

IReport for Network protocols

Tommy Chen
221865

Introduction

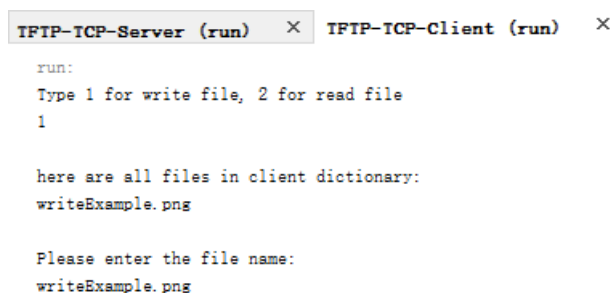
This report would describe how/where the specifications are implemented, 4 protocols would be described separately, all methods' comment would be listed, and a tree graph shows how these methods work (structure). Since TCP protocol is a reliable data transport, therefor it is easier to understand comparing to UDP, TCP protocol would be describe at first.

How to run TCP protocol?

There is a **writeExample.png** in client dictionary, and a **readExample.png** in the server dictionary, if want to test write request, please use **writeExample.png**, else use **readExample.png** to test read request, details would be described below.

Write a file

1. Running the client after the server is run
2. When the message "Type 1 for write file, 2 for read file" is print out, type 1, then the files in client dictionary should be printed out, choose a file to send after the message "Please enter the file name:", show on the **picture 1.0**, it shows the client want to write a file named "**writeExample.png**".
3. the server receives the request and data, the file is saved in a dictionary, the path is "**TFTP-TCP-Server\dictionary**" show on **picture 2.0**, the server class also print out the exact path, showed on **picture 2.1**.

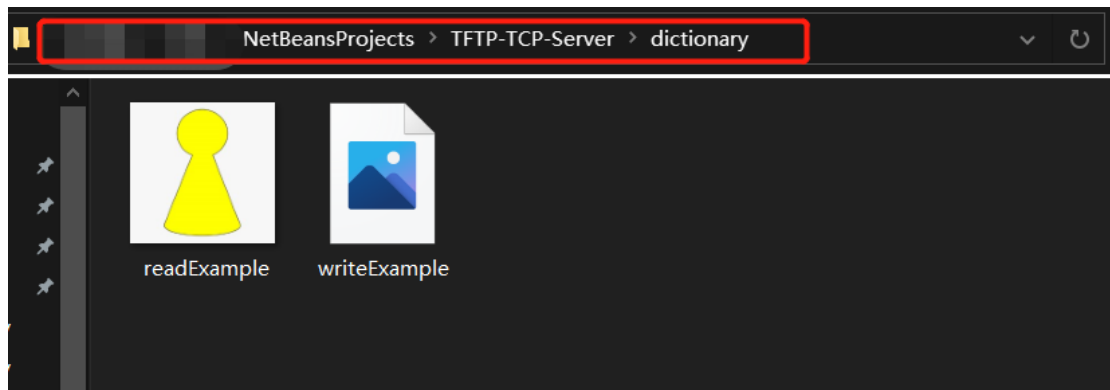


```
TFIP-TCP-Server (run) × TFIP-TCP-Client (run) ×
run:
Type 1 for write file, 2 for read file
1

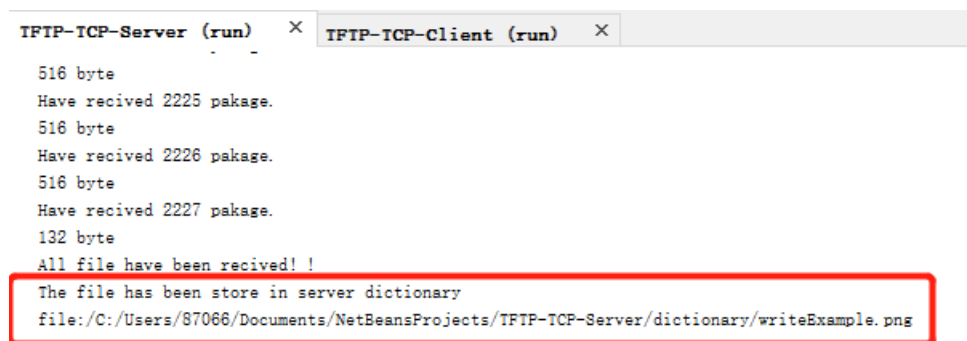
here are all files in client dictionary:
writeExample.png

Please enter the file name:
writeExample.png
```

