Zadanie 2
Komunikácia s využitím UDP protokolu

Laura Fulajtárová

Fakulta informatiky a informačných technológií STU

xfulajtarova@stuba.sk

ID: 120782

PKS – Utorok 18:00

2.12.2023

Obsah

Zadanie

https://github.com/fiit-ba/pks-course/tree/main/202324/assignments/2_communication_over_udp

Implementačné prostredie

Python som zvolila pre jeho jednoduchosť, čitateľný kód a rozsiahly ekosystém knižníc. Tieto vlastnosti zabezpečujú rýchly a efektívny vývoj, čo je ideálne pre implementáciu rôznych algoritmov. Python taktiež minimalizuje komplikácie spojené s programovaním a poskytuje univerzálnosť a prenositeľnosť.

Používateľské rozhranie

Užívateľské rozhranie som implementovala prostredníctvom menu v termináli, kde sú všetky nevyhnutné údaje prehľadne vypísané, čo zabezpečuje jednoduchú a intuitívnu interakciu s programom. Celý priebeh je podrobne zdokumentovaný v konzole, či už ide o aktiváciu servera, vykonávanie handshakov, prepínanie režimov, distribúciu správ alebo ukončenie komunikácie s návratom do hlavného menu.

Prehľad súborov

V rámci môjho programu som vytvorila niekoľko základných súborov, ktoré zabezpečujú jeho funkčnosť:

main.py

Tento súbor obsahuje celkovú logiku programu. Zahŕňa užívateľské rozhranie, vytváranie socketov pre klienta alebo server, spúšťanie procesov odosielania a prijímania správ alebo súborov, implementáciu keepalive, možnosť menenia rolí medzi klientom a serverom až po zatvorenie programu.

header.py

V tomto súbore sú definované funkcie pre jednoduché vytváranie a dekódovanie hlavičiek paketov, ako aj výpočet CRC hodnôt, čo je nevyhnutné pre overenie integrity dát.

keepalive.py

Tu je definovaná trieda pre keepalive, ktorá beží na samostatnom vlákne a pravidelne každých 5 sekúnd odosiela keepalive správy. Tento súbor tiež zabezpečuje zapínanie a vypínanie keepalive funkcie podľa potreby a poskytuje informácie o stave servera (či je pripojený, alebo či došlo k jeho pádu).

receiving.py

Tento súbor je kľúčový pre server, keďže obsahuje funkcie na prijímanie správ alebo súborov. Server reaguje na jednotlivé fragmenty potvrdzovacími správami (ack) a po prijatí celého obsahu zobrazuje relevantné informácie a ukladá súbory.

sending.py

V tomto súbore je implementovaná metóda stop and wait ARQ , kde postupne odosielame všetky potrebné fragmenty pre správu alebo súbor.

Spracovávanie komunikácie

Po zapnutí programu má používateľ možnosť vybrať si zo troch možností v **hlavnom menu**:

- 1 – Pokračovať ako server
- 2 – Pokračovať ako klient
- 3 – Ukončiť program

SERVER

Keď si používateľ vyberie možnosť 1, zadá port a IP adresu, na ktorých chce komunikovať v úlohe servera. Po zadaní týchto údajov sa spustí funkcia servera, ktorá funguje ako poslucháč. Server neustále prijíma dáta a na základe flagu v paketoch na ne odpovedá.

Ak server neprijme žiadne pakety do 30 sekúnd, automaticky sa vypne. Počas tejto doby, ak server neprijíma dáta, každých 10 sekúnd sa vypisuje upozornenie o tom, že neprichádzajú žiadne pakety.

Spojenie medzi klientom a serverom sa nadväzuje prostredníctvom 2-way handshake mechanizmu (SYN-ACK), po ktorom môže klient začať odosielať správy alebo súbory. Ak server zachytí keepalive správu, odpovie na ňu potvrdením (ack). Ak zachytí správu o ukončení spojenia (FIN), odpovie na ňu tiež potvrdením (FIN-ACK), zatvorí svoj socket a vráti sa do hlavného menu.

Pri požiadavke na výmenu rolí (switch) server pošle potvrdenie (SWITCH-ACK) a obe strany si vymenia svoje úlohy. Pri prijímaní správy alebo súboru server súbor uloží a zvolí file_path. Po prijatí správy alebo súboru má server možnosť v **server menu** pokračovať v počúvaní, vymeniť si úlohy s klientom alebo ukončiť komunikáciu. Tieto akcie vykonáva odoslaním príslušného paketu a spracovaním prijatej správy.

