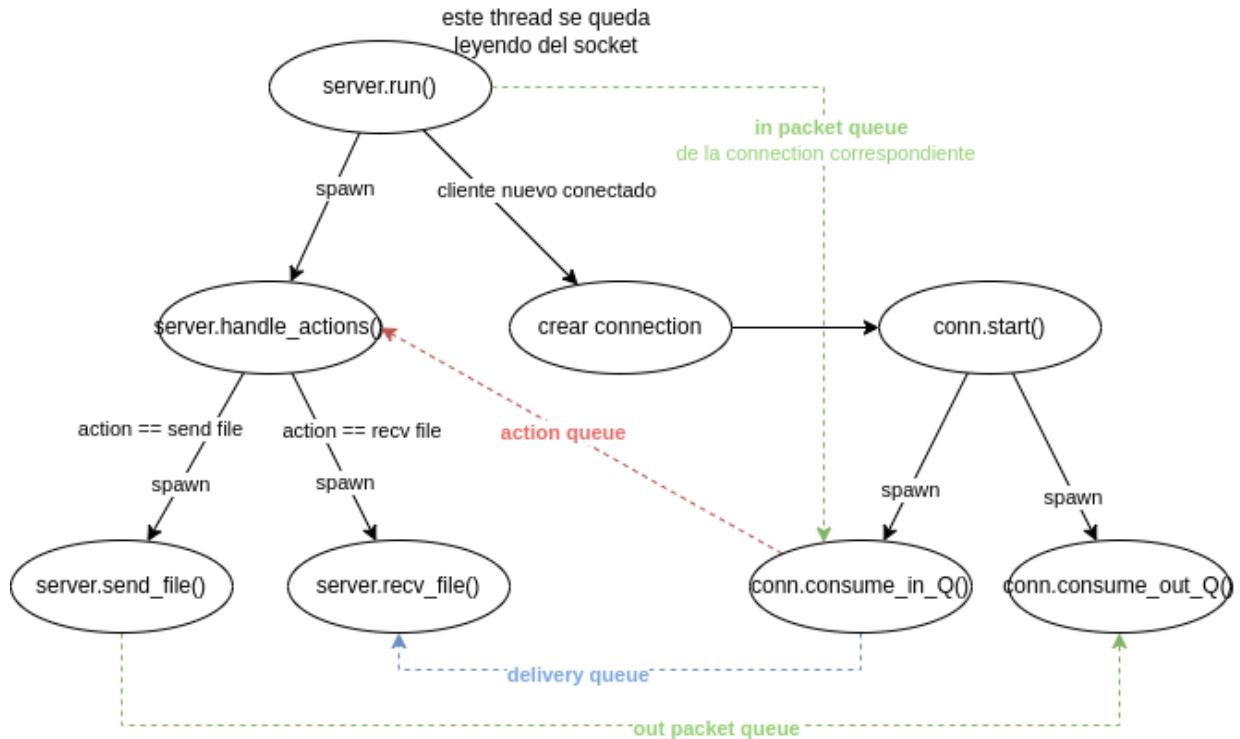


Figura 2: Marco general - Estructura del Servidor

Una vez que se inicia el Servidor se llevan a cabo los siguientes pasos dentro de mismo:

- Se parsean los argumentos pasados por parámetro(host, port, storage y protocol).
- Se inicia el servicio, se crea 1 socket UDP y se setea el timeout del mismo.
- A continuación se inicia un hilo de ejecución, se agrega un handler(handler action) y luego se entra en un bucle infinito por donde se va a estar recibiendo los mensajes que lleguen a través del socket. En este último paso se setea el tamaño del datagrama a recibir en cada lectura.
- En caso de recibir una excepción por timeout de un cliente(se identifica con el address(host y port)) se continúa con la ejecución.
- Se ejecuta una acción de nuevo cliente(en caso de no existir) o se agrega a la cola de mensajes de la conexión del cliente el datagrama recibido por el socket.
- Handlear una nueva conexión implica armar una conexión(UDPConnection) que va a tener el socket, el address y la cola de mensajes de acción. Además el Servidor mantiene la lista de conexiones abiertas y en uso. Luego se agrega a cola de mensajes a procesar de la conexión recién creada el datagrama recibido y se ejecuta la conexión.
- En el caso de que la conexión no sea nueva, directamente, el servidor agrega a la cola de mensajes recibidos el nuevo mensaje(datagrama) recibido.

El *handler_action* es la función que se encarga de recibir y enviar los mensajes al cliente. Es un bucle infinito que hace lo siguiente:

- Recupera una acción de la cola de acciones.
- Verifica la acción a ejecutar y basándonos en eso realiza la acción(RECV_FILE, SEND_FILE, CLOSE, STOP). Explicamos brevemente cada una de ellas.
 - RECV_FILE|SEND_FILE: inicia un hilo y ejecuta la función de recepción (recv_file_from_client) o envío de mensajes(send_file_to_client).
 - CLOSE: se hace un 'shutdown' de la conexión para el cliente.
 - STOP: se sale del bucle infinito.

Una breve explicación sobre las funciones de recepción y envío:

recv_file_from_client: se obtiene la conexión para el cliente específico y luego mientras esa conexión exista se obtiene un chunk(una tira de bytes) de la cola de la conexión del cliente(delivery_queue). Luego se escribe ese chunk en un archivo local al Server(path configurado al iniciar el Servidor) si el chunk esta vacío se corta la escritura.

send_file_to_client: *send* se comporta de manera similar, se tiene un bucle infinito, se recupera la conexión del cliente y luego se lee de un archivo chunks (con un tamaño previamente configurado) y se agrega a la cola de salida de la conexión (enqueue_out_data). Si no hay nada mas para agregar(byte vacío) entonces se corta la conexión del bucle infinito.

Otras consideraciones sobre el Server:

- Cuando el Server se para se recorre la lista de clientes(conexiones abiertas) que tiene, se cierra la conexión para c/u y se saca del diccionario. El diccionario tiene la forma address(host,port) -> udp_connection. Luego se agrega a la cola de prioridad con prioridad alta una acción de STOP.
- Por último se jonean todos los hilos de la lista de hilos que posee.

El Cliente se compone con la siguiente estructura:

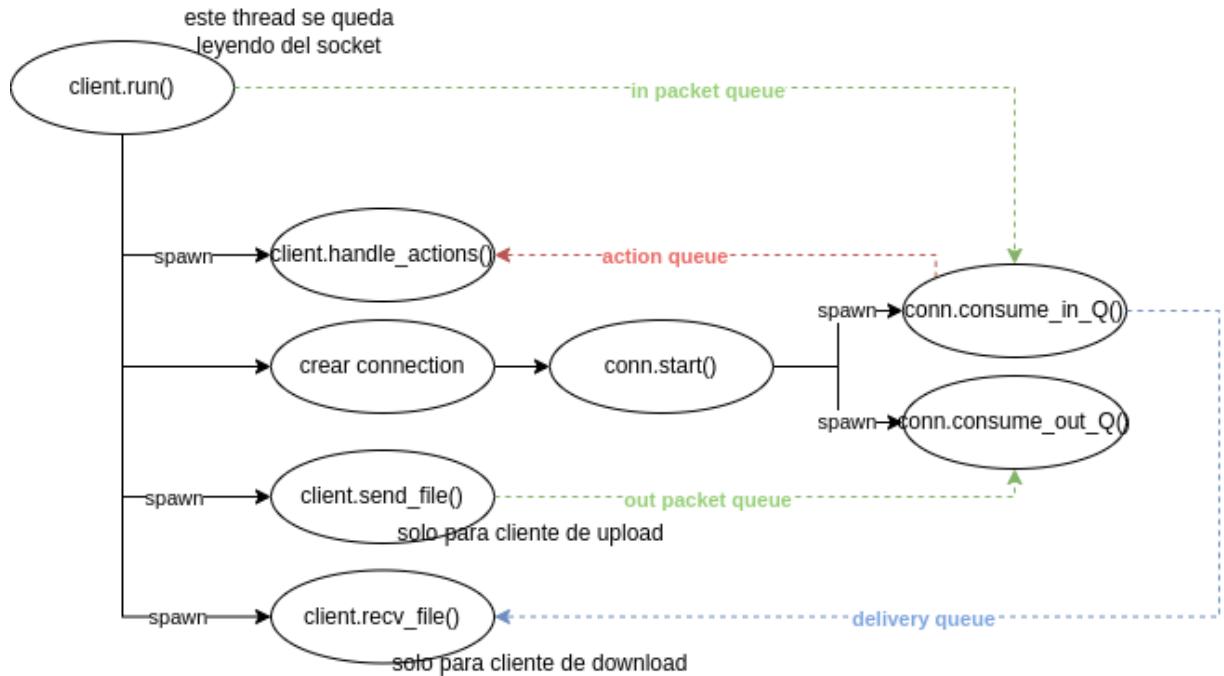


Figura 3: Marco general - Estructura del Cliente

Para el cliente se construyó una clase que tiene: host, port, file_path(path donde se encuentra el archivo a transferir), file_name(nombre del archivo), rdt_protocol y si es una acción de upload o download(boolean). Además una cola de prioridad de acciones(action_queue) y array de hilos(threads).

Como primer paso se instancia un UDPClient y se comienza(start). Se crea un socket UDP y se setea el timeout del cliente. Luego se crea una conexión(UDPConnection) usando el socket previamente creado, la address y la cola de acciones.

El segundo paso es correr el cliente(run). Al correr el cliente se inicializa un hilo y se le pasa el handler_handle_action. Luego se arranca la conexión(connection_start). El Cliente arranca un hilo de ejecución con un 'handler' dependiendo de si es una acción de 'upload' o 'download'.

A continuación entra en un bucle infinito y si el cliente está activo(not stopping) recibe un paquete usando el socket que tiene asociado(recvfrom). El paquete recibido es insertado en la cola de la conexión de ingreso(enqueue_in_data) para su conexión. Si el cliente hace un upload ejecuta la función 'send_file_to_server', que hace un bucle infinito mientras no este parado(stopping) y haya datos a enviar. Entonces lee un chunk de datos, y luego los inserta a la cola de la conexión de salida(enqueue_out_data). Esto se repite con cada iteración.

'recv_file_from_server' de manera similar, abre un archivo y hace un bucle infinito mientras existan chunck para recibir y el cliente no este parado(stopping). Cada chunck ob-

tenido de la cola de delivery(`delivery_queue`) de la conexión es escrito en el archivo abierto.

`'handle_action'` entra en un bucle infinito que chequea constantemente las acciones dentro de la cola de acciones del cliente(`action_queue`). Si es `'CLOSE'` el Servidor inicia un cierre de conexión, entonces se cierra la conexión(`shutdown`) y se para el cliente(`stopping = True`). Si es un `'STOP'` se sale del bucle infinito para detener el hilo cuando el cliente es el que inicia el cierre.

Las funciones de `start_thread` y `stop` funcionan igual que para el Server, pero el `stop` solo realiza el cierre para la única conexión que existe.

Para ver los detalles sobre el protocolo de aplicación desarrollado ver la sección de preguntas a responder.

4. Pruebas

Algunas de las pruebas llevadas a cabo para verificar el buen funcionamiento de la aplicación.

- **Transferencia de archivos con clientes simultaneo - upload y download - protocol udp-saw**

Se realizo una transferencia de un archivo pdf de 8702980 bytes en simultáneo con nombres diferentes de archivo y se verificó los checksum md5 para ambos archivos. En una segunda instancia se realizo la descarga de los mismos archivos recuperando los archivos originales.

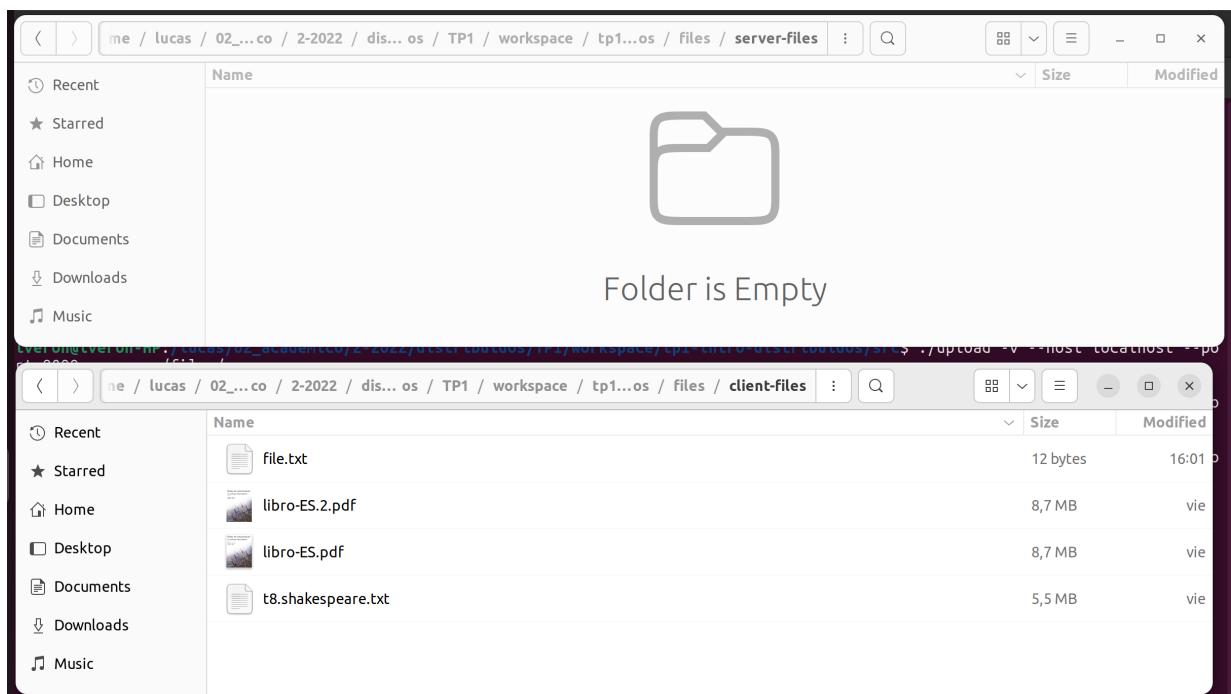


Figura 4: Vista del Servidor sin archivos

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ md5sum ..//files/client-files/libro-ES.pdf ..//files/client-files/libro-ES.2.pdf
4c4afac6e258c45ad9d31a148526bcb7 ..//files/client-files/libro-ES.pdf
4c4afac6e258c45ad9d31a148526bcb7 ..//files/client-files/libro-ES.2.pdf
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$
```

Figura 5: Vista checksum md5 inicial

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ stat ..//files/client-files/libro-ES.2.pdf
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ stat ..//files/client-files/libro-ES.2.pdf
  File: ..//files/client-files/libro-ES.2.pdf
  Size: 8702980   Blocks: 17008   IO Block: 4096   regular file
Device: 1030ah/66314d  Inode: 10497374  Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/ lveron)  Gid: ( 1000/ lveron)
Access: 2022-10-03 16:22:29.097089793 -0300
Modify: 2022-09-30 12:19:04.970606686 -0300
Change: 2022-09-30 12:19:04.970606686 -0300
 Birth: 2022-09-30 12:10:57.787695871 -0300
  File: ..//files/client-files/libro-ES.2.pdf
  Size: 8702980   Blocks: 17000   IO Block: 4096   regular file
Device: 1030ah/66314d  Inode: 10497125  Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/ lveron)  Gid: ( 1000/ lveron)
Access: 2022-10-03 16:25:44.022769493 -0300
Modify: 2022-09-30 12:19:04.970606000 -0300
Change: 2022-10-03 16:22:41.365193450 -0300
 Birth: 2022-10-03 16:22:29.093089760 -0300
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$
```

