Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Zadanie 2: Komunikácia s využitím UDP protokolu

Finálna dokumentácia

Peter Smreček

AIS ID: 103130

E-mail: xsmrecek@stuba.sk

Predmet: PKS

Deň a čas cvičenia: Pondelok 15:00

Semester: ZS 20/21

Ročník: 2.

Obsah

1. Zadanie úlohy

Navrhnite a implementujte program s použitím vlastného protokolu nad protokolom UDP (User Datagram Protocol) transportnej vrstvy sieťového modelu TCP/IP. Program umožní komunikáciu dvoch účastníkov v lokálnej sieti Ethernet, teda prenos textových správ a ľubovoľného súboru medzi počítačmi (uzlami).

Program bude pozostávať z dvoch častí – vysielacej a prijímacej. Vysielací uzol pošle súbor inému uzlu v sieti. Predpokladá sa, že v sieti dochádza k stratám dát. Ak je posielaný súbor väčší, ako používateľom definovaná max. veľkosť fragmentu, vysielajúca strana rozloží súbor na menšie časti - fragmenty, ktoré pošle samostatne. Maximálnu veľkosť fragmentu musí mať používateľ možnosť nastaviť takú, aby neboli znova fragmentované na linkovej vrstve.

Ak je súbor poslaný ako postupnosť fragmentov, cieľový uzol vypíše správu o prijatí fragmentu s jeho poradím a či bol prenesený bez chýb. Po prijatí celého súboru na cieľovom uzle tento zobrazí správu o jeho prijatí a absolútnu cestu, kam bol prijatý súbor uložený.

Program musí obsahovať kontrolu chýb pri komunikácii a znovuvyžiadanie chybných fragmentov, vrátane pozitívneho aj negatívneho potvrdenia. Po prenesení prvého súboru pri nečinnosti komunikátor automaticky odošle paket pre udržanie spojenia každých 10-60s pokiaľ používateľ neukončí spojenie. Odporúčame riešiť cez vlastne definované signalizačné správy.

2. Návrhu programu a komunikačného protokolu

Program je implementovaný v programovacom jazyku Python s používateľským rozhraním v konzole.

2.1 Nadviazanie a ukončenie spojenia

Po spustení programu v móde „server“ musí používateľ zadať port, na ktorom bude server prijímať komunikáciu. Je nutné zvoliť port z rozsahu 1024–49151, pretože porty menšie ako 1024 sú systémové a well-known porty a väčšie ako 49151 sú ephemeral porty.

Po spustení programu v móde „klient“ musí používateľ zadať IP adresu servera, nachádzajúceho sa v lokálnej sieti. Z dôvodu dištančnej výučby a prezentovania riešenia iba na svojom zariadení nastavujeme túto IP adresu na loopback `127.0.0.1`. Následne je nutné zadať port servera, na ktorom bude server počúvať.

Po zadaní IP a portu klient automaticky odošle signalizačnú správu pre otvorenie spojenia s príslušným flagom. Server signalizačnú správu spracuje, overí jej formát, zistí z nej port klienta a pošle signalizačnú správu klientovi pre potvrdenie otvorenia spojenia s príslušným flagom. Klient ju taktiež overí a ak je spojenie úspešne nadviazané, klient aj server

vypíšu hlásenie. Ak spojenie nie je úspešne nadviazané, program vypíše hlásenie, zatvorí socket na oboch stranách a opätovne ponúkne voľbu módu.

Ak je spojenie úspešne nadviazané, otvorí sa v programe klienta menu, kde si používateľ vyberie, či chce preniesť dáta (text alebo súbor), alebo chce zapnúť/ vypnúť udržiavanie spojenia pomocou keepalive správ, alebo uzatvoriť spojenie explicitne. Ak si používateľ vyberie prenos dát, vyberie, či chce preniesť text alebo súbor a vyberie maximálnu veľkosť fragmentu. Potom sa prenesú dáta. Po prenose dát sa klient vráti do menu. Ak klient zapne posielanie keepalive správ, začnú sa posielať keepalive správy s príslušným flagom v stanovených časových intervaloch. Ak používateľ nezvolí zapnutie keepalive správ, alebo ich po zapnutí neskôr vypne, server po troch chýbajúcich keepalive správach vyhodnotí spojenie ako spojenie bez akejkoľvek komunikácie a ukončí ho. Spojenie sa ukončí poslaním signalizačnej správy o ukončení spojenia s príslušným flagom. Ak používateľ v menu zvolí explicitné ukončenie spojenia, pošlú sa signalizačné správy s príslušným flagom o ukončení spojenia. Po akomkoľvek ukončení spojenia sa sockety zatvoria a program sa prepne do výberu módu server/ klient.

2.2 Štruktúra hlavičky

<table>
<tr><td colspan="2">Source Port</td><td>Destination Port</td></tr>
<tr><td colspan="2">Length</td><td>Checksum</td></tr>
<tr><td colspan="3">Poradové číslo</td></tr>
<tr><td colspan="3">CRC</td></tr>
<tr><td colspan="2">Veľkosť (veľkosť hlavičky + veľkosť dát)</td><td>Flag</td><td>Dáta</td></tr>
<tr><td colspan="3">Dáta (pokračovanie)</td></tr>
</table>

Source Port, Destination Port, Length a Checksum sú polia UDP hlavičky, ktoré hlavička obsahuje predvolene.

Pole Poradové číslo označuje poradie fragmentu v súbore. Vďaka nemu je možné zoradiť fragmenty do správneho poradia, aj keď prídu v rôznom poradí.

Pole CRC je kontrolný súčet hlavičky a dát. Klient ho vypočíta a zapíše do hlavičky, s tým, že počas výpočtu toto pole považuje za nulové. Server ho vypočíta samostatne a overí s tým, ktoré dostal doručené. Ak sa serverom vypočítané a doručené CRC nezhoduje, tak server označí fragment za chybný.

