Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Ilkovičova 2, 842 16 Bratislava 4

Počítačové a komunikačné siete

Dokumentácia

Zadanie 2 – UDP Komunikátor

Cvičenie: Štvrtok 18:00

Cvičiaci: Ing. Lukáš Mastiľak

Vypracoval: Dominik Klušák

Obsah:

2 Hlavička protokolu 1

3 Metóda kontrolnej sumy 2

4 ARQ metóda 2

5 Metóda na udržanie spojenia 3

6 Zmeny oproti návrhu 3

7 Aktivity diagram odosielateľa 4

8 Aktivity diagram prijímateľa 5

9 Zdokumentovanie testovacieho scenára 5

10 Popis používateľského rozhrania 8

11 Záver a zhodnotenie 9

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 zapnutí programu, komunikátor automaticky odosiela paket pre udržanie spojenia každých 5s pokiaľ používateľ neukončí spojenie ručne. Odporúčame riešiť cez vlastne definované signalizačné správy a samostatný thread.

2 Hlavička protokolu

Moja hlavička obsahuje dokopy **5B,** ako je vidieť ďalej aj na návrhu, **4B** sú potrebné kvôli checksum metóde ktorú používam a **1B** je na použité flagy. Nie je potrebné do hlavičky vkladať veľkosť fragmentu keďže si ho vie server spätne dopočítať. Taktiež **SQN** stačilo ošetriť jedným **1B** a to z dôvodu že ak využívame stop and wait, stačí použiť 2 hodnoty v mojom prípade 6 a 7, ak mi príde 6 viem že ďalší packet musí mať číslo 7 ak toto nespĺňa viem že mi prišiel zly packet alebo mohla nastať niekde chyba a vyžiadam si opäť číslo 7.

4B	**1B**
CHECKSUM	FLAGY
Využívaná CRC32 metóda vyžaduje 4B	*SYN, SYN ACK, NACK, FIN, FIN ACK, SQN1, SQN2, SQN1 ACK, SQN2 ACK, SWITCH, SWITCH ACK, FILE, FILE ACK*

SYN – používa sa na začiatok komunikácie.
SYN ACK – prijímateľ odpovedá týmto flagom že chce komunikovať.
NACK – V prípade že ma packet nesprávny SQN /checksum prijímateľ odošle tento flag
FIN – Posiela sa ak sa jedna o posledný packet zo strany odosielateľa.
FIN ACK – Posiela sa aj prijímateľ prijme posledný packet a je v poriadku.
SQN1 – Sekvenčne číslo pre packety.
SQN2 – Sekvenčne číslo pre packety.
SQN1 ACK – Potvrdenie na strane prijímateľa že je to ten packet ktorý mal doraziť.
SQN2 ACK – Potvrdenie na strane prijímateľa že je to ten packet ktorý mal doraziť.
SWITCH – Inicializuje výmenu úloh. (posiela odosielateľ)
SWITCH ACK – Odpovedá výmenu úloh že ju prijíma. (posiela prijímateľ)

FILE – V prípade súboru sa odošle tento flag aby sa preniesla aj cesta/názov súboru.
FILE ACK – Potvrdenie zo strany prijímateľa aby vedel že nasleduje súbor/cesta.

3 Metóda kontrolnej sumy

CRC (Cyklický redundantný súčet) je špeciálna hašhovacia funkcia využívaná na detekciu chýb behom prenosu alebo ukladaniu dát. Pre svoju jednoduchosť a dobré matematické vlastnosti ide o veľmi rozšírený spôsob realizácie kontrolného súčtu.

V našom projekte budeme používať CRC-32, používa teda algoritmus CRC32 na zisťovanie zmien medzi zdrojovými a cieľovými údajmi. Funkcia CRC32 konvertuje reťazec na 8-znakový reťazec, ktorý je textovou reprezentáciu hexadecimálnej hodnoty 32 a dokážeme ju teda reprezentovať ako 4 bity.

