FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Introducción a los Sistemas Distribuidos (75.43)
Trabajo Práctico 1

File Transfer

Grupo N° 17 - 2° Cuatrimestre 2022

Integrantes:
CINALLI, MARIANO ROBERTO - 95456
MASSONE, MARIO BERNARDO - 102141
MENDOZA CORONADO, KEVIN MARCELO - 98038
ROLDAN, MARIA CECILIA - 101939
WALDMAN, LUIS ESTEBAN - 79279

Índice

Introducción

El presente trabajo práctico tiene como objetivo la creación de una aplicación de red que permita la transmisión de archivos de modo fiable. Para tal finalidad, fue necesario comprender cómo se comunican los procesos a través de la red y cuál es el modelo de servicio que la capa de transporte le ofrece a la capa de aplicación.

Además, para poder lograr el objetivo planteado, aprendimos el uso de la interfaz de sockets UDP que ofrece Python y los principios básicos de la transferencia de datos confiable.

Analizamos dos protocolos: Stop-and-wait y Go-Back-N.

En el protocolo Stop-and-wait el emisor envía un paquete y se queda esperando la confirmación de recepción del mismo (llamado ACK, por acknowledgement).Una vez recibida la confirmación, envía un nuevo paquete repitiendo el ciclo hasta enviarlos a todos. En caso de quedarse esperando un ACK y no recibirlo por un tiempo, reenvía el mismo paquete hasta obtener la confirmación.

En el protocolo Go-Back-N el emisor envía una ráfaga de N paquetes y espera las confirmaciones afirmativas sobre las recepciones de los mismos, a medida que van llegando, va mandando nuevos paquetes. Llamamos “ventana” al total de paquetes en vuelo; decimos entonces que a medida que llegan los ACK se va corriendo la venta hasta terminar de pasar el mensaje.
Dependiendo de las implementaciones, el receptor puede solo aceptar el paquete que estaba esperando en orden y descarta otros si se “adelantan”. Del mismo modo, si el emisor recibe una confirmación afirmativa con un número de secuencia mayor al esperado, entiende que el menor esperado, o los menores al número recibido, también fueron recibidos con éxito.

Observamos que estos protocolos son similares y solo difieren en la cantidad de paquetes en vuelo y las estrategias para reconocer los mismos. Es importante remarcar que el protocolo de Stop-and-Wait está contenido en GBN, ya que el Stop-and-Wait es un GBN con N=1.

Hipótesis y suposiciones realizadas

Se listan a continuación las distintas hipótesis y suposiciones realizadas en el análisis previo a la hora implementación de la solución propuesta.

- Para la descarga de un archivo desde el servidor, si el mismo no existe la conexión se cierra, mostrando un error por pantalla.
- Para la carga de un archivo al servidor, si el mismo ya existe la conexión se cierra, mostrando un error por pantalla.
- No se enviarán archivos mayores a 5MB tanto para la carga como la descarga.
- No es necesario configurar MSS para la transferencia, todos los segmento UDP puede tener un payload máximo de 65 KB.
- La carpeta indicada para el almacenamiento o descarga de archivos debe existir, de lo contrario la aplicación termina mostrando un error por pantalla.
- No se permiten distintos archivos con el mismo nombre.
- Si cliente y servidor corren en el mismo host, no pueden usar el mismo directorio para leer y escribir archivos.

Implementación

Sockets

Para comunicar dos procesos en terminales distintas se utilizaron sockets UDP. Se utilizaron estos antes que sockets TCP ya que el trabajo práctico tenía como objetivo implementar una transferencia de datos confiables, servicio que TCP la provee.

Mensajes

Los mensajes se modelaron a través de 8 diferentes tipos:

- **HOLA:** Primer mensaje que envía el cliente al servidor y viceversa.
- **HOLA ACK:** Mensaje de confirmación afirmativa del cliente hacía el servidor para comenzar con la transferencia de datos.
- **PARTE:** Mensaje que contiene un parte de datos del archivo en transferencia.
- **ACK:** Mensaje sin datos en payload para confirmar afirmativamente la recepción de un paquete.
- **ERROR:** Mensaje que contiene notificación de un error ocurrido.
- **CHAU:** Mensaje de cierre de conexión del emisor al receptor.
- **OBTENER LISTADO:** Mensaje del cliente que solicita el listado de los archivos que brinda para la descarga.

Operaciones

La implementación cuenta con tres tipos de operaciones:

- **DOWNLOAD:** Transferencia de un archivo del servidor hacia el cliente.
- **UPLOAD:** Transferencia de un archivo del cliente hacia el servidor.
- **LISTADO:** Obtiene la lista de archivos disponibles para descargar

Paquetes

Los paquetes contienen un fragmento de datos del archivo que se encuentra en transferencia. Estos se envían a través de los sockets UDP antes mencionados.

Cabecera de paquetes

- Tipo de paquete: Contiene un valor numérico de la suma del tipo de mensaje y el tipo de operación. Los valores de cada tipo son:

    - Tipo de mensajes:
        - HOLA: 1
        - CHAU: 2
        - PARTE: 3
        - ACK: 4
        - ERROR: 5
        - OBTENERLISTADO: 6
        - HOLA_ACK: 7

    - Tipo de operación:
        - DOWNLOAD: 10
        - UPLOAD: 20

- Cantidad total de partes: Cantidad total de paquetes distintos con datos que debería recibir el receptor del archivo.

- Parte en vuelo: Número de referencia del paquete actual.

- Tamaño del payload: Tamaño en bytes de los datos contenidos en ese paquete.

- Checksum

Estructura del paquete

Tipo de paquete (1 Byte)	**Cantidad total de partes** (1 Byte)	**Parte en vuelo** (1 Byte)	**Tamaño del payload** (2 Bytes)	**Checksum** (2 Bytes)
Payload (64 KBytes)				

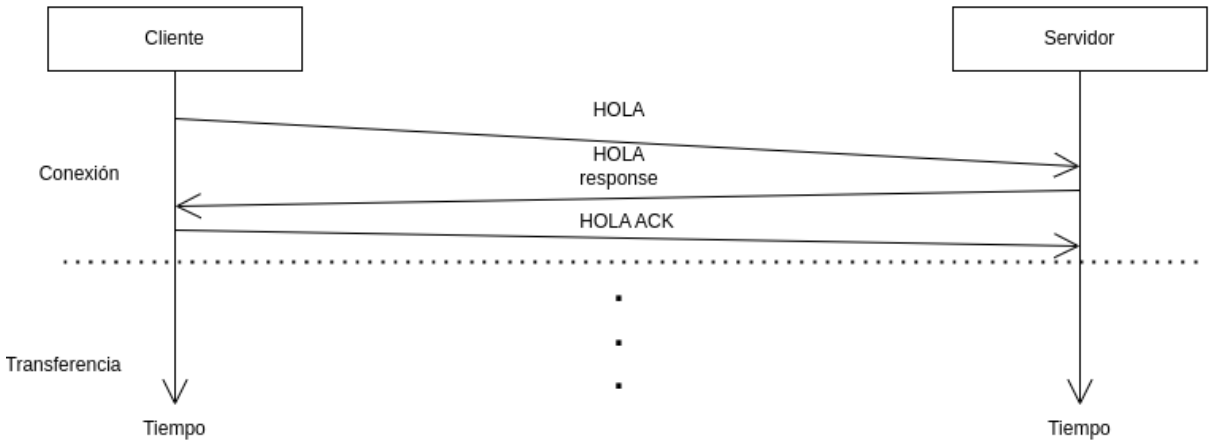
Checksum

Se calcula utilizando los otros cuatro encabezados (tipo de paquete, cantidad total de partes y tamaño del payload) y el payload (en caso de que haya). Dado que el payload no es un valor numérico, se lo debe transformar a un número para sumar con el resto de los encabezados. Primero se aplica la función de hash SHA256 al payload y luego se suman de a dos bytes. Por último se calcula la suma de los encabezados y el valor calculado del payload. Para evitar overflow en caso de que esta suma supere los dos bytes, nos quedamos con los últimos 16 bits para calcular su complemento y agregarlo al header.
Al recibir el paquete se vuelve a calcular el checksum y se verifica que este valor más el valor del header sumen 65.535.
Esto se realiza en la clase src.Traductor, que es la encargada de transformar mensajes en paquetes y paquetes a mensajes

Conexión

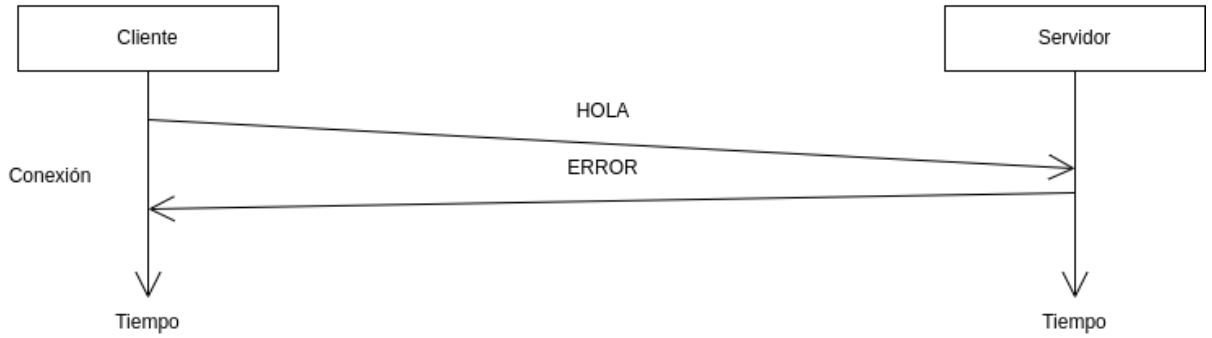
El cliente (download.py o upload.py) envía un mensaje HOLA al servidor en el que incluye el tipo de operación que está interesado en realizar en el campo “Tipo de paquete” y el nombre del archivo que desea descargar o subir en el payload.
El servidor recibe la petición, realiza un chequeo sobre el archivo pedido/ofertado (en el metodo Servidor.assert_archivo) y responde con un mensaje HOLA en caso afirmativo o con un mensaje ERROR en caso negativo.
En el caso afirmativo, el cliente recibe el mensaje y contesta con un mensaje HOLA ACK para comenzar la transferencia.