Pole Veľkosť (veľkosť hlavičky + veľkosť dát) určuje veľkosť v bajtoch, ktorú má mnou navrhnutá hlavička (polia, ktoré sú označené v tabuľke modrou) + veľkosť prenášaných dát.

Pole Flag obsahuje jeden bajt určujúci typ správy. Podľa flagu dokáže program očakávať, čo bude uložené v poli Dáta (názov súboru, fragment dát, číslo nesprávne doručeného fragmentu, ...). Hodnoty Flagu som navrhol nasledovne:

1. a určuje inicializáciu spojenia

2. `b` určuje posielanie fragmentu textovej správy
3. `c` určuje posielanie posledného fragmentu textovej správy
4. `d` určuje posielanie názvu súboru
5. `e` určuje posielanie fragmentov súboru
6. `f` určuje posielanie posledného fragmentu súboru
7. `g` určuje ukončenie spojenia
8. `p` určuje pozitívne potvrdenie
9. `n` určuje negatívne potvrdenie
10. `k` určuje keepalive

UDP hlavička sama o sebe má 8 bajtov. Poradové číslo a CRC sú premenné typu `unsigned int`, Veľkosť je premenná typu `short`, Flag je premenná typu `char`. Mnou navrhnutá hlavička má teda 11 bajtov. Celá UDP hlavička má teda 19 bajtov.

Pole Dáta môže prenášať 0 až 1461 bajtov. 1461 bajtov je maximum, ktoré je možné preniesť bez fragmentácie na linkovej vrstve. Táto hodnota sa počíta ako maximálna veľkosť payloadu Ethernet II rámca – veľkosť IP hlavičky – veľkosť UDP hlavičky – veľkosť vlastnej hlavičky, tj. 1500 – 20 – 8 – 11 = 1461. Pole Dáta slúži na prenos samotných dát (súboru alebo textovej správy), alebo názvu súboru, alebo na prenos čísel nesprávne doručených fragmentov. V prípade fragmentácie súboru, alebo textovej správy môže byť maximálna veľkosť fragmentu zvolená používateľom od 1B do 1461B, keďže to je maximálna veľkosť, ktorú dokáže pole dát preniesť bez fragmentácie na linkovej vrstve.

2.3 Metóda kontrolnej sumy

Pre potreby kontroly integrity informácií v hlavičke a samotných dát som vybral ako metódu kontroly CRC (Cyclic Redundancy Check), teda kontrolu cyklickým kódom. Na výpočet hodnoty CRC používam funkciu `mkCrcFun` z balíka `crcmod`.

CRC má v mnou navrhnutej hlavičke vlastné pole. CRC sa vypočítava z hlavičky a prenášaných dát. Klient hodnotu CRC vypočíta a zapíše do hlavičky, s tým, že počas výpočtu pole CRC v hlavičke považuje za nulové. Server, po prijatí paketu vypočíta vlastné CRC, kde taktiež počas výpočtu považuje pole CRC za nulové. Server vlastné vypočítané CRC overí s tým, ktoré dostal doručené v hlavičke. Ak sa serverom vypočítané a v hlavičke doručené CRC nezhodujú, tak server označí fragment za chybný a opätovne si ho vyžiada podľa pravidiel použitej ARQ metódy.

2.4 Metóda vnesenia chyby do prenosu

Do prenosu je vložená chyba takým spôsobom, že pri výpočte CRC na strane klienta je výsledok výpočtu zmenšený o 1 a až potom uložený do hlavičky. Takto odoslaný paket bude na strane servera, po opätovnom výpočte CRC, vyhodnotený ako chybný, pre nezhodujúce sa hodnoty CRC.

V programe je možnosť zapnúť vnášanie chyby do paketu. Chyba bude vnesená do prvého paketu nesúceho dáta. Teda pri textovej správe to bude úplne prvý paket, pri prenose súboru to bude druhý paket, pretože prvý paket obsahuje názov súboru a poškodzujem iba dátové pakety.

2.5 Metóda ARQ

Selective Repeat ARQ je metóda, ktorá opätovne vyžiada iba pakety, ktoré boli vyhodnotené ako poškodené, alebo nedoručené.

Server postupne potvrdzuje prijatie fragmentov od klienta tak, že klient na toto potvrdenie nečaká, ale posiela ďalšie fragmenty dovtedy, dokým nevyčerpá možnosti svojho okna. Ak klient vyčerpal možnosti svojho okna a má od servera doručené potvrdenia o doručení paketov, posúva okno a pokračuje v prenose ďalších fragmentov. Ak nejakému fragmentu vyprší čas čakania na potvrdenie doručenia, alebo príde od serveru negatívne potvrdenie značiace príjem chybného paketu, klient opätovne tento paket odošle. Týmto sa zabezpečí, že serveru budú doručené všetky pakety. V prípade, že sú pakety doručené v nesprávnom poradí, server ich usporiada do správneho poradia.

2.6 Metóda udržiavania spojenia

Server automaticky uzatvorí spojenie po tom, ako nedostane 3 keepalive správy, alebo žiadne dáta.

Klient po nadviazaní spojenia má v menu možnosť zvoliť, či chce posielať dáta (text alebo súbor), udržiavať spojenie pomocou keepalive správ, alebo explicitne ukončiť spojenie. V prípade voľby explicitného ukončenia spojenia pošle klient správu s príslušným flagom a server na ňu odpovie správou s príslušným flagom pre ukončenie spojenia. Následne server aj klient zavrú socket a program sa vráti do výberu módu.

Ak je u klienta aktívne posielanie keepalive správ, posiela keepalive správu s príslušným flagom serveru každých 10 sekúnd. Na každú keepalive správu server odpovie správou s príslušným flagom. Ak neprebieha prenos dát a posielanie keepalive správ je prerušené, server po 3 neprijatých keepalive správach spojenie ukončí vyslaním signalizačnej správy s príslušným flagom o ukončení prenosu a následne zatvorí socket a vráti sa výberu módu. Ak klient obdrží správu o ukončení spojenia od serveru, zatvorí socket a vráti sa do výberu módu programu.

