TFTP Protocol

J. Bermudez

Departamento de ingeniería, Pontificia Universidad Javeriana

[1]juand.bermudez@javeriana.edu.co

Abstract— **Este articulo presenta el protocolo TFTP, sus características, funcionamiento, documentación. Así mismo se describe el proceso que se espera seguir para el desarrollo de un servidor TFTP propio, y las pruebas necesarias para comprobar su funcionamiento.**

Keywords— **TFTP, Protocol, Data, Files, Application.**

I. INTRODUCCION

El protocolo TFTP es un protocolo de la capa de aplicación, es usado para la transferencia de archivos, su nombre completo es Trivial File Transfer Protocol, fue originalmente diseñado por Noel Chiappa, pero fue posteriormente rediseñado por el mismo Noel en compañía de Bob Baldwin y Dave Clark [1].

El protocolo TFTP se especifica en la RFC 1350 de Julio del 1992, fue "lanzado" inicialmente en la RFC 783 de junio del año 1981, pero la RFC 1350 anteriormente mencionada dejaría obsoleta esta primera RFC, siendo así que todo el funcionamiento del protocolo TFTP se especifica en la RFC 1350.

TFTP es un protocolo cliente-servidor, por lo que debe haber un cliente TFTP corriendo el código TFTP client, y también un servidor corriendo el código TFTP server [2]. Así mismo el protocolo TFTP es un protocolo bastante liviano, con solo 2 posibles funciones, que son leer y escribir.

El protocolo TFTP es poco seguro, recae en el protocolo de capa de transporte UDP, por lo que no está orientado a la conexión, no tiene autenticación ni encriptación, por lo que cualquier paquete capturado de este protocolo es totalmente legible, es por esta razón que no es un protocolo usado comúnmente, su carencia de funcionalidades adicionales que podrían brindar seguridad, asi como su transporte basado en el protocolo UDP le brindan las que son asi mismo sus cualidades, las cuales son básicamente que este protocolo es un protocolo bastante sencillo y ligero, es rápido, asi mismo el código que se requiere tanto para cliente como para servidor es bastante corto y fácil de hacer en comparación a otros protocolos, esto lo convierte en una alternativa a usar en redes sencillas en donde la seguridad no sea un inconveniente o preocupación.

II. TFTP VS FTP

El protocolo FTP al igual que el TFTP es un protocolo de capa de aplicación que permite la transferencia de archivos, llamandose File Transfer Protocol. A diferencia del protocol oTFTP, el FTP es una opción más pesada pero así mismo más confiable y con más funcionalidades, este protocolo recae en el protocolo de capa de transporte TCP, por lo que es orientado a la conexión, así mismo permite autenticación con username y contraseña, permite encriptar los datos, puede recuperarse de un error, cosa que el protocolo TFTP no puede hacer [3].

Adicionalmente el protocolo FTP a diferencia del TFTP, tiene más funcionalidades adicionales a solo leer y escribir, como lo son listar los archivos guardados (desde un cliente), moverse entre directorios, renombrar archivos, eliminar archivos (desde el cliente) [4].

Toda la seguridad adicional y opciones adicionales del protocolo FTP lo hacen un protocolo más seguro y más recomendado para usar en la mayoría de casos, sobre todo en los que la seguridad es una preocupación, sin embargo, también es un protocolo mucho más pesado, lento, y complicado que lo que lo es el protocolo TFTP, este mismo aparece como la opción más liviana, sencilla y rápida del protocolo FTP.

III. DESCRIPCION PROTOCOLO

El protocolo TFTP usa el puerto 69 por parte del servidor, por lo que las comunicaciones deben ser dirigidas a este puerto. Como ya se mencionó anteriormente este protocolo solo tiene 2 posibles acciones a realizar desde el cliente, las cuales son leer (READ) y escribir (WRITE) [2].

El protocolo TFTP se considera un protocolo stop and wait, esto porque requiere un mensaje de acknolegded después de cada mensaje para comprobar que el mismo fue recibido, y hasta que ese ACK no se reciba no se enviara el siguiente paquete [2].

El protocolo soporta 3 modos de transferencia, siendo el primero el netascii, es decir, se envía la información siguiendo el código ASCII (de 8 bits), el segundo es octet, el cual es el modo binario que envía bytes de 8 bits, y el ultimo es el mail, aunque este último actualmente se encuentra obsoleto y no debe ser usado [1].

En total son 5 los posibles tipos de mensajes (distinto a las acciones) que tiene el protocolo TFTP, los mismos que se documentan en la RFC 1350 [1] se muestran a continuación.

TABLA I
TIPOS DE MENSAJE TFTP

OPCODE	OPERATION
1	Read Request (RRQ)
2	Write Request (WRQ)
3	Data (DATA)
4	Acknoledgment (ACK)
5	Error (ERROR)

En la tabla I se pueden ver los tipos de mensajes, estos se especifican a continuación:

A. Read Request

Este es el mensaje que envía el cliente TFTP al servidor TFTP cuando el mismo desea leer un archivo, este mensaje consta de 3 partes, el Opcode que en este caso sería el 1, el Opcode consta de 2 bytes, seguido del Filename (nombre del archivo), este se envía en una cadena string (codigo ASCII), se indica la terminación del nombre con un byte en 0, finalmente sigue el Mode (modo), que también se envía en una cadena de string cuya terminación se indica por un byte en 0, el modo puede ser “netascii” o “octet”, es importante mencionar que el modo no debe necesariamente estar en minúscula, puede estar escrito en cualquier combinación de minúsculas y mayúsculas, o totalmente en mayúsculas [1].

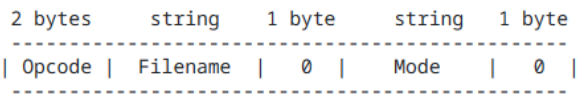


Figura 1. Estructura RRQ

En la figura 1 se puede apreciar de mejor forma los componentes y orden del tipo de mensaje RRQ.

B. Write Request

Este mensaje es parecido al Read Request, solo que en vez de solicitarle al servidor leer un archivo, le solicita escribir un archivo, los componentes son los mismos que tiene el mensaje RRQ como se ve en la Figura 1, solamente que ahora el Opcode sería el número 2.

C. Data

Este mensaje puede ser enviado tanto por el servidor como por el cliente, es el mensaje que contiene los datos, el servidor los enviara como respuesta a un RRQ, o el cliente los enviara por una WRQ. Debido a que se usa el protocolo UDP, el cual no garantiza orden de los paquetes, este orden debe realizarse en la capa de aplicación, es así que cada mensaje de tipo DATA tiene un Block number, indicando el número de ese bloque, con una capacidad de 2 bytes, así mismo tiene el Opcode de 2 bytes que sería el número 3, y finalmente tiene los datos, el número de bytes debe ser de 512 en cada mensaje de tipo DATA, es así que los archivos se deben dividir en bloques de 512 bytes, cuando se recibe un mensaje de tipo DATA con un tamaño menor a estos 512 bytes, se considera el final de la transferencia [1]

Figura 2. Estructura mensaje DATA

En la figura 2 se puede ver el orden de los 3 componentes del mensaje de tipo DATA.