4 ARQ metóda

V našom projekte je implementovaná ARQ metóda ako **stop and wait**. Čo znamená že je poslaný packet zo strany odosielateľa a na čo mu prijímateľ odpovedá správou, áno prijal som tvoj packet, áno prijal som tvoj packet ale je nesprávny, alebo nedorazí odpoveď. V našom prípade sme ošetrili všetky tieto stavy, ak sme prijali správny packet všetko je v poriadku ideme na ďalší. Ak nám dorazil packet s nesprávnym SQN, v mojom programe používam iba 2 hodnoty pre SQN pretože ak máme metódu stop and wait vždy čakáme len na jeden packet súčasne a môžeme si to dovoliť tým že sa budú striedať hodnoty. Napríklad máme SQN1 a SQN2 ak sa pošle jednotka a prijímateľ prijal jednotku, vie že ďalší musí doraziť SQN2 aby bol teda rozdielný, mohlo sa stať že odpoveď nedorazila a tak ju odosielateľ poslal ešte raz ale teda už nebude akceptovaný a vypýta si predošlý ešte raz.

5 Metóda na udržanie spojenia

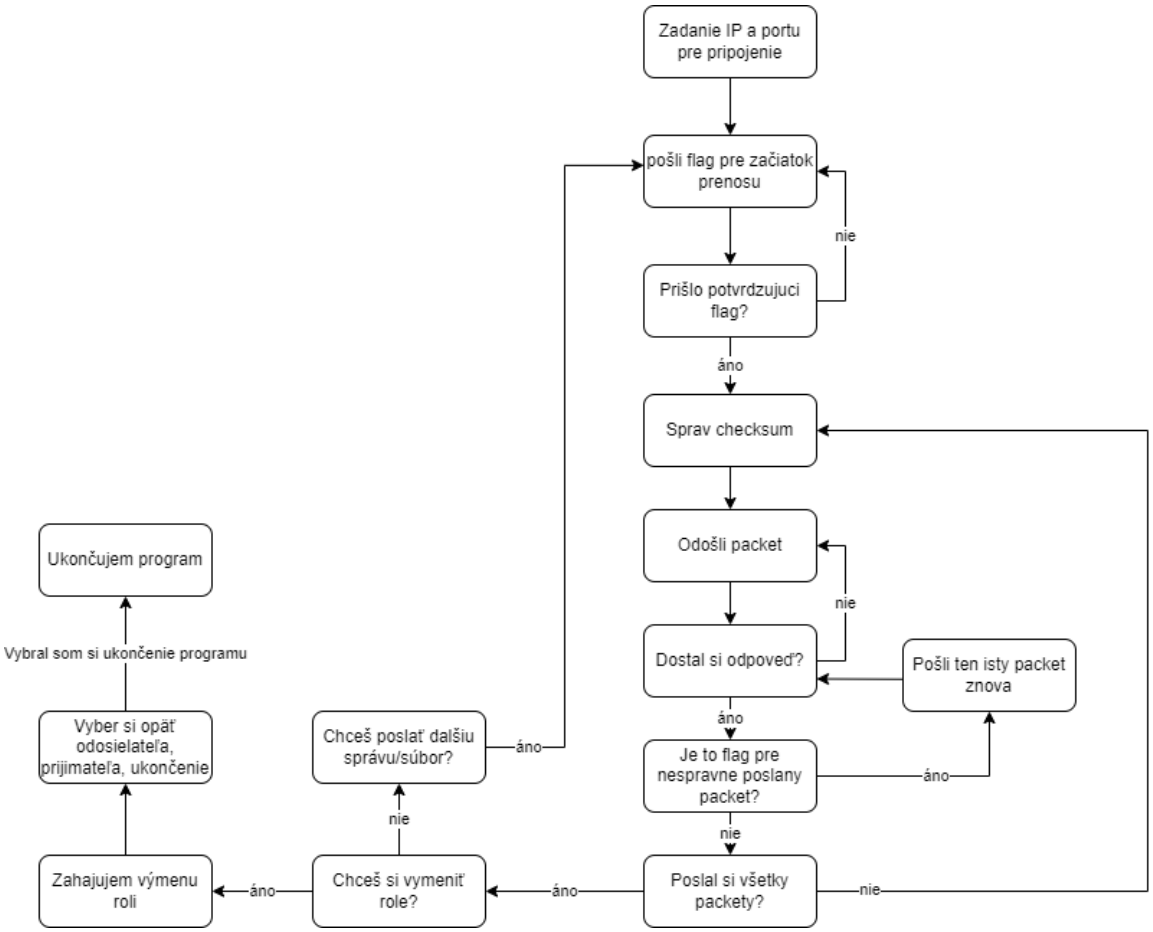
Keep alive metódu som sa rozhodol neimplementovať do svojho programu.

