



Linnéuniversitetet

Kalmar Vaxjö

Report

Assignment 3

IDV701



Author1: Melat Haile
Author2: Nahomie Haile
Semester: Spring 2023
Email : mh225ic@student.lnu.se
Email : nh222rt@student.lnu.se

Contents

1 Problem 1	I
1.1 Discussion	III
2 Problem 2	IV
2.1 Discussion	V
3 Problem 3	V
3.1 Discussion	VIII
4 Problem 4	IX
5 Summary	X

1 Problem 1

```
private InetAddress receiveFrom(DatagramSocket socket, byte[]
buf) {
    // Create datagram packet
    DatagramPacket packet = new DatagramPacket(buf, buf.length);
    // Receive packet

    // Get client address and port from the packet
    try {
        socket.receive(packet);
        checkForErrorPacket(buf);
    } catch (Exception e) {
        e.printStackTrace();
    }
    InetAddress socketAddress = new
InetAddress(packet.getAddress(), packet.getPort());

    return socketAddress;
}
```

```
private int ParseRQ(byte[] buf, StringBuffer requestedFile) {
    // See "TFTP Formats" in TFTP specification for the RRQ/WRQ request
contents
    int n = 0;
    int counter= 0;
    String mode = "";
    byte[] buffr = new byte[BUFSIZE];
    ByteBuffer buffer = ByteBuffer.wrap(buf);
    short opcode = buffer.getShort();
    for (int i = 1; i < buf.length; i++) {
        if (buf[i] == 0) {
            n++;
            if (n == 1) {
                String fileName = new String(buffr).trim();
                requestedFile.append(fileName);
                buffr = new byte[BUFSIZE];
            } else if (n == 2) {
                mode = new String(buffr).trim();
                if (!mode.equals("octet")) {
                    System.out.println("Transfer mode not octet!");
                    System.exit(0);
                }
                break;
            }
        }
    }
    if ((buf[i] != 0)) {
        buffr[counter] = buf[i];
        counter++;
    }
}
```

```

    }
}
return opcode;
}

```

```

private boolean send_DATA_receive_ACK(DatagramSocket socket, String
requestedFile) {
    boolean isDataSentAndAked = false;
    byte[] data = new byte[512];
    byte[] ack = new byte[BUFSIZE];
    int block = 1;
    int count = 0;
    File file = new File(".");
    try {
        file = new File(file.getCanonicalPath()+requestedFile);
        if(file.exists()){
            byte[] buffer =
Files.readAllBytes(Paths.get(file.getAbsolutePath()));
            for (int i = 0; i < buffer.length; i++) {
                data[count] = buffer[i];
                count++;
                if(count == 511){
                    ack = pushData(data, socket, block);
                    checkForErrorPacket(ack);

                    count = 0;
                    block++;
                    data = new byte[512];
                }
            }

            if(count < 512){
                ack = pushData(data, socket, block);
                checkForErrorPacket(ack);
                isDataSentAndAked = OP_ACK == ack[1]? true : false;
                System.out.println(isDataSentAndAked? " data has
been sent": "data not sent");
            }
        } else{
            send_ERR(socket, ERR_FILE_NOT_FOUND, "File not Found");
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return isDataSentAndAked;
}

```

```

    private boolean receive_DATA_send_ACK(DatagramSocket socket, String
requestedFile) {
        boolean isFileRecievedFully = false;
        int block = 0;
        File file = new File(".");
        file = new File(file.getAbsolutePath() + requestedFile);
        if(!file.exists()){
            byte[] buf = new byte[BUFSIZE];
            try {
                FileOutputStream out = new
FileOutputStream(file.getAbsolutePath());
                socket.send(dataAckPacket(block));
                while(true){
                    DatagramPacket packet = new DatagramPacket(buf,
buf.length);

                    socket.receive(packet);
                    checkForErrorPacket(buf);
                    socket.send(dataAckPacket(++block));
                    Thread.sleep(1000);
                    out.write(removeHeadersFromByteArray(buf));
                    out.flush();
                    if(packet.getLength() < 512){
                        out.close();
                        socket.close();
                        socket.disconnect();
                        return isFileRecievedFully = true;
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }else{
            send_ERR(socket, ERR_FILE_ALREADY_EXISTS, "File already
exists!");
        }

        return isFileRecievedFully;
    }

```

1.1 Discussion

In computer networking, a socket is a mechanism for establishing communication between two computer processes. A socket is identified by an IP address and a port number, and it can be used to listen for incoming connections or to initiate outgoing connections.

