STU Bratislava, Fakulta informatiky a informačných technológií

Počítačové a komunikačné siete – Zadanie 2

Komunikácia s využitím UDP protokolu

Dokumentácia

Meno: Andrii Rybak

AIS ID: 105840

Cvičiaci: Ing. Kristián Košťál, PhD.

Akademicky rok: 2021/2022

Andrii Rybak
ID: 105840

Content

Protocol design

Every protocol packet consists of a 1-byte operation code (opcode) field in the beginning. There are supported 6 following opcodes:

opcode	**operation**
1	Write request (WRQ)
2	Data (DATA)
3	Acknowledgment (ACK)
4	Resend request (RRQ)
5	Keep connected request (KCRQ)
6	Error (ERROR)

The packet format is:

1B	4B	0-1425B <--->	2B
Operation code	Fragment number - #	Data (optional)	Checksum

Data field is used to carry different information. In case of write request, the mentioned field is filled with the file's name. In case of data packet, the data fragment is carried in this field. If the operation code is Error, the error message is in here.

Data field can be also empty when it is not needed – in ACK, RRQ, KCRQ packets for example.

If the fragment number is not needed (in WRQ, KCRQ, ERROR packets), it will be filled with zeros.

The maximum size of data field specified in header is not random. The thing is, the maximum size of the Ethernet Type II frame is 1518 bytes, from which for nested data can be used up to 1500 bytes. Ipv4 header takes 20 bytes without options and up to 60 bytes with them. UDP header always takes 8 bytes. Therefore, protocol designed in this paper can use maximum 1432 bytes so that the packet would not be fragmented on link layer of OSI model.

Peer-to-peer

The protocol is designed as peer-to-peer, so that the role of each side is set automatically when one of the sides sends the data or write request packet. This side is considered as sender and opposite one is receiver. After the first file or message was successfully sent, the receiver can become sender by sending the data or write request packet.

Stop-and-wait ARQ

Each **data** packet contains one fragment of file or text message and must be acknowledged by an **acknowledgment** packet before the next packet can be sent. If the sending side does not receive the **acknowledgment** packet before a certain time (timeout), the packet is resent automatically until sender receives an acknowledgment or exceeds a predefined number of retransmissions.

The timeout countdown and number of retransmissions are reset after each frame transmission.

If newly calculated checksum and written one in the packet do not match, the side that received bad packet sends the **resend request** packet. After which the packet is sent again.

