Abd Alrahman Saleh
AIS-ID: 105042

STU FIIT

Slovak University of Technology

Faculty of Informatics and Information Technology

Computer and communication networks

Communication using UDP protocol

Trainer: Ing. Kristián Košťál, PhD.

Field of study: Information Secuirty

Year of studies: 1st year at Engineering

Academic year: 2020/2021

https://github.com/Abdo-Saleh/zadanie2

Contents

https://github.com/Abdo-Saleh/zadanie2

Homework

Design and implement a program using your own protocol over the protocol UDP (User Datagram Protocol) transport layer of the TCP / IP network model.

The program will allow communication of two participants in the local Ethernet network, ie transmission of text messages and any file between computers (nodes).

The program will consist of two parts - transmitting and receiving. The transmitting node sends file to another node on the network. Data is assumed to be lost on the network. If it is sent file larger than user-defined max. fragment size, the transmitting party decomposes file into smaller parts - fragments that it sends separately.

Maximum fragment size the user must be able to set it so that it is not fragmented again on the line layers. If the file is sent as a sequence of fragments, the destination node prints an acknowledgment message fragment with its order and whether it was transmitted without errors.

After receiving the entire file on the destination the node displays a message about its receipt and the absolute path where the received file was saved. The program must include checking for communication errors and re-requesting errors fragments, including both positive and negative confirmation. After transferring the first file at inactivity.

The idle communicator automatically sends a packet to maintain the connection every 10-60s if the user does not end the connection. We recommend solving via actually defined signaling messages

https://github.com/Abdo-Saleh/zadanie2

What are network protocols?

Network protocols are sets of established rules that dictate how to format, transmit and receive data so compute

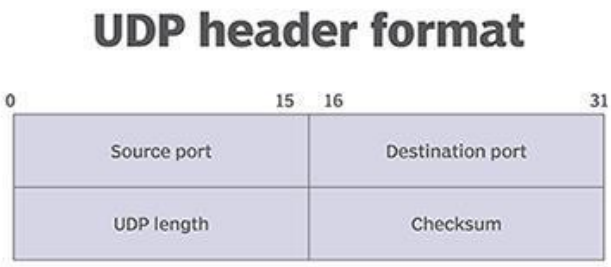
UDP

What is User Datagram Protocol (UDP/IP)?

UDP (User Datagram Protocol) is a communications protocol that is primarily used for establishing low-latency and loss-tolerating connections between applications on the internet.

User Datagram Protocol (UDP) is a **Transport Layer protocol**. UDP is a part of Internet Protocol suite, referred as UDP/IP suite. Unlike TCP, it is unreliable and connectionless protocol. Therefore, there is no need to establish connection prior to data transfer.

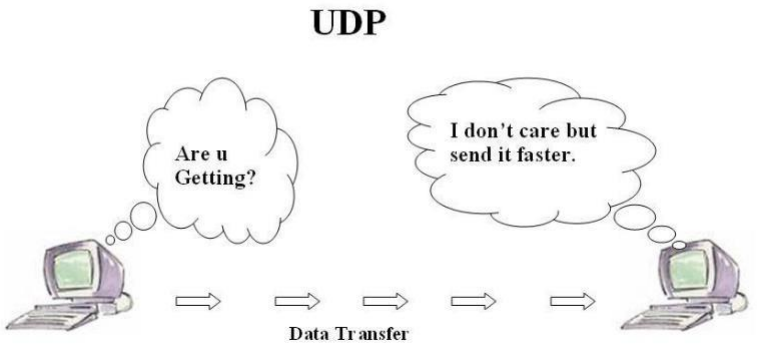
UDP Header



UDP header is 8-bytes fixed and simple header

1. Source Port : Source Port is 2 Byte long field used to identify port number of source.
2. Destination Port : It is 2 Byte long field, used to identify the port of destined packet.
3. Length : Length is the length of UDP including header and the data. It is 16-bits field.
4. Checksum : Checksum is 2 Bytes long field. It is the 16-bit one's complement of the one's complement sum of the UDP header, pseudo header of information from the IP header and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

How does UDP protocol works?



