Patrik Fusek

Návrh

Úvod

Implementáciu komunikačného programu budem robiť v programovacom jazyku python, najmä kvôli jeho jednoduchosti, keďže obsahuje množstvo funkcii a knižníc. Program bude konzolová aplikácia, čiže používateľ bude zadávať možnosti (napr. výber prijímača a vysielača) zo vstupu. Komunikácia bude prebiehať medzi dvomi účastníkmi v lokálnej sieti, ktorí si budú môcť odosielať textové a iné súbory prostredníctvom mojej vlastnej implementácie protokolu nad komunikačným protokolom UDP.

Funkcionalita

Po spustení programu budú na konzole 3 možnosti z ktorých si používateľ vyberá.

1.Klient

Ak bude daný používateľ reprezentovať vysielača (klienta), vyplní ďalšie informácie o IP adrese a porte servera, na ktorý sa chce pripojiť. V mojom prípade bude IP adresa localhost a port musí byť zhodný s portom, ktorý som zadal pri reprezentácii servera. Po vyplnení informácii IP adresy a porte servera bude klient automaticky odosielať na samostatnom threade keep alive signalizačnú správu, aby udržal spojenie so serverom v 5 sekundovom intervale. Ďalej sa môžem rozhodnúť, či bude moja správa vo forme textovej správy (message) alebo to bude ľubovoľný súbor do maximálnej veľkosti 2Mb. Po vybratí správy alebo ľubovoľného súboru ako používateľ zadám veľkosť fragmentu a rozbijem správu/súbor binárne v danej veľkosti. Dané fragmenty obsahujúce hlavičku a časť správy/súboru budem odosielať na server pomocou stop & wait ARQ metódy. Po úspešnom odoslaní celej správy/súboru ako používateľ môžem zadať ďalší prenos správy/súboru alebo ukončiť program.

2.Server

Ak bude daný používateľ reprezentovať prijímateľa (server), najskôr vyplní informáciu o porte, na ktorom bude daný server počúvať. Ďalej server čaká na spustenie klienta, ktorý mu hneď po spustení bude automaticky odosielať keep alive správy pre udržanie spojenia v určitom intervale. Server musí následne oboznámiť klienta o prijatí správy. Potom, čo si klient vyberie odosielanie ľubovoľnej správy alebo súboru, server bude čakať na fragment, ktorý skontroluje pomocou checksum validate. Ak validácia prebehla v poriadku, server uloží fragment podľa daného indexu poradia a oboznámi klienta o prijatí fragmentu. V opačnom prípade zahodí fragment a oboznámi klienta o zamietnutom/ neprijatom fragmente. Po obdržaní všetkých valídnych fragmentov server, pospája fragmenty v danom poradí a uloží ich pod daným typom do adresára.

3.Ukončenie programu

Táto voľba slúži na to ak už daný používateľ bude chcieť ukončiť program.

Patrik Fusek

Hlavička

ID	TYP	VELKOSŤ	CHECK SUM	DATA

ID: poradie daného fragmentu 1/100, 2/100...