D. Acknoledgment

Este mensaje se envia ya sea por el servidor o el cliente cada vez que se recibe un mensaje, como ya se mencionó se debe confirmar el recibimiento de dicho mensaje para poder continuar, es un mensaje bastante pequeño, consta de 4 bytes, siendo 2 del Opcode que para este mensaje sería el número 4, y 2 bytes para el numero de bloque del cual se está confirmando el recibimiento [1].

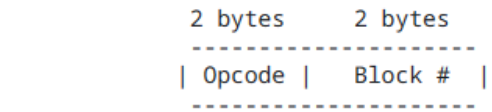


Figura 3. Estructura mensaje ACK

En la figura 3 se puede ver la estructura del mensaje ACK.

E. Error

Este mensaje se envía cuando sucede cualquier tipo de error durante el proceso, como ya se mencionó el protocolo TFTP no soporta errores, por lo que una vez sucede un error todo el proceso termina abruptamente [1]. Este mensaje se compone el Opcode que sería el número 5, 2 bytes que indican el código de error, un string con el mensaje de error terminado por un byte en 0.

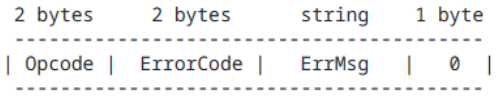


Figura 4. Estructura mensaje ERROR

En la figura 4 se puede ver la estructura del mensaje tipo ERROR. A continuación, se presenta la tabla con los códigos de error y su causa.

TABLA II
CÓDIGOS DE ERROR

Valor	Significado
0	No definido
1	Archivo no encontrado
2	Violación de acceso
3	Disco lleno
4	Operación TFTP ilegal
5	ID de transferencia desconocido
6	El archivo ya existe
7	No hay ese usuario

Una vez se tienen claro los distintos tipos de mensajes se puede profundizar en como estos mensajes interactúan, si el cliente quiere escribir (crear) un archivo, el orden será el siguiente, en este caso se imaginará que el nombre del archivo es "confg" y el tamaño es de 1590 bytes, se explicara la interacción por medio de lenguaje verbal.

1. El cliente envía un WRQ al servidor, con el nombre del archivo "confg", y el tipo se imaginará que será "octet".
2. El servidor responde con un ACK, con numero de bloque 0.
3. El cliente luego de recibir el ACK, envía el bloque #1 de DATA con 512 bytes.
4. El servidor recibe los datos y envía un ACK del bloque #1.
5. El cliente recibe el ACK y envía el siguiente bloque #1 de DATA con 512 bytes (ya van 1024 enviados).
6. El servidor recibe los datos y envía un ACK del bloque #2.
7. El cliente recibe el ACK y envía el bloque #3 de DATA con 512 bytes (ya van 1536).
8. El servidor recibe los datos y envía un ACK del bloque #3
9. El cliente recibe el ACK y envía los datos restantes, por lo que envía el bloque #4 con los bytes restantes, 54 bytes.
10. El servidor recibe los datos, se da cuenta que el número de bytes es menor a 512, por lo que sabe que es el bloque final, y envía el ACK del bloque #4.
11. El cliente recibe el ACK y termina el proceso.

Así es la interacción entre cliente servidor en un proceso sin interrupciones o errores, si se desea leer y no escribir, la interacción es exactamente igual, solo que ahora es el servidor quien envía los mensajes de DATA, y el cliente los de ACK, como se puede ver en el ejemplo a continuación sacado de [5].

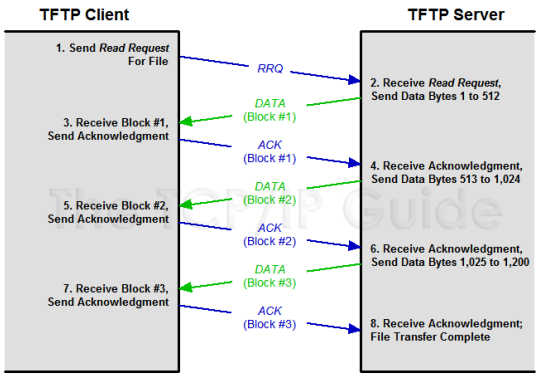


Figura 5. Proceso lectura TFTP

Si bien los escenarios descritos anteriormente son los ideales tanto de escritura como de lectura, no siempre son así, se contemplan 2 escenarios adicionales.

1. El primer escenario distinto es en el que alguno de los mensajes se pierde, no es recibido por la contraparte, el protocolo TFTP como ya se menciono es Stop and Wait, por lo que, una vez cualquiera de las partes envía un mensaje, espera un tiempo para recibir el ACK, si sucede el timeout y no ha recibido ese ACK, procede a reenviar el mismo mensaje, hasta que no se reciba el ACK no se continua con el proceso, y si el mensaje que se está esperando es uno de DATA, en dicho caso el mensaje que se habría perdido sería el de ACK, el procedimiento seria el mismo, la parte que está esperando la DATA envía el ACK, y si sucede el timeout y no ha recibido la DATA, reenviara el ACK.
2. El segundo escenario es en el que sucede un error, en este caso, simplemente, sea cual sea el error, se envía el mensaje a la contraparte, con el código del error, el mensaje, y esto finaliza la comunicación en ambas partes.

Si se da el caso donde el tamaño del archivo es divisible por 512, es decir, que hasta el último bloque tiene un tamaño de 512 bytes, en ese caso, igualmente se debe enviar un mensaje de tipo DATA adicional con 0 bytes de DATA para indicar que es el final, este último paquete conserva el mismo número de bloque que el anterior.

IV. PRUEBAS PROTOCOLO TFTP

Una vez se comprende a profundidad en teoría el funcionamiento del protocolo TFTP, se puede continuar a analizarlo en la práctica, para esto se realizarán operaciones TFTP en GNS3, hacienda uso de un router como cliente, y un servidor TFTP incluido en la appliance ToolBox [6].

Se guardará la configuración del router en el servidor TFTP.

```
R1#copy startup-config tftp
Address or name of remote host []? 192.168.0.2
Destination filename [r1-confg]? myExample
!!
1082 bytes copied in 0.084 secs (12881 bytes/sec)
```

Figura 6. Comandos escritura TFTP

Se escribe el archivo con nombre “myExample”, con esto se puede capturar el intercambio de mensajes.

192.168.0.1	192.168.0.2	TFTP	60	Write Request, File: myExample,
192.168.0.2	192.168.0.1	TFTP	46	Acknowledgement, Block: 0
192.168.0.1	192.168.0.2	TFTP	558	Data Packet, Block: 1
192.168.0.2	192.168.0.1	TFTP	46	Acknowledgement, Block: 1
192.168.0.1	192.168.0.2	TFTP	558	Data Packet, Block: 2
192.168.0.2	192.168.0.1	TFTP	46	Acknowledgement, Block: 2
192.168.0.1	192.168.0.2	TFTP	104	Data Packet, Block: 3 (last)
192.168.0.2	192.168.0.1	TFTP	46	Acknowledgement, Block: 3

Figura 7. Intercambio mensajes

En la figura 7 se puede ver el intercambio de mensajes, la dirección del router es la 192.168.0.1, y la del servidor es la 192.168.0.2. Se envía un WRQ por parte del router, luego el servidor envía el ACK de ese WRQ, luego el router envía el primer mensaje de DATA, si se sabe que se envían 512 bytes de DATA, y el tamaño del mensaje es de 558, se entiende que los bytes adicionales son 46.

