Dokumentacia zadania č. 2*

Hlib Kokin ID: 117991
Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
xkokin@stuba.sk

08.12.2022

*Dokumentacia zadania čislo 2 v predmete Počítačové a komunikačné siete, ak. rok 2022/23, vedenie: Ing. Miroslav Bahleda, PhD.

Abstrakt

Dokumentácia úlohy obsahuje úpravy vykonané v návrhu riešenia, popisy algoritmov výmeny signálov implementovaných v riešení, ako aj popis práce vlákien v programe.

Obsah

1 Upravy v návrhu riešenia a vlastnosti implementácie **4**
1.1 Štruktúra hlavičky vlastného protokolu a jej nastavenie 4
1.1.1 Flagy protokolu 4
1.1.2 Metóda kontrolnej sumy 5
1.2 Spracovávanie komunikácie 6
1.2.1 Otvorenie 6
1.2.2 Zatvorenie 6
1.2.3 Posielanie udajov 7
1.2.4 ARQ 8
1.2.5 Metóda pre udržanie spojenia 9
1.2.6 Zmena role 10
1.2.7 Odoslanie správy s poškodeným fragmentom 11

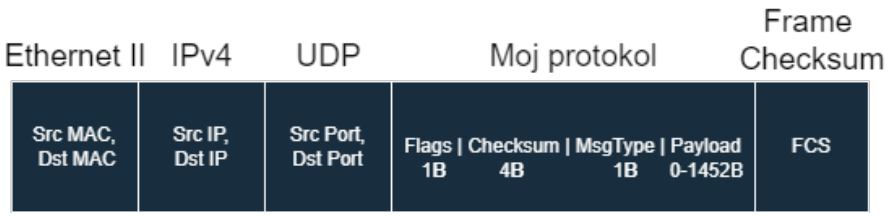
2 Štruktúra programu **12**
2.1 Spustenie programu 12
2.2 Vlákna 12

3 Príklady práce v rámci programu **13**
3.1 Príklady zmeny úlohy 13
3.2 Príklady odosielania a prijímania údajov 15
Dokumentacia zadania č. 2

1 Upravy v návrhu riešenia a vlastnosti implementácie

1.1 Štruktúra hlavičky vlastného protokolu a jej nastavenie

Bola implementovaná nasledujúca štruktúra hlavičky protokolu: Začíname prvým bajtom hlavičky a ten obsahuje číslo od 0 do 255 predstavujúce špecifické príznaky nastavené v pakete, potom 4 bajty uchovávajú kontrolný súčet vypočítaný z údajov odoslaných v pakete, nasleduje 1 bajt obsahujúci symbol zodpovedný za typ odoslanej správy (textová správa "Tälebo súbor "F"), ďalšie bajty sú bajty s údajmi správy. Rozhodnutie odstrániť 2 bajty, ktoré udržiavajú dĺžku paketu, bolo prijaté z dôvodu, že nebolo potrebné pridávať nuly do paketu, aby bola minimálna dĺžka paketu 64 bajtov, keďže podmienka minimálnej veľkosti paketu bola nepovinná. Do záhlavia bol pridaný ďalší bajt obsahujúci typ správy, aby bolo možné správne spracovať paket s príznakom PUSH (pre textové a súborové správy).



Obr. 1: Diagram z reprezentaciou štruktury hlavičky

Pri odosielaní paketu najprv zavoláme funkciu, ktorej odovzdáme údaje, ktoré sa majú odoslať, príznaky, ktoré sa majú nastaviť v pakete, typ správy (súbor, text) a logickú hodnotu, ktorá určí, či je fragment poškodený alebo nie. Vo funkcii nastavíme zložky hlavičky protokolu v poradí.

1.1.1 Flagy protokolu

Nižšie je uvedený zoznam príznakov, ktoré používaju sa v protokole, a ich účel.

Najprv máme v reťazci osem núl a prechádzame jednotlivé kontroly a vypĺňame ich jednotkami. Každý z 8 bitov predstavuje prítomnosť príznaku v pakete (0 alebo 1). Po vyplnení riadku číslami 0 a 1 ho prevedieme na int a potom na bajt a zapíšeme ten bajt do hlavičky.

