FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 1 - File Transfer

Introducción a los Sistemas Distribuidos (75.43)

Grupo 1

Andrés Fernández - 102220
Sebastián Capelli - 98316
Francisco Bertolotto - 102671
Agustín López Núñez - 101826
Santoni Mauro - 102654

Introducción

En este trabajo práctico se llevó a cabo la creación de una aplicación de red que consiste en la subida y descarga de archivos por medio de la interfaz de sockets y el protocolo de capa de transporte UDP. El principal desafío fue construir un protocolo confiable basado en UDP, en donde diversos problemas pueden presentarse.

A lo largo de este trabajo se explicarán en detalle estos problemas y como nuestro protocolo lidia con ellos. También se profundizará en las implementaciones realizadas y la performance de cada una. Esto es, los métodos Stop and Wait y Go Back N.

Manual de usuario

La aplicación consiste en un servidor que escuchará solicitudes de clientes. Estas solicitudes pueden ser solicitudes de descarga de archivos guardados en el storage del servidor o de subida de archivos locales de los clientes (ver **figura 1**).

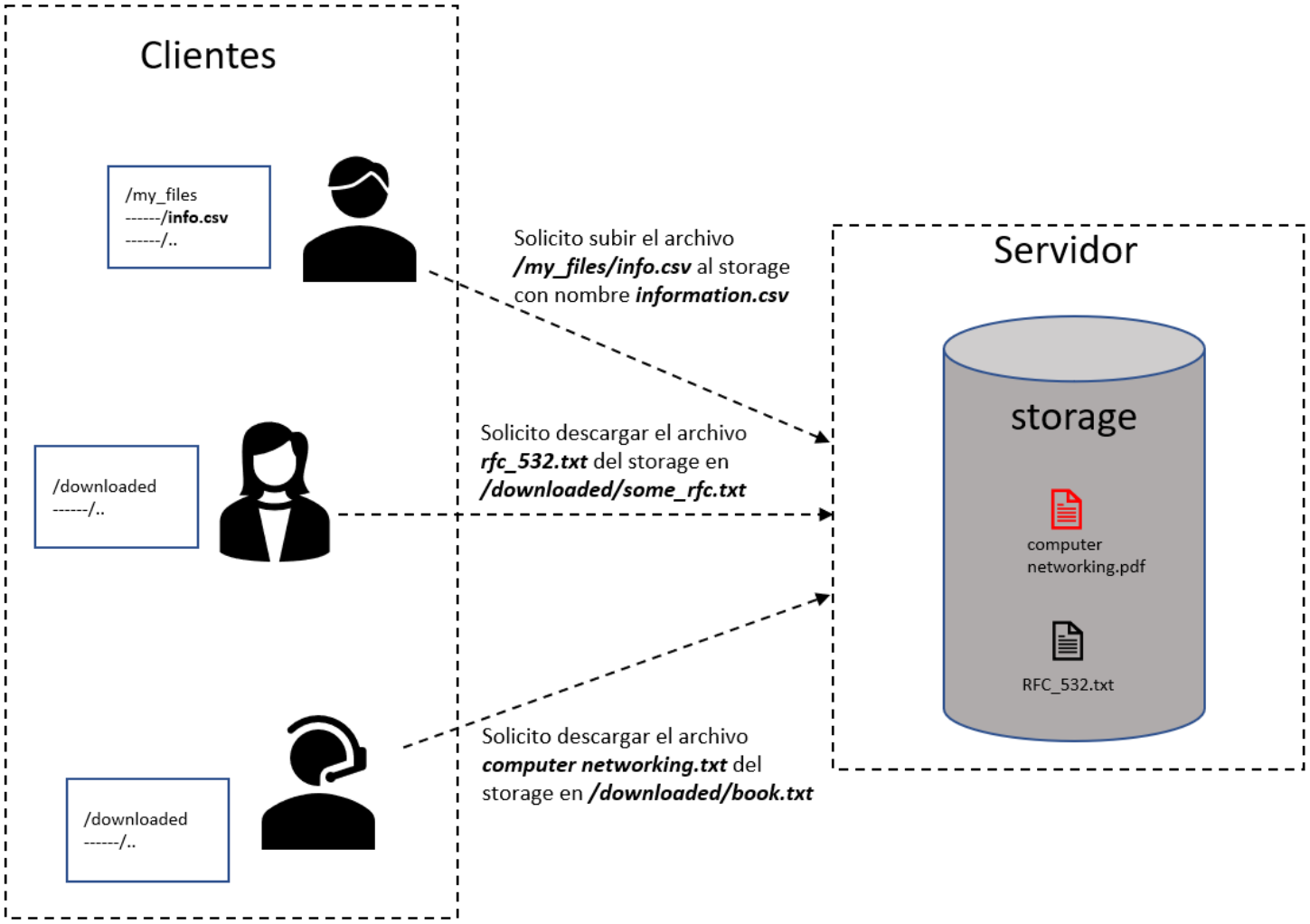


Figura 1. Solicitudes cliente-servidor

Comandos lado servidor

```
> python3 start-server.py -h
usage: start-server [-h] [-v | -q] [-H ADDR] [-p PORT] [-s DIRPATH]

Argumentos opcionales:
    -h, --help          Muestra mensaje de ayuda y se cierra
    -v, --verbose       Ejecución verborrágica
    -q, --quiet         Ejecución no verborrágica
    -H, --host          Dirección IP del servidor
    -p, --port          Puerto del servidor
    -s, --storage       Directorio que será el storage del servidor
                        (ejemplo: server_storage/)
```

Podrá definirse la dirección ip y puerto desde donde se escucharán las nuevas solicitudes de descarga y subida. También puede definirse la carpeta que se utilizará como storage.

Comandos lado cliente

Subida

```
> python3 upload.py -h
usage: upload [-h] [-v | -q] [-H ADDR] [-p PORT] [-s FILEPATH] [-n FILENAME]

Argumentos opcionales:
    -h, --help          Muestra mensaje de ayuda y se cierra
    -v, --verbose       Ejecución verborrágica
    -q, --quiet         Ejecución no verborrágica
    -H, --host          Direccion IP del servidor
    -p, --port          Puerto del servidor
    -s, --src           Archivo a subir
                        (ejemplo: /files/local_file.txt)
    -n, --name          Nombre deseado con el que se guardará el archivo storage del servidor
                        (ejemplo: new_file_in_storage.txt)
```

Podrá definirse la dirección ip y puerto hacia donde se enviarán las solicitudes. También pueden definirse el nombre del archivo a subir y el nombre deseado con el que se guardará en el storage del servidor.

Descarga

```
> python3 download.py -h
usage: download [-h] [-v | -q] [-H ADDR] [-p PORT] [-d FILEPATH] [-n FILENAME]

Argumentos opcionales:
    -h, --help          Muestra mensaje de ayuda y se cierra
    -v, --verbose       Ejecución verborrágica
    -q, --quiet         Ejecución no verborrágica
    -H, --host          Dirección IP del servidor
    -p, --port          Puerto del servidor
    -s, --src           Archivo a descargar
                        (ejemplo: /files/new_local_file.txt)
    -n, --name          Nombre del archivo en el storage del servidor
                        (ejemplo: file_in_storage.txt)
```

Podrá definirse la dirección ip y puerto hacia donde se enviarán las solicitudes. También pueden definirse el archivo que se quiere descargar del storage del servidor y el nombre que tendrá la copia local de ese archivo al descargarse.

Dificultades encontradas

- Manejo de chunks y de paquetes: en primer lugar, tuvimos que definir si los “chunks” en los que se dividiría un archivo para enviarse tendrían el mismo tamaño que los paquetes que se enviarían los sockets. Cómo determinamos que esto no tenía por qué ser así, tuvimos que contemplar, en la implementación de Go Back N, que habría una especie de doble ciclo (división del archivo en chunks y división del chunk en paquetes) a la hora de llevar los contadores de secuencia. Solucionamos esto reiniciando los contadores cada vez que se inicia el envío / recepción de un nuevo chunk.
- La validación de paquetes fuera del envío normal de información fue difícil de implementar, porque tuvimos que definir cómo manejar algunos casos borde. Por ejemplo, qué hacer con la última confirmación de recepción de un paquete de data. Si el receptor enviase un único ack, y ese ack se perdiera, el emisor nunca se enteraría de ese ack y pensaría que el receptor no recibió la información. La solución fue multiplicar el envío del último ack para reducir la probabilidad de que esa última confirmación no le llegara al emisor.
- El handshake se fue modificando a lo largo del desarrollo del trabajo. Es otro de los casos donde debíamos contemplar pérdidas de paquetes por fuera del ciclo de envío y recepción del archivo. La solución que encontramos será detallada más adelante.
- La necesidad de agregar el número de chunk. Uno de los problemas con los cuales nos encontrábamos era que, al simular la duplicación y demora (sobre todo esto último) en el envío de paquetes, nos encontrábamos con que un paquete perteneciente a un chunk de datos anterior modificaba incorrectamente el desarrollo del envío del chunk actual. Sin contar con el número de chunk, eso resultaba prácticamente imposible, pero una vez agregado, nos ayudó a resolver muchos de estos problemas.

- Implementación de Stop-And-Wait y Go-Back-N. Inicialmente, habíamos planteado dos formas muy diferentes de llevar a cabo ambas implementaciones, incluso con los headers distintos para cada una de ellas. Finalmente, llegamos a la conclusión de que Stop-And-Wait no es más que un caso particular de Go-Back-N con una ventana deslizante (RWND) de tamaño 1.

Hipótesis y supuestos

- El tamaño máximo de archivos que se pueden almacenar es de **1GB**.
- Los clientes solamente pueden descargar archivos del storage. Y, como los archivos del storage serán únicamente aquellos que fueron subidos por algún cliente, no se descargan archivos mayores a **1GB**.
- (Recomendación) Poner los nombres de archivos entre comillas, para evitar errores de parseo de nombres.
- Los nombres de los archivos no pueden contener '\0'.
- UDP descarta todos paquetes cuyo checksum no sea válido, por lo que el protocolo no realiza una segunda verificación sobre ese checksum.
- El timeout de los sockets utilizado en las funciones de envío de información es fijo, y es de 10 milisegundos. El timeout utilizado en las funciones de recepción se calcula en base a este timeout fijo.
- Si el cliente solicita subir un archivo con un nombre no disponible, se realizará el proceso de subida normalmente pero se le asignará otro nombre en el storage del servidor.
- Si el cliente descarga un archivo sin especificar el file path de destino se descargará con el nombre que se encuentra en el servidor.
- En una descarga, si el cliente especifica un file path de un archivo local ya existente, este se sobreescribirá por el nuevo archivo descargado.
- Al momento de descargar o subir un archivo las carpetas de destino y/o fuente deben existir (es decir, no se crearán carpetas).

Protocolo

Descripción de los métodos utilizados

Tanto el cliente como el servidor deberán ejecutar una serie de métodos para llevar a cabo la comunicación, internamente estos métodos utilizan distintos mecanismos para asegurar la comunicación confiable, como se detalla en la sección *¿por qué nuestro protocolo es confiable?*. Las figura 2 y figura 4 muestran la secuencia de mensajes intercambiados.

Métodos utilizados en lado server

Socket() crea un nuevo socket, se le debe pasar una direccion ip y un número de puerto

receive_first_conection() esperá nuevas solicitudes y al recibirlas retorna datos del cliente y de la operación solicitada.
Si se recibe un mensaje que no sea una solicitud se lanza un Exception.

send_sv_information() responde al cliente con la información necesaria para definir un flujo de comunicación.

Métodos utilizados en lado cliente

send_up_request() envía una nueva solicitud de subida al servidor

send_dl_request() envía una nueva solicitud de descarga al servidor

receive_sv_information() recibe la respuesta de la solicitud

Métodos utilizados en ambos lados

receive_data() y **send_data()** reciben y envían los chunks de datos respectivamente.

close() libera los recursos asociados al Socket. Se trata de una operación local que no intercambia mensajes con el otro extremo.