Typ: konkrétna reprezentácia pre server ale aj pre klienta

Veľkosť: veľkosť odoslaného fragmentu

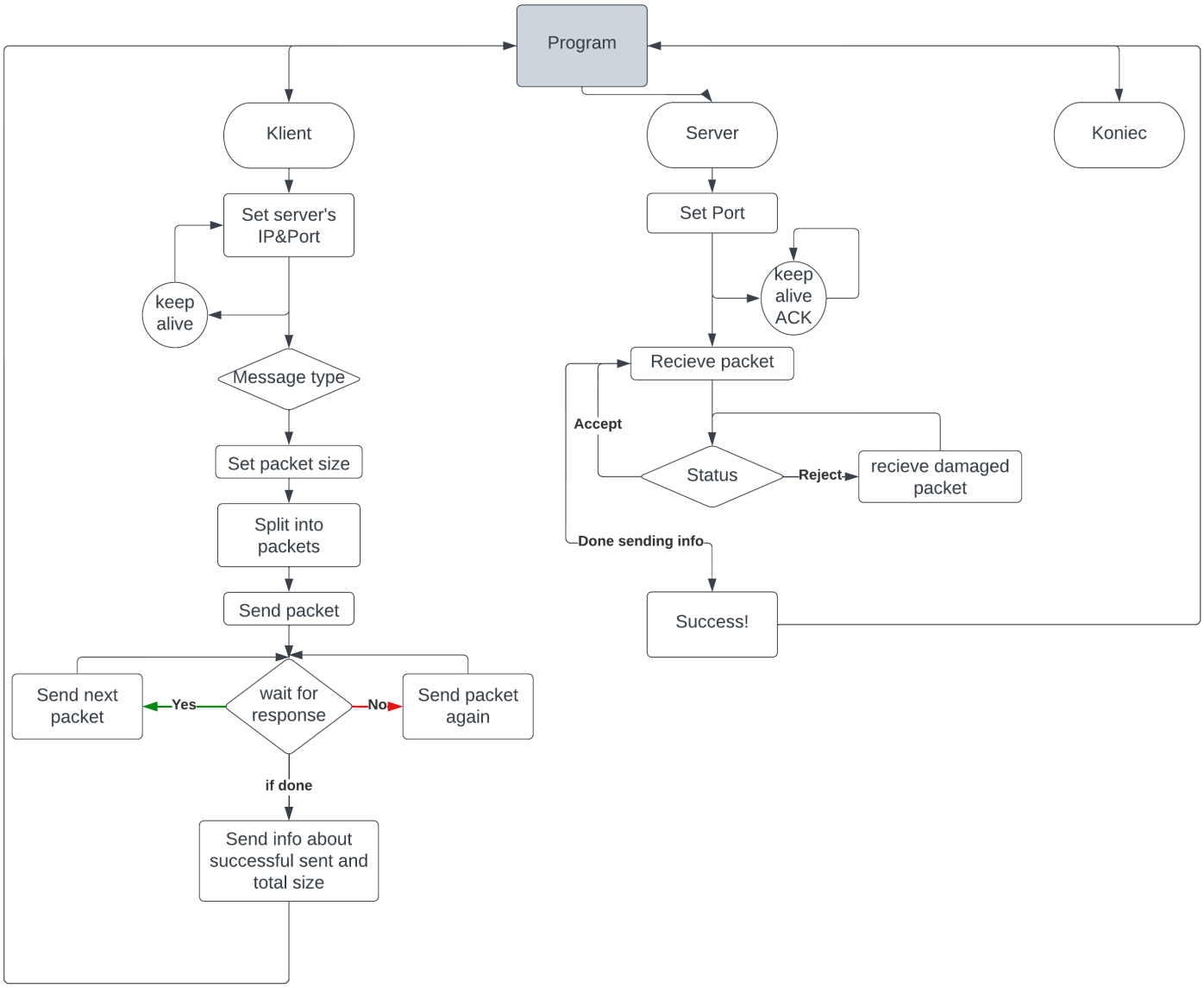
Check Sum: vypočítaná binárna hodnota daného fragmentu

Data: konkrétny úsek správy/súboru, ktorý chce klient odoslať

TYP			
	Dec	**Bin**	**Popis**
	1	0001	Keep alive signalizačná správa
	2	0010	prenos fragmentov
	3	0011	potvrdenie o prijatí fragmentu
	4	0100	potvrdenie o neprijatí fragmentu
	5	0101	oboznámenie o odoslaní celej správy/súboru zo strany klienta