Keep-alive method

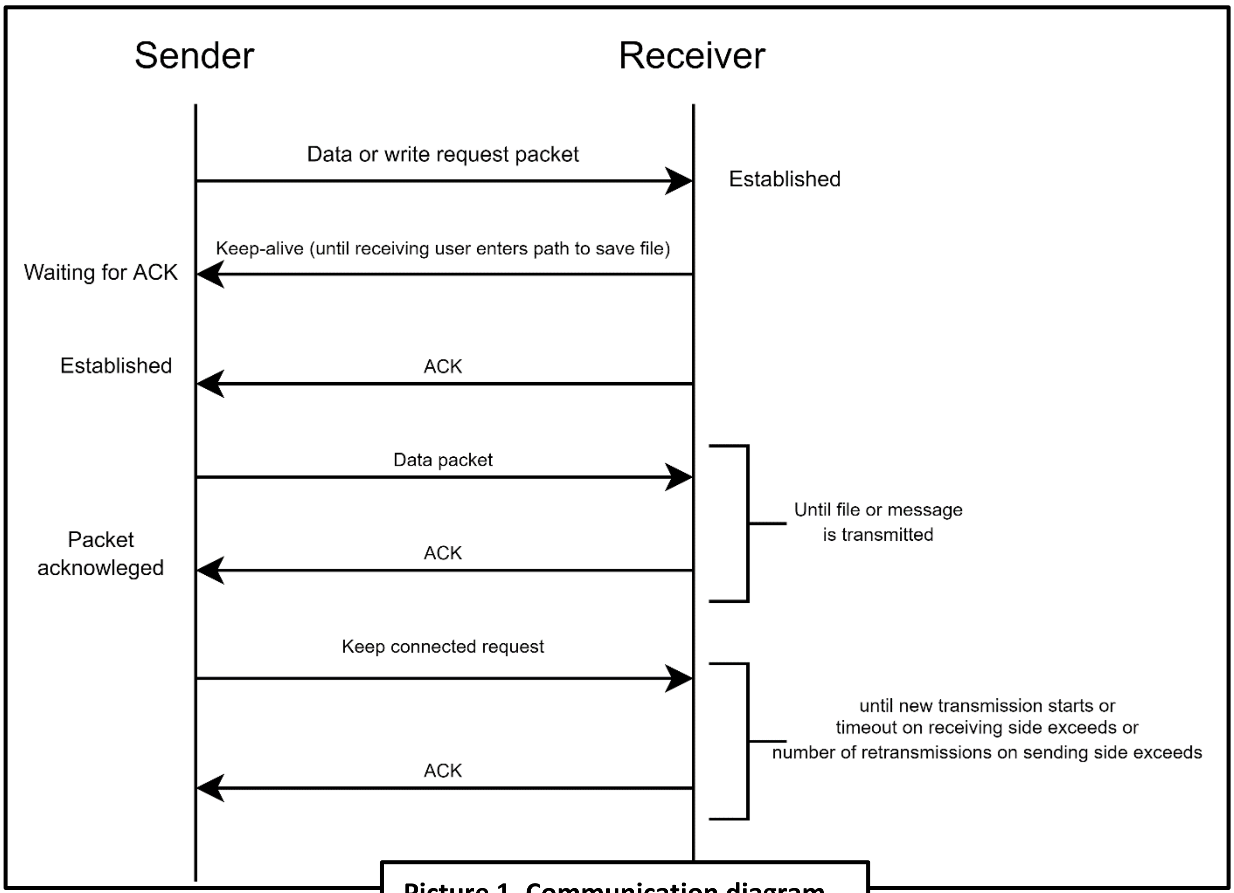
The connection is automatically kept alive after the file was successfully transmitted. The keep-alive process is performed by sending the **keep connected request** every timeout number of seconds and receiving the acknowledgment packet.

Checksum

It was decided to implement CRC-16/BUYPASS with polynomial 0x8005 and initially set on 0x0 in this protocol. The main reason is the reliability of this method. There is extremely low chance that mutated data will have the same checksum as primary one, because every bit has huge impact on the result, so that changing only one bit will lead to the absolutely different checksum.
Another advantage of CRC is that it is relatively easy to implement. Taking into consideration the fact that the checksum is in the end of the protocol, to check if the packet was successfully transmitted there is only needed to calculate CRC for the whole packet. If the result is 0, the packet is correct.

Andrii Rybak
ID: 105840

Communication process


Sender
Receiver
Data or write request packet
Established
Keep-alive (until receiving user enters path to save file)
Waiting for ACK
Established
ACK
Data packet
Until file or message is transmitted
Packet acknowleged
ACK
Keep connected request
until new transmission starts or timeout on receiving side exceeds or number of retransmissions on sending side exceeds
ACK

Picture 1. Communication diagram

Initial Connection

A transfer is established by sending a **write request** (WRQ) or **data** packet and receiving a positive reply – an **acknowledgment** packet. If connection is established after **write request** packet, the file will be sent, otherwise if the connection is triggered by **data** packet, the text message will be sent. An **acknowledgment** packet contains the fragment number of the data packet being acknowledged. Each **data** packet has associated with it a fragment number; fragment numbers are consecutive and begin with one. Since the positive response to a **write request** is an **acknowledgment** packet, in this special case the fragment number is zero. Zero is also used to acknowledge the **keep connected request** packets**.**
After **write request** packet was sent, the sending side receives **keep connected request** until user on the receiving side enters the path where the file has to be saved.

Data transmission

If the receiver grants the request, the connection is opened and the file or message is sent by chunks using **data** packets with the maximum size specified by user. However, the fragment cannot be larger than 1425 bytes, because otherwise, the packet will be fragmented on link layer of OSI model. Each **data** packet contains one fragment of data and must be acknowledged by an **acknowledgment** packet before the next packet can be sent.