Figura 6: Vista stat de archivos

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ ./start-server
2022-10-03 16:47:06,533: lib.utils: INFO: RDT protocol udp-saw
2022-10-03 16:47:06,534: lib.udp_server: INFO: The server is ready. Host:localhost Port:8080
2022-10-03 16:47:06,534: lib.udp_server: INFO: Server is running
2022-10-03 16:47:50,861: lib.udp_server: INFO: New connection from ('127.0.0.1', 43281)
2022-10-03 16:47:50,861: lib.udp_server: INFO: Handling action Action.RECV_FILE from client ('127.0.0.1', 43281)
2022-10-03 16:47:51,990: lib.udp_server: INFO: New connection from ('127.0.0.1', 45367)
2022-10-03 16:47:51,991: lib.udp_server: INFO: Handling action Action.RECV_FILE from client ('127.0.0.1', 45367)
2022-10-03 16:47:52,381: lib.udp_server: INFO: Handling action Action.CLOSE from client ('127.0.0.1', 43281)
2022-10-03 16:47:53,395: lib.udp_server: INFO: Handling action Action.CLOSE from client ('127.0.0.1', 45367)
```

Figura 7: Vista logs Server una vez transferidos los archivos

```
2022-10-03 16:46:40,163: __main__: INFO: Stopping client...
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ ./upload -v --host localhost --port 8080 --src ..//files/client-files/libro-ES.pdf --name libro-ES.pdf
2022-10-03 16:47:50,859: lib.logger: DEBUG: Logger configured
2022-10-03 16:47:50,859: __main__: DEBUG: Arguments: Namespace(verbose=10, host='localhost', port=8080, src='..//files/client-files/libro-ES.pdf', name='libro-ES.pdf')
2022-10-03 16:47:50,859: lib.udp_client: INFO: Action handling thread started
2022-10-03 16:47:50,861: lib.udp_client: INFO: Handling action Action.SEND_FILE
2022-10-03 16:47:52,381: lib.udp_client: INFO: Handling action Action.CLOSE
2022-10-03 16:47:53,381: __main__: INFO: Stopping client...
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$
```

Figura 8: Vista log Cliente 1 una vez transferidos los archivos

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ ./upload -v --host localhost --port 8080 --src ..//files/client-files/libro-ES.2.pdf --name libro-ES.2.pdf
2022-10-03 16:47:51,988: lib.logger: DEBUG: Logger configured
2022-10-03 16:47:51,989: __main__: DEBUG: Arguments: Namespace(verbose=10, host='localhost', port=8080, src='..//files/client-files/libro-ES.2.pdf', name='libro-ES.2.pdf')
2022-10-03 16:47:51,989: lib.udp_client: INFO: Action handling thread started
2022-10-03 16:47:51,991: lib.udp_client: INFO: Handling action Action.SEND_FILE
2022-10-03 16:47:53,392: lib.udp_client: INFO: Handling action Action.CLOSE
2022-10-03 16:47:54,392: __main__: INFO: Stopping client...
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$
```

Figura 9: Vista log Cliente 2 una vez transferidos los archivos

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ md5sum ..//files/server-files/libro-ES.pdf ..//files/server-files/libro-ES.2.pdf
4c4afac6e258c45ad9d31a148526bc7 ..//files/server-files/libro-ES.pdf
4c4afac6e258c45ad9d31a148526bc7 ..//files/server-files/libro-ES.2.pdf
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$
```

Figura 10: Vista Checksum md5 una vez completa el upload

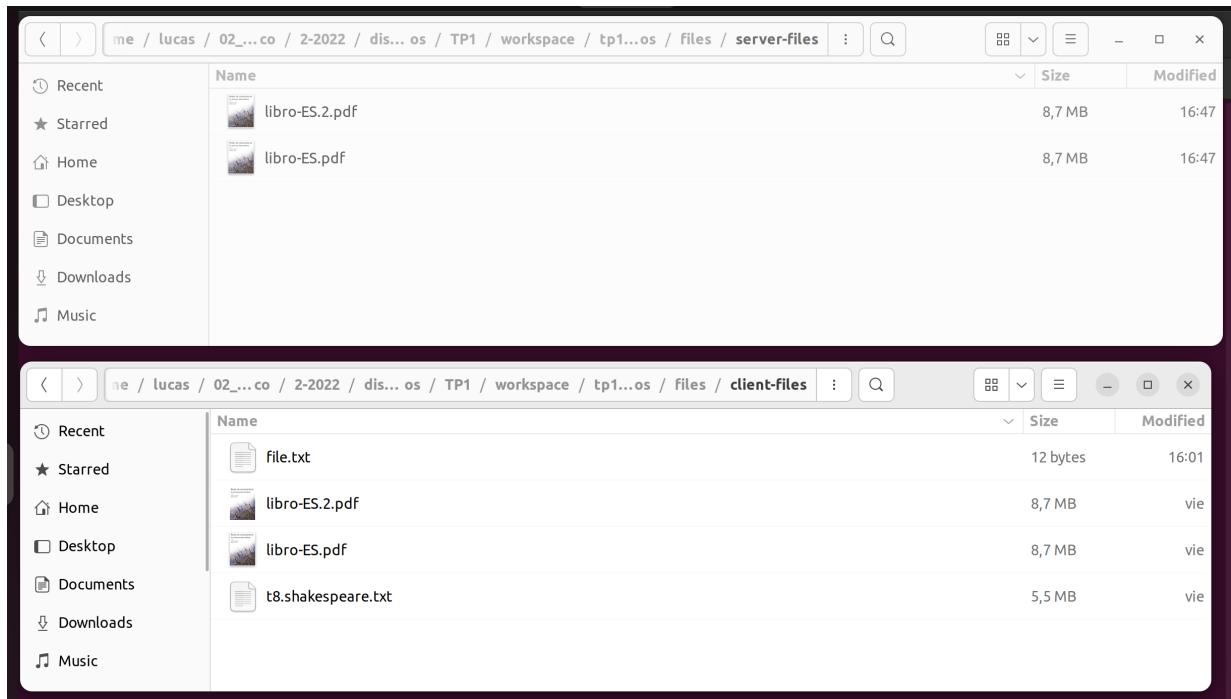


Figura 11: Vista contenido del Servidor una vez completo el upload

A continuación se realizo la misma prueba pero usando la función de *download*

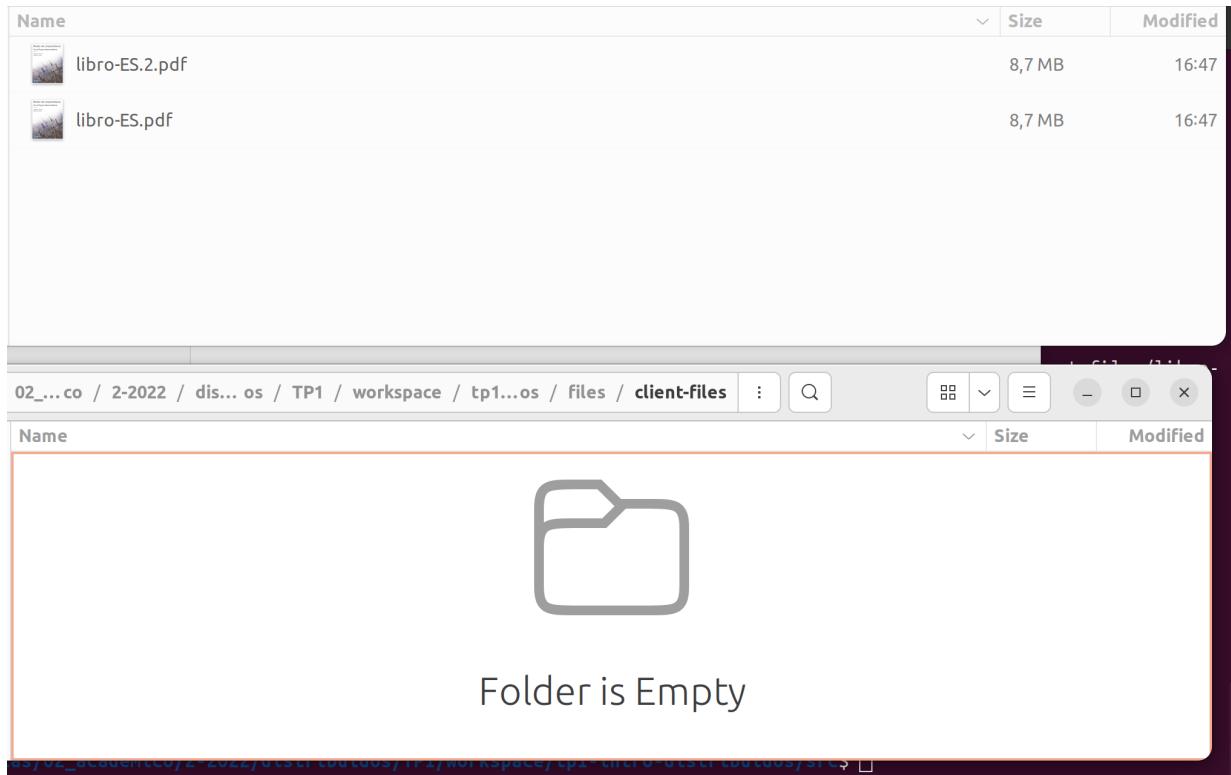


Figura 12: Vista contenido del Cliente antes de iniciar la descarga

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ ./start-server
2022-10-03 17:24:10,388: lib.utils: INFO: RDT protocol udp-saw
2022-10-03 17:24:10,389: lib.udp_server: INFO: The server is ready. Host:localhost Port:8080
2022-10-03 17:24:10,389: lib.udp_server: INFO: Server is running
2022-10-03 17:24:41,054: lib.udp_server: INFO: New connection from ('127.0.0.1', 36592)
2022-10-03 17:24:41,054: lib.udp_server: INFO: Handling action Action.SEND_FILE from client ('127.0.0.1', 36592)
2022-10-03 17:24:41,529: lib.udp_server: INFO: New connection from ('127.0.0.1', 39850)
2022-10-03 17:24:41,530: lib.udp_server: INFO: Handling action Action.SEND_FILE from client ('127.0.0.1', 39850)
2022-10-03 17:24:44,171: lib.udp_server: INFO: Handling action Action.CLOSE from client ('127.0.0.1', 36592)
2022-10-03 17:24:44,541: lib.udp_server: INFO: Handling action Action.CLOSE from client ('127.0.0.1', 39850)
```

Figura 13: Vista del Server una completado la descarga

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ ./download -v --host localhost --port 8080 --dst ..//files/client-files/libro-ES.pdf --name libro-ES.pdf
2022-10-03 17:24:41,051: lib.logger: DEBUG: Logger configured
2022-10-03 17:24:41,051: __main__: DEBUG: Arguments: Namespace(verbose=10, host='localhost', port=8080, dst='..//files/client-files/libro-ES.pdf', name='libro-ES.pdf')
2022-10-03 17:24:41,052: lib.udp_client: INFO: Action handling thread started
2022-10-03 17:24:41,055: lib.udp_client: INFO: Handling action Action.RECV_FILE
2022-10-03 17:24:44,171: lib.udp_client: INFO: Handling action Action.CLOSE
2022-10-03 17:24:45,172: __main__: INFO: Stopping client...
```

Figura 14: Vista del cliente 1 ejecutando la descarga

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ ./download -v --host localhost --port 8080 --dst ../files/client-files/libro-ES.2.pdf --name libro-ES.2.pdf
2022-10-03 17:24:41,526: lib.logger: DEBUG: Logger configured
2022-10-03 17:24:41,527: __main__: DEBUG: Arguments: Namespace(verbose=10, host='localhost', port=8080, dst='../files/client-files/libro-ES.2.pdf', name='libro-ES.2.pdf')
2022-10-03 17:24:41,528: lib.udp_client: INFO: Action handling thread started
2022-10-03 17:24:41,530: lib.udp_client: INFO: Handling action Action.RECV_FILE
2022-10-03 17:24:44,541: lib.udp_client: INFO: Handling action Action.CLOSE
2022-10-03 17:24:45,542: __main__: INFO: Stopping client...
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$
```