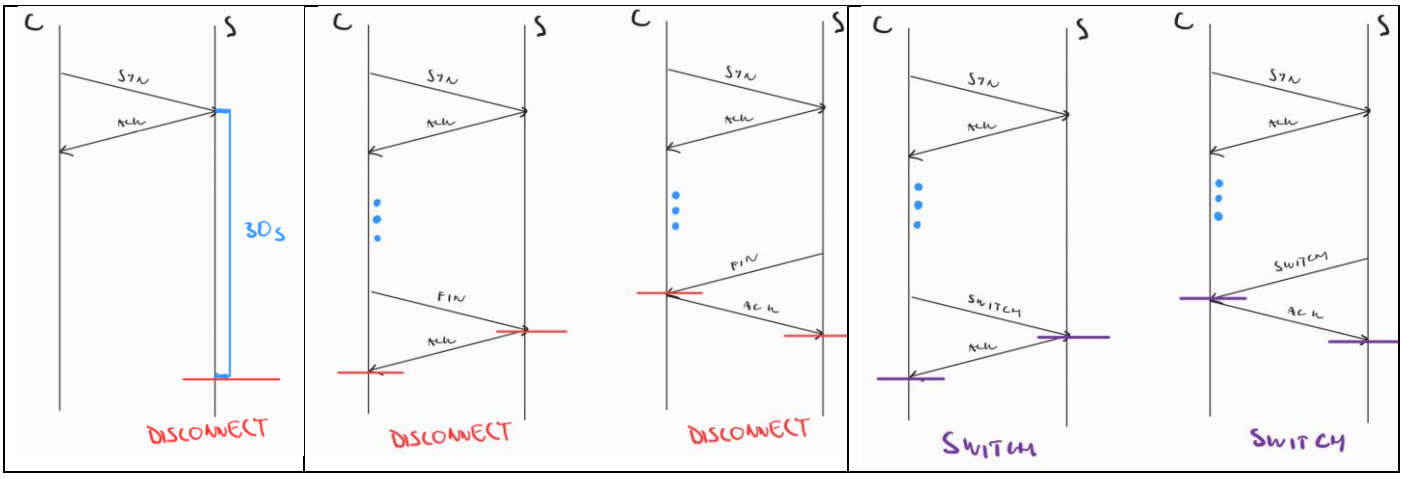
- 1 – Pokračovať v počúvaním
- 2 – Switchnúť mod s klientom
- 3 – Ukončiť spojenie a vrátiť sa do hlavného menu

KLIENT

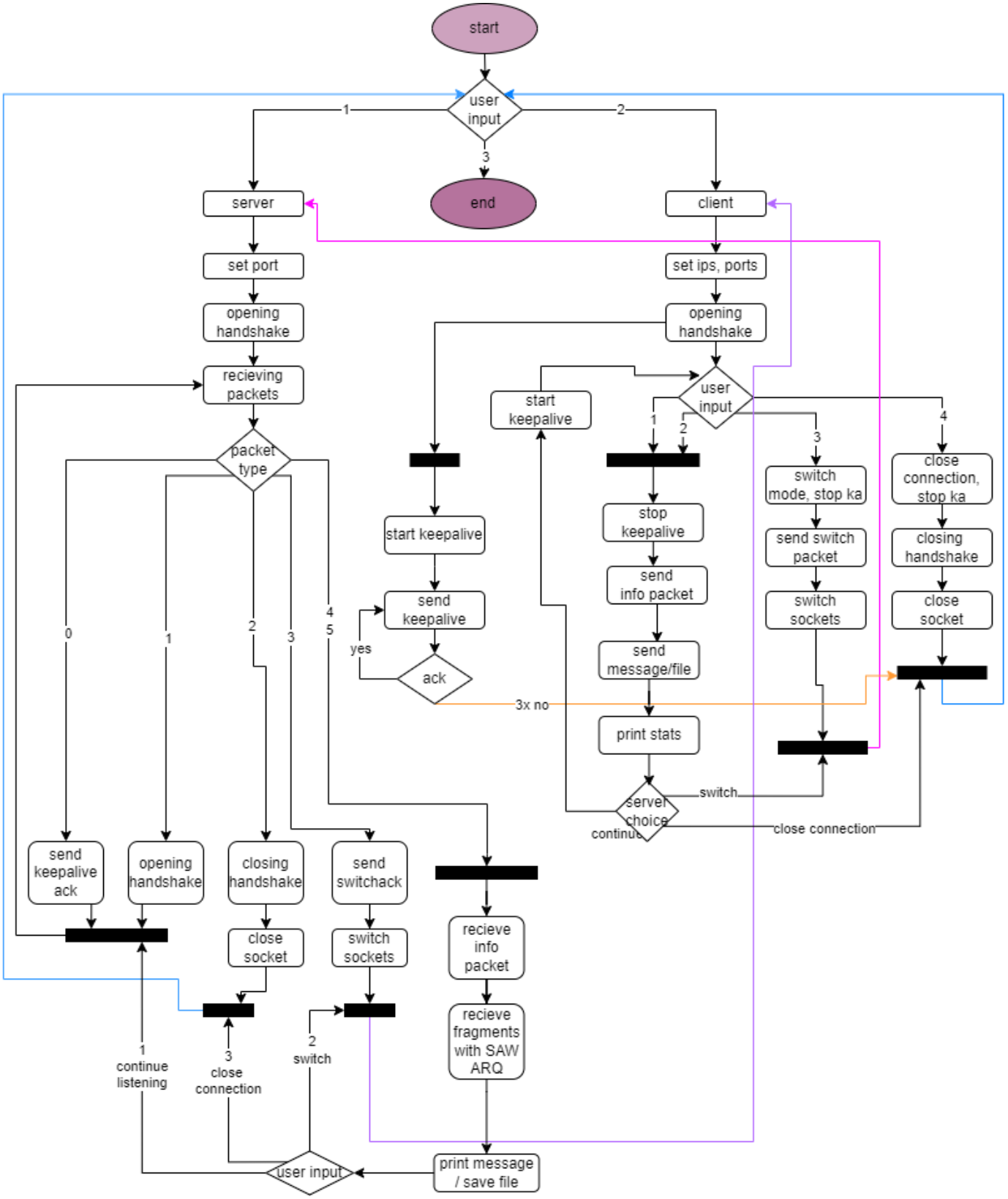
Klient najskôr nadviaže spojenie pomocou handshaku a po úspešnom nadviazaní je mu zobrazené **klient menu**, kde môže vybrať z 4 možností.