To listen for incoming connections on a specific port, a socket must be bound to that port. This is typically done using the `bind()` method of the socket object, which takes as argument the local bind point (i.e., the IP address and port number to bind to). For

example, `socket.bind(localBindPoint)` would bind the socket to the specified local bind point.

Once the socket is bound and listening for incoming connections, it can accept incoming requests and establish connections with clients. When a read or write request is received from a client, the socket can use the client's IP address and port (which were obtained during the listening phase) to connect to the client. Once the connection is established, the socket can be used to send and receive data to and from the client.

To implement a network protocol, such as the Trivial File Transfer Protocol (TFTP), various methods are used to handle incoming requests and respond appropriately. For example, the `recvFrom()` method can be used to receive a request packet from the client, and the packet buffer can be used to extract the client's IP address and port number.

Once the request has been received and parsed (e.g., using a `parseRQ()` method to extract the opcode, requested file, and mode), the appropriate response method can be called. For example, the `HandleRQ()` method might contain several sub-methods for handling different types of requests, such as `send_DATA_receive_ACK()`.

In the case of `send_DATA_receive_ACK()`, the requested file is read (if it exists), and the data is deposited in a packet. The packet is then sent to the client using the `sendSocket`, which is also used to receive an acknowledgement from the client. This process continues until all data has been sent to the client or an error occurs.

Overall, the use of sockets and various networking methods can enable the implementation of complex network protocols and applications, allowing for efficient and secure communication between distributed computer systems.

2 Problem 2

The HTTP response message contains the following components:

1. Status code: The status code is specified as 200. This indicates that the request has succeeded and the server has successfully fulfilled the client's request.
2. Status code description: The status code description is specified as "OK". This indicates that the server has successfully processed the request and has returned the requested resource.
3. Last-Modified: The Last-Modified header is specified. This is a date and time stamp that indicates the last time the resource was modified according to the origin server.
4. Content-Length: The Content-Length header is specified as 128. This is the size, in bytes, of the message body that is being sent in the response. In this case, the message body is the requested resource.

When the client receives the HTTP response message, it will interpret the status code and status code description to determine if the request was successful. If the request was successful, the client will then process the message body (in this case, the requested resource) according to the specifications of the HTTP protocol. The Last-Modified header can be used by the client to determine if the requested resource has been modified since the client last accessed it. The Content-Length header can be used by the

client to determine the size of the message body and to allocate sufficient memory for processing it.

2.1 Discussion

The given scenario involves an HTTP request message sent from a machine with IP address 192.168.0.32 to a destination with IP address 184.31.15.184. The HTTP request message contains the following components:

Request method: The request method is specified as GET. This indicates that the client is requesting the server to retrieve a resource identified by the request URL.

Request URL: The request URI is specified as “/wireshark-labs/HTTPwireshark-file1.html”. This indicates that the client is requesting the server to retrieve the file named “HTTPwireshark-file1.html” located in the “wireshark-labs” directory on the server.

Request version: The request version is specified as HTTP/1.1. This indicates that the client is using the HTTP/1.1 protocol to communicate with the server.

Host: The Host header is specified as “gaia.cs.umass.edu”. This indicates the hostname of the server that the client is communicating with.

When the HTTP request message is sent from the client machine to the destination server, the server will receive the message and use the information provided to retrieve the requested resource. The server will then construct an HTTP response message containing the requested resource and send it back to the client machine. The client machine will then receive the response message and display the requested resource to the user.