Proceso de subida

Métodos utilizados

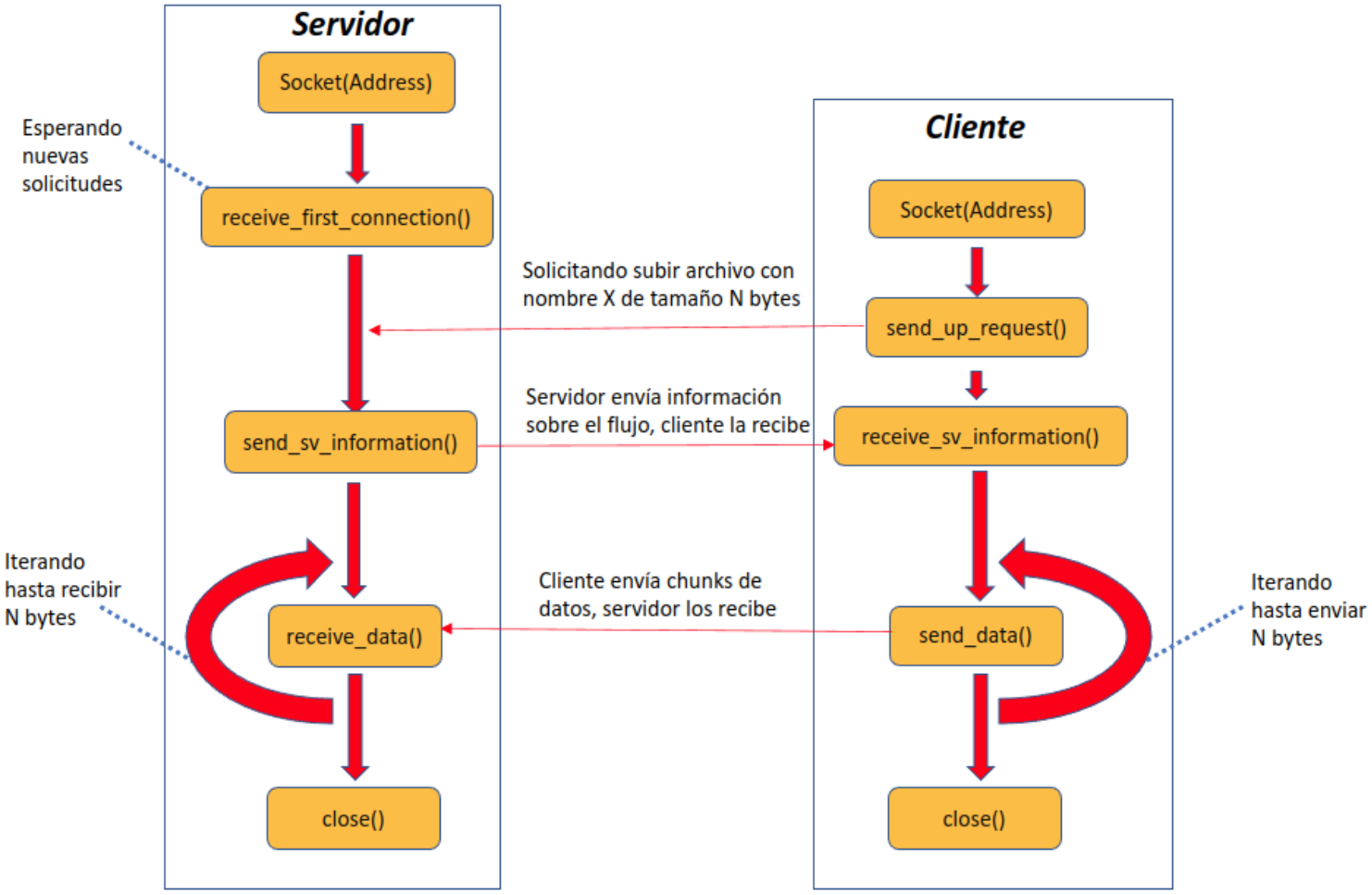


Figura 2. Mensajes intercambiados en proceso de subida

Diagrama de secuencia

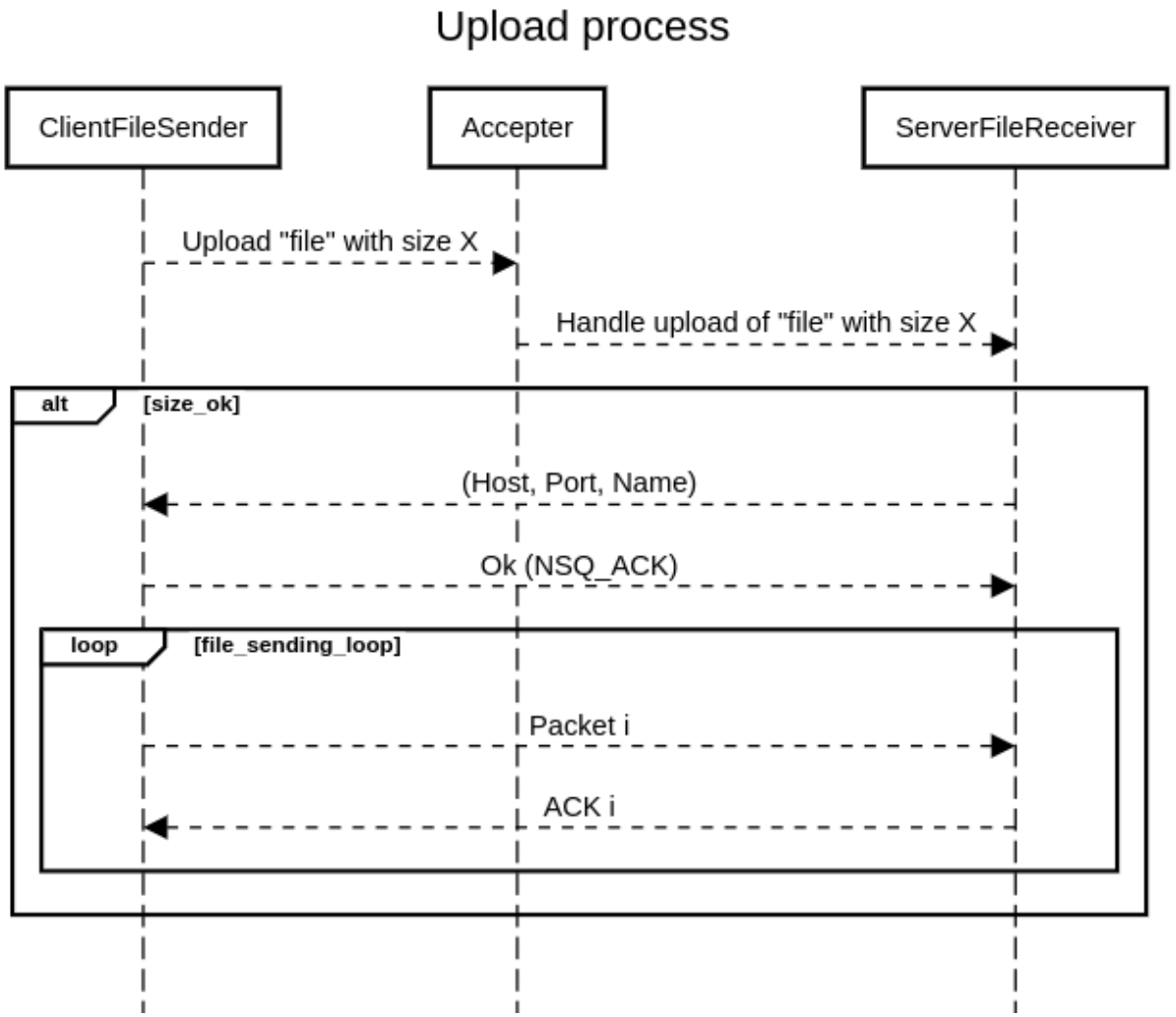


Figura 3.Diagrama de secuencia. En proceso de subida.

Proceso de descarga

Métodos utilizados

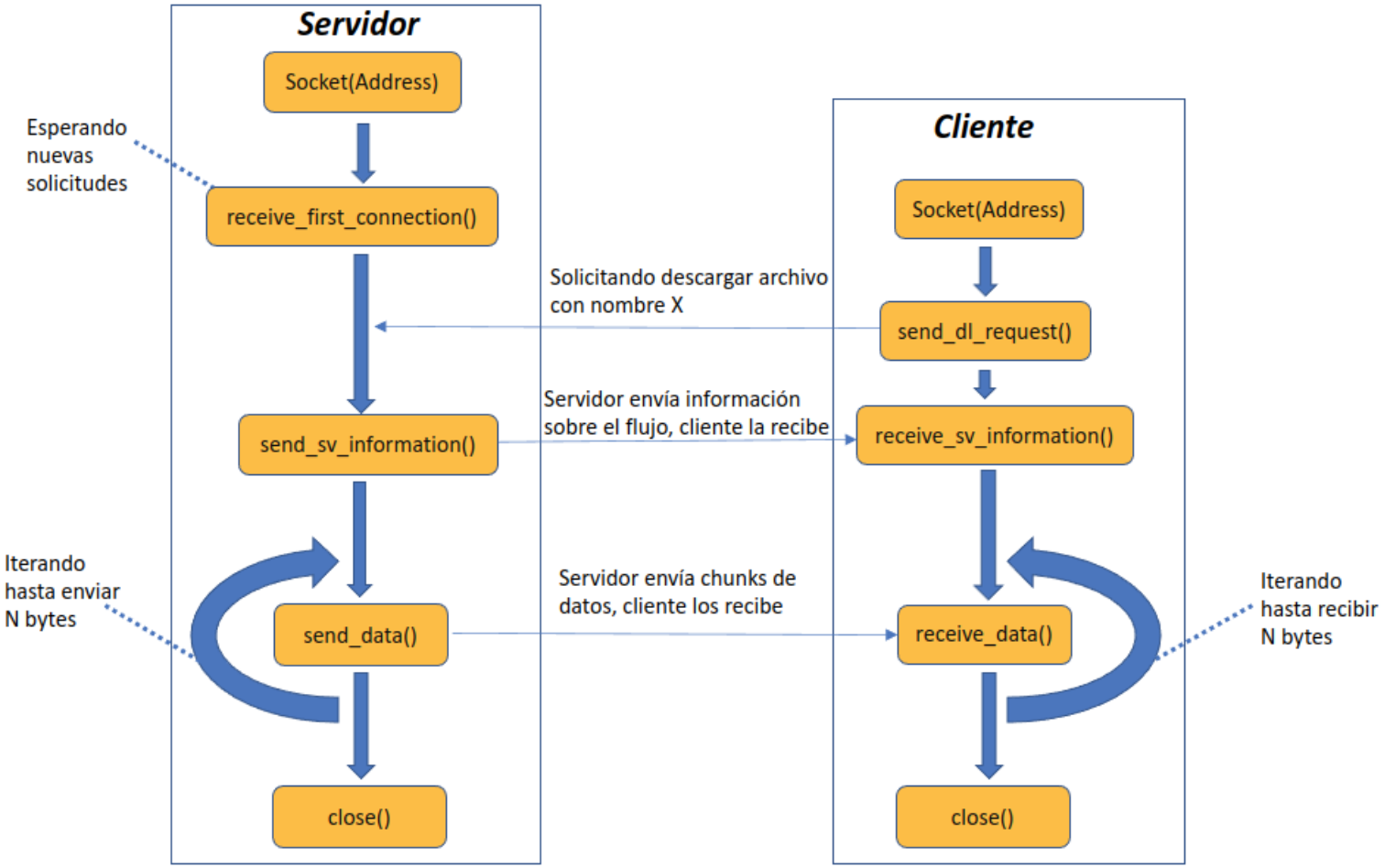


Figura 4. Mensajes intercambiados en proceso de descarga

Diagrama de secuencia

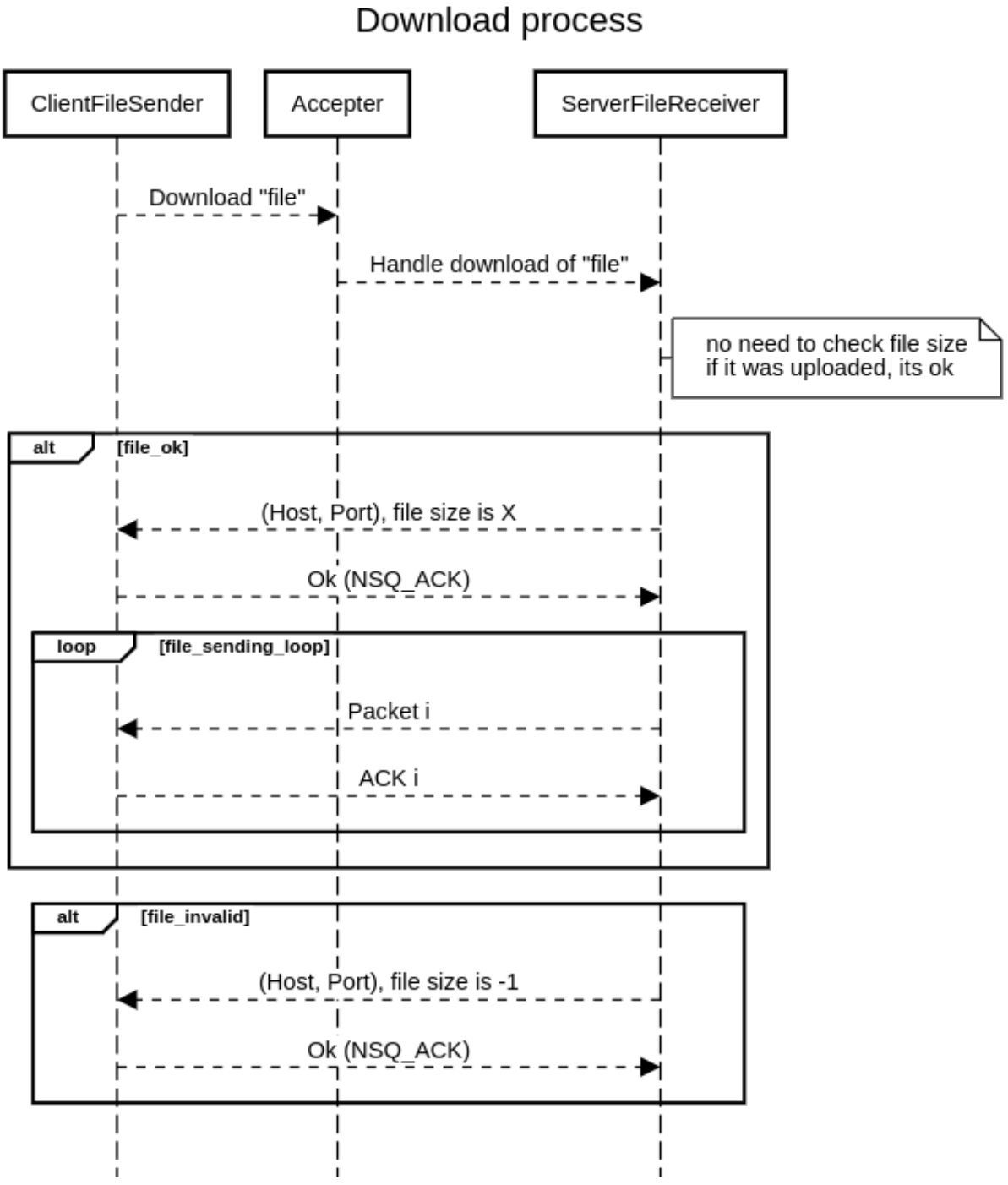


Figura 5. Diagrama de secuencia en proceso de descarga.

Mensajes

Formato

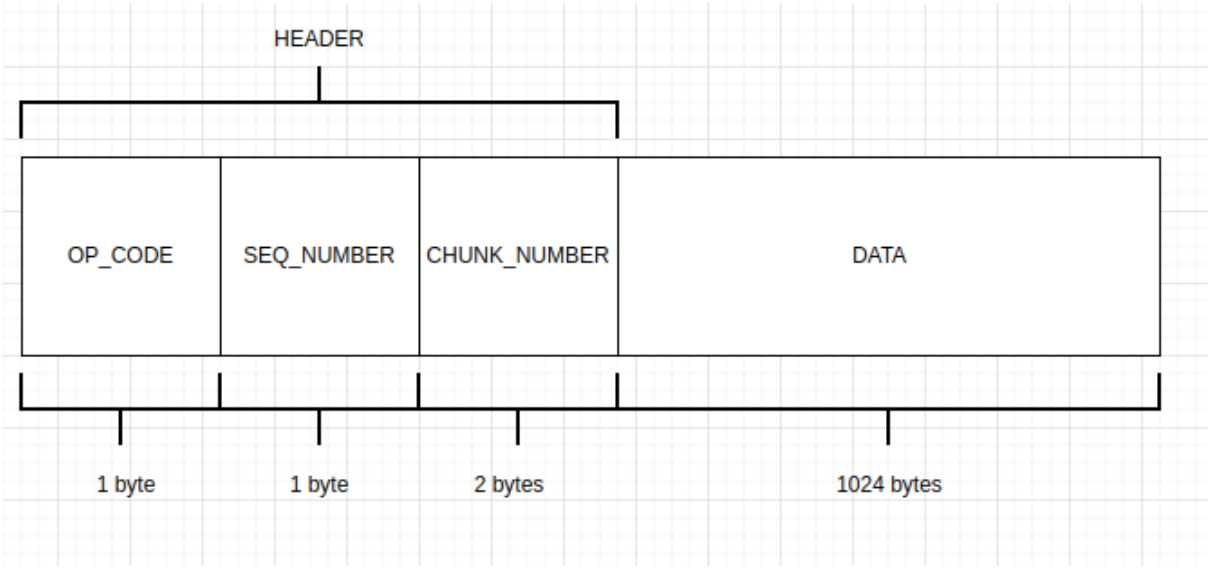


Figura 6. Formato genérico del paquete.