- 1 – Poslať správu
- 2 – Poslať súbor
- 3 – Switchnúť mod so serverom
- 4 – Ukončiť spojenie so serverom a vrátiť sa do hlavného menu

Pri posielaní správ zadá používateľ text správy a veľkosť jedného fragmentu. Pri posielaní súborov zadá používateľ názov, absolútnu cestu a veľkosť fragmentu. Používateľovi sa zobrazí možnosť simulácie chýb, pričom zadá indexy pakiet, ktoré budú chybné. Týmto spôsobom server identifikuje chyby. Po odoslaní a prijatí všetkých fragmentov je poslaný finalný paket a klient čaká na odpoveď či sa chce server prepnúť, ukončit spojenie alebo pokračovať v počúvaní. Podľa volby servera sa tak aj zachová a v prípade pokračovania sa klient vrráti do klient menu. Switchovanie modu znamená odoslanie správy, čakanie na ack a prechod do funkcie servera. Ukončenie spojenia odoslaním správ cez handshake uzavrie oba sockets a klient sa vráti do hlavného menu.

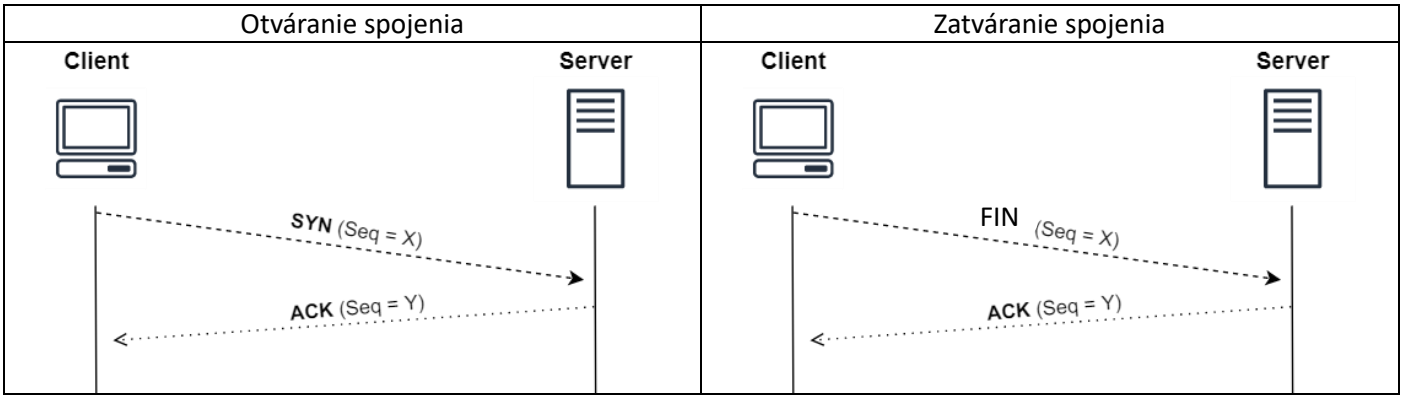


Activity diagram spracovávania komunikácie



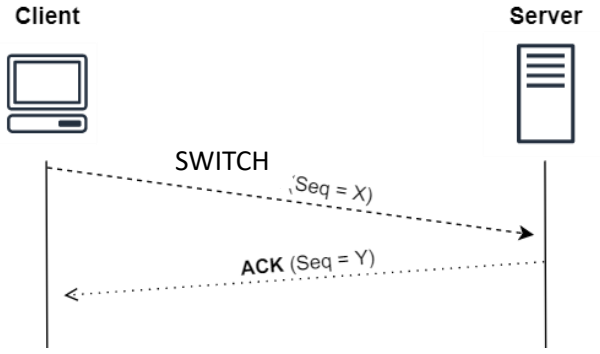
Otváranie a zatváranie komunikácie

Pri navrhovaní komunikácie som zvolila model 2-way handshake. V tomto modeli jedna strana, či už server alebo klient, pošle druhej strane synchronizačnú správu (SYN) a potom čaká na potvrdenie tejto správy (SYN-ACK). Keď je tento proces úspešne dokončený, klient je spojený so serverom a môže začať posielať správy alebo súbory. Ukončenie komunikácie môže iniciovať buď server alebo klient. Klient má možnosť ukončiť komunikáciu prostredníctvom menu klienta, a server má možnosť ju ukončiť po prijatí správy alebo súboru. Stačí, keď jedna strana pošle druhej ukončovaciu správu (FIN) a následne čaká na jej potvrdenie (FIN-ACK). Po úspešnom ukončení komunikácie obe strany zatvoria svoje sokety a vrátia sa späť do hlavného menu.



Switchovanie

