

STU FIIT

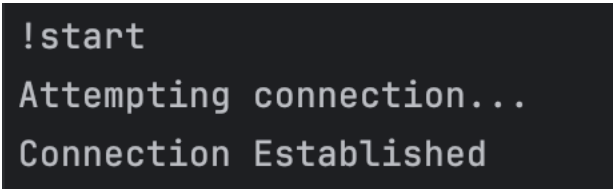
Počítačové a komunikačné siete

“Komunikácia s využitím UDP protokolu”

Finálne odovzdanie

Made by: Meredov Nazar
Cvičiaci: Ing. Ladislav Zemko
AIS ID: 122092

Obsah

Obsah	2
Štruktúra hlavičiek protokolu	3
Schéma:	3
Popis:	3
HandShake:	5
Keep-Alive:	6
CheckSum:	6
Fragmentacia:	7
ARQ (SR):	8
Lua Script:	10
WireSharkColor:	12
Ako spustiť program:	13
	14
Working Diagram	16
Záver a výsledky:	17

Štruktúra hlavičiek protokolu

Schéma:

Bytes	1	2	2	1	Data
Bits	8	8	12	8	
	Flags	Checksum	SeqNum	TimeStamp	PayLoad

Popis:

Pole Flags predstavuje 2 typy flagov - MessageType a SendType (4 a 4 bity)

MessageType:

[1] = "Control Packet"

[2] = "Single Message Packet"

[3] = "Multi-Fragment Message Packet"

[4] = "File Transfer Packet"

[5] = "ACK/NACK Packet"

SendType:

[1] "Control Packet"

[2] = "SYN"

[3] = "SYN-ACK"

[4] = "KEEP_ALIVE"

[5] = "KEEP_ALIVE-ACK"

[8] = "FIN"

[9] = "FIN-ACK"

[2] "Single Message Packet"

[1] = "Single Message"

[5] = "All Fragments Send"

[3] "Multi-Fragment Message Packet"

[2] = "Fragments size Transfer"

[4] = "Fragment Message"

[4] "File Transfer Packet"

[1] = "Filename Transfer"

[2] = "File Size Transfer"

[3] = "Fragment Count Transfer"

[4] = "File Fragment part"

[6] = "Look missing fragment"

[5] "ACK/NACK Packet"

[1] = "ACK"

[2] = "NACK"

Checksum:

- Používa algoritmus CRC-16-CCITT
 - Pracuje s každým bajtom užitočného zaťaženia správy
 - Umožňuje detekciu chýb prenosu
- Ukladá súčet bajtov našej správy

SeqNum - sa používa počas fragmentácie a ukladá číslo sekvenčného čísla fragmentu

TimeStamp - sa používa počas ARQ a vysiela čas, v ktorom má byť paket potvrdený.

PayLoad - prenášané údaje

HandShake:

Implementované v súbore „sendThread.py a receiveThread.py“:

1) Nastavenie pripojenia

Postupnosť:

1. Odosielanie SYN:

2. odosielanie SYN

3. odpoveď SYN-ACK

4. Nadviazanie spojenia

5. Pravidelná kontrola stavu

2) Prerušenie spojenia

Postupnosť:

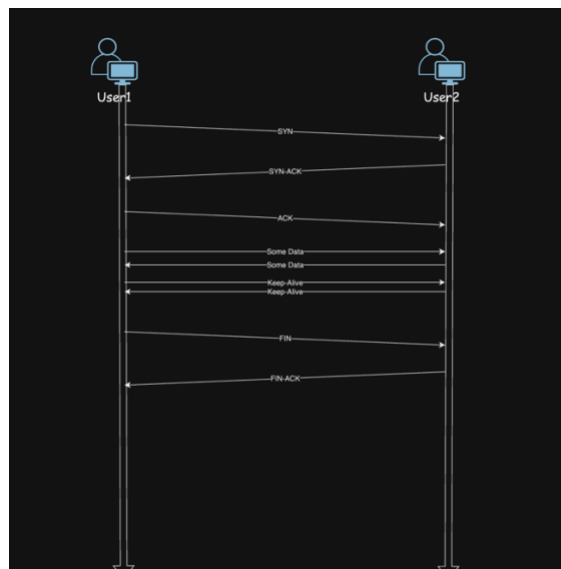
1. odoslanie FYN

2. odpoveď FYN-ACK

3. Prerušenie spojenia

```
synMSG = manager( msgType: 1, flags=2)
print("To start talking, type !start")
```

```
# control messages
if parsedMessage['msgType'] == 1:
    # Handle control messages
    with controlThread.connection_lock:
        if parsedMessage['flags'] == 2:
            controlThread.hasConnectionToPeer = True
            response = manager( msgType: 1, flags=3)
            sendMSG(sock, response, ip, responsePort)
            print(f"A peer has connected: {addr}")
            receiver_window = ReceiverWindow(8)
        elif parsedMessage['flags'] == 3:
            controlThread.hasConnectionToPeer = True
```



HandShake je obojsmerny!

Keep-Alive:

Implementované v controlThread.py a receiveThread.py:

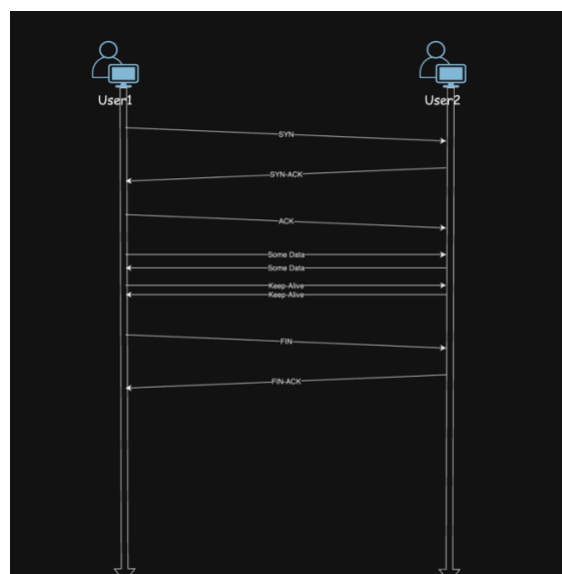
```
message = manager(msgType: 1, flags=4) # keep alive mes.  
sendMSG(sock, message, ip, port)
```

Mechanizmus funguje takto:

Periodické odosielanie riadiacich paketov

Ak nie je prijatá žiadna odpoveď, spojenie sa považuje za stratené

Spracovanie sa vykonáva prostredníctvom globálnych premenných expectingResponse a hasConnectionToPeer



Keep-Alive je tiež obojsmerný!

Checksum:

Súbor window_manager.py implementuje funkciu hashMSG(), ktorá sa používa na výpočet kontrolného súčtu:

```
def hashMsg(bytes): 4 usages ± Nazar Faustyn
    crc = 0xFFFF
    polynomial = 0x1021

    for byte in bytes:
        crc ^= byte << 8
        for i in range(8):
            if crc & 0x8000:
                crc = (crc << 1) ^ polynomial
            else:
                crc <<= 1
            crc &= 0xFFFF
    return crc
```

Implementačné funkcie:

Používa sa algoritmus CRC-16

Pracuje s každým bajtom užitočného zaťaženia správy

Umožňuje detekciu chýb prenosu

Použitie:

Metóda manager.parse() kontroluje integritu správy

Ak sa zistí nekonzistentnosť, hodnota lastMessageCorrupted sa nastaví na True

Fragmentacia:

Implementované v sendThread.py:

Kľúčové vlastnosti:

Rozdelenie správ na fragmenty s veľkosťou až 1494 bajtov

Podporuje správy s jedným aj viacerými fragmentmi

Číslovanie a sledovanie fragmentov

Ak je dĺžka správy menšia ako odoslané údaje, správu nefragmentujeme, ak je dlhšia, rozdelíme ju na fragmenty.

```
if len(fragments) == 1:
    message = manager(msgType=2, flags=1,
                      payload=fragments[0],
                      fragmentSeq=len(fragments))
    sendMSG(sock, message, ip, port)
    print("Sent single fragment message")

# Add to unacknowledged messages
with window_manager.window_lock:
    if window_manager.sender_window is None:
        window_manager.sender_window = SenderWindow(WINDOW_SIZE)
    window_manager.sender_window.add_packet(Packet(sequence_number=0, fragments[0], time.time()))

else:
    message_id = get_new_message_id()
    message = manager(msgType=3, flags=2, fragmentSeq=len(fragments), timestamp=message_id)
    sendMSG(sock, message, ip, port)

    for i in range(0, len(fragments), WINDOW_SIZE):
```

ARQ (SR):

Implementované v súboroch sendThread.py a window_manager.py:

Mechanizmy SR:

Posuvné prenosové okno

Časový limit pre retransmisiu

Potvrdenie (ACK/NAK) každého fragmentu

Spracovanie stratených a poškodených paketov

```
class WindowManager: 1 usage  👤 Nazar Faustyn
    def __init__(self):  👤 Nazar Faustyn
        self.sender_window = None
        self.receiver_window = None
        self.window_lock = threading.Lock()
window_manager = WindowManager()
```


Mam window pre Receiver a Sender a funguju nasledovne:

```
class SenderWindow: 5 usages  ⚡ Nazar Faustyn
    def __init__(self, size):  ⚡ Nazar Faustyn
        self.size = size
        self.base = 0
        self.next_seq_num = 0
        self.packets = {}

    def is_full(self): 1 usage  ⚡ Nazar Faustyn
        return len(self.packets) >= self.size

    def can_send(self, seq_num): 1 usage  ⚡ Nazar Faustyn
        return (seq_num >= self.base and
                seq_num < self.base + self.size and
                len(self.packets) < self.size)

    def add_packet(self, packet): 5 usages  ⚡ Nazar Faustyn
        if self.can_send(packet.sequence_number):
            self.packets[packet.sequence_number] = packet
            return True
        return False

    def remove_packet(self, seq_num): 2 usages (2 dynamic)  ⚡ Nazar Faustyn
        if seq_num in self.packets:
            del self.packets[seq_num]
            # Update base to the next unacknowledged packet
            while self.base not in self.packets and self.base != self.next_seq_num:
                self.base = (self.base + 1) % (MAX_SEQ_NUM + 1)
```

```

class ReceiverWindow: 4 usages  ± Nazar Faustyn
    def __init__(self, size): ± Nazar Faustyn
        self.size = size
        self.base = 0
        self.received_buffer = {}

    def is_in_window(self, seq_num): 2 usages (1 dynamic) ± Nazar Faustyn
        if self.base <= seq_num < self.base + self.size:
            return True
        # Handle wrap-around
        if self.base + self.size > MAX_SEQ_NUM:
            if seq_num >= self.base or seq_num < (self.base + self.size) % (MAX_SEQ_NUM + 1):
                return True
        return False

    def receive_packet(self, seq_num, payload): 1 usage (1 dynamic) ± Nazar Faustyn
        if self.is_in_window(seq_num):
            self.received_buffer[seq_num] = payload

            while self.base in self.received_buffer:
                del self.received_buffer[self.base]
                self.base = (self.base + 1) % (MAX_SEQ_NUM + 1)
            return True
        return False

```

Lua Script:

Nazov protokolu

```

-- MeredovN Custom UDP Protocol
local mcup = Proto("MCUP", "Meredov Custom UDP Protocol (MCUP)")

```

Definicia pre typy Packetov

```

-- Comprehensive Message Type Definitions
local msg_types = {
    [1] = "Control Packet",
    [2] = "Single Message Packet",
    [3] = "Multi-Fragment Message Packet",
    [4] = "File Transfer Packet",
    [5] = "ACK/NACK Packet"
}