Header

En ambas implementaciones, GBN y SAW, el header del protocolo tiene sólo 4 bytes:

- El primer byte corresponde al op_code, que define el tipo de mensaje (ver abajo).
- El segundo byte corresponde al número de secuencia, utilizado principalmente por el receptor para llevar el registro del último paquete recibido correctamente (de forma consecutiva),
- El tercer y cuarto byte contienen el número de “chunk”, o porción, de la información que se está enviando en ese momento. Se puede ver la diferencia entre chunk number y sequence number en esta figura:

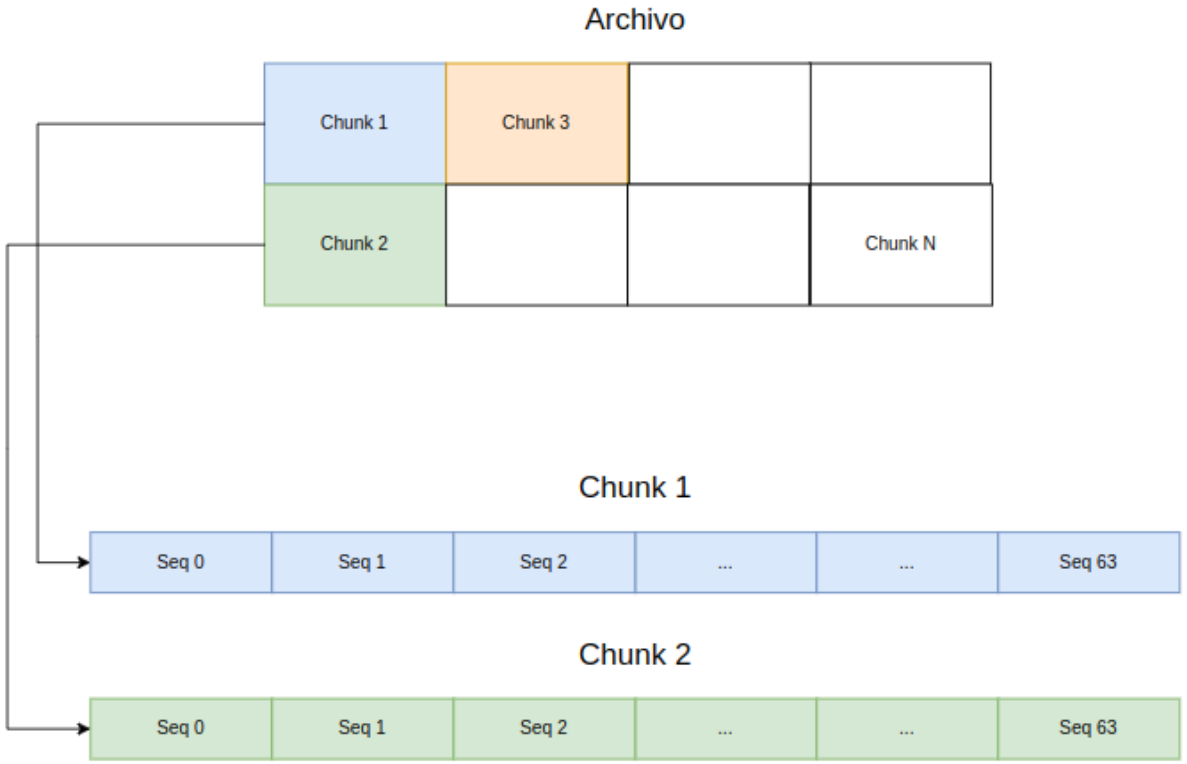


Figura 7. Chunks de datos y paquetes con sus números de secuencia.

Longitud de los mensajes

En ambas implementaciones, la longitud de los mensajes es de 1028 bytes:

- 4 bytes para el header:
  - 1 byte de Operation Code.
  - 1 byte de Sequence Number.
  - 2 bytes de Chunk Number.
- 1024 bytes de data. En general, el contenido del archivo a enviar, pero en algunos mensajes del handshake (ver abajo) tiene otra información.

Handshake

El handshake, tanto en la descarga como en la subida de archivos, consiste de 3 mensajes, de acuerdo a lo que se muestra a continuación:

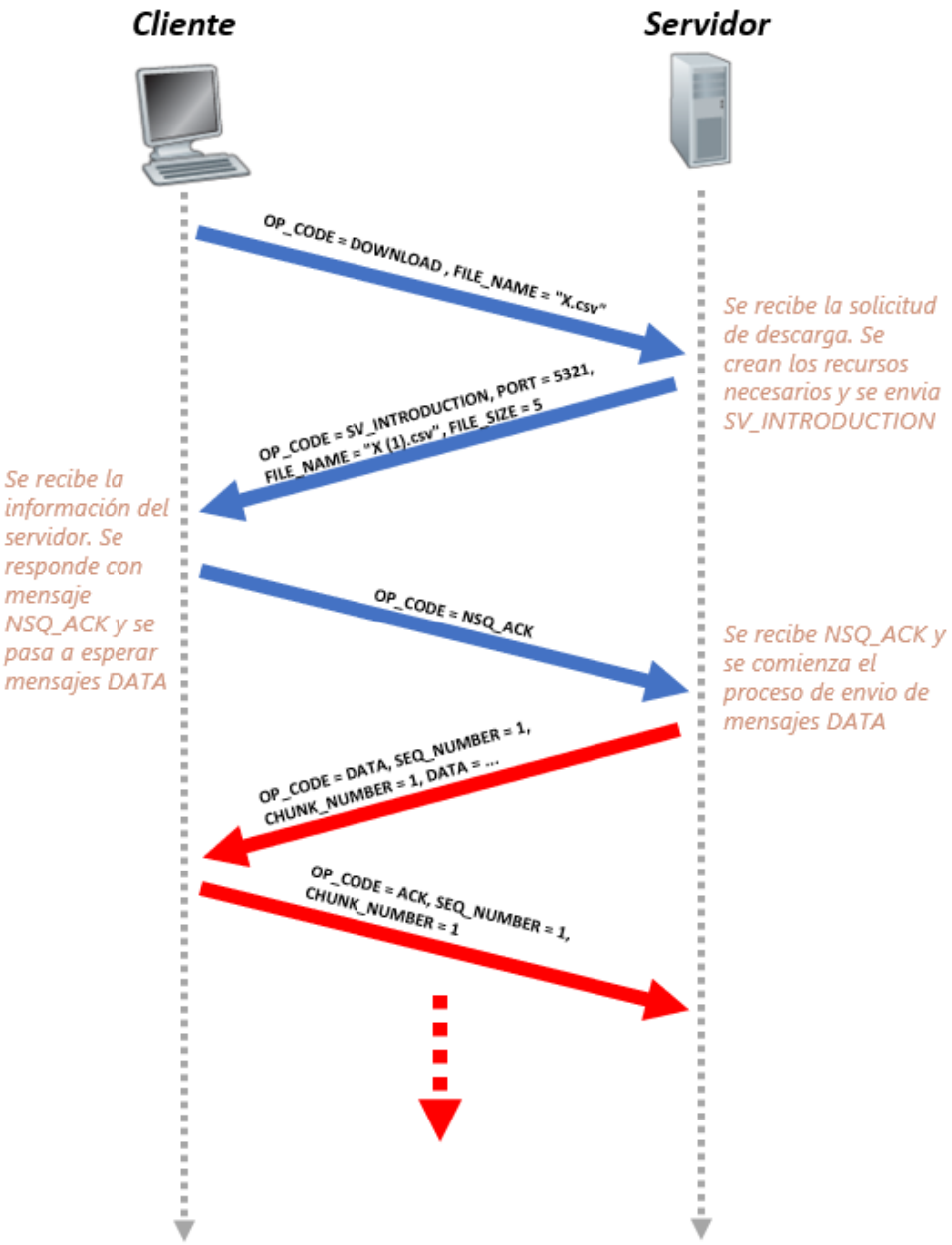


Figura 8. Descarga. Mensajes intercambiados en el handshake (en azul).

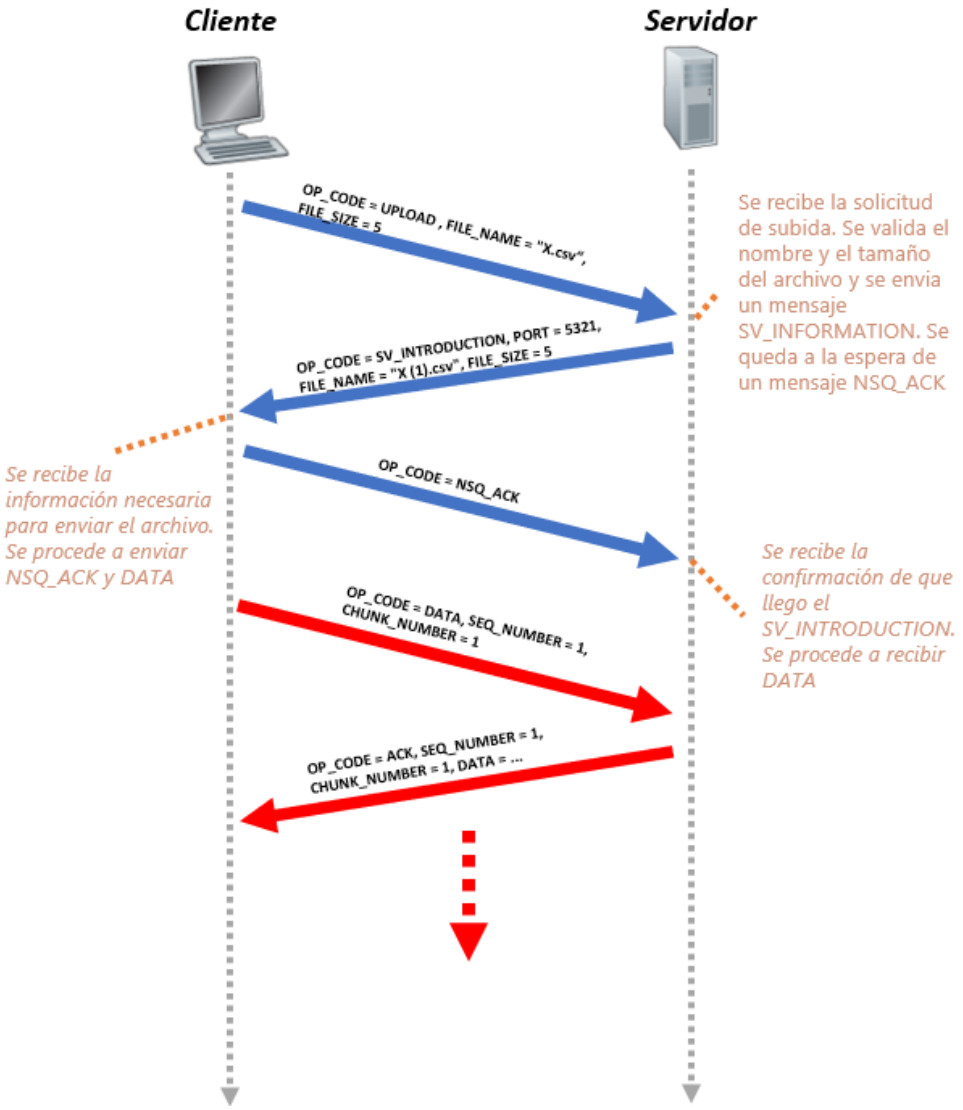
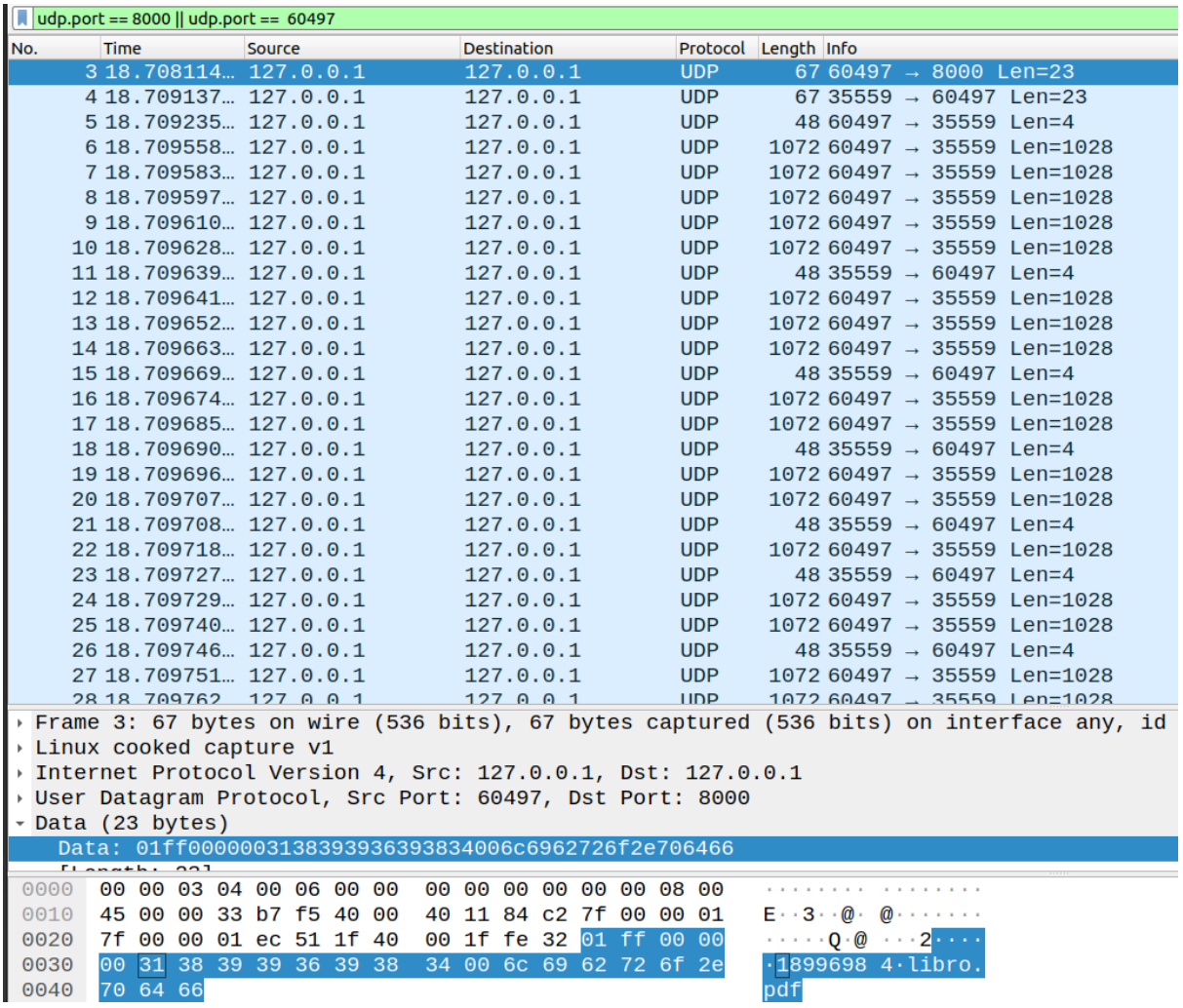