3 Problem 3

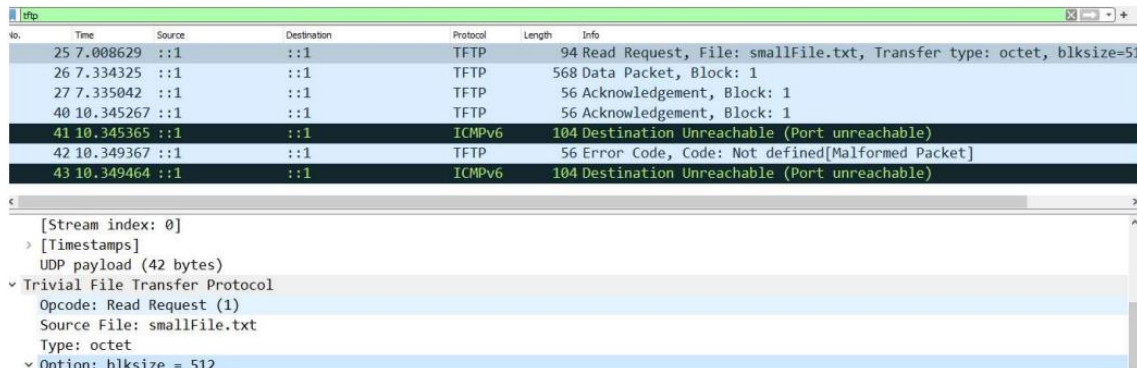
Analysis of a read request packet in the Wireshark network analysis tool. The read request packet is a part of the TFTP (Trivial File Transfer Protocol) protocol and is used to request a file from a remote server.

The Wireshark analysis of the read request packet involves the following components:

1. **Source and destination IP addresses:** The read request packet contains the source and destination IP addresses of the client and the server, respectively. These addresses are used to identify the source and destination of the packet.
2. **Source and destination ports:** The read request packet also contains the source and destination port numbers of the client and the server, respectively. These port numbers are used to identify the application that is sending or receiving the packet.
3. **Opcode:** The opcode field in the read request packet indicates the type of packet being sent. In this case, the opcode field will contain a value of 1, which indicates that this is a read request packet.
4. **Requested file name:** The read request packet also contains the name of the file that the client is requesting. This is specified in the data field of the packet.

5. Transfer mode: The read request packet also contains the transfer mode that the client is requesting. This can be either "netascii", "octet", or "mail", and is also specified in the data field of the packet.

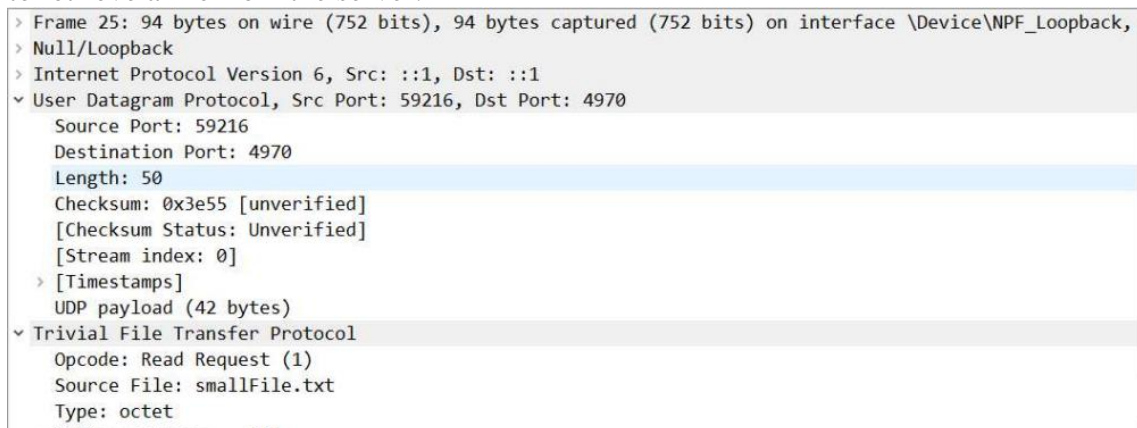
The Wireshark analysis of the read request packet can be used to troubleshoot issues related to TFTP file transfers, such as slow transfer speeds or errors during the transfer process. By analyzing the packet contents, network administrators can determine the cause of the issue and take steps to resolve it.



