

Slovak University of Technology in Bratislava

Faculty of Informatics and Information Technologies

Computer and Communication Networks

Assignment 2

Communicator over UDP protocol

Table of Contents

The assignment	3
Design of the program	4
Part 1: Main menu.....	4
Part 2: Server	4
Part 3: Client.....	4
Part 4: Initialization of Communication	4
Part 5: Keep alive	5
Part 6: ARQ	5
Part 7: Checksum.....	5
Part 8: Diagram of communication process	6
Part 9: Programming language, IDE, and libraries.....	6
Design of the communication protocol	7
Implementation	8
PART 1: Preparation.....	8
PART 2: IP and port settings	8
PART 3: Transferring a file smaller than the set fragment size.....	8
PART 4: Simulating a File Transfer Error.....	9
PART 5: Transferring a 2MB file	10
PART 6: Keep alive	11
PART 7: Differences between design and final implementation	11
ARQ Method	11
CRC	11
Client's username	11
Switching between sender and receiver	12
Custom Header	13
Sequence diagram of Communication	14

The assignment

Design and implement an application using a custom protocol over the User Datagram Protocol (UDP) of the transport layer of the TCP/IP network model. The application will enable communication between two participants in a local Ethernet network, i.e., the transfer of text messages and any file between computers (nodes).

The application consists of two parts: transmitting and receiving. The sending node sends the file to another node in the network. It is assumed that there is data loss in the network. If the sent file is larger than the user-defined maximum fragment size, the sending party breaks the file into smaller parts - fragments, which it sends separately. The user must be able to set the maximum fragment size so that they are not fragmented again on the lower layer.

If the file is sent as a sequence of fragments, the destination node reports information about the message and whether the message was transferred without errors. When the entire file is received on the destination node, it will display a message about its reception and the absolute path where the received file was saved. The application must include communication error checking and re-requesting of erroneous fragments, including both positive and negative acknowledgement. After starting the program, the communicator automatically sends a packet to maintain the connection every 5 seconds until the user ends the connection. We recommend solving via self-defined signalling messages and separate thread.


Task num.	Task name	Max points	Min points
1	Design of the program and communication protocol	4	2
2	Preparation	0.5	7.5
3	IP and port settings	0.5	
4	Transferring a file smaller than the set fragment size	2	
5	Simulating a File Transfer Error	4	
6	Transferring a 2MB file	2	
7	Keep alive	3	
8	Final documentation and overall quality	3	
9	Added functionality (additional implementation) directly in the exercise	1	0.5
	Total	20	10

Design of the program

Part 1: Main menu

After the program is executed, the user has the option to designate the current console/terminal as either the server or the client.

Part 2: Server

If the user selects 'server,' the question about port will be prompted. The user can input a specific port number ranging from 1 to 65535. Alternatively, they can press  'Enter' to generate a random port. After the port is either manually entered or randomly generated, the server initiates the listening process, eagerly awaiting connections from clients.

```
Enter your choice: 1
Enter port or press enter to use a random port.
PORT:
Server is running on 192.168.1.109:5
|
```

Figure 1 Server setup

Part 3: Client

If the user opts for 'client,' they need to input three crucial pieces of information: their chosen username, the IP address of the server they wish to connect to, and the server's port number.

```
Enter your choice: 2
Username: David

IP and PORT are required to connect to the server.
IP: 192.168.0.10
Port: 12345
```

Figure 2 Client setup

Part 4: Initialization of Communication

When the client sends an initialization message to the server, it patiently waits for an acknowledgment confirming the successful connection. Upon receiving the initialization message, the server promptly adds the username and client socket to the 'connected_clients' set. However, if the client doesn't receive the acknowledgment message within 5 seconds, it prompts the user to re-enter the IP and Port, indicating a potential error in the provided information. Additionally, the server has a timeout set to 60 seconds after initiating its listening process, preventing it from waiting indefinitely for messages.

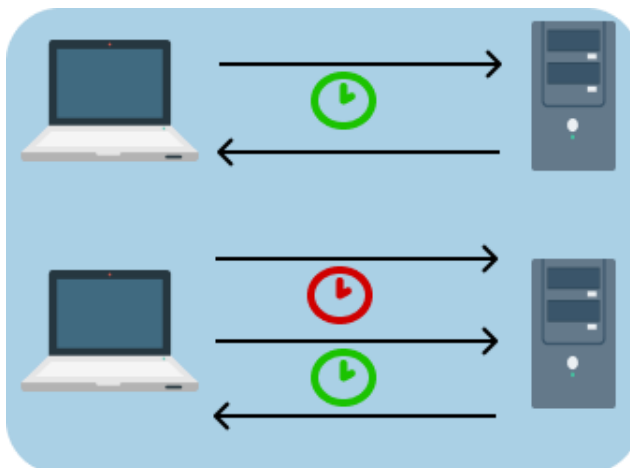


Figure 3 Connection between client and server

Part 5: Keep alive

To ensure active communication and prevent clients from becoming passive for extended periods, a keep-alive method will be implemented. This mechanism will involve the exchange of signaling messages between the server and the client at regular intervals, with a timer set to 60 seconds. This method should be enhancing the overall reliability of the system.