When the file was successfully sent, the keep-alive process is performed. While connection is keeping, the sender or receiver can start sending a new file or text message by sending the **write request** or **data** packet respectively. If the receiving side starts sending the file or text message, the roles are automatically switched. The sending side becomes receiving and receiving side becomes sending.

The transmission is considered as successful when the keep connected request was received by the receiver side.

Termination

In normal case the connection on sending side is terminated automatically when a predefined number of retransmissions was exceeded. On receiving side, the connection is terminated if it does not get any packets from the sending side in [timeout * (max number of retransmissions+2)] seconds.

When the error on one of the sides occurs, it sends the **error** packet with error code and the connection is terminated on both sides. This packet is not acknowledged, and not retransmitted, so the other end of the connection may not receive it. Therefore, timeouts and number of retransmissions are used to detect such a termination when the error packet has been lost.

Andrii Rybak
ID: 105840

Code description

This protocol will be implemented in python 3.9 in IDE PyCharm.

Libraries that are planned to be used:

os.path – to check the existence of file and path

socket – to work with network (data sending/receiving)

bitarray or **bitstring –** to work with bits (needed to calculate the CRC)

libscrc (probably) – to calculate CRC

Classes
Packet – contains all fields of the protocol packet and methods such as encode, calc_checksum.

Functions
packet_decode – converts byte string to instance of class Packet

packet_encode – converts instance of class Packet to byte string (needed for sending)

crc16 – calculate checksum for the data given as argument

role_send – function that controls the sending process

role_receive – function that controls the receiving process

Conclusion

- The whole solution will be implemented in python using libraries mentioned above.
- Developed program turned out to be peer-to-peer, what means the sides switch the roles automatically.
- The protocol header consists of operation code, fragment number, data (optionally) and checksum in the end.
- To verify the correctness of the transmitted packet is used CRC-16/BUYPASS, that provides high reliability and efficiency.
- Receiving side supports sending positive and negative acknowledges.
- Both sides keep connection until it will be terminated by one of the sides.
- If the packet was lost or damaged during transmission it will be automatically resent.
- Program provides the possibility to transmit files and text messages.

Andrii Rybak
ID: 105840

Changes that occurred during implementation

Header

The size of fragment field was decreased and is 3 Bytes now, so the maximum data field size is 1426 against 1425 in design.

Added two more opcodes:

opcode	operation
7	Switch (SWITCH)
8	Finish (FIN)

Peer-to-peer

The implemented communication is no more peer-to-peer

Keep-alive

No changes here. Only description was extended:

After file was successfully transmitted, sending side sends keep-alive packet, and waits for ACK. When the ACK was received, the next keep-alive packet is sent after timeout. The process continues until the connection is terminated. (Note that keep-alive packet in this documentation is called “Keep connected request”)

The keep-alive process is normally terminated when one of the sides sends packet with FIN opcode. Connection on sending side is terminated automatically when a predefined number of retransmissions was exceeded. On receiving side, the connection is terminated if it does not get any packets from the sending side in [timeout * (max number of retransmissions+2)] seconds.

Initial connection

The keep-alive process is not performed during user is entering path to save the file. The timeout is used instead, that is 180 seconds, but can be changed by changing the global variable ENTER_PATH_TIMEOUT

Role switching

The switch is performed by sending packet with SWITCH opcode by receiving side during keep-alive process.

Code

Libraries for work with bits or bytes turned out to be unneeded. Extended list of used libraries:

```
import copy
import socket
import multiprocessing
import time
import libscrc
import sys
import os  # to check existence of paths and files
import select  # to set socket timeout
```