```

Definicia Flagov pre Payload Types

```

-- Control Packet Flags
[1] = {
    [2] = "SYN",
    [3] = "SYN-ACK",

    [4] = "KEEP_ALIVE",
    [5] = "KEEP_ALIVE-ACK",

    [8] = "FIN",
    [9] = "FIN-ACK"
},
-- Single Message Packet Flags
[2] = {
    [1] = "Single Message",

    [5] = "All Fragments Send"
},
-- Multi-Fragment Message Packet Flags
[3] = {
    [2] = "Fragments size Transfer",
    [4] = "Fragment Message"
},
-- File Transfer Packet Flags
[4] = {
    [1] = "Filename Transfer",
    [2] = "File Size Transfer",
    [3] = "Fragment Count Transfer",
    [4] = "File Fragment part",

    [6] = "Look missing fragment"
},
-- ACK/NACK Packet Flags
[5] = {
    [1] = "ACK",
    [2] = "NACK"
}
}

```

Inicializacia polej hlavičky

```

-- Protocol Fields
local f_msgType_flags = ProtoField.uint8("mcup.msgType_flags", "Message Type and Flags", base.HEX)
local f_checksum = ProtoField.uint16("mcup.checksum", "Checksum", base.HEX)
local f_fragmentSeq = ProtoField.uint16("mcup.fragmentSeq", "Fragment Sequence", base.DEC)
local f_timestamp = ProtoField.uint8("mcup.timestamp", "Timestamp", base.DEC)
local f_payload = ProtoField.bytes("mcup.payload", "Payload")
local f_detailed_type = ProtoField.string("mcup.detailed_type", "Detailed Type")

-- Register protocol fields
mcup.fields = {
    f_msgType_flags,
    f_checksum,
    f_fragmentSeq,
    f_timestamp,
    f_payload,
    f_detailed_type
}

```

Dissector pre hlavičku

```

-- Parse first byte (message type and flags)
local msgType_flags = buffer(0, 1):uint()
local msgType = bit.rshift(msgType_flags, 4)
local flags = bit.band(msgType_flags, 0x0F)
subtree:add(f_msgType_flags, buffer(0, 1), msgType_flags)

-- Add checksum
local checksum = buffer(1, 2):uint()
subtree:add(f_checksum, buffer(1, 2), checksum)

-- Add fragment sequence
local fragmentSeq = buffer(3, 2):uint()
subtree:add(f_fragmentSeq, buffer(3, 2), fragmentSeq)

-- Add timestamp
local timestamp = buffer(5, 1):uint()
subtree:add(f_timestamp, buffer(5, 1), timestamp)

-- Determine detailed packet type
local detailed_type = "Unknown"
if flag_definitions[msgType] and flag_definitions[msgType][flags] then
    detailed_type = flag_definitions[msgType][flags]
end
subtree:add(f_detailed_type, buffer(), detailed_type)

-- Add payload if exists
if buffer:len() > 6 then
    subtree:add(f_payload, buffer(6, buffer:len() - 6))
end

-- Update info column
pinfo.cols.info = string.format(formatstring: "%s (%s), Seq: %d, Flags: 0x%x",
    msg_types[msgType] or "Unknown",
    detailed_type)

```

WireSharkColor:

Ake farby používam

Name	Filter
<input checked="" type="checkbox"/> NACK	mcup.msgType_flags == 0x52
<input checked="" type="checkbox"/> ACK	mcup.msgType_flags == 0x51
<input checked="" type="checkbox"/> Check missing part	mcup.msgType_flags == 0x46
<input checked="" type="checkbox"/> Fragment of file	mcup.msgType_flags == 0x44
<input checked="" type="checkbox"/> file FRAG count	mcup.msgType_flags == 0x43
<input checked="" type="checkbox"/> file SIZE	mcup.msgType_flags == 0x42
<input checked="" type="checkbox"/> file NAME	mcup.msgType_flags == 0x41
<input checked="" type="checkbox"/> Frag size Transfer	mcup.msgType_flags == 0x34
<input checked="" type="checkbox"/> Frag Mess	mcup.msgType_flags == 0x32
<input checked="" type="checkbox"/> All frag send	mcup.msgType_flags == 0x25
<input checked="" type="checkbox"/> Single Mess	mcup.msgType_flags == 0x21
<input checked="" type="checkbox"/> FIN-ACK	mcup.msgType_flags == 0x19
<input checked="" type="checkbox"/> FIN	mcup.msgType_flags == 0x18
<input checked="" type="checkbox"/> SYN-ACK	mcup.msgType_flags == 0x13
<input checked="" type="checkbox"/> SYN	mcup.msgType_flags == 0x12
<input checked="" type="checkbox"/> KEEP-ALIVE-ACK	mcup.msgType_flags == 0x15
<input checked="" type="checkbox"/> KEEP-ALIVE	mcup.msgType_flags == 0x14

Ako to vyžera

No.	Time	Source	Destination	Protocol	Length	Info
129	16.785284	10.10.19.34	10.10.19.34	MCUP	38	Control Packet (SYN), Seq: 0, Flags: 0x2
130	16.785551	10.10.19.34	10.10.19.34	MCUP	38	Control Packet (SYN-ACK), Seq: 0, Flags: 0x3
131	16.911887	10.10.19.34	10.10.19.34	MCUP	38	Control Packet (KEEP_ALIVE), Seq: 0, Flags: 0x4
132	16.912170	10.10.19.34	10.10.19.34	MCUP	38	Control Packet (KEEP_ALIVE-ACK), Seq: 0, Flags: 0x5
133	16.972898	10.10.19.34	10.10.19.34	MCUP	38	Control Packet (KEEP_ALIVE), Seq: 0, Flags: 0x4
134	16.973284	10.10.19.34	10.10.19.34	MCUP	38	Control Packet (KEEP_ALIVE-ACK), Seq: 0, Flags: 0x5
167	21.842880	10.10.19.34	10.10.19.34	MCUP	43	Single Message Packet (Single Message), Seq: 1, Flags: 0x1
168	21.842624	10.10.19.34	10.10.19.34	MCUP	38	ACK/NACK Packet (ACK), Seq: 1, Flags: 0x1
169	21.917171	10.10.19.34	10.10.19.34	MCUP	38	Control Packet (KEEP_ALIVE), Seq: 0, Flags: 0x4
170	21.917529	10.10.19.34	10.10.19.34	MCUP	38	Control Packet (KEEP_ALIVE-ACK), Seq: 0, Flags: 0x5
171	21.978224	10.10.19.34	10.10.19.34	MCUP	38	Control Packet (KEEP_ALIVE), Seq: 0, Flags: 0x4
172	21.978535	10.10.19.34	10.10.19.34	MCUP	38	Control Packet (KEEP_ALIVE-ACK), Seq: 0, Flags: 0x5
173	24.721374	10.10.19.34	10.10.19.34	MCUP	42	Single Message Packet (Single Message), Seq: 1, Flags: 0x1
174	24.721742	10.10.19.34	10.10.19.34	MCUP	38	ACK/NACK Packet (ACK), Seq: 1, Flags: 0x1
207	26.928458	10.10.19.34	10.10.19.34	MCUP	38	Control Packet (KEEP_ALIVE), Seq: 0, Flags: 0x4
208	26.928806	10.10.19.34	10.10.19.34	MCUP	38	Control Packet (KEEP_ALIVE-ACK), Seq: 0, Flags: 0x5
209	26.983418	10.10.19.34	10.10.19.34	MCUP	38	Control Packet (KEEP_ALIVE), Seq: 0, Flags: 0x4
210	26.983747	10.10.19.34	10.10.19.34	MCUP	38	Control Packet (KEEP_ALIVE-ACK), Seq: 0, Flags: 0x5
211	29.244669	10.10.19.34	10.10.19.34	MCUP	38	Control Packet (FIN), Seq: 0, Flags: 0x8
212	29.244926	10.10.19.34	10.10.19.34	MCUP	38	Control Packet (FIN-ACK), Seq: 0, Flags: 0x9

Ako spustyt program:

Ako spustyt:

```
(.venv) faustyn@eduroam-163-42 protocol_final % python main.py
Enter the target IP:
set the IP to the IP of local host (10.10.19.34)
Enter the target port: 2233
Enter the port to listen on: 3322
To start talking, type !start
```

Ake commands môžeme používať?