Part 6: ARQ

The main function of this method is to check if the frame was sent correctly. After a frame is sent the ack message must be received from server in the set time interval. If the ack is not received the frame is sent again,

In the program will be used selective repeat method of ARQ. This method is based on the principle of retransmitting only the frames that were not successfully received, optimizing the efficiency of data transfer, and minimizing redundancy in the communication process.

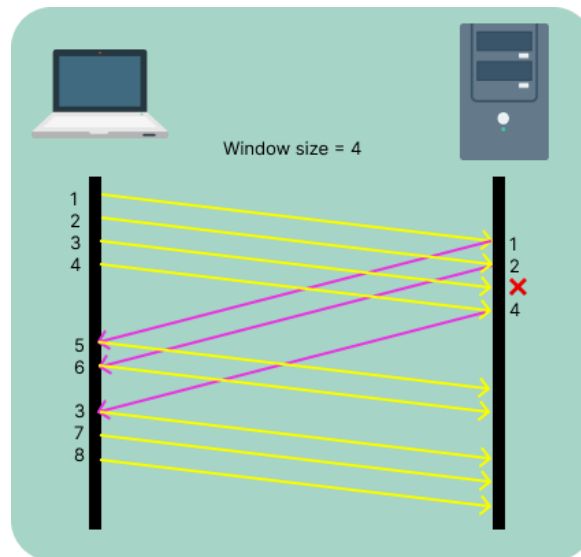


Figure 4 ARQ - Sequence diagram of communication ¹

Part 7: Checksum

The Cyclic Redundancy Check (CRC) employed by zlib² plays a crucial role in ensuring data integrity during transmission. The CRC process involves generating a checksum number based on the data within a frame. This checksum acts as a unique identifier for the frame's content. To calculate the checksum, zlib utilizes a polynomial division algorithm. Each bit in the frame contributes to the calculation, with the resulting checksum appended to the frame before transmission.

Upon receiving a frame, the CRC check is performed to verify its correctness. The recipient uses the same polynomial division algorithm to calculate a checksum from the received frame's data. If the calculated checksum matches the one appended to the frame, the data is considered intact. Any disparity indicates potential errors, prompting the need for frame retransmission to maintain the integrity of the communication.³

¹ Source: https://www.youtube.com/watch?v=WflhQ3o2xow&t=5s&ab_channel=NesoAcademy