UDP uses IP to get a datagram from one computer to another. UDP works by gathering data in a UDP packet and adding its own header information to the packet. This data consists of the source and destination ports to communicate on, the packet length and a checksum. After UDP packets are encapsulated in an IP packet, they're sent off to their destinations.

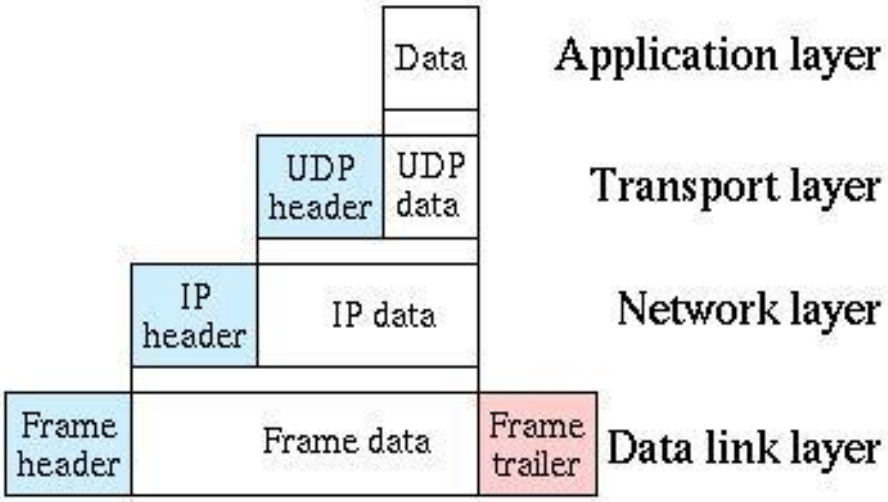
Unlike TCP, UDP doesn't guarantee that the packets will get to the right destinations. That means that UDP doesn't connect to the receiving computer directly -- which TCP does. Rather, it sends the data out and relies on the devices in between the sending and receiving computers to correctly get the data where it's supposed to go.

Most applications that use UDP wait for any replies that are expected as a result of packets sent using the communication protocol. If an application doesn't receive a reply within a certain time frame, the application sends the packet again, or it stops trying.

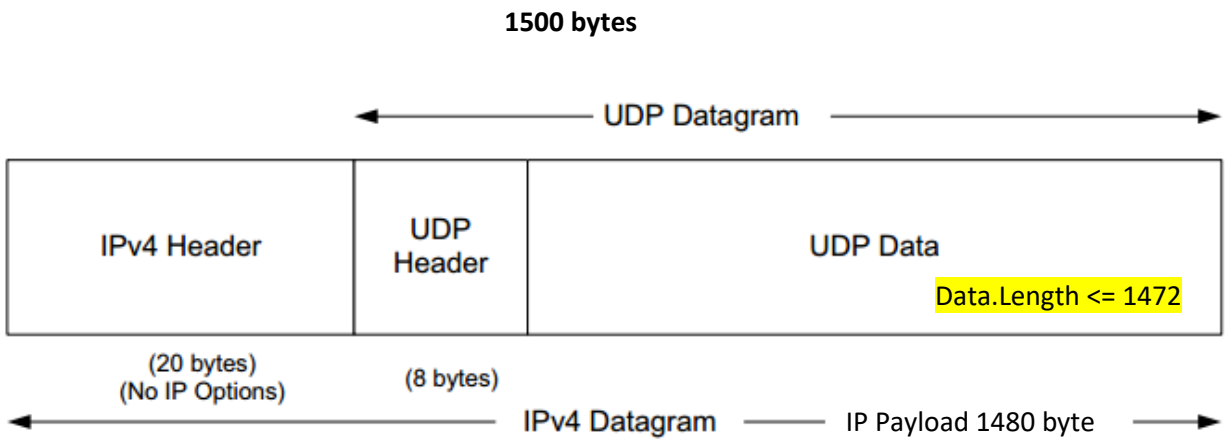
UDP uses a simple transmission model that doesn't include **handshaking** dialogues to provide reliability, ordering or **data integrity**. **Consequently, UDP's service is unreliable**, and packets may arrive out of order, appear to have duplicates or disappear without warning.