1. ACK: odepoveda sa v prípade pozitívnej odpovede alebo signálu, že je všetko v poriadku.
2. PSH: nastavuje sa keď posielame údaje;
3. SYN: posiela sa, keď chceme otvoriť komunikáciu;
4. FIN: posiela sa ak chceme dobrovoľne ukončiť komunikáciu;

5. RST: posiela sa v prípade nutneho prerušenia komunikácie (ak účastník hovoru neodpovedá);
6. KA(Keep-Alive): odosiela sa na podporu pripojenia;
7. ERR(poškodeny paket): odosiela sa pri prijatí poškodeného balíka alebo pre negatívne odpovede;
8. REQ: odošle žiadosť o výmenu rolí partnerovi s cieľom vymeniť si roly[1];

1.1.2 Metóda kontrolnej sumy

Ako metódu kontrolného súčtu som použil nasledujúci algoritmus: najprv určme, že veľkosť kontrolného súčtu bude 4 bajty, teda 32 bitov; všetky naše údaje rozdelíme na 32-bitové segmenty a všetky ich pridáme; ak sa ukáže, že výsledný súčet je väčší ako 32 bitov, odstránime počet bitov, o ktoré súčet presahuje hranicu 32 bitov, od začiatku výsledného súčtu a pripočítame ich k už 32 bitovému súčtu, inak nechá všetko tak, ako je; potom vezmeme negáciu prijatej sumy - toto bude náš kontrolný súčet, zapíšeme ho do balíka a odošleme príjemcovi; na strane príjemcu opäť vypočítame súčet z dát prenášaných v pakete podľa rovnakého princípu, ako sme vypočítali súčet na strane odosielateľa a potom k výsledku pripočítame kontrolný súčet zaznamenaný v pakete a vezmeme negáciu výsledku, ak je výsledok 0, potom bol paket prenesený bez poškodenia, inak je paket poškodený.

V návrhu riešenia bola veľkosť kontrolného súčtu 16 bitov, ale rozhodol som sa ju zmeniť na 32, aby sa veľké fragmenty nemuseli rozširovať dvakrát tak často, takže kontrolný súčet zaberá 4 bajty. Svoju implementáciu kontrolného súčtu používam na výpočet údajov odosielaných v bežných signáloch (napríklad SYN, FIN, ACK atď., a tiež pre textové správy. Zatiaľ čo pri odosielaní súborov používam funkciu výpočtu crc32 z balíka zlib v Python, robím to preto, lebo nemôžem odovzdať bajty zo súboru do svojej funkcie a správne vypočítať kontrolný súčet na oboch koncoch spojenia a musím odovzdať bajty funkcii crc32.

Stručný opis fungovania crc32: Jedným zo základných parametrov CRC je generujúci polynóm.

$$g(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1.$$

Obr. 2: CRC32 generujúci polynóm

Prvé 32-bitové slovo sa prevezme zo súboru. Ak je najvýznamnejší bit v slove "1", potom sa slovo posunie doľava o jednu číslicu, po ktorej nasleduje operácia XOR na generujúcom polynóme. Ak je najvýznamnejší bit v slove "0", po posune sa nevykoná žiadna operácia XOR. Po posune sa stratí najvýznamnejší bit, na miesto najmenej významného bitu sa načíta ďalší bit zo súboru a operácia sa opakuje, kým sa nenačíta posledný bit zo súboru. Po posunutí celého

[1] Účel príznaku REQ bol zmenený od napísania návrhu riešenia, pôvodne sa mal používať ako žiadosť o odoslanie údajov na server, ale zistilo sa, že to pre túto úlohu nemá zmysel, a preto sa tento príznak používa ako žiadosť o zmenu rolí.

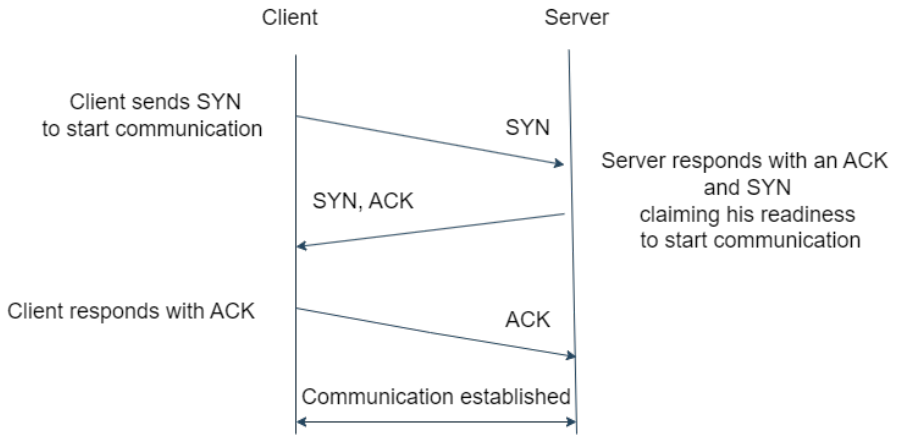
súboru zostane v slove zvyšok, ktorý predstavuje kontrolný súčet. Túto sumu získame v mojej implementácii výpočtu kontrolného súčtu aj v analógii knižnice Python, výsledok prevedieme na bajty pomocou funkcie to_bytes a zapíšeme ho do hlavičky

1.2 Spracovávanie komunikácie

1.2.1 Otvorenie

Komunikácia sa otvára trojcestným podaním rúk

Komunikáciu môže otvoriť len klient a len po spustení servera. Keď sa klient spustí, vykoná funkciu establish_comm a odošle signál SYN na server (ktorý čaká na tento špecifický signál), server tento signál prijme a vykoná funkciu handle_syn, kde odošle SYN, ACK a čaká na odpoveď ACK.