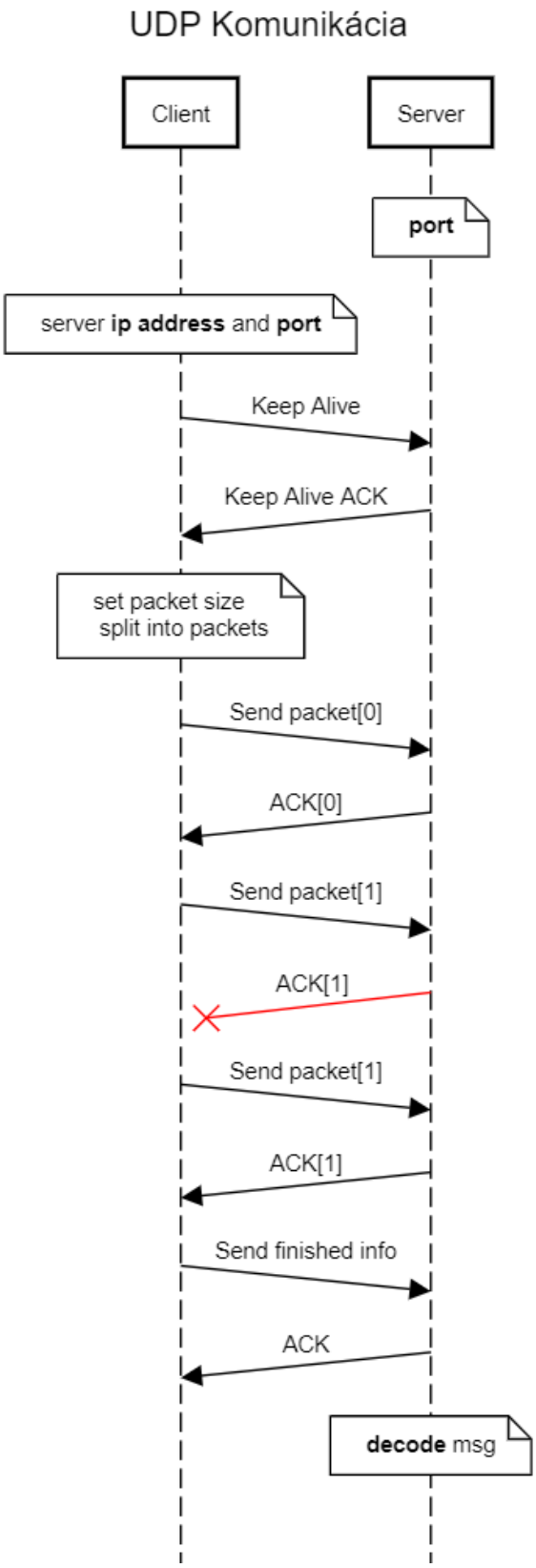
Patrik Fusek

Flowchart diagram



Patrik Fusek

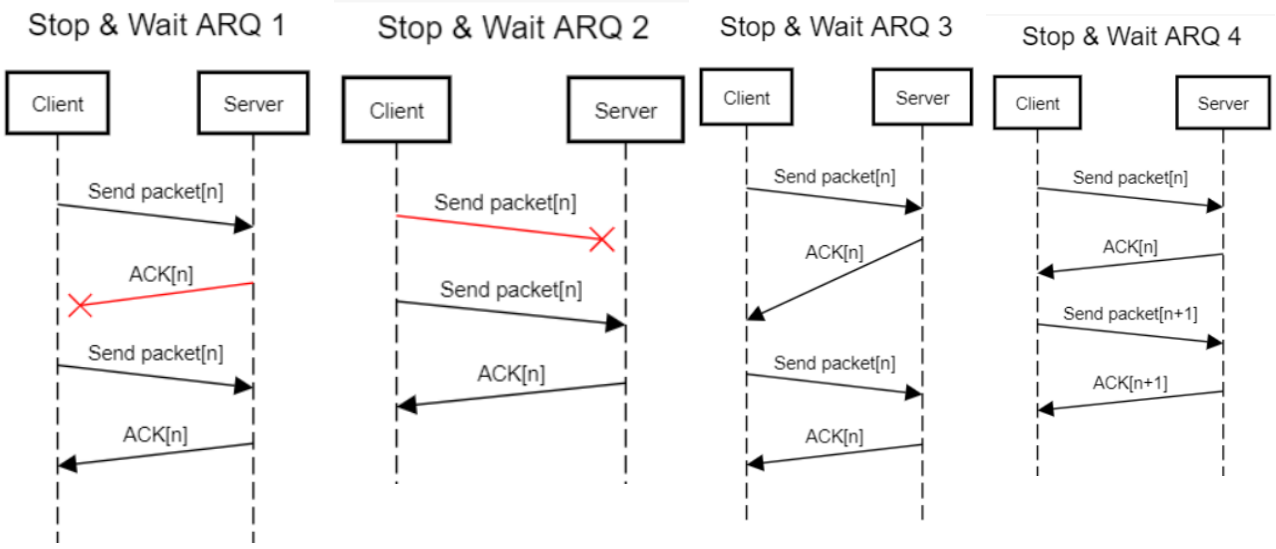
Sekvenčný diagram



Stop & Wait ARQ

Ako metódu pre odosielanie a prijímanie fragmentov použijem Stop & Wait ARQ. Tá funguje tak, že vysielač odošle fragment, a čaká po dobu x sekúnd na ACK zo strany prijímača. Po obdržaní ACK odošle ďalší fragment a znovu čaká na ACK. Na obrázkoch nižšie sú znázornené všetky možné scenáre metódy Stop & Wait ARQ.

1. Klient odošle fragment, čaká x sekúnd na ACK zo strany servera, ktoré zlyhalo. Klient po uplynutí časového limitu a neobdržaní ACK znovu odošle rovnaký fragment.
2. Klient odošle fragment, ktorého odosielanie zlyhalo. Klient po uplynutí časového limitu a neobdržaní ACK znovu odošle rovnaký fragment.
3. Klient odošle fragment, čaká x sekúnd na ACK zo strany servera, ktoré sa odoslalo až po časovom limite. Klient po uplynutí časového limitu znovu odošle fragment.
4. Klient odošle fragment, čaká x sekúnd na ACK zo strany servera, ktoré prebehlo v časovom limite. Klient po uplynutí časového limitu a obdržaní ACK odošle ďalší fragment.



Check Sum

