

**FIIT STU**

Ilkovičova 2, 842 16 Karlova Ves

## Komunikácia s využitím UDP protokolu

Autor: Martin Pirkovský

2020/2021

## Obsah

1) Návrh protokolu .....	3
2) Zmeny voči návrhu .....	7
3) Dôležité z použitých knižníc.....	7
4) Priebeh komunikácie .....	8
5) Užívateľské prostredie.....	9
6) Záver .....	11
Zdroje .....	12

## 1) Návrh protokolu

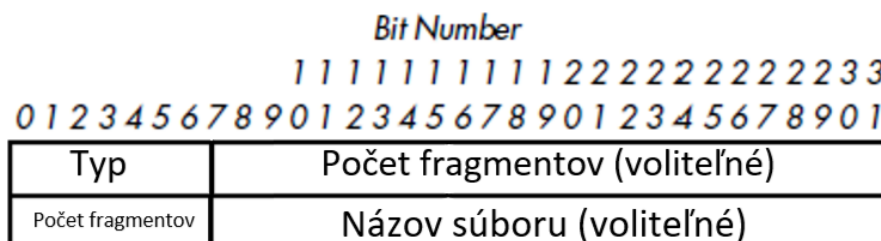
Program je implementovaný v programovacom jazyku Python. Ako typ komunikácie som si zvolil server klient. Používateľ si môže vybrať či chce byť serverom alebo klientom. Keď si vyberie server, tak nastaví port, na ktorom má počúvať respektíve, z ktorého mu príde inicializačný packet. Server následne len prijíma počas posielania a keď dostane zlý packet, tak pošle správu aby ho poslal klient znovu. Po prenose server dostáva každých 60 sekúnd keep alive aby sa udržalo spojenie. Ak server nedostane keep alive tak ukončí spojenie. Ďalej môže server zmeniť svoju rolu a stať sa klientom. Ak si zvolí užívateľ klienta tak môže zadať port a IP servera, na ktorý chce posielať dáta, veľkosť fragmentu a text správy alebo súbor, ktorý chce poslať. Ďalej klient iba posiela dáta a po skončení posiela keep alive správu, pričom ak nedostane potvrdenie od servera každých 60 sekúnd, tak ukončí spojenie.

Veľkosť fragmentu si môže používateľ určiť od 1 do 1466. 1466 pretože od veľkosti 1500 B je potrebné odčítať veľkosť IP hlavičky (20 B), tiež veľkosť UDP hlavičky (8 B) a ešte veľkosť mojej hlavičky (6B). Do úvahy berieme 1500 B, pretože nechceme aby sa nám dáta pri posielaní neboli znova fragmentované na linkovej vrstve.

$$1500 - 28 - 6 = 1466 \text{ B}$$

### Vlastná hlavička:

Informačný paket:



Veľkosť informačnej hlavičky: 1 B + 4 B (voliteľné) + názov súboru (voliteľné)

Dátový paket:



Veľkosť dátovej hlavičky: 4 B + 2 B = 6 B

### Informačný paket:

1) Typ paketu: 1 B

0	→	nadviazanie spojenia
1	→	potvrdenie úspešného doručenia fragmentu
2	→	Keep alive správa
3	→	text
4	→	súbor
5	→	chybný fragment
6	→	ukončenie spojenia
7	→	dátový packet

2) Počet fragmentov: 4 B

- Počet koľko bude poslaných dátových fragmentov.
- Potrebujem na to  $2^{21}$  fragmentov, aby som vedel poslať súbor o veľkosti 2 MB po jednom byte.

3) Názov súboru (voliteľné):

- Tento údaj je voliteľný, nakoľko sa inicializuje iba ak posielam súbor.
- Veľkosť nie je potrebná, nakoľko sa iba dá nakoniec hlavičky ako posledný údaj.

### Dátový paket:

1) Poradie fragmentu: 4 B

- Určuje poradie fragmentu, aby som nemohol mať pakety poprehadzované ak by prišli v inom poradí.

2) CRC: 2 B

- Na CRC, ktoré mi dá funkcia `crc_hqx(data, crc)` z knižnice `binascii`, mi stačia 2 B, nakoľko výstupom funkcie je maximálne 16 bitové číslo.

3) Dáta

- Dáta, ktoré sú posielané. Môže to byť text alebo súbor.