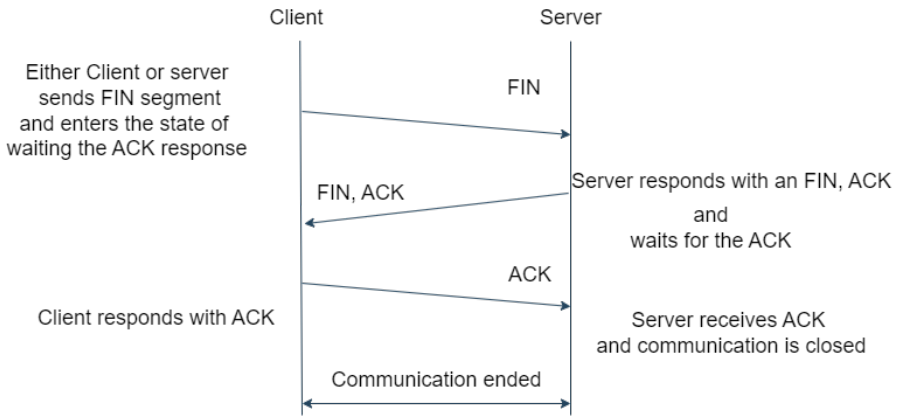


Obr. 3: Diagram fungovania otvorenia spojenia

1.2.2 Zatvorenie

Ukončenie komunikácie sa vykonáva rovnakým spôsobom ako jej otvorenie, a to trojcestným podaním ruky. Ukončenie komunikácie môže inicializovať klient alebo server. Pre tento proces existujú dve funkcie, finish_con a handle_fin, z ktorých prvá sa spúšťa na strane iniciátora ukončenia komunikácie, zatiaľ čo druhá sa spúšťa na druhej strane po prijatí signálu FIN.

V ojedinelých prípadoch je tiež možné, že server alebo klient preruší spojenie bez odpovede, potom pošle svojmu partnerovi signál PST a uzavrie program, ale častejšie algoritmus ARQ pracuje, kým sa partner nevráti.



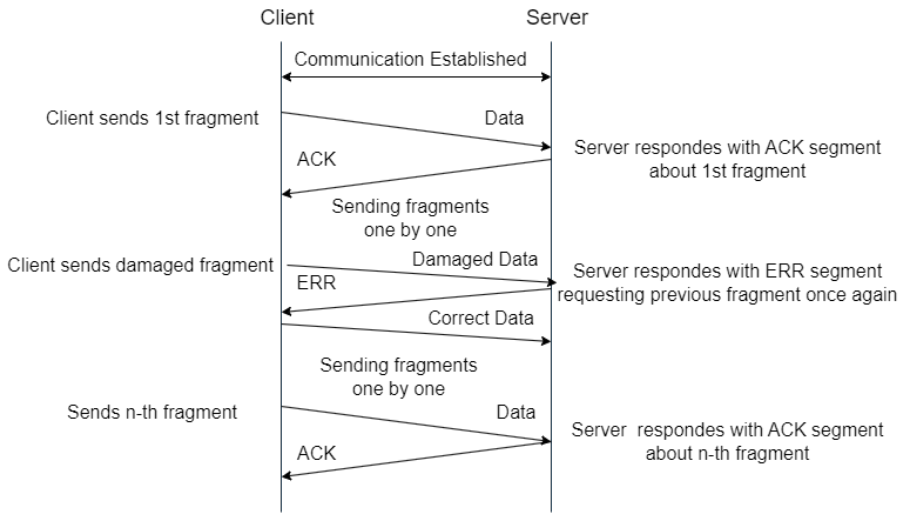
Obr. 4: Diagram fungovania zatvorenia spojenia

1.2.3 Posielanie udajov

Prenos údajov sa vykonáva klasicky na strane klienta. K správe (alebo ku každému fragmentu, ak je fragmentovaný) sa pridá hlavička protokolu a výsledok sa odošle druhej strane, ktorá ako odpoveď očakáva signál ACK alebo ERR. Ak príde signál ACK, funkcia odosielania správy sa ukončí (alebo pokračuje, ak je ešte potrebné odoslať fragmenty), v opačnom prípade sa fragment odošle znova (a bude sa posielať znova a znova, kým nedostane ako odpoveď signál ACK).

