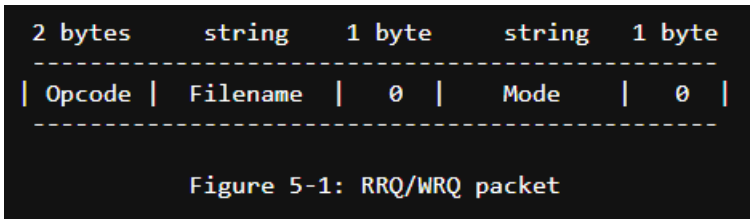
Computer Networks Coursework - 262984

Describing my protocols, the source code, and the design decisions:

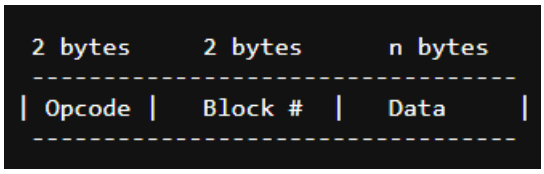
To implement the protocol correctly, I began by reading how the RFC 1350 TFTP Client–Server works to ensure that I understood how they communicate effectively as well as what the packet structures are for each type, when they are used, and how they are implemented. The packet structure of each packet that I have used in my code is the exact one from RFC 1350. I have read/write requests, acknowledgments, timeouts, error handling, and support for simultaneous file transfers as required.

Packet Structure in my code:

Read Request (RRQ)/ Write Request (WRQ)

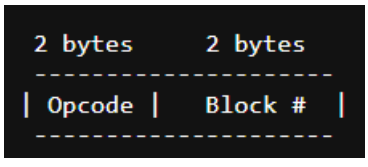


Data Packet (DATA)

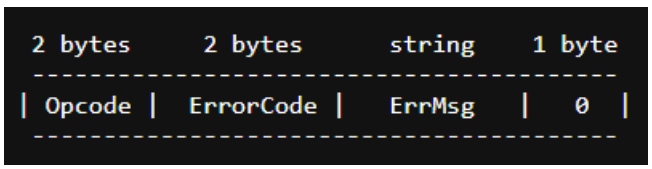


Acknowledgment Packet (ACK)

2 bytes 2 bytes



Error Packet (ERROR)



The Read Request (RRQ) and Write Request (WRQ) packets are made in the createReadRequest() and createWriteRequest(). The error (ERROR) packets are also created in their own method called sendErrorMessage(). However, the Acknowledgement (ACK) and Data (DATA) packets that are created don't have a method each instead they are created whenever it's necessary. These packets are sent as bytes as mentioned in the requirements and RFC 1350. Finally, there are methods to handle these packets as well when they are received.

Handling TFTP Requests:

Handling RRQ Packets

When a read request is sent to the server from the client, the server will unpack the request and check if it's a read request. If it's not a read request, then it will be handled accordingly. If it is a read request, then the handleReadRequest() is called. Then the server tries to find the file in the current directory, if the file is not found, an ERROR packet is sent to the client with the error code and the error message. If the file does exist, then the server immediately starts sending the DATA packets to the client.

Handling WRQ Packets

When a write request is sent to the server from the client, the server will unpack the request and check if it's a write request. If it's not a write request, then it will be handled accordingly. If it is a write request, then the handleWriteRequest() is called. Then the server sends an ACK packet back letting the client know that it's ready for file transfer.

Handling DATA and ACK Packets

DATA packets are a maximum size of 512 bytes (excluding headers) each (as mentioned in RFC 1350). The ACK packets are 4 bytes each (as mentioned in RFC 1350). When a DATA packet is sent from the sender and is received by the recipient, the data is extracted and checked its block with the recipient's block code+ 1. If it's not the same, it will be handled accordingly. If it is the same, then the recipient will send its corresponding ACK packet back. Once the sender receives the ACK, it will send the next DATA packet and this process will continue until the end of the file transfer. If the DATA packet received is less than 512 bytes (excluding headers), then it means the file transfer is done. These DATA packets are converted to files and saved in the current directory to make it easier to find where all the files are read/written to in both the server and the client.

Handling ERROR Packets

When an ERROR Packet is sent (only sent if File not Found as the requirement says), both the client and server have a sendError() which builds the ERROR packet and sends it to the recipient, and a handleError() which extracts the error code and the error message from the ERROR packet which then is outputted.

Client Implementation

In both, I have implemented a console-based menu that allows the user to execute read and write. So when you run the code, it will ask if the user wants to retrieve a file or send a file and after that, the user will input the filename. Then depending on what the inputs are, a method will be called to handle this request.

Main Differences:

UDP:

- Datagram Socket
- Datagram Packet
- Sending data is a raw sequence of bytes (as the requirement says)
- Done in octet mode (as the requirement says)
- Building Packets
- Handling Packets
- End of File transfer – Packet size < 512 (excluding headers)

TCP:

- Buffered Reader
- Buffered Writer
- Print Writer
- TCP Sockets
- No building or handling Packets
- End of File transfer – A simple “EOFT” message sent

Server Implementation:

In the UDP server, I have used Datagram Sockets to receive and send data over. Here, I didn’t need to implement a separate class just for a client handler as the server interacts directly with clients using their IP addresses and port numbers. Both handleReadRequest() and handleWriteRequest() have the IP address and client port as parameters to ensure that the server can support simultaneous file transfers to and from multiple clients (as the requirement says). When a request is sent from the client, the server checks what the request is, and then handles it accordingly using various methods in my code. The server listens on port 9000 (which can be changed to 69 to test for interoperability with Third-Party).