Server aj klient majú schopnosť flexibilne meniť svoje role. Tento proces sa spúšťa tým, že jedna strana pošle požiadavku na zmenu role, na ktorú druhá strana reaguje potvrdzovacou správou (SWITCH-ACK). Keď sa obe strany dohodnú na zmene, ukončia svoje aktuálne úlohy a prepínajú sa na opačnú úlohu. Klient sa môže prepínať v menu klienta, zatiaľ čo server má možnosť meniť svoju rolu po prijatí správy alebo súboru. Počas tohto procesu zmeny rolí si obe strany zachovávajú svoje pôvodné IP adresy a porty. Jediná zmena, ktorá nastáva, je ich funkcia - zo servera sa stáva klient a z klienta server.



Štruktúra hlavičky

Používam jednu univerzálnu hlavičku pre všetky typy paketov, čo zjednodušuje spracovanie a zabezpečuje jednotnosť. Táto hlavička obsahuje niekoľko kľúčových informácií:

- Flag správy: Informuje o typu paketu, či už ide o správu, file, potvrdenie, handshaky alebo výmenu rolí.
- CRC : Slúži na kontrolu integrity dát, aby sa overilo, či boli všetky dáta správne doručené a neobsahujú žiadne chyby.
- Fragment size, count a sequence: Tieto údaje definujú maximálnu veľkosť každého fragmentu dát, celkový počet fragmentov a poradie aktuálneho fragmentu v rámci prenosu.

Z dôvodu zabezpečenia prenosu súborov väčších ako 2 MB som sa rozhodla zväčšiť veľkosť polí pre fragment size a sequence na 4 bajty. Toto umožňuje menšie veľkosti jednotlivých fragmentov a zároveň zachováva schopnosť spracovať veľké súbory. Samotné dáta na prenos nasledujú po hlavičke. Celková veľkosť kompletnej hlavičky je 16 bajtov, čo zabezpečuje, že každý paket obsahuje všetky potrebné informácie pre jeho správne spracovanie a doručenie. Maximálnu veľkosť dát som určila nasledovne:

Payload Size = total packet size – IP header size − UDP header size – my header size

Payload Size = 1500 – 20 – 8 – 16 = **1454 B**

flag	CRC	frag. size	frag. count	frag. seq.	data...
2 B	4 B	2 B	4 B	4 B	

Flag

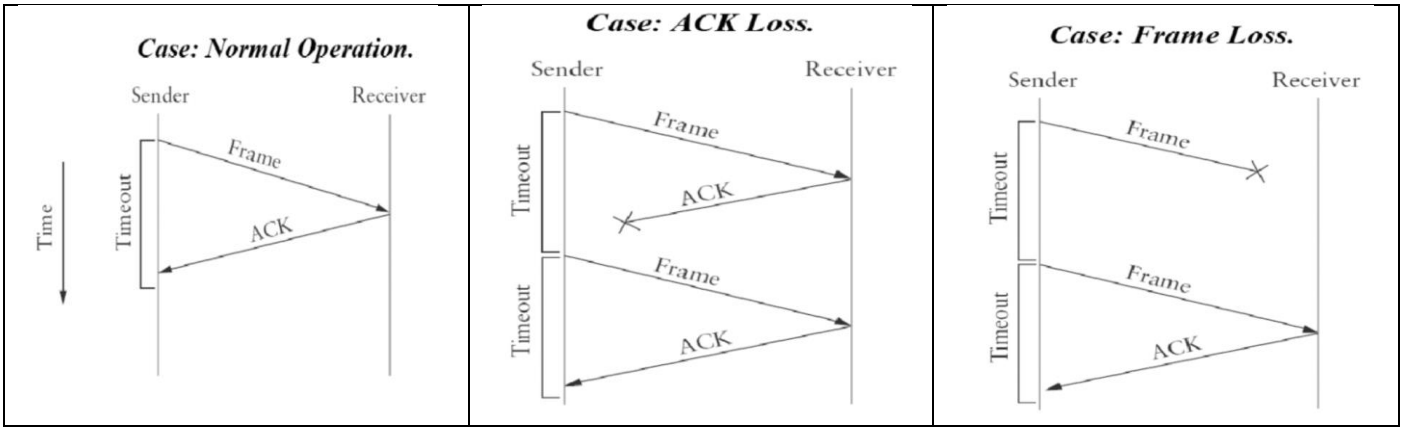
- 0 – keepalive
- 1 – opening
- 2 – closing
- 3 – switching modes
- 4 – text message
- 5 – file
- 6 – ACK
- 7 – NACK