Metóda kontrolnej sumy funguje tak, že dáta vo fragmente rozdelím na niekoľko segmentov. Segmenty spočítam binárne a prehodím na komplement(1 na 0 a opačne). Výsledný komplement pripojím do hlavičky Check Sum. Po prijatí fragmentu aj s Check Sum, server spočíta prijaté dáta rozdelené na segmenty a Check Sum binárne. Podľa toho server potvrdí prijatú/neprijatú správu Klientovi.

Keep alive

Metóda Keep alive bude udržiavať spojenie Klienta so Serverom. Klient odosiela keep alive signalizačnú správu v 5 sekundovom intervale. Ak by nastala situácia, do daného časového limitu Klient neodošle keep alive správu alebo klient neodošle keep alive ACK, spojenie sa ukončí.

Patrik Fusek

Zmeny počas implementácie

Plánovanú konzolovú komunikačnú aplikáciu som urobil pomocou grafického užívateľského rozhrania knižnice python tkinter. Taktiež som zmenil a pridal typ signalizačných správ a z hlavičky som odstránil celkovú veľkosť fragmentu. Tú si vie klient a server dopočítať. Ako algoritmus kontrolného súčtu som použil crc32 knižnice python binascii. Táto metóda sa používa na výpočet 32-bitového kontrolného súčtu dát.

Hlavička

ID [3 bajty]	TYP [2 bajty]	CHECK SUM [4 bajty]	DATA

ID: poradie daného fragmentu 1, 2...