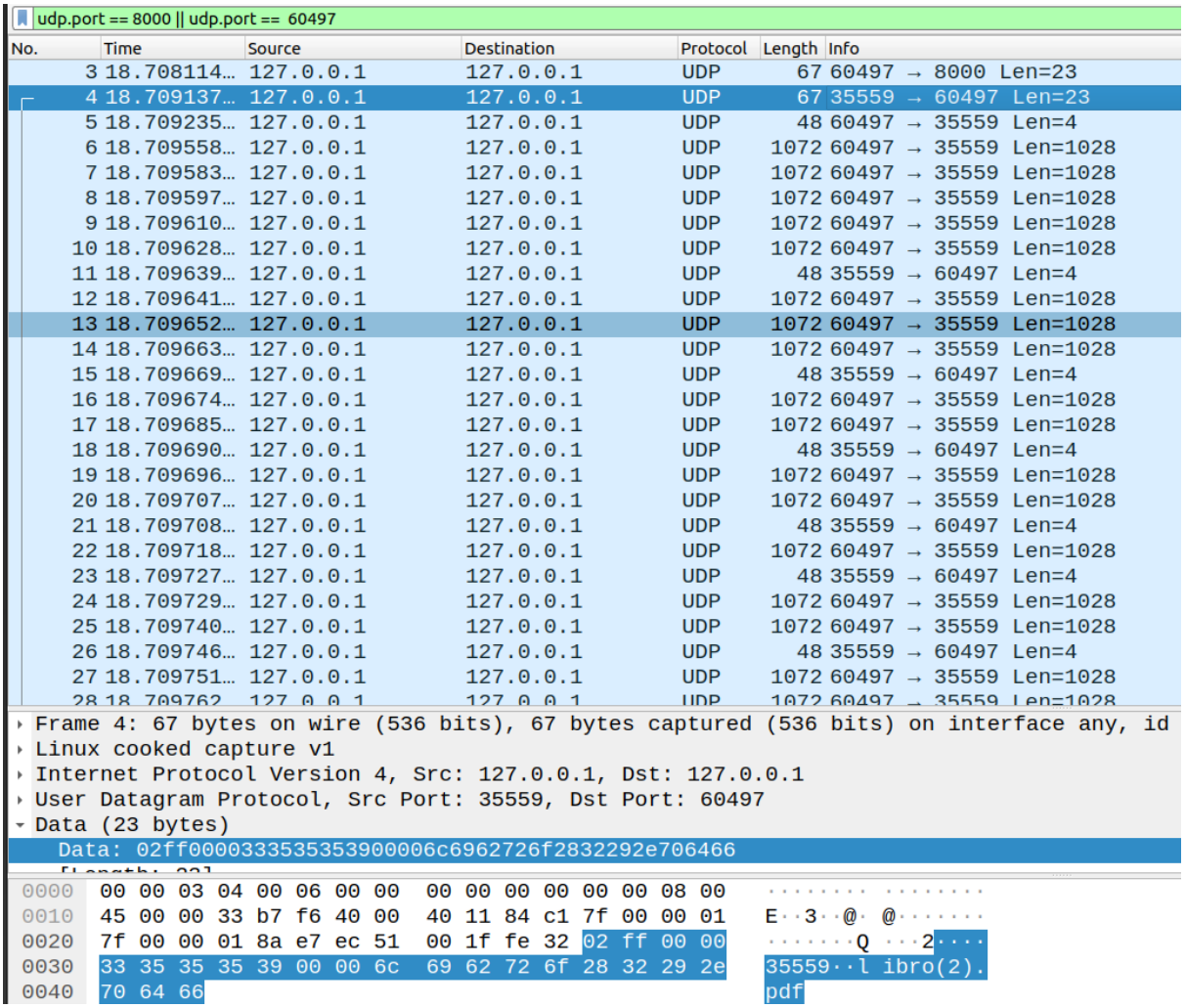
Figura 9. Subida. Mensajes intercambiados en el handshake (en azul).

Wireshark

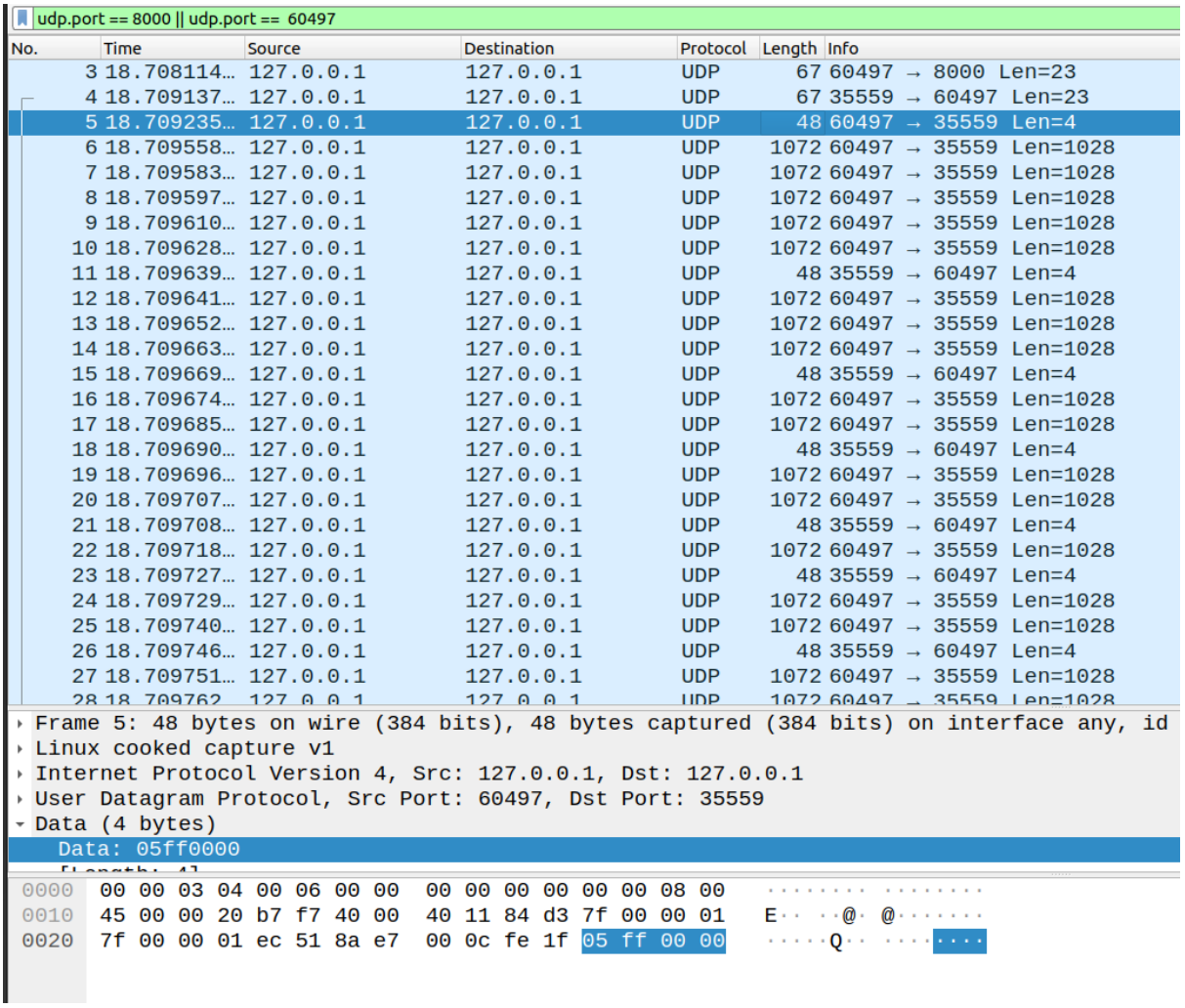
En esta primera imagen podemos observar el primer mensaje que le envía el cliente al servidor al momento de realizar una subida. Si vemos el primer byte (01) vemos que coincide con el código de operación de *UPLOAD (1)*, seguido a eso vemos el tamaño del archivo y el nombre con el cual lo quiere subir el cliente.



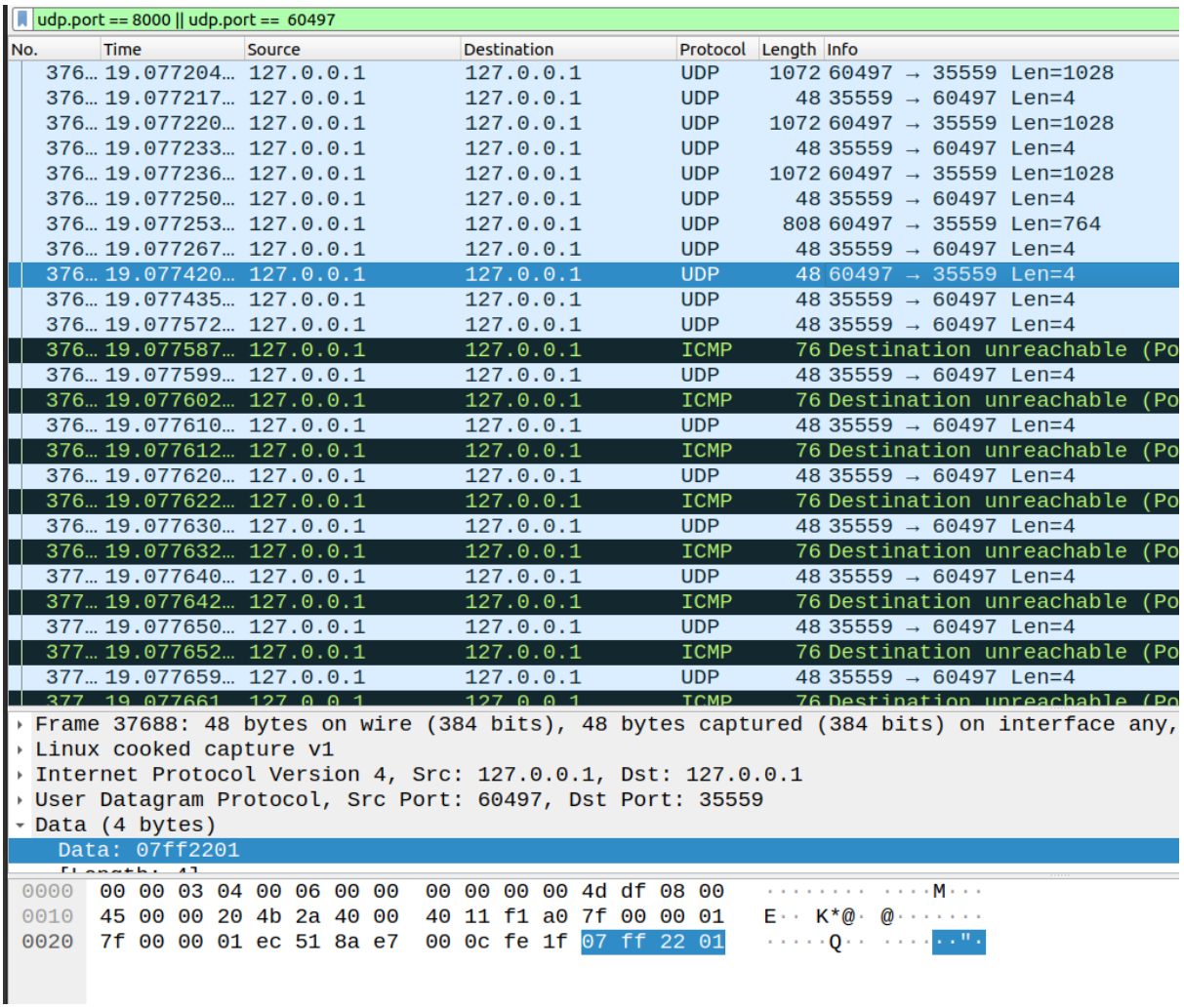
En esta segunda imagen se observa la respuesta del servidor ya desde el puerto nuevo con el thread correspondiente. Se envía el código *SV_INTRODUCTION (2)* junto con la información del handshake que va a contener el puerto con el que el cliente debe comunicarse junto con el nombre del archivo que se almacenará en el servidor.



Por último, vemos el *NSQ_ACK (05)* del cliente al servidor del handshake que confirma la recepción de la información del servidor.

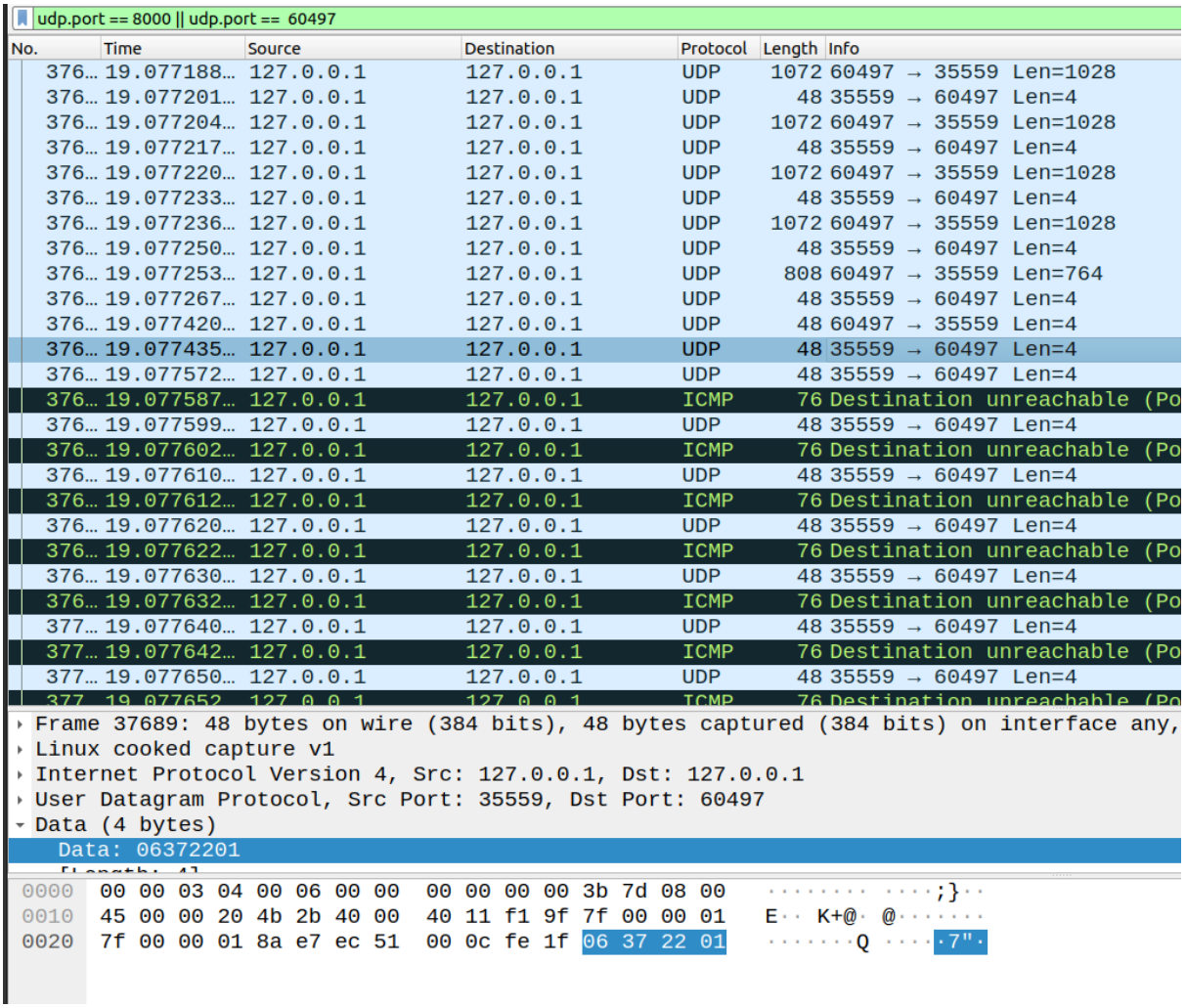


Al finalizar la transferencia, el cliente envía el mensaje *END (7)* para informar que terminó de enviar.