Ak je veľkosť odosielanej správy menšia ako zadaná veľkosť fragmentu, k fragmentácii nikdy nedôjde a správa bude odoslaná ako jeden fragment, v opačnom prípade sa zavolá funkcia fragment, ktorá správu rozdelí na fragmenty danej veľkosti a zapíše ich do zoznamu (ak správa nie je fragmentovaná, v zozname bude len jedna položka), funkcie, ktoré vykonávajú fragmentáciu, sa nazývajú fragment_message a fragment_message_f (funkcia pre fragmentáciu súboru a líši sa tým, že vytvára nové objekty triedy byte, nie str).

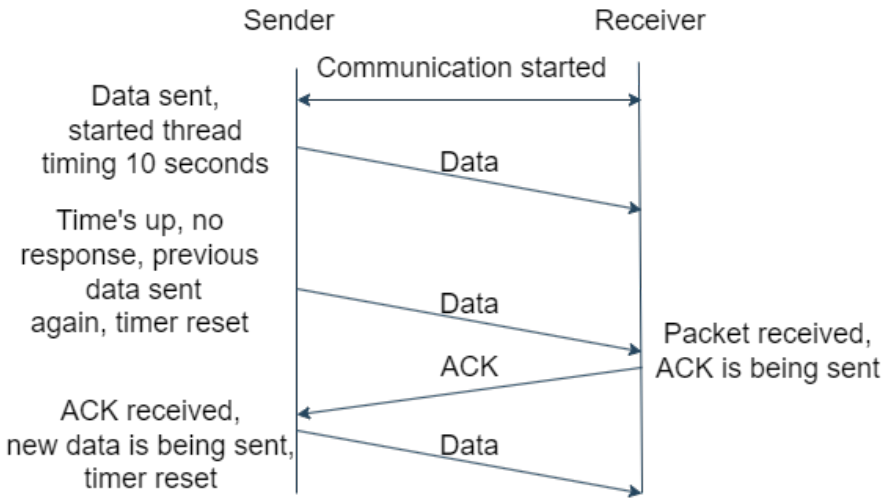
Po vykonaní fragmentácie - funkcia prejde všetky prvky zoznamu, v ktorých boli zapísané fragmenty, a ak je značka chyby pozitívna, funkcia vytvorí v údajoch chybu so 40-percentnou pravdepodobnosťou až do tretieho zasiahnutého fragmentu, v ktorom sa spustí pravdepodobnosť.



Obr. 5: Diagram fungovania posielania udajov

1.2.4 ARQ

Automatic Repeat Request sa spustí po každom odoslanom pakete, na ktorý nebola prijatá odpoveď, ak po 10 sekundách (nastavených ako časový limit v nastaveniach zásuvky) nie je na zásuvku prijatá žiadna odpoveď - predchádzajúci paket/signál sa odošle znova a bude sa posielať v slučke, kým sa naň neodpovie.