Typ: konkrétna reprezentácia pre server ale aj pre klienta

Check Sum: vypočítaná crc32 hodnota daného fragmentu

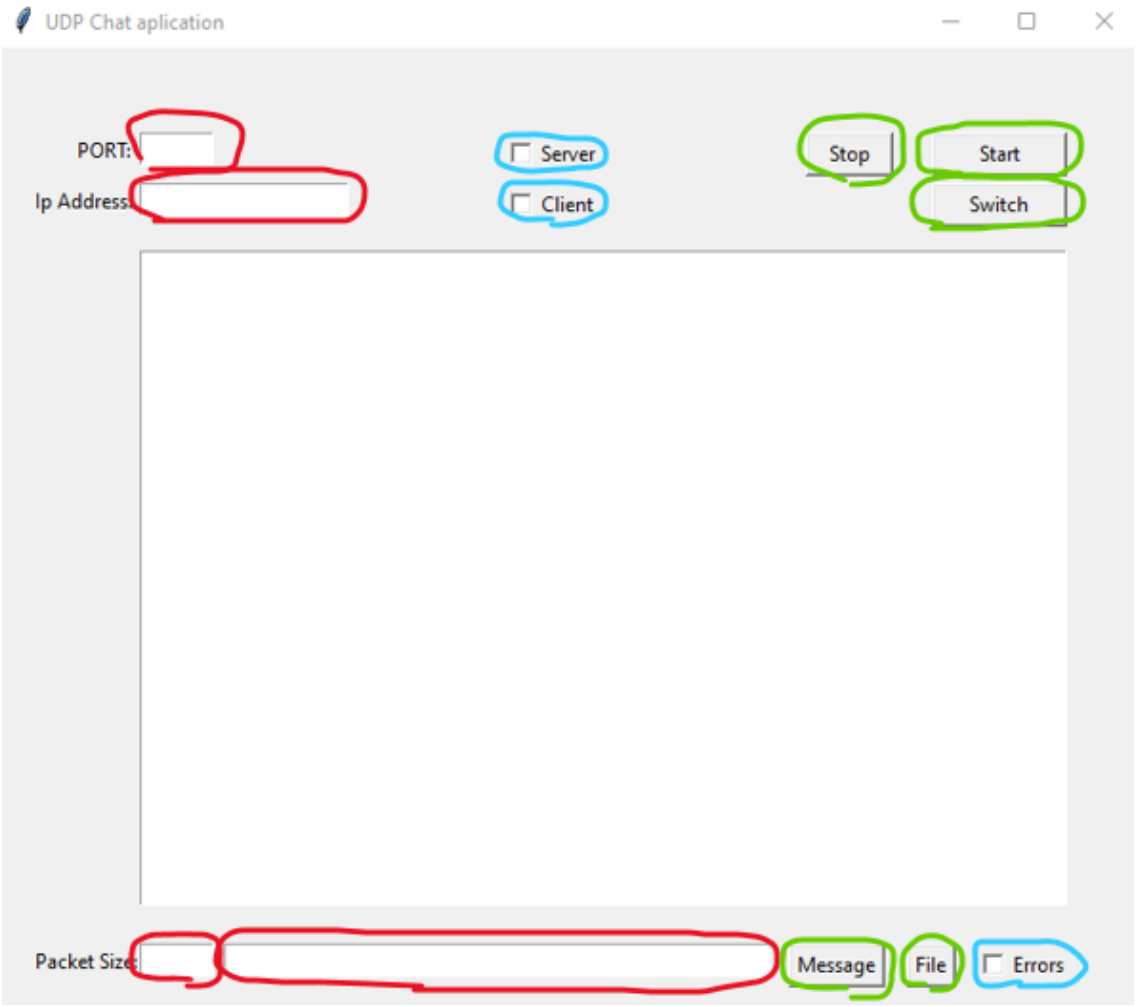
Data: konkrétny úsek správy/súboru, ktorý chce klient odoslať