Como mensaje final, el servidor le envía un *END_ACK (06)* para informarle la correcta recepción del END.

El resto de mensajes ICMP que se observan son parte de la rafaga final de END_ACK que envia el servidor para que este último tenga las mismas posibilidades que los que no son final



¿Por qué nuestro protocolo es confiable?

Al estar construido sobre UDP y IPv4, nuestro protocolo hereda la mayoría de características de comunicación best effort de IP. Estas pueden presentar los siguientes problemas:

- Duplicación
- Pérdida
- Corrupción
- Entregas desordenadas

UDP incluye verificación de paquetes por medio de checksums. Por lo que asumimos que no tendremos el problema de corrupción.

Los problemas de duplicación y entregas desordenadas se contemplan con el uso de números de secuencia y números de chunks de nuestro protocolo de aplicación. Los casos de pérdidas también se contemplan con timers y números de secuencia/chunk.

¿Cómo se contemplan las pérdidas de paquetes?

Por medio de diversos mecanismos: Timers, números de secuencia y ACKs.

Casos Descarga

Las figuras 10, 11 y 12 muestran el comportamiento ante la pérdida de mensajes del handshake. En estos casos, los timers permiten establecer un tiempo límite de espera, a partir del cual se realizarán reenvíos hasta obtener la respuesta esperada. El efecto de cada mensaje se explica en detalle en la respuesta a la pregunta 3 de la sección *Preguntas*. La figura 12 muestra el caso de pérdida del mensaje NSQ_ACK, la cual provoca un desfasaje que eventualmente se corrige. En ese caso resulta particularmente importante validar que los códigos de mensajes sean los esperados.

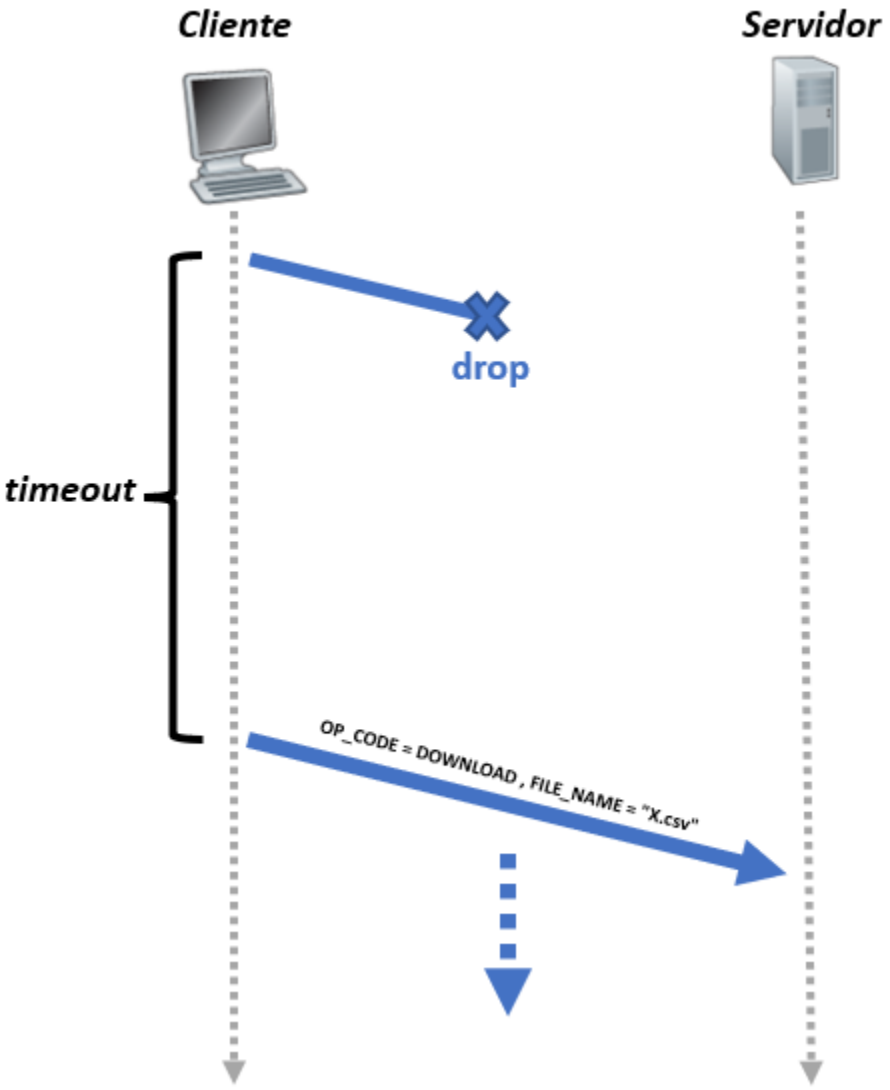


Figura 10. Pérdida de mensaje DOWNLOAD.

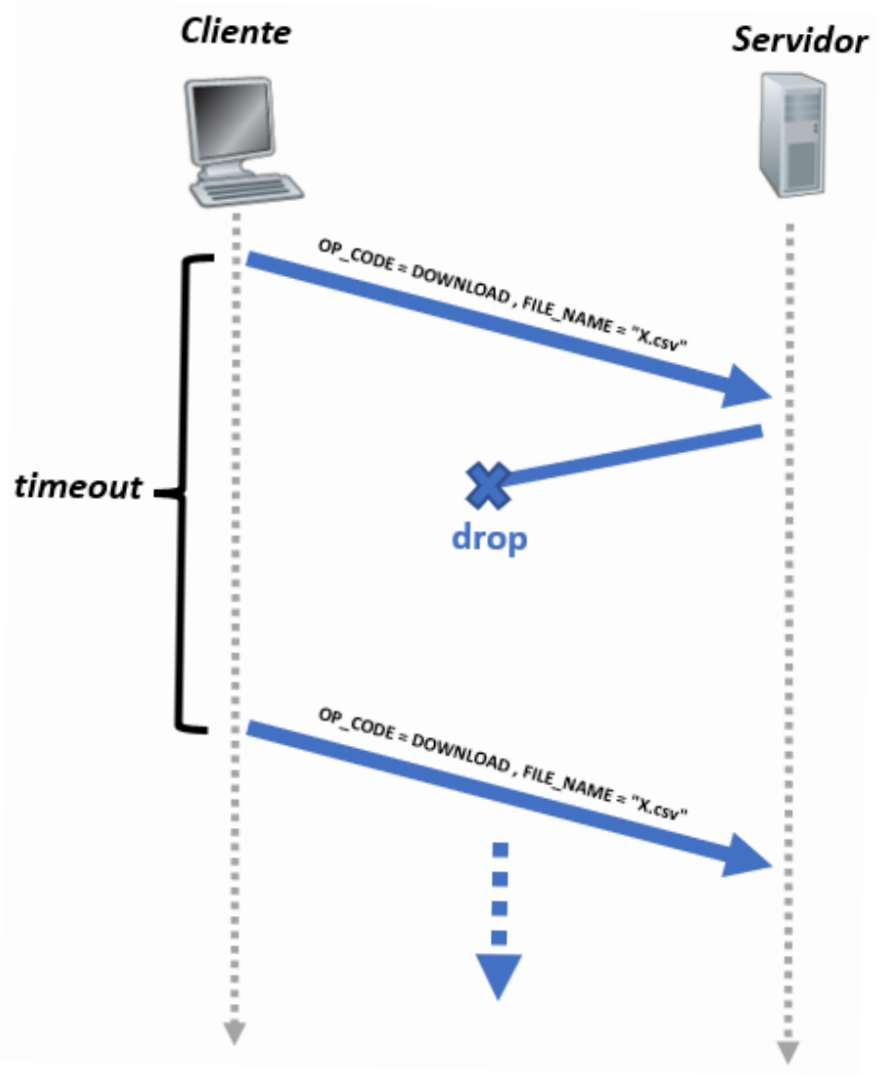


Figura 11. Pérdida de mensaje SV_INTRODUCTION.

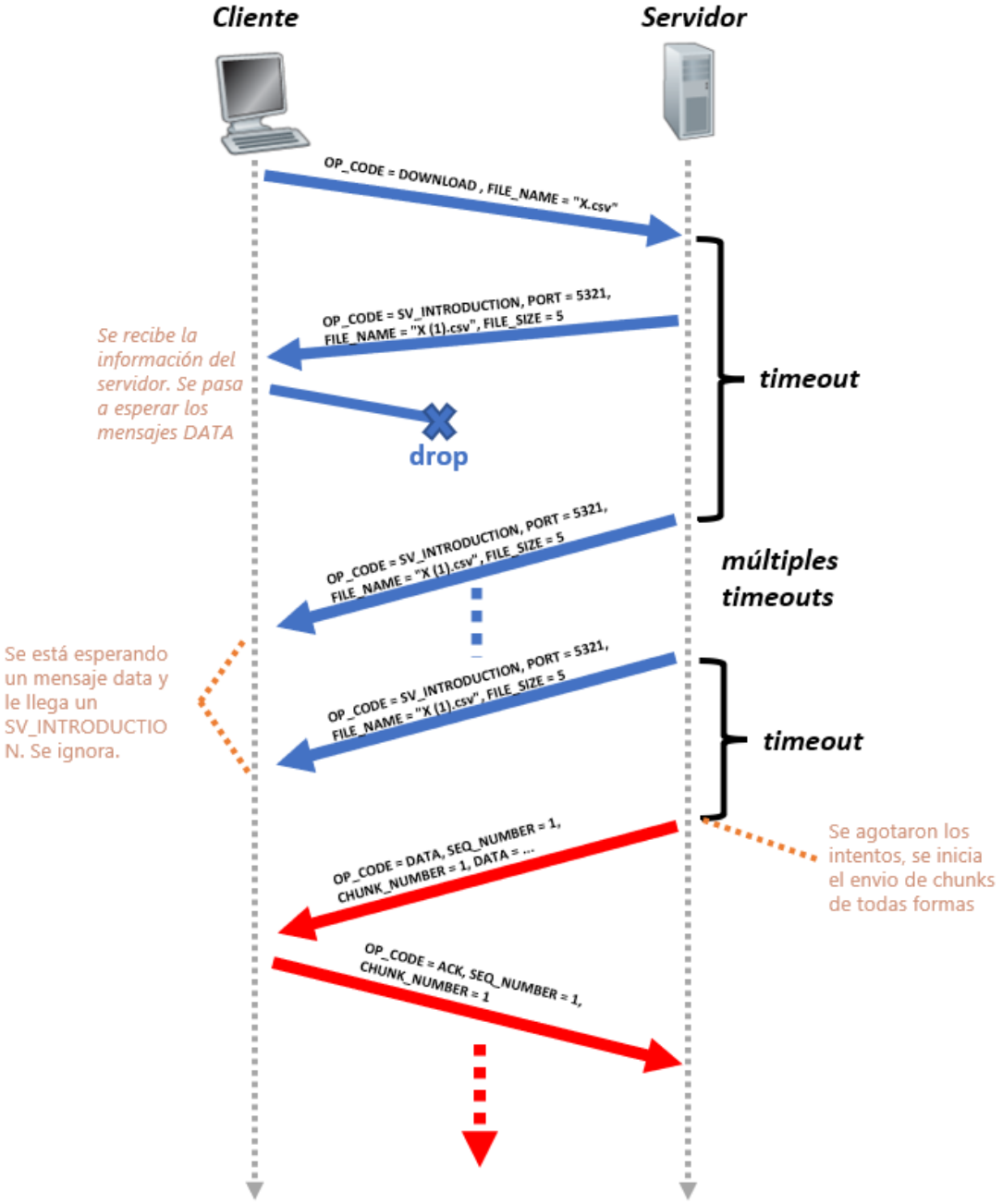


Figura 12. Perdida de mensaje NSQ_ACK.

Casos Subida

De forma análoga a la descarga, la pérdida de mensajes en el proceso de subida estará controlada principalmente por los timers. Las figuras 13, 14 y 15 muestran el comportamiento del sistema ante estas pérdidas. El caso más conflictivo resulta el de la pérdida del mensaje NSQ_ACK visto en la figura 15, en donde se lleva a cabo un intercambio de mensajes inválidos que eventualmente se corrige y comienza el proceso de subida.