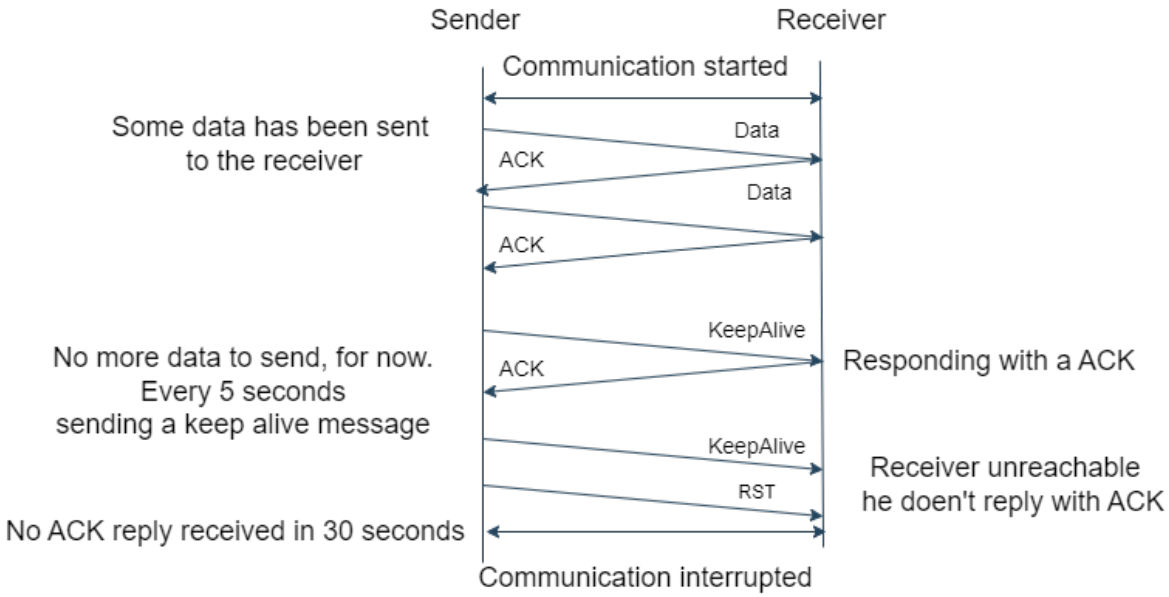


Obr. 6: Diagram fungovania ARQ

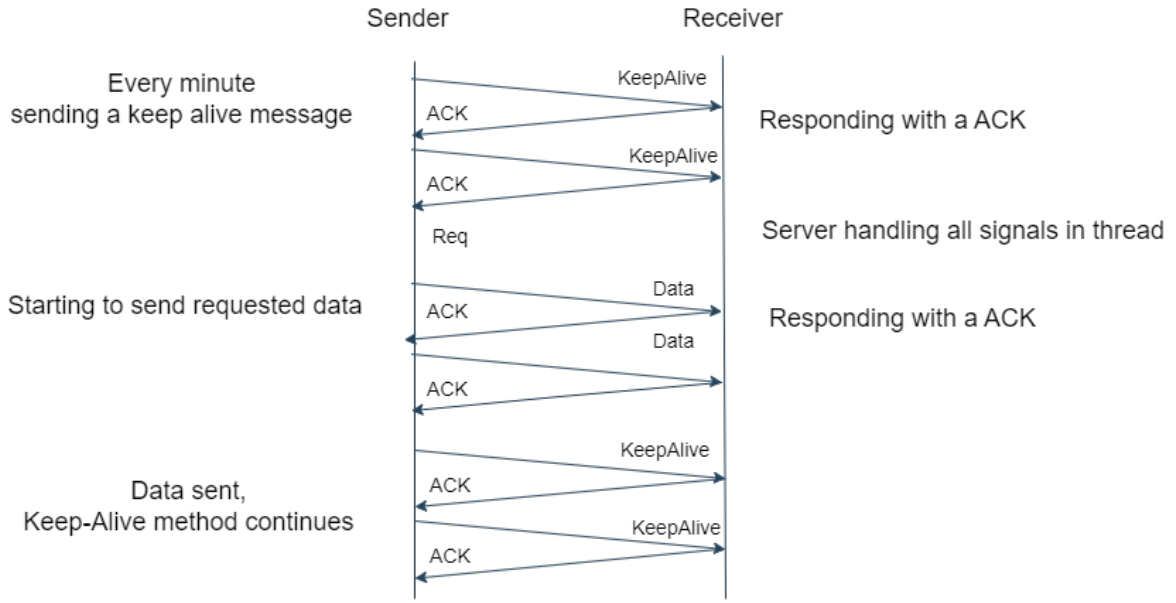
1.2.5 Metóda pre udržanie spojenia

Metóda komunikácie je implementovaná vo vlákne, ktoré beží na oboch stranách spojenia a funguje odlišne pre server a klienta. Klient pri spustení vlákna okamžite pošle serveru signál KA (keep-alive), server vo svojom vlákne čaká 10 sekúnd, kým dostane nejaký signál (nie nevyhnutne KA) a v prípade KA odpovie signálom ASK, na strane klienta sa po prijatí ACK na 5 sekúnd uspí a potom pokračuje vo svojej funkcii. Po odoslaní ACK sa servo vráti k počúvaniu.

Ak klient nedostane ACK do 10 sekúnd, ukončí spojenie. Ak server nedostane signál ACK do 8 sekúnd, pošle ho klientovi znova (v domnienke, že sa signál stratil), toto riešenie zabráni zbytočnému prerušeniu spojenia, pretože časový limit servera je kratší ako časový limit klienta, takže ak sa signál KA od klienta stratí, ACK stále príde klientovi, len neskôr).



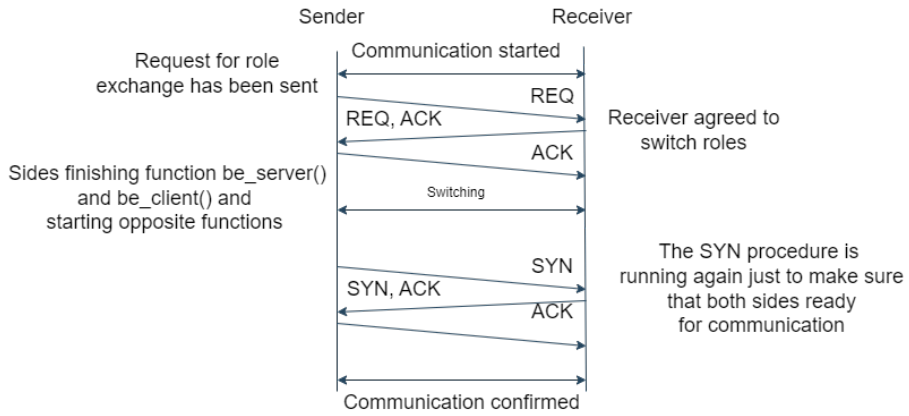
Obr. 7: Diagram fungovania udržania spojenia (Keep-Alive)