El servidor envía el ACK del bloque 1, luego el router envía el bloque 2 con el mismo tamaño 558, y el servidor responde con el ACK, finalmente el router envía un mensaje de DATA con solo 104 bytes, por ende, se entiende que es el último, a lo que el servidor responde con el ultimo ACK.

Ahora se puede examinar a fondo cada uno de esos mensajes, comenzando por el de WRQ.

```
0000  56 c8 5d ef 4a 21 c0 01  05 c8 00 00 08 00 45 00
0010  00 2e 00 00 00 00 ff 11  3a 6b c0 a8 00 01 c0 a8
0020  00 02 cd 01 00 45 00 1a  6f 88 00 02 6d 79 45 78
0030  61 6d 70 6c 65 00 6f 63  74 65 74 00
```

Figura 8. Mensaje WRQ

En la figura 8 se puede ver el mensaje en hexadecimal, en azul están los 2 bytes del Opcode, como es un mensaje WRQ el Opcode es el 2, marcados de rojo están los bytes que indican el nombre del archivo, con un byte final en 0, y enmarcado en verde están los bytes que indican el modo, en este caso octet.

Wireshark traduce automáticamente estas cadenas de caracteres, sin embargo, para este ejemplo se usa un conversor online de hexadecimal a código ASCII para demostración (se puede confirmar manualmente con la tabla ASCII, sin embargo, para efectos de practicidad se usará la página web) [7]

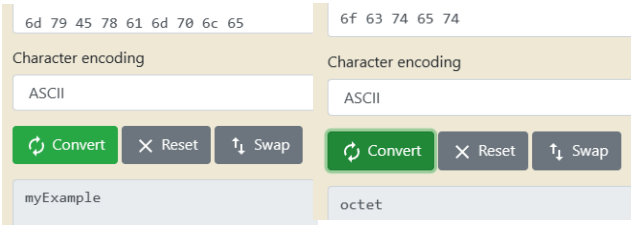


Figura 9. Traducción cadenas string

En la figura 9 se puede corroborar el envío del nombre de archivo y el modo en formato ASCII

```
0000  c0 01 05 c8 00 00 56 c8  5d ef 4a 21 08 00 45 00
0010  00 20 7c da 00 00 40 11  7c 9f c0 a8 00 02 c0 a8
0020  00 01 b3 ad cd 01 00 0c  fd cb 00 04 00 03
```

Figura 10. Mensaje ACK

En la figura 10 se puede ver el mensaje ACK, solo es los últimos 4 bytes, siendo los 2 azules los que indican el Opcode, en este caso el 4, y los otros 2 indicando el número de bloque, en este ejemplo el número 3.

```
00 02 cd 01 b3 ad 02 0c  46 3b 00 03 00 02 6f 0a   ........ F;....o·
21 0a 69 6e 74 65 72 66  61 63 65 20 53 65 72 69   !·interf ace Seri
61 6c 31 2f 30 0a 20 6e  6f 20 69 70 20 61 64 64   al1/0· n o ip add
72 65 73 73 0a 20 73 68  75 74 64 6f 77 6e 0a 20   ress· sh utdown·
73 65 72 69 61 6c 20 72  65 73 74 61 72 74 2d 64   serial r estart-d
65 6c 61 79 20 30 0a 21  0a 69 6e 74 65 72 66 61   elay 0·! ·interfa
63 65 20 53 65 72 69 61  6c 31 2f 31 0a 20 6e 6f   ce Seria l1/1· no
20 69 70 20 61 64 64 72  65 73 73 0a 20 73 68 75    ip addr ess· shu
74 64 6f 77 6e 0a 20 73  65 72 69 61 6c 20 72 65   tdown· s erial re
73 74 61 72 74 2d 64 65  6c 61 79 20 30 0a 21 0a   start-de lay 0·!·
69 6e 74 65 72 66 61 63  65 20 53 65 72 69 61 6c   interfac e Serial
31 2f 32 0a 20 6e 6f 20  69 70 20 61 64 64 72 65   1/2· no  ip addre
73 73 0a 20 73 68 75 74  64 6f 77 6e 0a 20 73 65   ss· shut down· se
72 69 61 6c 20 72 65 73  74 61 72 74 2d 64 65 6c   rial res tart-del
61 79 20 30 0a 21 0a 69  6e 74 65 72 66 61 63 65   ay 0·!·i nterface
```

Figura 11. Mensaje DATA

En la figura 11 se puede ver un mensaje de tipo DATA, en este caso solo tiene 3 partes, iniciando por la marcada en azul que es el Opcode, el número 3, seguido del número de bloque, para este ejemplo el número 2, y todo lo demás son los bytes de datos, para este caso se incluye la traducción que realiza Wireshark a la derecha, se puede apreciar que efectivamente se está enviando la configuración del router.

Ahora se realizará una lectura para comprobar la diferencia en la comunicación.

```
R1#copy tftp: flash:
Address or name of remote host [192.168.0.2]?
Source filename [saved]? myExample
Destination filename [myExample]? saved
Accessing tftp://192.168.0.2/myExample...
Erase flash: before copying? [confirm]n
Loading myExample from 192.168.0.2 (via FastEthernet0/0): !
[OK - 1082 bytes]
```

Figura 12. Lectura de TFTP

En la figura 12 se puede ver los comandos para leer el archivo guardado en el servidor TFTP.

Figura 13. Intercambio mensajes lectura

En la figura 13 se puede ver el intercambio de mensajes, casi idéntico al de escritura, solo que esta vez es el servidor quien envía los mensajes DATA y el cliente los ACK.

No se realiza análisis de los mensajes ya que incluso se puede ver que los tamaños son los mismos, por lo que son iguales a los de escritura. Finalmente se solicitará la lectura de un archivo inexistente, esto para comprobar los mensajes de tipo ERROR.

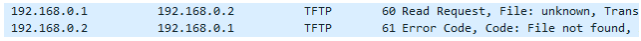


Figura 13. Intercambio error

En la figura 13 se puede ver el intercambio de mensajes, el primero es la solicitud de lectura del archivo “unknown”, el cual es inexistente, por lo que el servidor responde con un mensaje de ERROR, y esto finaliza la comunicación.

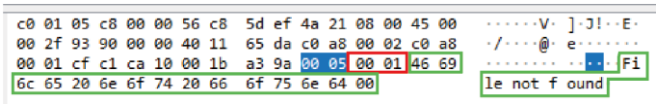


Figura 14. Mensaje ERROR.