Picture 1.0 write request: writeExample.png



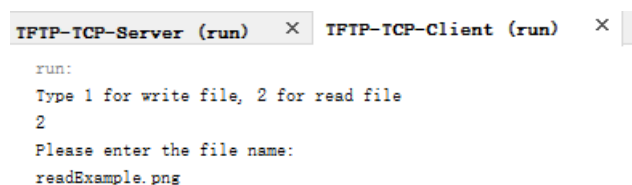
Picture 2.0 how to find the file in server dictionary



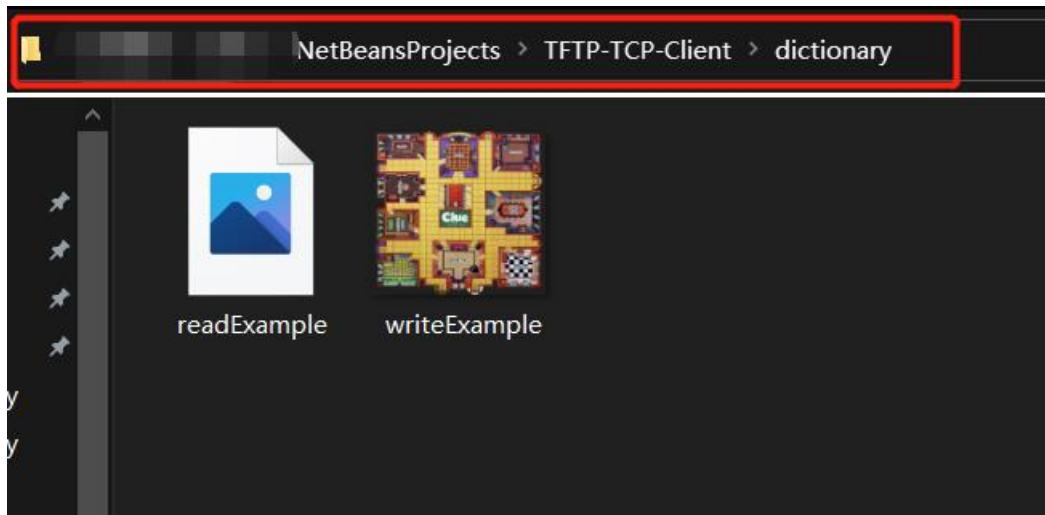
Picture 2.1 the server prints out the location of file

Read a file

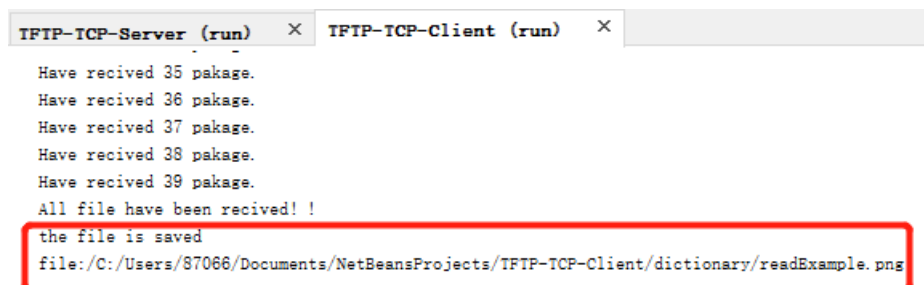
1. Running the client after the server is run
2. When the message "Type 1 for write file, 2 for read file" is print out, type 2, input a file to send after the message "Please enter the file name:", show on the **picture 3.0**, it shows the client want to read a file named "**readExample.png**".
3. the server receives the request and send the data, the file is saved in a dictionary, the path is "**TFTP-TCP-Client\dictionary**" show on **picture 4.0**, the client class also print out the exact path, showed on **picture 4.1**.



Picture 3.0 read request: readExample.png



Picture 4.0 how to find the file in client dictionary



Picture 4.1 the client prints out the location of file

How to run UDP protocol?