6 Zmeny oproti návrhu

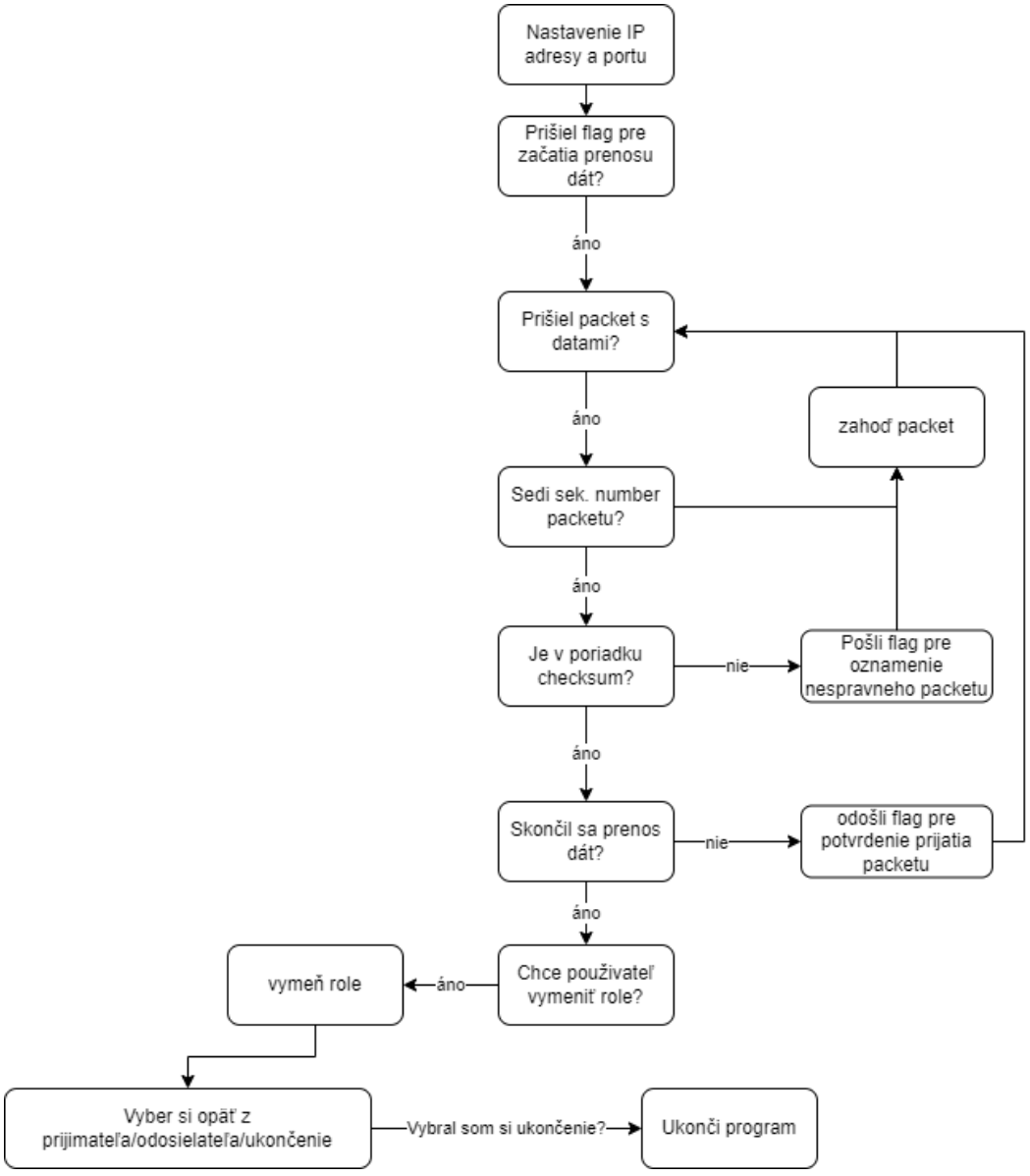
Zmeny oproti návrhu neboli v podstate žiadne, hneď po tom ako sme si uvedomili že prijímateľ si vie fragmentaciu dopočítať a taktiež na kontrolu sekvenčného čísla sa da použiť 1B sme sa vrátili k pôvodnej hlavičke.