En la figura 14 se puede analizar el mensaje de error, marcado de azul está el Opcode, el número 5, seguido por el código de error de 2 bytes, el cual es el 1, como se puede confirmar en la Tabla 2, el cual es el código de error para un archivo no encontrado. Marcado de verde esta la cadena de string cuya terminación está indicada por un byte en 0, se puede ver la traducción de esa cadena a la derecha, la cual efectivamente es “File not found”.

V. PLANEACION DE PRUEBAS

Una vez se conoce el funcionamiento del protocolo TFTP en la práctica, se puede realizar la planeación para las pruebas sobre las cuales se comprobará el funcionamiento del servidor TFTP a realizar.

Para la comprobación del funcionamiento se escoge el uso de Virtual box como herramienta para tener distintas máquinas virtuales con distintos sistemas operativos, en donde estas máquinas harán parte de la red interna, por medio de esta red interna podrán comunicarse con el host local, el cual estará corriendo el servidor TFTP realizado en java.

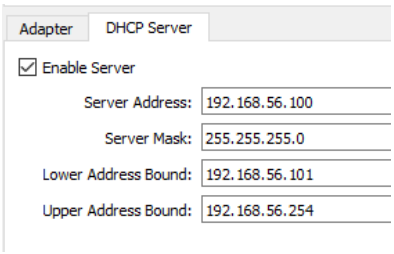


Figura 15. Configuración Adaptador

En Virtual box se configura una Host-only network, la cual es una red que permitirá la comunicación solamente con el host local, como se puede ver en la figura 15, la dirección IP asignada al host local es la 192.168.56.1 / 24.

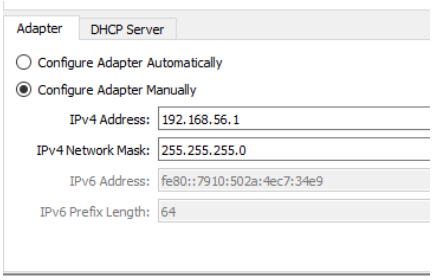


Figura 16. Configuración DHCP

En la figura 16 se puede ver la configuración del servidor DHCP para la red Host-only, como se puede ver las direcciones IP disponibles son de la 192.168.56.101 /24 a la 192.168.56.254 /24, por lo que, las máquinas virtuales que se usarán para las pruebas obtendrán una dirección IPv4 dentro dicho rango:

- Direccion IPv4 TFTP server: 192.168.56.1 / 24
- Direcciones IPv4 TFTP client: 192.168.56.101 /24 -192.168.56.254 /24

Teniendo claro las direcciones IP a usar, se debe comprobar el funcionamiento de las máquinas virtuales, se planea probar con una máquina virtual que corra Ubuntu, y otra que corra Windows, estas máquinas virtuales se deben crear en Virtual box con sus respectivas imágenes disponibles en [8,9].

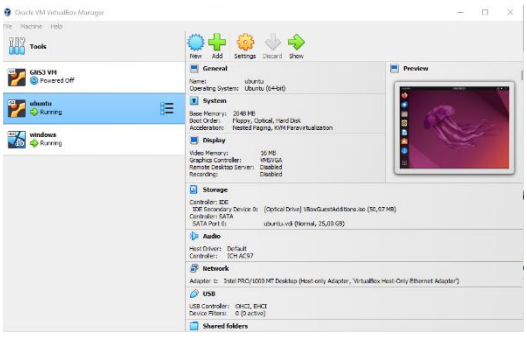


Figura 17. Virtual box con VM’S

En la figura 17 se puede ver Virtual box con la máquina virtual corriendo el sistema operativo Ubuntu (distribución de

Linux), y la otra máquina virtual corriendo el sistema operativo Windows.

Dentro de cada máquina virtual se deben realizar las configuraciones necesarias para obtener el cliente TFTP, en el caso de la máquina virtual Ubuntu, se debe instalar dicho cliente, ya que esto requiere conectividad con el exterior.

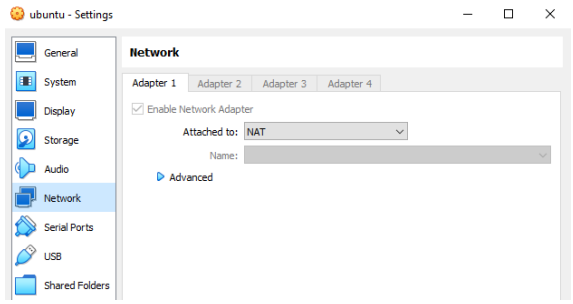


Figura 18. Red NAT

Debido a que para instalar el cliente TFTP se requiere conexión a internet se debe configurar inicialmente la red como NAT, con esto la máquina virtual puede acceder al internet.

Cabe resaltar que el uso de la red Host-only se requiere debido a que esas máquinas virtuales que corren en Virtual box están corriendo en un host local que es a su vez otra máquina virtual con dirección IP estática, así mismo el host local (máquina virtual), debido a que tiene una dirección IP estática no tiene un servidor DHCP al que pedir direcciones IP. Sin embargo, si se quisiese se podría hacer uso de la red Bridge Adapter, esta permite que las máquinas virtuales dentro de Virtual box pidan una dirección IP al mismo servidor DHCP que tenga el host local, en el caso de un host que no tenga dirección IP estática y que si tenga un servidor DHCP al que pedir direcciones, este Bridge adapter funcionaria ya que las máquinas virtuales podrían recibir una dirección IPv4 por parte de ese servidor DHCP, y con esa misma dirección IP podrían tanto comunicarse con el host local como con el exterior (se realizaron pruebas en maquina personal).

Para descargar el cliente TFTP dentro de la máquina virtual Ubuntu se debe abrir la terminal, y dentro de la misma correr los siguientes comandos.

- sudo apt update
- sudo apt install tftp-hpa

El primer comando se usa para actualizar las listas de paquetes con las últimas versiones, no es necesario, pero es recomendable, y el segundo comando es el que instala el cliente TFTP, una vez se tiene el cliente TFTP instalado se pueden leer o escribir archivos tan fácil como usando los siguientes comandos.

- tftp tftp_server_address
- put filename
- get filename

Antes de poder usar las operaciones TFTP es importante cambiar la configuración de red a la Host-only (esto para las condiciones propias del proyecto y donde se planea ejecutar como se explicó anteriormente).

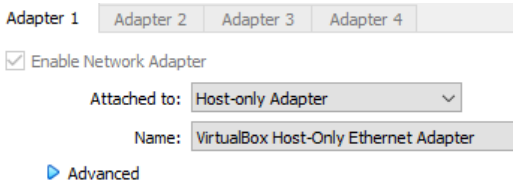


Figura 19. Red Host-only

En la figura 19 se puede ver como se selecciona la red Host-only, una vez hecho esto, ya se podrá tener conectividad con el host local, esto se puede comprobar por medio de pings, antes de realizarlos es importante verificar que se habiliten los paquetes ICMPv4 en el firewall.