In the TCP server, I have used Server Sockets to accept requests from a client socket. To make it so the server can support simultaneous file transfers to and from multiple clients (as the requirement says), I have implemented a client handler that would create a new thread for each client using their client Socket. The client's request is then handled according by readFile() or writeFile() on the server side. The server listens on port 9000.

In both servers, I have implemented timeouts in case the server was listening and was not receiving anything for a specific duration of time (I have set it to 60 seconds). Furthermore, in both servers, there is some way to communicate errors such as the “File not Found” error.

Interoperability

To show that my UDP Client-Server is implemented correctly as described in RFC 1350, I used a third-party application called Tftpd64. To demonstrate that the server works correctly, I have decided to download the RFC1350 and save it as a text file called test.txt which we will be using.

Link to Tftpd64:

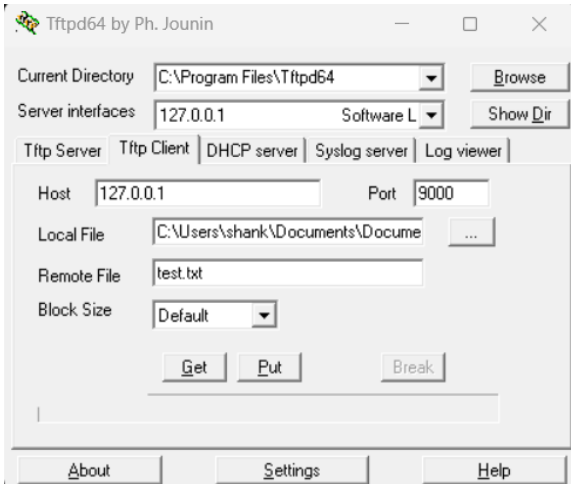
https://pjo2.github.io/tftpd64/

Third-Party Client retrieves a File from my TFTP Server:

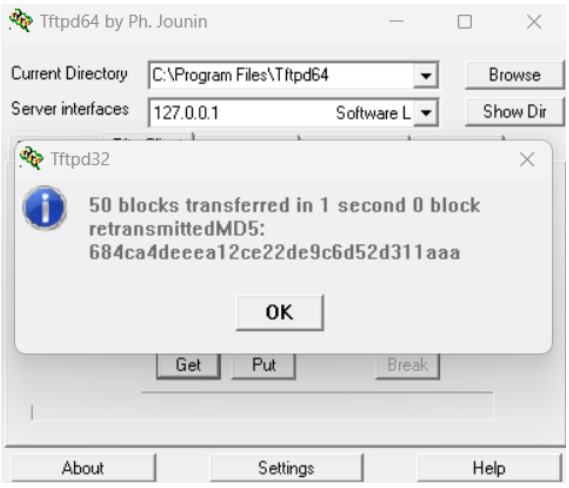
My TFTP Server runs locally on port 9000. In the screenshots below, I have ensured that the packets sent and received are traceable so it’s easier to understand. Additionally, the client should be on octet mode for this file transfer to work. To retrieve a file from the server, we use GET to retrieve the file. The Local file directory is where I want to save the file, and the Remote file is the file I send from the server (in our case, test.txt).

Tftp64:

Before File Transfer



After File Transfer



Before File Transfer

```
C:\Users\shank\.jdks\openjdk-21.0.2
Server listening on port 9000...
```

After File Transfer – Start of the output from the server.

```
C:\Users\shank\.jdks\openjdk-21.0.2\bin\java.exe "-java
Server listening on port 9000...
Received packet from 127.0.0.1: 62700
127.0.0.1: Received read request for file - test.txt
127.0.0.1: Sent for Packet: 1, Data Packet Size: 512
127.0.0.1: Acknowledgment Received For Packet 1
127.0.0.1: Sent for Packet: 2, Data Packet Size: 512
127.0.0.1: Acknowledgment Received For Packet 2
127.0.0.1: Sent for Packet: 3, Data Packet Size: 512
127.0.0.1: Acknowledgment Received For Packet 3
127.0.0.1: Sent for Packet: 4, Data Packet Size: 512
127.0.0.1: Acknowledgment Received For Packet 4
127.0.0.1: Sent for Packet: 5, Data Packet Size: 512
127.0.0.1: Acknowledgment Received For Packet 5
127.0.0.1: Sent for Packet: 6, Data Packet Size: 512
127.0.0.1: Acknowledgment Received For Packet 6
127.0.0.1: Sent for Packet: 7, Data Packet Size: 512
```

After File Transfer – End of the output from the server.

