

# Trivial File Transfer Protocol

Mateos Hoyos, Alfonso José  
Luis Lazo, Gabino

**Abstract**—En esta práctica se documenta la realización del protocolo de transferencia de archivos TFTP, el cual está pensado para UDP, pero por motivos académicos también se ha implementado en TCP. En los posteriores puntos se comentan las diferencias en ambas implementaciones, además de las pruebas realizadas.

## I. INTRODUCCIÓN

En esta práctica se ha implementado el protocolo de transferencia de archivos TFTP, el cual es una versión más simple del protocolo FTP. La implementación consta de dos versiones, una para TCP y otra para UDP. Es importante tener en cuenta las principales diferencias de la aplicación dependiendo del protocolo de transporte:

- Para TCP es necesario tener en cuenta, ya que es un protocolo con conexión y asentimiento, que no son necesarios los ACK de cada paquete; no obstante, con la intención de seguir el protocolo, se han implementado los paquetes de asentimiento de bloques y mensajes de error.
- Para UDP, ha sido necesario implementar timeouts en las funciones bloqueantes de recepción de mensajes del socket, debido a que no existe conexión entre los participantes.

Para asegurar la existencia de concurrencia en ambas implementaciones, cada conexión nueva se procesa en hilos diferentes.

En la siguientes secciones se detallan la arquitectura y las pruebas de funcionamiento realizadas.

## II. ARQUITECTURA

El programa consta de 4 fuentes principales:

- packet.c: en este fuente se especifican las estructuras que se usan para el envío, recepción y procesamiento de los mensajes del protocolo. En concreto:
  - Estructura request\_msg: donde se guardan las cadenas filename y mode usadas en los mensajes de petición de escritura o lectura.
  - Estructura data\_msg: la estructura principal de la aplicación para la transferencia de archivos, donde se guardan el número de bloque, el dato en concreto y el tamaño del mismo.
  - Estructura ack\_msg: que contiene el número del bloque al cual se asiente.
  - Estructura err\_msg: donde se guarda un código y un mensaje de error.
  - Estructura packet: en la cual se guardan los mensajes típicos del protocolo para el procesamiento en la aplicación. El primer campo es el código de operación (2

bytes) y el segundo campo es una unión que contiene las estructuras anteriores.

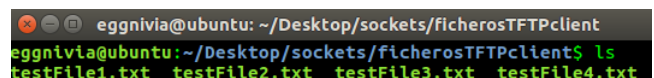
Además, se implementan las funciones de serialización de la estructura packet para la transferencia del protocolo TFTP.

- transfer.c: en este fuente se encapsulan las funciones de transferencia de sockets, las cuales son diferente para UDP y TCP, utilizando dinámicamente ambos protocolos de transporte e implementando los timeouts correspondientes para UDP.
- action.c: es la capa más alto nivel, donde se encapsulan las operaciones típicas de TFTP (get y put), diferenciando el rol de cliente o servidor.
- server.c/client.c: recoge las funciones principales del programa, donde se realiza toda la inicialización. El servidor quedará a la escucha de los sockets, uno TCP y otro UDP. Cada vez que una nueva conexión sea recibida en un socket, se creará un nuevo hilo para gestionarla. En el caso de UDP, como no existe conexión explícita, se creará un puerto efímero en el servidor para la sesión con el cliente. Por otro lado, el cliente ejecutará la orden recibida y se iniciará la comunicación.

## III. PRUEBAS

La primera prueba se realizará en un entorno local, consistirá en enviar cuatro archivos de cuatro clientes diferentes al servidor y recibir otros cuatro archivos distintos a otros cuatro clientes. Por lo tanto, existirán ocho clientes en esta prueba que el servidor deberá gestionar concurrentemente. La mitad de los clientes usaran TCP y la otra mitad UDP.