Figura 15: Vista del cliente 2 ejecutando la descarga

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ md5sum ..//files/client-files/libro-ES.pdf
4c4afac6e258c45ad9d31a148526bcb7 ..//files/client-files/libro-ES.pdf
4c4afac6e258c45ad9d31a148526bcb7 ..//files/client-files/libro-ES.2.pdf
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$
```

Figura 16: Vista del checksum md5 una vez completada la descarga

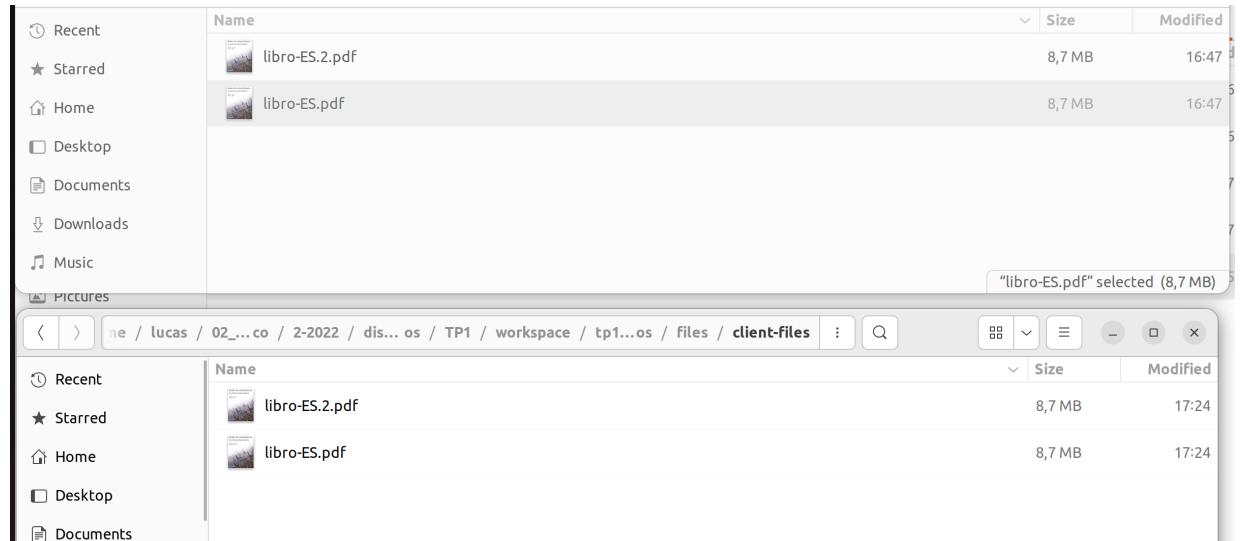


Figura 17: Vista de los archivos descargados una vez completada la descarga

- Transferencia de archivos con clientes simultaneo - upload con perdida - protocolo udp-gbn

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/comcast$ ./comcast --device=lo --packet-loss=20% --target-add r=127.0.0.0/8 --target-proto=udp --target-port=1024:65535
sudo tc qdisc show | grep "netem"
sudo tc qdisc add dev lo handle 10: root htb default 1
sudo tc class add dev lo parent 10: classid 10:1 htb rate 1000000kbit
sudo tc class add dev lo parent 10: classid 10:10 htb rate 1000000kbit
sudo tc qdisc add dev lo parent 10:10 handle 100: netem loss 20.00%
sudo iptables -A POSTROUTING -t mangle -j CLASSIFY --set-class 10:10 -p udp --dport 1024:65535 -d 127.0.0.0/8
Packet rules setup...
Run `sudo tc -s qdisc` to double check
Run `./comcast --device lo --stop` to reset
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/comcast$
```

Figura 18: Vista inicio comcast, perdida del 20 %

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ ./upload -v --host localhost --port 8080 --src ../files/client-files/libro-ES.pdf --name libro-ES.pdf
2022-10-04 00:30:46,095: lib.logger: DEBUG: Logger configured
2022-10-04 00:30:46,095: __main__: DEBUG: Arguments: Namespace(verbose=10, host='localhost', port=8080, src='../files/client-files/libro-ES.pdf', name='libro-ES.pdf')
2022-10-04 00:30:46,096: lib udp_client: INFO: Action handling thread started
2022-10-04 00:30:46,098: lib udp_client: INFO: Handling action Action.SEND_FILE
2022-10-04 00:32:36,310: lib udp_client: INFO: Handling action Action.CLOSE
2022-10-04 00:32:37,310: __main__: INFO: Stopping client...
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$
```

Figura 19: Vista upload Cliente 1

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ ./upload -v --host localhost --port 8080 --src ../files/client-files/libro-ES.2.pdf --name libro-ES.2.pdf
2022-10-04 00:30:47,096: lib.logger: DEBUG: Logger configured
2022-10-04 00:30:47,096: __main__: DEBUG: Arguments: Namespace(verbose=10, host='localhost', port=8080, src='../files/client-files/libro-ES.2.pdf', name='libro-ES.2.pdf')
2022-10-04 00:30:47,097: lib udp_client: INFO: Action handling thread started
2022-10-04 00:30:47,202: lib udp_client: INFO: Handling action Action.SEND_FILE
2022-10-04 00:32:29,869: lib udp_client: INFO: Handling action Action.CLOSE
2022-10-04 00:32:30,071: __main__: INFO: Stopping client...
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$
```

Figura 20: Vista upload Cliente 2

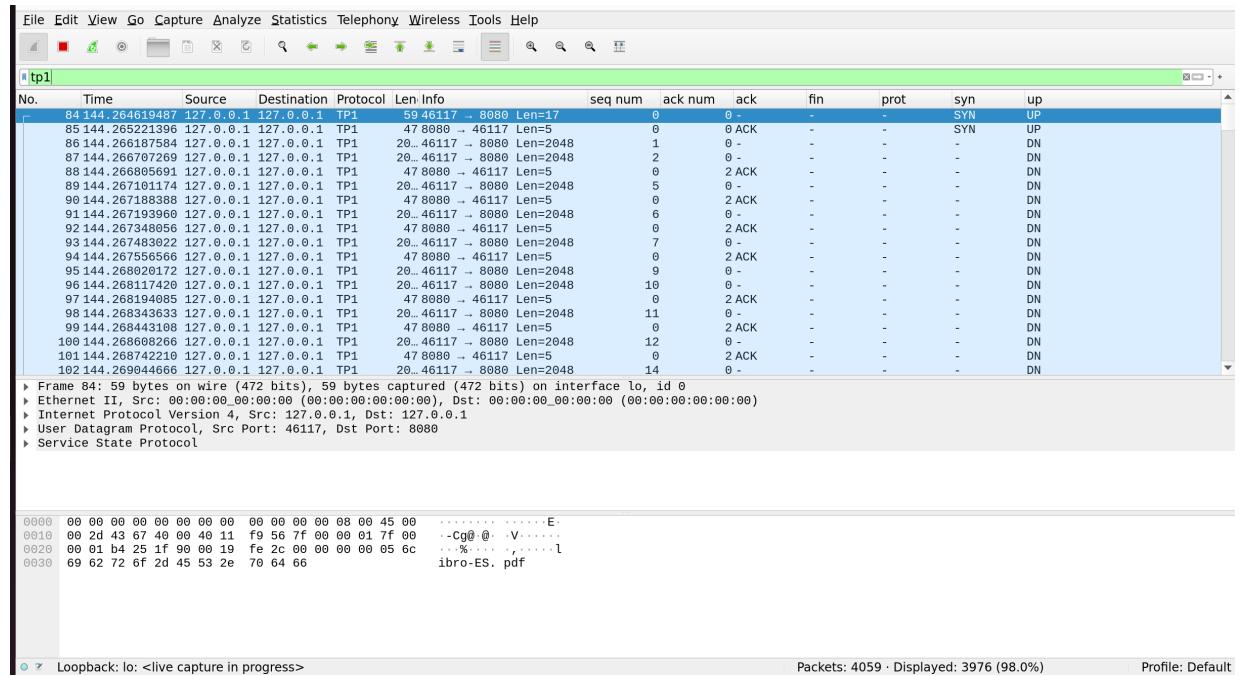


Figura 21: Vista wireshark haciendo el syn

Trabajo Práctico 1

Introducción a los Sistemas Distribuidos

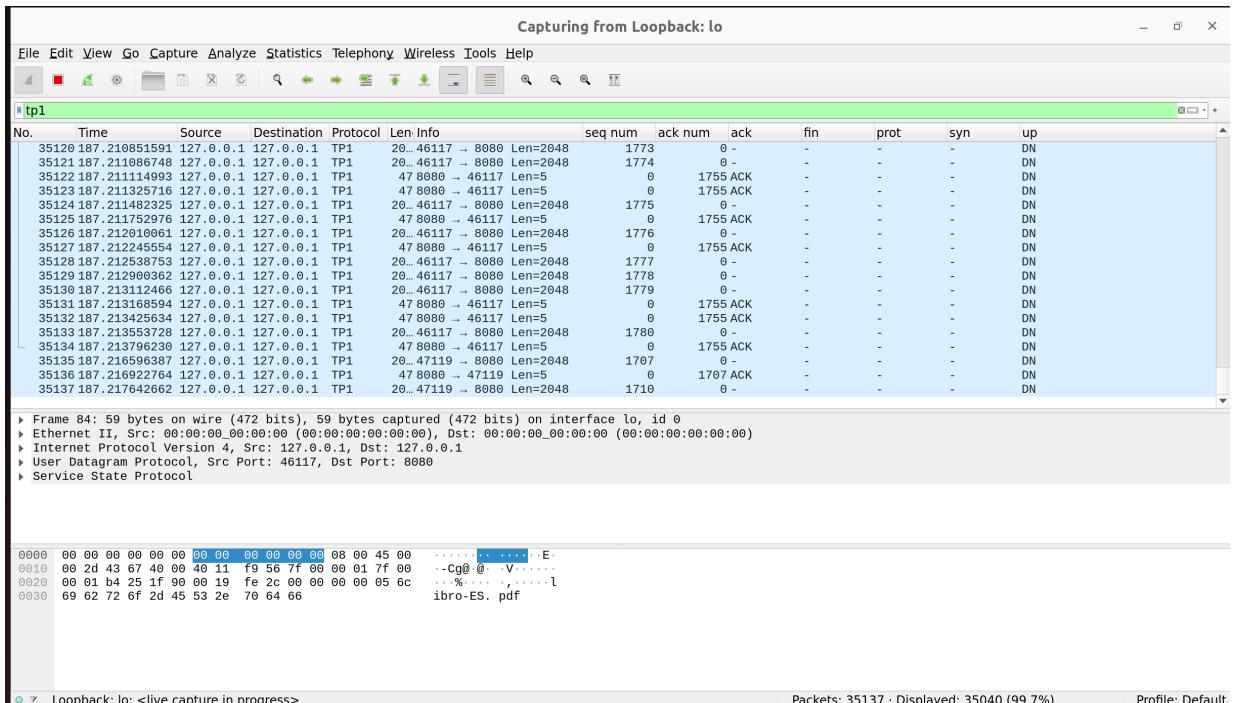


Figura 22: Vista wireshark recibiendo paquetes con perdida

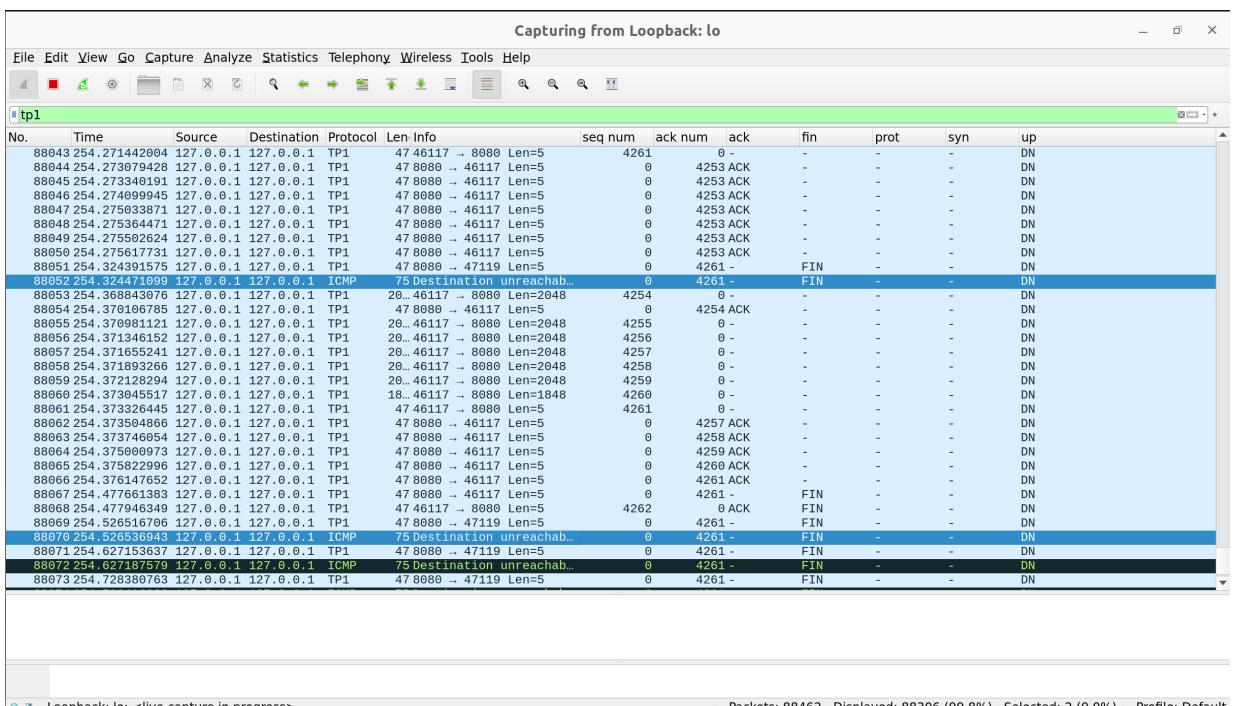


Figura 23: Vista wireshark finalizando conexión

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ ./start-server -P udp-gbn
2022-10-04 00:27:24,728: lib.utils: INFO: RDT protocol udp-gbn
2022-10-04 00:27:24,728: lib_udp_server: INFO: The server is ready. Host:localhost Port:8080
2022-10-04 00:27:24,728: lib_udp_server: INFO: Server is running
2022-10-04 00:30:46,097: lib_udp_server: INFO: New connection from ('127.0.0.1', 46117)
2022-10-04 00:30:46,098: lib_udp_server: INFO: Handling action Action.RECV_FILE from client ('127.0.0.1', 46117)
2022-10-04 00:30:47,200: lib_udp_server: INFO: New connection from ('127.0.0.1', 47119)
2022-10-04 00:30:47,202: lib_udp_server: INFO: Handling action Action.RECV_FILE from client ('127.0.0.1', 47119)
2022-10-04 00:32:36,310: lib_udp_server: INFO: Handling action Action.CLOSE from client ('127.0.0.1', 46117)
2022-10-04 00:32:59,868: lib_udp_server: INFO: Handling action Action.CLOSE from client ('127.0.0.1', 47119)
```

Figura 24: Vista Servidor recibiendo todos los paquetes

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/comcast$ md5sum ..../tp1-intro-distribuidos/files/server-files/libro-ES.pdf ..../tp1-intro-distribuidos/files/client-files/libro-ES.2.pdf ..../tp1-intro-distribuidos/files/client-files/libro-ES.pdf
4c4afac6e258c45ad9d31a148526bc7 ..../tp1-intro-distribuidos/files/server-files/libro-ES.pdf
4c4afac6e258c45ad9d31a148526bc7 ..../tp1-intro-distribuidos/files/server-files/libro-ES.2.pdf
4c4afac6e258c45ad9d31a148526bc7 ..../tp1-intro-distribuidos/files/client-files/libro-ES.pdf
4c4afac6e258c45ad9d31a148526bc7 ..../tp1-intro-distribuidos/files/client-files/libro-ES.2.pdf
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/comcast$
```