Conexión afirmativa



En el caso negativo, el cliente recibe el mensaje y finaliza el intento de conexión.

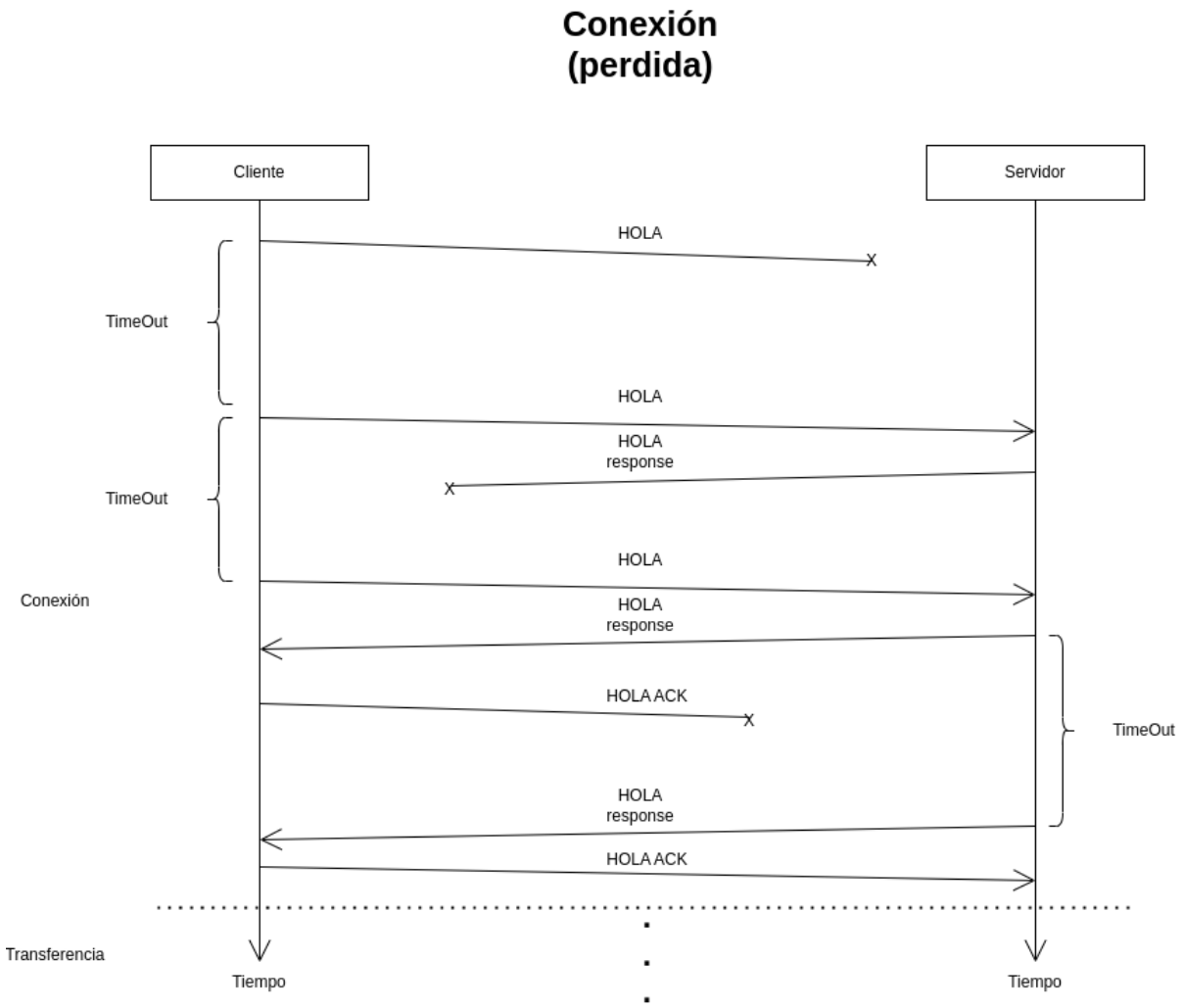
Conexión negativa



Si se produce pérdida del paquete HOLA, el cliente vuelve a enviarlo cuando se produce timeout.
Si se produce pérdida del paquete HOLA response, puede detectarse porque el cliente mandó otro mensaje HOLA, o porque se produce timeout.

Si se produce pérdida del HOLA ACK, el servidor enviará el HOLA response.
En el caso de que el cliente es emisor y el servidor receptor, el cliente luego de enviar el HOLA ACK va a proceder enviar los paquetes con partes del archivo. Si durante el comienzo del envió recibe un HOLA response, quiere decir que el servidor no ha recibido el HOLA ACK, por lo que el cliente envía otro HOLA ACK y reinicia la transferencia de partes.
Por otro lado, en el caso de que el cliente es receptor y el servidor emisor, el cliente luego de enviar el HOLA ACK va a proceder a esperar paquetes con partes del archivo. Si durante el comienzo de la recepción de paquetes de partes recibe un HOLA response, quiere decir que el servidor no ha recibido el HOLA ACK, por lo que el cliente envía otro HOLA ACK y reinicia la espera de paquetes con partes del archivo.

A continuación se muestra un gráfico que muestra algunas de las cuestiones mencionadas:

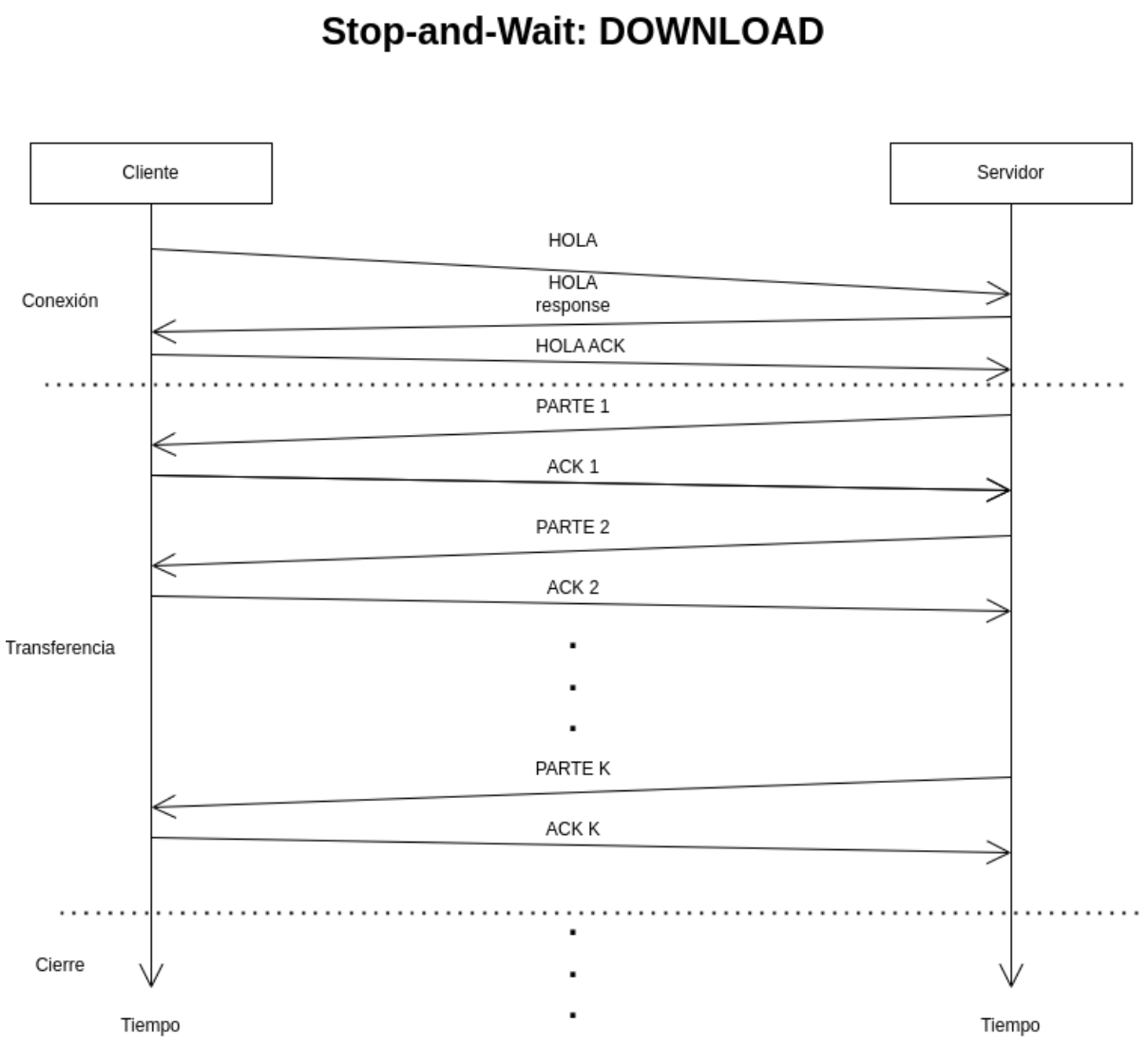


Transferencia de datos:

Dado que Stop-and-wait es un caso especial de Go-Back-N, el Stop-and-wait es equivalente a GBN con N = 1, se decidió implementar solo GBN y hacer el N variable a través de un parámetro de entrada.