Akurát v tomto prípade nastala zmena a to že som sa rozhodol neimplementovať metódu na udržanie spojenia, teda prijímateľ stále len počúva a nevymieňajú sa po skončení prenosu súboru/správy packety. Taktiež som napravil definovanie jednotlivých flagov a čo ktoré znamenajú, čo predtým v návrhu nebolo keďže som ešte programovú časť nemal hotovú a teda nebol som si ešte čo presne ako použijem.

7 Aktivity diagram odosielateľa



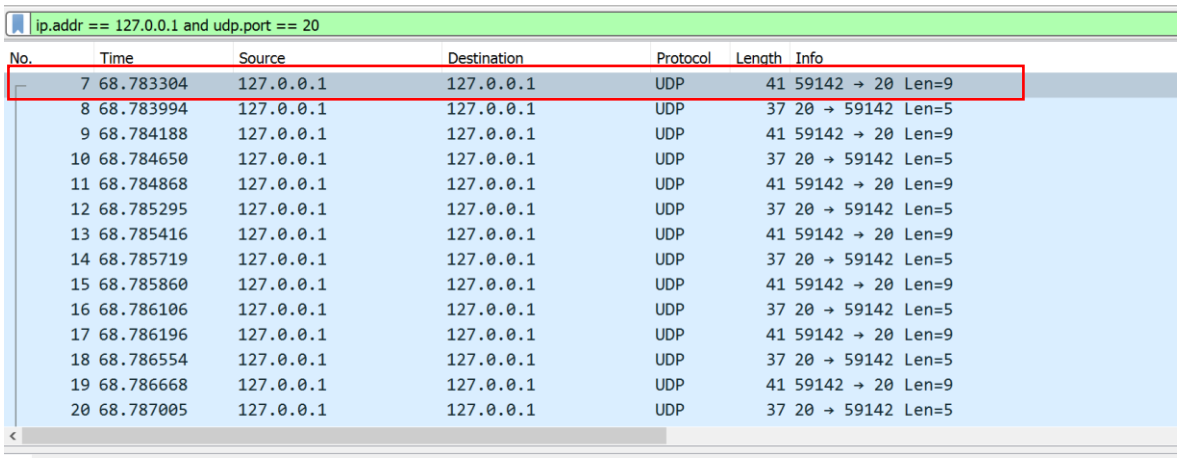
8 Aktivity diagram prijímateľa



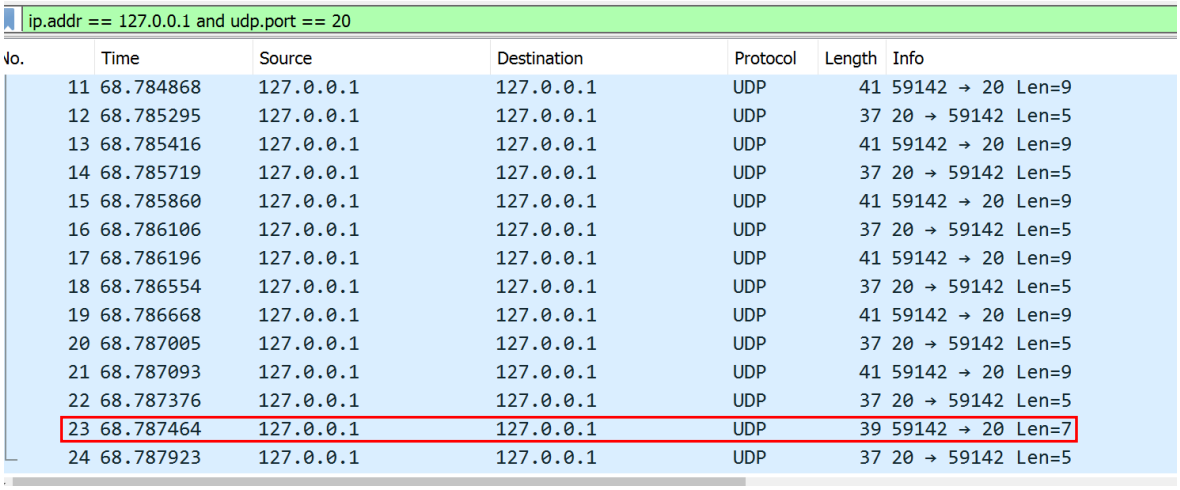
9 Zdokumentovanie testovacieho scenára

Pre testovací scenár som si zvolil odoslanie nejakej správy prijímateľovi, testovať to budeme na localhoste teda IP adresa 127.0.0.1 a port zvolíme 20. Fragmentáciu nastavíme na 4B.

Túto komunikáciu si vyfiltrujeme v programe Wireshark pomocou filtra ip.addr == 127.0.0.1 and udp.port == 20. Keď sa pozrieme na komunikáciu z wiresharku vidíme že sa ako prvý packet posiela z dlžkou 9B čo je správne pretože máme prvý flag SYN ale zároveň už pridávame aj dáta, teda hlavička 5B + 4B (fragmentácia) je 9B čo nám zobrazuje aj wireshark.