Typ		
	ahoj	Inicializacna sprava spojenia klienta so serverom
	-1	Neprijaty packet zo strany servera
	5	Keep Alive
	10	Info, ze klient ide odoslat textovu spravu
	15	Info, ze klient ide odoslat file
	20	Odosielanie packetu spravy/suboru
	25	Info o uspesnom odoslani vsetkych packetov
	30	Switch roles
	35	Klient stop

Patrik Fusek

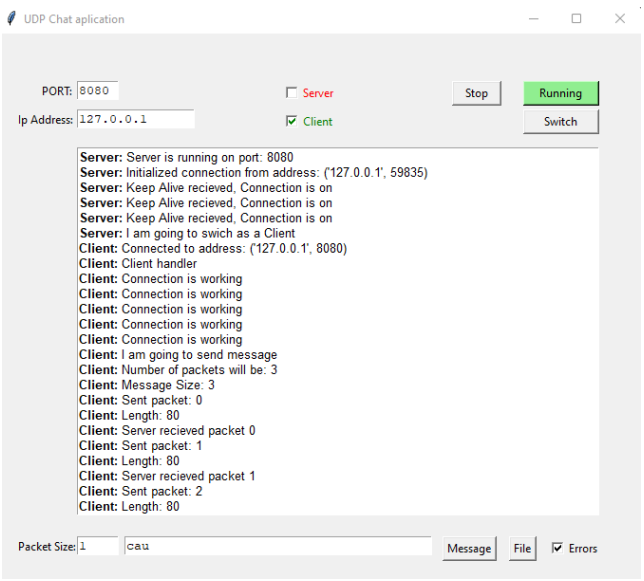
GUI

Po spustení programu pre klienta a server sa zobrazia 2 okná. Okno obsahuje ***4x TextBox***, kde používateľ zadá ip adresu, port, veľkosť fragmentu a textovú správu/cestu k danému súboru, ***5x Button***: Start, Stop, Switch, Message a File, pomocou ktorých používateľ riadi spustenie klienta/servera, stopnutie klienta, prepnutie rolí, odoslanie správy/súboru, ***1x Multiline MessageBox***, kde sa bude vypisovať stav klienta/servera, a všetky informácie o odosielaní fragmentov a správy/súboru, ***3x CheckButton***, kde si používateľ vyberie rolu klienta alebo servera a či chce simulovať chyby pri odosielaní textovej správy/súboru.