Figura 25: Vista md5sum check transferencia archivos

- Transferencia de archivos con clientes simultaneo - upload con perdida 10% - protocolo udp-saw

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/comcast$ ./comcast --device=lo --packet-loss=10% --target-add
r=127.0.0.8 --targetproto=udp --targetport=1024:65535
sudo tc qdisc show | grep "netem"
sudo tc qdisc add dev lo handle 10: root htb default 1
sudo tc class add dev lo parent 10: classid 10:1 htb rate 1000000kbit
sudo tc class add dev lo parent 10: classid 10:10 htb rate 1000000kbit
sudo tc qdisc add dev lo parent 10:10 handle 100: netem loss 10.00%
sudo iptables -A POSTROUTING -t mangle -j CLASSIFY --set-class 10:10 -p udp --dport 1024:65535 -d 127.0.0.0/8
Packet rules setup...
Run `sudo tc -s qdisc` to double check
Run `./comcast --device lo --stop` to reset
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/comcast$
```

Figura 26: Vista iniciando comcast, 10% perdida

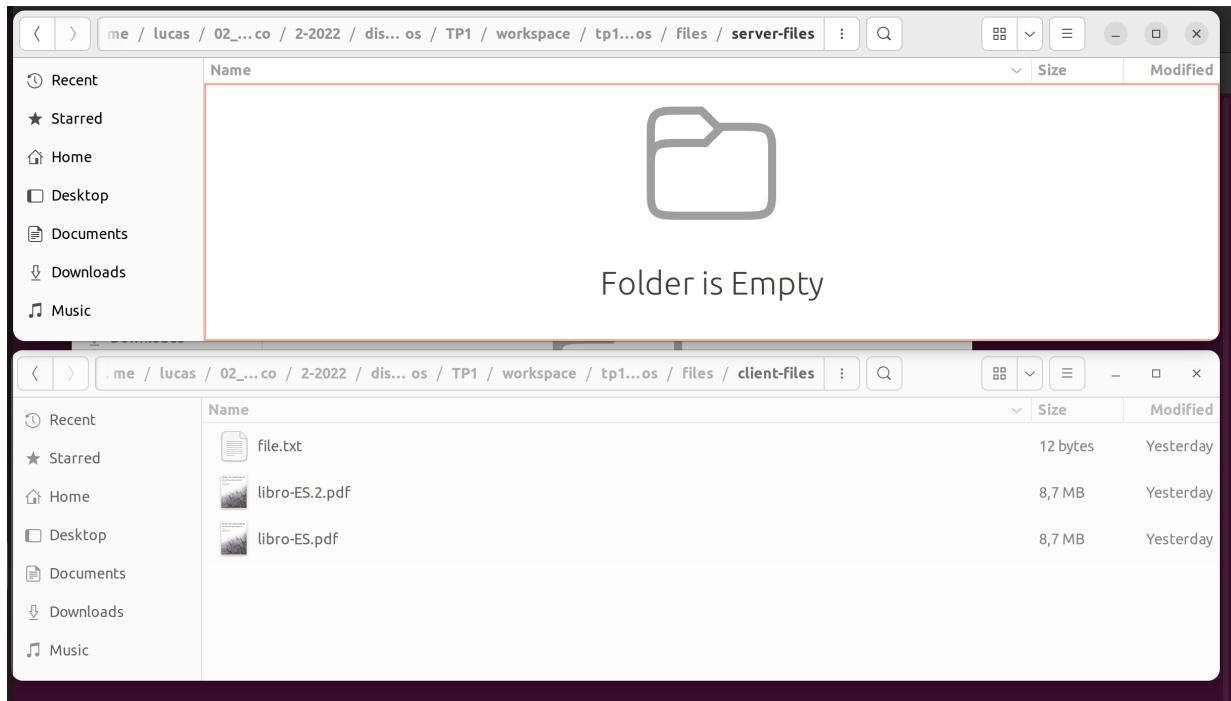


Figura 27: Vista carpeta del Servidor vacío

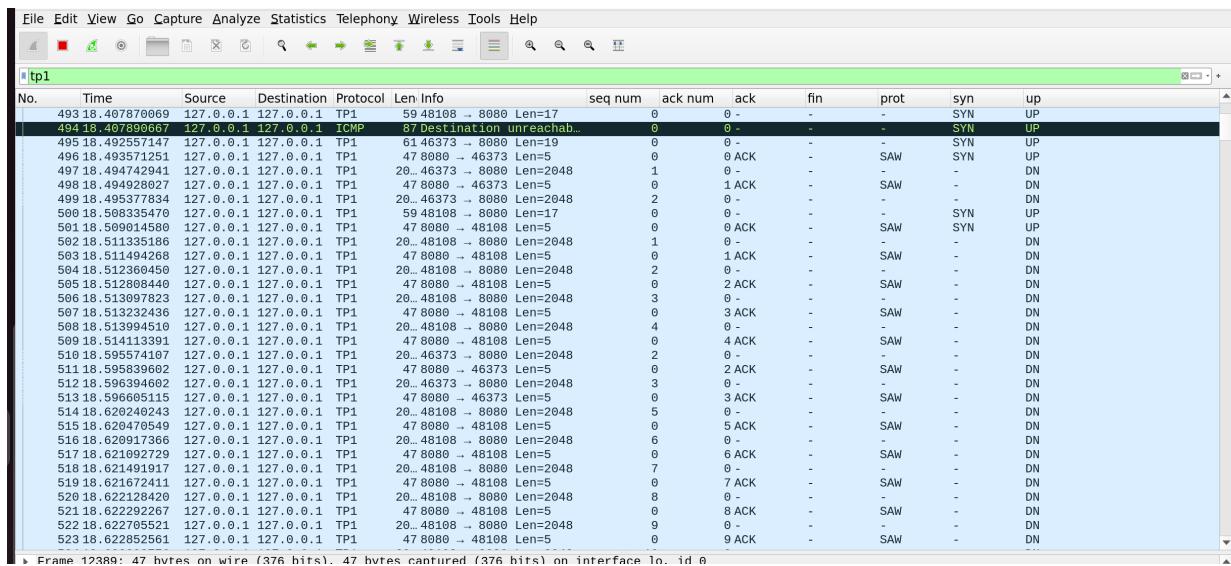


Figura 28: Vista wireshark iniciando la conexión

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ ./upload -v --host localhost --port 8080 --src ../files/client-files/libro-ES.pdf --name libro-ES.pdf
2022-10-04 01:06:49,864: lib.logger: DEBUG: Logger configured
2022-10-04 01:06:49,864: __main__: DEBUG: Arguments: Namespace(verbose=10, host='localhost', port=8080, src='../files/client-files/libro-ES.pdf', name='libro-ES.pdf')
2022-10-04 01:06:49,865: lib_udp_client: INFO: Action handling thread started
2022-10-04 01:07:04,018: lib_udp_client: INFO: Handling action Action.SEND_FILE
2022-10-04 01:08:05,664: lib_udp_client: INFO: Handling action Action.CLOSE
2022-10-04 01:08:06,665: __main__: INFO: Stopping client...
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$
```

Figura 29: Vista upload Cliente 1

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ ./upload -v --host localhost --port 8080 --src ../files/client-files/libro-ES.2.pdf --name libro-ES.2.pdf
2022-10-04 01:06:50,747: lib.logger: DEBUG: Logger configured
2022-10-04 01:06:50,747: __main__: DEBUG: Arguments: Namespace(verbose=10, host='localhost', port=8080, src='../files/client-files/libro-ES.2.pdf', name='libro-ES.2.pdf')
2022-10-04 01:06:50,748: lib_udp_client: INFO: Action handling thread started
2022-10-04 01:07:04,003: lib_udp_client: INFO: Handling action Action.SEND_FILE
2022-10-04 01:08:22,652: lib_udp_client: INFO: Handling action Action.CLOSE
2022-10-04 01:08:23,653: __main__: INFO: Stopping client...
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$
```

Figura 30: Vista upload Cliente 2

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos/src$ ./start-server -P udp-saw
2022-10-04 01:07:03,989: lib.utils: INFO: RDT protocol udp-saw
2022-10-04 01:07:03,989: lib_udp_server: INFO: The server is ready. Host:localhost Port:8080
2022-10-04 01:07:03,989: lib_udp_server: INFO: Server is running
2022-10-04 01:07:04,002: lib_udp_server: INFO: New connection from ('127.0.0.1', 46373)
2022-10-04 01:07:04,002: lib_udp_server: INFO: Handling action Action.RECV_FILE from client ('127.0.0.1', 46373)
2022-10-04 01:07:04,017: lib_udp_server: INFO: New connection from ('127.0.0.1', 48108)
2022-10-04 01:07:04,018: lib_udp_server: INFO: Handling action Action.RECV_FILE from client ('127.0.0.1', 48108)
2022-10-04 01:08:05,665: lib_udp_server: INFO: Handling action Action.CLOSE from client ('127.0.0.1', 48108)
2022-10-04 01:08:22,655: lib_udp_server: INFO: Handling action Action.CLOSE from client ('127.0.0.1', 46373)
```

Figura 31: Vista upload Servidor, archivos transferidos

No.	Time	Source	Destination	Protocol	Len:Info	seq num	ack num	ack	fin	prot	syn	up
18653 96, 958945098	127.0.0.1	127.0.0.1	TP1		47 8080 → 46373 Len=5	0	4249	ACK	-	SAW	-	DN
18654 96, 9589450972	127.0.0.1	127.0.0.1	TP1		20, 46373 → 8080 Len=2048	4250	0	-	-	-	-	DN
18655 96, 959569187	127.0.0.1	127.0.0.1	TP1		47 8080 → 46373 Len=5	0	4250	ACK	-	SAW	-	DN
18656 96, 960352560	127.0.0.1	127.0.0.1	TP1		20, 46373 → 8080 Len=2048	4251	0	-	-	-	-	DN
18657 96, 986398760	127.0.0.1	127.0.0.1	TP1		20, 46373 → 8080 Len=2048	4251	0	-	-	-	-	DN
18658 97, 032224662	127.0.0.1	127.0.0.1	TP1		20, 46373 → 8080 Len=2048	4251	0	-	-	-	-	DN
18659 97, 032428032	127.0.0.1	127.0.0.1	TP1		47 8080 → 46373 Len=5	0	4251	ACK	-	SAW	-	DN
18660 97, 032849113	127.0.0.1	127.0.0.1	TP1		20, 46373 → 8080 Len=2048	4252	0	-	-	-	-	DN
18661 97, 033619842	127.0.0.1	127.0.0.1	TP1		47 8080 → 46373 Len=5	0	4252	ACK	-	SAW	-	DN
18662 97, 033327818	127.0.0.1	127.0.0.1	TP1		20, 46373 → 8080 Len=2048	4253	0	-	-	-	-	DN
18663 97, 062105583	127.0.0.1	127.0.0.1	TP1		20, 46373 → 8080 Len=2048	4253	0	-	-	-	-	DN
18664 97, 062558121	127.0.0.1	127.0.0.1	TP1		47 8080 → 46373 Len=5	0	4253	ACK	-	SAW	-	DN
18665 97, 063273104	127.0.0.1	127.0.0.1	TP1		20, 46373 → 8080 Len=2048	4254	0	-	-	-	-	DN
18666 97, 063559970	127.0.0.1	127.0.0.1	TP1		47 8080 → 46373 Len=5	0	4254	ACK	-	SAW	-	DN
18667 97, 064253249	127.0.0.1	127.0.0.1	TP1		20, 46373 → 8080 Len=2048	4255	0	-	-	-	-	DN
18668 97, 081238002	127.0.0.1	127.0.0.1	TP1		20, 46373 → 8080 Len=2048	4255	0	-	-	-	-	DN
18669 97, 081508367	127.0.0.1	127.0.0.1	TP1		47 8080 → 46373 Len=5	0	4255	ACK	-	SAW	-	DN
18670 97, 082317731	127.0.0.1	127.0.0.1	TP1		20, 46373 → 8080 Len=2048	4256	0	-	-	-	-	DN
18671 97, 082854558	127.0.0.1	127.0.0.1	TP1		47 8080 → 46373 Len=5	0	4256	ACK	-	SAW	-	DN
18672 97, 134099864	127.0.0.1	127.0.0.1	TP1		20, 46373 → 8080 Len=2048	4257	0	-	-	-	-	DN
18673 97, 134465445	127.0.0.1	127.0.0.1	TP1		47 8080 → 46373 Len=5	0	4257	ACK	-	SAW	-	DN
18674 97, 135529145	127.0.0.1	127.0.0.1	TP1		20, 46373 → 8080 Len=2048	4258	0	-	-	-	-	DN
18675 97, 136967727	127.0.0.1	127.0.0.1	TP1		47 8080 → 46373 Len=5	0	4258	ACK	-	SAW	-	DN
18676 97, 137767371	127.0.0.1	127.0.0.1	TP1		20, 46373 → 8080 Len=2048	4259	0	-	-	-	-	DN
18677 97, 138215237	127.0.0.1	127.0.0.1	TP1		47 8080 → 46373 Len=5	0	4259	ACK	-	SAW	-	DN
18678 97, 13969194	127.0.0.1	127.0.0.1	TP1		18, 46373 → 8080 Len=1848	4260	0	-	-	-	-	DN
18679 97, 1409123608	127.0.0.1	127.0.0.1	TP1		47 8080 → 46373 Len=5	0	4260	ACK	-	SAW	-	DN
18680 97, 141274268	127.0.0.1	127.0.0.1	TP1		47 46373 → 8080 Len=5	4261	0	-	-	-	-	DN
18681 97, 141699786	127.0.0.1	127.0.0.1	TP1		47 8080 → 46373 Len=5	0	4261	ACK	-	SAW	-	DN
18682 97, 142457065	127.0.0.1	127.0.0.1	TP1		47 8080 → 46373 Len=5	0	4261	FIN	-	-	-	DN
18683 97, 1430605933	127.0.0.1	127.0.0.1	TP1		47 46373 → 8080 Len=5	4262	0	ACK	FIN	SAW	-	DN