!start – Inicializácia spojenia

```
!start
Attempting connection...
Connection Established
```

!end – Ukončenie spojenia

```
!end
Cutting Connection
```

!frag – Nastaviť max fragment

```
Enter message: !frag
Enter the maximum fragment length: 2
Fragment length set to 2
```

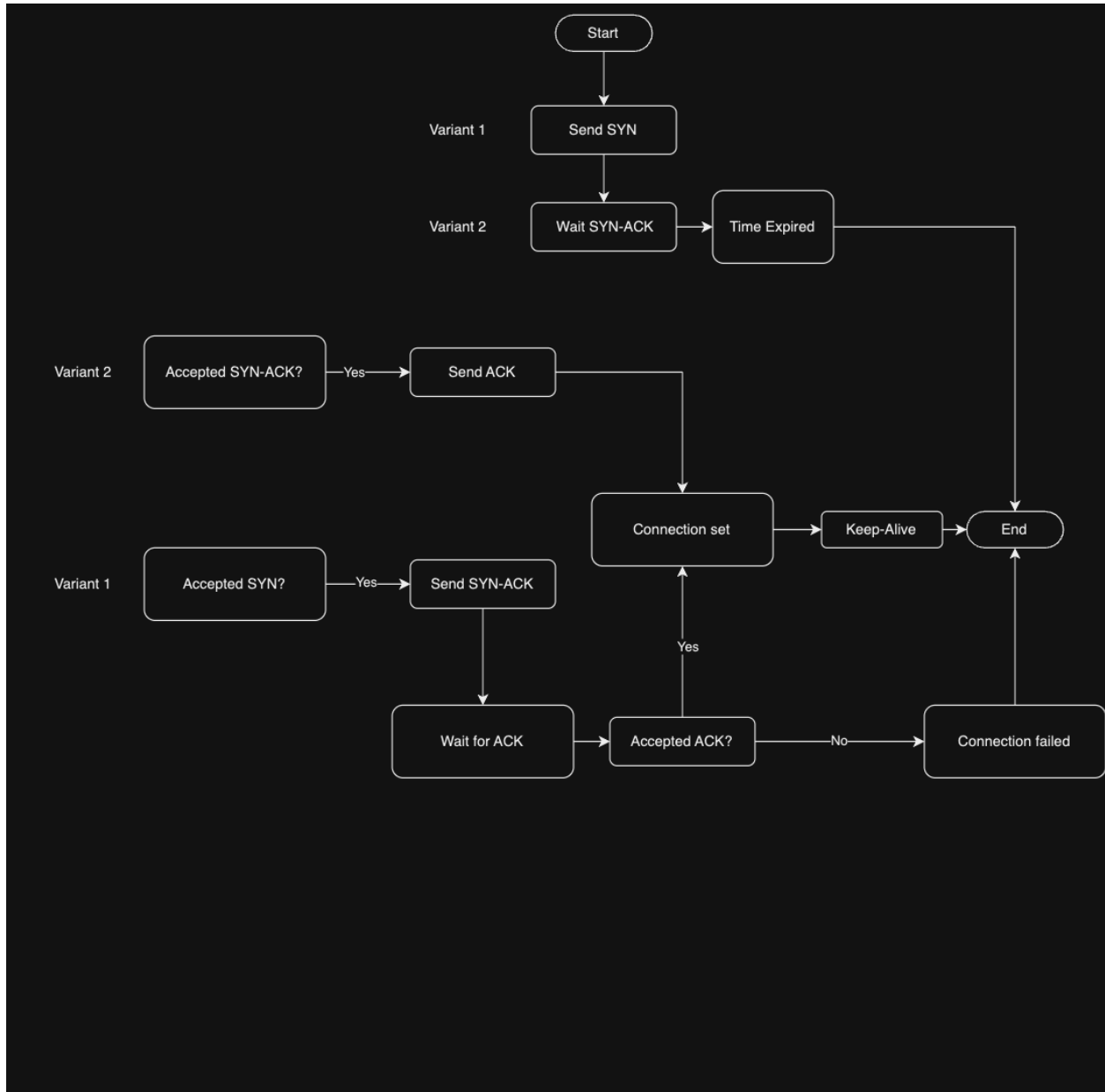
!err – Umyselná chyba (Error simulation)

```
Choose what type of message you want to corrupt:
1. Corrupt a message
2. Corrupt a file
Enter choice: 1
Enter the message to corrupt: hi
Send mess b'!b\x03\x00\x00\x11hi'
Sent corrupted single fragment message, with seq 0
```