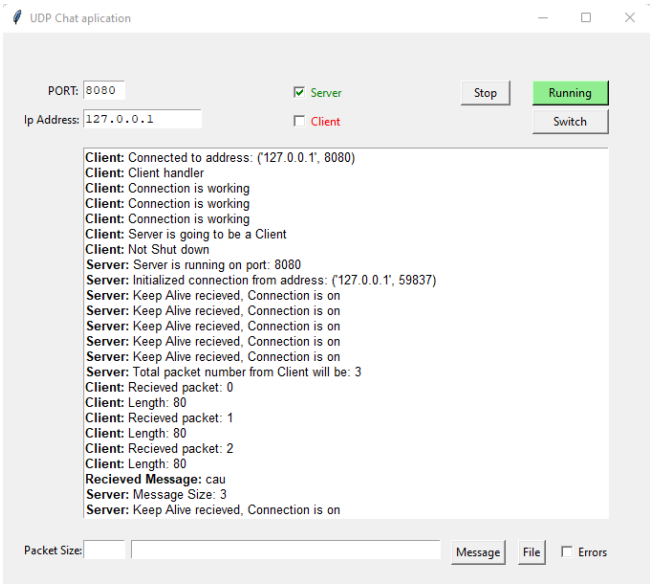


Patrik Fusek

Predvedenie Programu



GUI 1: Spustil som program ako Server



GUI 2: Spustil som program ako Klient

1. V okne GUI 1 som zadal port 8080, na ktorom bude server počúvať, označil som checkButton servera a stlačil Button Start, ktorý sa zmenil na zelenú farbu a text sa zmenil na Running.
2. V okne GUI 2 som zadal ip adresu servera 127.0.0.1 a port servera 8080, označil som checkButton klienta a stlačil Button Start, ktorý sa zmenil na zelenú farbu a text sa zmenil na Running.
3. Klient poslal na server inicializačnú správu, následne odosiela klient keep alive signalizačnú správu pre udržanie spojenia klienta so serverom v časovom intervale 5 sekúnd.
4. V okne GUI 2 som stlačil Button Switch slúžiaci na výmenu rolí. Klient odošle na server signalizačnú správu na výmenu rolí a zastaví Keep Alive signalizačné správy. Server uvoľní port, Klient sa prepne na Server a socket sa naviaže na port servera 8080, následne Server sa prepne na Klienta a znovu spustí Keep Alive signalizačné správy.
5. V okne GUI 1, ktoré po zmene rolí reprezentuje Klienta, zadám veľkosť fragmentu 1 a správu cau, ktorú po kliknutí na Button Message odošlem na Server ako textovú správu rozdelenú na fragmenty o veľkosti 1 a preruším Keep Alive signalizačné správy.
6. Klient aj Server vypísal poradie odoslaných a prijatých fragmentov aj s veľkosťou dát a výslednú správu, ktorú Server prijal od Klienta.
7. Po úspešnom odoslaní správy Klient pokračuje v odosielaní Keep Alive signalizačných správ v 5 sekundovom časovom intervale

Patrik Fusek

ireshark

No.	Time	Source	Source Port	Dst Port	Destination	Protocc	Length	Info
1	0.000000	127.0.0.1	49773	8080	127.0.0.1	UDP	36	49773 → 8080 Len=4
2	0.000388	127.0.0.1	8080	49773	127.0.0.1	UDP	33	8080 → 49773 Len=1
3	0.002464	127.0.0.1	49773	8080	127.0.0.1	UDP	33	49773 → 8080 Len=1
4	0.006959	127.0.0.1	8080	49773	127.0.0.1	UDP	33	8080 → 49773 Len=1
5	5.017718	127.0.0.1	49773	8080	127.0.0.1	UDP	33	49773 → 8080 Len=1
6	5.021531	127.0.0.1	8080	49773	127.0.0.1	UDP	33	8080 → 49773 Len=1
7	10.030406	127.0.0.1	49773	8080	127.0.0.1	UDP	33	49773 → 8080 Len=1
8	10.036914	127.0.0.1	8080	49773	127.0.0.1	UDP	33	8080 → 49773 Len=1
9	15.046618	127.0.0.1	49773	8080	127.0.0.1	UDP	33	49773 → 8080 Len=1
11	15.055062	127.0.0.1	8080	49773	127.0.0.1	UDP	33	8080 → 49773 Len=1
15	18.853269	127.0.0.1	49773	8080	127.0.0.1	UDP	34	49773 → 8080 Len=2
16	18.855488	127.0.0.1	8080	49773	127.0.0.1	UDP	34	8080 → 49773 Len=2
17	19.893165	127.0.0.1	49773	8080	127.0.0.1	UDP	33	49773 → 8080 Len=1
18	19.893323	127.0.0.1	8080	49773	127.0.0.1	UDP	32	8080 → 49773 Len=0
19	19.893570	127.0.0.1	49773	8080	127.0.0.1	UDP	38	49773 → 8080 Len=6
20	19.894403	127.0.0.1	8080	49773	127.0.0.1	UDP	32	8080 → 49773 Len=0
21	19.894980	127.0.0.1	49773	8080	127.0.0.1	UDP	1532	49773 → 8080 Len=1500
22	19.913337	127.0.0.1	8080	49773	127.0.0.1	UDP	33	8080 → 49773 Len=1
23	19.922805	127.0.0.1	49773	8080	127.0.0.1	UDP	1532	49773 → 8080 Len=1500
24	19.923019	127.0.0.1	8080	49773	127.0.0.1	UDP	33	8080 → 49773 Len=1
25	19.936126	127.0.0.1	49773	8080	127.0.0.1	UDP	1341	49773 → 8080 Len=1309
26	19.936577	127.0.0.1	8080	49773	127.0.0.1	UDP	33	8080 → 49773 Len=1
27	19.962383	127.0.0.1	49773	8080	127.0.0.1	UDP	37	49773 → 8080 Len=5
28	20.071615	127.0.0.1	49773	8080	127.0.0.1	UDP	33	49773 → 8080 Len=1
29	20.078424	127.0.0.1	8080	49773	127.0.0.1	UDP	33	8080 → 49773 Len=1
30	25.085921	127.0.0.1	49773	8080	127.0.0.1	UDP	33	49773 → 8080 Len=1
31	25.103476	127.0.0.1	8080	49773	127.0.0.1	UDP	33	8080 → 49773 Len=1