Figura 32: Vista wireshark fin de transmisión

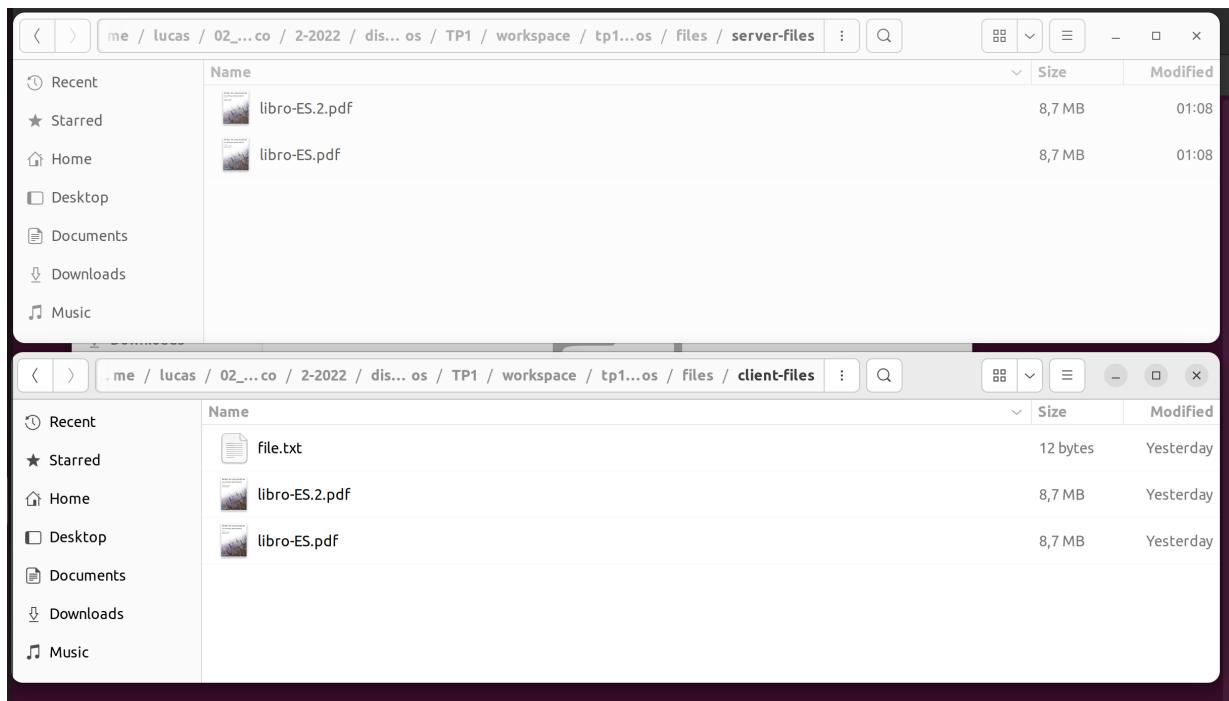


Figura 33: Vista carpeta con los archivos transferidos

```
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos$ md5sum files/client-files/libro-ES.pdf
files/client-files/libro-ES.2.pdf files/server-files/libro-ES.pdf files/server-files/libro-ES.2.pdf
4c4afac6e258c45ad9d31a148526bcb7 files/client-files/libro-ES.pdf
4c4afac6e258c45ad9d31a148526bcb7 files/client-files/libro-ES.2.pdf
4c4afac6e258c45ad9d31a148526bcb7 files/server-files/libro-ES.pdf
4c4afac6e258c45ad9d31a148526bcb7 files/server-files/libro-ES.2.pdf
lveron@lveron-HP:/lucas/02_academico/2-2022/distribuidos/TP1/workspace/tp1-intro-distribuidos$
```

Figura 34: Vista md5 checksum de los archivos transferidos

5. Análisis

Una comparativa entre los algoritmos de Stop & Wait y Go-Back-N respecto a los tiempos insumidos para cada uno de las soluciones se puede ver en los siguientes gráficos. Analizamos diferentes casos:

Ejecución upload con tamaño de archivo de 8,7 MB con 2 clientes simultaneo y 10% de perdida de paquetes

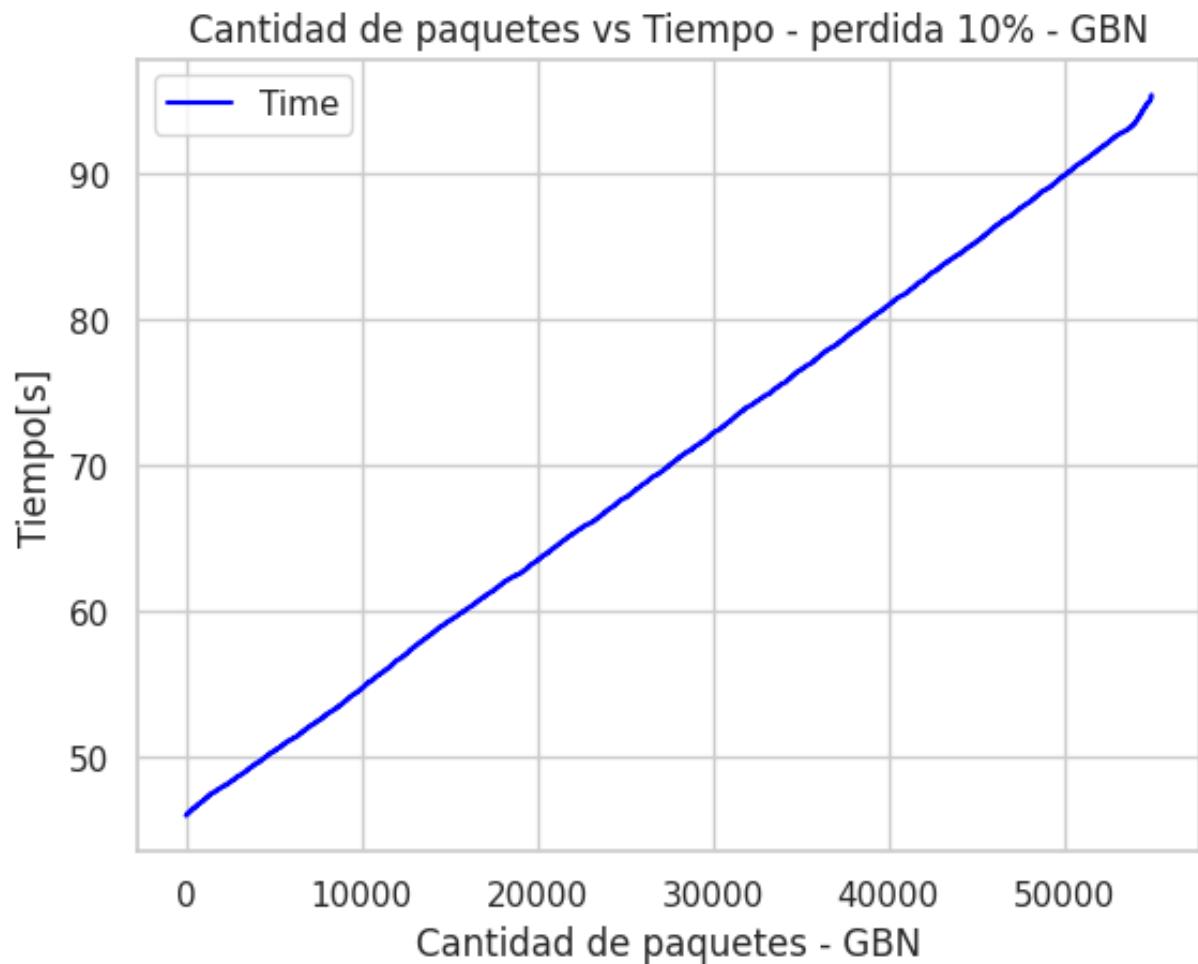


Figura 35: Gráfico tiempos algoritmo GBN

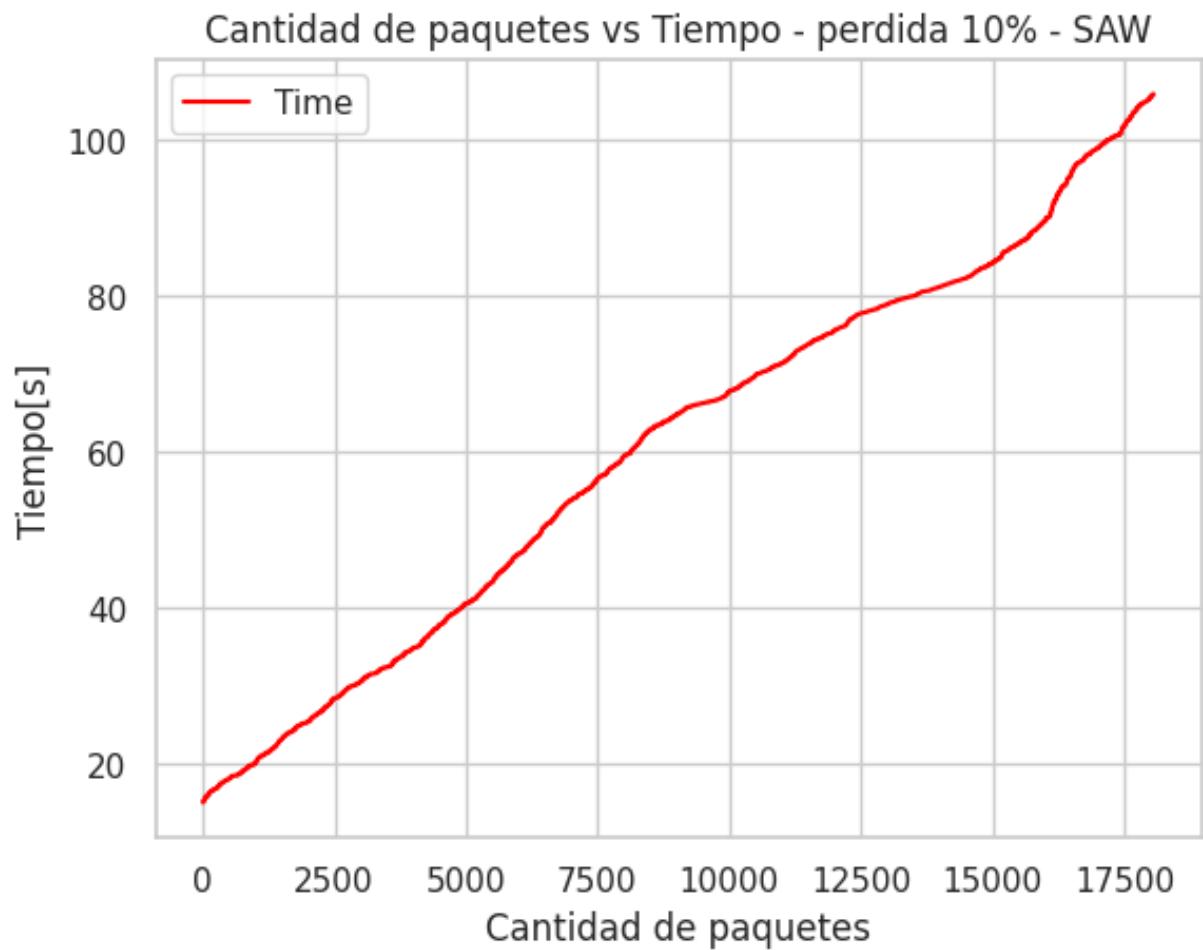


Figura 36: Gráfico tiempos algoritmo SAW

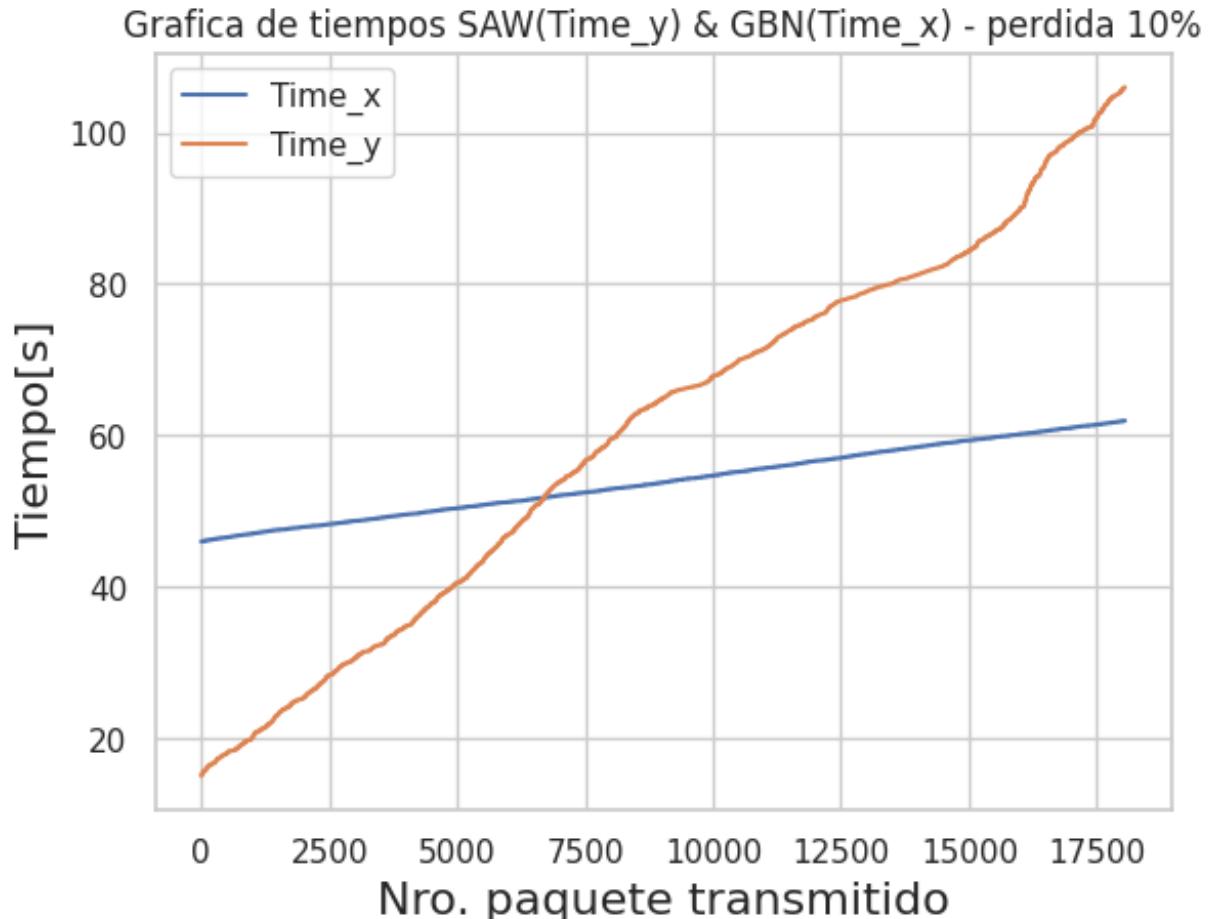


Figura 37: Gráfico comparativo de tiempos algoritmos SAW & GBN

Como puede verse en los gráficos como prácticamente ambos algoritmos demoran lo mismo, sin embargo SAW insume una menor cantidad de paquetes para completar la transferencia. Por otro lado GBN usa muchos mas paquetes, pero insumiendo la misma cantidad de tiempo aproximadamente. En la gráfica puede verse los efectos que tienen cada uno de los algoritmos. GBN al emitir paquetes por ráfagas "puede alcanzar tasas de transmisión de paquetes mas altas, sin embargo la perdida de paquetes hace que deba volver a retransmitir una cantidad mayor de paquetes nuevamente. SAW en cambio al perder 1 paquete solo debe retransmitir ese paquete perdido, disminuyendo la tasa de retransmisión.