No.	Time	Source	Destination	Protocol	Length	Info
25	7.008629	::1	::1	TFTP	94	Read Request, File: smallFile.txt, Transfer type: octet, blksize=512
26	7.334325	::1	::1	TFTP	568	Data Packet, Block: 1
27	7.335042	::1	::1	TFTP	56	Acknowledgement, Block: 1
40	10.345267	::1	::1	TFTP	56	Acknowledgement, Block: 1
41	10.345365	::1	::1	ICMPv6	104	Destination Unreachable (Port unreachable)
42	10.349367	::1	::1	TFTP	56	Error Code, Code: Not defined[Malformed Packet]
43	10.349464	::1	::1	ICMPv6	104	Destination Unreachable (Port unreachable)

[Stream index: 0]	
[Timestamps]	
UDP payload (42 bytes)	
Trivial File Transfer Protocol	
Opcode: Read Request (1)	
Source File: smallFile.txt	
Type: octet	
Option: blksize = 512	

Packet 25 represents a client read request. In TFTP, a read request is made by the client to retrieve a file from the server.



Frame 25: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface \Device\NPF_{...}, Null/Loopback	
Internet Protocol Version 6, Src: ::1, Dst: ::1	
User Datagram Protocol, Src Port: 59216, Dst Port: 4970	
Source Port: 59216	
Destination Port: 4970	
Length: 50	
Checksum: 0x3e55 [unverified]	
[Checksum Status: Unverified]	
[Stream index: 0]	
[Timestamps]	
UDP payload (42 bytes)	
Trivial File Transfer Protocol	
Opcode: Read Request (1)	
Source File: smallFile.txt	
Type: octet	

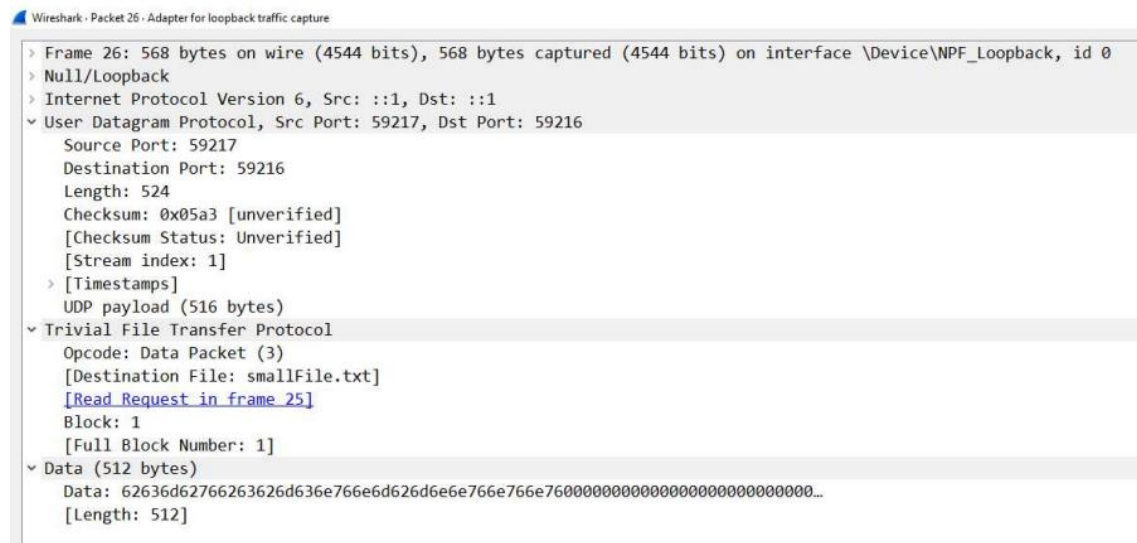
The information provided refers to a data packet being sent from a server with a source port number of 4970, and the payload of the packet is 42 bytes in length. The data packet is being transmitted using the Trivial File Transfer Protocol (TFTP), a simple file transfer protocol that allows files to be transferred between network devices.