Ak je u klienta aktívne posielanie keepalive správ a zároveň chce používateľ odoslať dáta, počas odosielania dát je posielanie keepalive správ prerušené a obnovené je až po úspešnom odoslaní všetkých dát.

2.7 Diagram spracovávania komunikácie

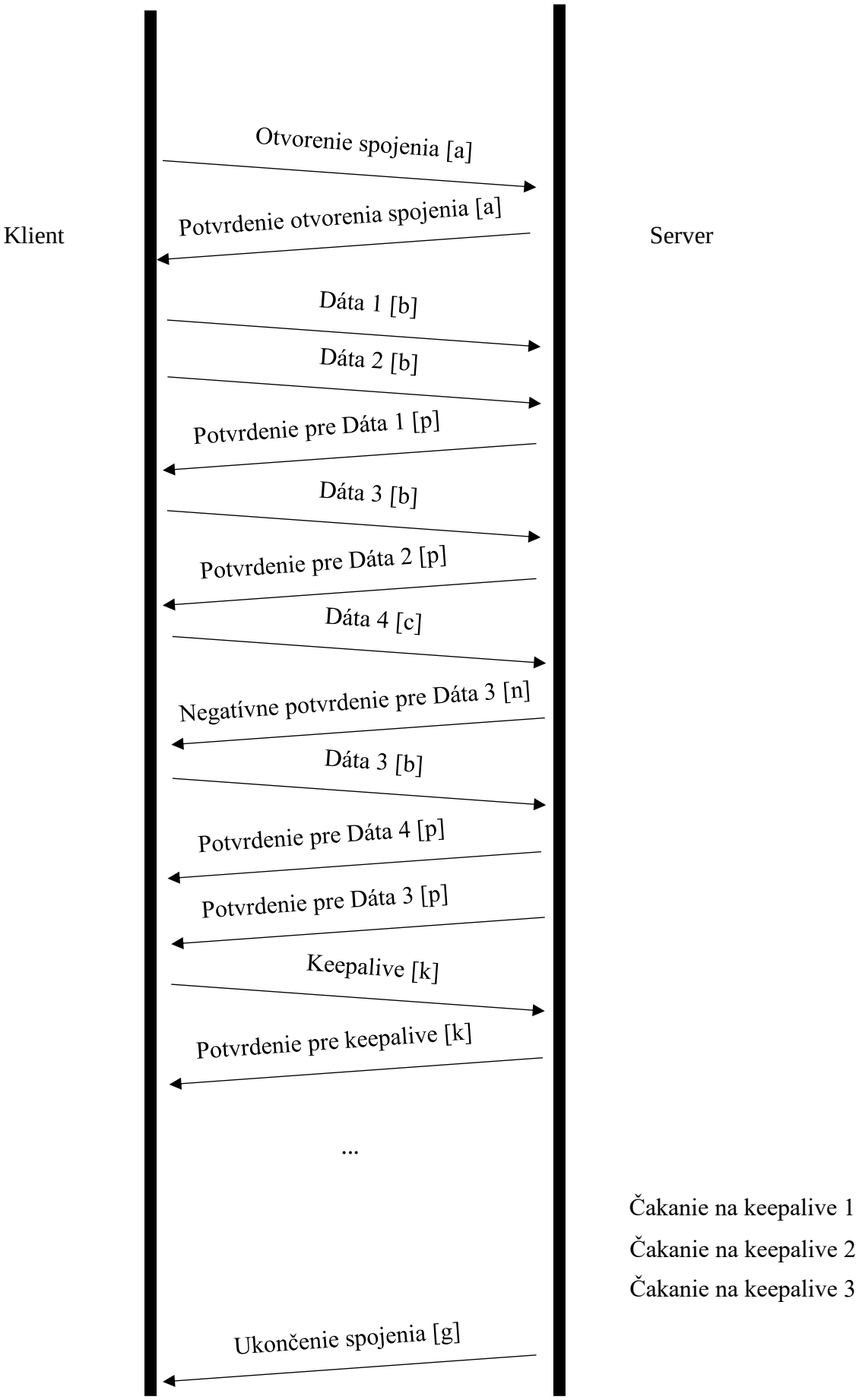

Klient
Server
Otvorenie spojenia [a]
Potvrdenie otvorenia spojenia [a]
Dáta 1 [b]
Dáta 2 [b]
Potvrdenie pre Dáta 1 [p]
Dáta 3 [b]
Potvrdenie pre Dáta 2 [p]
Dáta 4 [c]
Negatívne potvrdenie pre Dáta 3 [n]
Dáta 3 [b]
Potvrdenie pre Dáta 4 [p]
Potvrdenie pre Dáta 3 [p]
Keepalive [k]
Potvrdenie pre keepalive [k]
...
Čakanie na keepalive 1
Čakanie na keepalive 2
Čakanie na keepalive 3
Ukončenie spojenia [g]

Diagram zobrazuje nadviazanie spojenia medzi klientom a serverom. Po nadviazaní spojenia klient odosiela textovú správu zloženú zo 4 fragmentov, kde 3. fragment je poškodený a bude nutné ho odoslať znova.

Prvé tri fragmenty sú označené flagom `b`, posledný je označený flagom `c`, ktorý symbolizuje, že je to posledný fragment. Okno má pre vizuálnosť nastavenú veľkosť 2. Fragment Dáta 3 simuluje prenos chyby, server túto chybu odhalí a pomocou správy s flagom `n` si fragment 3 vyžiada znova. Fragment je opätovne poslaný, bez straty fragmentu 4, ktorý bol poslaný korektne, no ešte pred korektným doručením fragmentu 3. Každý korektne prijatý fragment server potvrdí správou s flagom `p`. Klient nečaká na tieto správy po každom fragmente, ale prichádzajú postupne a posúvajú tak okno.