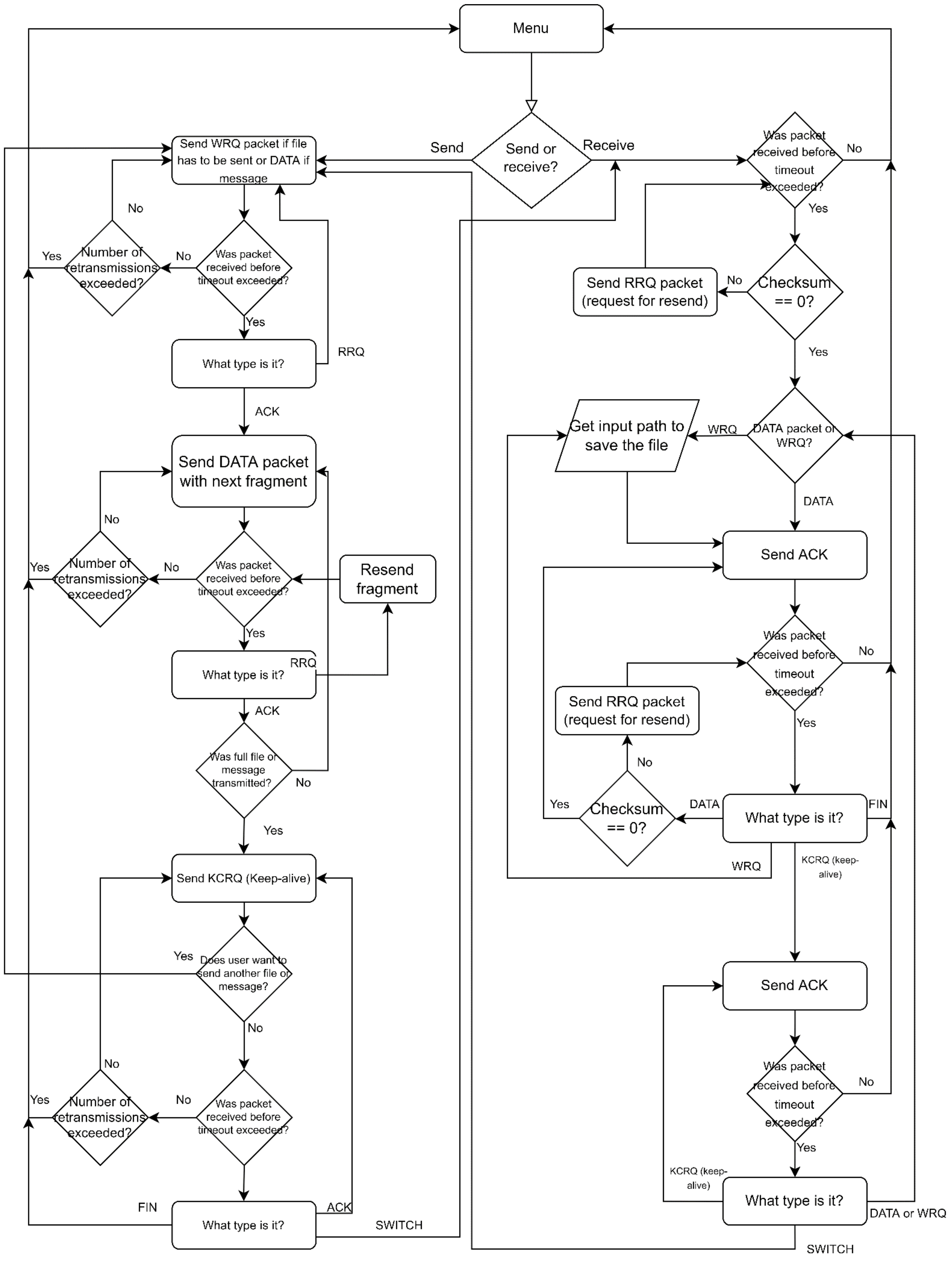
Communication diagram (Picture 2)

- **Completely reworked**

Diagram description above is valid for a new one.