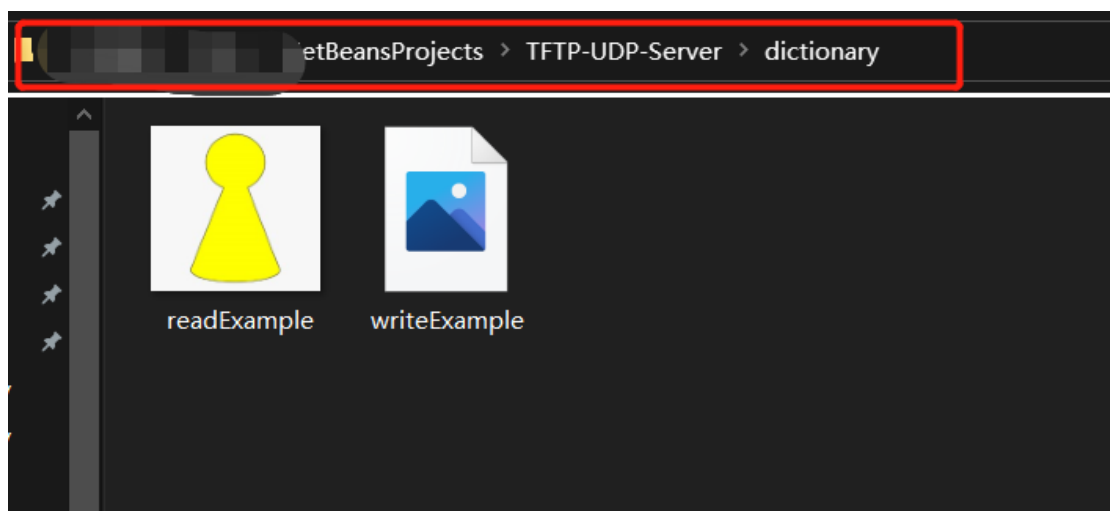
There is a **writeExample.png** in client dictionary, and a **readExample.png** in the server dictionary, if want to test write request, please use **writeExample.png**, else use **readExample.png** to test read request, details would be described below.

Write a file

1. Running the client after the server is run
2. When the message "Type 1 for write file, 2 for read file" is print out, type 1, then the files in client dictionary should be printed out, choose a file to send after the message "Please enter the file name:", show on the **picture 5.0**, it shows the client want to write a file named "**writeExample.png**", after the ack 00 was received, the file transmission is started.
3. the server receives the request and data, the file is saved in a dictionary, the path is "**TFTP-UDP-Server\dictionary**" show on **picture 6.0**, the server class also print out the exact path, showed on **picture 6.1**.

```
TFTP-UDP-Server (run) × TFTP-UDP-Client (run) ×  
  
run:  
Welcome! the ip address is /127.0.0.1  
press 1 to store file, press 2 to retrieve file  
1  
  
here are all files in client dictionary:  
writeExample.png  
  
Please enter file name:  
writeExample.png  
received ack: 00
```

Picture 5.0 write request: writeExample.png



Picture 6.0 how to find the file in server dictionary


```
TFTP-UDP-Server (run) × TFTP-UDP-Client (run) ×  
sending an ack: 100  
sending an ack: 131  
sending an ack: 132  
All file have been received! !  
The file has been store in server dictionary  
file:/C:/Users/87066/Documents/NetBeansProjects/TFTP-UDP-Server/dictionary/writeExample.png  
here are all files in client dictionary:  
readExample.png  
writeExample.png
```

Picture 6.1 the server prints out the location of file

Read a file

1. Running the client after the server is run

- When the message "Type 1 for write file, 2 for read file" is print out, type 2, input a file to send after the message" Please enter the file name:", show on the **picture 7.0**, it shows the client want to read a file named "**readExample.png**".
- the server receives the request and send the data, the file is saved in a dictionary, the path is "**TFTP-UDP-Client\dictionary**" show on **picture 8.0**, the client class also print out the exact path, showed on **picture 8.1**.