² Source: <https://docs.python.org/3/library/zlib.html>

³ Source: <https://copyprogramming.com/howto/zlib-crc32-in-python>

Part 8: Diagram of communication process

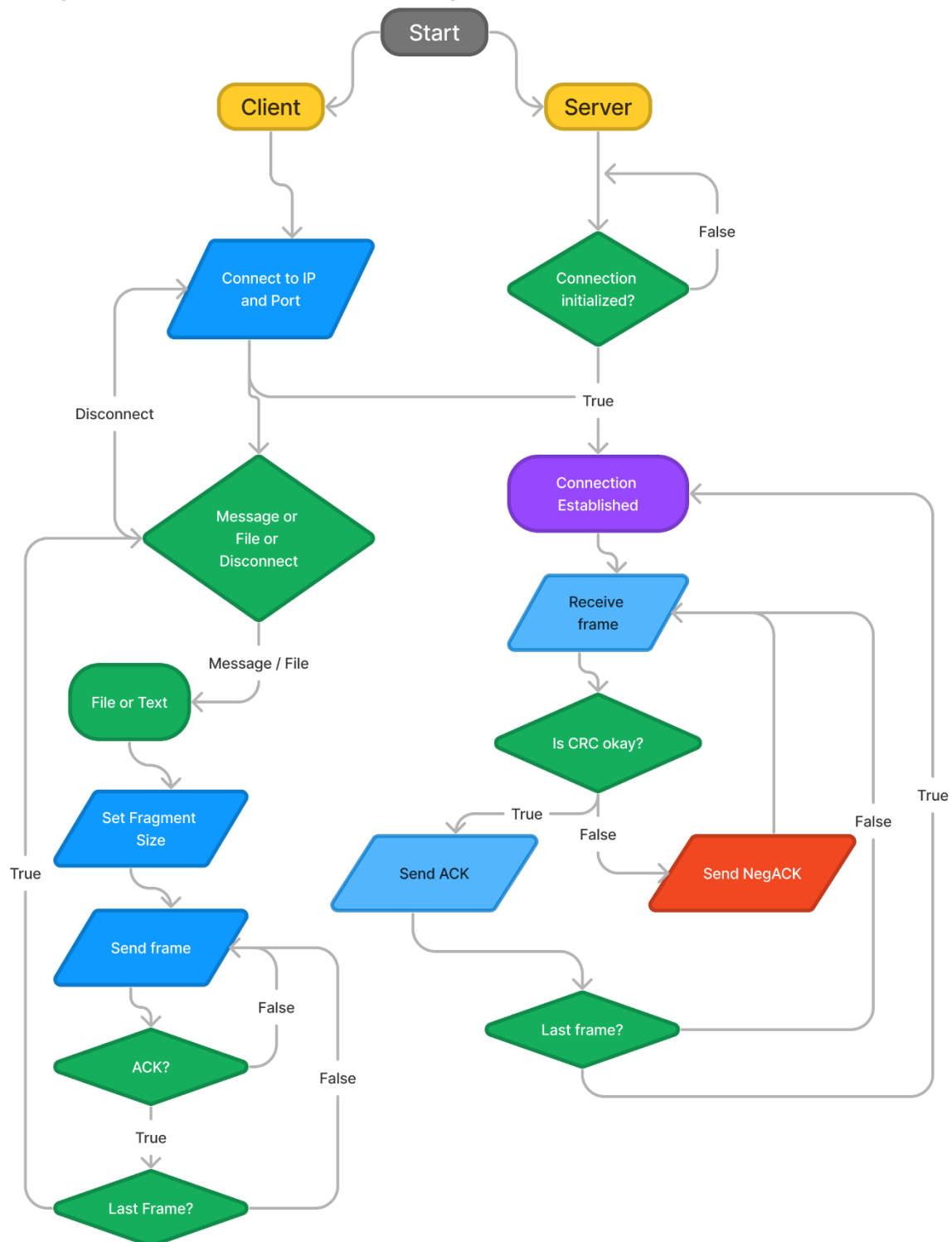


Figure 5 Diagram of communication process

Part 9: Programming language, IDE, and libraries

This program has been developed using Python due to its extensive libraries and functionalities that greatly facilitate the overall process. Additionally, PyCharm 2023.2 serves as the integrated development environment (IDE) of choice, offering an exceptional debug mode and a myriad of functionalities for enhanced development. The utilized libraries include socket, zlib, os, struct, math, and random.