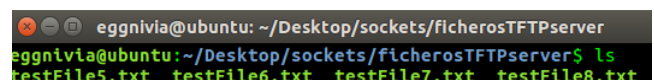
Los archivos utilizados por los clientes son los siguientes:



```
eggnivia@ubuntu: ~/Desktop/sockets/ficherosTFTPclient
eggnivia@ubuntu:~/Desktop/sockets/ficherosTFTPclient$ ls
testFile1.txt testFile2.txt testFile3.txt testFile4.txt
```

Fig. 1. ficherosTFTPclient

Los archivos guardados en el servidor son los siguientes:



```
eggnivia@ubuntu: ~/Desktop/sockets/ficherosTFTPserver
eggnivia@ubuntu:~/Desktop/sockets/ficherosTFTPserver$ ls
testFile5.txt testFile6.txt testFile7.txt testFile8.txt
```

Fig. 2. ficherosTFTPserver

La ejecución del programa será la siguiente:

```
./server
./client $host TCP e testFile1.txt &
./client $host UDP e testFile2.txt &
./client $host TCP e testFile3.txt &
./client $host UDP e testFile4.txt &
./client $host TCP l testFile5.txt &
./client $host UDP l testFile6.txt &
./client $host TCP l testFile7.txt &
./client $host UDP l testFile8.txt &
```

Fig. 3. Script prueba 1

Tras la ejecución del programa se comprueba el log común para todos los clientes y el log del servidor. Se observa como las conexiones se han realizado satisfactoriamente en puertos diferentes. En la Figura 5 vemos como aparecen las cuatro conexiones TCP que se han establecido durante esta primera prueba.

```
Sat Dec 1 20:01:35 2018: Connected to localhost on port 58102
All done at Sat Dec 1 20:01:35 2018
Sat Dec 1 20:01:35 2018: Connected to localhost on port 58100
All done at Sat Dec 1 20:01:35 2018
Sat Dec 1 20:01:35 2018: Connected to localhost on port 44077
All done at Sat Dec 1 20:01:35 2018
Sat Dec 1 20:01:35 2018: Connected to localhost on port 58098
All done at Sat Dec 1 20:01:35 2018
Sat Dec 1 20:01:35 2018: Connected to localhost on port 33667
All done at Sat Dec 1 20:01:35 2018
Sat Dec 1 20:01:35 2018: Connected to localhost on port 39651
All done at Sat Dec 1 20:01:35 2018
Sat Dec 1 20:01:35 2018: Connected to localhost on port 58104
All done at Sat Dec 1 20:01:35 2018
Sat Dec 1 20:01:35 2018: Connected to localhost on port 47721
All done at Sat Dec 1 20:01:35 2018
```

Fig. 4. Log cliente prueba 1

```
Startup from localhost port 58102 at Sat Dec 1 20:01:35 2018
Startup from localhost port 58100 at Sat Dec 1 20:01:35 2018
Startup from localhost port 58098 at Sat Dec 1 20:01:35 2018
Startup from localhost port 58104 at Sat Dec 1 20:01:35 2018
```

Fig. 5. Log servidor prueba 1

Mirando los directorios, se comprueba como los archivos se han recibido de forma satisfactoria.

```
eggnavia@ubuntu:~/Desktop/sockets/ficherosTFTPclient$ ls
testFile1.txt testFile3.txt testFile5.txt testFile7.txt
testFile2.txt testFile4.txt testFile6.txt testFile8.txt
```

Fig. 6. ficherosTFTPclient

```
eggnavia@ubuntu:~/Desktop/sockets/ficherosTFTPserver$ ls
testFile1.txt testFile3.txt testFile5.txt testFile7.txt
testFile2.txt testFile4.txt testFile6.txt testFile8.txt
```