Fragmenty:

1. Inicializačné spojenie klienta so serverom. *Fragment [1, 2]*

2. Keep Alive Signalizačné správy pre udržanie spojenia. *Fragment [3 – 11, 28 – 31]*

3. Odoslanie inicializačnej informácie o odosielaní súboru. *Fragment [15, 16]*

4. Odoslanie a prijatie informácie o počte fragmentov súboru. *Fragment [17, 18]*

5. Odoslanie a prijatie informácie o názve súboru**.** *Fragment [19, 20]*

6. Odoslanie a prijatie fragmentov súboru. *Fragment [21 – 26]*

7. Odoslanie o úspešnom odoslaní súboru. *Fragment [27]*

Patrik Fusek

Použité knižnice

```
import binascii
import math
import sys
import threading
import time
from tkinter import *
import socket
import select
import os
import random
global thread_status
```

Splnené požiadavky

1. Program je napísaný v programovacom jazyku Python.
2. Program umožňuje nastaviť na strane klienta IP Adresu a Port, na ktorom server počúva, na strane servera port na ktorom bude počúvať
3. Program umožňuje nastaviť veľkosť fragmentu vždy pred odosielaním správy/súboru.
4. Klient aj Server zobrazujú: stav o spojení z danej ip adresy a portu, stav keep alive signalizačnej správy, informáciu o zmene rolí, inicializačná informácia na odosielanie správy/súboru, celkovú dĺžku správy/súboru, názov a cesta k súboru, poradie a dĺžka odoslaného a prijatého fragmentu, poradie a dĺžka neúspešne odoslaného a prijatého fragmentu, výsledná správa/názov prijatého súboru, informácia o stopnutí Klienta
5. Program umožňuje simuláciu chybného prenosu fragmentov, náhodne pripočítam 0 alebo 1 do kontrolnej sumy
6. Program umožňuje znovu vyžiadanie správy ak Server odpovedal ACK správou o chybnom fragmente alebo sa stratil ACK
7. Program je schopný odosielať ľubovoľný súbor väčší ako 2Mb
8. Ošetrené duplicitné prijatie fragmentu
9. Klient aj Server sú schopní sa prepínať za opačné roly
10. Program umožňuje komunikáciu medzi 2 počítačmi pripojenými na rovnakú sieť

Záver

Moja vlastná implementácia komunikačného programu slúži ako simulácia odosielania textových správ a ľubovoľného súboru nad komunikačným protokolom UDP. Program je rozdelený na 2 časti: Server (prijímač), ktorý je schopný prijímať správy od Klienta a odpovedať o prijatí správy. Klient (vysielač), ktorý je schopný odosielať správy a prijímať stav prijatej správy zo strany Servera. Testovanie programu prebehlo na jednom počítači.