Design of the communication protocol

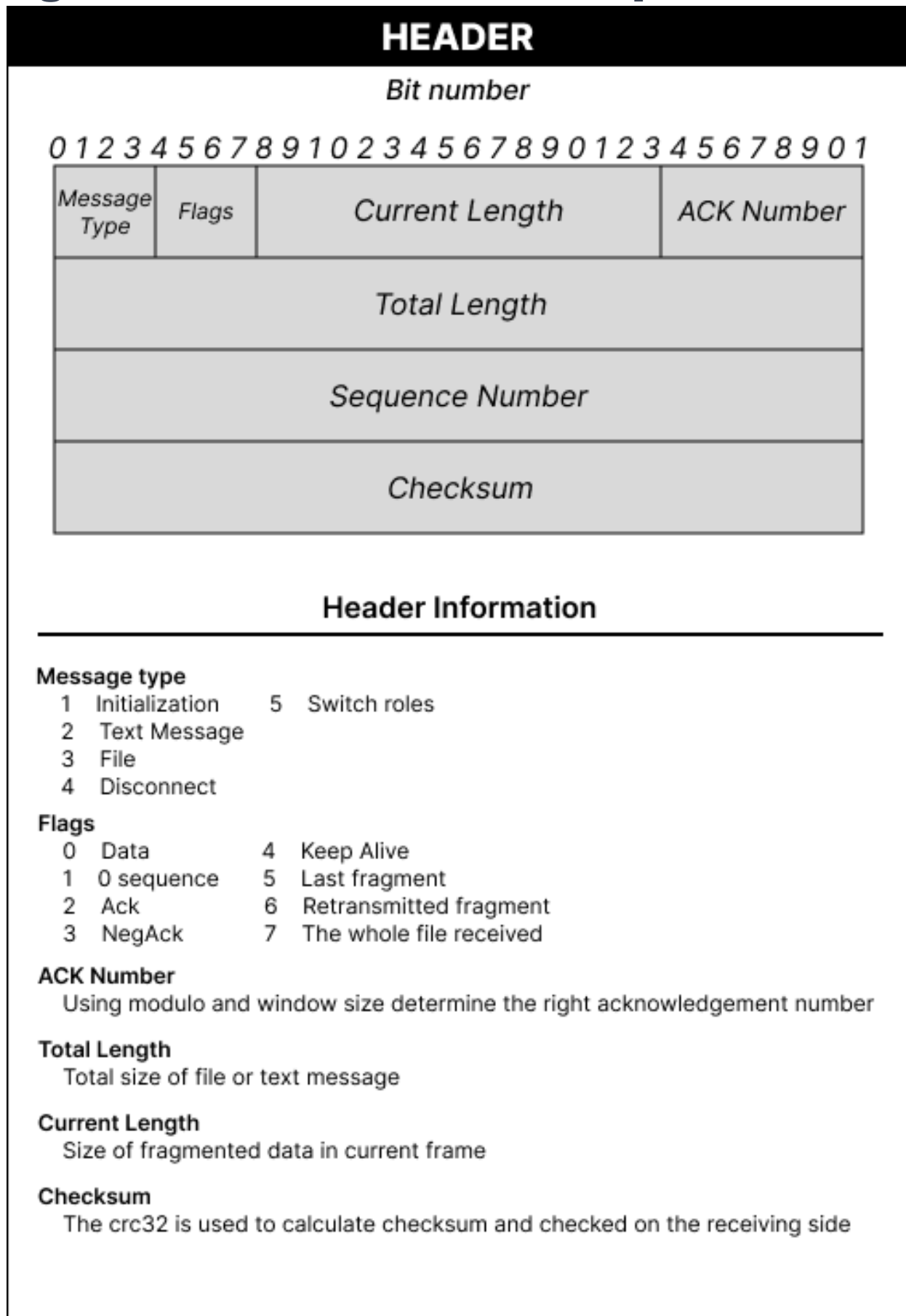


Figure 6 Custom Header Layout

Implementation

PART 1: Preparation

The initial phase of our project involves the establishment and verification of connectivity between two nodes. This is accomplished by utilizing Wireshark to monitor and analyze the communication between these nodes.



Figure 7 Filter used in Wireshark.

After applying the specified filter, the communication can be analyzed. The first step is the establishment of a connection between the client and server, with the message type being "I Initialization."

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	169.254.229.253	169.254.237.159	PKS	60	57661 → 65001 Len=8
2	0.001103	169.254.237.159	169.254.229.253	PKS	50	65001 → 57661 Len=8

PKS

Type: 1

Flags: 0

Number of Fragments: 0

Sequence Number: 0

Checksum: 0

Data:

Figure 8 Connection establishment.

PART 2: IP and port settings

The program allows the user to set the port number on the server side, where the application listens. Additionally, the user can configure the IP address and port number of the receiver on the client (transmitting) side.

```
1. Server | 2. Client | 3. Exit
Enter your choice: 1
Enter port or press enter to use a random port.
PORT: 5
Server is running on 169.254.237.159:5
```

Figure 9 Server settings

```
1. Server | 2. Client | 3. Exit
Enter your choice: 2
Your IP address is: 169.254.237.159

IP and PORT are required to connect to the server.
Enter the IP address of the server: 169.254.237.159
Enter the port of the server: 5
Connecting to server...
Connected to ('169.254.237.159', 5).
```

Figure 10 Client settings

PART 3: Transferring a file smaller than the set fragment size

This section provides a brief overview of how a file or message is transmitted when the fragment size is large enough to transfer the entire file/message.

5	3.405087	169.254.229.253	169.254.237.159	PKS	60	57661 → 65001 Len=9
6	3.405974	169.254.237.159	169.254.229.253	PKS	50	65001 → 57661 Len=8

Figure 11 Message transfer - Message is smaller or equal to fragment size

PKS
Type: 2
Flags: 5
Number of Fragments: 1
Sequence Number: 1
Checksum: 9842
Data: 1

Figure 12 Frame No.5 - Flag "5 Last fragment".

PKS
Type: 2
Flags: 7
Number of Fragments: 0
Sequence Number: 1
Checksum: 0
Data:

Figure 13 Frame No.6 - Flag "7 last fragment received".

PART 4: Simulating a File Transfer Error

The program enables users to simulate errors during message or file transfers. Although the design initially mentioned ARQ with selective repeat, the final implementation features ARQ with stop and wait due to time constraints. In this method, the sender waits for an acknowledgment message from the receiver before sending another frame.