User interface

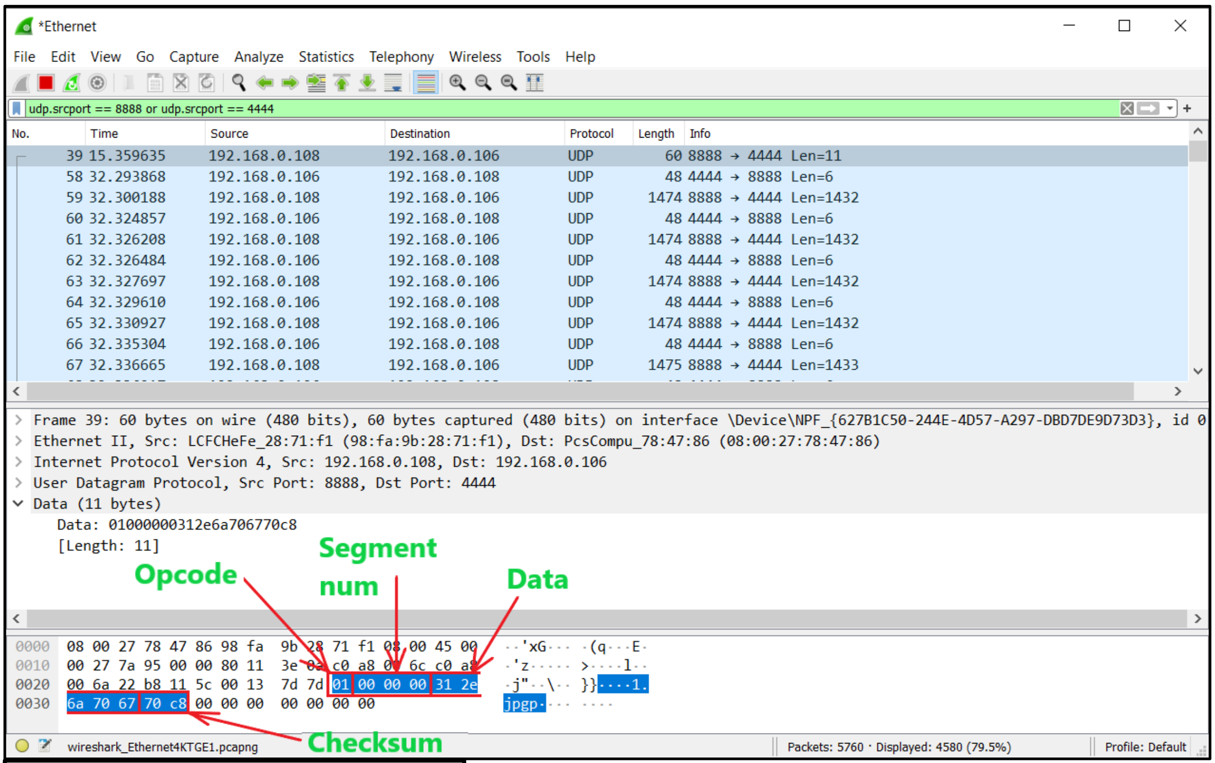
User interface is quite safe and is fully self-explanatory (intuitive). The program is controlled via console.


Menu
Send or receive?
Send
Receive
Send WRQ packet if file has to be sent or DATA if message
Was packet received before timeout exceeded?
Number of retransmissions exceeded?
No
Yes
What type is it?
RRQ
ACK
Send DATA packet with next fragment
Resend fragment
Was full file or message transmitted?
Send KCRQ (Keep-alive)
Does user want to send another file or message?
FIN
SWITCH
Checksum == 0?
Send RRQ packet (request for resend)
DATA packet or WRQ?
WRQ
DATA
Get input path to save the file
Send ACK
KCRQ (keep-alive)
DATA or WRQ

Picture 2. Communication diagram

Andrii Rybak
ID: 105840

Pictures from Wireshark


*Ethernet
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
udp.srcport == 8888 or udp.srcport == 4444
Opcode
Segment num
Data
Checksum