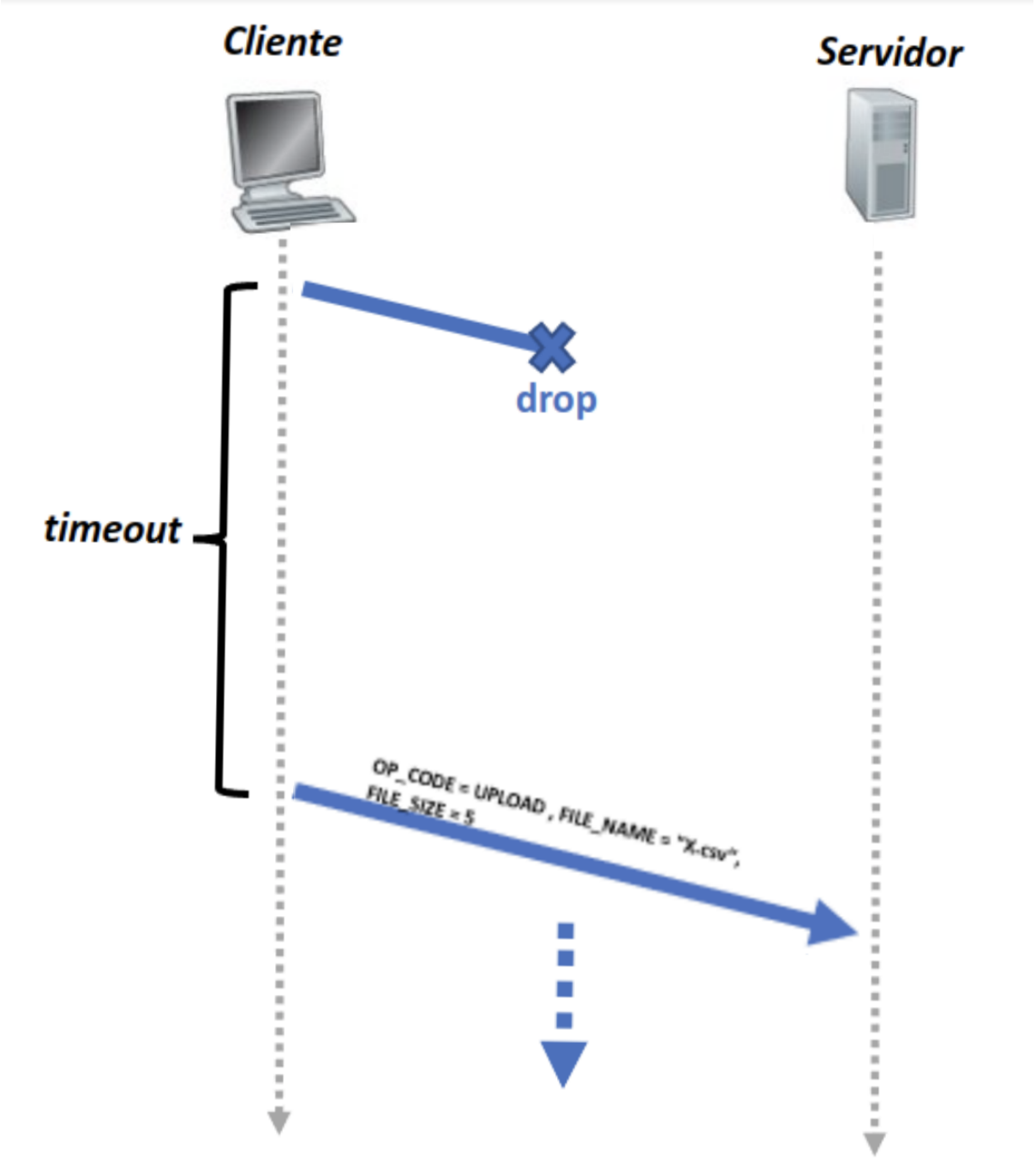


Figura 13. Pérdida de mensaje UPLOAD

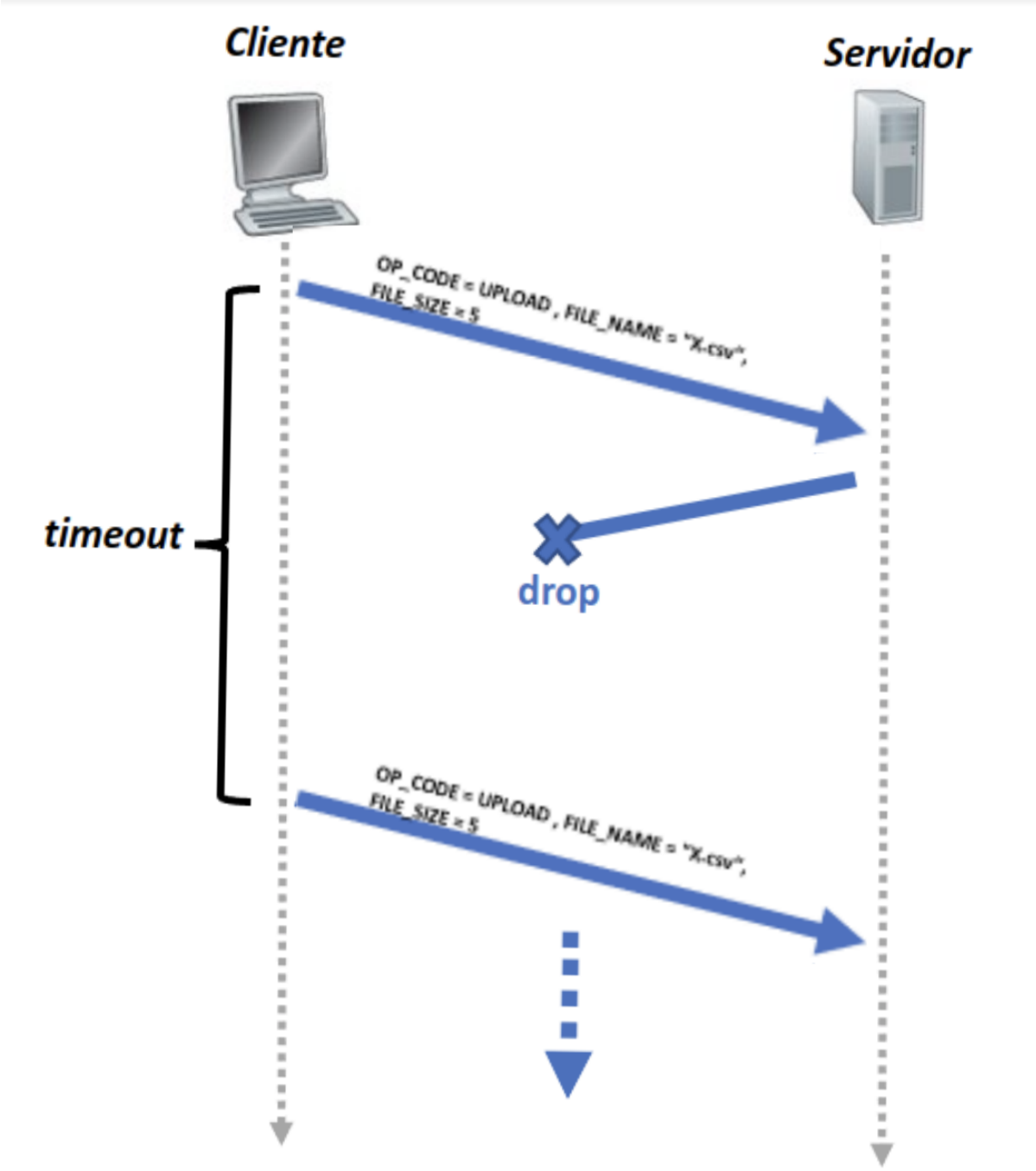


Figura 14. Pérdida de mensaje SV_INTRODUCTION

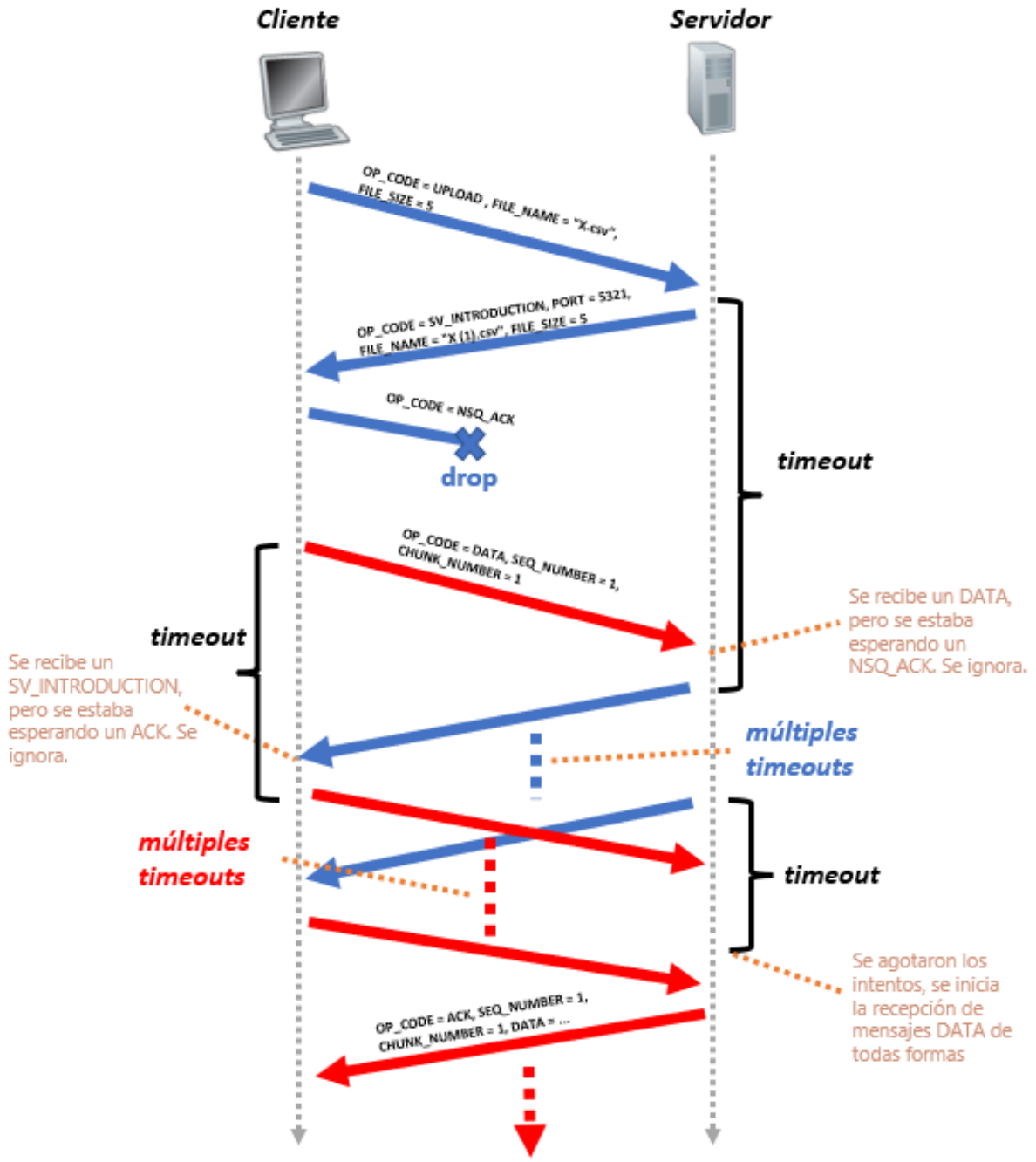


Figura 15. Pérdida de mensaje NSQ_ACK.

Casos subida y descarga durante envío de Mensajes DATA o ACK

Las figura 16 y 17 muestran el comportamiento del sistema ante pérdida de mensajes DATA o ACK. En ambos casos los timers permitirán detectar pérdidas y darán lugar a reenvíos de mensajes DATA.

Aquí también resultan importantes los números de secuencia, pues podría darse la situación de que un mensaje llegue duplicado como en la figura 16, en donde la pérdida de un mensaje ACK provoca el eventual reenvío del mensaje DATA con número de secuencia 4. Será responsabilidad del lado que recibe descartar ese mensaje.

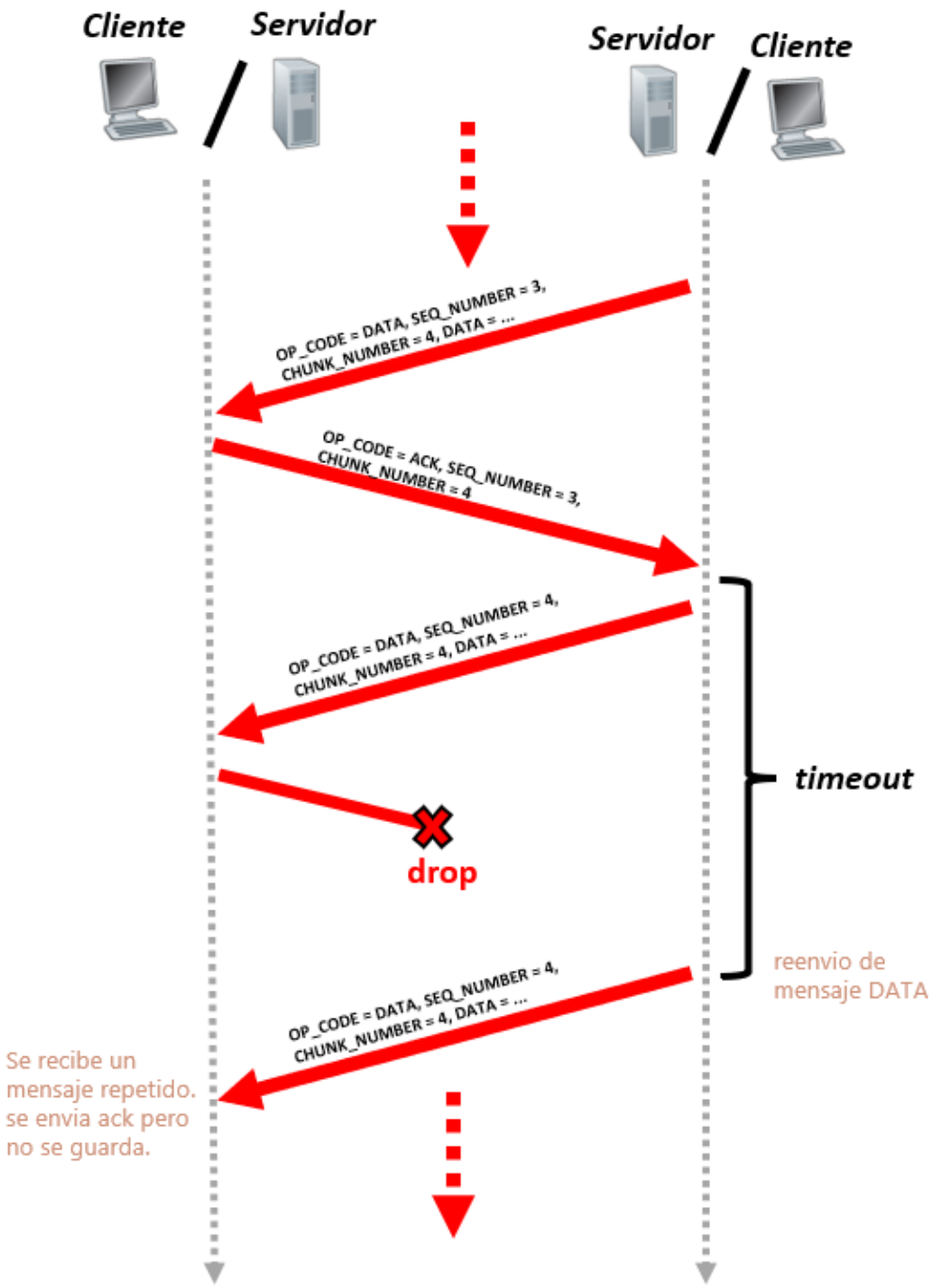


Figura 16. Pérdida de mensaje ACK.

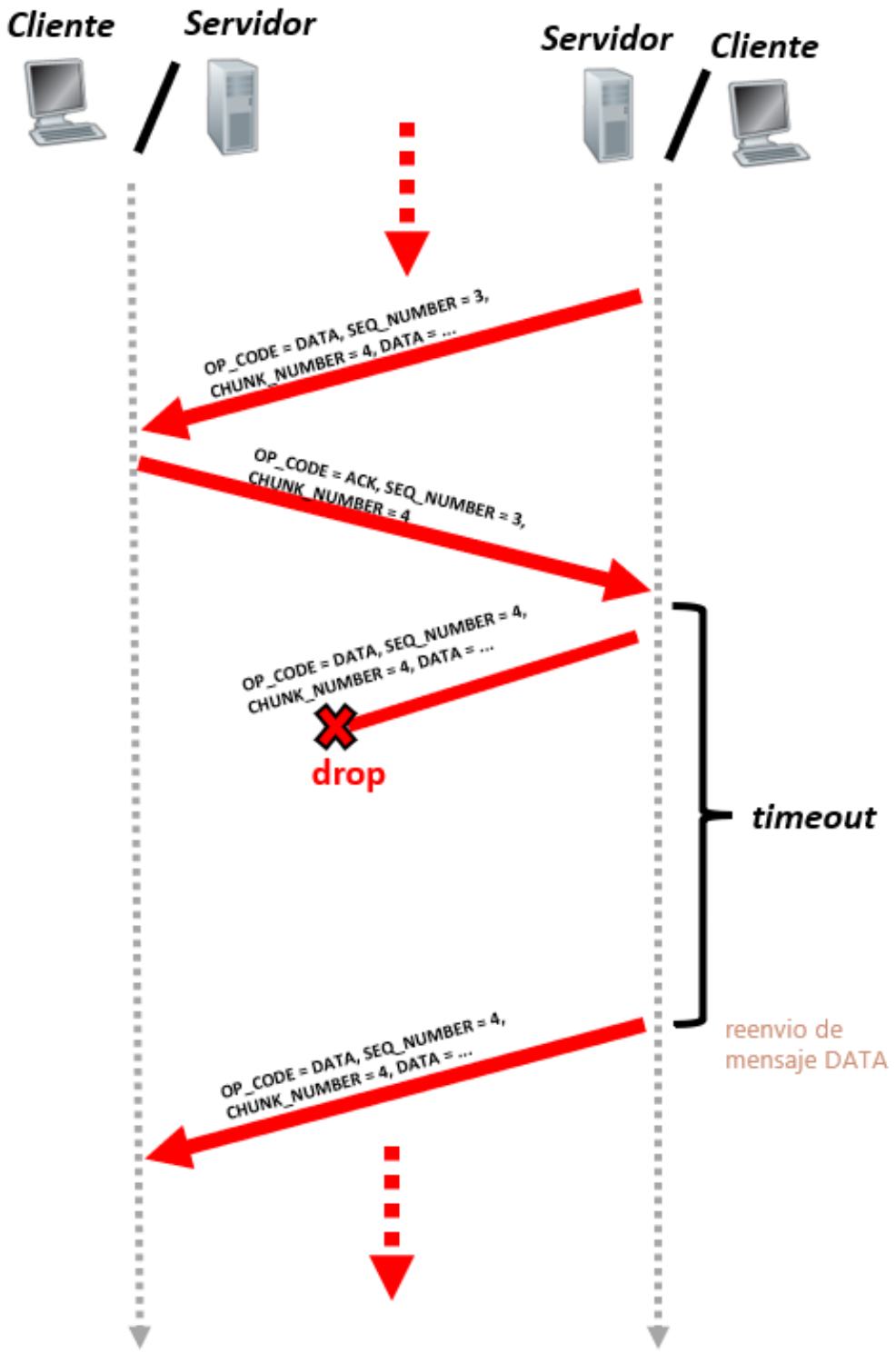


Figura 17. Pérdida de mensaje DATA.