Obr. 8: Diagram fungovania udržania spojenia (Keep-Alive)

1.2.6 Zmena role

Pre algoritmus zmeny rolí som použil prvky grafického rozhrania tkinter. Funguje to takto: keď sa klient alebo server rozhodne poslať požiadavku na zmenu roly (zadaním písmena c do príkazového riadku), druhá strana dá pri prijatí tohto signálu na výber, či chce zmeniť rolu alebo nie. Voľba sa vykoná stlačením tlačidla na vyskakovacom okne s jednoduchým rozhraním a otázkou "Súhlasíte so zmenou roly?", ak stlačíte tlačidlo áno, strana, ktorá prijala žiadosť o zmenu roly, pošle kladnú odpoveď s protižiadosťou (3way handshake) a ak dostanete kladnú odpoveď, roly sa zmenia, inak všetko zostane tak, ako bolo, a obe krajiny budú pokračovať vo svojej práci odosielania a prijímania paketov.

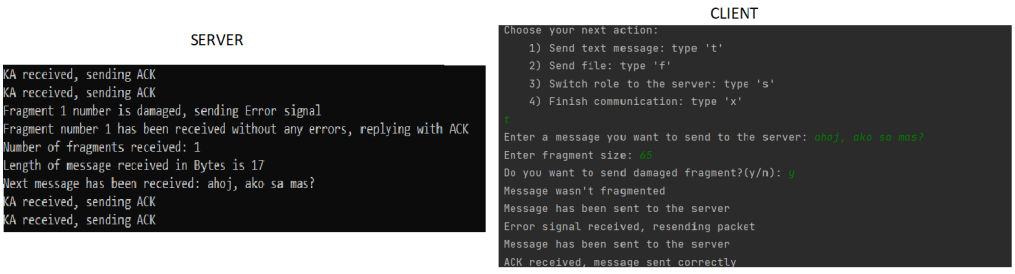


Obr. 9: Priklad spustenia programu a rozhrania dvoch stran

1.2.7 Odoslanie správy s poškodeným fragmentom

Funguje to takto: po výbere typu správy, ktorú chce klient odoslať, a dĺžky fragmentu sa mu dá na výber, či chce odoslať poškodený fragment alebo nie, ak klient vyberie možnosť áno, nastaví sa premenná boolean na hodnotu true a spolu s ďalšími argumentmi sa odovzdá funkcii na odosielanie správ cln_SP, v samotnej funkcii pri odosielaní fragmentov s pravdepodobnosťou 40 percent vo funkcii, ktorá zhromažďuje hlavičku a údaje v jednom celku (apply_protocol) pre tri(maximalne, ale može ich byť aj menej) z fragmentov by mala byť odoslaná značka, že je potrebné premiešať obsah správy a už vo funkcii apply_protocol budú údaje premiešané(údaje sa zamenia tak, že druhá polovica nahradí prvú polovicu a prvá polovica nahradí druhú polovicu), takže sa vytvorí poškodený balík.

Výberom možnosti áno alebo nie (y/n) môžete rozhodnúť, či sa má poškodený fragment odoslať okamžite po zadaní dĺžky fragmentu.



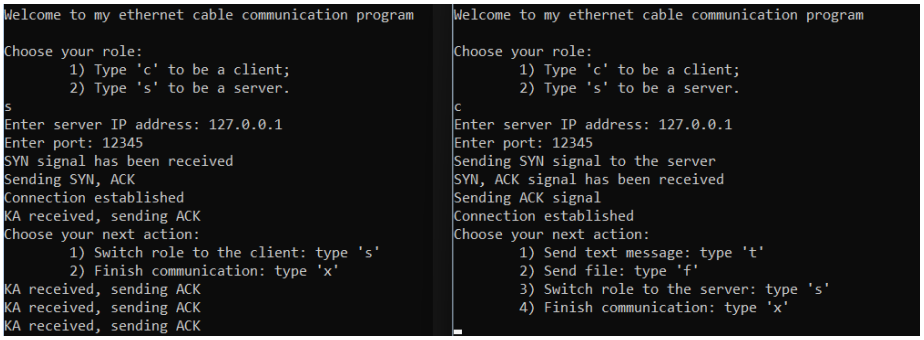
Obr. 10: Správy vypísané do konzol klienta a servera pri odoslaní poškodeného paketu.