No keď už prejdeme na koniec komunikácie, cela správa mala dĺžku 34B, tým pádom nám na posledný fragment ostali 2B, preto ak sa pozrieme nižšie na poslednú správu čo posiela odosielateľ, môžeme vidieť že je dĺžky 7B -> hlavička 5B + 2B čo nám zostali. Za tým už len nasleduje odpoveď servera že potvrdzujem doručenie všetkých dát.



Keď sa pozrieme na programovú časť a samotného prijímateľa vyzerá to nejako takto:

```
1:  Dostal som SYN, Odpovedam SYN ACK
Velkost fragmentu: 4
2:  Dostal som SQN1, Odpovedam SQN1 ACK
Velkost fragmentu: 4
3:  Dostal som SQN2, Odpovedam SQN2 ACK
Velkost fragmentu: 4
4:  Dostal som SQN1, Odpovedam SQN1 ACK
Velkost fragmentu: 4
5:  Dostal som SQN2, Odpovedam SQN2 ACK
Velkost fragmentu: 4
6:  Dostal som SQN1, Odpovedam SQN1 ACK
Velkost fragmentu: 4
7:  Dostal som SQN2, Odpovedam SQN2 ACK
Velkost fragmentu: 4
8:  Dostal som SQN1, Odpovedam SQN1 ACK
Velkost fragmentu: 4
9: Dostal som FIN, Odpovedam FIN ACK
Velkost fragmentu: 2

posielana sprava/subor: testujem posielanie spravy pre pks
Pocet odosielanych fragmentov: 9
Velkost posielanej spravy/suboru: 34B
```

Teda vidíme že veľkosť posielanej správy je 34B čo pri fragmentácií po 4B je 8*4B + 1*2B, zároveň toto môžeme vidieť bolo doručených 8 packetov po 4B a posledný packet mal veľkosť fragmentu 2B. Správa bola úspešne doručená.