Error probability can be adjusted through the variable named `ERROR_PROBABILITY` (currently set to 5).

```
ERROR_PROBABILITY = 5
```

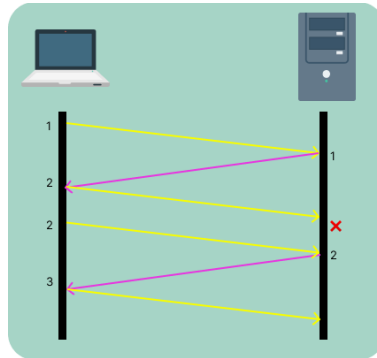


Figure 14 ARQ - Stop & Wait implemented in program.

Error in program is simulated with corrupting the crc value before sending fragment. If error is '1' then the crc value is increased by 1.

```
# Simulate a corrupted message
error = 1 if random.randint(1, 100) <= ERROR_PROBABILITY else 0

# Error simulation for testing - add 1 to crc
if error == 1:
    data_crc = data_crc + 1
```

Error simulation involves corrupting the CRC value before sending a fragment. If an error occurs, the CRC value is increased by 1. After sending a corrupted message, the receiver calculates the CRC and compares it with the checksum from the frame. If unequal, a NegAck (Flag "3") message is sent back to the sender, prompting the resend of the message/fragment.

```
Fragment 1 received. 2 fragments remaining. Size 2B
Checksums do not match
Checksums do not match
Fragment 2 received. 1 fragments remaining. Size 2B
Fragment 3 received. 0 fragments remaining. Size 2B
('192.168.1.109', 60100) sent a file. Data length: 6B
The file has been saved at: C:\David\FIIT\3. Semester\P
```

Figure 15 Corrupted fragments at receiver side.

5	4.942064	192.168.1.109	192.168.1.109	PKS	49	60100 → 65001 Len=17
6	4.942188	192.168.1.109	192.168.1.109	PKS	42	60100 → 65001 Len=10
7	4.942300	192.168.1.109	192.168.1.109	PKS	40	65001 → 60100 Len=8
8	4.942555	192.168.1.109	192.168.1.109	PKS	42	60100 → 65001 Len=10
9	4.942693	192.168.1.109	192.168.1.109	PKS	40	65001 → 60100 Len=8
10	4.942867	192.168.1.109	192.168.1.109	PKS	42	60100 → 65001 Len=10
11	4.942972	192.168.1.109	192.168.1.109	PKS	40	65001 → 60100 Len=8
12	4.943121	192.168.1.109	192.168.1.109	PKS	42	60100 → 65001 Len=10
13	4.943233	192.168.1.109	192.168.1.109	PKS	40	65001 → 60100 Len=8
14	4.943446	192.168.1.109	192.168.1.109	PKS	42	60100 → 65001 Len=10

Figure 16 Captured error - Flag "3"

```
PKS
Type: 3
Flags: 3
Number of Fragments: 0
Sequence Number: 0
Checksum: 0
Data:
```

Figure 17 NegAck

PART 5: Transferring a 2MB file

To complete this section, a 2 MB file named 2MB.txt is generated. The program was tested on *.pdf files as well. On the sender side menu, choice 2. Send File must be selected, followed by specifying the fragment size and file path.

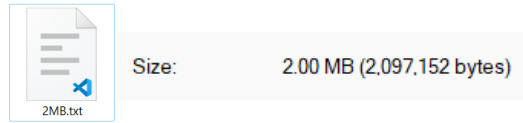


Figure 18 2MB File to transfer

On the receiver side, after receiving all fragments, the original file is reconstructed and saved to the folder `"/server_downloads /"`.

```
Choose an option:      1. Send text    2. Send file
Enter your choice:
Keep alive...
Keep alive...
2
Enter fragment size (0 - 1464): 1000
Enter the path of the file to send: 2MB.txt
```

Figure 19 Sending fragmented file

On the receiver side

```
('192.168.1.109', 62169) connected to the server.
Keep alive...
Keep alive...
Fragment 1 received. 2097 fragments remaining
Fragment 2 received. 2096 fragments remaining
Fragment 3 received. 2095 fragments remaining
```

...

```
Fragment 2097 received. 1 fragments remaining
Fragment 2098 received. 0 fragments remaining
('192.168.1.109', 62169) sent a file. Data length: 2097152B
The file has been saved at: C:\David\FIIT\3. Semester\Pocitacove a Komunikacne Siete\Zadania\Zadanie 2\sent_files\2MB.txt
Keep alive...
```