Alternatívne, ak klient vyčerpá možnosti svojho okna a zároveň neprišla žiadna pozitívna alebo negatívna potvrdzujúca správa, nemôže poslať ďalší fragment, pokým nejakému fragmentu nevyprší čas na prijatie potvrdzujúcej správy od servera. Ak nejakému fragmentu vyprší čas na prijatie potvrdzujúcej správy, klient predpokladá, že fragment nebol doručený a klient pošle fragment ktorému chýba potvrdenie automaticky znova.

Spojenie sa v tomto konkrétnom diagrame ukončuje po 3 nedoručených keepalive správach. Alternatívne je možné ho ukončiť z klientovej strany odoslaním správy s flagom `g`, na ktorú by server odpovedal správou s flagom `g`.

Diagram zobrazuje prenos textových dát, alternatívne v prípade prenosu súboru najskôr klient odosiela paket s flagom `d` nesúcim meno súboru a potom posiela fragmenty súboru s flagmi `e`, pričom posledný fragment má flag `f`.

V diagrame neuvádzam šípky potvrdení a negatívnych potvrdení začínajúce v ústí šípky fragmentu dát, ako by to po správnosti malo byť. Pre prehľadnosť diagramu šípky potvrdení a negatívnych potvrdení uvádzam tak, aby sa nekrižovali. V diagrame nad šípkami uvádzam popis operácie a v [] flag, ktorý je zapísaný v hlavičke a značí typ paketu.

2.8 Popis kódu

Program som implementoval v programovacom jazyku Python. V programe som použil balík `socket` na prenos správ po sieti, balík `struct` na vytváranie štruktúr podobných jazyku C a balík `crcmod` na počítanie CRC kódu, ktorý používam ako checksum.

3. Zmeny oproti návrhu

V tomto dokumente uvádzam celý, nezmenený pôvodný návrh programu a komunikačného protokolu. V tejto kapitole popíšem zmeny oproti návrhu, ktoré som urobil počal samotnej implementácie.

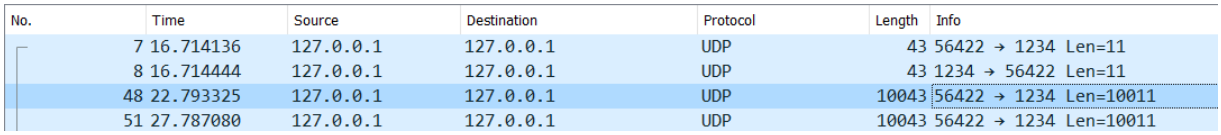
3.1 Zmena ukončenia spojenia

Spojenie sa ukončí explicitne odoslaním správy s príslušným flagom ak sa odhlási klient. Ak vyprší lehota na prijatie keepalive na serveri, tak sa spojenie s klientom považuje za neaktívne a žiadne správy pre jeho ukončenie sa neposielajú. Rovnako to funguje, ak server neodpovie na keepalive správy klientovi, vtedy klient považuje server za neaktívny.

3.2 Metóda kontrolnej sumy – doplnenie odôvodnenia

Pre potreby riešenia som zvolil 32b funkciu pre výpočet CRC, teda CRC-32. Používam štandardný generujúci polynóm s hodnotou 0x04C11DB7. Takýto výpočet prináša takmer 100% úspešnú kontrolu integrity dát a hlavičky. Pre potreby kontroly integrity 1472B hlavičky a dát, ktoré dokáže prenášať môj program v Ethernete II maximálne, by stačilo použiť aj CRC-16 a výsledky by boli podobne dobré.

Problém nastáva s testovaním na localhoste, kde som empiricky zistil, že dáta nie sú prenášané v Ethernet II rámcoch a ich maximálna veľkosť teda nie je 1500B. V mojom programe som ošetril vstupy, aby žiaden paket nepresiahol veľkosť dát 1500B, ale keby som to neošetril, bolo by možné na localhoste úspešne poslať a prijať aj 10000B textu v jednom pakete, bez toho, aby bol akokoľvek fragmentovaný na linkovej vrstve. Dôkaz na obrázku nižšie.



No.	Time	Source	Destination	Protocol	Length	Info
7	16.714136	127.0.0.1	127.0.0.1	UDP	43	56422 → 1234 Len=11
8	16.714444	127.0.0.1	127.0.0.1	UDP	43	1234 → 56422 Len=11
48	22.793325	127.0.0.1	127.0.0.1	UDP	10043	56422 → 1234 Len=10011
51	27.787080	127.0.0.1	127.0.0.1	UDP	10043	56422 → 1234 Len=10011

Hoci teda môj program ošetruje vstupy a nie je možné poslať viac ako 1461B dát + 11B hlavičky, teoreticky je možné na localhoste, minimálne na mojom zariadení, posielať oveľa väčšie fragmenty dát, ako je možné cez Ethernet II. Z tohto dôvodu ponechávam v programe 4B dlhú hodnotu CRC vytvorenú pomocou CRC-32, aby bolo CRC vierohodné a spoľahlivé pri akejkoľvek veľkosti dát.

3.3 Zmena metódy ARQ

Z dôvodu nedostatku času som nezvládol implementovať komplexnejšiu metódu ARQ a miesto Selective Repeat ARQ som použil Stop-and-wait ARQ.

Klient odosiela fragmenty po jednom a vždy čaká na potvrdenie zo servera. Potvrdenie zo servera môže prísť kladné a klient pošle nasledujúci fragment. V prípade, že príde negatívne potvrdenie zo servera, klient pošle paket znova a opäť čaká na potvrdenie serverom. V prípade, že klient na paket nedostane odpoveď do 5s, odošle ho znova. Ak ho po pôvodnom odoslaní klient 3 krát pošle opätovne bez akejkoľvek odozvy servera, klient vyhodnotí server ako neaktívny a ďalšie pakety neposiela.

Táto metóda ARQ zabezpečí, že na aktívny server budú doručené všetky fragmenty. Potvrdzujúce správy zo servera s príslušným flagom nesú vo svojom poli dát čísla fragmentu, ktorý potvrdzujú.