Ejecución upload con tamaño de archivo de 2 MB con 2 clientes simultaneo y 10 % de perdida de paquetes

Para este caso se tiene una comportamiento similar al caso anterior. La implementación con SAW es mas rápida que usando GBN y la cantidad de paquetes transmitidos es 3 veces mas para al usar GBN que usando SAW. La explicación es la misma pero en este caso para

un archivo mas chico.

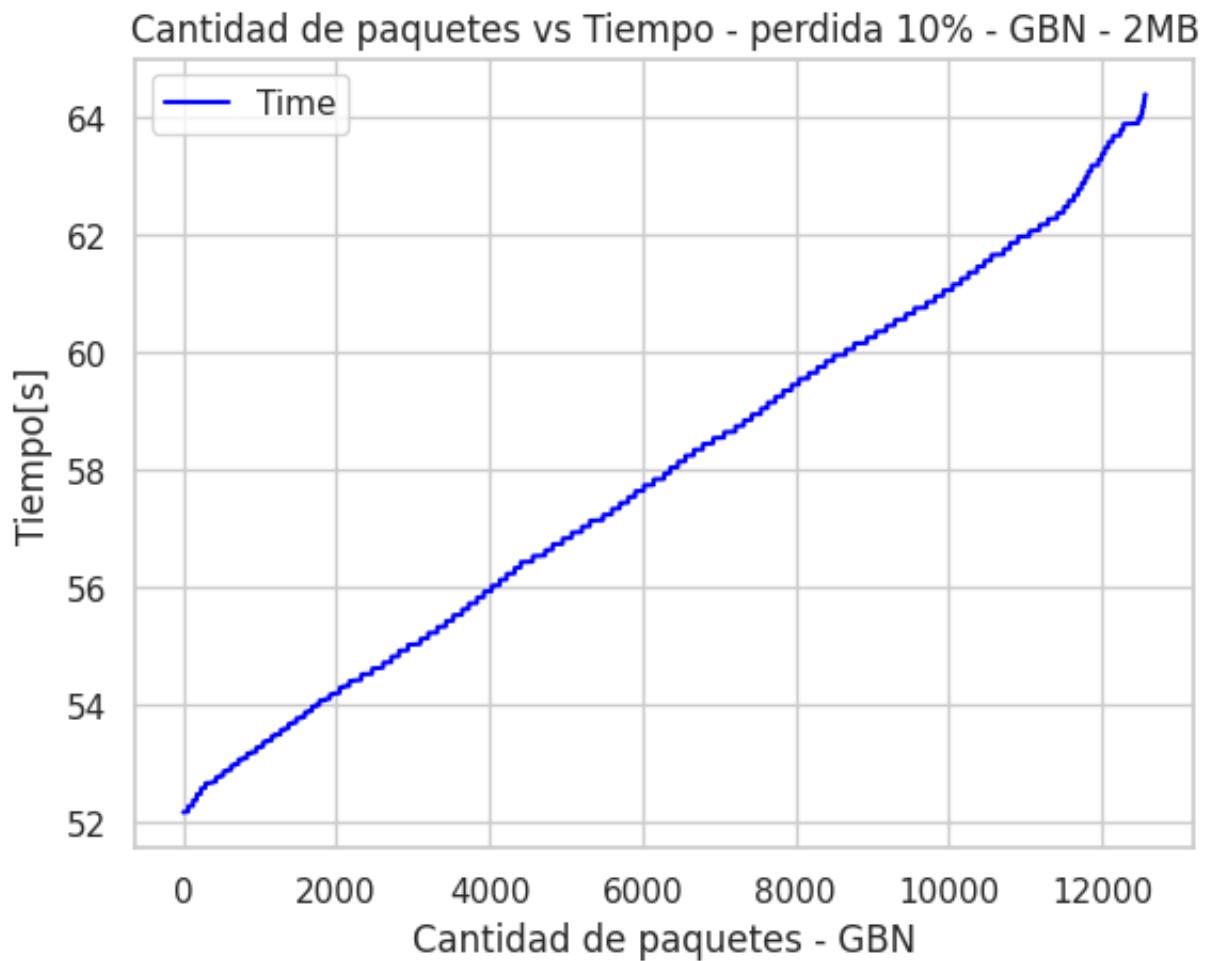


Figura 38: Gráfico tiempos vs cantidad de paquetes, algoritmos GBN, 2MB

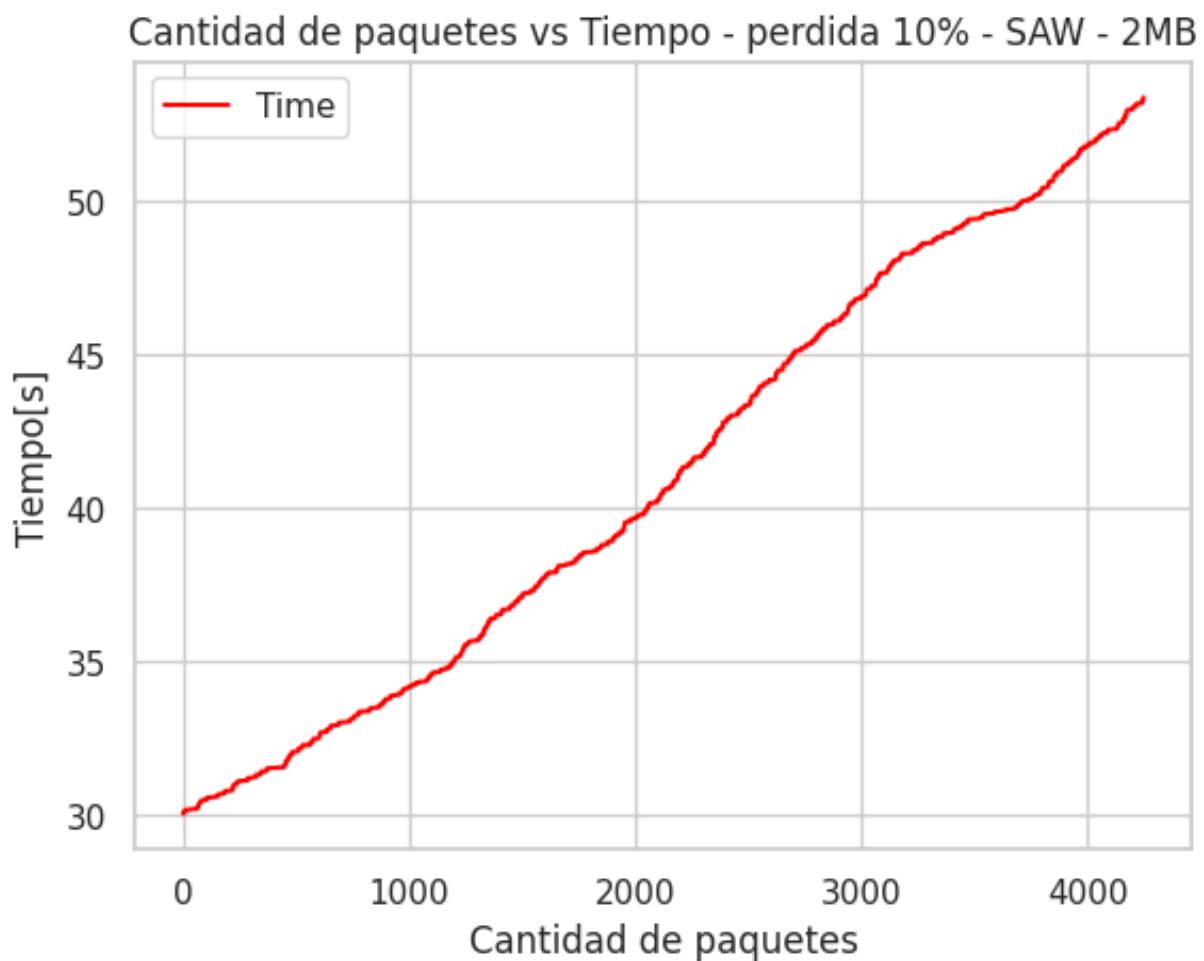


Figura 39: Gráfico tiempos vs cantidad de paquetes, algoritmos SAW, 2MB

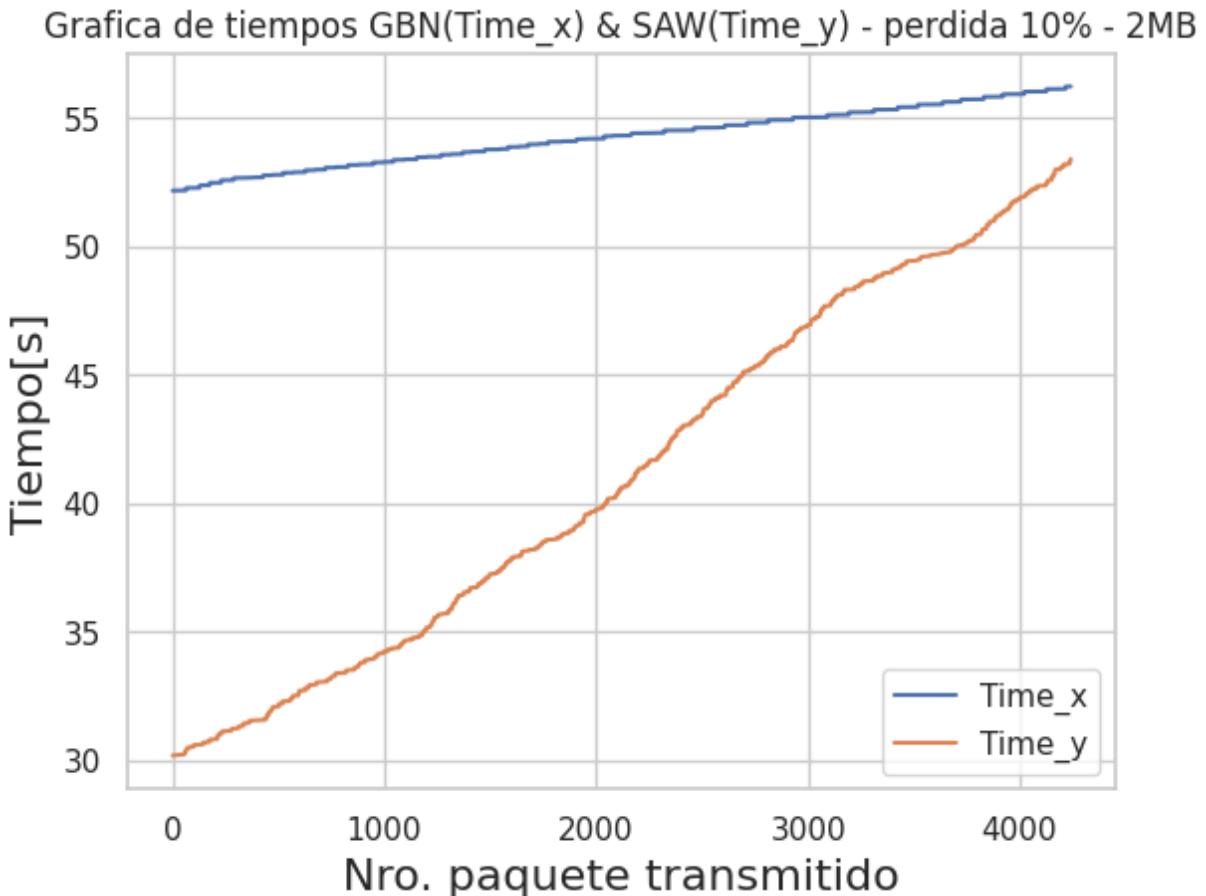


Figura 40: Gráfico comparativo tiempos vs cantidad de paquetes, algoritmos SAW y GBN, 2MB

Ejecución upload con tamaño de archivo de 2 MB con 2 clientes simultaneo sin perdida

En las gráficas se puede ver como GBN demora 0.5 segundos en realizar la descarga completa y utiliza unos 4000 paquetes para realizar la transferencia. Por otro lado SAW demora 0.7 segundos aproximadamente en completar la transferencia. La cantidad de paquetes transmitidos es la misma alrededor de 4000 paquetes. Por lo tanto GBN es mas rapido en este caso que SAW, con una transferencia de paquetes similar.