Although this transmission method doesn't guarantee that the data being sent will reach its destination, it does have low overhead, and it's popular for services that don't absolutely

UDP Encapsulation



Anatomy in the network layer

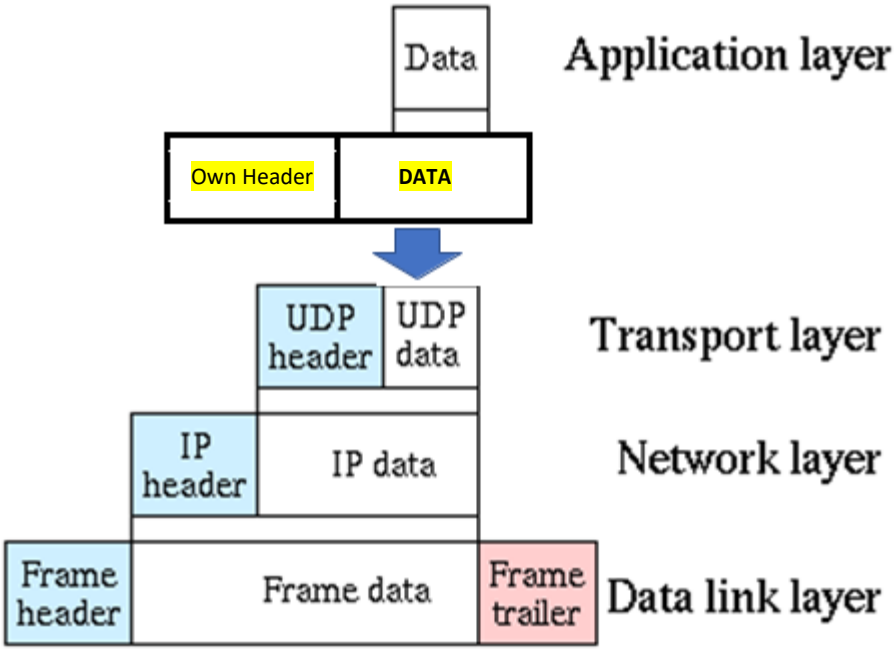


Abd Alrahman Saleh
AIS-ID: 105042

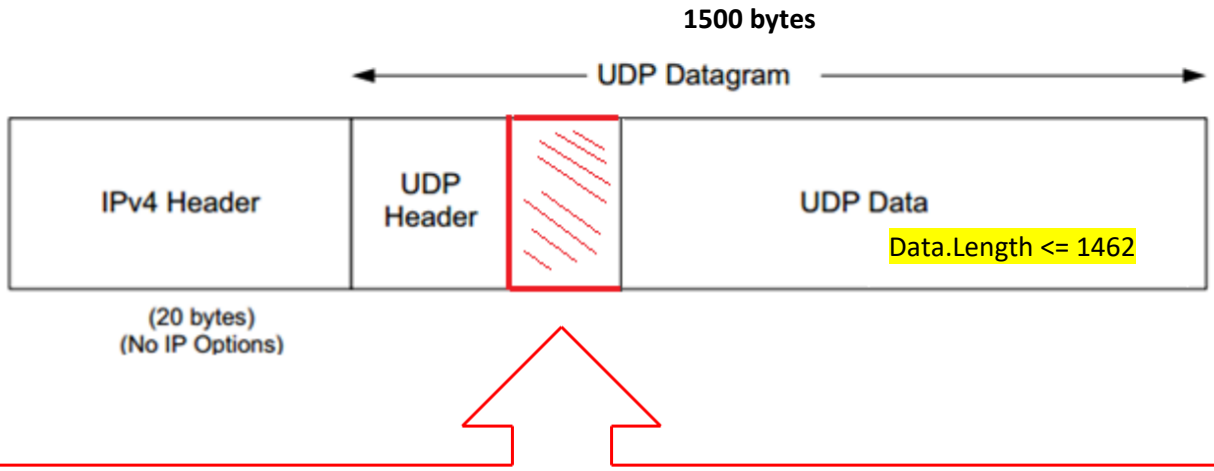
Suggested Solution for the homework

To ensure implementing the homework requirements of data-integrity and consequently, handshaking.

We should add some extra information for our UDP header, our solution should be located on the application layer.



Our own header will look like.