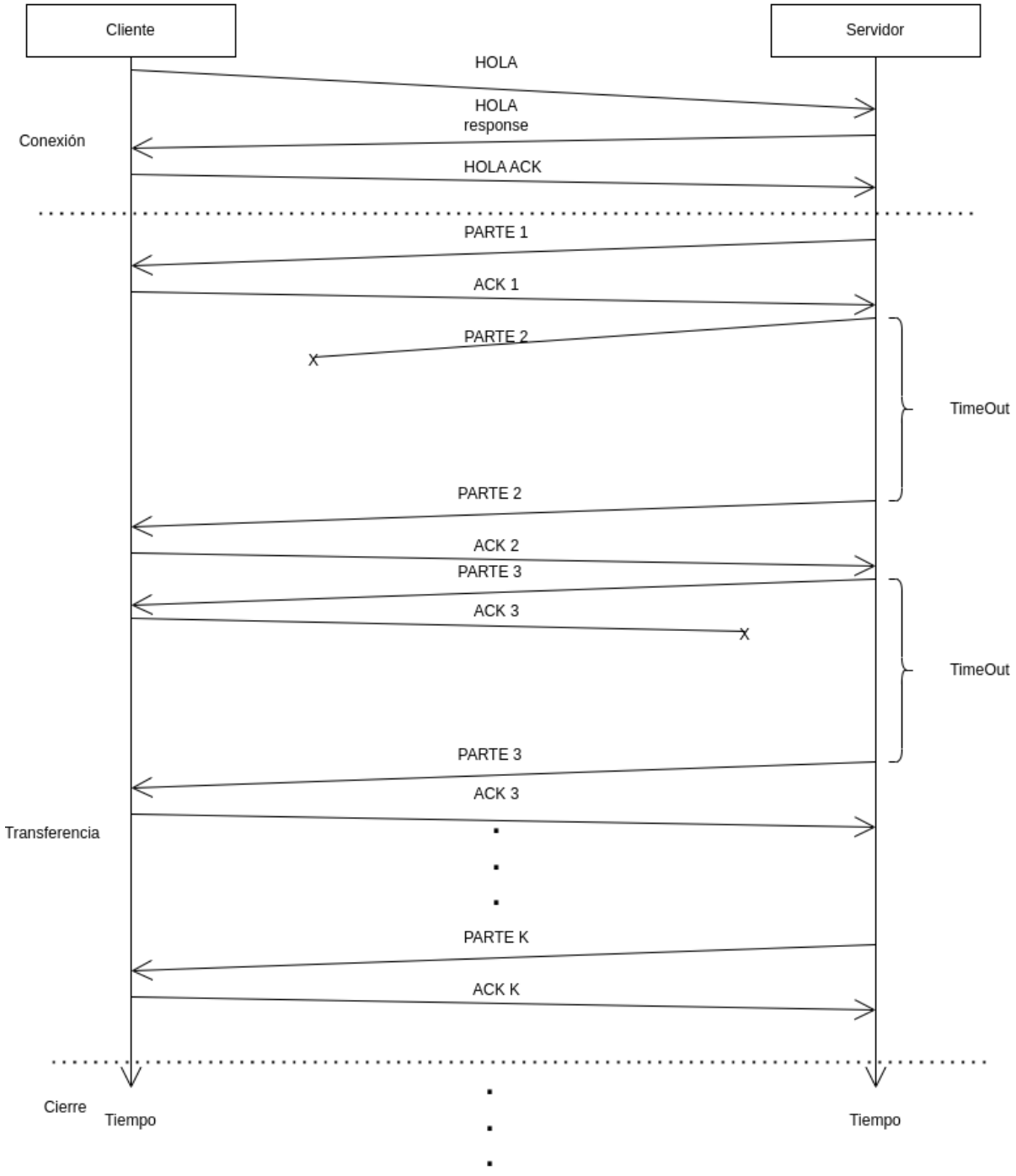
DOWNLOAD: Stop-and-wait

Luego de una conexión exitosa comienza la transferencia de datos. En este caso para la descarga, el servidor luego de recibir el HOLA ACK comienza a enviar un paquete, esperar un ACK de dicho paquete, y luego enviar el siguiente.



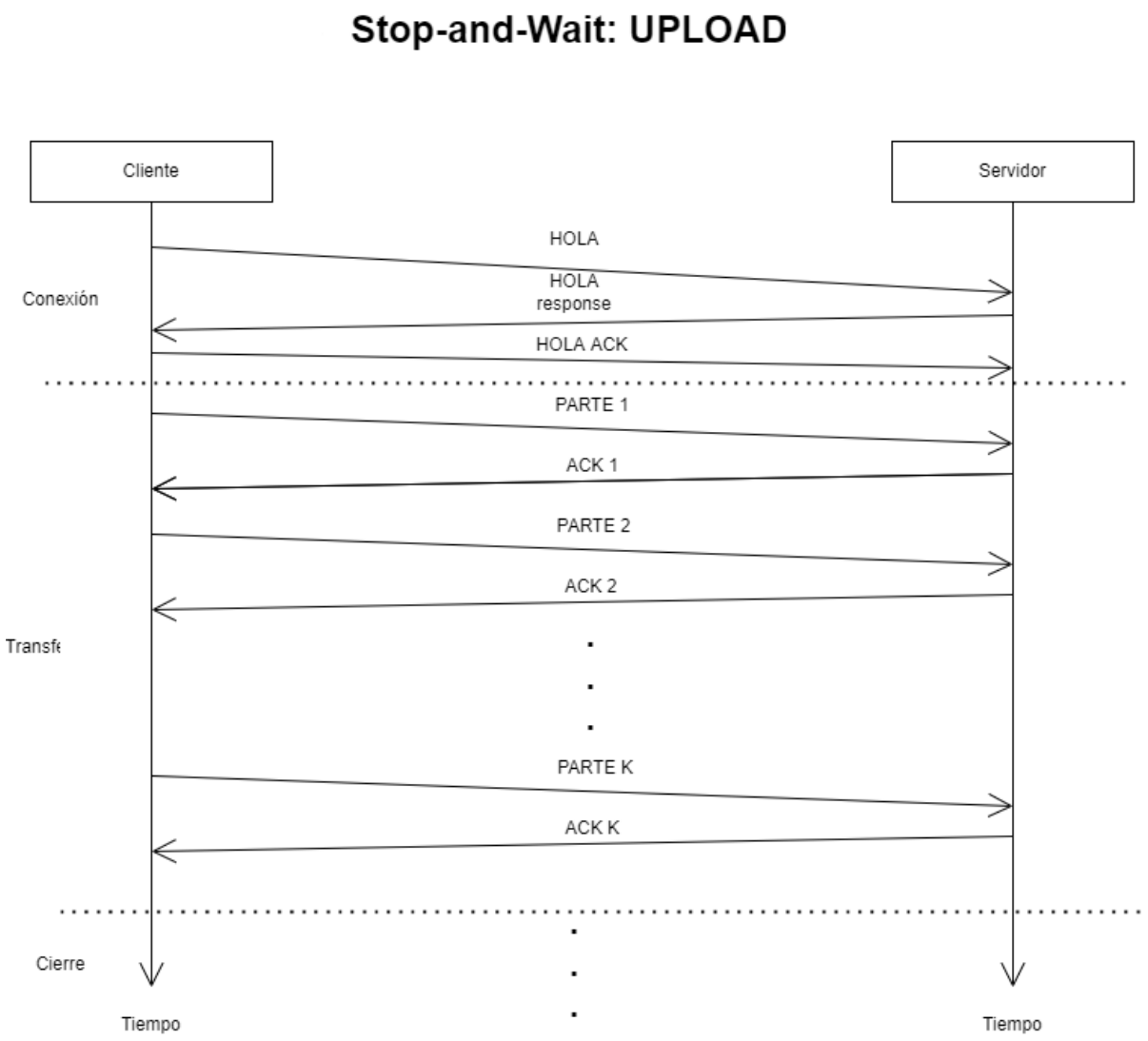
Si se produce un timeout esperando el ACK, se envía de nuevo. Esto puede generar la pérdida del paquete con la parte, la pérdida del paquete con el ACK o simplemente demora excesiva por la congestión en la red.

Stop-and-Wait: DOWNLOAD (perdida)



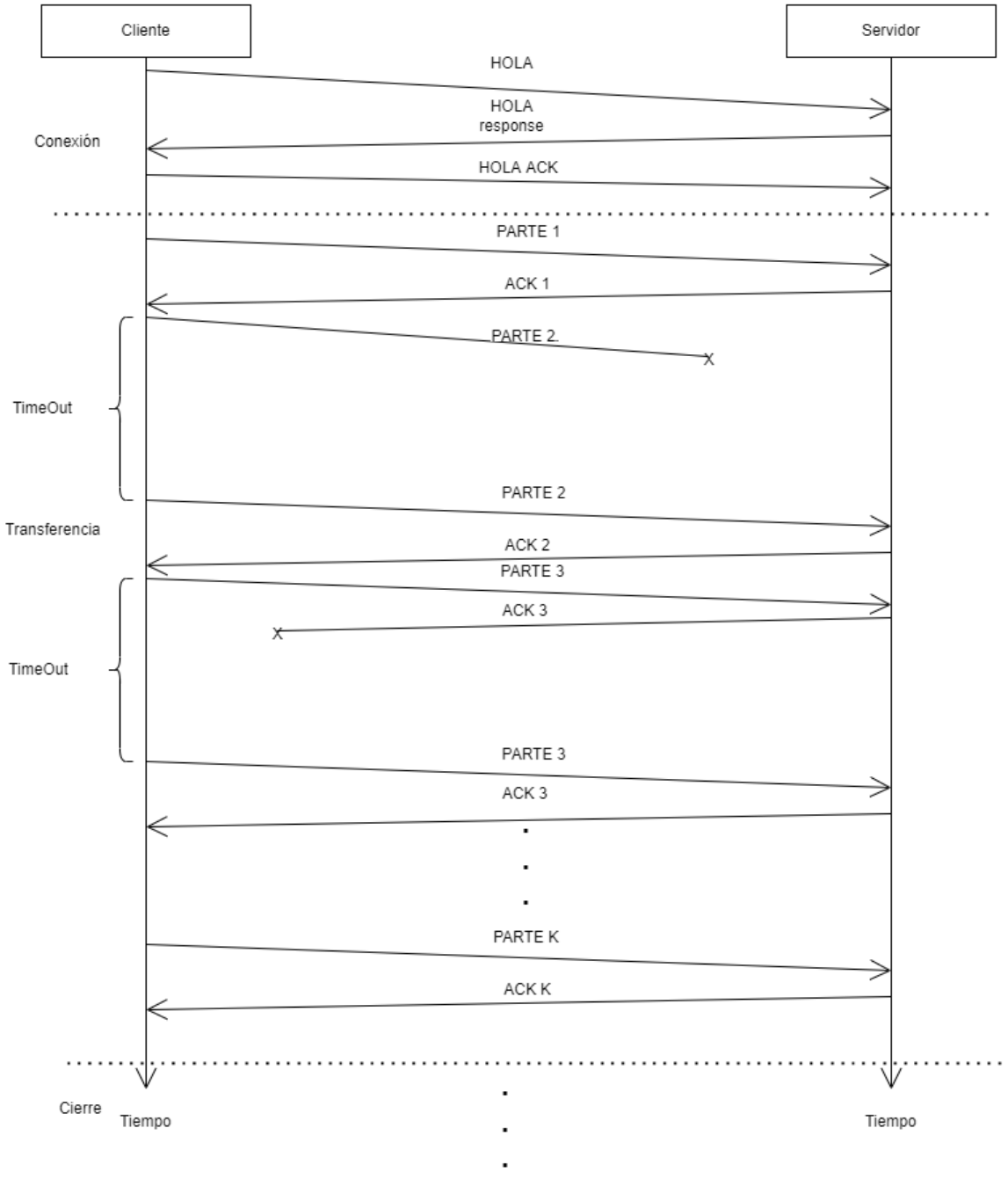
UPLOAD: Stop-and-wait

Luego de una conexión exitosa comienza la transferencia de datos. En este caso para la descarga, el cliente luego de enviar el HOLA ACK comienza a enviar un paquete, esperar un ACK de dicho paquete, y luego enviar el siguiente.