Figure 20 Receiving fragmented file

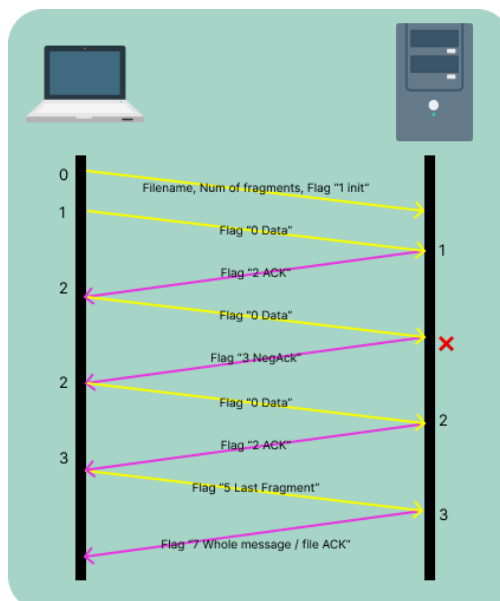


Figure 21 Sending file

PART 6: Keep alive

An essential part of this assignment is the implementation of a method that constantly checks if the connection is still alive. Multithreading is used for this purpose. The thread sends a frame to the receiver every 5 seconds with the flag "4 – Keep alive" and waits for a response. If no response is received within 5 seconds, a warning is printed, and after 5 failed attempts, the client disconnects.

Additionally, the keep alive thread is used to switch between receiver and sender. If the response to the keep alive message is a switch message (Type "5", Flag "4"), the client knows that the server wants to switch.

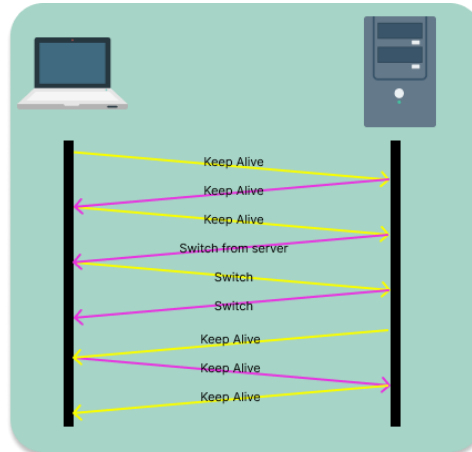


Figure 22 Keep alive and switch from server side

PART 7: Differences between design and final implementation

As the project progressed, certain adjustments were made to ensure the successful completion of the program.

ARQ Method

The original design proposed the Selective Repeat ARQ method, but due to time constraints, the final implementation adopted the simpler ARQ Stop & Wait.

To examine differences, see Figure 4 ARQ - Sequence diagram of communication and Figure 14 ARQ - Stop & Wait implemented in program.

CRC

In the design, CRC32 from the zlib library was initially chosen. However, the assignment required an efficient header, leading to the implementation of CRC16 from the binascii library instead. This choice resulted in a header size reduction by 2 bytes. The CRC16 from binascii utilizes a fixed mask recommended by the CCITT organization:

$$x^{16} + x^{12} + x^5 + 1$$

Client's username

Due to various challenges encountered during the process of switching between the sender and receiver with the username, the final version of the program excludes this functionality. The decision was made to streamline the implementation and address the identified issues.

Switching between sender and receiver

In the final implementation, a crucial functionality was introduced, allowing seamless switching between the sender and receiver roles. On the client side, an additional option was incorporated into the menu.

```
Choose an option:  1. Send text  2. Send file  3. Disconnect  4. Switch
Enter your choice:
```

Figure 23 Switch from sender side

The server-side implementation involved the addition of keyboard library and function `keyboard.add_hotkey()`. If the combination 'ctrl + s' is pressed on keyboard, server send it's answer to Keep alive message with message type "5 – switch" and flags "4 – keep alive". Then the sender starts normal switch method.

```
('147.175.161.145', 51374) connected to the server.
Keep alive SERVER
Switching to CLIENT...

Choose an option:  1. Send text  2. Send file  3. Disconnect  4. Switch
Enter your choice:
```

Figure 24 Switch from receiver side

```
Choose an option:      1. Send text  2. Se
Enter your choice:
Keep alive CLIENT
Press enter to confirm switch to SERVER...
```

Figure 25 Clients has to confirm, that he wants to switch too with enter

This enhancement in functionality provides a dynamic and interactive element to the program, allowing users to effortlessly transition between the sender and receiver roles.