UDP Header extra information (16 bytes)

State	Typ	Seq.No	Fragments.No	Fragments size	Checksum
1B	1B	2B	2B	2B	2B

https://github.com/Abdo-Saleh/zadanie2

Own Header

Status (1B)

Show the state of the current connection.

Value	Command name	Description
0	INVALID	Response to an incorrectly delivered packet
2	SYN	Initialization flag
4	ACK	Acknowledgment flag, response to a correctly delivered packet
12	FIN	Connection termination
14	KEEP-ALIVE	Flag Keep the
16	CLOSE	connection alive.

Type (1B)

Store the type of the being sent data, it could be, FILE, TEXT.

Value	Command name	Description
1	TXT	Text message is being sent
2	File, .txt	File is being sent..
3	File, .pdf	File is being sent..
4	File, .jpg	File is being sent..

Sequence Number (4B)

Maintains information about the order of the fragment.

Fragments Number (2B)

Stores the number of the fragments, it is either one or more based on the size of the being sent data.

Fragments Size (1B)

Stores size of each fragment, **it's not going to be greater than 1462.**

Checksum (3B)

Maintain the data-integrity, and checking errors, which is going to be done by CRC.

What is CRC?

CRC or Cyclic Redundancy Check is a method of detecting accidental changes/errors in the communication channel.
CRC uses Generator Polynomial which is available on both sender and receiver sides.

An example generator polynomial is of the form like x^3 + 1. This generator polynomial represents key 1001. Another example is x^2 + x. that represents key 110.

```
function crc(bit array bitString[1..len], int len) {
    remainderPolynomial := polynomialForm(bitString[1..n])   // First n bits of the message
    // A popular variant complements remainderPolynomial here; see § Preset to -1 below
    for i from 1 to len {
        remainderPolynomial := remainderPolynomial * x + bitString[i+n] * x^0   // Define bitString[k]=0 for k>len
        if coefficient of x^n of remainderPolynomial = 1 {
            remainderPolynomial := remainderPolynomial xor generatorPolynomial
        }
    }
    // A popular variant complements remainderPolynomial here; see § Post-invert below
    return remainderPolynomial
}
```

IMAGE 1 - Simple polynomial division (Pseudo Code)

[Implementation python]():

binascii.crc_hqx(data, value)

Compute a 16-bit CRC value of data, starting with value as the initial CRC, and return the result. This uses the CRC-CCITT polynomial x16 + x12 + x5 + 1, often represented as 0x1021. This CRC is used in the binhex4 format.