Simulácia chyby pri prenose súboru a správy a použitá ARQ metóda

Implementovala som metódu STOP AND WAIT. Táto metóda funguje na princípe, že klient pošle fragment dát serveru a čaká na potvrdenie (acknowledgement) od servera pred odoslaním ďalšieho fragmentu. Ak server neodpovie - či už kvôli chybe v dátach, kde CRC hodnoty nesedia, alebo ak sa paket stratí - klient po uplynutí timeoutu (v mojom prípade nastavenom na 2 sekundy) odosiela tento istý fragment znova a znova čaká na potvrdenie od servera. Po odoslaní všetkých fragmentov klient pošle samostatný paket, ktorý signalizuje úplné odoslanie všetkých fragmentov.

Na strane klienta som tiež implementovala simuláciu chyby, kde si užívateľ môže vybrať, či chce simulovať chybu zmenou dátovej časti alebo neodoslaním fragmentu. Ak si užívateľ zvolí túto možnosť, zadá poradové čísla fragmentov, pre ktoré chce túto chybu simulovať. Ak server prijme paket s chybným CRC, neodpovie naň. Klient potom skúša odoslať tento chybný fragment najviac 10-krát, kým neukončí odosielanie, keďže server neodpovedá. Na serverovej strane, ak neprijme žiadne dáta do 20 sekúnd, automaticky detekuje problém a ukončuje prijímanie správy alebo fragmentu.

Po doručení všetkých fragmentov súboru, má užívateľ možnosť vybrať umiestnenie uloženia súboru. Následne sa na obrazovke zobrazia všetky relevantné informácie, ako napríklad správa, názov súboru, cesta k súboru, dĺžka dát, počet fragmentov, počet správne prijatých fragmentov a ďalšie údaje.

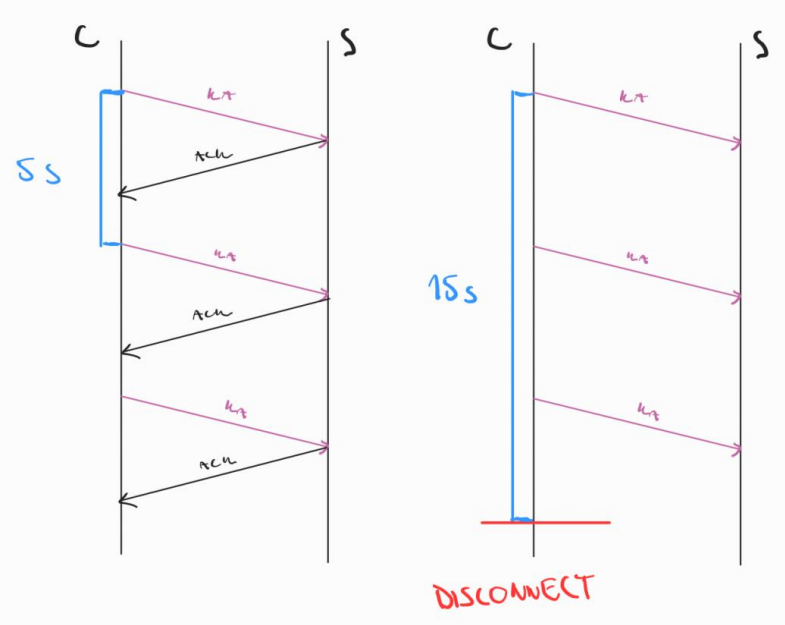


Udržanie spojenia – Keepalive (KA)

V procese udržiavania spojenia medzi klientom a serverom používam keepalive (KA) metódu, ktorá zahŕňa pravidelné odosielanie KA správ od klienta k serveru v intervale 5s. Ak klient neobdrží potvrdenie (ack) na svoju KA správu, interpretuje to ako možné prerušenie spojenia.

V takom prípade klient opakovane odosiela KA správu, pričom maximálny počet pokusov je obmedzený na tri vrátane pôvodného pokusu. Informácie o každom pokuse sa zobrazujú na obrazovke klienta, čo umožňuje užívateľovi sledovať proces a byť informovaný o aktuálnom stave spojenia.

Ak klient po troch pokusoch stále neobdrží ack, automaticky sa odpojí, predpokladajúc, že spojenie bolo prerušené. Pri posielaní súborov alebo správ zastavujem keepalive funkciu.