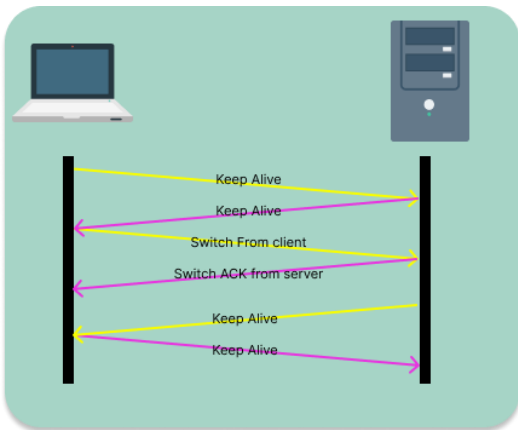


Figure 26 Switch from sender side

PKS	40 53509 → 65001 Len=8
PKS	40 65001 → 53509 Len=8
PKS	40 53509 → 65001 Len=8
PKS	40 65001 → 53509 Len=8
PKS	40 65001 → 53509 Len=8
PKS	40 53509 → 65001 Len=8

Figure 27 Captured switch from Figure 26

Custom Header

In the final implementation, a significant modification was made to the custom header used for communication. This adjustment was driven by the assignment's requirement for the efficient utilization of space, resulting in a reduction of the header size from 16 Bytes to 8 Bytes.

With the introduction of the new CRC and ARQ method, certain fields in the header became redundant. Specifically, the fields Total Length, Current Length, and ACK number were omitted. The Checksum field was reduced to 2 Bytes, and the Sequence number was also reduced to 2 Bytes.

This optimization not only aligns with the space-efficiency requirement but also contributes to a more streamlined and resource-effective communication process within the program.