3.4 Úprava metódy udržiavania spojenia

Ak klient explicitne ukončí spojenie so serverom odhlásením, odošle správu s príslušným flagom a ukončí spojenie. Server okamžite po prijatí takejto správy ukončí spojenie tiež a potvrdzujúcu správu neodosiela.

Interval posielania keepalive správ som zmenil z 10s na 30s, kvôli veľkej rušivosti výpisov v konzole.

Ak server neobdrží 3 keepalive správy, uzatvorí spojenie, ale klienta o tom správou neinformuje, implicitne sa predpokladá, že klient je neaktívny, keď neposielal keepalive.

Zmenou oproti návrhu je aj to, že klient taktiež monitoruje, či server na keepalive správy odpovedá. Ak klient nedostane odpoveď na 3 keepalive správy, predpokladá, že server je neaktívny a vypíše hlásenie.

3.5 Zmena a doplnenie diagramu spracovávania komunikácie

Keďže som zmenil metódu ARQ, predošlý diagram nie je aktuálny, preto ho mením. Taktiež pridávam aj diagram komunikácie ako vývojový diagram programu.

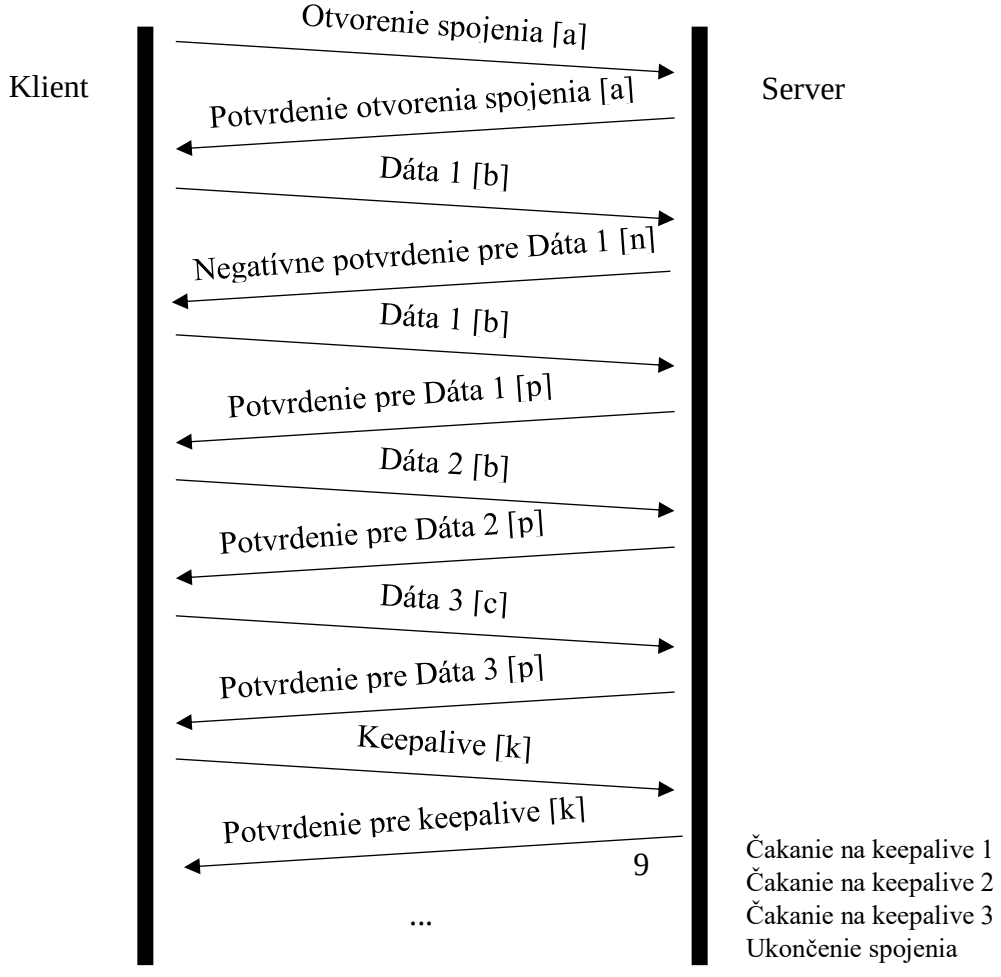
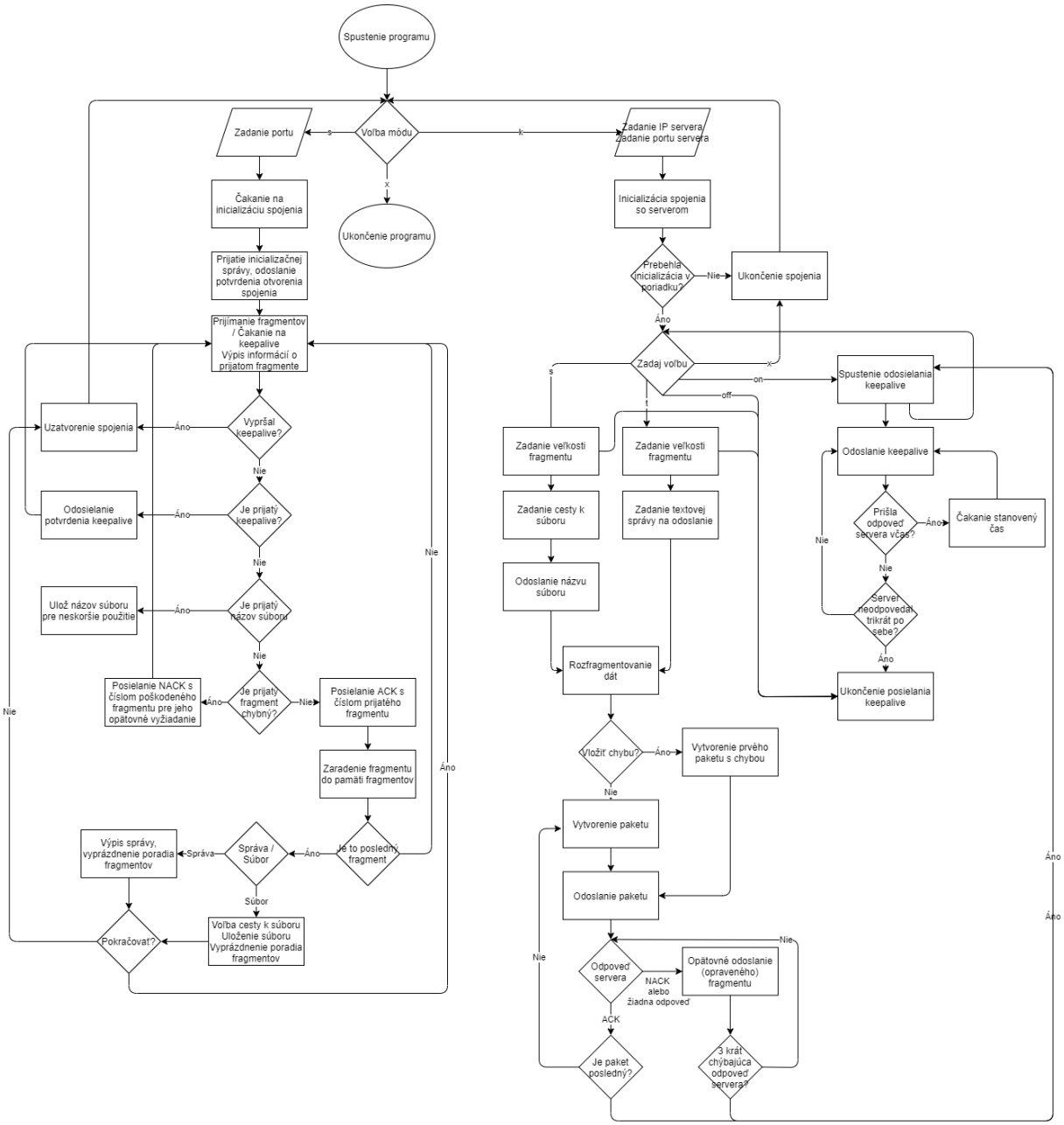