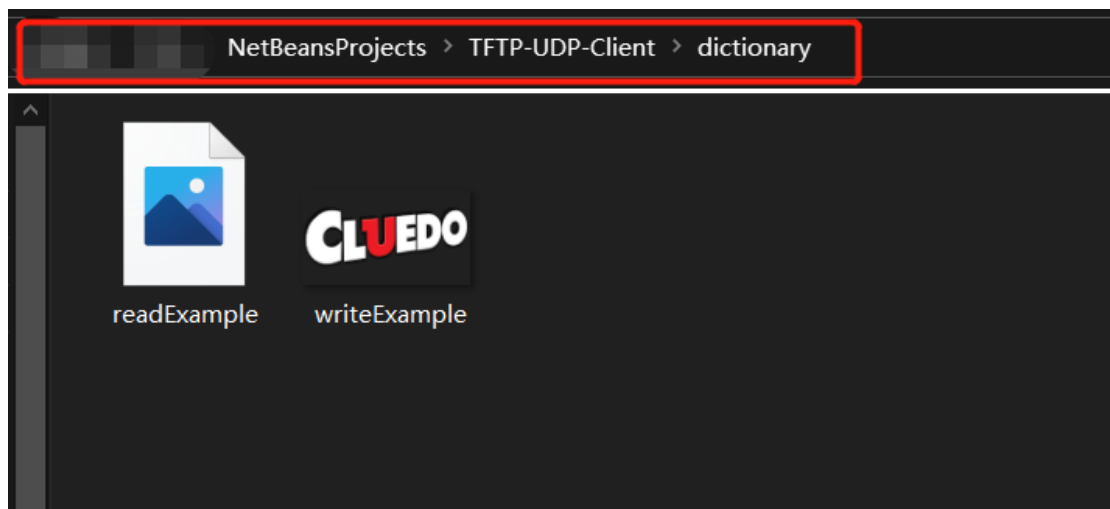


```

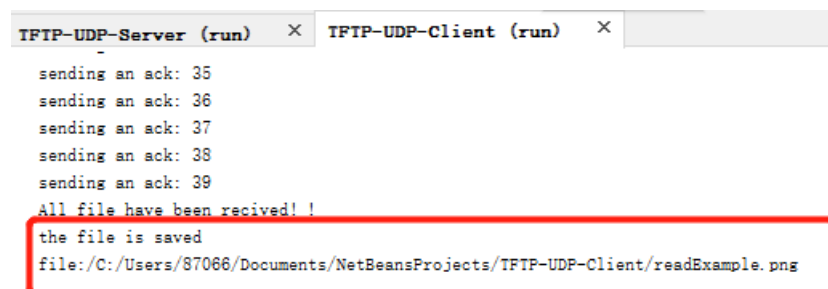
TFTP-UDP-Server (run) x TFTP-UDP-Client (run) x
run:
Welcome! the ip address is /127.0.0.1
press 1 to store file, press 2 to retrieve file
2
Please enter file name:
readExample.png

```

Picture 7.0 read request: readExample.png



Picture 8.0 how to find the file in client dictionary



```

TFTP-UDP-Server (run) x TFTP-UDP-Client (run) x
sending an ack: 35
sending an ack: 36
sending an ack: 37
sending an ack: 38
sending an ack: 39
All file have been recived! !
the file is saved
file:/C:/Users/87066/Documents/NetBeansProjects/TFTP-UDP-Client/readExample.png

```

Picture 8.1 the client prints out the location of file

Source code description for TCP client

run (): The method describes how to use user input to generate the data, for example, when this method is called, it setups the connection and asks" press 1 to store file, press 2 to

retrieve file" and "Please enter file name:" to generate a WRQ/RRQ by createRequest(), if it is a WRQ, then wait a server response(check whether this file exist), if file not exist, write() method was used. Else if it is a RRQ, after the request is sent, the file data or the error data (check whether the file is exist) would be received by receiveFile() and the data would be saved in dictionary by writeFile().

receiveFile(): the method would receive all input from sever, since the RRQ has been sent, sever response can only output a error message or file data, if it is a error message, then print out, else if it is file data, then the data would be written in a ByteArrayOutputStream, if this data package is not a full package(the length is not equal to 516), while loop is broken, the method returns the ByteArrayOutputStream.

sendFile(byte[] content): the method has a content input, if the length of byte[] is bigger than 512, this method would divide byte[] into (length/512), and send these packages one by one.

createRequest(final byte opCode, final String fileName): the method is to return a WRQ/RRQ request byte[] array, and two inputs are required, opcode is to know it is a WRQ or RRQ, and the file name.

reportError(byte errorCode): the method would return a String type error message, for example, if input is 1, the String output would be "File not found."

writeFile(ByteArrayOutputStream b, String fileName): the method needs 2 inputs, the first input b is the content of file, the second one fileName is the file name, then this file would be stored in server dictionary.

findFile(String fileName): the method has a input file name, and the file name is used to search the file in server dictionary and transfer the data into byte[] array, the method would return this byte[] array .

fileExist(String fileName): the method return true if the file is exist in the sever dictionary, otherwise return false.