Pruebas simulando pérdida de paquetes.

A continuación se detallan las distintas pruebas realizadas en la transferencia de archivos, con distintos porcentajes de pérdida de paquetes. Esto es con el objetivo de medir distintos tiempos de transferencia en ocasiones de distinto contexto y la respuesta del sistema ante ellos. Todos los tiempos indicados son en segundos.

Subida (GBN)

Foto 1mb (1,07 MB (1.125.918 bytes))				
N° corrida	**Sin pérdidas (en segundos)**	**1% pérdidas (en segundos)**	**5% pérdidas (en segundos)**	**10% pérdidas (en segundos)**
1	0.02	0.31	1.12	2.90
2	0.02	0.16	1.28	2.96
3	0.02	0.23	1.31	3.05
4	0.02	0.18	1.43	2.89
5	0.02	0.23	1.47	2.88
6	0.02	0.27	1.21	2.67
7	0.02	0.23	1.24	2.83
8	0.02	0.31	1.31	2.91
9	0.02	0.30	1.45	2.69
10	0.02	0.24	1.43	3.11
Promedio:	**0.02**	**0.25**	**1.32**	**2.89**

Foto 5mb (5,39 MB (5.658.596 bytes))				
N° corrida	**Sin pérdidas (en segundos)**	**1% pérdidas (en segundos)**	**5% pérdidas (en segundos)**	**10% pérdidas (en segundos)**
1	0.08	1.26	6.67	14.33
2	0.08	1.33	6.87	14.23
3	0.09	1.24	6.73	14.16
4	0.09	1.23	6.88	14.84
5	0.09	1.50	6.38	14.93
6	0.08	1.49	6.56	14.03

7	0.09	1.26	6.48	14.72
8	0.08	1.30	6.58	13.61
9	0.08	1.25	6.94	14.35
10	0.08	1.37	6.49	15.04
Promedio:	**0.08**	**1.32**	**6.66**	**14.42**

Libro.pdf (18,1 MB (18.996.984 bytes))				
N° corrida	**Sin pérdidas (en segundos)**	**1% pérdidas (en segundos)**	**5% pérdidas (en segundos)**	**10% pérdidas (en segundos)**
1	0.27	4.30	21.56	47.73
2	0.29	4.48	23.10	47.54
3	0.27	4.65	21.95	47.08
4	0.28	4.53	21.67	48.22
5	0.27	4.58	22.84	48.70
6	0.25	4.51	22.28	48.38
7	0.25	4.41	23.34	48.38
8	0.28	4.80	22.20	48.96
9	0.26	4.67	22.33	46.99
10	0.26	4.49	22.76	49.76
Promedio:	**0.27**	**4.54**	**22.40**	**48.17**

Descarga (GBN)

Foto 1mb (1,07 MB (1.125.918 bytes))				
N° corrida	**Sin pérdidas (en segundos)**	**1% pérdidas (en segundos)**	**5% pérdidas (en segundos)**	**10% pérdidas (en segundos)**
1	0.02	0.38	1.51	2.84
2	0.02	0.32	1.55	2.86
3	0.02	0.21	1.32	2.91
4	0.02	0.30	1.44	3.05
5	0.02	0.24	1.34	2.56
6	0.03	0.21	1.28	2.82
7	0.02	0.26	1.28	2.74
8	0.02	0.30	1.18	2.80
9	0.02	0.22	1.17	3.04
10	0.02	0.26	1.42	2.72
Promedio:	**0.02**	**0.27**	**1.35**	**2.83**

Foto 5mb (5,39 MB (5.658.596 bytes))				
N° corrida	**Sin pérdidas (en segundos)**	**1% pérdidas (en segundos)**	**5% pérdidas (en segundos)**	**10% pérdidas (en segundos)**
1	0.09	1.29	6.96	14.72
2	0.09	1.49	6.71	14.13
3	0.09	1.48	6.79	14.54
4	0.09	1.44	6.89	15.55
5	0.09	1.57	6.43	15.07
6	0.09	1.12	6.92	14.70
7	0.09	1.19	7.35	14.44
8	0.08	1.29	6.55	13.92
9	0.09	1.31	6.49	14.78
10	0.09	1.31	6.96	14.88
Promedio:	**0.09**	**1.35**	**6.80**	**14.67**

Libro.pdf (18,1 MB (18.996.984 bytes))				
N° corrida	**Sin pérdidas (en segundos)**	**1% pérdidas (en segundos)**	**5% pérdidas (en segundos)**	**10% pérdidas (en segundos)**
1	0.28	4.64	23.17	48.25
2	0.28	4.61	21.93	47.36
3	0.28	4.45	23.42	48.31
4	0.30	4.36	22.41	47.45
5	0.29	4.40	22.98	47.58
6	0.28	4.24	22.39	48.39
7	0.28	4.72	21.40	47.56
8	0.27	4.28	23.04	48.46
9	0.29	4.09	23.21	48.34
10	0.28	4.40	22.03	47.05
Promedio:	**0.28**	**4.42**	**22.60**	**47.87**

Subida (SaW)

Foto 1mb (1,07 MB (1.125.918 bytes))				
N° corrida	**Sin pérdidas (en segundos)**	**1% pérdidas (en segundos)**	**5% pérdidas (en segundos)**	**10% pérdidas (en segundos)**
1	0.05	0.61	3.02	6.34
2	0.05	0.57	2.75	6.75
3	0.05	0.58	2.90	6.01
4	0.06	0.42	2.49	5.56
5	0.05	0.50	2.57	6.11
6	0.05	0.48	3.13	6.35
7	0.05	0.48	2.80	6.36
8	0.05	0.45	2.58	6.04
9	0.05	0.53	2.60	5.85
10	0.05	0.53	2.83	6.21
Promedio:	**0.05**	**0.51**	**2.77**	**6.16**

Foto 5mb (5,39 MB (5.658.596 bytes))				
N° corrida	**Sin pérdidas (en segundos)**	**1% pérdidas (en segundos)**	**5% pérdidas (en segundos)**	**10% pérdidas (en segundos)**
1	0.24	2.97	13.25	30.57
2	0.25	2.51	13.95	31.15
3	0.24	2.34	13.79	30.52
4	0.26	2.53	13.32	30.78
5	0.24	2.33	12.76	30.25
6	0.23	2.27	13.65	31.00
7	0.24	2.63	13.44	30.39
8	0.24	2.31	14.23	30.89
9	0.23	2.69	13.46	30.49
10	0.23	2.51	12.50	29.68
Promedio:	**0.24**	**2.51**	**13.43**	**30.57**

Libro.pdf (18,1 MB (18.996.984 bytes))				
N° corrida	**Sin pérdidas (en segundos)**	**1% pérdidas (en segundos)**	**5% pérdidas (en segundos)**	**10% pérdidas (en segundos)**
1	0.80	8.80	45.51	102.53
2	0.76	8.97	45.87	101.01
3	0.78	8.49	45.69	101.80
4	0.79	8.94	43.83	100.80
5	0.76	8.96	44.93	101.57
6	0.81	8.92	44.05	100.29
7	0.78	8.69	45.06	102.37
8	0.79	8.87	44.87	102.48
9	0.81	8.93	46.13	102.37
10	0.79	8.87	46.25	102.18
Promedio:	**0.79**	**8.84**	**45.22**	**101.74**

Descarga (SaW)

Foto 1mb (1,07 MB (1.125.918 bytes))				
N° corrida	**Sin pérdidas (en segundos)**	**1% pérdidas (en segundos)**	**5% pérdidas (en segundos)**	**10% pérdidas (en segundos)**
1	0.05	0.44	2.72	6.27
2	0.04	0.44	3.11	5.92
3	0.05	0.54	2.72	6.15
4	0.05	0.64	2.34	6.10
5	0.05	0.64	2.45	6.45
6	0.04	0.71	2.53	6.02
7	0.04	0.72	3.04	5.74
8	0.05	0.54	2.40	6.32
9	0.04	0.41	2.52	5.91
10	0.05	0.42	2.84	5.91
Promedio:	**0.05**	**0.55**	**2.66**	**6.08**

Foto 5mb (5,39 MB (5.658.596 bytes))				
N° corrida	**Sin pérdidas (en segundos)**	**1% pérdidas (en segundos)**	**5% pérdidas (en segundos)**	**10% pérdidas (en segundos)**
1	0.23	2.60	13.64	30.36
2	0.23	2.67	13.27	31.51
3	0.23	2.68	14.45	30.66
4	0.23	2.80	13.48	31.99
5	0.24	2.66	13.19	31.59
6	0.24	2.63	13.77	30.81
7	0.23	2.64	13.86	29.94
8	0.25	2.49	13.45	30.74
9	0.23	2.40	13.44	30.33
10	0.23	2.38	13.10	30.24
Promedio:	**0.23**	**2.59**	**13.56**	**30.81**

Libro.pdf (18,1 MB (18.996.984 bytes))				
N° corrida	**Sin pérdidas (en segundos)**	**1% pérdidas (en segundos)**	**5% pérdidas (en segundos)**	**10% pérdidas (en segundos)**
1	0.78	8.71	44.99	102.54
2	0.79	8.17	43.48	103.60
3	0.82	9.20	44.39	102.20
4	0.76	9.11	46.77	102.62
5	0.82	8.94	44.89	103.97
6	0.76	8.98	45.56	101.13
7	0.80	8.69	44.87	101.44
8	0.77	8.39	45.04	100.91
9	0.78	8.98	44.90	102.42
10	0.78	9.48	44.20	102.01
Promedio:	**0.78**	**8.86**	**44.91**	**102.28**

Análisis

Upload GBN, tiempos promedios (seg). Sin, 1%, 5% y 10% perdidas respectivamente.				
Foto 1mb	**0.02**	**0.25**	**1.32**	**2.89**
Foto 5mb	**0.08**	**1.32**	**6.66**	**14.42**
Libro 18mb	**0.27**	**4.54**	**22.40**	**48.17**

Upload SaW, tiempos promedios (seg). Sin, 1%, 5% y 10% perdidas respectivamente.				
Foto 1mb	**0.05**	**0.51**	**2.77**	**6.16**
Foto 5mb	**0.24**	**2.51**	**13.43**	**30.57**
Libro 18mb	**0.79**	**8.84**	**45.22**	**101.74**

En el caso de la subida, podemos observar que consistentemente el tiempo que tomó en GBN fue casi la mitad del que le llevó a SaW realizar lo mismo.

Esto ocurre porque SaW manda de a un paquete y en cada caso espera a que sea confirmado para continuar con el siguiente. GBN trabaja con varios paquetes al mismo tiempo lo que genera que tarde mucho menos.

También podemos observar que sin pérdidas el tiempo que lleva es casi instantáneo como si fuera copiar un archivo, lo cual es consistente con la realidad.

En los casos con pérdidas vemos que la relación entre cada proceso con distinta cantidad de pérdida de paquetes es relativamente proporcional.

Download GBN, tiempos promedios (seg). Sin, 1%, 5% y 10% perdidas respectivamente.				
Foto 1mb	**0.02**	**0.27**	**1.35**	**2.83**
Foto 5mb	**0.09**	**1.35**	**6.80**	**14.67**
Libro 18mb	**0.28**	**4.42**	**22.60**	**47.87**

Download SaW, tiempos promedios (seg). Sin, 1%, 5% y 10% perdidas respectivamente.				
Foto 1mb	**0.05**	**0.55**	**2.66**	**6.08**
Foto 5mb	**0.23**	**2.59**	**13.56**	**30.81**
Libro 18mb	**0.78**	**8.86**	**44.91**	**102.28**

En el caso de la descarga, se puede observar que las relaciones de tiempos entre GBN y SaW se mantienen como sucede para la subida.

También es importante remarcar que tanto para subida como para descarga para una misma forma de operar los tiempos evidenciados son muy similares, de forma que ambas funcionan sin mayores inconvenientes y de forma consistente entre sí.

Vemos que incluso con pérdidas se obtienen tiempos razonables tanto para la subida como para la descarga en ambas implementaciones. Por ejemplo, sabiendo que el tamaño del libro es de 18mb, una subida con GBN y 10% de pérdidas tomó ~48 segundos, esto es alrededor de 0,40mb/s.

Preguntas

1. Describa la arquitectura cliente-servidor.

La arquitectura de una aplicación define como es la estructura de la aplicación entre los hosts que la ejecutan. En el tipo de arquitectura cliente-servidor hay un host “servidor”, siempre activo, que responde solicitudes de los hosts “clientes” que generan esas solicitudes.
Características: La dirección ip y puerto del servidor están formalmente establecidas.
Observación: Si un host server no puede dar soporte a la cantidad de solicitudes de clientes, se expanden a un conjunto de hosts que conforman un servidor virtual en un “data center”, lo que incurre en costos adicionales.

¿Cómo es la estructura de nuestros hosts clientes y servidores?

En el siguiente gráfico se muestra un diagrama de clases pertinente al servidor. Para remarcar se tiene al FileHandler y, en particular, sus clases hijas FileReceiver y FileSender, que en el servidor son un hilo para poder mantener una comunicación continua con múltiples clientes en paralelo. FileHandler tiene un CustomSocket que puede ser un socket con política Go-Back-N o uno con Stop and Wait. En la implementación utilizada, SaW es un caso particular de GBN con N = 1, por lo que se hizo que herede de éste.