## Selective Reject ARQ

Na zabezpečenie proti chybám som si vybral selective reject ARQ metódu. Táto metóda kontinuálne posieľa pakety po jednom za sebou. Prijímateľ na správne doručené pakety odpovedá správou o úspešnom prijatí, no ak dostane chybný fragment, tak pošle správu o chybnom fragmente, čím požiada odosielateľa o znovu poslanie daného fragmentu. Odosielateľ takto pošle znova jeden daný chybný fragment, pričom neprestal posieľať ďalšie fragmenty počas riešenia problému.

## Keep Alive

Ako som už spomenul vyššie, tak na udržanie spojenia som sa rozhodol použiť keep alive časovač, ktorý je nastavený na 60 sekúnd. Teda každých 60 sekúnd, mimo posielania dát, pošle keep alive správu serveru aby udržal spojenie otvorené. Server pošle späť klientovi tiež keep alive správu, čím mu dá vedieť, že správu dostal a teda ostáva spojenie otvorené ďalších 60 sekúnd.

## CRC (Cyclic Redundancy Check Code)

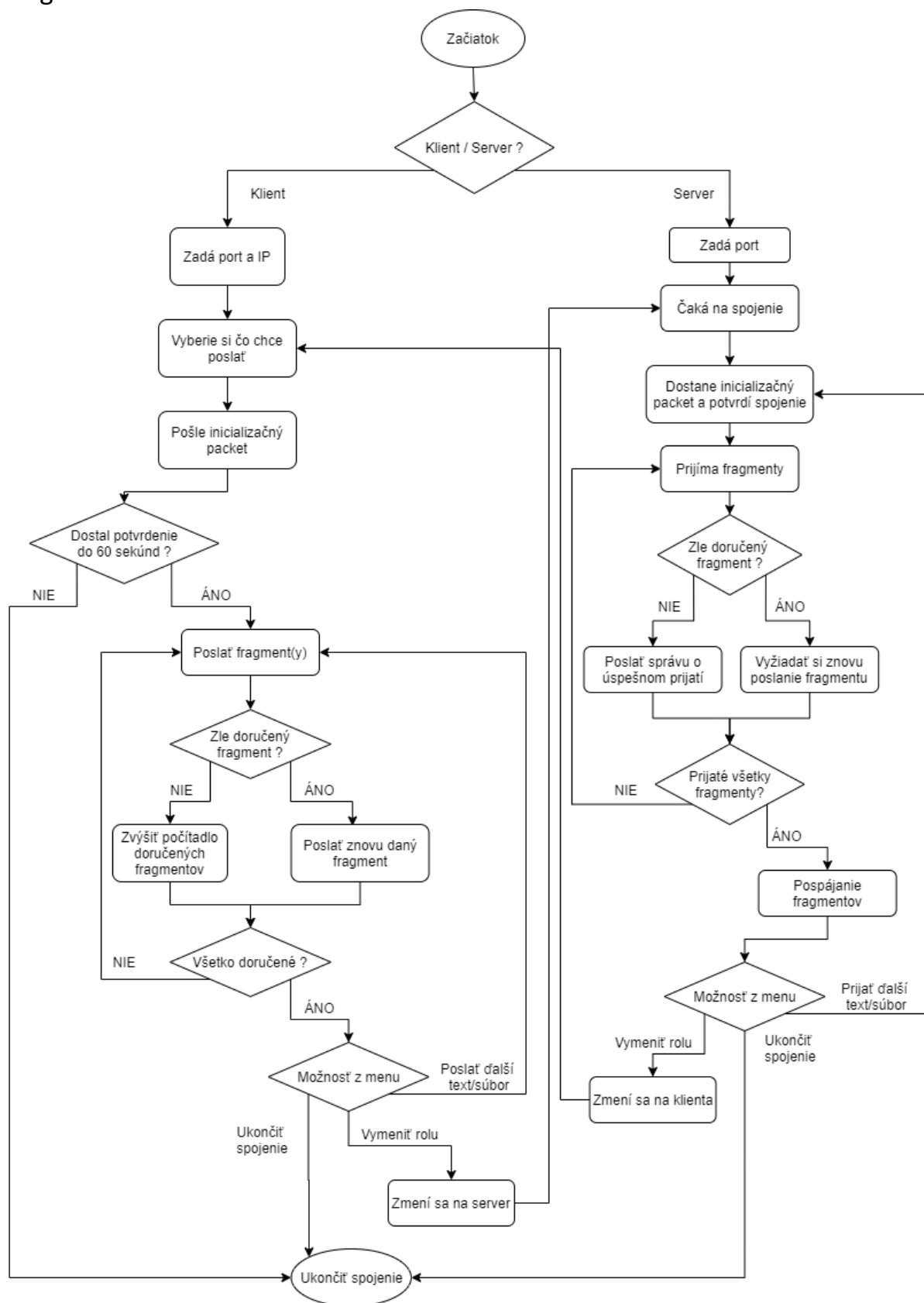
Na detekciu chybného fragmentu som sa rozhodol použiť vstavanú funkciu knižnice „binascii“, ktorá sa volá „crc\_hqx(data, crc)“, kde data sú logicky naše dáta v byte-och, ktoré chceme poslať do CRC a crc nám predstavuje číslo, ktoré sa pridá pred dáta pri rátaní crc ako inicializačná hodnota. Ja som sa rozhodol použiť ako inicializačnú hodnotu „0xffff“, čo znamená, že pred dáta sa dá 16 jednotiek. Táto funkcia využíva CRC-CCITT s deliteľom zadaným ako polynóm „ $x^{16} + x^{12} + x^5 + 1$ “, čo je v binárnom tvare 1000 1000 0001 0000 1. Výpočet CRC „A“ by vyzeralo nejak takto:

- červenou sú inicializačné bity
- čiernou je ascii hodnota „A“ (65) v binárnom tvare
- modrou je (k-1) nulových bit-ov, ktoré sa pridávajú nakoniec, kde „k“ je dĺžka polynómu

```
1111 1111 1111 1111 0100 0001 0000 0000 0000 0000
1000 1000 0001 0000 1
----- XOR
 111 0111 1110 1111 11
.....
```

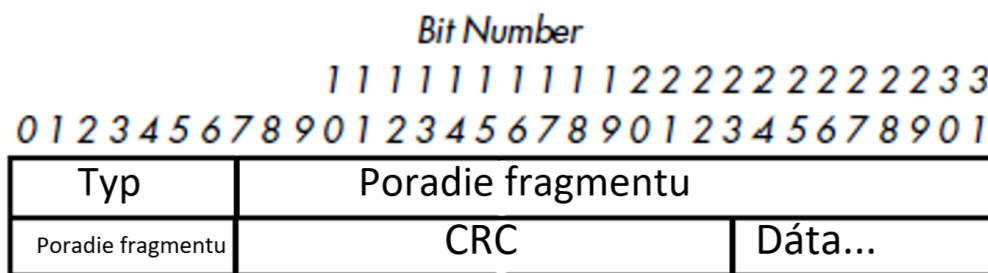
Takto by to pokračovalo až by sme nakoniec dostali hodnotu CRC = 1001 0100 0111 1001, ktorú následne pošleme spolu s dátami a poradovým číslom fragmentu serveru. Vysvetlenie fungovania mojej CRC funkcie mám zo stránky [srecord.sourceforge.net](http://srecord.sourceforge.net) [6].

## Diagram



## 2) Zmeny voči návrhu

Najväčšou zmenou voči návrhu je pridanie poľa „typ“ do dátovej hlavičky (obrázok 1), čo je samostatná hlavička, ktorú využívam iba pri posielaní dát, či už textu alebo súboru. Zároveň som medzi typy pridal aj typ „7“, čo hovorí, že sa jedná o dátový packet. Nakoľko som pridal do dátovej hlavičky ďalšie políčko tak sa mi aj jej veľkosť zmenila zo 6 B na 7 B, čo mi zároveň aj ovplyvnilo maximálnu možnú veľkosť dát, ktoré môžem prenášať naraz v jednom packete, a to z 1466 B na 1465 B. K tomuto číslu som sa dopracoval odčítaním od 1500 B (veľkosť dátovej časti v ethernetete) veľkosť IP hlavičky (20 B), UDP hlavičky (8 B) a mojej dátovej hlavičky (7 B).[1] Dátová hlavička a informačná hlavička sú 2 samostatné hlavičky, ktoré používam pri komunikácii.



*Obrázok 1*

Druhou vecou, ktorú som pridal oproti návrhu je riešenie stratených packet-ov. Nakoľko prenos súboru skončí až keď je súbor celý správne prijatý, tak keď ubehne sekunda bez toho aby sa prijímali packety a ešte nie sú všetky fragmenty správne prijaté, tak server pošle po jednom žiadosť o každý fragment, ktorý mu chýba.