2 Štruktúra programu

Táto časť opisuje vlastnosti programu, používanie vlákien a opisuje algoritmy a funkcie, ktoré neboli opísané v navrhe.

2.1 Spustenie programu

Po spustení súboru main.py si musíte vybrať svoju rolu, či ste klient (c) alebo server (s). Potom musíte zadať IP adresu servera a port, na ktorom bude prebiehať komunikácia, potom prebehne vyššie opísaný postup otvorenia komunikácie (začína odoslaním požiadavky klientom na server) a po otvorení komunikácie môžete vybrať, čo sa bude diať ďalej (na obrázku nižšie vidíte rozhranie klienta a servera).



Obr. 11: Priklad spustenia programu a rozhrania dvoch stran

2.2 Vlákna

Hneď ako sa začne komunikácia, spustí sa aj vlákno, ktoré je zodpovedné za udržiavanie spojenia, klient ho uspí, keď chce niečo poslať (správu, súbor alebo požiadavku na zmenu role), a server, keď si chce vymeniť roly s klientom.

Ak chcete zistiť, či vlakna fungujú na oboch stranách, pozrite sa na konzolu spustenú ako server, keď prijme signál KA a odošle odpoveď ACK, vypíše správu. Klient zasa nevypíše do konzoly správu, že KA bola odoslaná alebo ACK bola prijatá, pretože to klientovi bráni v zadaní správy, ak sa ju rozhodne odoslať.

Vlákno servera je serializované tak, že prijíma všetky správy v slučke a potom koná podľa príznaku správy. Ak napríklad vlákno dostane signál o ukončení komunikácie, spustí funkciu handle_fin a celý program sa ukončí, keď sa funkcia handle_fin skončí. Zároveň, ak server začne prijímať správy alebo súbory, vlákno ich rozlíši príznakom a začne ich zapisovať do príslušných premenných; keď opäť prijme CA, oznámi, že prijímanie údajov sa skončilo a všetko, čo vlákno počas prijímania zapísalo, sa uloží do súboru a/alebo zapíše do konzoly a potom pokračuje v účasti na algoritme údržby spojenia.

Taktiež, ak sa klient alebo server rozhodne spustiť akciu zadaním určitého písmena na príkazovom riadku - v hlavnom hlavnom vlákne sa zavolá funkcia, vlákno pre KA sa uspí, a keď funkcia skončí, buď sa opäť prebudí, alebo sa

program ukončí (v závislosti od funkcie, ktorá bola zavolaná, takže funkcia posielania správ na strane klienta opäť prebudí vlákno a funkcia zmeny role, ak je pozitívna, ukončí klientsky program a spustí serverový program na tom istom sokete, ale s novými vláknami)

3 Príklady práce v rámci programu

3.1 Príklady zmeny úlohy

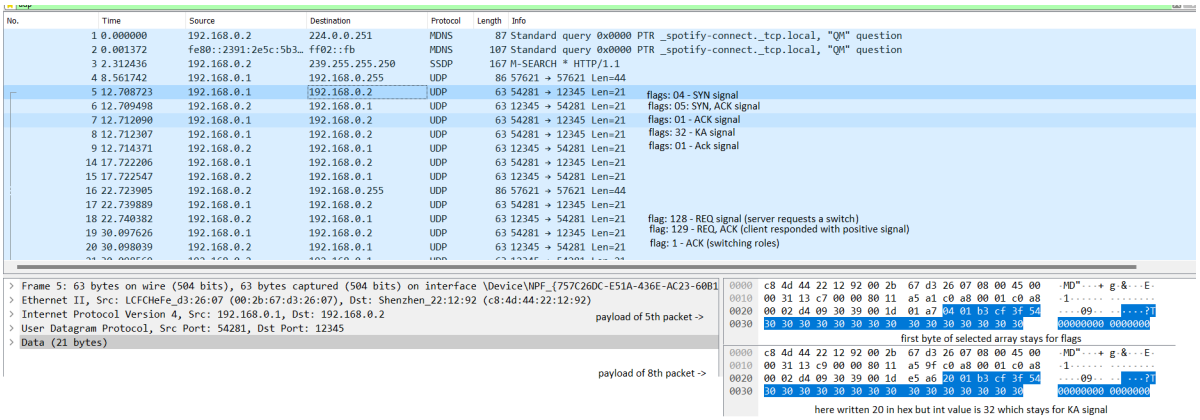
```
Welcome to my ethernet cable communication program

Choose your role:
    1) Type 'c' to be a client;
    2) Type 's' to be a server.
c
Enter server IP address: 192.168.0.2
Sending SYN signal to the server
SYN, ACK signal has been received
Sending ACK signal
Connection established
Choose your next action:
    1) Send text message: type 't'
    2) Send file: type 'f'
    3) Switch role to the server: type 's'
    4) Finish communication: type 'x'
REQ, ACK signal has been sent
ACK signal has been received, switching role
Press enter to switch role

Start working as server
SYN signal has been received
Sending SYN, ACK
Connection established
Choose your next action:
    1) Switch role to the client: type 's'
    2) Finish communication: type 'x'
```