CRC

Implementovaná CRC (Cyclic Redundancy Check) metóda využíva knižnicu zlib na jednoduchý a efektívny výpočet 32-bitovej kontrolnej hodnoty. Táto metóda, charakterizovaná jednoduchosťou implementácie a rýchlosťou výpočtu, slúži na spoľahlivú detekciu chýb v prenášaných dátach v komunikačných protokoloch a súboroch. Zvolenie CRC-32 pred CRC-16 zabezpečuje väčšiu presnosť pri identifikácii rôznych typov chýb.

Testovacie scénare

Všetku Wireshark testovania nájdete tu: https://github.com/fulajtarova/pks_wireshark_files

1. Otváranie komunikácie, keepalive, zatvorenie zo strany klienta

```
Choose your role or exit:
1 for server
2 for client
3 to exit
Enter choice: 1
Enter server ip: 10.10.16.83
Enter server port: 5001

SERVER MODE
Server address: ('10.10.16.83', 5001)

Received SYN
Sending SYN-ACK...

Connection established with ('10.10.16.83', 5002)

Received keepalive
Received keepalive
Received FIN
Sending FIN-ACK...

Connection from client closed


Choose your role or exit:
1 for server
2 for client
3 to exit
Enter choice:
```

```
Choose your role or exit:
1 for server
2 for client
3 to exit
Enter choice: 2
Enter server ip: 10.10.16.83
Enter server port: 5001
Enter client ip: 10.10.16.83
Enter client port: 5002

CLIENT MODE
Client address: ('10.10.16.83', 5002)
Server address: ('10.10.16.83', 5001)

Sending SYN...
Rceived SYN-ACK

Connection established

Client Menu:
1 to send message
2 to send file
3 to switch mode
4 to close connection and exit
Enter choice: 4
Sending FIN...
Received FIN-ACK

Connection closed


Choose your role or exit:
1 for server
2 for client
3 to exit
Enter choice:
```

2. Odosielanie správy

```
Recieving message...
Fragment 0 recieved with error, crc not matching, waiting for resend
Fragment 0 received correctly
Fragment 1 received correctly
Fragment 2 received correctly
Fragment 3 recieved with error, crc not matching, waiting for resend
Fragment 3 received correctly
Fragment 4 received correctly
Fragment 5 received correctly
Fragment 6 received correctly
Client finished sending data

Successfully received all fragments.

Message: ahoj ako sa mas ja celkom odbre

Message size: 31
Number of fragments: 7
Fragment size: 5
```

```
Client Menu:
1 to send message
2 to send file
3 to switch mode
4 to close connection and exit
Enter choice: 1
Enter message: ahoj ako sa mas ja celkom odbre
Data size: 31
Enter fragment size: 5
Number of fragments: 7
Do you want to simulate data error? (y/n): y
Enter fragments indexes with error separated by space from 0 to 6: 0 3
Do you want to simulate lost packet? (y/n): y
Enter fragments indexes with error separated by space from 0 to 6: 6
Fragment order: 0 of 6 (sent with error)
Fragment not received correctly, trying again
Fragment received correctly
Fragment order: 1 of 6 (sent correctly))
Fragment received correctly
Fragment order: 2 of 6 (sent correctly))
Fragment received correctly
Fragment order: 3 of 6 (sent with error)
Fragment not received correctly, trying again
Fragment received correctly
Fragment order: 4 of 6 (sent correctly))
Fragment received correctly
Fragment order: 5 of 6 (sent correctly))
Fragment received correctly
Fragment order: 6 of 6 (not sent)
Fragment not received correctly, trying again
Fragment received correctly

Whole message/file was successfully send and received

Message: ahoj ako sa mas ja celkom odbre

Message size: 31
Number of fragments: 7
Fragment size: 5
```

3. Odosielanie súboru

```
Recieving file...
Fragment 0 received correctly
Fragment 1 received correctly
Fragment 2 recieved with error, crc not matching, waiting for
resend
Fragment 2 received correctly
Fragment 3 received correctly
Fragment 4 received correctly
Client finished sending data

Successfully received all fragments.
Enter path to save file [  C:\Users\Laura\Documents\škola\3sem
ester\PKS\project2\recieved_files  ]:
C:\Users\Laura\Documents\škola\3semester\PKS\project2\recieved
_files

File name: small_photo.png
New file path: C:\Users\Laura\Documents\škola\3semester\PKS\pr
oject2\recieved_files\small_photo.png

File size: 414
Number of fragments: 5
Fragment size: 100
```

```
Enter choice: 2
Enter file name: small_photo.png
Enter file path: C:\Users\Laura\Documents\škola\3semester\PKS\project2\files_to_send\s
mall_photo.png
Data size: 414
Enter fragment size: 100
Number of fragments: 5
Do you want to simulate data error? (y/n): y
Enter fragments indexes with error separated by space from 0 to 4: 2
Do you want to simulate lost packet? (y/n): y
Enter fragments indexes with error separated by space from 0 to 4: 3
Fragment order: 0 of 4 (sent correctly))
Fragment received correctly
Fragment order: 1 of 4 (sent correctly))
Fragment received correctly
Fragment order: 2 of 4 (sent with error)
Fragment not received correctly, trying again
Fragment received correctly
Fragment order: 3 of 4 (not sent)
Fragment not received correctly, trying again
Fragment received correctly
Fragment order: 4 of 4 (sent correctly))
Fragment received correctly

Whole message/file was successfully send and received

File name: small_photo.png
File path: C:\Users\Laura\Documents\škola\3semester\PKS\project2\files_to_send\small_p
hoto.png

File size: 414
Number of fragments: 5
Fragment size: 100
```