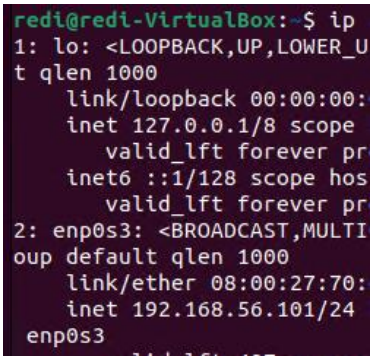


Figura 20. Dirección IP recibida

En la figura 20 se puede comprobar que ahora la máquina virtual recibió una dirección IPv4 dentro del rango que se analizó anteriormente, en este caso la 192.168.56.101 /24.

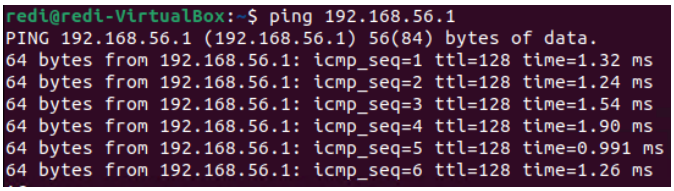


Figura 21. Ping con host local

En la figura 21 se puede ver como el ping realizado desde la máquina virtual hacia el host local es exitoso.

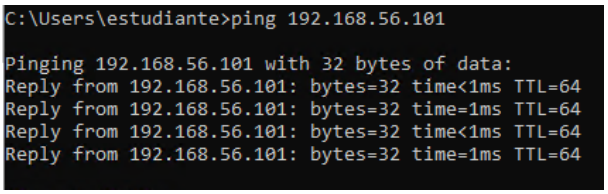


Figura 22. Ping con máquina virtual

En la figura 22 se comprueba que desde el host local el ping hacia la máquina virtual también es exitoso, por lo que se comprueba que hay conectividad, lo cual es fundamental para

el funcionamiento del servidor TFTP que correrá en el host local.

```
redi@redi-VirtualBox:~$ tftp 192.168.56.1
tftp> get file.txt
```

Figura 23. Comando TFTP

En la figura 23 se puede ver un ejemplo de cómo se usarían los comandos para usar el servidor TFTP desde el cliente, primero se ingresa la dirección IP del servidor, y luego el tipo de operación, en este caso, un get (leer) un archivo con nombre "file.txt". Al leer el archivo este debería aparecer en la carpeta "home/" dentro de la máquina virtual, así mismo, en esa misma carpeta se buscarán los archivos cuando se realice una operación put (escribir), a menos que se especifique el directorio. De esta forma se comprobará que el servidor TFTP funcione, cuando al realizar una operación get de un archivo existente en el host local desde la máquina virtual, este archivo aparezca en la carpeta home/ de la máquina virtual, y que al escribir un archivo al servidor TFTP, este aparezca en la carpeta especificada dentro del código en el host local.

Para comprobar el funcionamiento de las múltiples peticiones a la vez, se usaran las 2 máquinas virtuales, que también usan distintos sistemas operativos, con esto también se comprobara el funcionamiento para cualquier tipo de cliente TFTP independiente del sistema operativo, teniendo las dos máquinas virtuales, en donde al estar las dos corriendo una tendrá la dirección 192.168.56.101 /24 y la otra la 192.168.56.102 /24, se realizaran peticiones ya sean de leer o escribir al mismo tiempo, ambas peticiones deberían ser exitosas, es decir, si era de leer, el archivo debe aparecer en la maquine virtual en cuestión, y si era de escribir, el archivo debe aparecer en el host local.

Cabe resaltar que no solo deben poderse realizar las dos peticiones a la vez, sino también procesarse a la vez, esto es obligatorio ya que, si ambas peticiones se realizan a la vez, es probable que uno de los clientes no pueda esperar hasta que el proceso se termine con el otro, los paquetes entrantes y salientes de ambas peticiones se deben procesar concurrentemente, esto se realizara por medio de lógica, administración, e hilos, se imprimirá en consola cada paso que realiza el servidor para ver la interacción con ambas peticiones a la vez.

VI. PLANEACION SOLUCION

Finalmente, solo queda la planeación del servidor TFTP en si, es decir, el código del mismo, la lógica que usara, y como lograra cumplir a cabalidad todo lo propuesto anteriormente incluyendo el procesamiento simultaneo, para esto, en primera instancia, se diseña la siguiente estructura representada en el siguiente diagrama de clases, es importante aclarar que esto solo es un boceto realizado con lo que se puede analizar sin escribir código, es decir, el producto final puede variar, tanto en métodos, atributos, e incluso en clases.

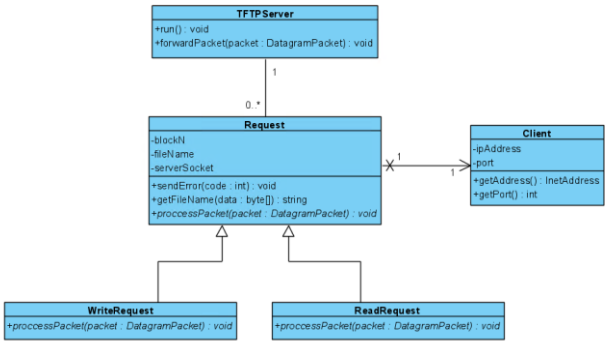


Figura 24. Diagrama de clases.

En la figura 24 se puede ver el diagrama de clases diseñado, en el mismo se puede ver lo siguiente.

- ***Clase TFTPServer:*** De esta clase solo habrá 1 instancia, es la correspondiente al servidor, y tiene un método run(), que se correrá desde el main, así mismo tiene un segundo método llamado forwardPacket, esta clase será la única que reciba paquetes entrantes, y debido a que se busca un diseño que maneje múltiples peticiones a la vez, se debe tener en cuenta que podrán entrar paquetes al mismo puerto (69) pero que corresponden a usuarios distintos, el método forwardPacket es el que se encargara de enviar ese paquete a la request de la que hace parte para que esta lo procese.
- ***Clase Abastracta Request***: Esta será una clase abstracta, ya que, no habrán instancias de esta clase, sin embargo, de esta clase heredaran los 2 tipos de requests posibles, la instancia de TFTPServer tendrá una lista de esta clase Request, en esa lista estarán los requests que se encuentran activos (procesándose), por eso la cardinalidad 1- 0..*, es bidireccional, ya que se planea que desde la Request se pueda informar a la instancia de TFTPServer cuando el proceso de la request en cuestión haya finalizado, para que el TFTPServer se encargue de eliminar ese elemento Request, esta clase declara atributos en común para los 2 tipos de requests, como el número de bloque, el nombre del archivo y el serverSocket (se enviara por medio del constructor). Tiene 3 métodos, 1 para enviar error, este método es común para ambos tipos de requests por lo que se implementa en esta clase, otro método para obtener el nombre del archivo de la request, también se implementa acá ya que es común, y el tercero es el método de procesamiento de la request que requiere del paquete recibido, este método es abstracto ya que depende del tipo de request. Finalmente, esta clase tiene una instancia de la clase Cliente, clase que solo se encarga de guardar

los datos del cliente como la dirección IP y el puerto

- ***Clase Cliente***: Como ya se mencionó esta clase solo se encarga de guardar los datos del cliente de dicha request.
- ***Clase WriteRequest***: Hereda de la clase Request y se encarga de implementar el método de procesamiento de acuerdo a la operación de escritura.
- ***Clase ReadRequest***: Hereda de la clase Request y se encarga de implementar el método de procesamiento de acuerdo a la operación de lectura.