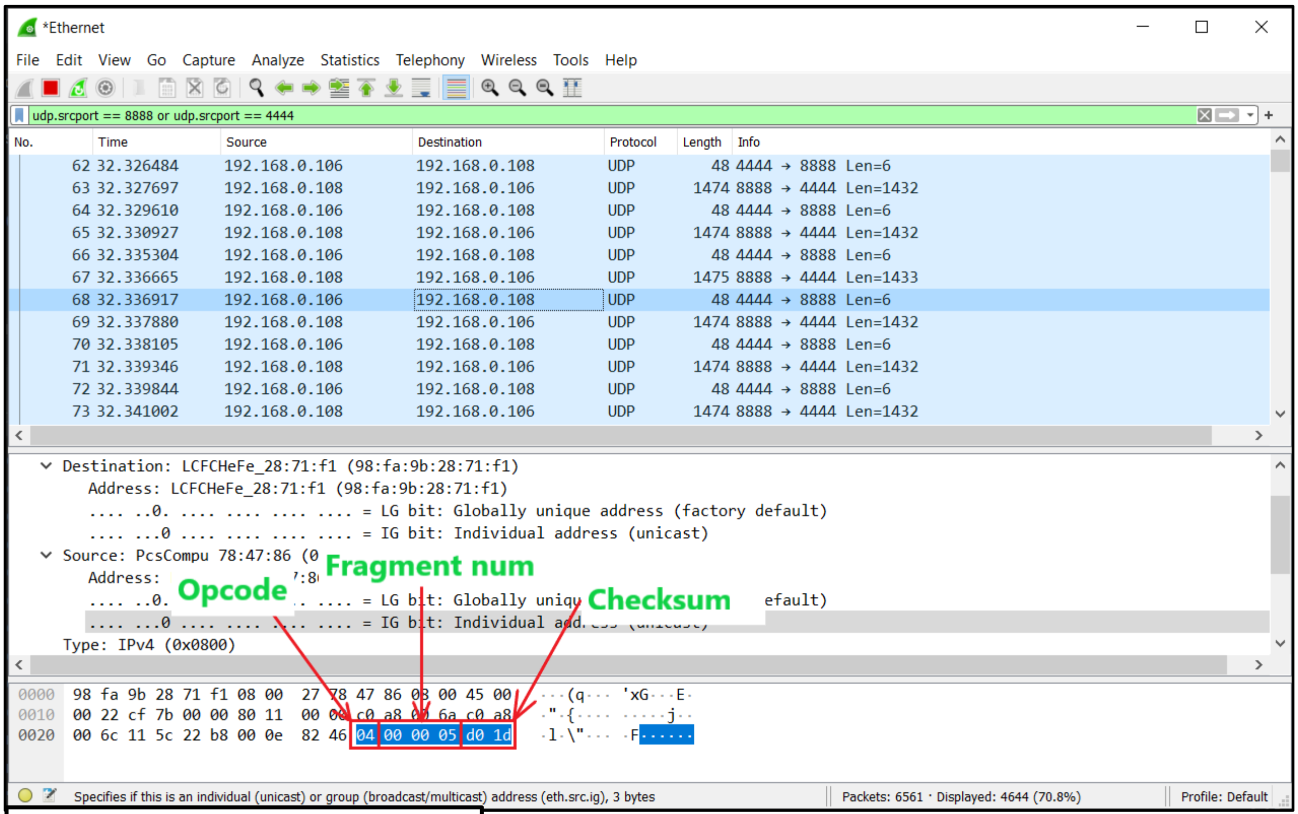
```
No.  Time        Source         Destination    Protocol Length Info
  39 15.359635   192.168.0.108  192.168.0.106  UDP        60 8888 → 4444 Len=11
  58 32.293868   192.168.0.106  192.168.0.108  UDP        48 4444 → 8888 Len=6
  59 32.300188   192.168.0.108  192.168.0.106  UDP      1474 8888 → 4444 Len=1432
  60 32.324857   192.168.0.106  192.168.0.108  UDP        48 4444 → 8888 Len=6
  61 32.326208   192.168.0.108  192.168.0.106  UDP      1474 8888 → 4444 Len=1432
  62 32.326484   192.168.0.106  192.168.0.108  UDP        48 4444 → 8888 Len=6
  63 32.327697   192.168.0.108  192.168.0.106  UDP      1474 8888 → 4444 Len=1432
  64 32.329610   192.168.0.106  192.168.0.108  UDP        48 4444 → 8888 Len=6
  65 32.330927   192.168.0.108  192.168.0.106  UDP      1474 8888 → 4444 Len=1432
  66 32.335304   192.168.0.106  192.168.0.108  UDP        48 4444 → 8888 Len=6
  67 32.336665   192.168.0.108  192.168.0.106  UDP      1475 8888 → 4444 Len=1433

> Frame 39: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface \Device\NPF_{627B1C50-244E-4D57-A297-DBD7DE9D73D3}, id 0
> Ethernet II, Src: LCFCHeFe_28:71:f1 (98:fa:9b:28:71:f1), Dst: PcsCompu_78:47:86 (08:00:27:78:47:86)
> Internet Protocol Version 4, Src: 192.168.0.108, Dst: 192.168.0.106
> User Datagram Protocol, Src Port: 8888, Dst Port: 4444
v Data (11 bytes)
     Data: 01000000312e6a706770c8
     [Length: 11]

0000  08 00 27 78 47 86 98 fa  9b 28 71 f1 08 00 45 00   ··'xG··· ·(q···E·
0010  00 27 7a 95 00 00 80 11  3e 0a c0 a8 00 6c c0 a8   ·'z····· >····l··
0020  00 6a 22 b8 11 5c 00 13  7d 7d 01 00 00 00 31 2e   ·j"··\·· }}····1.
0030  6a 70 67 70 c8 00 00 00  00 00 00 00               jpgp···· ····

wireshark_Ethernet4KTGE1.pcapng    Packets: 5760 · Displayed: 4580 (79.5%)    Profile: Default
```