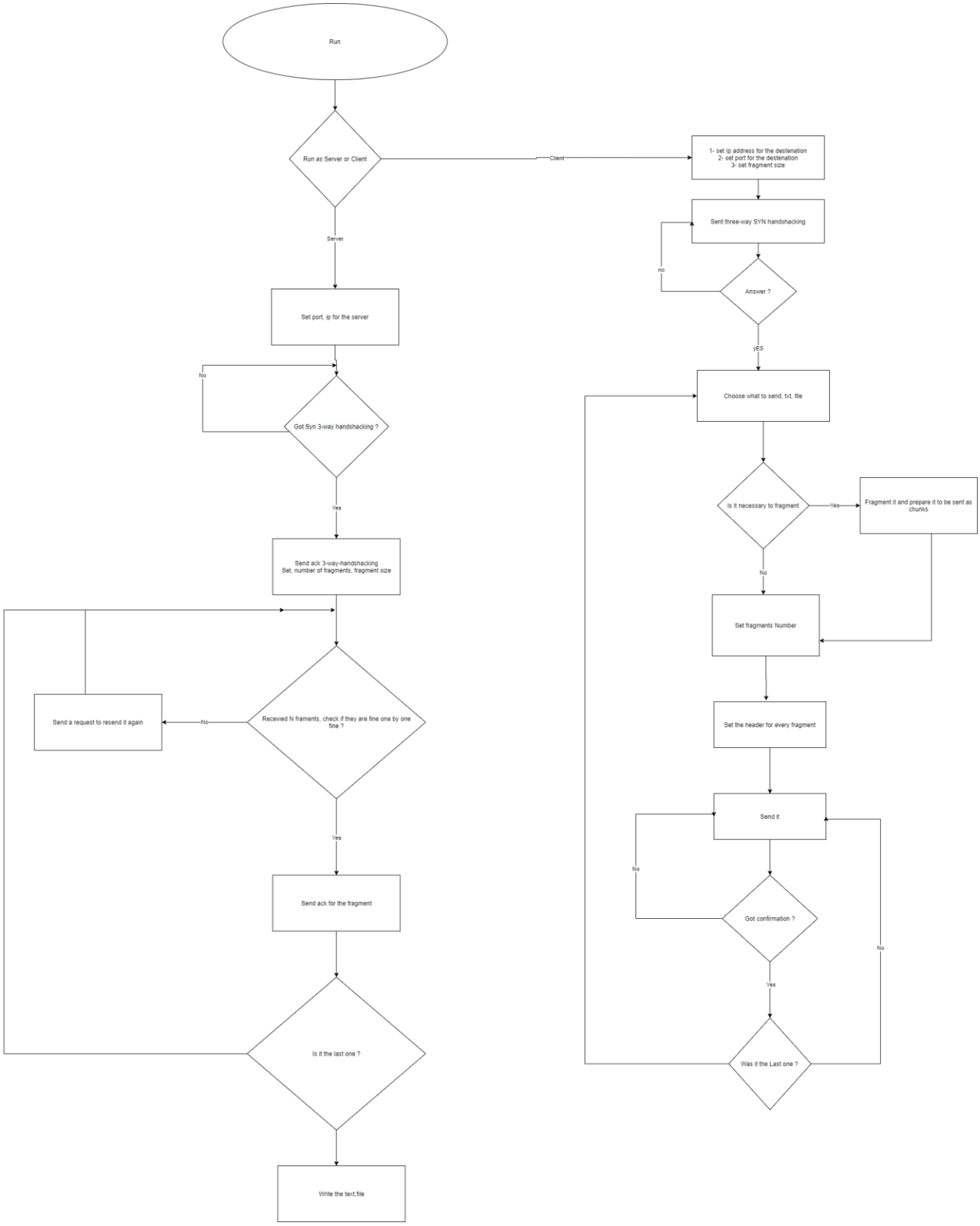


Image 2- Basic diagrams

https://github.com/Abdo-Saleh/zadanie2

Application Features:

1- choose mode (server or client)
2- set IP for the destenation (client mode)
3- set port for the destenation (client mode)
4- set fragment size (client mode)
5- send text smaller than the fragment size
6- send text bigger than the fragment size
7- simulate error of a trasfered packet (small file)
8- logging the fragments states
9- keep alive
10- ARQ stop-wait
11- Server check the coorection of the sent fragment
    a. Ask again for the same fragment in case error-detection
12- CRC 16 bit (check sum)
13- Implemented in python

Libraries:

```
import binascii
import socket as sc
import sys
from struct import *
import time
import math
import os
import threading
```


https://github.com/Abdo-Saleh/zadanie2

Snippts from the code:

```python
def receiverSocket():
    global sock_t
    host = "127.0.0.1"
    port = 12345
    address = (host, port)
    sock = sc.socket(sc.AF_INET, sc.SOCK_DGRAM)
    sock_t = (sock, address)
    sock.bind(address)
    print("Waiting for connection...")
    data, address_r = sock.recvfrom(1500)
    returned_value = (data, address_r)
    if returned_value:
        syn_3_r = header.unpack(data)
        if syn_3_r[0] == 2:
            print('SYN Received')
            syn_ack_3 = header.pack(4, 1, seq, 0, 0, 0)
            sock.sendto(syn_ack_3, address_r)
            print('Sent SYN/ACK')
        ack_3, address_r = sock.recvfrom(1500)
        if ack_3[0] == 4:
            print('Received ACK')
            print('Connection Established')
            print("Waiting to receive packets")
    return sock, address_r
```