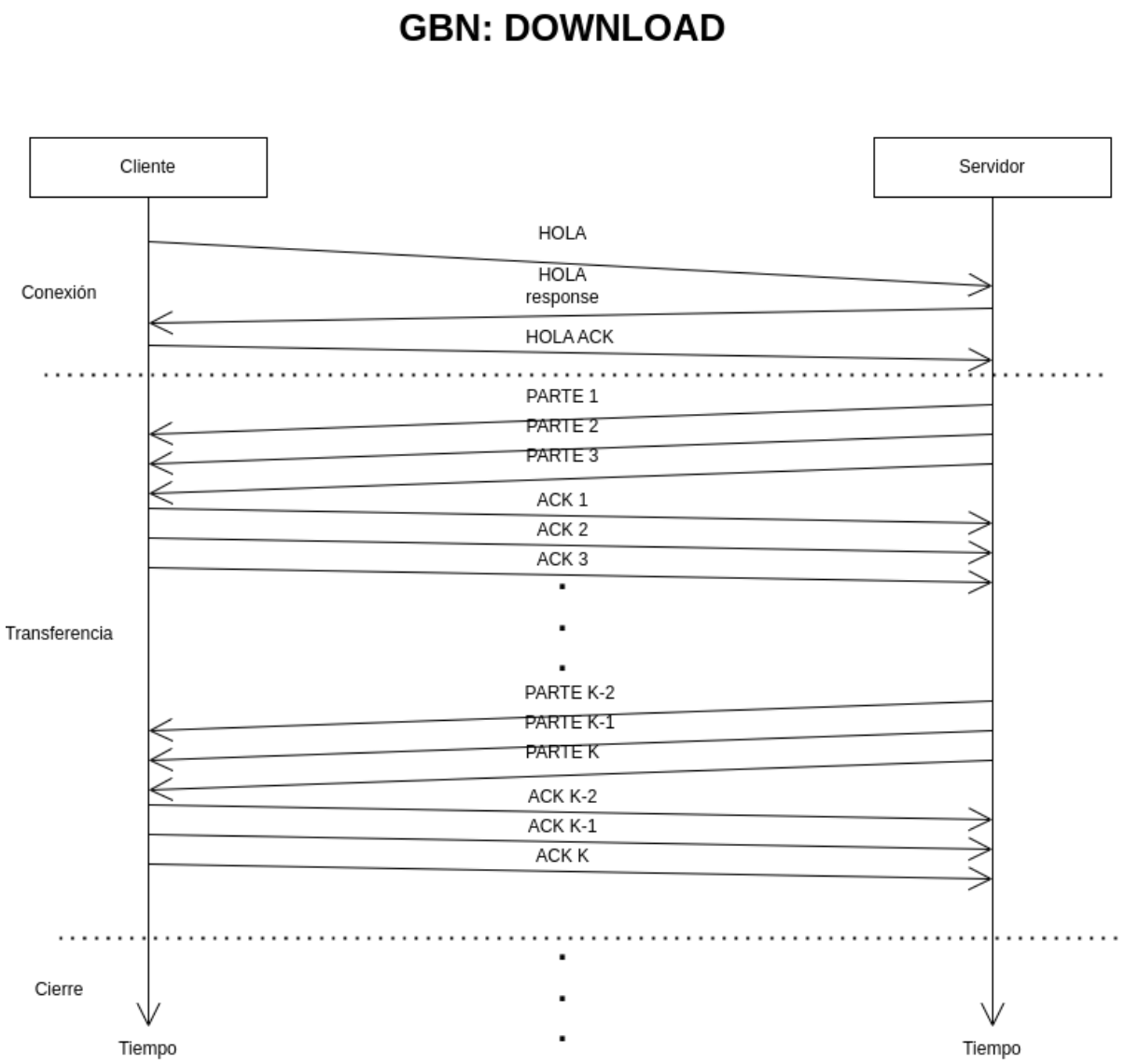
Si se produce un timeout esperando el ACK, se envía de nuevo. Aquí puede ocurrir la pérdida del paquete con la parte, la pérdida del paquete con el ACK o simplemente demora excesiva por la congestión en la red.

Stop-and-Wait: UPLOAD (perdida)



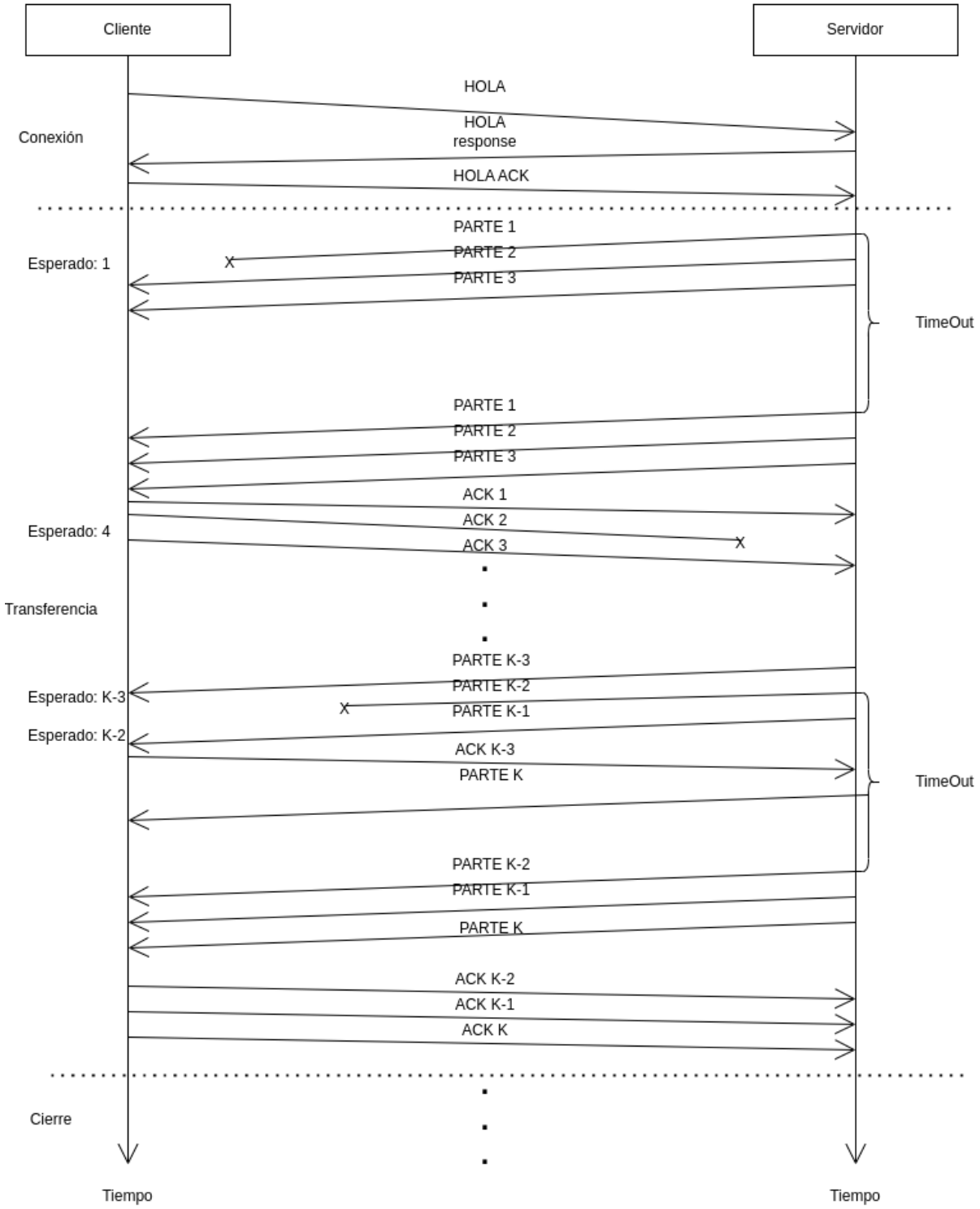
DOWNLOAD: Go-Back-N

Luego de una conexión exitosa comienza la transferencia de datos. En este caso para la descarga, el servidor luego de recibir el HOLA ACK comienza a enviar rafagas de paquetes, esperar los ACK y a medida que van llegando, si es el mínimo esperado, envía un nuevo paquete. A continuación se muestra la dinámica de transferencia. A fines de clarificar la ilustración se muestra la ráfaga de ACKs luego de la recepción de paquetes, pero en realidad son a medida que llegan los ACK.