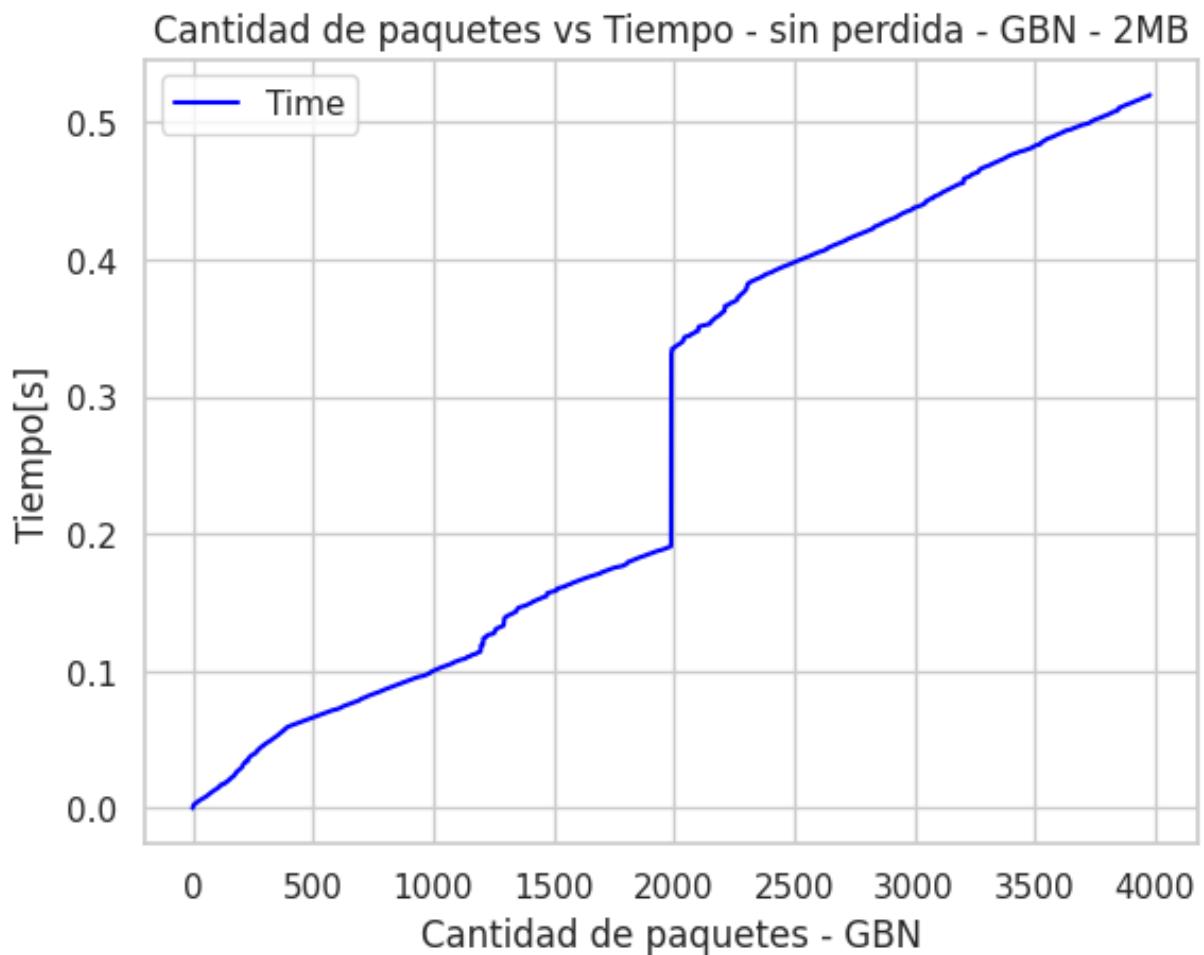


Figura 41: Gráfico tiempos vs cantidad de paquetes, algoritmos GBN, 2MB

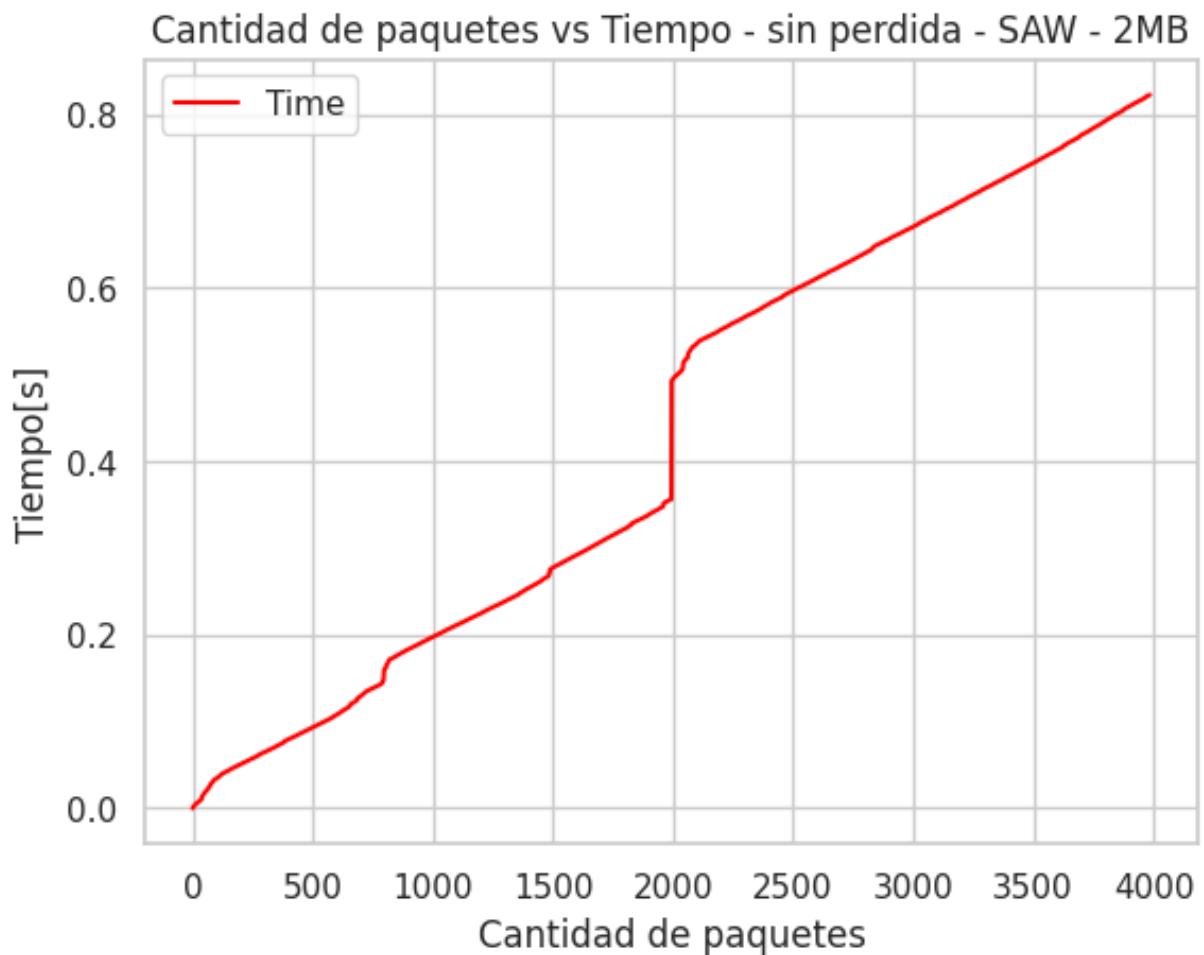


Figura 42: Gráfico tiempos vs cantidad de paquetes, algoritmos SAW, 2MB

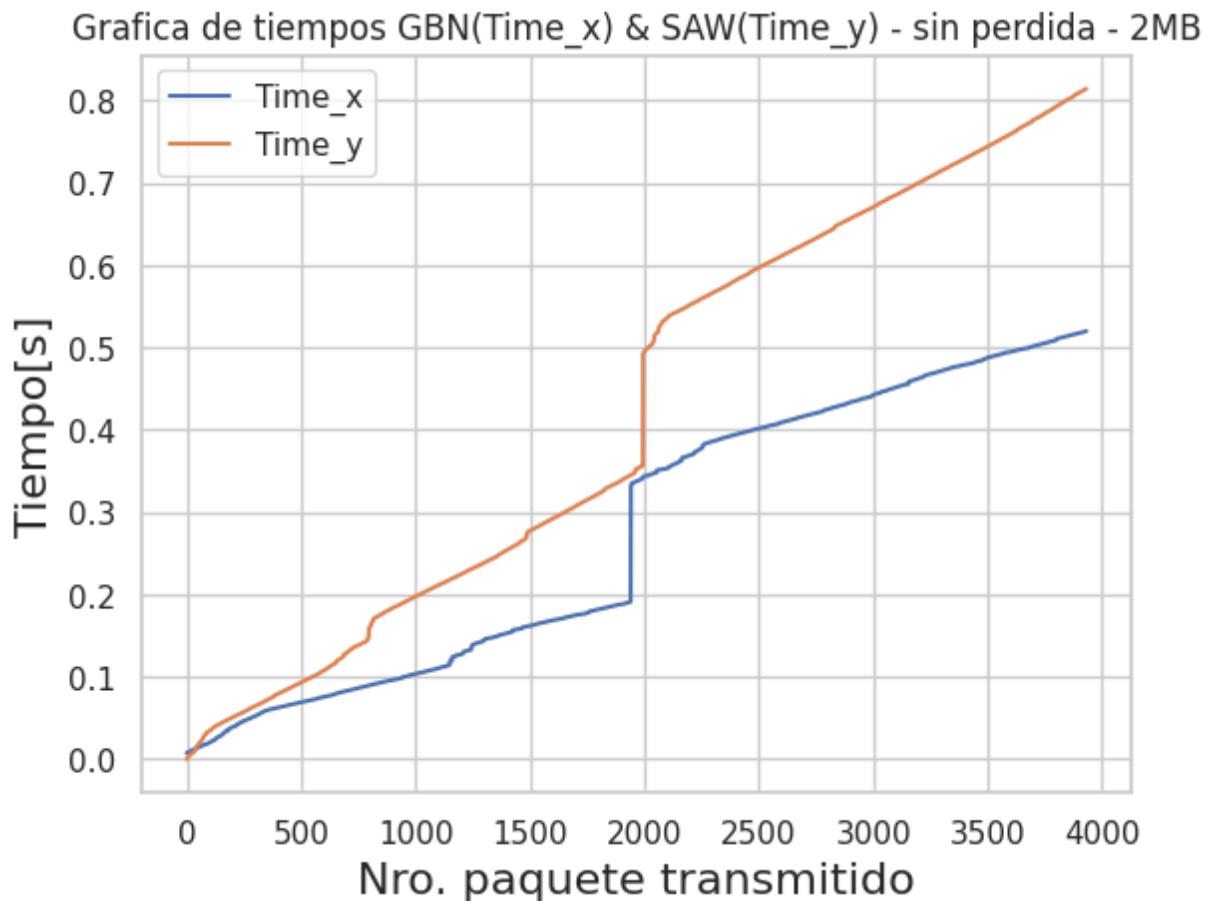


Figura 43: Gráfico comparativo tiempos vs cantidad de paquetes, algoritmos SAW y GBN, 2MB

6. Preguntas a responder

1. Describa la arquitectura Cliente-Servidor.

La arquitectura cliente-servidor es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.

En esta arquitectura es siempre el Cliente quien inicia la conexión con el Servidor.

Algunos ejemplos de aplicaciones que usen el modelo cliente-servidor son el Correo electrónico, un Servidor de impresión y la World Wide Web.

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

La capa de aplicación es donde residen las aplicaciones de red y sus protocolos de nivel de aplicación. La capa de aplicación de Internet incluye muchos protocolos, tales como el protocolo HTTP (que permite la solicitud y transferencia de documentos web), SMTP (que permite la transferencia de mensajes de correo electrónico) y FTP (que permite la transferencia de archivos entre dos sistemas terminales).

Un protocolo de la capa de aplicación está distribuido entre varios sistemas terminales, con la aplicación en un sistema terminal utilizando el protocolo para intercambiar paquetes de información con la aplicación que se ejecuta en otro sistema terminal. A este paquete de información de la capa de aplicación lo denominaremos mensaje.

Un protocolo de la capa de aplicación define cómo los procesos de una aplicación, que se ejecutan en distintos sistemas terminales, se pasan los mensajes entre sí. En particular, un protocolo de la capa de aplicación define:

- Los tipos de mensajes intercambiados; por ejemplo, mensajes de solicitud y mensajes de respuesta.
- La sintaxis de los diversos tipos de mensajes, es decir, los campos de los que consta el mensaje y cómo se delimitan esos campos.
- La semántica de los campos, es decir, el significado de la información contenida en los campos.
- Las reglas para determinar cuándo y cómo un proceso envía mensajes y responde a los mismos.

Algunos protocolos de la capa de aplicación están especificados en documentos RFC y, por tanto, son de dominio público. Por ejemplo, el protocolo de la capa de aplicación para la Web, HTTP (HyperText Transfer Protocol [RFC 2616]), está disponible como un RFC.

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

Se detalla a continuación la estructura de los paquetes y el funcionamiento del protocolo implementado

Estructura de los paquetes

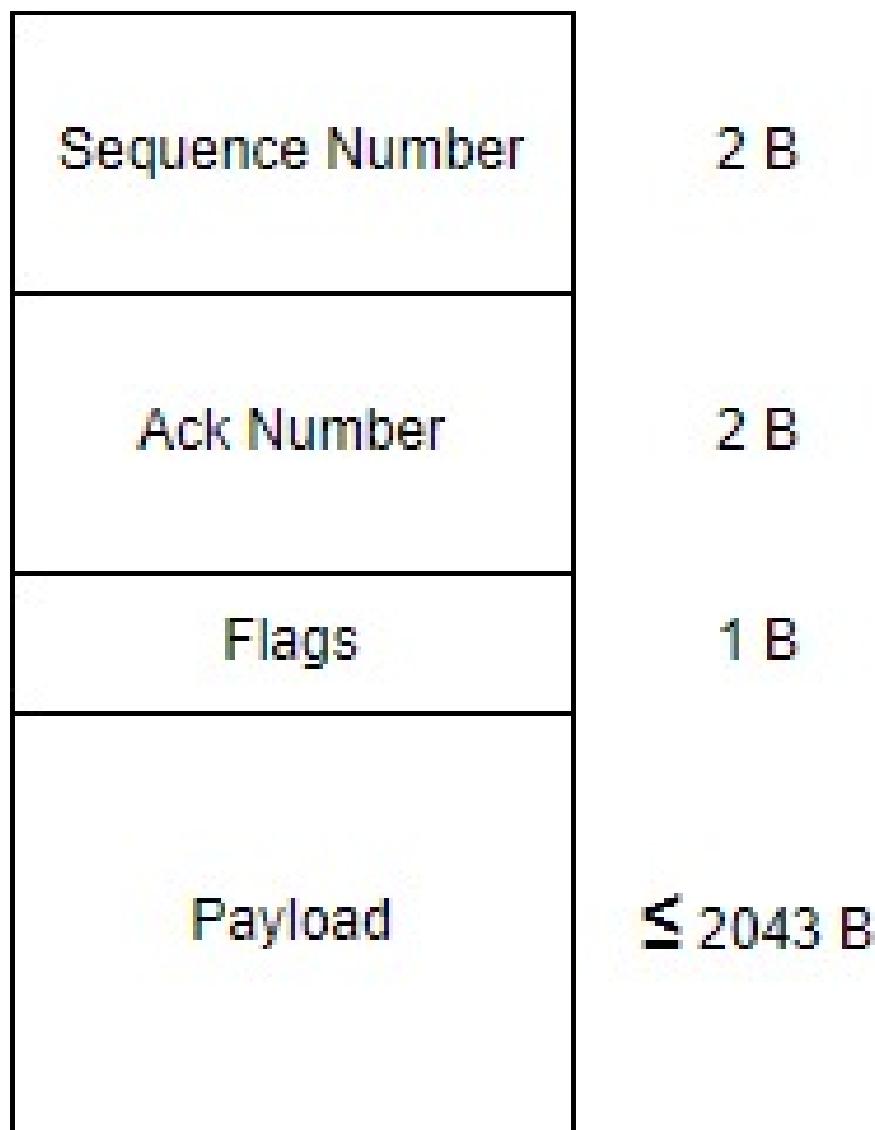


Figura 44: Estructura de los paquetes

Los paquetes están conformados por 1 header y un payload. El header(cabecera) consiste de 3 campos, a saber, **sequence number**, **ack number** y **flags**.