Strana odosielateľa vyzerá nasledovne:

```
1: Posielam SYN
Prislo mi: {'checksum': 0, 'flag': 1, 'data': bytearray(b'')}
2: Posielam SQN1
Prislo mi: {'checksum': 0, 'flag': 8, 'data': bytearray(b'')}
3: Posielam SQN2
Prislo mi: {'checksum': 0, 'flag': 9, 'data': bytearray(b'')}
4: Posielam SQN1
Prislo mi: {'checksum': 0, 'flag': 8, 'data': bytearray(b'')}
5: Posielam SQN2
Prislo mi: {'checksum': 0, 'flag': 9, 'data': bytearray(b'')}
6: Posielam SQN1
Prislo mi: {'checksum': 0, 'flag': 8, 'data': bytearray(b'')}
7: Posielam SQN2
Prislo mi: {'checksum': 0, 'flag': 9, 'data': bytearray(b'')}
8: Posielam SQN1
Prislo mi: {'checksum': 0, 'flag': 8, 'data': bytearray(b'')}
9: Posielam FIN
Prislo mi: {'checksum': 0, 'flag': 5, 'data': bytearray(b'')}
```

Teda len zobrazujem poradie odosielaného packetu, plus informáciu čo posielam. Ako výpis som dal výpis môjho celého fragmentu ako hlavička + data aby som prehliadanejšie videl aký flag odpovedá odosielateľ. Dátova časť bude vždy prázdna keďže odosielateľ nemá posielať dáta.

10 Popis používateľského rozhrania

Pri zahájení programu dostane používateľ na výber či chce si vybrať prijímateľa, odosielateľa alebo ukončiť program.

```
1 pre odosielatela
2 pre prijimatela
3 pre exit

1
```

Vyberieme si jednu z možností, teda zvolíme si odosielateľa, kde dostaneme otázku na akú IP adresu prijímateľa sa chceme pripojiť a jeho port.

```
Zadaj IP adresu servera: 127.0.0.1
Zadaj port na ktorom chces komunikovat: 20
```

Keď už toto máme, dostaneme na výber zo 4 možností a to je odoslanie správy, odoslanie súboru, výmenu rolí a ukončenie programu.

```
Vyber si moznost co chces robit:
 1 - odoslat spravu
2 - odoslat subor
3 - vymena
4 - ukoncit program
```

Pri výbere poslania správy, zvolíme akú správu chceme odoslať, následne si vyberieme či chceme simulovať chybu pri prenose a vyberieme veľkosť fragmentácie.

```
Zadaj spravu ktoru chces poslat:sdajkdjsadjsal
Chces simulovat chybu pri prenose(1-ano,0-nie)?0
Zadaj velkost fragmentacie 1-1467B:1
```

Potom sa nám už len zobrazí spomínaný prenos dát ktorý sme mohli vidieť v časti 9. zobrazenie testovacieho prípadu. Pre prijímateľa to vyzerá veľmi jednoducho ten len zadá IP a port na ktorom počúva o ostatok sa stará odosielateľ.

11 Záver a zhodnotenie

Implementácia samotného UDP komunikátora sa nám podarila myslím si že úspešne. Dokázali sme pri testovaní prenášať úspešne správy a súbory. Taktiež pri simulácií chyby sa program zachoval správne a odpovedal flagom pre zle doručený packet a vyžiadal si daný packet na novo. Implementované bolo aj ošetrenie prerušenia v prípade že by sme prerušili prenos tým že vytiahneme kábel. Nastane timeout po 60 sekundách kde sa oba programy vrátia do menu.

Projekt bol pre nás dobrou skúsenosťou z programovej časti ale taktiež aj z vedomostnej časti. Bolo potrebné najskôr nadobudnúť znalosti ako vlastne UDP komunikácia prebieha, aké flagy sa môžu používať a podobne. Myslím si že tento projekt bol prínosom.