Poslednou zmenou je keep alive, ktorý ak je zapnutý, funguje nasledovne. Každých 15 sekúnd je poslaný informačný packet typu „keep alive“ zo strany klienta, pričom ak nedostane odpoveď do 15 sekúnd tak spojenie ukončí. Zo strany servera funguje podobne, avšak server po prijatí prvého súboru čaká na správu typu „keep alive“, ak ju dostane pošle obratom späť „keep alive“ klientovi, no ak ju nedostane do 30 sekúnd rovnako ako klient spojenie ukončí. Počas posielania súboru/textu je keep alive vypnutý.

## 3) Dôležité z použitých knižníc

V mojom projekte som použil viacero knižníc, ktoré programovací jazyk Python podporuje. Takými hlavnými sú:

### a) Socket [2]

Túto knižnicu som využil na prácu so socketmi, a vlastne na realizáciu celej komunikácie, ako to aj bolo odporúčané v zadaní. Teda na príjem a posielanie packetov.

**b) Struct [3]**

Knižnicu struct som použil na zapuzdrenie čísel, ktoré posielam v hlavičke ako údaje o danom packete, alebo dokonca ako údaje týkajúce sa celej komunikácie. Táto knižnica zmení dané číslo na bajty, tak aby malo požadovanú veľkosť, napríklad 1 B keď posielam typ packetu.

**c) Binascii [4]**

Z tejto knižnice som využil funkciu `crc_hqx()`, ktorú som opísal už v návrhu ako funkciu na výpočet CRC.

**d) Threading [5]**

Threading som využil hlavne pri klientovi, aby som si vytvoril thread, ktorý má za úlohu iba počúvať, a odpovedať na správy, ktoré dostáva od servera. Takými sú napríklad „keep alive“ správy, alebo potvrdenia o úspešnom/neúspešnom doručení dátového fragmentu.

## 4) Priebeh komunikácie

Začneme z pohľadu servera. Server si ako prvé vyžiada port, na ktorom má očakávať klienta. Následne čaká kým sa klient pripojí na server. Po úspešnom spojení server počúva čo chce klient robiť, či chce poslať správu/súbor, posiela „keep alive“ na udržanie spojenia alebo chce ukončiť spojenie. Pri prijímaní správy alebo súboru si vypne timeout, a môže v klude čakať na packet, ktorý mu povie koľko fragmentov bude prijímať a ak sa jedná o súbor, tak aký je názov súboru. Následne počúva a prijíma fragmenty. Ak je fragment dobrý (sedí mu vyrátané CRC s tým, ktoré sa nachádza v hlavičke fragmentu), pošle potvrdenie o dobrom prijatí, ak je prijatý fragment zlý, pošle správu aby mu daný fragment poslal znovu. Ak je viac ako sekundu prijímanie neaktívne, tak server vyžiada všetky fragmenty, ktoré ešte nedostal. Po skončení prijímania server uloží súbor do priečinku „Received files“, vypíše cestu k súboru spolu s údajmi o súbore a vyžiada si od používateľa či chce pokračovať ako server, zmeniť svoju rolu na klienta alebo chce komunikáciu ukončiť. Ak by si zvolil pokračovať ako server prebehne rovnaký cyklus. Pri možnosti skončiť pošle informačný packet o ukončení spojenia, počká kým sa uloží súbor ak sa tak ešte nestalo a uzavrie spojenie. Ak si zvolí zmeniť rolu na klienta, tak rovnako ako pri uzavretí počká kým sa uloží súbor ak je to potrebné a zmení svoju rolu.

Teraz sa pozrieme na priebeh komunikácie z pohľadu klienta. Ako prvé si klient vyžiada od užívateľa IP adresu a port, na ktorý sa chce pripojiť a posilať dáta. Po úspešnom spojení sa vytvorí thread (listener), ktorý bude slúžiť na počúvanie servera a jeho správ, ktoré bude posilať počas komunikácie. Zároveň tento thread slúži, aby posielal keep alive správy. Po vytvorení threadu užívateľ zadá, či chce poslať správu, súbor, ukončiť spojenie alebo zmeniť svoju rolu na server. Pri zvolení správy sa ako prvý pošle informačný packet, ktorý hovorí, že sa bude posilať správa, a tým zastaví u servera keep alive timeout a nakoľko mu server pošle späť rovnakú správu, tak aj u seba vypne timeout. Následne užívateľ zadá či chce simulovať chybu, veľkosť fragmentu a napíše správu, ktorú chce poslať. Následne program fragmentuje správu, fragmenty sú číslované od nuly, a pošle ju po častiach. Po poslaní čaká kým dostane potvrdenie, že sa celá správa doručila správne. Tieto potvrdenia prijíma thread, ktorý som spomenul vyššie. Ak prišlo potvrdenie o dobrom doručení, tak listener zvýši počítadlo o správne poslaných fragmentoch. Ak však príde správa, že bol doručený chybný fragment,