Diagram zobrazuje nadviazanie spojenia medzi klientom a serverom. Po nadviazaní spojenia klient odosiela textovú správu zloženú z 3 fragmentov, kde 1. fragment je poškodený a bude nutné ho odoslať znova.

Prvé dva fragmenty sú označené flagom b, posledný je označený flagom c, ktorý symbolizuje, že je to posledný fragment. Fragment Dáta 1 simuluje prenos chyby, server túto chybu odhalí a pomocou správy s flagom n si fragment 1 vyžiada znova. Fragment je opätovne poslaný a až potom pokračuje komunikácia. Každý korektne prijatý fragment server potvrdí správou s flagom p. Klient čaká potvrdzovacie správy po každom fragmente.

Spojenie sa v tomto konkrétnom diagrame ukončuje po 3 nedoručených keepalive správach. Alternatívne je možné ho ukončiť z klientovej strany odoslaním správy s flagom g,



Tento diagram zobrazuje celú funkčnosť programu aj so zobrazením riešenia hraničných situácií. Plnú verziu vývojového diagramu odovzdávam v samostatnom súbore.

4. Finálna dokumentácia programu

Keďže som sa pri programovaní zadania držal návrhu a prípadné zmeny som opísal vyššie, stručne popíšem funkčnosť a ovládanie programu. Vývojový diagram programu je uvedený v časti Zmena a doplnenie diagramu spracovávania komunikácie.

4.1 Server – prijímač

Po spustení programu a po zvolení módu serveru musí používateľ zadať port. Po zadaní portu server čaká na inicializáciu.

```
Zvol s pre server, zvol k pre klient, zvol x pre skoncenie programu: s
SERVER - Zadaj port: Zadaj cislo z intervalu <1024 az 49151>: 4242
SERVER - Zvoleny port 4242
```

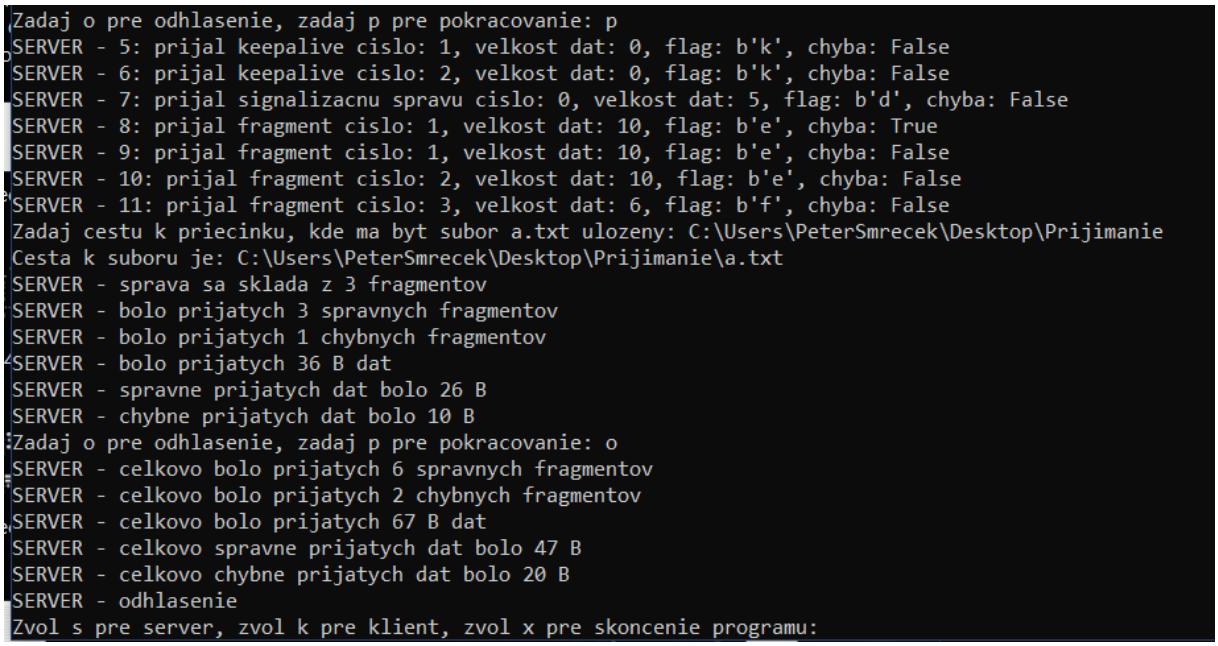
Po inicializácií a nadviazaní spojenia je vypísané hlásenie a server čaká na prijatie fragmentov.

```
Zvol s pre server, zvol k pre klient, zvol x pre skoncenie programu: s
SERVER - Zadaj port: Zadaj cislo z intervalu <1024 az 49151>: 4242
SERVER - Zvoleny port 4242
SERVER - Prijatie otvorenia spojenia z adresy ('127.0.0.1', 64317)
SERVER - Spojenie s ('127.0.0.1', 64317) bolo uspesne nadviazane
```