Si se produce un timeout esperando el ACK de menor número de parte, se envía de nuevo la ráfaga de paquetes nuevamente. Observando el siguiente gráfico podemos distinguir los distintos casos de pérdidas y reacciones:

GBN: DOWNLOAD (perdida)

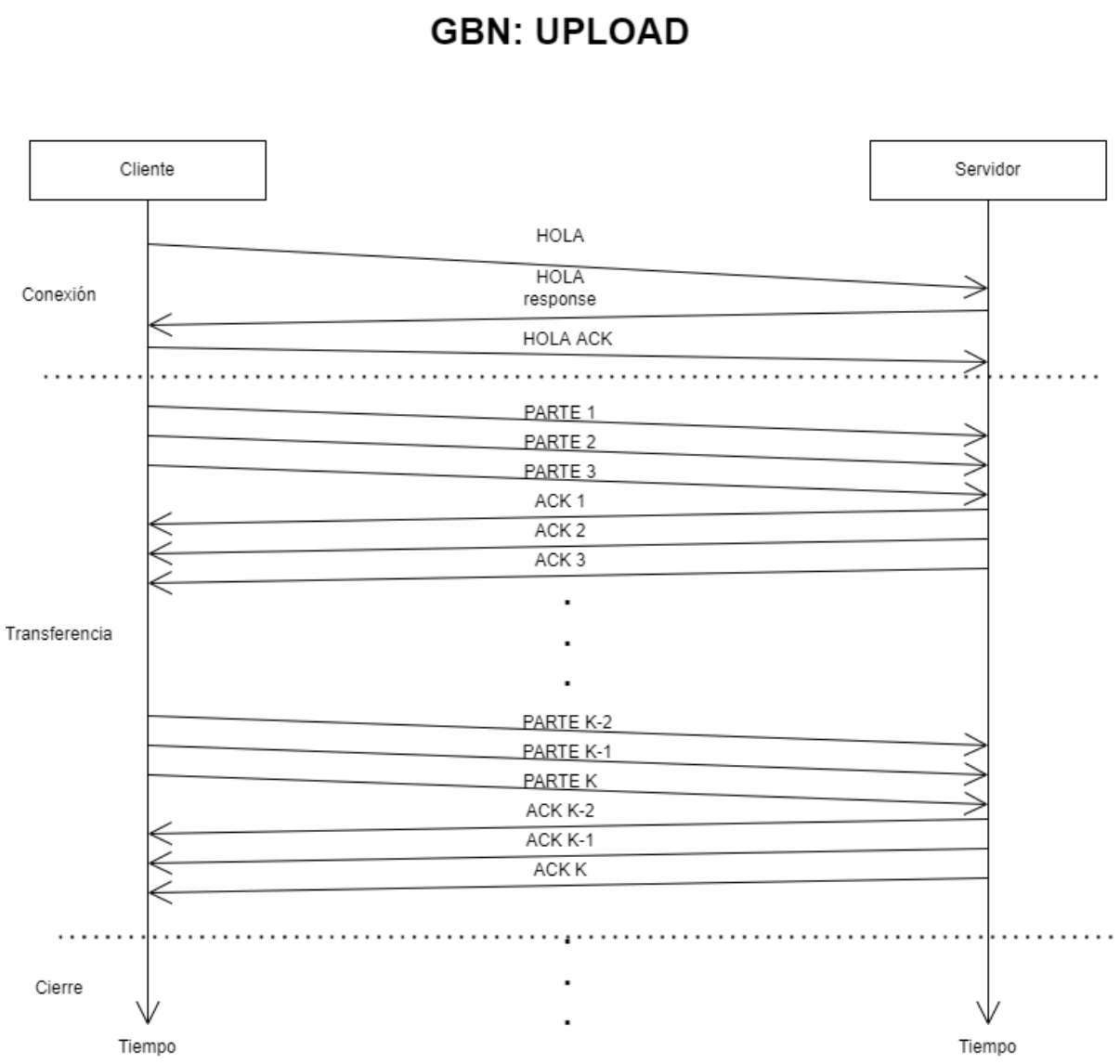


- El receptor (en este caso el cliente) descarta los paquetes con número de parte mayor al esperado.
- Si un ACK se pierde, pero el emisor (en este caso el servidor) recibe uno mayor, lo acepta y actualiza el mínimo esperado como su siguiente. Esto es porque los ACKs

indican que se recibió afirmativamente esa parte y todas las anteriores por lo nombrado en el ítem anterior.

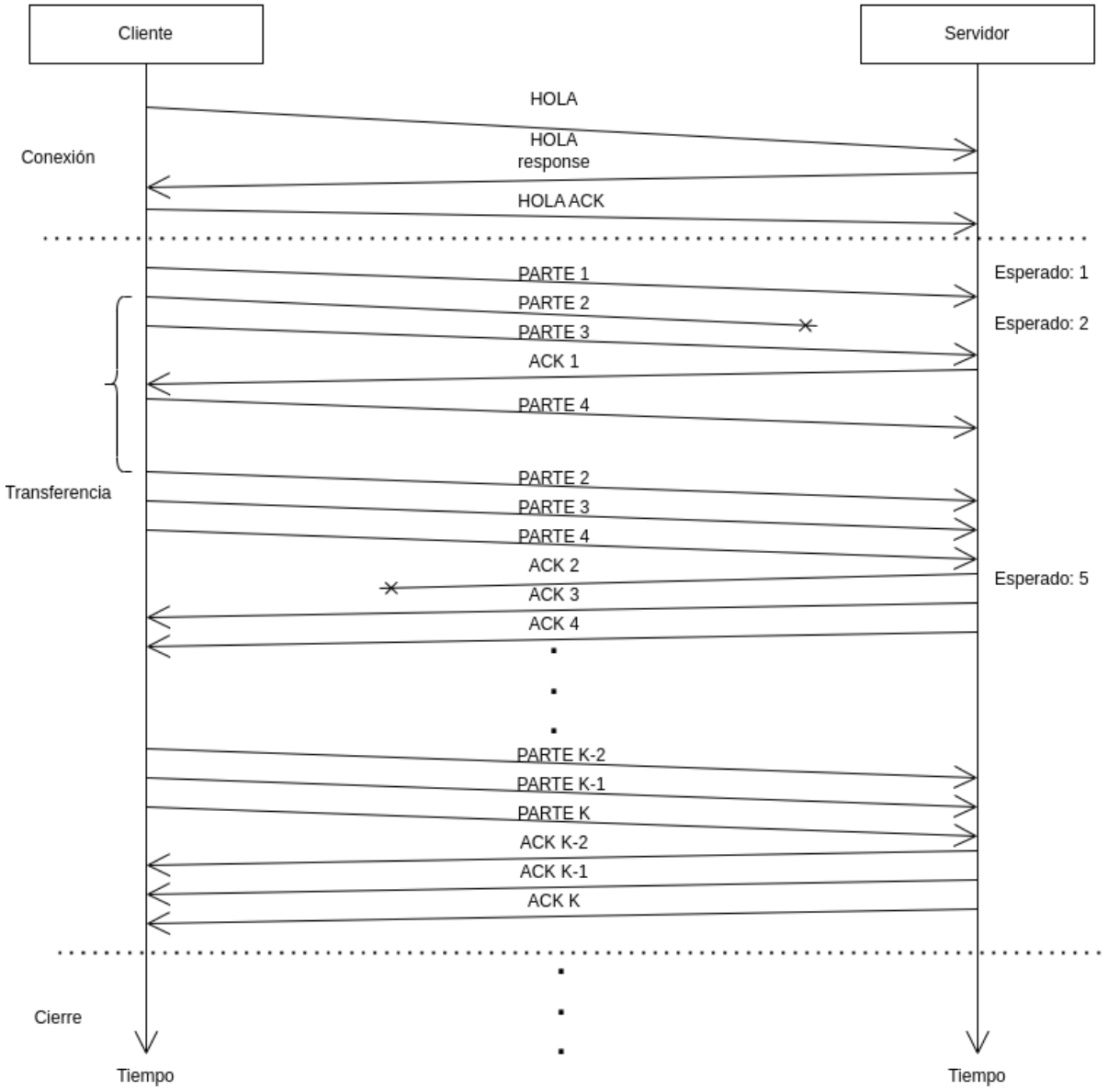
UPLOAD: Go-Back-N

De manera similar, salvo que ahora el cliente es el emisor y el servidor el receptor, luego de una conexión exitosa comienza la transferencia de datos.
El cliente envía rafagas de paquetes, espera los ACK y a medida que van llegando, si es el mínimo esperado, envía un nuevo paquete.



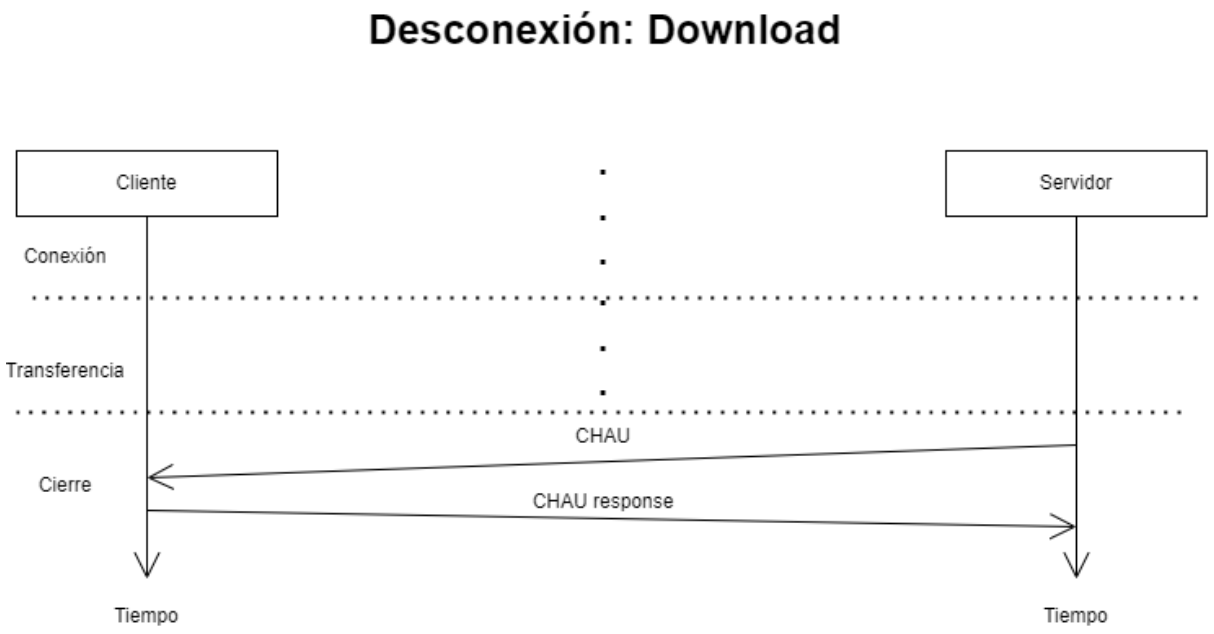
Si se produce un timeout esperando el ACK de menor número de parte, se envía de nuevo la ráfaga de paquetes nuevamente.