El *sequence number* de modo similar a TCP indica el orden del paquete que se está enviando. Luego de enviar cada paquete, el sender incrementa el sequence number en 1, de forma que cada paquete sea identifiable y así poder reenviarlo en caso de no recibir el ack correspondiente en debido tiempo. En el caso de los paquetes ack, no se incrementa el sequence number luego de ser enviados ya que estos paquetes no necesitan ser reenviados.

El *ack number* es un campo que toma validez al estar el flag ACK seteado en 1, de otro modo el campo no se toma en cuenta. Su valor se corresponde con el sequence number del paquete que se está confirmando (ack).

Estos campos, al ser de 2 bytes, permiten hasta un sequence/ack number de 65535, y luego vuelve a empezar de 0. El sequence/ack number inicial es siempre 0.

Flags: *ACK*: Se usa para indicar que el paquete en cuestión se trata de un paquete de acknowledge *SYN*: Se usa para los paquetes de inicio de conexión *FIN*: Se usa para los paquetes de fin de conexión *UP*: Se usa durante el proceso de inicio de conexión para, desde el cliente, indicarle al servidor si se quiere realizar un upload o download *PROT*: Se usa durante el proceso de inicio de conexión para, desde el servidor, indicarle al cliente qué protocolo está utilizando (Go Back-N / Stop and Wait)

Intercambio de mensajes

En el caso más general, el Sender envía un paquete con sequence number K y espera a recibir un paquete ACK con ack number K. En caso de no recibir dicho paquete en cierto intervalo de tiempo, reenvía el paquete con el mismo sequence number K.

El Receiver, al recibir este paquete, verificará que está en orden, es decir que el sequence number esperado era K, en caso negativo descarta el paquete y envía un paquete ACK indicando el último paquete recibido correctamente en ack number. En caso positivo, el Receiver enviará al Sender un paquete ACK con ack number K e incrementará su sequence number esperado a K+1.

En el caso Go-Back-N, el Sender puede enviar hasta N paquetes sin necesidad de esperar el ACK de un paquete anterior. Ejemplificando, al inicio de la conexión el Sender podría enviar los paquetes de sequence number 0 a N-1. Al enviar el primero (0), inicia un timer. De no recibir ACK en cierto tiempo, se reenvían todos los paquetes de la ventana. Al recibir un ACK con ack number K, el Sender asume que todos los paquetes con sequence number $\leq K$ han llegado a destino, por lo que los quita de la Ventana, haciendo espacio para K paquetes nuevos, de sequence number N hasta N+K. Esta suposición es correcta ya que el Receiver solo enviará ACKs con ack number igual al sequence number del último paquete recibido. Si el Sender recibe un ACK K sin haber recibido ACK K-1, es porque el ACK K-1 se perdió/retrasó, pero el Receiver ha recibido correctamente los paquetes K-1 y K.

El caso de Stop-and-Wait no es más que Go-Back-N con N=1, ya que la Ventana tiene espacio para 1 solo paquete, y hasta no recibirse el ACK de ese paquete, no podrá enviarse uno nuevo.

Inicio de conexión

Como es propio de las arquitecturas Cliente-Servidor, es el cliente quien inicia la conexión con el servidor. Para realizar esto, envía un paquete con el flag SYN=1, y el flag UP=1 si quiere realizar la carga de un archivo, o UP=0 si quiere realizar una descarga.

Al igual que con otros paquetes, el cliente seguirá reenviando este paquete SYN hasta recibir su ACK correspondiente.

Del lado del servidor, al recibir SYN de un cliente UPLOAD, simplemente envía el SYN ACK correspondiente y se prepara para recibir el archivo. En caso de recibir otro SYN del mismo cliente, se trata del caso en que el SYN ACK se perdió, por lo que vuelve a enviar un SYN ACK sin hacer nada más.

Del lado del servidor, al recibir SYN de un cliente DOWNLOAD, envía el SYN ACK correspondiente y comienza a enviarle el archivo. Del lado del cliente, hasta no recibir ese SYN ACK, los paquetes contenido el archivo serán ignorados por lo que el servidor eventualmente realizará el reenvío por timeout. Si el servidor recibe un SYN duplicado, vuelve a enviar un SYN ACK sin tomar otra acción.

Fin de conexión

El fin de la conexión la realizará el Receiver al recibir un paquete que indica el final del archivo. Es decir el servidor en caso de upload, o el cliente en caso de download. Cuando se recibe este último paquete, envía un paquete FIN de forma habitual, esperando su ACK correspondiente. Una vez recibido el FIN ACK, se cierra la conexión y se liberan los recursos. Del otro lado, al recibir un FIN, se envía un FIN ACK y posteriormente se cierra la conexión y se liberan los recursos. En caso de que este último FIN ACK se pierda, el otro lado seguirá reenviando un paquete FIN hasta que salga por timeout de socket. Estos FIN reenviados no serán procesados del otro lado, ya que en el caso del cliente, ya habrá salido, y del caso del servidor, cerrará la conexión con ese cliente y no volverá a entablar una a menos que se reciba un paquete SYN.

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?

UDP, definido en el documento [RFC 768], hace casi lo mínimo que un protocolo de transporte debe hacer. Además de la función de multiplexación/de multiplexación y de algún mecanismo de comprobación de errores, no añade nada a IP. De hecho, si el desarrollador de la aplicación elige UDP en lugar de TCP, entonces prácticamente es la aplicación la que se comunica directamente con IP. UDP toma los mensajes procedentes del proceso de la aplicación, asocia los campos correspondientes a los números de puerto de origen y de destino para proporcionar el servicio de multiplexación/ de multiplexación, añade dos campos pequeños más y pasa el segmento resultante a la capa de red.

UDP no tiene lugar una fase de establecimiento de la conexión entre las entidades de la capa de transporte emisora y receptora previa al envío del segmento. Por esto, se dice que UDP es un protocolo sin conexión. Algunas de las características de UDP son:

- * Mejor

control en el nivel de aplicación sobre qué datos se envían y cuándo. Con UDP, tan pronto como un proceso de la capa de aplicación pasa datos a UDP, UDP los empaqueta en un segmento UDP e inmediatamente entrega el segmento a la capa de red. Al contrario que con TCP que dispone de otros mecanismos. * Sin establecimiento de la conexión. Como se explicará más adelante, TCP lleva a cabo un proceso de establecimiento de la conexión en tres fases antes de iniciar la transferencia de datos. UDP inicia la transmisión sin formalidades preliminares. Por tanto, UDP no añade ningún retardo a causa del establecimiento de una conexión. * Sin información del estado de la conexión. UDP no mantiene información acerca del estado de la conexión en los sistemas terminales.

TCP es un protocolo de la capa de transporte de Internet, fiable y orientado a la conexión. TCP incluye los mecanismos de detección de errores, las retransmisiones, los reconocimientos acumulativos, los temporizadores y los campos de cabecera para los números de secuencia y de reconocimiento.

Se dice que TCP está 'orientado a la conexión' porque antes de que un proceso de la capa aplicación pueda comenzar a enviar datos a otro, los dos procesos deben primero "establecer una comunicación" entre ellos; es decir, tienen que enviarse ciertos segmentos preliminares para definir los parámetros de la transferencia de datos que van a llevar a cabo a continuación.

Una conexión TCP proporciona un 'servicio full-dúplex': si existe una conexión TCP entre el proceso A que se ejecuta en un host y el proceso B que se ejecuta en otro host, entonces los datos de la capa de aplicación pueden fluir desde el proceso A al proceso B en el mismo instante que los datos de la capa de aplicación fluyen del proceso B al proceso A.

Una conexión TCP casi siempre es una 'conexión punto a punto', es decir, entre un único emisor y un único receptor. La "multidifusión", es decir, la transferencia de datos desde un emisor a muchos receptores en una única operación de envío, no es posible con TCP.

TCP crea un servicio de 'transferencia de datos fiable' sobre el servicio de mejor esfuerzo pero no fiable de IP. El servicio de transferencia de datos fiable de TCP garantiza que el flujo de datos que un proceso extrae de su buffer de recepción TCP no está corrompido, no contiene huecos, ni duplicados y está en orden; es decir, el flujo de bytes es exactamente el mismo flujo que fue enviado por el sistema terminal existente en el otro extremo de la conexión.

TCP proporciona un 'servicio de control de flujo' a sus aplicaciones para eliminar la posibilidad de que el emisor desborde el buffer del receptor. El control de flujo es por tanto un servicio de adaptación de velocidades (adapta la velocidad a la que el emisor está transmitiendo frente a la velocidad a la que la aplicación receptora está leyendo).

Otro componente clave de TCP es su mecanismo de control de congestión. TCP tiene que utilizar un control de congestión terminal a terminal en lugar de un control de congestión asistido por la red, ya que la capa IP no proporciona una realimentación explícita a los sistemas terminales en lo referente a la congestión de la red. El método empleado por TCP consiste en que cada emisor limite la velocidad a la que transmite el tráfico a través de su conexión en función de la congestión de red percibida. Si un emisor TCP percibe que en la ruta entre él mismo y el destino apenas existe congestión, entonces incrementará su

velocidad de transmisión; por el contrario, si el emisor percibe que existe congestión a lo largo de la ruta, entonces reducirá su velocidad de transmisión.

Sobre el uso de cada uno, como es de esperar, el correo electrónico, la Web y la transferencia de archivos, por ejemplo se ejecutan sobre TCP, ya que todas estas aplicaciones necesitan el servicio fiable de transferencia de datos de TCP. No obstante, muchas aplicaciones importantes se ejecutan sobre UDP en lugar de sobre TCP. Por ejemplo, UDP se utiliza para transmitir los datos de la administración de red (SNMP). En este caso, UDP es preferible a TCP, ya que las aplicaciones de administración de la red a menudo se tienen que ejecutar cuando la red se encuentra en un estado de sobrecarga (precisamente en las situaciones en las que es difícil realizar transferencias de datos fiables y con control de la congestión). DNS se ejecuta sobre UDP, evitando de este modo los retardos de establecimiento de la conexión TCP.

Tanto UDP como TCP se utilizan con aplicaciones multimedia, como la telefonía por Internet, las videoconferencias en tiempo real y la reproducción de flujos de vídeos y audios almacenados. Todas estas aplicaciones toleran la pérdida de una pequeña cantidad de paquetes, por lo que una transferencia de datos fiable no es absolutamente crítica para que la aplicación funcione correctamente. Además, las aplicaciones en tiempo real, como la telefonía por Internet y la videoconferencia, responden muy mal a los mecanismos de control de congestión de TCP. Por estas razones, los desarrolladores de aplicaciones multimedia pueden elegir ejecutar sus aplicaciones sobre UDP y no sobre TCP. Sin embargo cuando la tasa de pérdidas de paquetes es baja y teniendo en cuenta que UDP tiene problemas de bloqueo (por seguridad en distintas organizaciones) TCP se está usando más como protocolo para el transporte de flujos multimedia. Aunque hoy en día es común emplear UDP para ejecutar las aplicaciones multimedia, continúa siendo un tema controvertido. Como ya hemos dicho, UDP no proporciona mecanismos de control de congestión, y estos mecanismos son necesarios para impedir que la red entre en un estado de congestión en el que se realice muy poco trabajo útil.

Una breve lista de ejemplos es:

Aplicacion	Protocolo capa aplicacion	Protocolo capa transporte
Web	HTTP	TCP
Correo electrónico	SMTP	TCP
Traducción de nombres	DNS	UDP
Flujos multimedia	-	UDP o TCP
Telefonía(por Internet)	-	UDP o TCP

Figura 45: Tabla comparativa de protocolos aplicación y transporte

7. Dificultades encontradas

Tanto la definición de un protocolo que permita la comunicación confiable como el hecho de verificar la correcta transmisión haciendo simulaciones de perdidas de paquetes fueron alguna de los problemas que mas esfuerzo insumió en la realización del trabajo. El hecho de manejar solamente un socket para el servidor y varios hilos para los diferentes clientes también presentó dificultades. El uso preciso del protocolo a nivel de bits también fue una fuente de errores que motivó algunas iteraciones a la hora de llevar a cabo la implementación.

8. Conclusión

El presente trabajo práctico nos permitió trabajar sobre la implementación de un protocolo de comunicación y poder verificar y probar las diferentes dificultades a la hora de hacerlo. Los diferentes tipos de algoritmos implementados, a saber, Stop & Wait y GBN testifican que las diferentes implementaciones tienen un impacto importante a la hora de llevar a cabo tal implementación, así como los tiempos insumidos en la carga y en la descarga de los archivos. Las pruebas con perdidas de paquetes y con archivos de diferentes tamaños y tipos también permitieron ver y verificar que algunos simples detalles de implementación, como el tamaño del buffer a usar en la lectura y/o escritura pueden ocasionar bugs que en principio parecen no tan perceptibles.