!file – Odoslat' subor

```
Enter message: !file
Enter the source file path: /Users/faustyn/Developer/STU_2324/PKS/PKS_Project/PKS_Project/protocol_final/test_files/cat.jpg
Sending file: /Users/faustyn/Developer/STU_2324/PKS/PKS_Project/PKS_Project/protocol_final/test_files/cat.jpg
Sending file: cat.jpg
Received ACK for packet 0
Sending file size: 1468589
Received ACK for packet 0
Sending file fragments count: 983
Received ACK for packet 0
Sending fragment 0 (sequence number 0)
Sending fragment 1 (sequence number 1)
Sending fragment 2 (sequence number 2)
Sending fragment 3 (sequence number 3)
```

Working Diagram



Záver a výsledky:

Tento projekt sa zameriaval na implementáciu vlastného UDP protokolu s pokročilými sieťovými mechanizmami a funkciami. Hlavné výsledky a prínosy projektu možno zhrnúť do niekoľkých kľúčových bodov:

1. Vlastný protokol
 - Navrhnutý vlastný UDP protokol s komplexnou štruktúrou hlavičky
 - Podpora rôznych typov správ a paketov
 - Implementácia špecifických flagov pre rôzne typy komunikácie
2. Mechanizmy spoľahlivosti
 - Úspešná implementácia HandShake mechanizmu pre nadviazanie a ukončenie spojenia
 - Keep-Alive mechanizmus pre udržiavanie aktívneho spojenia
 - CheckSum (CRC-16) pre detekciu chýb prenosu
3. Fragmentácia správ
 - Podpora fragmentácie správ až do veľkosti 1494 bajtov
 - Mechanizmus sledovania a číslovania fragmentov
 - Schopnosť prenášať správy s jedným aj viacerými fragmentmi
4. ARQ (Automatic Repeat reQuest) mechanizmus
 - Implementácia Selective Repeat (SR) algoritmu
 - Posuvné prenosové okno pre efektívny prenos
 - Podpora potvrdzovania (ACK/NACK) každého fragmentu
 - Riešenie stratených a poškodených paketov
5. Podpora nástrojov
 - Vytvorenie Lua skriptu pre Wireshark na analýzu protokolu
 - Vlastné farebné zvýraznenie pre lepšiu vizualizáciu prevádzky
6. Funkcionalita programu
 - Podpora príkazov ako !start, !end, !frag, !err, !file
 - Obojsmerná komunikácia
 - Flexibilné nastavenia prenosu

Prínosy projektu:

- Hlboké pochopenie princípov sieťovej komunikácie
- Praktická implementácia pokročilých sieťových mechanizmov
- Riešenie spoľahlivosti prenosu dát na úrovni aplikačnej vrstvy

Projekt predstavuje komplexné riešenie prenosu dát s dôrazom na spoľahlivosť, fragmentáciu a efektivitu komunikácie prostredníctvom vlastného UDP protokolu!!