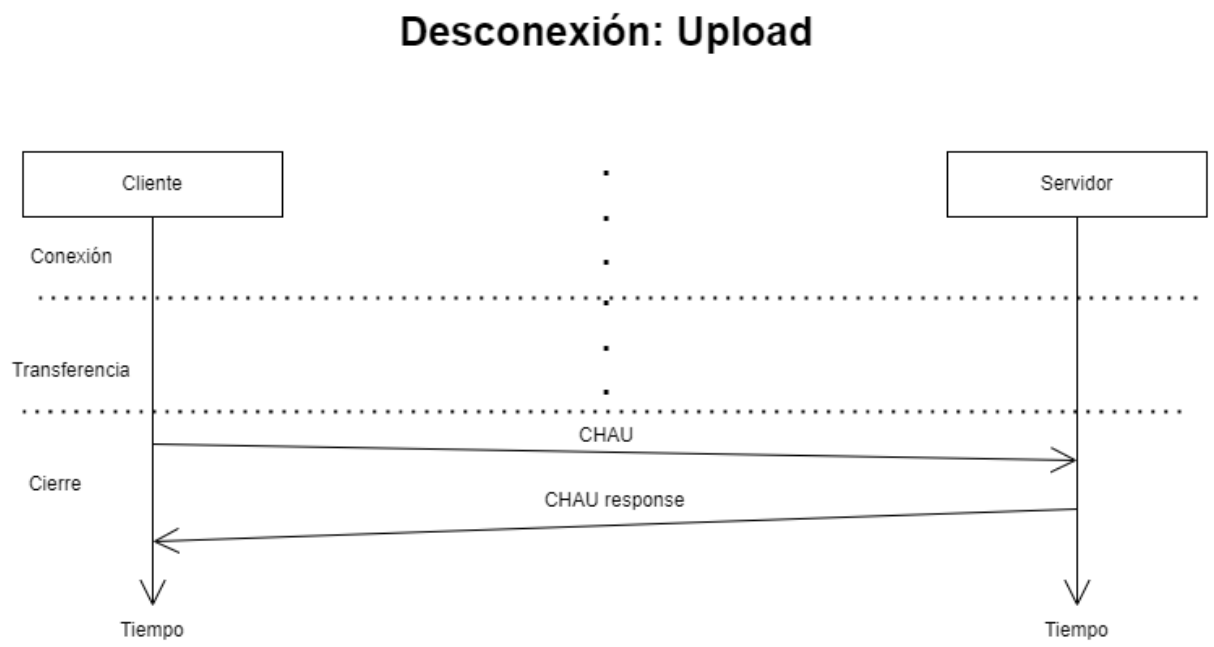
GBN: UPLOAD


Cliente
Servidor
HOLA
HOLA response
HOLA ACK
Conexión
PARTE 1
PARTE 2
PARTE 3
ACK 1
PARTE 4
Esperado: 1
Esperado: 2
Transferencia
PARTE 2
PARTE 3
PARTE 4
ACK 2
ACK 3
ACK 4
Esperado: 5
PARTE K-2
PARTE K-1
PARTE K
ACK K-2
ACK K-1
ACK K
Cierre
Tiempo
Tiempo

Desconexión

El emisor de datos, el servidor en download y el cliente en upload, una vez que recibe todos acks de los paquetes de partes de datos finaliza la conexión enviando un mensaje CHAU y esperando su respuesta.



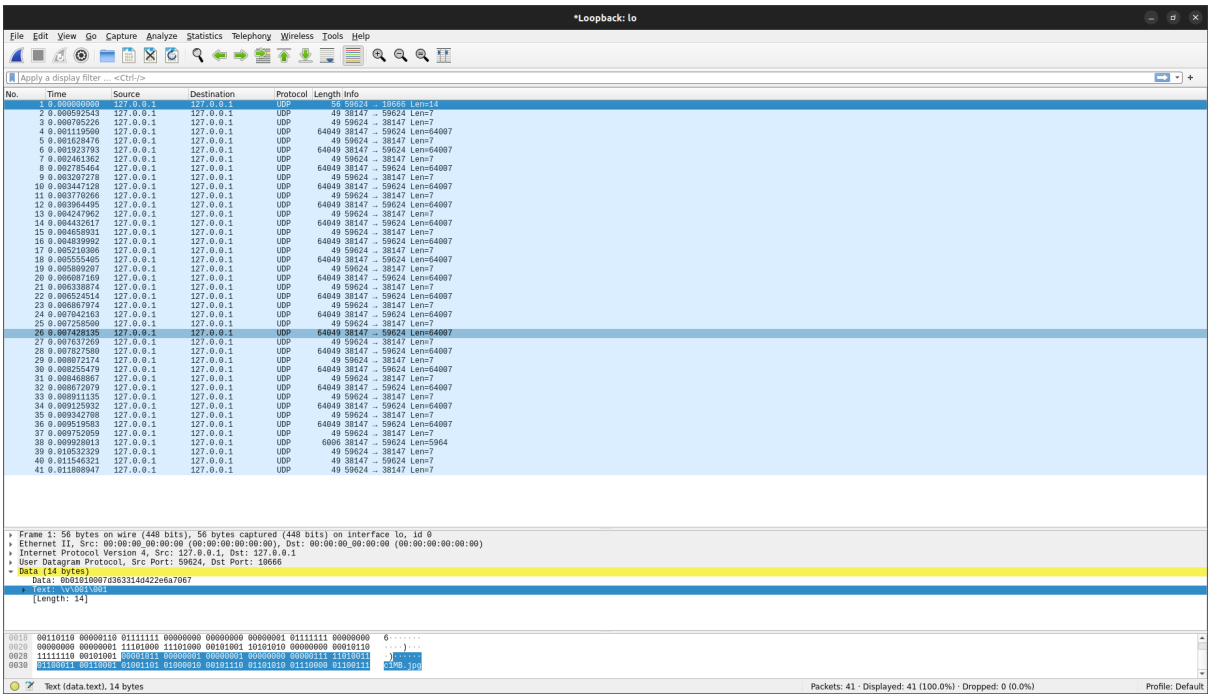


Al igual que los mensajes nombrados anteriormente, si el mensaje CHAU se pierde, se produce un timeout que indica el reenvío.
Si luego de 5 envíos no se obtiene respuesta, se cierra la conexión.