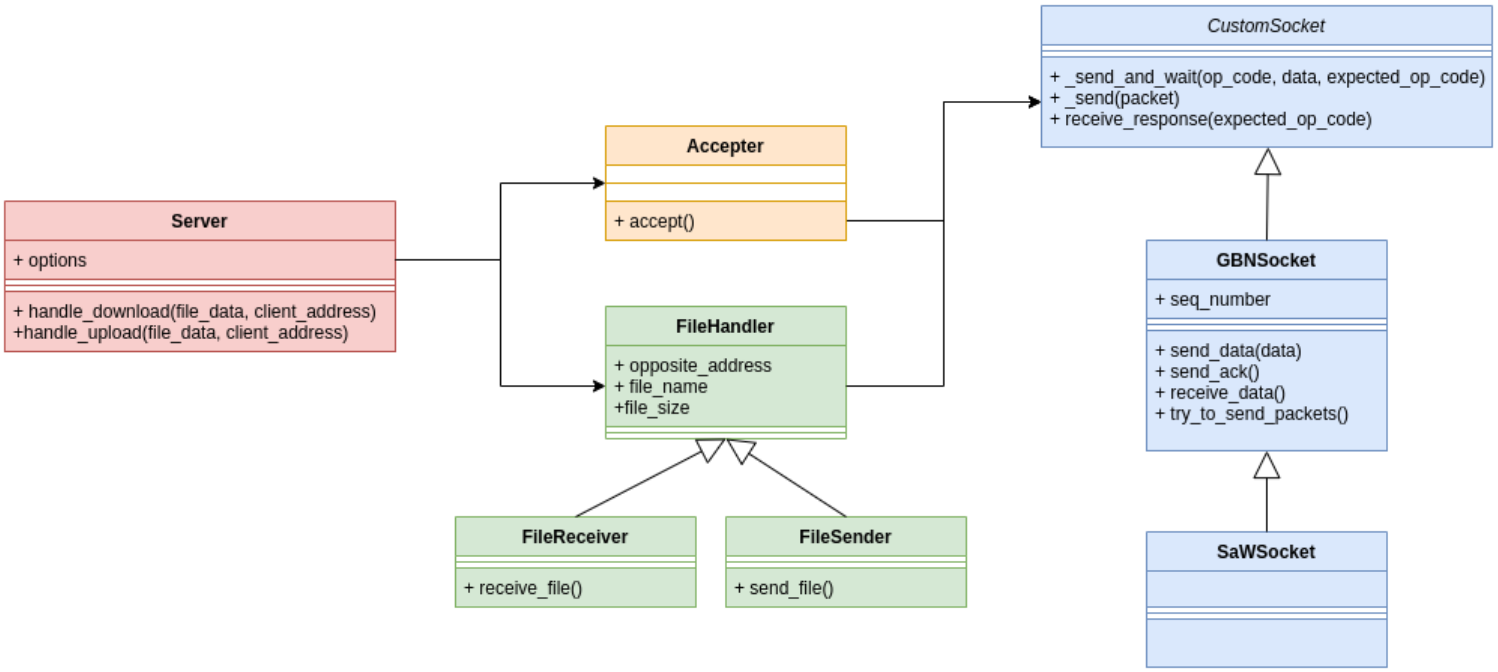


Figura 18.Diagrama de clase Servidor.

Por otro lado tenemos el diagrama del cliente, muy parecido al servidor, pero sin aceptador. Para este caso FileReceiver y FileSender no son hilos ya que el cliente no necesita paralelizar el trabajo.

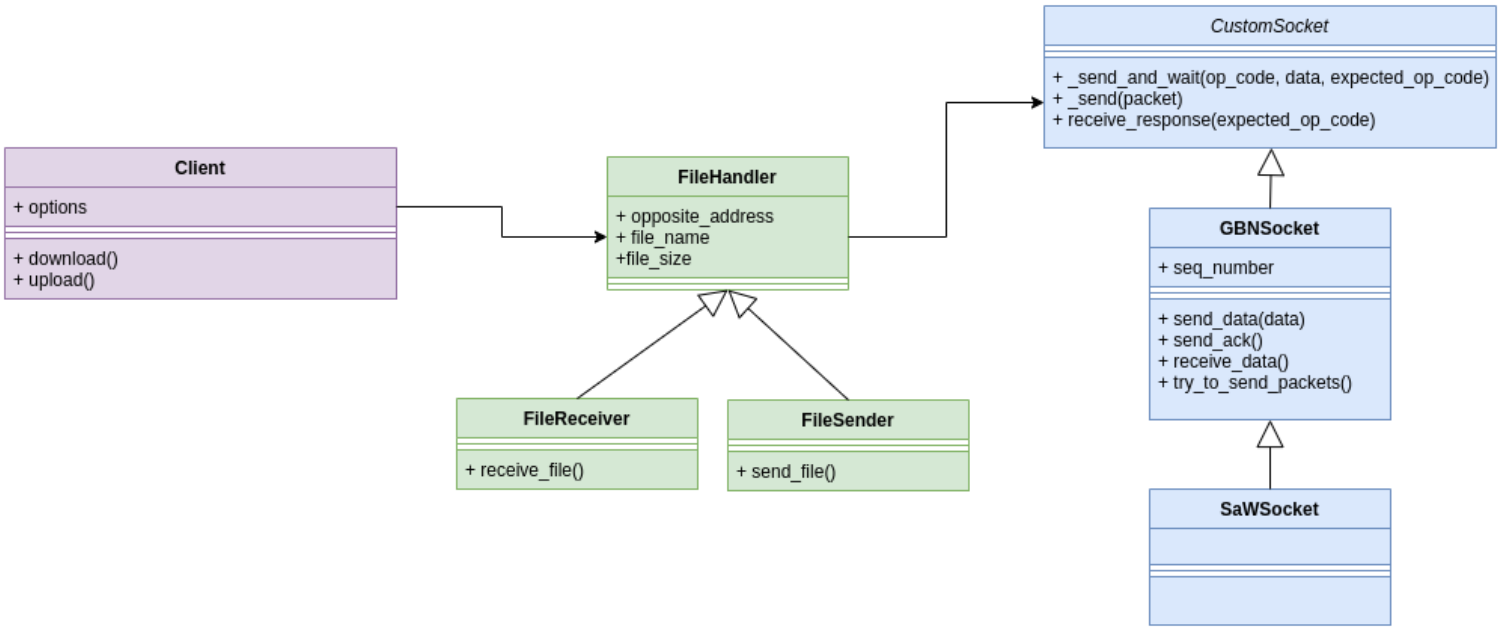


Figura 19.Diagrama de clase Cliente

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

Un protocolo en la capa de aplicación define como los procesos de la aplicación distribuida se pasan mensajes. Esto es:

- Los tipos de mensajes que se intercambian entre procesos.
- La sintaxis de estos mensajes.
- La semántica, o significado, de la información en cada uno de los mensajes.
- Las reglas a partir de las cuales los procesos saben cuándo y cómo enviar o responder mensajes.

Observación: Los protocolos de aplicación se consideran un componente de una aplicación de red.

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

El protocolo de aplicación desarrollado utiliza los servicios del protocolo de capa de transporte UDP. Esto es, los servicios de envío best effort (heredado del protocolo de red IP)

y recepción de mensajes no corruptos (provisto por udp por medio del mecanismo de checksums).

Nuestro protocolo se encargará entonces de construir un servicio de comunicación confiable para el intercambio de mensajes entre aplicaciones (***¿Por qué nuestro protocolo es confiable?***)

Tipos de mensajes

DATA, ACK, SV_INTRODUCTION, NSQ_ACK, DOWNLOAD, UPLOAD, END, END_ACK.

Sintaxis de mensajes

Se trata de un protocolo binario. La figura 20 detalla la estructura de cada tipo de mensaje. La idea es que según el código de operación, indicado en el primer byte, se sepa interpretar el resto del mensaje.

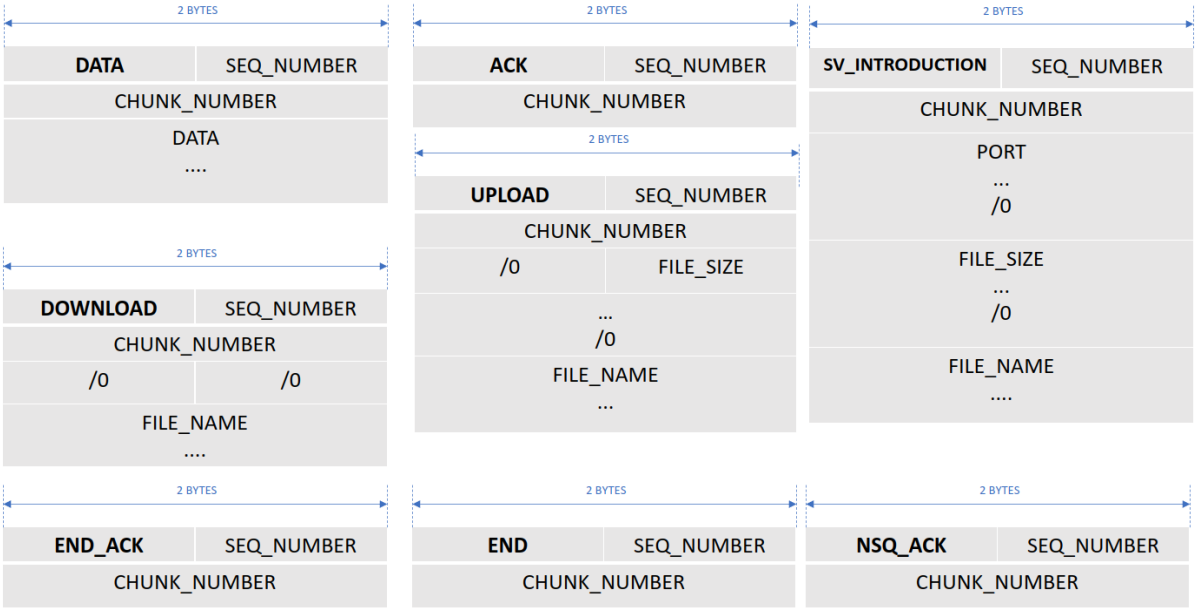


Figura 20. Sintaxis de cada tipo de mensaje.

Semántica de cada tipo de mensaje

- **DOWNLOAD** (0): se utiliza sólo en el primer mensaje desde el cliente al servidor para solicitar una operación de descarga. Representa una solicitud de descarga.
- **UPLOAD** (1): se utiliza sólo en el primer mensaje desde el cliente al servidor. Representa una solicitud de una operación de subida.
- **SV_INTRODUCTION** (2): Representa la respuesta del servidor al cliente de mensajes DOWNLOAD y UPLOAD. Aquí se envía la información de un nuevo puerto

asignado para la comunicación con el servidor, y según el caso, el tamaño del archivo y nombre del archivo.
- **DATA** (3): Indica que el mensaje contiene una porción del archivo que está siendo transferido. Los números de secuencia y de chunk permiten identificar de qué porción de datos se trata.
- **ACK** (4): (Acknowledge) Indica cual es el último mensaje DATA recibido correctamente al momento de generar el mensaje por medio de los campos SEQ_NUMBER y CHUNK_NUMBER
- **NSQ_ACK** (5): (Non Sequential Acknowledge) Indica que un mensaje del handshake se recibió correctamente. A diferencia del ACK, su recepción no genera modificaciones en el número de secuencia de los sockets.
- **END_ACK** (6): Indica que el mensaje END enviado por el emisor fue recibido. Al igual que el NSQ_ACK, su recepción no genera cambios en el número de secuencia, pero le indica al emisor que puede iniciar el envío del próximo chunk.
- **END** (7): Utilizado para indicar que se terminó de enviar un chunk del archivo.

Reglas

- *Durante el handshake*, se intercambiaran mensajes de tipo UPLOAD/DOWNLOAD, NSQ_ACK y SV_INTRODUCTION.
- Luego, *durante la transferencia*, serán mensajes DATA y ACK. Se procesarán aquellos con el número de secuencia correcto y se descartarán los que tengan número incorrecto.
- *Al finalizar el envío del archivo*, se enviarán mensajes END y END_ACK.

Cualquier mensaje recibido fuera de orden será descartado sin mayor procesamiento. Por ejemplo, si durante la transferencia se recibe un SV_INTRODUCTION.

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?

Servicios UDP

- **Multiplexado y demultiplexado**

  El multiplexado consta de recibir múltiples chunks de datos de diferentes sockets, agregarles el header, que será usado en el demultiplexado, creando los paquetes y pasarlos a la capa de red. En el caso del demultiplexado se utilizan los campos del header para saber a qué socket pertenece el paquete recibido de la capa de red.

- **Control de integridad**

  Utiliza el campo checksum del paquete para realizar un chequeo rápido de que el mismo no sufrió modificaciones.

Servicios TCP

- **Transferencia de datos confiable (reliable data transfer service)**

  Garantiza que todos los paquetes se enviarán, llegarán en orden, sin errores y sin duplicados.

- **Orientado a una conexión**

  TCP realiza una transferencia de datos previa a la transferencia de datos de la capa de aplicación, esta transferencia de datos es el handshake. El handshake de TCP se denomina three-way handshake, ésto es porque intercambia 3 mensajes para establecer la conexión. El primer mensaje usa el bit SYN en 1 para indicar que se quiere realizar una sincronización, el segundo mensaje es en respuesta a éste prendiendo ahora el bit SYN y el bit ACK para indicar que está ok para realizar la conexión y el último mensaje simplemente prende el bit ACK para acusar el recibo del segundo mensaje. Una vez realizado el handshake se dice que se tiene una conexión full-duplex, es decir hay un flujo en el que ambos hosts pueden intercambiar información.

- **Control de congestión**

  Este servicio que provee TCP utiliza mecanismo de tiempo para ir midiendo que tan congestionada está la red e ir controlando que tantos paquetes envía a la misma, es decir a más congestión enviará menos paquetes.

- **Multiplexado y demultiplexado**

- **Control de integridad**

Usos

TCP puede sonar el protocolo ideal ya que tiene muchos más servicios y garantías por sobre UDP, pero estos chequeos y servicios vienen a un costo de tiempo, por lo que la decisión de usar uno u otro depende del uso que le demos.

Usos para TCP

- Páginas WEB
- Transferencia de archivos
- Mail

Usos para UDP

- Streaming
- Juegos online
- VoIP

En conclusión cuando tengamos la necesidad de que la información llegue sin importar el tiempo nos inclinaremos a usar TCP, mientras que cuando queramos menores tiempos y no nos importe perder algo de información el ideal será UDP.

Conclusiones

Hemos construido un protocolo de aplicación confiable para la red de internet basándonos en los diversos mecanismos desarrollados en este trabajo. Hemos estudiado y comparado la performance de nuestras dos implementaciones (Go Back N y Stop and Wait) y en base a los resultados de nuestras mediciones hemos corroborado que los resultados coinciden con los esperados. Esto es, la estrategia de pipelining de Go Back N ofrece un mejor desempeño.

También hemos aprendido que, si bien protocolos como TCP proveen confiabilidad, no es necesariamente la única alternativa y que, basándonos en UDP, podemos construir poderosas aplicaciones que sean confiables y que nos den mayor libertad en términos de implementación.

También es cierto que no todas las aplicaciones necesitan confiabilidad. Pues vimos que UDP en muchos casos es más que suficiente, e incluso en nuestra implementación vimos que la pérdida de ciertos paquetes no es tan grave (como algunos casos de drops de mensajes ACKs).

En adelante, como desarrolladores de aplicaciones distribuidas que requieran comunicación confiable, probablemente usemos TCP, pero bien sabemos que no es la única opción.