El diseño planteado debe servir como estructura principal o base para la solución final que pondrá en funcionamiento total al servidor TFTP.

VII. IMPLEMENTACION

Al momento de realizar la implementación de lo planeado para el servidor TFTP se encontró que el modelo propuesto anteriormente requerirá ciertas modificaciones, siendo las más básicas la inclusión de nuevos métodos, métodos que sirvan de apoyo para la conversión de números binarios a decimales y viceversa, teniendo en cuenta que dos bytes separados representan 1 solo numero decimal (como el número de bloque).

Sin embargo, el cambio mas significativo fue el cambio de la estructuración, básicamente se vio la necesidad de agregar nuevas clases y cambiar las clases ya existentes a clases abstractas, a continuación se muestra el modelo final implementado.

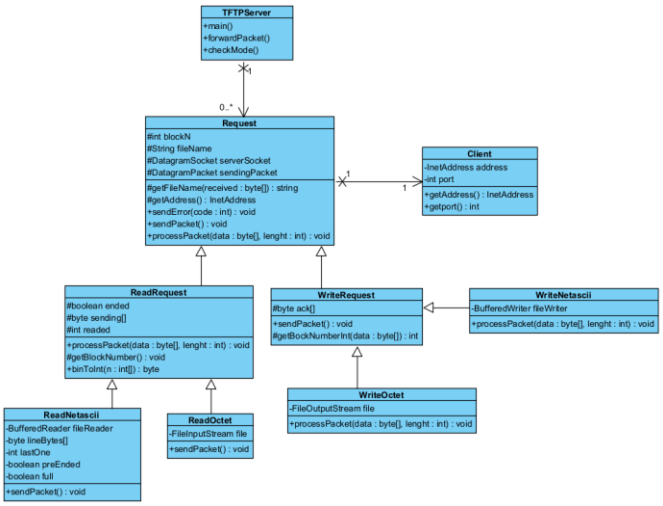


Figura 25. Modelo de clases

En la figura 25 se puede ver el modelo de clases usado para el servidor TFTP. Los cambios realizados fueron.

- ***Clase ReadRequest:*** Esta clase en esta ocasión se vuelve abstracta al igual que la clase Request, esto ya que, se decidió que para implementar los dos tipos de lectura, Octet y Netascii, se requiere crear una base que proporcione ciertos métodos en común entre los dos tipos de lectura, pero se deje el método de sendPacket() vacío para ser implementado por cada tipo de read Request, así mismo cada tipo de read Request requiere de variables propias.
- ***Clase WriteRequest:*** Por la misma razón que el ítem anterior esta clase se vuelve abstracta para proporcionar una base a los dos tipos de Write Request, pero aun así dejar el método de processPacket() a disposición de cada tipo de Write Request.
- ***Clases Read Octet y Netascii:*** Se crean dos clases que heredan de ReadRequest, implementan la lógica del método sendPacket() para cada tipo de request.
- ***Clases Write Octet y Netascii:*** Se crean dos clases que heredan de WriteRequest, implementan la lógica del método processPacket() para cada tipo de request.

El servidor implementa el multiprocesamiento simultaneo por medio de una combinación de lógica secuencial y multi hilo, para esto conserva en una lista todas las requests activas, es necesario que el servidor tenga la clase central TFTPServer que se encarga de recibir todos los paquetes, y luego los distribuye a la request en cuestión para que esta lo procese de la forma en la que se requiera.

Para conseguir lo anterior el proceso es el siguiente.

A. TFTPServer recibe un paquete
B. TFTPServer crea un nuevo Thread para que en este se encarge de procesar dicho paquete.
C. El TFTPServer vuelve a escuchar y recibir paquetes en el Thread principal mientras que el Thread secundario procesa el paquete ya recibido. Si se recibe un nuevo paquete, este tendrá su propio Thread, de esta forma habrá un Thread para cada paquete que se este procesando, mientras que el TFTPServer no tendrá que dejar de escuchar hasta que se terminen de procesar los paquetes recibidos.
D. El Thread del paquete verifica el tipo de paquete, si el paquete tiene un Opcode 1 o 2, se crea la nueva Request para este cliente, en la nueva Request se vincula el cliente con la dirección y el puerto usado, de acuerdo al tipo de Request se crea el objeto de ese tipo, ya sea Read Netascii, Write Octet, etc.
E. Si el paquete recibido tiene cualquier otro Opcode, la dirección de dicho paquete se compara con la dirección del cliente de cada uno de los Requests que ya se tienen activos, al encontrar la Request, a esta Request se le invoca al método processPacket(), para esto se uso el polimorfismo, ya que, se pueden recorrer todos los Requests como tipo Request, y llamar a su método processPacket, esto desde el TFTPServer sin tener que

verificar que tipo de Request es, el Request en cuestión realizara la lógica del processPacket() dependiendo del tipo de Request concreto.

F. La comunicación continua de esa manera, pero en el momento en el que el Request identifica un error o termina el proceso exitosamente, este Request se remueve el mismo de la lista del TFTPServer y cierra los archivos usados, el Garbage Collector se encargara de eliminar esta instancia que ya no está referenciada.

VIII. PRUEBAS.

Para realizar las pruebas se usará la máquina virtual en VirtualBox de Ubuntu, se comenzará con una prueba de lectura de un archivo.

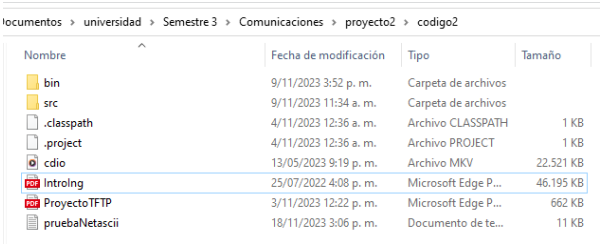


Figura 26. Archivos de prueba

En la figura 26 se puede ver los archivos que se usaran para las pruebas, primero se leerá el documento pdf llamado “ProyectoTFTP.pdf”, este es el documento de la primera entrega del presente proyecto. La lectura se realizará en modo Octet.