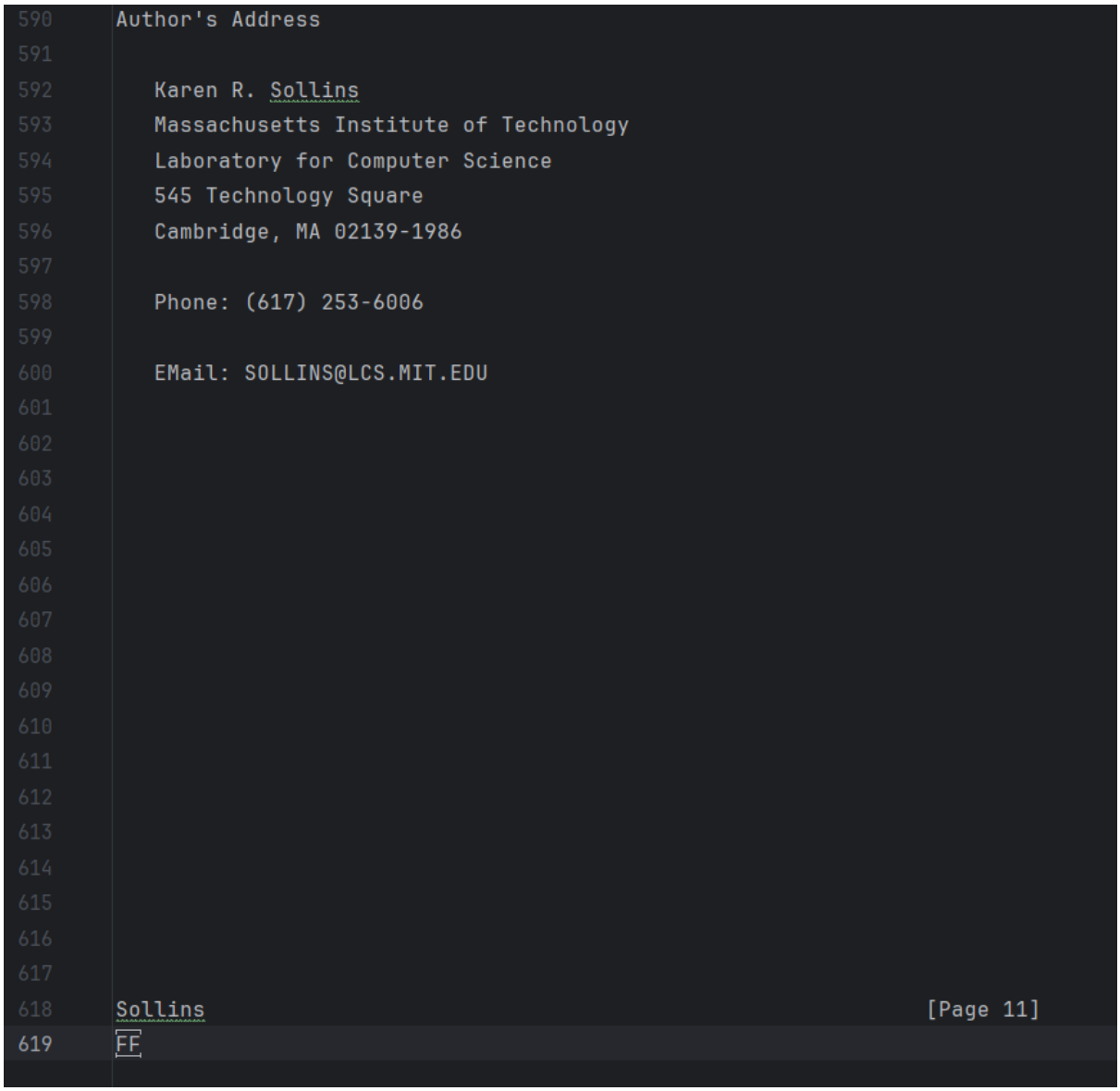
```
127.0.0.1: Acknowledgment Received For Packet 43
127.0.0.1: Sent for Packet: 44, Data Packet Size: 512
127.0.0.1: Acknowledgment Received For Packet 44
127.0.0.1: Sent for Packet: 45, Data Packet Size: 512
127.0.0.1: Acknowledgment Received For Packet 45
127.0.0.1: Sent for Packet: 46, Data Packet Size: 512
127.0.0.1: Acknowledgment Received For Packet 46
127.0.0.1: Sent for Packet: 47, Data Packet Size: 512
127.0.0.1: Acknowledgment Received For Packet 47
127.0.0.1: Sent for Packet: 48, Data Packet Size: 512
127.0.0.1: Acknowledgment Received For Packet 48
127.0.0.1: Sent for Packet: 49, Data Packet Size: 512
127.0.0.1: Acknowledgment Received For Packet 49
127.0.0.1: Sent for Packet: 50, Data Packet Size: 129
127.0.0.1: Acknowledgment Received For Packet 50
127.0.0.1: End of file transfer.
127.0.0.1: File sent to client successfully
```

As you can see from the outputs, they work perfectly due to the fact that they both say block number 50 at the end before the file transfer is finished. Now let's check the file to see if the file has been sent correctly to double-check.