Fig. 7. ficherosTFTPserver

La siguiente prueba, con el objetivo de forzar un error, se ha realizado inmediatamente después de la anterior. Como existen los mismos ficheros en ambos directorios, se deberá producir errores de existencia de archivos, tanto para los clientes como para el servidor.

```
Sat Dec 1 20:18:48 2018: Connected to localhost on port 58108
Sat Dec 1 20:18:48 2018: File already exists at 'get client'(filename=fiche
All done at Sat Dec 1 20:18:48 2018
Sat Dec 1 20:18:48 2018: Connected to localhost on port 51128
Sat Dec 1 20:18:48 2018: File already exists at 'get client'(filename=fiche
All done at Sat Dec 1 20:18:48 2018
Sat Dec 1 20:18:48 2018: Connected to localhost on port 59851
Sat Dec 1 20:18:48 2018: File already exists at 'put_server' (put client)
All done at Sat Dec 1 20:18:48 2018
Sat Dec 1 20:18:48 2018: Connected to localhost on port 42085
Sat Dec 1 20:18:48 2018: File already exists at 'get client'(filename=fiche
All done at Sat Dec 1 20:18:48 2018
Sat Dec 1 20:18:48 2018: Connected to localhost on port 58106
Sat Dec 1 20:18:49 2018: File already exists at 'put_server' (put client)
All done at Sat Dec 1 20:18:49 2018
Sat Dec 1 20:18:48 2018: Connected to localhost on port 35041
Sat Dec 1 20:18:49 2018: File already exists at 'put_server' (put client)
All done at Sat Dec 1 20:18:49 2018
Sat Dec 1 20:18:49 2018: Connected to localhost on port 58110
Sat Dec 1 20:18:49 2018: File already exists at 'get client'(filename=fiche
All done at Sat Dec 1 20:18:49 2018
Sat Dec 1 20:18:49 2018: Connected to localhost on port 58112
Sat Dec 1 20:18:49 2018: File already exists at 'put_server' (put client)
All done at Sat Dec 1 20:18:49 2018
```

Fig. 8. Log cliente prueba 2

En efecto, se observan dos tipos de errores en el log de los clientes: un tipo de error generado por los mismos clientes al intentar hacer una operación de lectura (get) para un archivo que ya existe en su directorio, y otro tipo, siendo un mensaje de error enviado por el servidor, indicando que ya existe el archivo a enviar.

```
Startup from localhost port 58106 at Sat Dec 1 20:18:48 2018
Startup from localhost port 58108 at Sat Dec 1 20:18:49 2018
Sat Dec 1 20:18:49 2018: rcv shutdown received
Startup from localhost port 58110 at Sat Dec 1 20:18:49 2018
Sat Dec 1 20:18:49 2018: rcv shutdown received
Startup from localhost port 58112 at Sat Dec 1 20:18:49 2018
```

Fig. 9. Log servidor prueba 2

En el log del servidor, se observan las conexiones TCP de modo análogo a la prueba anterior, pero además, los cierres de conexión (shutdown) de los clientes TCP de lectura, que abren una conexión en el socket solo para cerrarlas poco después al comprobar que ya existe el archivo que querían recibir.

La tercera prueba consistirá en lanzar el servidor en una máquina y un cliente en otra, en una subred diferente, para así comprobar que se envía de forma correcta un archivo a través de internet.



Fig. 10. Prueba 3

Para ello, se hace port-forwarding en el router a una máquina de la subred donde residirá el servidor. El cliente, por su parte, deberá indicar al ejecutarse que el host está en la IP pública del router del servidor, conectándose al puerto previamente abierto en el mismo.

Teniendo el servidor activo lanzamos un cliente desde una subred diferente, en modo escritura, para enviarle un archivo y así comprobar el correcto funcionamiento.

```
eggnivla@ubuntu: ~/Desktop/sockets
eggnivla@ubuntu:~/Desktop/sockets$ ./client 90.69.206.102 TCP e fileTest2.jpg
```

Fig. 11. Cliente prueba 3

Se observa el directorio del servidor y se comprueba que el fichero ha llegado correctamente.

```
gabino@virtualUbuntu: ~/univ/3.1/sockets/code/ficherosTFTPserver
gabino@virtualUbuntu:~/univ/3.1/sockets/code/ficherosTFTPserver$ ls
FileTest2.jpg
```

Fig. 12. Directorio servidor prueba 3

Por último, se realiza la cuarta y última prueba, la cual consistirá en lanzar el servidor en una máquina que está conectada a la red 'olivo.fis.usal.es'. Para ello, se ha utilizado una de las máquinas de nuestra facultad, en concreto del aula 'sun'.

El estado inicial del directorio del servidor será el siguiente:

```
i1028058@sundia07: ~/Z/sockets/code
Archivo Editar Ver Buscar Terminal Ayuda
/home/i1028058/sockets/code/ficherosTFTPserver$ ls
testFile5.txt testFile6.txt testFile7.txt testFile8.txt
```

Fig. 13. ficherosTFTPserver prueba 4 (antes)

Se lanza el servidor y salimos de olivo. Desde la misma máquina pero si estar logeados en olivo, se lanza el mismo script usado en la prueba 1 (Figura 3).