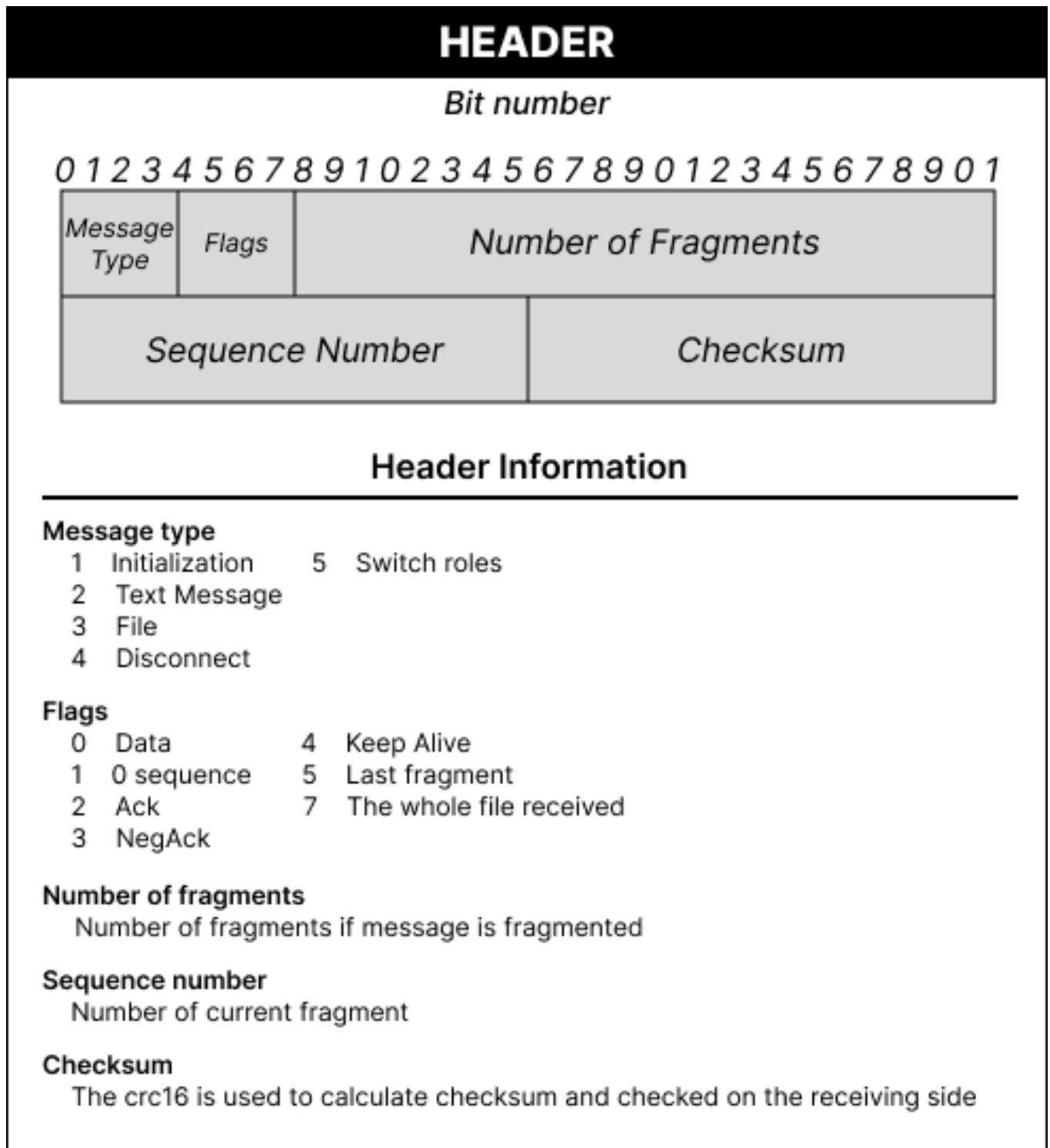


Figure 28 Final implementation custom header layout

Sequence diagram of Communication

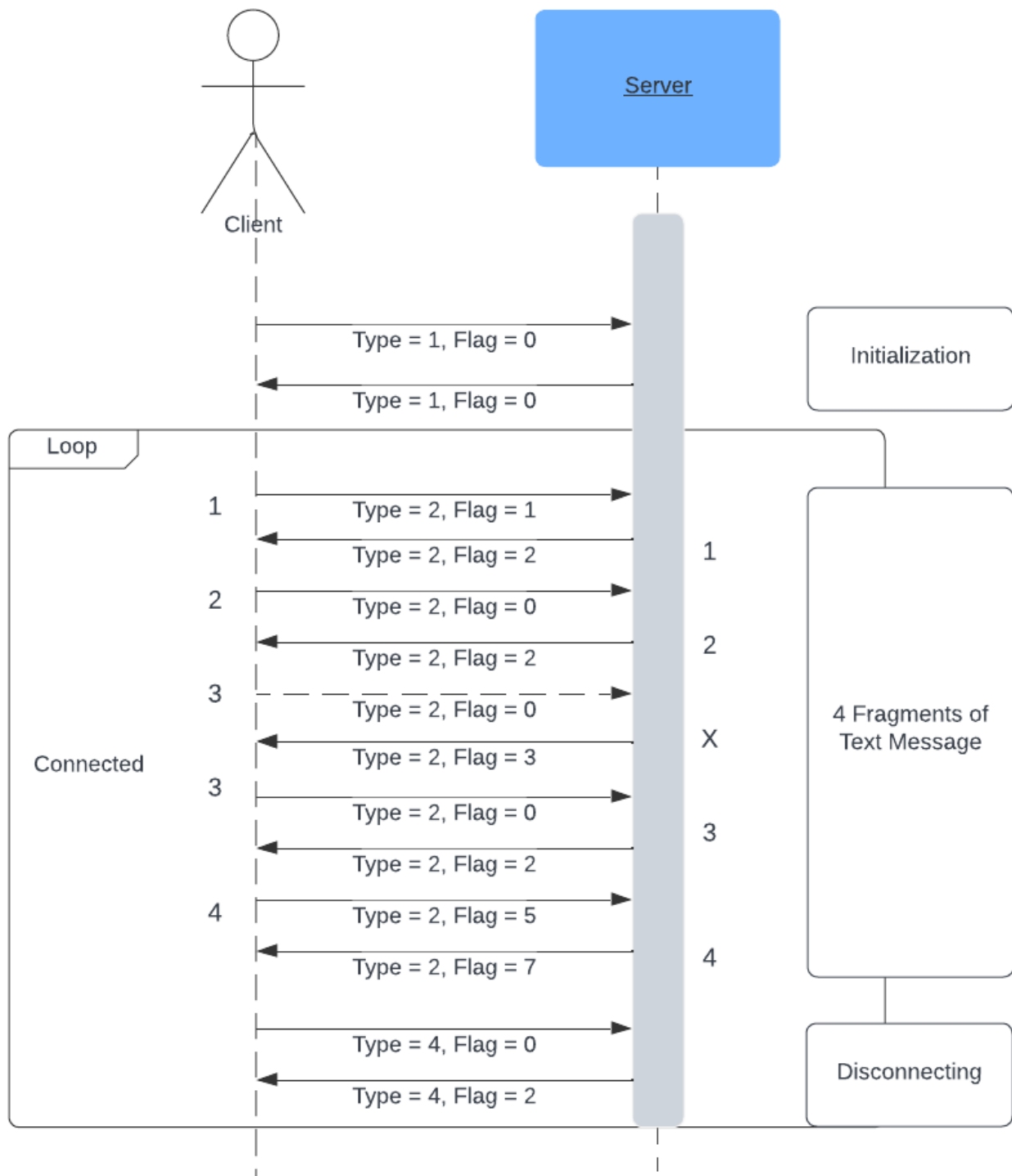


Figure 29 Sequence Diagram of Communication