Top of the test.txt (File that was Transfer)

```
Network Working Group                                          K. Sollins
Request For Comments: 1350                                            MIT
STD: 33                                                        July 1992
Obsoletes: RFC 783


                        THE TFTP PROTOCOL (REVISION 2)

Status of this Memo

   This RFC specifies an IAB standards track protocol for the Internet
   community, and requests discussion and suggestions for improvements.
   Please refer to the current edition of the "IAB Official Protocol
   Standards" for the standardization state and status of this protocol.
   Distribution of this memo is unlimited.

Summary

   TFTP is a very simple protocol used to transfer files.  It is from
   this that its name comes, Trivial File Transfer Protocol or TFTP.
   Each nonterminal packet is acknowledged separately.  This document
   describes the protocol and its types of packets.  The document also
   explains the reasons behind some of the design decisions.

Acknowlegements
```

Bottom of the test.txt (File that was Transfer)



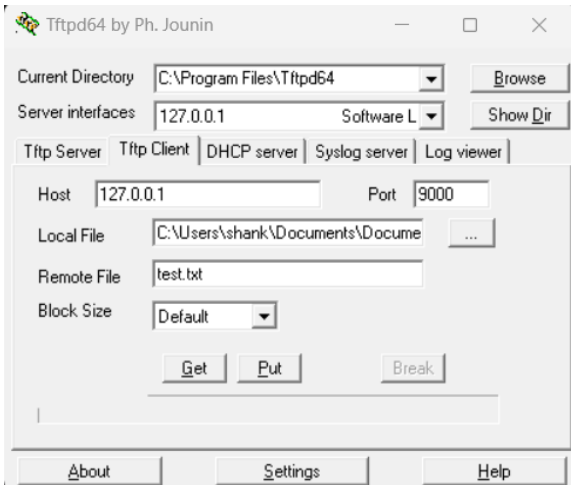
From the pictures above, you can see that the file transfer for indeed successful, and the file has been saved under the current directory to make it easier to test and check the file transfer. Furthermore, I have compared the number of lines to ensure no data was lost and I can confirm that it's the same, so no data was lost.

Third-Party Client sends a File to my TFTP Server:

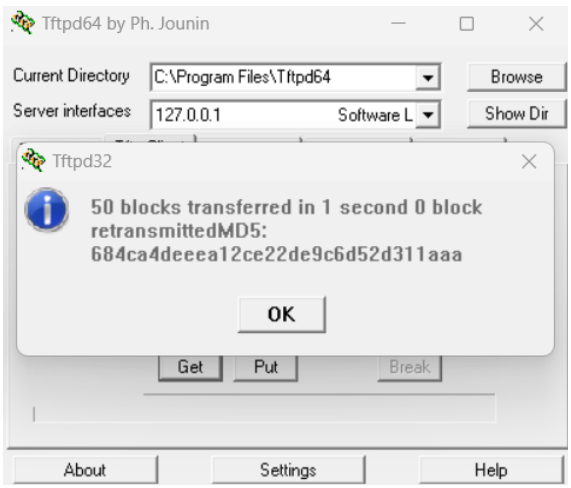
To send a file to the server, we use PUT to send the file. The Local file directory is the file I'm sending (in our case, test.txt), and the Remote file is where I store it on the server. But before we do this, let's make sure to delete the file from the server to ensure that the client is actually sending the file rather than it existing from the previous test (GET).

Tftp64:

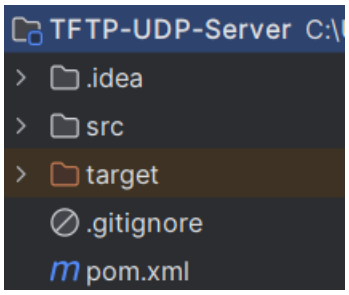
Before File Transfer



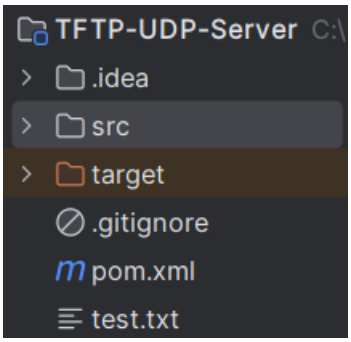
After File Transfer



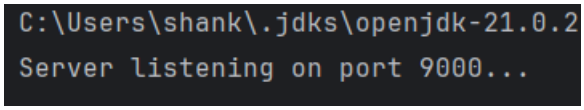
Before File Transfer



After File Transfer



Before File Transfer



After File Transfer – Start of the output from the server.

```
Server listening on port 9000...
Received packet from 127.0.0.1: 49324
127.0.0.1: Received write request for file - test.txt
127.0.0.1: Acknowledgment sent for Packet: 0
127.0.0.1: Received Data Packet: 1, Data Packet Size: 512
127.0.0.1: Acknowledgment sent for Packet: 1
127.0.0.1: Received Data Packet: 2, Data Packet Size: 512
127.0.0.1: Acknowledgment sent for Packet: 2
127.0.0.1: Received Data Packet: 3, Data Packet Size: 512
127.0.0.1: Acknowledgment sent for Packet: 3
127.0.0.1: Received Data Packet: 4, Data Packet Size: 512
```

After File Transfer – End of the output from the server.