4. Ukončenie komunikácie zo strany servera

```
1 continue listening for data
2 switch mode
3 close connection
Enter choice: 3

Sending FIN...
Received FIN-ACK

Connection closed


Choose your role or exit:
1 for server
2 for client
3 to exit
Enter choice:
```

```
Waiting if server wants to switch mode, close connection or continue...
Received FIN
Sending FIN-ACK...

Connection closed


Choose your role or exit:
1 for server
2 for client
3 to exit
Enter choice:
```

5. Switchnutie zo strany klienta

```
SERVER MODE
Server address: ('10.10.16.83', 5001)

Received SYN
Sending SYN-ACK...

Connection established with ('10.10.16.83', 5002)

Received keepalive
Received keepalive
Received keepalive
Received SWITCH
Sending SWITCHACK...

Switching to client mode...


CLIENT MODE
Client address: ('10.10.16.83', 5001)
Server address: ('10.10.16.83', 5002)

Sending SYN...
Rceived SYN-ACK

Connection established

Client Menu:
1 to send message
2 to send file
3 to switch mode
4 to close connection and exit
Enter choice:
```

```
CLIENT MODE
Client address: ('10.10.16.83', 5002)
Server address: ('10.10.16.83', 5001)

Sending SYN...
Rceived SYN-ACK

Connection established

Client Menu:
1 to send message
2 to send file
3 to switch mode
4 to close connection and exit
Enter choice: 3
Sending SWITCH...
Received SWITCH-ACK

Switching to server mode...


SERVER MODE
Server address: ('10.10.16.83', 5002)

Received SYN
Sending SYN-ACK...

Connection established with ('10.10.16.83', 5001)

Received keepalive
Received keepalive
```

6. Switchnutie zo strany servera

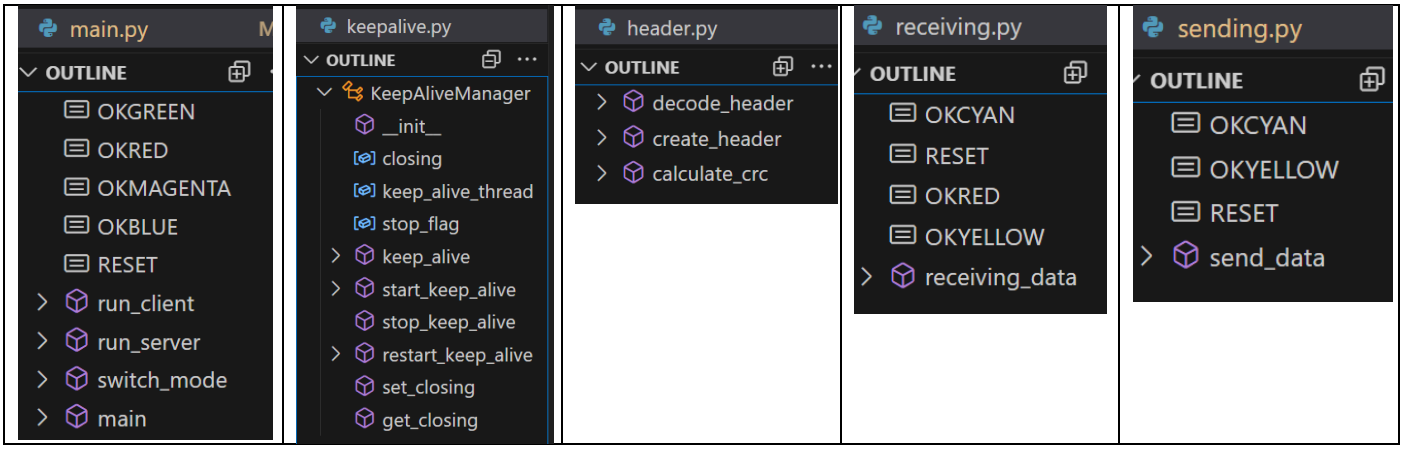
```
1 continue listening for data
2 switch mode
3 close connection
Enter choice: 2

Sending SWITCH...
Received SWITCH-ACK

Switching to client mode...


CLIENT MODE
Client address: ('10.10.16.83', 5002)
Server address: ('10.10.16.83', 5001)

Sending SYN...
Rceived SYN-ACK

Connection established

Client Menu:
1 to send message
2 to send file
3 to switch mode
4 to close connection and exit
Enter choice:
```

```
Waiting if server wants to switch mode, close connection or co
ntinue...
Received SWITCH
Sending SWITCH-ACK...

Switching to server mode...


SERVER MODE
Server address: ('10.10.16.83', 5001)

Received SYN
Sending SYN-ACK...

Connection established with ('10.10.16.83', 5002)

Received keepalive
```