The TFTP section of the data packet contains specific information regarding the file transfer request. It shows that the file being transferred is named "smallFile.txt," and the opcode for the request is 1, which indicates a read request. Additionally, the transfer mode is specified as octet, which is a type of binary file transfer that ensures that the data is transferred without any modifications.

Packet 26

Line 26 represents a response from the server. The response is a data packet that includes a block number and data. The block number is used to keep track of the sequence of blocks being transferred, and in this case, it corresponds to the first block being sent. The data contained in the packet is the actual data being transferred from the server to the client.

It is important to note that TFTP is a simple file transfer protocol that lacks many of the security features found in other file transfer protocols. Therefore, it is primarily used in situations where security is not a concern, such as in local area networks (LANs) or for transferring non-critical files.



The provided screen shot depicts a data packet that was sent by the server in a file transfer protocol known as TFTP. The data packet's size is indicated as 516 bytes, and it is composed of several sections, including the TFTP section and the data section.

The TFTP section contains specific information about the type of request being made, such as the opcode. In this particular data packet, the opcode is 3, which is used to indicate that the data being sent is part of a data transfer operation. Additionally, the destination file name, in this case, `smallFile.txt`, and the block number, which is 1, are also included in this section.

The data section of the packet contains the actual data being transferred, which, in this case, is represented in hexadecimal format. The length of the data section is specified as 512 bytes.

Packet 27

Regarding the specific line mentioned, which is line 27, it refers to an acknowledgement from the client to the server. When the client receives a data packet from the server, it sends an acknowledgement packet back to the server indicating that the data was received successfully. In this case, line 27 represents such an acknowledgement packet, confirming the successful receipt of the first block of data, which is indicated by the block number of 1 in the TFTP section of the original data packet.

```
Wireshark - Packet 27 - Adapter for loopback traffic capture

Length: 12
Checksum: 0x312d [unverified]
[Checksum Status: Unverified]
[Stream index: 1]
> [Timestamps]
UDP payload (4 bytes)
▼ Trivial File Transfer Protocol
  Opcode: Acknowledgement (4)
  [Destination File: smallFile.txt]
  [Read Request in frame 25]
  Block: 1
  [Full Block Number: 1]
```

```
Wireshark - Packet 37 - Adapter for loopback traffic capture

> Frame 37: 96 bytes on wire (768 bits), 96 bytes captured (768 bits) on interface \Device\NPF_{...}_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 6, Src: ::1, Dst: ::1
▼ User Datagram Protocol, Src Port: 61275, Dst Port: 4970
  Source Port: 61275
  Destination Port: 4970
  Length: 52
  Checksum: 0xff13 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 0]
  > [Timestamps]
  UDP payload (44 bytes)
  ▼ Trivial File Transfer Protocol
    Opcode: Write Request (2)
    Destination File: smallFile.txt
    Type: octet
    > Option: blksize = 512
    > Option: tsize = 512
```

3.1 Discussion

Read and write requests are two fundamental types of requests that are used to access and manipulate data on a networked device or system. The main difference between a read request and a write request is the opcode used to identify the type of request.