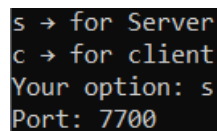


thread listener pošle znovu daný chybný fragment. Keď sú už všetky fragmenty doručené program vypíše správu, údaje o nej a čaká na možnosť čo ďalej. Ak si používateľ zvolil poslať súbor prebieha komunikácia podobne ako pri posielaní súboru. Najprv sa zastaví keep alive a následne používateľ zadá či chce simulovať chybu, zadá fragmentáciu a názov súboru, ktorý chce poslať. Následne už prebieha posielanie rovnako ako pri správe. Po doručení sa rovnako vypíšu údaje o súbore spolu s cestou k súboru, ktorý sa posielal a následne sa vyžiada opäť od užívateľa možnosť ako pokračovať. Pri možnosti skončiť sa pošle informačný packet serveru o ukončení komunikácie. Ak si zvolí, že chce zmeniť rolu na server, počká sa kým sa ukončí thread listener a zmení sa rola na server, ktorý počúva a čaká na pokyny od klienta.

## 5) Užívateľské prostredie

Server:

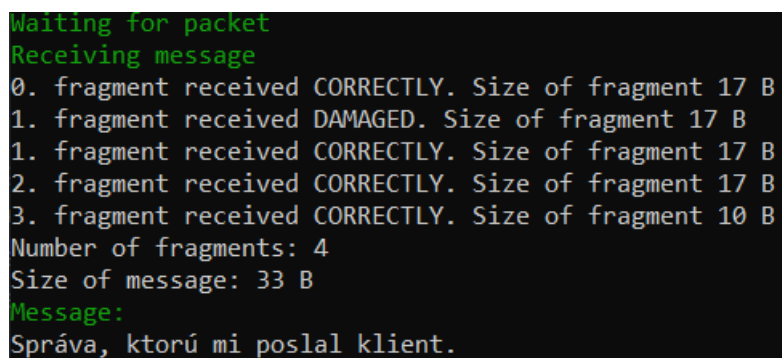
Ako prvé je potrebné zadať port, na ktorom chce používateľ počúvať (obrázok 2).



```
s → for Server  
c → for client  
Your option: s  
Port: 7700
```

Obrázok 2

Následne server čaká čo ďalej. Ak sa posielala správa, tak sa vypisujú správy o správnom alebo nesprávnom prijatí. Na obrázku 3 môžeme vidieť, že simulácia chyby je zapnutá a bola simulovaná na fragmente číslo 1. Ten bol následne prijatý už správne. Na konci výpisu o prijatí môžeme vidieť veľkosť fragmentu vrátane hlavičky dátového protokolu, čo je 7.



```
Waiting for packet  
Receiving message  
0. fragment received CORRECTLY. Size of fragment 17 B  
1. fragment received DAMAGED. Size of fragment 17 B  
1. fragment received CORRECTLY. Size of fragment 17 B  
2. fragment received CORRECTLY. Size of fragment 17 B  
3. fragment received CORRECTLY. Size of fragment 10 B  
Number of fragments: 4  
Size of message: 33 B  
Message:  
Správa, ktorú mi poslal klient.
```

Obrázok 3

Následne sa vypíše menu, čo chce ďalej užívateľ robiť (obrázok 4). Pri pokračovaní ako server prebehne rovnaký cyklus ako pred chvíľou, pri skončení sa ukončí komunikácia a pri zmene role sa zmení server na klienta.

```
Správa, ktorú mi poslal klient.  
Press "c" to continue as server  
Press "e" to end connection  
Press "g" to change to client  
Your option: c
```

Obrázok 4

Pri prijímaní súboru je proces veľmi podobný avšak, s tou zmenou, že na konci sa vypíše spolu s údajmi o súbore aj absolútna cesta, kde je súbor uložený (obrázok 5).

```
Number of fragments: 120  
Size of file: 144.238 KB  
Saving file...  
File is saved there: D:\Internet toto nie je\Skola\3 semester\PKS -  
Pocitacove a Komunikacne Siete\Zadania\Druhe\protocol\Received files  
\dog_1.png
```