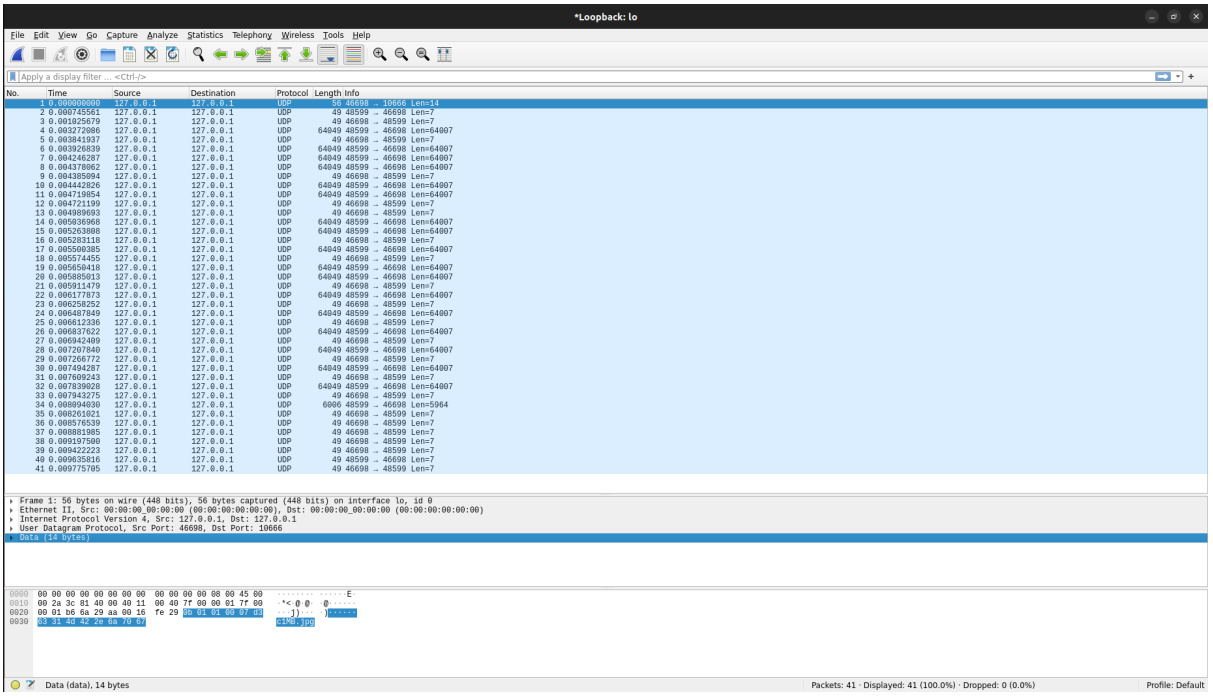
Pruebas

Descarga sin pérdida

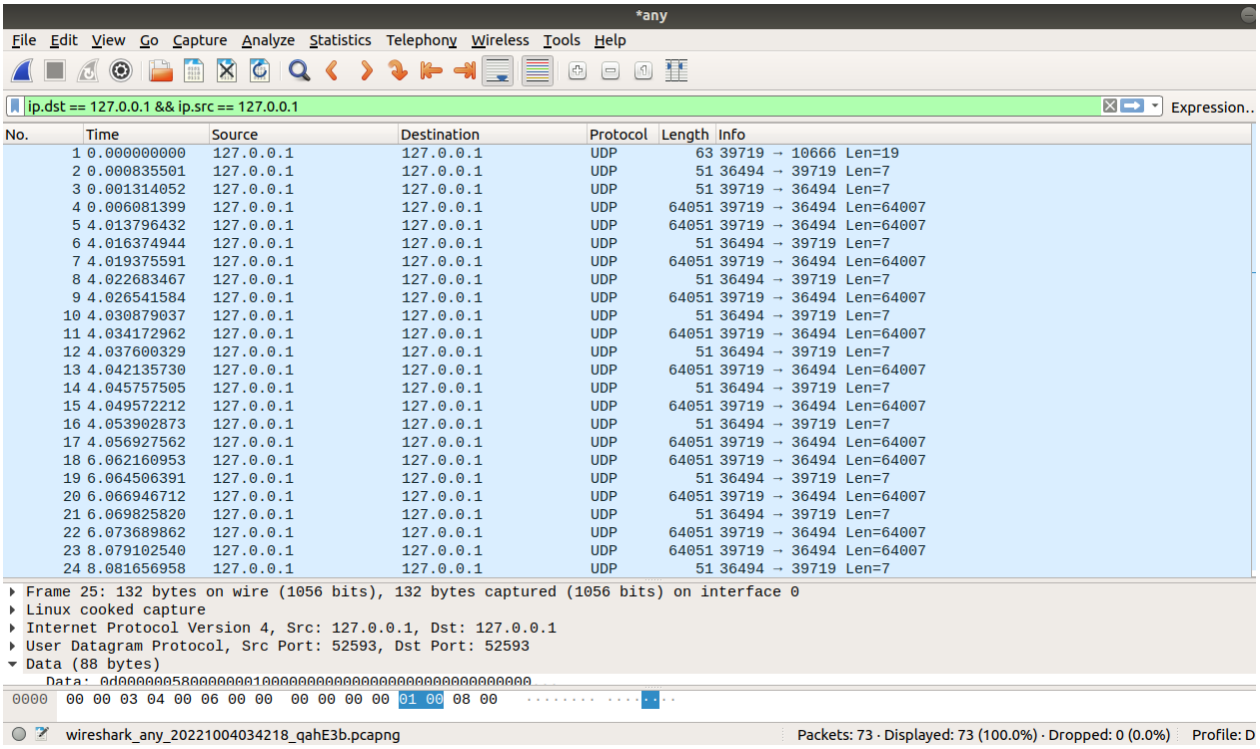
Para las siguientes pruebas se utilizó un archivo de 1MB.



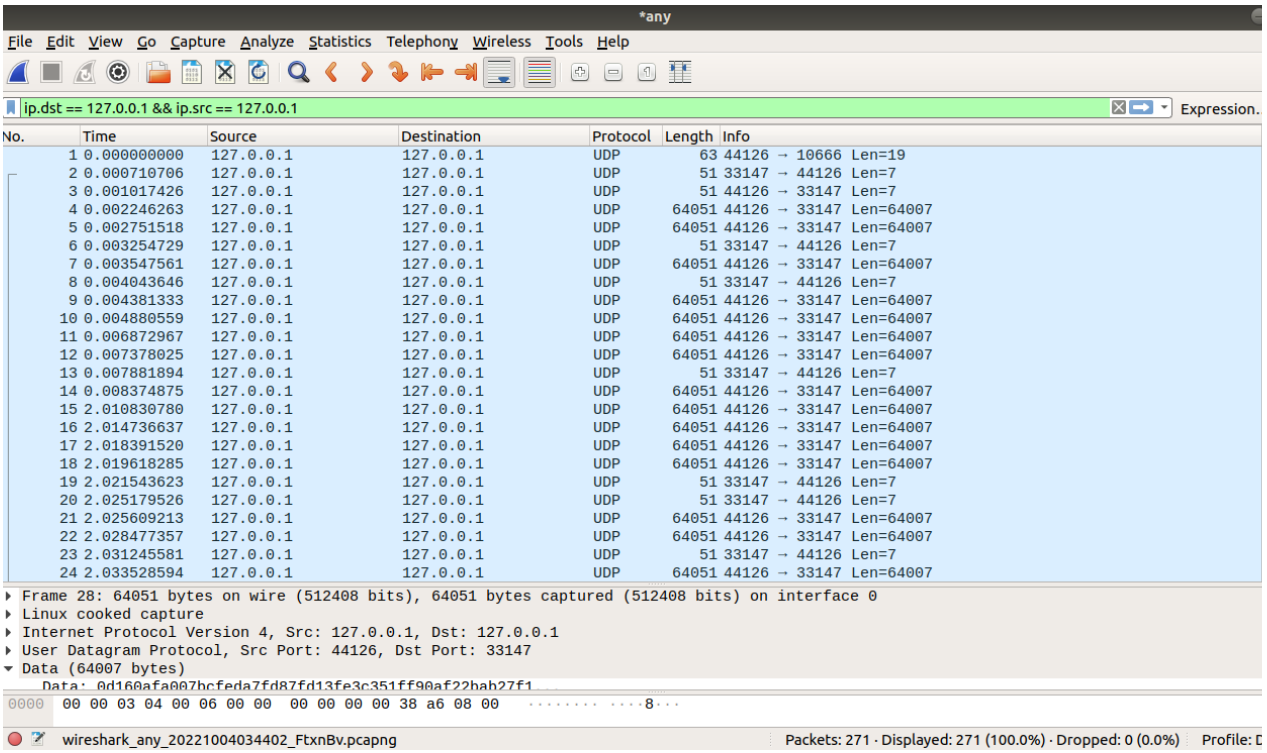
Descarga sin pérdida utilizando Stop and wait (GBN N=1)



Descarga sin pérdida utilizando GBN N=5



Descarga con pérdida del 20% utilizando Stop and wait (GBN N=1)



Descarga con pérdida del 20% utilizando GBN N=5

Descripción

Handshake:

Los puertos mencionados a continuación son de la imágen: “Descarga sin pérdida utilizando GBN N=5” .

1. Mensaje **HOLA**, enviado del cliente, en el puerto 46698, al servidor, en el puerto 10666, pidiendo el archivo *1MB.jpg*
2. El servidor, ahora en 48599 porque se creó un nuevo thread para atender el pedido, envía un mensaje **HOLA** al cliente en el puerto 46698.
3. El cliente envía un **ACK** del servidor

Transferencia:

Los paquetes 4 a 39 corresponden a la transferencia del archivo.
El servidor envía 18 paquetes al cliente, a su vez el cliente envía 18 **ACK** al servidor.

Desconexión:

Los paquetes 40 y 41 corresponden a la desconexión.
El paquete 40 es el mensaje **CHAU** enviado por el servidor y el 41 es el **ACK** enviado por el cliente.

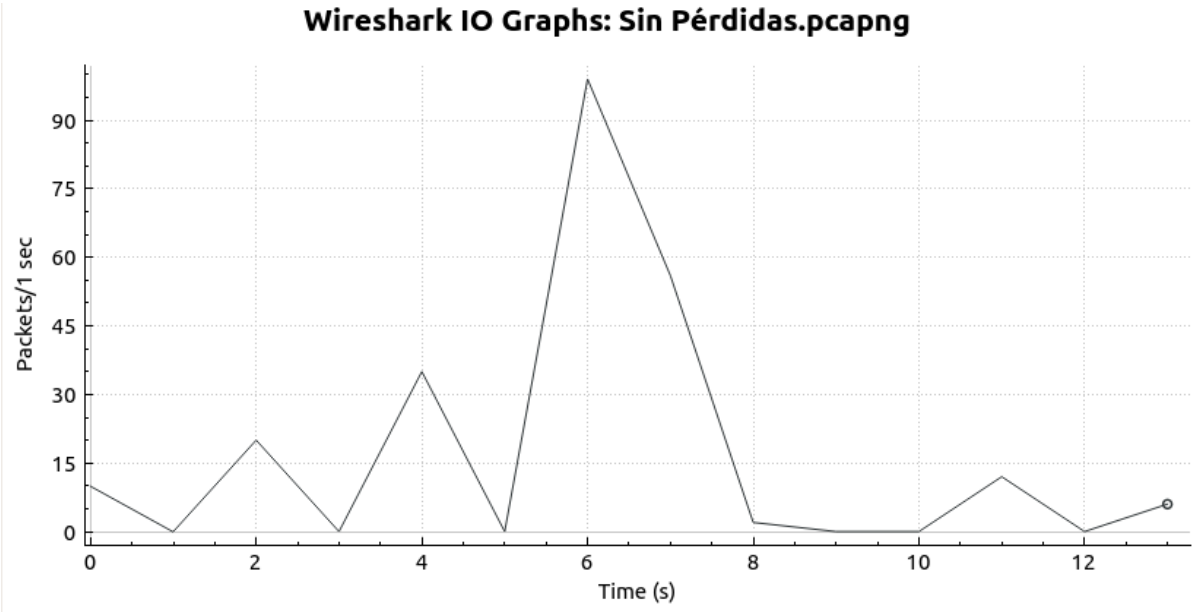
Observaciones

En la captura de Stop and Wait se ve claramente que el servidor envía un paquete y espera a recibir el ACK. Por el contrario, en la captura de GBN N = 5, no se pueden distinguir las ráfagas. Esto probablemente es causado por:

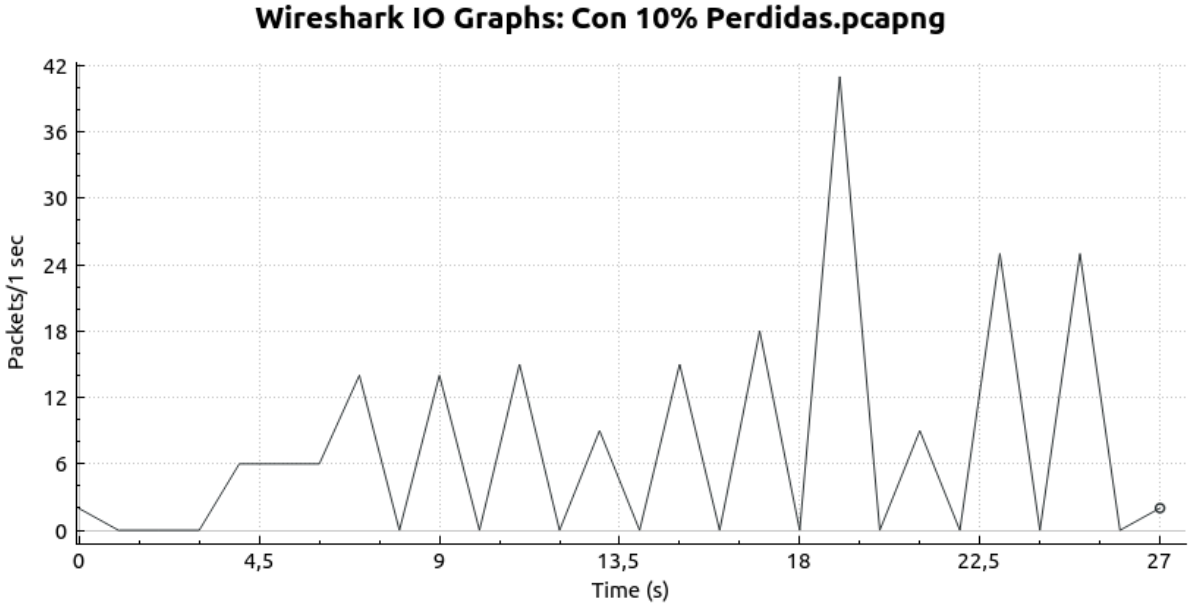
- Por cada paquete dentro de la ráfaga, el servidor crea un thread y al momento de que llegue el paquete al cliente este lo escribe y hace ACK. Por lo tanto, se observa que llegan ACK del cliente mientras se están enviando parte de las ráfagas del cliente.
- El cliente tiene que escribir el archivo y esto le causa overhead. Esto contribuye a que los ACK lleguen de manera impredecible.