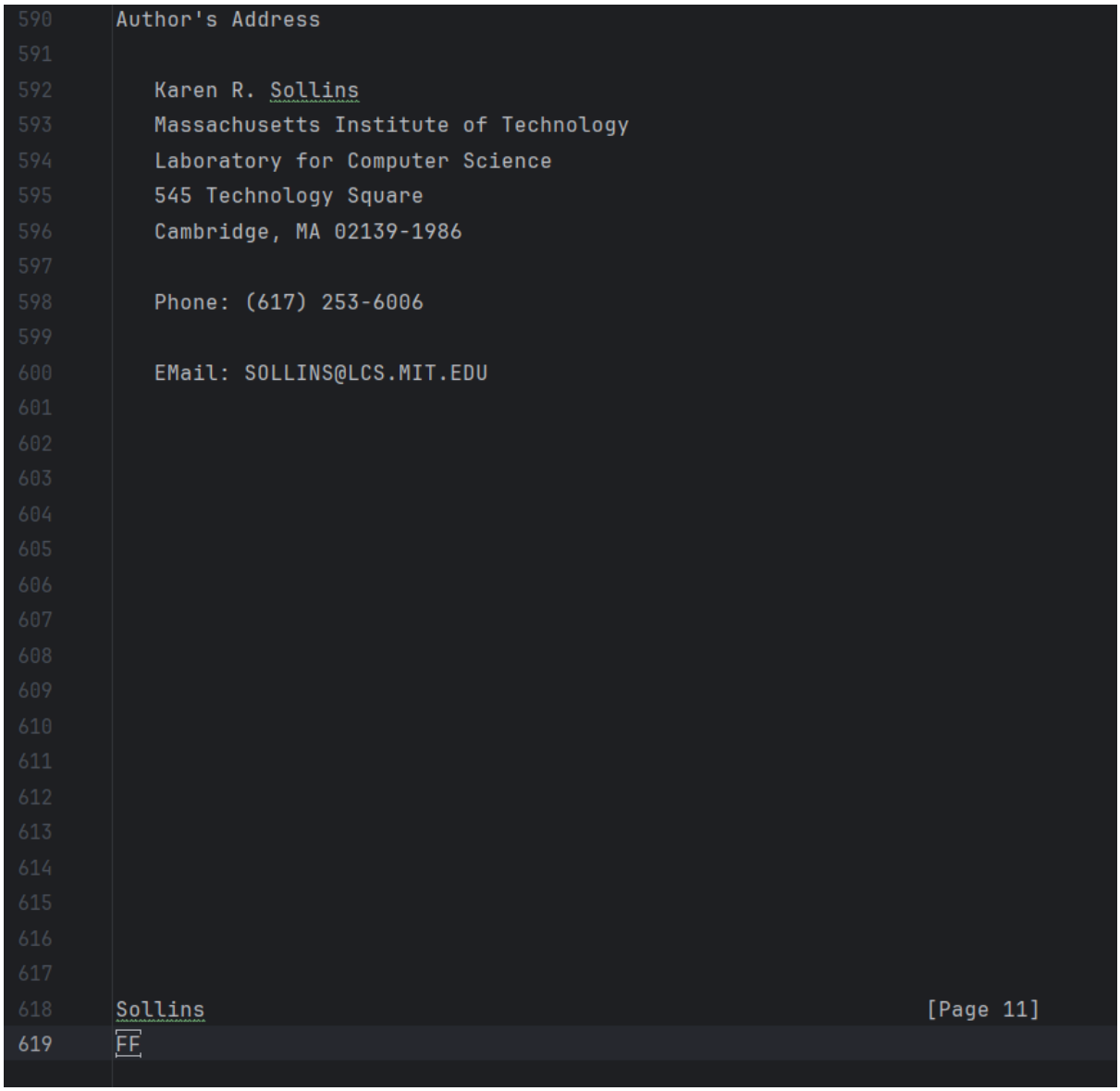
```
127.0.0.1: Received Data Packet: 47, Data Packet Size: 512
127.0.0.1: Acknowledgment sent for Packet: 47
127.0.0.1: Received Data Packet: 48, Data Packet Size: 512
127.0.0.1: Acknowledgment sent for Packet: 48
127.0.0.1: Received Data Packet: 49, Data Packet Size: 512
127.0.0.1: Acknowledgment sent for Packet: 49
127.0.0.1: Received Data Packet: 50, Data Packet Size: 129
127.0.0.1: Acknowledgment sent for Packet: 50
127.0.0.1: End of file transfer.
127.0.0.1: File received from client successfully.
```

As you can see from the outputs, they work perfectly due to the fact that they both say block number 50 at the end before the file transfer is finished. Now let's check the file to see if the file has been sent correctly to double-check.

Top of the test.txt (File that was Transfer)

```
Network Working Group                                          K. Sollins
Request For Comments: 1350                                            MIT
STD: 33                                                        July 1992
Obsoletes: RFC 783


                        THE TFTP PROTOCOL (REVISION 2)

Status of this Memo

   This RFC specifies an IAB standards track protocol for the Internet
   community, and requests discussion and suggestions for improvements.
   Please refer to the current edition of the "IAB Official Protocol
   Standards" for the standardization state and status of this protocol.
   Distribution of this memo is unlimited.

Summary

   TFTP is a very simple protocol used to transfer files.  It is from
   this that its name comes, Trivial File Transfer Protocol or TFTP.
   Each nonterminal packet is acknowledged separately.  This document
   describes the protocol and its types of packets.  The document also
   explains the reasons behind some of the design decisions.

Acknowlegements
```

Bottom of the test.txt (File that was Transfer)



```
590  Author's Address
591
592     Karen R. Sollins
593     Massachusetts Institute of Technology
594     Laboratory for Computer Science
595     545 Technology Square
596     Cambridge, MA 02139-1986
597
598     Phone: (617) 253-6006
599
600     EMail: SOLLINS@LCS.MIT.EDU
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618  Sollins                                                [Page 11]
619  FF
```