El estado inicial del directorio del cliente será el siguiente:

```
i1028058@sundia07: ~/sockets/code/ficherosTFTPclient
Archivo Editar Ver Buscar Terminal Ayuda
i1028058@sundia07:~/sockets/code/ficherosTFTPclient$ ls
testFile1.txt testFile2.txt testFile3.txt testFile4.txt
```

Fig. 14. ficherosTFTPclient prueba 4 (antes)

Ejecutamos el script usado en la Figura 3, cambiando la IP, en este caso se usa un nombre:

```
i1028058@sundia07: ~/Z/sockets/code
Archivo Editar Ver Buscar Terminal Ayuda
/home/i1028058/sockets/code$ ./lanzaServidor.sh olivo.fis.usal.es
```

Fig. 15. Script prueba 4

Tras ejecutar los ocho clientes, miramos de nuevo los directorios:

```
i1028058@sundia07: ~/Z/sockets/code
Archivo Editar Ver Buscar Terminal Ayuda
/home/i1028058/sockets/code/ficherosTFTPserver$ ls
testFile1.txt testFile3.txt testFile5.txt testFile7.txt
testFile2.txt testFile4.txt testFile6.txt testFile8.txt
```

Fig. 16. ficherosTFTPserver prueba 4 (después)

```
i1028058@sundia07: ~/sockets/code/ficherosTFTPclient
Archivo Editar Ver Buscar Terminal Ayuda
i1028058@sundia07:~/sockets/code/ficherosTFTPclient$ ls
testFile1.txt testFile3.txt testFile5.txt testFile7.txt
testFile2.txt testFile4.txt testFile6.txt testFile8.txt
```

Fig. 17. ficherosTFTPclient prueba 4 (después)

Y se comprueba que la transmisión de los ocho ficheros se ha realizado con éxito. Para concluir con esta prueba, observamos el contenido de los logs que se han generado:

```
Mon Dec 3 11:47:05 2018: Startup from 172.20.2.203 port 36864.Mon Dec
Completed 172.20.2.203 port 36864
Mon Dec 3 11:47:21 2018: Startup from 172.20.2.203 port 36865.Mon Dec
Completed 172.20.2.203 port 36865
Mon Dec 3 11:47:37 2018: Startup from 172.20.2.203 port 36866.Mon Dec
Completed 172.20.2.203 port 36866
Mon Dec 3 11:47:53 2018: Startup from 172.20.2.203 port 36867.Mon Dec
Completed 172.20.2.203 port 36867
```

Fig. 18. Log cliente prueba 4

```
Mon Dec 3 11:46:33 2018: Connected to olivo.fis.usal.es on port 43688
All done at Mon Dec 3 11:46:33 2018
Mon Dec 3 11:46:33 2018: Connected to olivo.fis.usal.es on port 47607
All done at Mon Dec 3 11:46:33 2018
Mon Dec 3 11:46:33 2018: Connected to olivo.fis.usal.es on port 58887
All done at Mon Dec 3 11:46:34 2018
Mon Dec 3 11:46:33 2018: Connected to olivo.fis.usal.es on port 38335
All done at Mon Dec 3 11:46:34 2018
Mon Dec 3 11:46:33 2018: Connected to olivo.fis.usal.es on port 36864
All done at Mon Dec 3 11:47:05 2018
Mon Dec 3 11:46:33 2018: Connected to olivo.fis.usal.es on port 36865
All done at Mon Dec 3 11:47:27 2018
Mon Dec 3 11:46:33 2018: Connected to olivo.fis.usal.es on port 36866
All done at Mon Dec 3 11:47:38 2018
Mon Dec 3 11:46:33 2018: Connected to olivo.fis.usal.es on port 36867
All done at Mon Dec 3 11:47:54 2018
```

Fig. 19. Log server prueba 4

Vemos como todo ha ido bien y además el programa implementado es capaz de resolver nombres.