Picture 3. Write Request packet


*Ethernet
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
udp.srcport == 8888 or udp.srcport == 4444
Fragment num
Opcode
Checksum

```
No.  Time        Source         Destination    Protocol Length Info
  62 32.326484   192.168.0.106  192.168.0.108  UDP        48 4444 → 8888 Len=6
  63 32.327697   192.168.0.108  192.168.0.106  UDP      1474 8888 → 4444 Len=1432
  64 32.329610   192.168.0.106  192.168.0.108  UDP        48 4444 → 8888 Len=6
  65 32.330927   192.168.0.108  192.168.0.106  UDP      1474 8888 → 4444 Len=1432
  66 32.335304   192.168.0.106  192.168.0.108  UDP        48 4444 → 8888 Len=6
  67 32.336665   192.168.0.108  192.168.0.106  UDP      1475 8888 → 4444 Len=1433
  68 32.336917   192.168.0.106  192.168.0.108  UDP        48 4444 → 8888 Len=6
  69 32.337880   192.168.0.108  192.168.0.106  UDP      1474 8888 → 4444 Len=1432
  70 32.338105   192.168.0.106  192.168.0.108  UDP        48 4444 → 8888 Len=6
  71 32.339346   192.168.0.108  192.168.0.106  UDP      1474 8888 → 4444 Len=1432
  72 32.339844   192.168.0.106  192.168.0.108  UDP        48 4444 → 8888 Len=6
  73 32.341002   192.168.0.108  192.168.0.106  UDP      1474 8888 → 4444 Len=1432

  v Destination: LCFCHeFe_28:71:f1 (98:fa:9b:28:71:f1)
      Address: LCFCHeFe_28:71:f1 (98:fa:9b:28:71:f1)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  v Source: PcsCompu 78:47:86 (0
      Address:                 :8
      .... ..0.            . .... = LG bit: Globally uniqu                  efault)
      .... ...0 .... .... .... .... = IG bit: Individual add
    Type: IPv4 (0x0800)

0000  98 fa 9b 28 71 f1 08 00  27 78 47 86 08 00 45 00   ···(q··· 'xG···E·
0010  00 22 cf 7b 00 00 80 11  00 00 c0 a8 00 6a c0 a8   ·"·{···· ·····j··
0020  00 6c 11 5c 22 b8 00 0e  82 46 04 00 00 05 d0 1d   ·l·\"··· ·F······

Specifies if this is an individual (unicast) or group (broadcast/multicast) address (eth.src.ig), 3 bytes    Packets: 6561 · Displayed: 4644 (70.8%)    Profile: Default
```

Picture 4. Packet without data field (Resend Request packet RRQ)

Conclusion

Generally, there was implemented communication program that completely corresponds to the requirements. Although the communication is performed on UDP protocol, it is reliable (every data packet is acknowledged by receiving side). The ARQ method is Stop-and-wait. This implementation handles network cutoffs by resending packet for a defined number of times. It also keeps the connection after file or message was sent and shows when connection is terminated. The program can send files of any size, because only one fragment is stored at a time. Usage of checksum provides the correctness of transmitted fragments. User can change the listening port on current side, IP address and port of opposite side during one program run and there is no requirement to restart it.