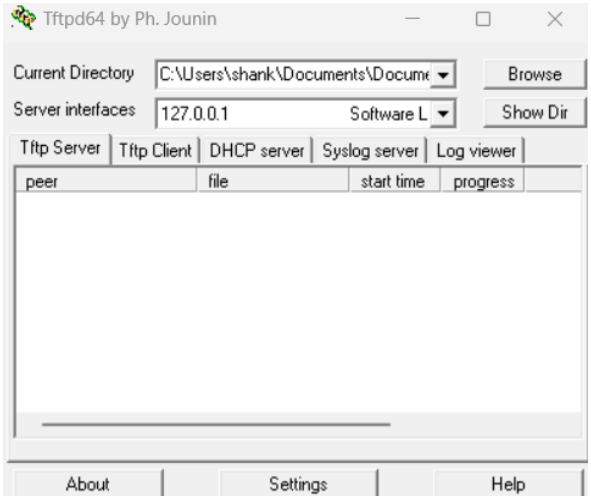
From the pictures above, you can see that the file transfer for indeed successful, and the file has been saved under the current directory to make it easier to test and check the file transfer. Furthermore, I have compared the number of lines to ensure no data was lost and I can confirm that it's the same, so no data was lost.

My Client retrieves a File from the Third-Party TFTP Server:

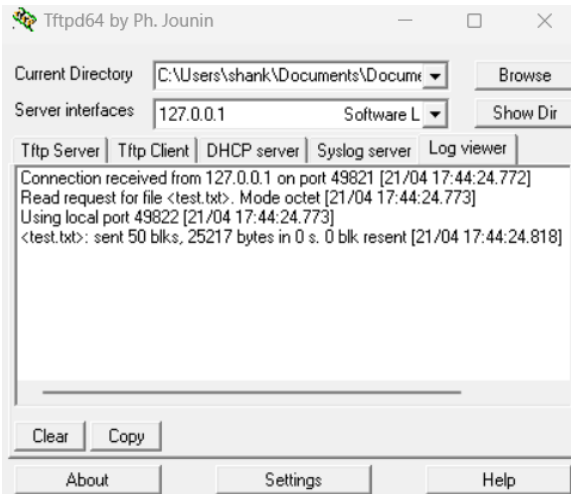
The Third-Party TFTP Server runs locally on port 69 so I had to change my port from 9000 to 69. In the screenshots below, I have ensured that the packets sent and received are traceable so it's easier to understand. Additionally, the client should be on octet mode for this file transfer to work. To retrieve a file to the server, we can use the simple console-based menu I have created with my client. I ensured that the current director is the one that contains the file so I set it to the project folder (current directory as the requirement says) as it contains 'test.txt'.

Tftp64:

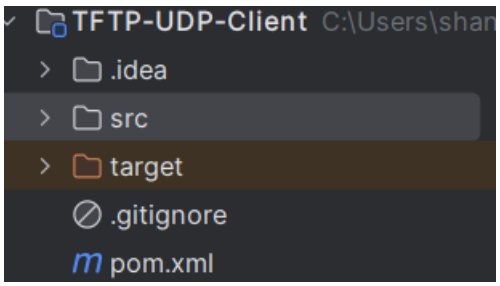
Before File Transfer



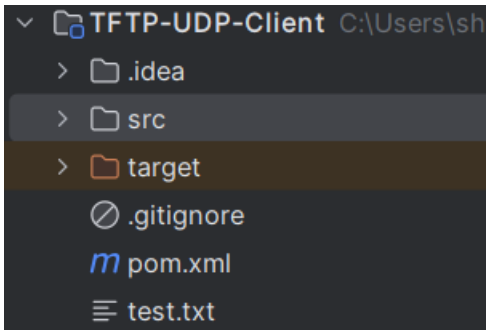
After File Transfer



Before File Transfer



After File Transfer



Console-Base Menu

After File Transfer – Start of the output from the client.

```
C:\Users\shank\.jdks\openjdk-21
Select an option:
1. Retrieve a file
2. Send a file
1
Enter the filename to read:
test.txt
Received Data Packet 1
Sent Acknowledgment Packet 1
Received Data Packet 2
Sent Acknowledgment Packet 2
Received Data Packet 3
Sent Acknowledgment Packet 3
Received Data Packet 4
Sent Acknowledgment Packet 4
Received Data Packet 5
```

After File Transfer – End of the output from the client.

```
Received Data Packet 45
Sent Acknowledgment Packet 45
Received Data Packet 46
Sent Acknowledgment Packet 46
Received Data Packet 47
Sent Acknowledgment Packet 47
Received Data Packet 48
Sent Acknowledgment Packet 48
Received Data Packet 49
Sent Acknowledgment Packet 49
Received Data Packet 50
Sent Acknowledgment Packet 50
File downloaded successfully.
```

As you can see from the outputs, they work perfectly due to the fact that they both say block number 50 at the end before the file transfer is finished. Another way we know it's correct is that we got 50 when we tested for my TFTP Server above. Now let's check the file to see if the file has been sent correctly to double-check.