Server môže prijať textovú správu. Ak prijme textovú správu, vypíše štatistiku prijímania dát a vypíše aj samostatnú správu. Následne čaká, či sa chce používateľ odhlásiť, alebo prijať ďalšie dáta. Počas tejto doby čakania na voľbu používateľa sa neposielajú odpovede na keepalive správy a preto ak používateľ dlho nepotvrdí pokračovanie, klient vyhodnotí server ako neaktívny.

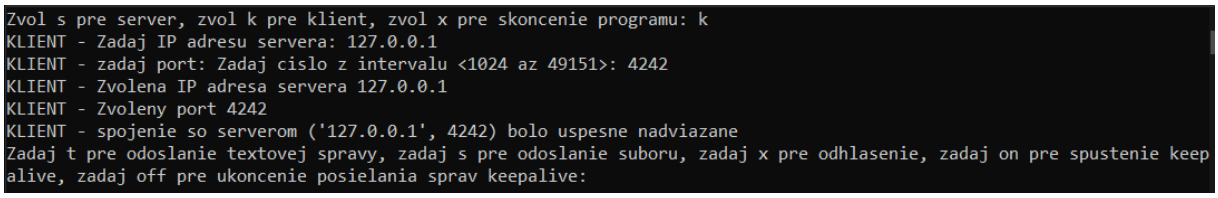
```
Zvol s pre server, zvol k pre klient, zvol x pre skoncenie programu: s
SERVER - Zadaj port: Zadaj cislo z intervalu <1024 az 49151>: 4242
SERVER - Zvoleny port 4242
SERVER - Prijatie otvorenia spojenia z adresy ('127.0.0.1', 64317)
SERVER - Spojenie s ('127.0.0.1', 64317) bolo uspesne nadviazane
SERVER - 1: prijal fragment cislo: 1, velkost dat: 10, flag: b'b', chyba: True
SERVER - 2: prijal fragment cislo: 1, velkost dat: 10, flag: b'b', chyba: False
SERVER - 3: prijal fragment cislo: 2, velkost dat: 10, flag: b'b', chyba: False
SERVER - 4: prijal fragment cislo: 3, velkost dat: 1, flag: b'c', chyba: False
SERVER - sprava sa sklada z 3 fragmentov
SERVER - bolo prijatych 3 spravnych fragmentov
SERVER - bolo prijatych 1 chybnych fragmentov
SERVER - bolo prijatych 31 B dat
SERVER - spravne prijatych dat bolo 21 B
SERVER - chybne prijatych dat bolo 10 B
SERVER - cele znenie spravy:
Teraz testujem server
Zadaj o pre odhlasenie, zadaj p pre pokracovanie:
```

V prípade prijatia súboru server vyzve používateľa na zadanie cesty na uloženie. Potom súbor uloží, vypíše cestu, kam sa súbor uložil a štatistiku prijímania dát. Opäť sa server spýta používateľa, či chce pokračovať. Po zvolení možnosti odhlásiť sa, sa program vráti do menu výberu módu.