Source code description for TCP server

run(): when the method is called, the connection with client is setup, and waiting for WRQ/RRQ request, if it is a WRQ request, it would check whether the file is exist, then send a error array if the file is exist, otherwise send the request array back, then wait for the data package by using receiveFile() and write the data into a file using writeFile(). Else if it is a RRQ, fileExist() is used to check the file, if the file is not exist, then sent an error array, otherwise the file is in server dictionary, the data of file would be read and transferred the content into byte[] array, then send the file by sendFile().

writeFile(ByteArrayOutputStream b, String fileName): the method is as same as TCP client's receiveFile().

receiveFile(): the method is as same as TCP client's receiveFile().

createError(byte errorCode): the method needs a errorCode input, the error code determine the error message and the error array which is sent to the client.

sendFile(byte[] temp): the method needs a byte[] input, if the length is bigger than 512, then the byte array need to be divided into (length/512) packages, and these packages are sent one by one.

findFile(String fileName): the method is as same as TCP client's findFile().

fileExist(String fileName): the method return true if the file is exist in the sever dictionary, otherwise return false.

Source code description for UDP client

getConnection(): when the method is called, the connection is setup, the method would print "press 1 to store file, press 2 to retrieve file" to know it is a write or read request, then print "Please enter file name:" to get file name, if it is a WRQ, then the method write(String fileName) would be called, else if RRQ, read(String fileName) method is called.

read(String fileName): the method has a input filename. When the method is called, receiveFile() is used and get a byte[] array, and save the data by using writeFile().

write(byte[] content): if the length of content is smaller than 512, the data would be put into the package directly, else the data would be separated into (content.length/512 bytes) packages, after a package is sent, a ack package is waiting, then confirm the ack or do the retransmission.

createRequest(final byte opCode, final String fileName): the method is to return a WRQ/RRQ request byte[] array, and two inputs are required, opcode is to know it is a WRQ or RRQ, and the file name.

receiveFile(): if the received file is a error package, then reportError is called, else if it is a ack package the block number would be written in a ByteArrayOutputStream, else if it is a data package, the recieved data is written in ByteArrayOutputStream. Each package is checked by isLastPacket() to know whether it is a last package, if it is not a last package, the method would continue to receive package, otherwise return the ByteArrayOutputStream.

sendAcknowledgment(byte[] blockNumber): the method sent a ack package with the block number.

isLastPacket(DatagramPacket datagramPacket): the method is to check whether the length of input datagramPacket is smaller than 512, if smaller than 512, that is the last data package, return true, else return false.

reportError(): the method would print out the error code and message according to the error package.

Source code description for UDP server

run(): when the method is called, the input datagram package is waiting, if the package is a WRQ request, then the server would check whether the file is exist in dictionary (by `fileExist()`), if file not exist, then the WRQ is confirmed and a ask 00 is sent, then the server start to receive the data package or a error package by `receiveFile()`, and the data is saved by `writeFile()`.

receiveFile(): if the received file is a error package, then `reportError` is called, else if it is a ack package the block number would be written in a `ByteArrayOutputStream`, else if it is a data package, the received data is written in `ByteArrayOutputStream`. Each package is checked by `isLastPacket()` to know whether it is a last package, if it is not a last package, the method would continue to receive package, otherwise return the `ByteArrayOutputStream`.

sendData(byte[] content): the method has a input content, if the length of content is smaller than 512, the data would be put into the package directly, else the data would be separated into $(\text{content.length}/512 \text{ bytes})$ packages, after a package is sent, a ack package is waiting, then confirm the ack or do the retransmission.

sendError(byte errorCode): according to the input `errorCode`, the `errorCode` determine the content of error package, for example, if the `errorCode` is 1, the error message is "File not found", then the error package is sent.

sendAcknowledgment(byte[] blockNumber): the method sent a ack package with the block number.

writeFile(ByteArrayOutputStream b, String fileName): the method would get data `byte[]` from `ByteArrayOutputStream b` and create a file with file name and store this file by using `FileOutputStream`.

isLastPacket(DatagramPacket datagramPacket): the method is as same as UDP clients' `isLastPacket(DatagramPacket datagramPacket)`.

findFile(String fileName): the method would find the file and transfer the file data into `byte[]`, then return the `byte[]` array.

fileExist(String fileName): the method return true if the file is exist in the sever dictionary, otherwise return false.