Top of the test.txt (File that was Transfer)

```
1
2
3
4
5
6
7   Network Working Group                                    K. Sollins
8   Request For Comments: 1350                                      MIT
9   STD: 33                                                  July 1992
10  Obsoletes: RFC 783
11
12
13                      THE TFTP PROTOCOL (REVISION 2)
14
15  Status of this Memo
16
17     This RFC specifies an IAB standards track protocol for the Internet
18     community, and requests discussion and suggestions for improvements.
19     Please refer to the current edition of the "IAB Official Protocol
20     Standards" for the standardization state and status of this protocol.
21     Distribution of this memo is unlimited.
22
23  Summary
24
25     TFTP is a very simple protocol used to transfer files.  It is from
26     this that its name comes, Trivial File Transfer Protocol or TFTP.
27     Each nonterminal packet is acknowledged separately.  This document
28     describes the protocol and its types of packets.  The document also
29     explains the reasons behind some of the design decisions.
30
31  Acknowlegements
```

Bottom of the test.txt (File that was Transfer)

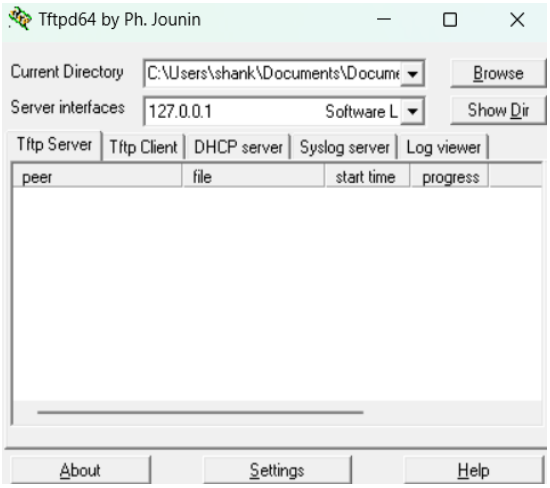
```
590  Author's Address
591
592     Karen R. Sollins
593     Massachusetts Institute of Technology
594     Laboratory for Computer Science
595     545 Technology Square
596     Cambridge, MA 02139-1986
597
598     Phone: (617) 253-6006
599
600     EMail: SOLLINS@LCS.MIT.EDU
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618  Sollins                                                [Page 11]
619  FF
```

From the pictures above, you can see that the file transfer for indeed successful, and the file has been saved under the current directory to make it easier to test and check the file transfer. Furthermore, I have compared the number of lines to ensure no data was lost and I can confirm that it's the same, so no data was lost.

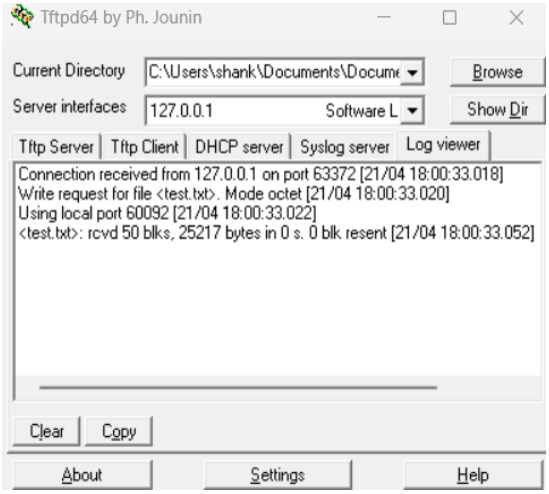
My Client sending a File to the Third-Party TFTP Server:

To send a file to the TFTP Server, we can use the simple console-based menu I have created with my client. Before we do the test for this, let's make sure to delete the file from the server to ensure that the client is actually sending the file rather than it existing from the previous test. Once again, the test.txt file will be used.

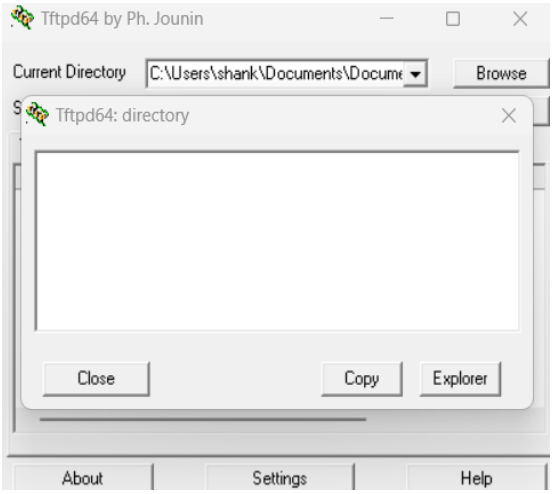
Before File Transfer



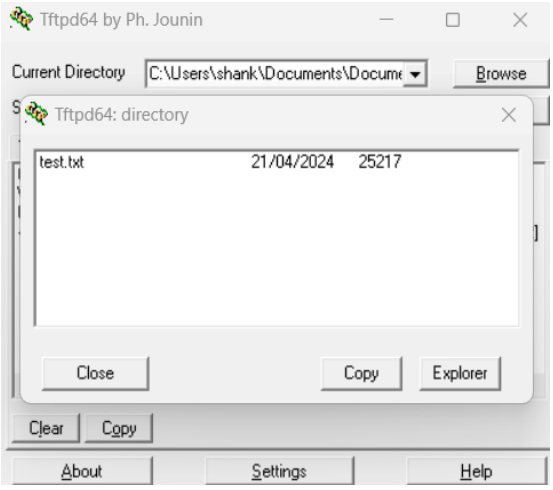
After File Transfer



Directory Before File Transfer



Directory After File Transfer



Console-Base Menu

After File Transfer – Start of the output from the client.

```
C:\Users\shank\.jdks\openjdk-21.0.2\bin\java.exe "-javaagent:C:\Users
Select an option:
1. Retrieve a file
2. Send a file
2
Enter the filename to write:
test.txt
Acknowledgment received from server. Attempting to send file data...
Sent Packet 1
Acknowledgment received from server for packet: 1
Sent Packet 2
Acknowledgment received from server for packet: 2
Sent Packet 3
Acknowledgment received from server for packet: 3
Sent Packet 4
Acknowledgment received from server for packet: 4
Sent Packet 5
Acknowledgment received from server for packet: 5
```