Obrázok 5

Ak si zvolí používateľ klienta na začiatku, je potrebné aby zadal IP adresu a port, na ktorý chce posielať dáta. IP adresu je možné zadať aj ako „localhost“ (obrázok 6) ak chceme posielať na localhost, no je možné ho zadať aj obyčajne 127.0.0.1.

```
s → for Server  
c → for client  
Your option: c  
IP address:localhost  
Port: 7700
```

Obrázok 6

Po pripojení sa nám zobrazí menu, kde si môže vybrať používateľ z možností na obrázku 7.

```
Connected to server  
Press "m" to send message  
Press "f" to send file  
Press "g" to change to server  
Press "e" to end connection  
Press "Enter" to update  
Your option: m
```

Obrázok 7

Ak si zvolil poslanie správy, tak si vyberie, či chce simulovať chybu, akú chce maximálnu veľkosť fragmentu a následne môže napísať správu, ktorú chce poslať.

Nakoľko je na obrázku 8 zapnutá simulácia chyby, tak môžeme vidieť, že fragment číslo 1 je poslaný 2 krát, čo si môžeme vysvetliť, že prvé poslanie je s chybou a druhé už bez. Nakoniec sa vypíše, že bola správa úspešne poslaná spolu s veľkosťou správy v bajtoch. Napriek tomu, že sa nám môže zdať, že veľkosť správy by mal byť 35 B tak nie má to byť 38 B nakoľko diakritiku napríklad rozdeľuje „utf-8“ do dvoch bajtov a nakoľko tam mám 3 písmena s diakritikou, tak to sedí.

```
Sending message
Simulate damaged fragment: y/n
Your choice: y
Fragment size: 10
Write message: Správa, ktorú chcem poslať serveru.
0. fragment sent. Size of fragment: 17 B
1. fragment sent. Size of fragment: 17 B
1. fragment sent. Size of fragment: 17 B
2. fragment sent. Size of fragment: 17 B
3. fragment sent. Size of fragment: 15 B
Message sent
Number of fragments: 4
Size of message: 38 B
```

Obrázok 8

Ak si zvolil používateľ poslať súbor, tak je rovnako potrebné aby zadal, či chce simulovať chybu, musí zadať názov súboru, ktorý chce poslať a veľkosť fragmentu rovnako ako pri posielaní správy (obrázok 9).

```
Sending file
Simulate damaged fragment: y/n
Your choice: n
File name: video.mp4
Fragment size: 1465
```

Obrázok 9

Po úspešnom poslaní súboru sa vypíše správa o úspechu spolu s cestou k súboru, ktorý sme posielali. Rovnako ako pri posielaní správy sa vypíše počet fragmentov súboru a veľkosť súboru.

```
Everything sent
Path to file, which was sent: D:\Internet toto nie je\Skola\3 semester\PKS -
Pocitacove a Komunikacne Siete\Zadania\Druhe\protocol\video.mp4
Number of fragments: 2417
Size of file: 3.376 MB
```

Obrázok 10

Ak si používateľ zvolí možnosť skončenia komunikácie, klient dá vedieť serveru, že končí komunikáciu a ukončí spojenie. Ak si zvolí zmeniť svoju rolu, počká kým sa zmení na server, pričom po zmene role sa vymaže konzola.

## 6) Záver

V tomto zadaní bolo mojou úlohou urobiť nadstavbu nad protokolom UDP a môj vlastný protokol mal urobiť komunikáciu nad UDP spoľahlivejšiu. To si myslím, že sa mi podarilo, nakoľko som využíval `crc_hqx()` funkciu z knižnice `binascii` [4], ktorou si viem skontrolovať, či dáta po prenose sú nezmenené. To mi zabezpečí aby v prípade, že dôjde k zmene dát, som si vyžiadal daný fragment znovu, pričom tento cyklus prebieha až kým nedostanem fragment nepoškodený. Podobne ak by došlo počas prenosu k strate packetu, tak si program vyžiada poslať znova stratený alebo stratené fragmenty

## Zdroje

1. Druhá prednáška o ethernet a linkovej vrstve
2. <https://docs.python.org/3/library/socket.html>
3. <https://docs.python.org/3/library/struct.html>
4. <https://docs.python.org/2/library/binascii.html>
5. <https://docs.python.org/3/library/threading.html>
6. <http://srecord.sourceforge.net/crc16-ccitt.html>