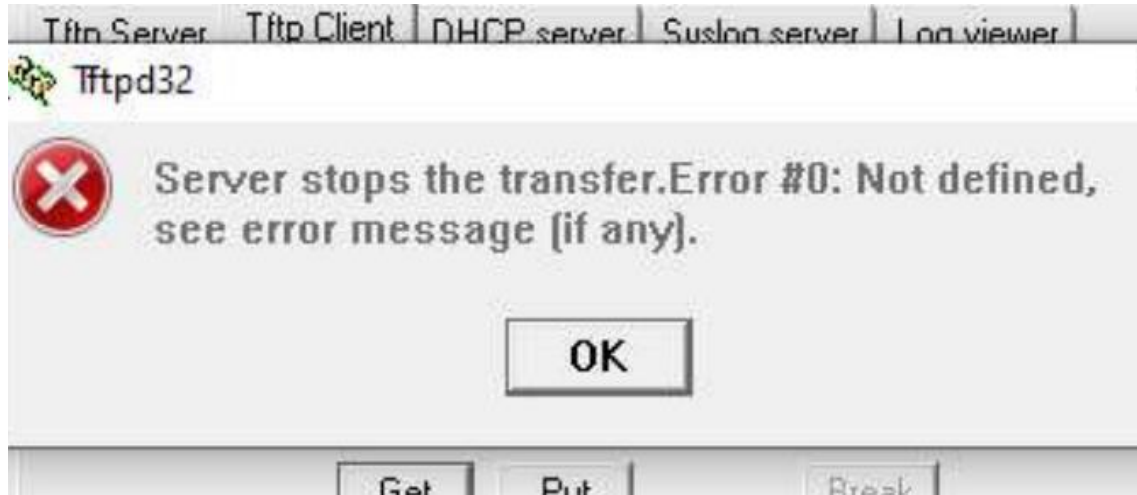
An opcode is a code that indicates the type of operation being requested. In this context, opcode 1 is used to indicate a read request, while opcode 2 is used to indicate a write request. Therefore, when a client sends a request to read data from a server, it uses opcode 1 to specify that it is a read request. On the other hand, when a client sends a request to write data to a server, it uses opcode 2 to specify that it is a write request.

Read requests are typically used when a client needs to retrieve data from a server, while write requests are used when a client needs to update or modify data on a server. The server will then process the request and send a response back to the client, indicating whether the operation was successful or not.

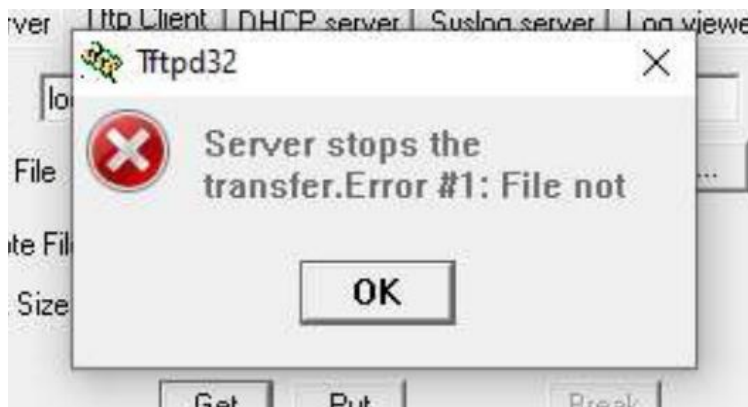
It is important to note that there may be other opcodes used for different types of requests, depending on the specific protocol or network architecture being used. However, in the context of the screen shot provided, opcode 1 is used for read requests, while opcode 2 is used for write requests.

4 Problem 4

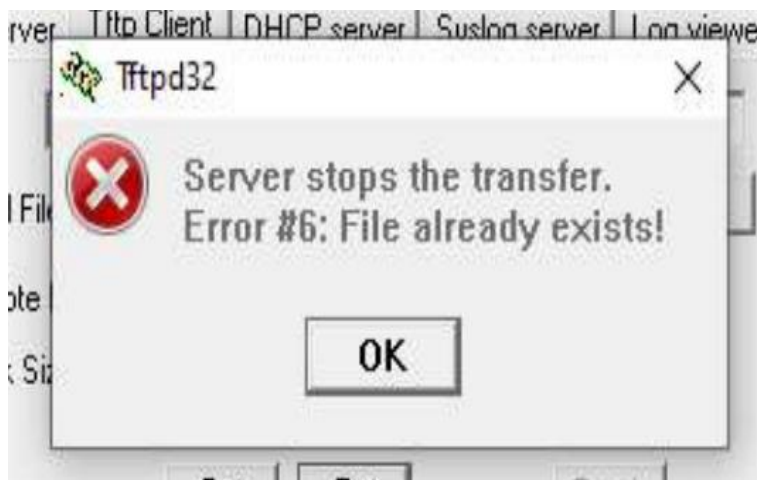
Screenshot shows not defined error.



Screenshot shows file not found error.



Screenshot shows file already exists error.



5 Summary

Melat Haile: 50%

Nahomie Haile: 50%

Both of us worked together in all of the tasks.