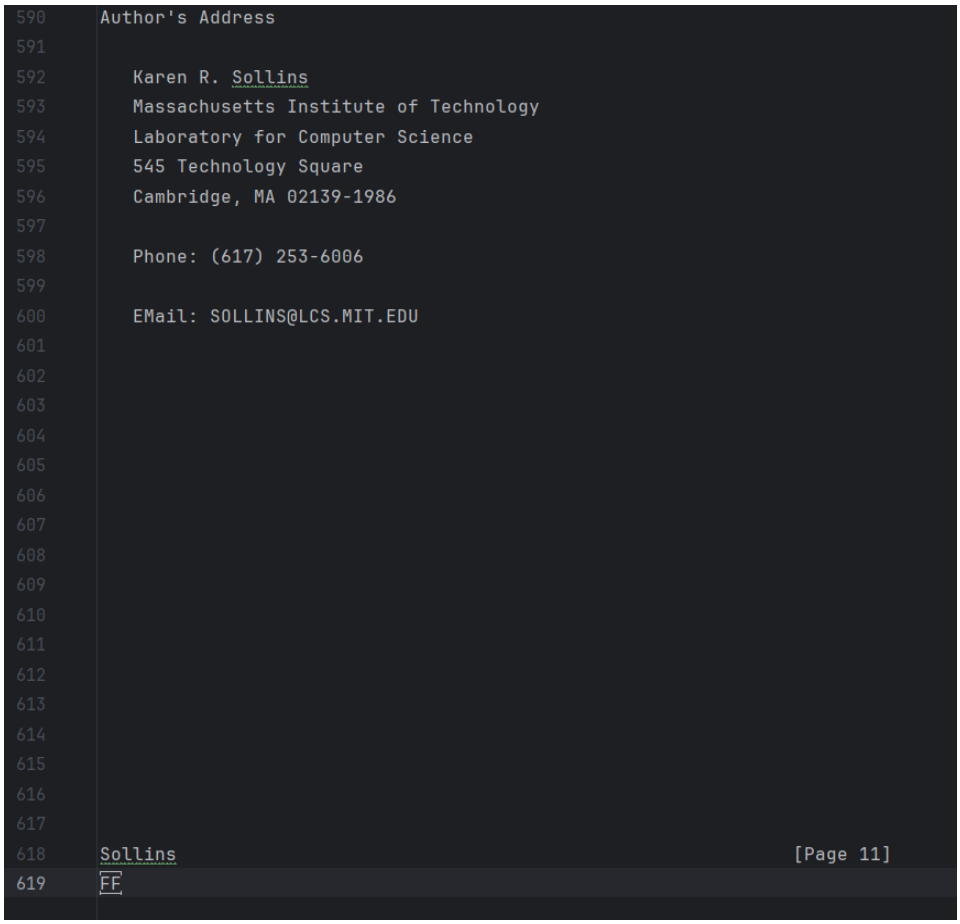
After File Transfer – End of the output from the client.

```
Acknowledgment received from server for packet: 43
Sent Packet 44
Acknowledgment received from server for packet: 44
Sent Packet 45
Acknowledgment received from server for packet: 45
Sent Packet 46
Acknowledgment received from server for packet: 46
Sent Packet 47
Acknowledgment received from server for packet: 47
Sent Packet 48
Acknowledgment received from server for packet: 48
Sent Packet 49
Acknowledgment received from server for packet: 49
Sent Packet 50
Acknowledgment received from server for packet: 50
File sent to server.
```

Top of the test.txt (File that was Transfer)

```
Network Working Group                                          K. Sollins
Request For Comments: 1350                                            MIT
STD: 33                                                        July 1992
Obsoletes: RFC 783



                    THE TFTP PROTOCOL (REVISION 2)

Status of this Memo

   This RFC specifies an IAB standards track protocol for the Internet
   community, and requests discussion and suggestions for improvements.
   Please refer to the current edition of the "IAB Official Protocol
   Standards" for the standardization state and status of this protocol.
   Distribution of this memo is unlimited.

Summary

   TFTP is a very simple protocol used to transfer files.  It is from
   this that its name comes, Trivial File Transfer Protocol or TFTP.
   Each nonterminal packet is acknowledged separately.  This document
   describes the protocol and its types of packets.  The document also
   explains the reasons behind some of the design decisions.

Acknowlegements
```

Bottom of the test.txt (File that was Transfer)



From the pictures above, you can see that the file transfer for indeed successful, and the file has been saved under the current directory to make it easier to test and check the file transfer. Furthermore, I have compared the number of lines to ensure no data was lost and I can confirm that it's the same, so no data was lost.