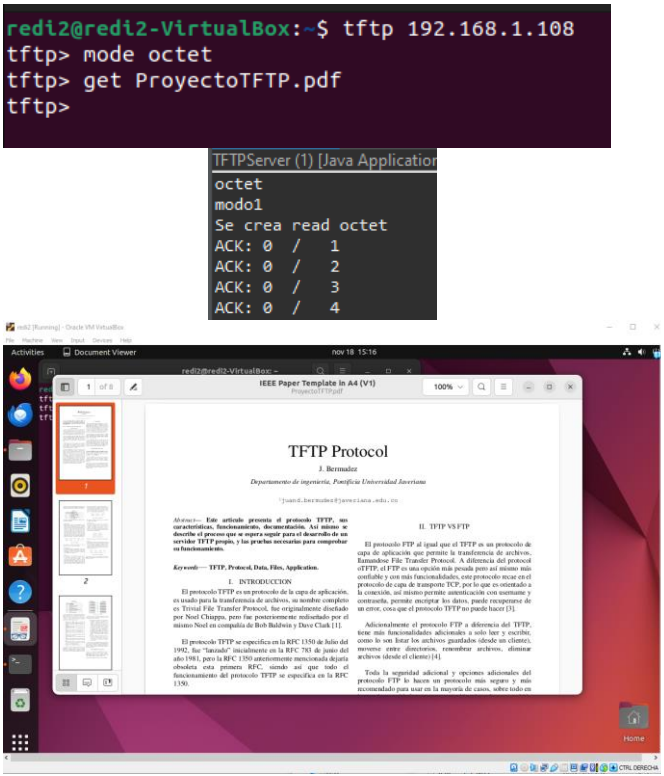


Figura 27. Primera prueba TFTP

En las 3 imágenes anteriores se ve el éxito en la operación get, del modo Octet, en la primera imagen se puede ver el proceso desde la maquina virtual en la que se pide el archivo pdf, en la segunda imagen se ve como desde el servidor se identifica el modo de la request, y como comienzan a recibirse los ACKS.

A continuación, se comprobará la operación PUT (escritura) del modo Octet, esto al escribir este mismo archivo recibido en el servidor, por lo que primero, se intentara enviar aun siendo que ya se tiene el archivo, esto para comprobar el error de File Already Exists.

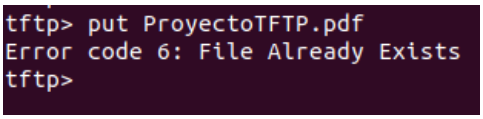


Figura 28. Error File Already Exists

Ahora se eliminará el archivo para poder ser escrito.

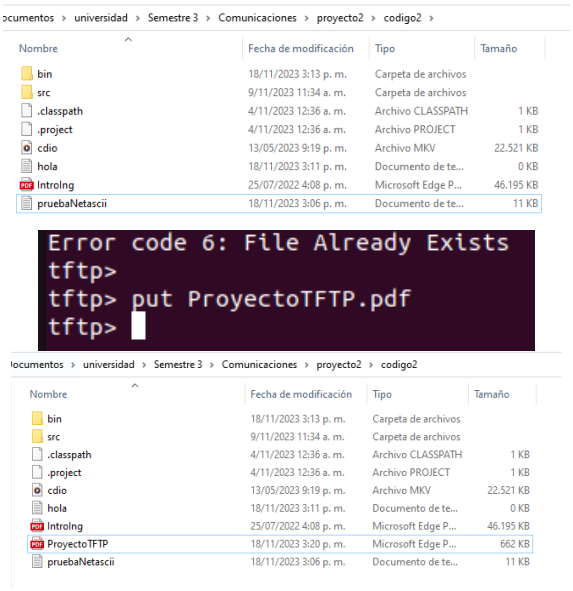
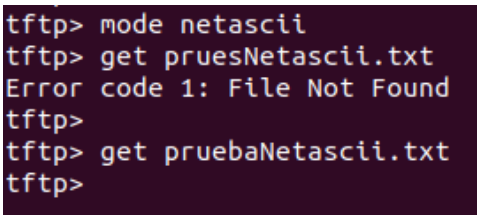


Figura 29. Escritura TFTP

Después de comprobar la escritura en modo Octet, se realizaran las pruebas en modo Netascii, para esto se usara el archivo “pruebaNetascii.txt” se intentara leer, pero antes, se escribirá el nombre incorrectamente, para recibir y comprobar el error “File Not Found”.



```
TFTPServer (1) [Java Application] C:\Users\Usuario
ACK RECIBIDO: 0-1  /192.168.1.101
ACK RECIBIDO: 0-2  /192.168.1.101
ACK RECIBIDO: 0-3  /192.168.1.101
ACK RECIBIDO: 0-4  /192.168.1.101
ACK RECIBIDO: 0-5  /192.168.1.101
ACK RECIBIDO: 0-6  /192.168.1.101
ACK RECIBIDO: 0-7  /192.168.1.101
ACK RECIBIDO: 0-8  /192.168.1.101
ACK RECIBIDO: 0-9  /192.168.1.101
ACK RECIBIDO: 0-10  /192.168.1.101
ACK RECIBIDO: 0-11  /192.168.1.101
ACK RECIBIDO: 0-12  /192.168.1.101
ACK RECIBIDO: 0-13  /192.168.1.101
ACK RECIBIDO: 0-14  /192.168.1.101
ACK RECIBIDO: 0-15  /192.168.1.101
ACK RECIBIDO: 0-16  /192.168.1.101
ACK RECIBIDO: 0-17  /192.168.1.101
ACK RECIBIDO: 0-18  /192.168.1.101
ACK RECIBIDO: 0-19  /192.168.1.101
ACK RECIBIDO: 0-20  /192.168.1.101
ACK RECIBIDO: 0-21  /192.168.1.101
Se termina request /192.168.1.101
```

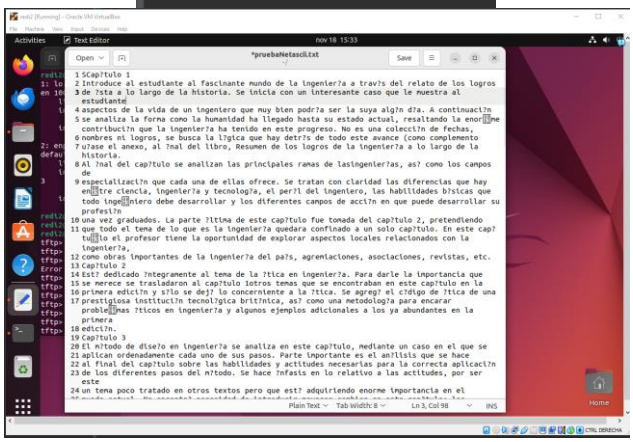


Figura 30. Prueba Lectura Netascii

En la figura 30 se puede ver el comando de lectura desde la maquina virtual, como el servidor recibe los ACK de la maquina virtual, y finalmente el archivo de texto en la maquina virtual, ahora se escribirá el archivo para probar la operación de escritura, para esto se eliminará el archivo del servidor TFTP.

```
tftp>
tftp> put pruebaNetascii.txt
tftp>
ACK ENVIADO: 0 /192.168.1.101
ACK ENVIADO: 1 /192.168.1.101
ACK ENVIADO: 2 /192.168.1.101
ACK ENVIADO: 3 /192.168.1.101
ACK ENVIADO: 4 /192.168.1.101
ACK ENVIADO: 5 /192.168.1.101
ACK ENVIADO: 6 /192.168.1.101
ACK ENVIADO: 7 /192.168.1.101
ACK ENVIADO: 8 /192.168.1.101
ACK ENVIADO: 9 /192.168.1.101
ACK ENVIADO: 10 /192.168.1.101
ACK ENVIADO: 11 /192.168.1.101
ACK ENVIADO: 12 /192.168.1.101
ACK ENVIADO: 13 /192.168.1.101
ACK ENVIADO: 14 /192.168.1.101
ACK ENVIADO: 15 /192.168.1.101
ACK ENVIADO: 16 /192.168.1.101
ACK ENVIADO: 17 /192.168.1.101
ACK ENVIADO: 18 /192.168.1.101
ACK ENVIADO: 19 /192.168.1.101
ACK ENVIADO: 20 /192.168.1.101
ACK ENVIADO: 21 /192.168.1.101
Se termina request /192.168.1.101
```