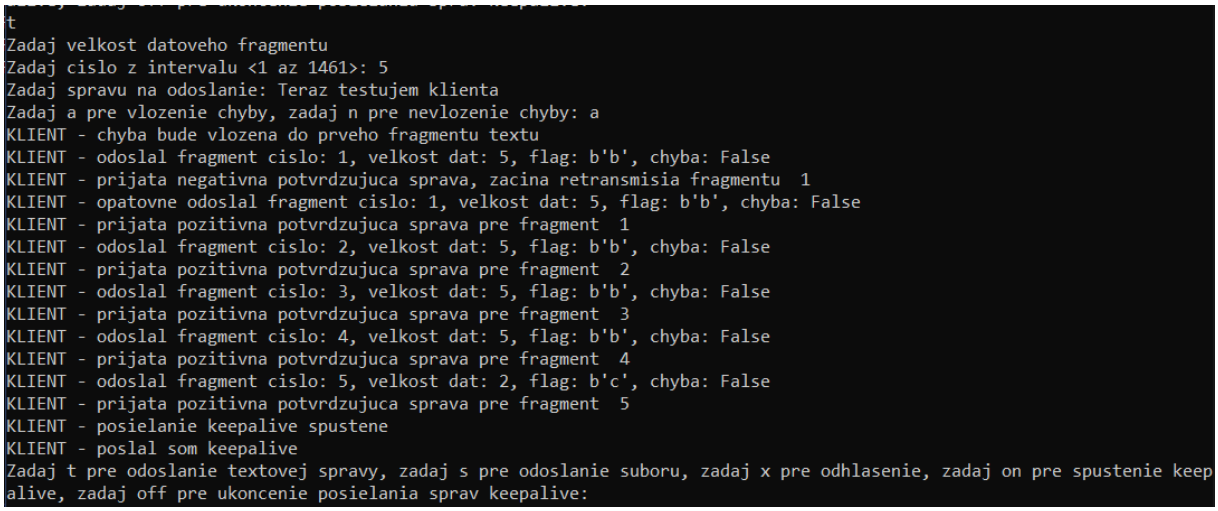


4.2 Klient – vysielač

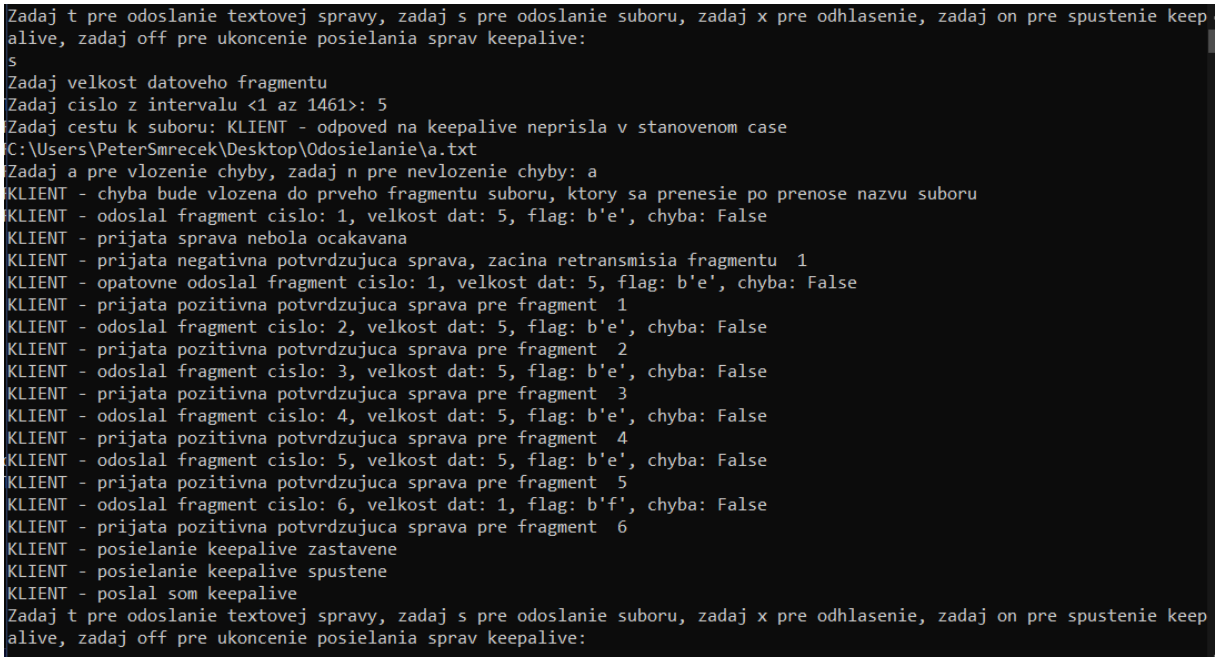
Po spustení programu v režime klient je nutné zadať adresu servera a port, na ktorom bude prebiehať komunikácia. Následne sa nadviaže spojenie. V prípade úspechu sa vypíše menu, v prípade neúspechu sa program vráti do výberu režimu. Obrázok nižšie zobrazuje stav po úspešnom nadviazaní spojenia.



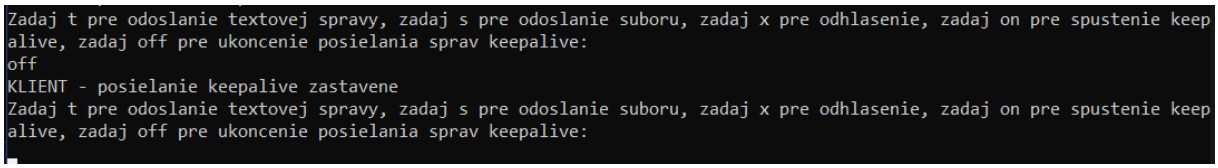
V menu si používateľ vyberie, či zvolí `s` pre odoslanie súboru, `t` pre odoslanie textu, `on` pre zapnutie posielania keepalive správ, `off` pre vypnutie posielania keepalive správ alebo `x` pre návrat do menu výberu režimu. Obrázok nižšie zobrazuje stav po poslaní textovej správy.



Obrázok nižšie zobrazuje stav konzoly po poslaní súboru.



Obrázok nižšie zobrazuje stav konzoly po vypnutí posielania správ keepalive.



4.3 Popis kódu

Program som implementoval v programovacom jazyku Python s využitím balíkov `socket` pre komunikáciu nad protokolom UDP, `crcmod` na výpočet CRC, `os` na prácu s cestami k súborom, `threading` a `time` na prácu s niťami a časovaním. Balík `struct` som nakoniec nepoužil.

Vo funkcií `main` je základná voľba režimu programu. Vo funkcií `server_riadic` sa nastavujú možnosti servera, nadväzuje spojenie a volá sa funkcia prijímania dát. Vo funkcií `klient_riadic` sa nastavujú možnosti klienta, nadväzuje spojenie so serverom a vypisuje menu, na základe ktorého sú volané funkcie pre sprostredkovanie funkcionality klienta.

Komentáre k samostatným funkciám do tejto dokumentácie neuvádzam, sú v kóde v doc komentároch pri každej funkcií.

4.4 Záver

V programovacom prostredí Python som implementoval UDP komunikátor schopný fungovať ako klient alebo server podľa voľby používateľa. Vytvoril som vlastný protokol na

komunikáciu, teda prenos dát a posielanie signalizačných správ. Program dokáže simulovať chybu prenosu a opätovne odoslať chybný fragment použitím Stop-and-wait ARQ. Po prenesení textu alebo dát klient odosiela keepalive správy serveru pre udržanie spojenia v pevných časových intervaloch.

Program je schopný pracovať na localhoste aj na rôznych zariadeniach v tej istej podsieti s tým, že server počúva na zadanom porte a klient funguje na vlastnom, náhodne zvolenom porte.