It is responsable of preparing the socket for **the sender part** and achive the 3-way handshacking

```python
def senderSocket(host_l, port_l):
    global sock_t
    address = (host_l, int(port_l))
    sock = sc.socket(sc.AF_INET, sc.SOCK_DGRAM)
    sock_t = (sock, address)
    syn_3 = header.pack(2, 1, seq, 0, 0, 0)
    sock.sendto(syn_3, address)
    print("SYN Sent")
    data, address_r = sock.recvfrom(1500)
    returned_value = (data, address_r)
    if returned_value:
        syn_ack_3 = header.unpack(data)
        # if it's ack == 4 and same seq
        if syn_ack_3[0] == 4 and syn_ack_3[2] == seq:
            print("Received a SYN/ACK")
            ack_3 = header.pack(4, 1, seq, 0, 0, 0)
            print("Send ACK ")
            sock.sendto(ack_3, address)
            enable_keep_alive = 1
    return sock, address_r
```

It is responsable of preparing the socket for **the recevier part** and achive the 3-way handshacking

https://github.com/Abdo-Saleh/zadanie2

```python
def run_keep_alive():
    global enable_keep_alive, sock_t
    while True:
        if enable_keep_alive == 1:
            try:
                keep_alive_pckt = header.pack(14, 1, 0, 0, 0, 0)
                sock_t[0].sendto(keep_alive_pckt, sock_t[1])
                # print("\n Alive")
                packet, address_r = sock_t[0].recvfrom(1500)
            except:
                print("sock not init")
        time.sleep(3)


def my_inline_function():
    # do some stuff
    download_thread = threading.Thread(target=run_keep_alive, name="keep alive")
    download_thread.start()
    # continue doing stuff
```

It is responsable of acheving the **keep-alive feature**

```python
wasItSentSuccessfully = False
while not wasItSentSuccessfully:
    if headerInfo[0] == 4:
        # received ack on the sent text
        print('Text has been received at the receiver successfully')
        wasItSentSuccessfully = True
    elif headerInfo[0] == 0:
        print('Text has been received with problems')
        print('The Text is being resent again')
        sock.sendto(packet, address_r)
        packet, address_r = sock.recvfrom(1500)
        headerInfo = header.unpack(packet[:headerSize])
    else:
        print('Text has been received with some thing idk')
```

Example of checking the ack for a certain packet, and send it in case the answer was invalid at the sender side

https://github.com/Abdo-Saleh/zadanie2

```
wasItReceivedSuccessfully = False
while not wasItReceivedSuccessfully:
    if crc_for_received_fragment == headerInfo[5]:
        head = header.pack(4, headerInfo[1], headerInfo[2], headerInfo[3], headerInfo[4],
                           headerInfo[5])
        packet = b"".join([head, fragmentToBeReceived])
        wasItReceivedSuccessfully = True
        sock.sendto(packet, address_r)
        serverBuffer.extend(fragmentToBeReceived)
        print("Sent Ack state for the fragment {}".format(headerInfo[2]))
    else:
        head = header.pack(0, headerInfo[1], headerInfo[2], headerInfo[3], headerInfo[4],
                           headerInfo[5])
        packet = b"".join([head, fragmentToBeReceived])
        sock.sendto(packet, address_r)
        print("Sent invalid state for the fragment {}".format(headerInfo[2]))
```

Example of checking the CRC value for a certain packet, and ask for it again in case the CRC value was not correct at the recevier side

Conclustion:

The program allows two-way communication from the client to the server. It also gives an opportunity set the client's IP address and server port. The program allows you to resize the fragment, during the transfering allows you to simulate error packet transfer.

There is a possibility to send the file as well indication of terminating connection (disconnected client / server).

the protocol proposed by me is sufficiently effective and reliable for this type of message. This assignment taught me to work better with python sockets and threads and showed me many problems that can occur during data transfer and also their subsequent solution.

https://github.com/Abdo-Saleh/zadanie2

References:

KOTULIAK, I.: Počítačové a komunikačné siete, Lokálne počítačové siete podvrstva MAC, Fakulta informatiky a informačných technológií, 2014

• KOTOCiOVAo, M.: Počítačové a komunikačné siete, TCP/IP a RM OSI Transportná až aplikačná vrstva, Fakulta informatiky a informačných technológií, 2013
• https://www.gatevidyalay.com/checksum-checksum-example-errordetection/
• https://www.geeksforgeeks.org/differences-between-tcp-and-udp/
• https://en.wikibooks.org/wiki/Communication_Networks/TCP_and_UDP_Protocols