Obr. 12: Príklad spustenia programu a spracovania pozitívneho signálu zmeny úlohy

```
Choose your next action:
    1) Send text message: type 't'
    2) Send file: type 'f'
    3) Switch role to the server: type 's'
    4) Finish communication: type 'x'
s
Keep-Alive thread is sleeping
REQ signal has been sent
Didn't receive any response
REQ signal has been sent
Didn't receive any response
REQ signal has been sent
REQ, ACK signal has been received, sending ACK signal and switching role
Start working as server
SYN signal has been received
Sending SYN, ACK
Connection established
Choose your next action:
    1) Switch role to the client: type 's'
    2) Finish communication: type 'x'
KA received, sending ACK
KA received, sending ACK
KA received, sending ACK
Fragment 1 number is damaged, sending Error signal
Fragment number 1 has been received without any errors, replying with ACK
Fragment number 2 has been received without any errors, replying with ACK
```

Obr. 13: Príklad zmeny roly iniciovanej serverom

Zachytené balíky od Weirshark a komentáre k nim. Popisuje príznaky paketov, v ktorých je zaznamenané otvorenie komunikácie, ako aj zmena úlohy.



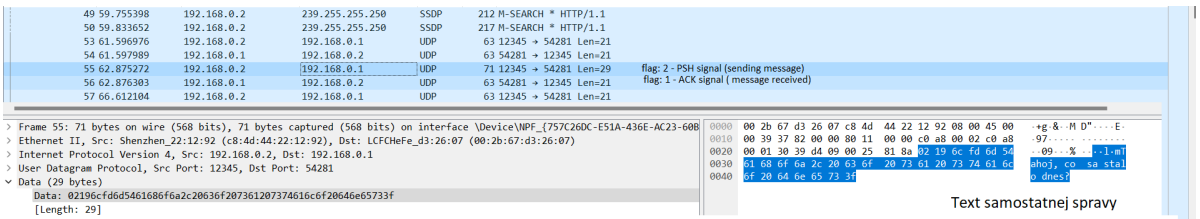
Obr. 14: WireShark: Otvorenie komunikacie a vymena role

3.2 Príklady odosielania a prijímania údajov

Obrázok ukazuje, čo sa vypíše v konzole po prijatí textovej správy.

```
KA received, sending ACK
Fragment number 1 has been received without any errors, replying with ACK
Number of fragments received: 1
Length of message received in Bytes is 23
Next message has been received: ahoj, co sa stalo dnes?
KA received, sending ACK
KA received, sending ACK
KA received, sending ACK
```

Obr. 15: Príklad prijímania údajov bez strat



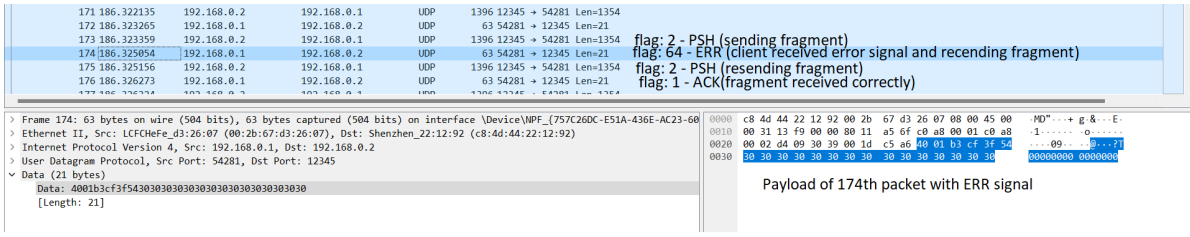
Obr. 16: WireShark: Posielanie spravy

Nasledujúci obrázok ukazuje, ako server pracuje, ak dostane poškodený fragment

```
Connection established
Choose your next action:
    1) Switch role to the client: type 's'
    2) Finish communication: type 'x'
KA received, sending ACK
KA received, sending ACK
KA received, sending ACK
Fragment 1 number is damaged, sending Error signal
Fragment number 1 has been received without any errors, replying with ACK
Fragment number 2 has been received without any errors, replying with ACK
Fragment number 3 has been received without any errors, replying with ACK
```

Obr. 17: Príklad prijímania údajov zo stratmi

Na obrázku vyššie môžete vidieť, že paket bol prijatý poškodený (potvrdené na obrázku nižšie), takže server poslal chybovú správu a prijal rovnaký fragment bez chýb.



Obr. 18: WireShark: Odoslanie poškodeného fragmentu