Použité knižnice

```python
import socket
import time
import struct
import math
import threading
import os
import zlib
```

Použité classy a metódy



Zmeny oproti návrhu

V priebehu vývoja môjho projektu som urobila niekoľko zásadných zmien v dizajne a implementácii. Jednou z hlavných zmien je upravený spôsob, akým prebieha výmena rolí (switching) medzi serverom a klientom. Teraz, po každej prijatej správe alebo súbore, má server možnosť sa prepnúť na klienta, ukončiť spojenie alebo pokračovať v počúvaní. Táto flexibilita umožňuje užívateľom väčšiu kontrolu nad chodom programu.

Ďalšou zmenou je prechod z 3-way handshake mechanizmu na 2-way handshake. Toto rozhodnutie som urobila kvôli jednoduchšej a efektívnejšej implementácii, čo by malo výrazne zjednodušiť a zrýchliť proces nadväzovania spojení.

V hlavičke paketov som tiež urobila úpravy. Namiesto použitia samostatných polí pre typ a flag som sa rozhodla použiť len flag. Táto zmena by mala hlavičku urobiť kompaktnejšou a zjednodušiť spracovanie paketov. Okrem toho som zväčšila veľkosť polí pre sériové číslo fragmentu a počet fragmentov z 2 bajtov na 4 bajty. Táto úprava bola potrebná, aby sa zabezpečila lepšia podpora pri odosielaní väčších súborov, kde je potrebné viac priestoru na správne zaradenie a sledovanie jednotlivých fragmentov.

Nakoniec, zmenila som veľkosť payloadu z pôvodných 1024 bajtov na 1500 bajtov mínus veľkosť všetkých hlavičiek. Týmto sa zvyšuje efektívnosť prenosu dát, umožňujúc prenášať viac údajov v jednom pakete, čo je obzvlášť dôležité pri prenose väčších súborov. Tieto zmeny sú výsledkom môjho snaženia o vylepšenie a optimalizáciu programu, aby bol čo najviac efektívny a užívateľsky prívetivý.

Záver

V rámci tohto zadania sa mi podarilo úspešne naprogramovať komunikátor pracujúci na princípe klient-server, ktorý je založený na vlastnom protokole pracujúcom nad UDP. Tento program umožňuje efektívnu výmenu textových správ a súborov medzi dvoma uzlami v lokálnej sieti Ethernet. Implementácia zahŕňa vysielaciu a prijímaciu komponentu, ktoré spolupracujú na zabezpečení spoľahlivého prenosu dát, aj napriek prípadným stratám v sieti.

Pri realizácii tohto projektu som sa naučila pracovať so soketmi a implementovala som algoritmus stop-and-wait, čo umožňuje kontrolu chýb a znovuodoslanie poškodených fragmentov súborov. Táto funkcionalita je kľúčová pre zabezpečenie integrity dát pri prenose. Program tiež podporuje dynamickú zmenu maximálnej veľkosti fragmentu súboru používateľom, čo umožňuje optimalizáciu prenosu v závislosti od podmienok v sieti.

Zvláštny dôraz bol kladený na užívateľskú interakciu, umožňujúc používateľovi špecifikovať cieľovú IP adresu a port pre odosielanie súborov. Okrem toho program umožňuje obom komunikujúcim stranám zobrazovať dôležité informácie, ako sú názov a cesta k súboru, veľkosť a počet fragmentov, ako aj celková veľkosť prenesených správ a súborov. Tento prístup zvyšuje transparentnosť komunikácie a umožňuje lepšiu diagnostiku a riadenie prenosov.

Celkovo tento projekt predstavuje komplexné riešenie pre sieťovú komunikáciu s vysokou úrovňou spoľahlivosti a efektivity, čo je dôležité pre moderné komunikačné systémy. Prínosom tejto práce je aj možnosť simulácie chýb pri prenose, čo umožňuje lepšie pochopenie a analýzu správania sa siete v rôznych podmienkach. Tento projekt poskytuje pevný základ pre ďalšie rozšírenie a optimalizáciu sieťovej komunikácie v rôznych aplikáciách.

Použitá literatúra

1. https://blog.worldline.tech/2018/01/29/keepalive.html
2. https://www.geeksforgeeks.org/stop-and-wait-arq/
3. https://www.researchgate.net/figure/4-Stop-and-Wait-ARQ-Protocol-Timeline-for-Four-Different-Scenarios-31_fig17_236259629