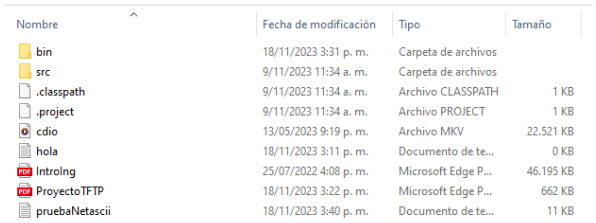
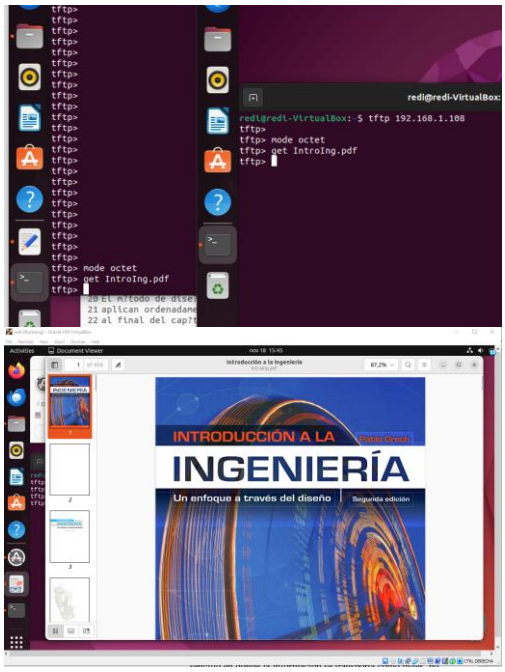


Figura 31. Prueba escritura Netascii

En la figura 3 se puede ver el proceso exitoso de escritura, incluyendo la petición, los ACKS enviados por el servidor, y finalmente el archivo nuevamente en el servidor con el mismo tamaño.

Finalmente se probará el procesamiento simultaneo, para esto se realizará la lectura del archivo “IntroIng.pdf” desde dos maquinas virtuales Ubuntu, se escoge este archivo ya que tiene un peso que requerirá segundos en enviarse completamente, y esto da tiempo a iniciar el proceso en ambas máquinas virtuales.




```
ACK RECIBIDO: 29-66  /192.168.1.102
ACK RECIBIDO: 137-71  /192.168.1.101
ACK RECIBIDO: 29-67  /192.168.1.102
ACK RECIBIDO: 137-72  /192.168.1.101
ACK RECIBIDO: 29-68  /192.168.1.102
ACK RECIBIDO: 137-73  /192.168.1.101
ACK RECIBIDO: 29-69  /192.168.1.102
ACK RECIBIDO: 137-74  /192.168.1.101
ACK RECIBIDO: 29-70  /192.168.1.102
ACK RECIBIDO: 137-75  /192.168.1.101
ACK RECIBIDO: 29-71  /192.168.1.102
```

Figura 32. Procesamiento simultaneo

En la figura 32 se puede ver como se realizan las requests de lectura desde dos maquinas virtuales a la vez, también se ve como desde la máquina virtual se recibió el pdf en perfectas condiciones, y finalmente, se puede ver desde el servidor como

se reciben ACKS de ambas direcciones, es decir, ambos clientes a la misma vez, es decir, intercalado, básicamente, procesamiento simultaneo de ambas requests.

Con esto se comprueban todas las funcionalidades del servidor TFTP, de lectura y escritura Octet, Netascii, y, por último, el procesamiento simultaneo.

IX. CONCLUSIONES

El protocolo TFTP es un protocolo eficiente de transferencia de archivos, sin embargo bastante inseguro, es útil en casos en donde se priorice la velocidad y la seguridad no sea inconveniente, este protocolo es fácil de implementar a diferencia de otros protocolos, esto debido a su funcionamiento sencillo en donde la información se transporta como bytes, no se requiere establecer conexión como con TCP, y en donde no hay mayores procesos a realizar más que él envió y recibimiento de datos, datos que se envían al "desnudo", sin proceso de encriptación ni autorización.

Para comprobar el funcionamiento de un servidor TFTP es necesario tener una red ya sea real o simulada, en la que se pueda tener una máquina que sea el cliente TFTP, y otra que corra el servidor TFTP realizado, al usar un cliente TFTP ya existente, es decir, no desarrollarlo por autoría propia, se garantiza que, si la lectura y escritura de archivos es exitosa, esto implica que el servidor TFTP fue exitosamente desarrollado cumpliendo a cabalidad todos los requerimientos y normas establecidas en la RFC 1350, esto puesto que el cliente TFTP ya existente a utilizar estará programado para seguir así mismo las reglas a cabalidad de la RFC 1350, por lo que solo podrá funcionar correctamente si la contraparte en el servidor cumple dichas reglas.

A la hora de desarrollar un servidor TFTP es importante tener en cuenta el lenguaje de programación que se usara, esto más que por temas de sintaxis o comodidad, es en pro de la disponibilidad de recursos, ya sea bibliotecas existentes que tengan clases que implementen funcionalidades importantes para la comunicación en la red, o también la disponibilidad de información en internet sobre cómo usar dichas librerías y clases para el correcto funcionamiento de una aplicación de red, es por esto que para el caso del proyecto actual se decide usar el lenguaje JAVA.

REFERENCIAS

[1] K. Sollins (1992) The tftp protocol (revisión 2) [Online]. Available: https://datat racker .ietf.org/doc/html/rfc1350

[2] Brisman (2014) How tftp Works? [Online]. Available: https://computernetworkingsimplified.wordpress.com/2014/01/26/tftp-works/

[3] Mks075 (2023) Difference between FTP and TFTP [Online]. Available: https://www.geeksforgeeks.org/difference-between-ftp-and-tftp/

[4] M. Horan (2023) How does an FTP server work and what are its benefits? [Online]. Available: https://www.sharetru.com/blog/how-does-an-ftp-server-work-the-benefits

[5] (2005) TFTP detailed operation and messaging [Online]. Available: http://www.tcpipguide.com/free/t_TFTPDetailedOperationandMessaging-2.htm

[6] J. Grossmann (2023) Toolbox [Online]. Available: https://www.gns3 .com/marketplace/appliances/networkers-toolkit

[7] (2023) RapidTables [Online]. Available: https://www.rapidtables.com/convert/number/hex-to-ascii.html

[8] (2023) Ubuntu. [Online]. Available: https://ubuntu.com/download/desktop

[9] (2023). Windows 10 [Online]. Available: https://www.microsoft.com/en-us/software-download/windows10