Análisis del tiempo con wireshark

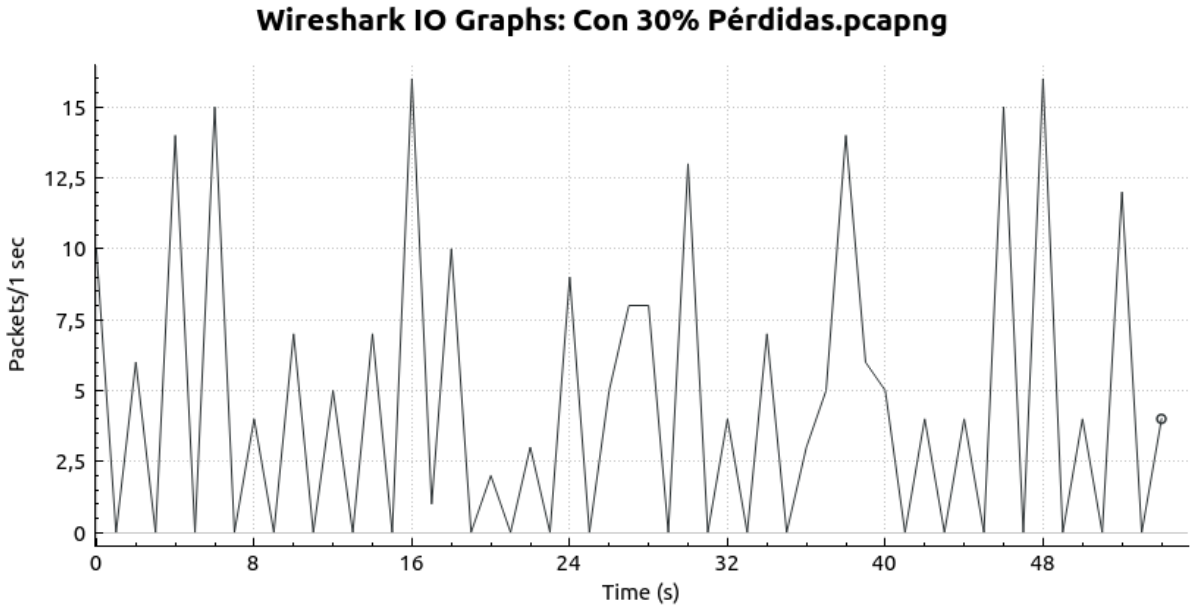
Por otro lado se evaluó mediante wireshark el tiempo de transmisión total y la cantidad de paquetes enviados transfiriendo un archivo de 4 MB utilizando el protocolo GBN N = 5. El resultado lo podemos ver en el siguiente gráfico:



Podemos ver que el tiempo total de transmisión superó los 12 segundos y la cantidad de paquetes enviados en el tiempo fue variada, con un máximo de más de 90 paquetes por segundo. Luego se decidió comparar estos resultados transfiriendo el mismo archivo con el mismo protocolo y en este caso estableciendo una pérdida del 10% de los paquetes.



En este nuevo gráfico podemos ver que aumentó el tiempo de transferencia a más del doble agregando pérdidas. Por otro lado, sigue la oscilación en la cantidad de paquetes enviados por segundo, pero en este caso el máximo de paquetes llega a los 42. Se decidió aumentar aún más la pérdida de paquetes para analizar su influencia en el tiempo de transferencia. Se estableció la pérdida del 30% de los paquetes:



En este caso podemos ver que la cantidad máxima de paquetes enviados por segundo bajó y en consecuencia el tiempo total aumentó. Comparando con el envío sin pérdidas el tiempo de transferencia fue de cuatro veces más, pasó de 12 a 50 segundos, un aumento considerable.

Preguntas a responder

1. Describa la arquitectura Cliente-Servidor.

La arquitectura Cliente-Servidor es un modelo de diseño en el que dos procesos establecen una comunicación. El proceso que inicia la conexión (realizando una consulta al servidor) se denomina Cliente, mientras que el proceso que escucha consultas, las procesa, y (opcionalmente) responde, es el Servidor.
En general, suele haber un host Servidor que ofrece un servicio para resolver consultas de uno o más clientes, a veces incluso de forma simultánea.
Esta arquitectura es usada para los servicios de correo o de impresión y para la World Wide Web.
Esta estructura permite un control centralizado de los accesos y recursos; y facilita la actualización de los datos.

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

El protocolo de capa de aplicación establece las reglas y el formato de los mensajes que hace posible la comunicación entre dos Aplicaciones. En particular define:

- Los tipos de mensajes intercambiados.
- La sintaxis de los tipos de mensajes.
- La semántica de los campos.
- Las reglas que determinan cuándo y cómo cada proceso envía o recibe mensajes.

Ejemplos clásicos de protocolos de aplicación son: HTTP, SMTP y DNS

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

Desarrollamos un protocolo que está por encima de UDP y que implementa Go-Back-N para garantizar que los archivos lleguen completos (cosa que UDP no hace). El detalle está dentro de la sección implementación.

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

Los protocolos de la capa de transporte proveen servicios de multiplexación y demultiplexación. Así, permite ampliar el servicio de entrega de IP entre dos sistemas terminales a un servicio de entrega entre dos procesos que estén ejecutándose en los

sistemas terminales. También proporcionan servicios de comprobación de la integridad de los datos al incluir campos de detección de errores en las cabeceras de sus segmentos.
El protocolo UDP solo provee envío de mensaje sin establecer una conexión, mientras que TCP provee full duplex, esto es, una conexión entre cliente y servidor que permite enviar y recibir datos de forma concurrente, en ambas direcciones.

Características:

- UDP:
    - Provee de los servicios mínimos e indispensables que debe proveer un protocolo de transporte.
    - No está orientado a la conexión.
    - No garantiza ni la correcta llegada (sin errores ni alteración en los bytes) de los mensajes, ni su entrega, ni tampoco que de llegar, la información llegue de forma ordenada.

- TCP:
    - Orientado a la conexión: requiere que se establezca una conexión antes de que se puedan comenzar a enviar los segmentos.
    - Garantiza la llegada de forma ordenada y correcta de los mensajes.
    - Incluye control de congestión, siendo beneficioso para toda la red en general.

Propósitos:
La principal ventaja de UDP es también su principal desventaja: no provee ningún tipo de garantías, por lo que no es confiable. Sin embargo, al no implementar mecanismos que permitan proveer estas garantías, el mismo también es más simple, lo que lo hace más rápido y resulta útil en ciertos casos.
Puede ser utilizado por aplicaciones que no requieran la comprobación inmediata de llegada, como la aplicación de DNS por ejemplo. Es muy útil también en aplicaciones que puedan soportar una cierta tasa de pérdida de paquetes, como son las aplicaciones en tiempo real de vídeo llamadas, videojuegos o llamadas de voz.
No está de más decir que si bien UDP es un protocolo sin garantías, puede implementarse en la capa de aplicación un mecanismo que provea las garantías necesarias, como es el caso del protocolo QUIC de Google o el que desarrollamos para el presente trabajo práctico.

TCP, a diferencia de UDP, permite la comunicación instantánea y confiable entre aplicaciones a costo de un sistema más complejo y mayor uso de tráfico de datos por cada mensaje a enviar.
A su vez, TCP implementa controles de tráfico, por lo que es beneficioso para toda la red.
Se utiliza principalmente en aplicaciones que no puedan soportar pérdida de paquetes y que necesiten de las garantías que este protocolo provee, como podrían ser aplicaciones de home banking, de e-mail, de la web y demás.

Dificultades encontradas

Manejo de los timeouts y cantidad de reenvío de mensajes

Se requirió realizar pruebas para definir la cantidad de reenvío de mensajes cuando ocurre un timeout.
Dicha situación es usual encontrarla cuando se producen pérdidas.

“Conexión y desconexión”

Como no existen conexiones y desconexión en UDP tuvimos que inventar una forma de iniciar y finalizar una operación.
Si incluímos la pérdida de paquetes, la reconexión es simplemente volver a realizar el pedido.

Conclusión

En este trabajo se detalló el funcionamiento de una aplicación de red que provee el servicio de transferencia de archivos de forma confiable utilizando UDP. Para cumplir con los requisitos de entrega confiable se implementaron dos protocolos distintos, Stop-and-wait y Go-Back-N.

El desarrollo de la aplicación nos dió experiencia en:

- las dificultades que tiene implementar un protocolo que considere la pérdida de paquetes
- programación con sockets en Python
- uso de wireshark para el análisis de flujos
- uso de comcast para la simulación de pérdida de paquetes
- uso de concurrencia en escenarios donde el tiempo es un factor clave

También destacamos la importancia de la existencia del protocolo TCP, si solo existiera UDP todo lo hecho en este trabajo práctico debería ser replicado en cada aplicación que desee funcionar sobre internet.