SLOVENKSÁ TECHNICKÁ UNIVERZITA
V BRATISLAVE
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 19 Bratislava 4

PKS – 2. Zadanie: Komunikácia s využitím UDP protokolu

Martin Beňa
FIIT STU
Cvičenie: Utorok 16:00
11.12.2022

Úvod

Zadanie som vypracoval v programovacom jazyku Python verzia 3.10. Cely program je postavený na jednej triede (Class), ktorá reprezentuje komunikačný uzol v sieti. Táto trieda obsahuje 19 metóda, ktoré slúžia na vytvorenie, udržanie komunikácie a zároveň umožňuje posielať a prijímať dáta. Na udržanie spojenia posiela klient každých 5 sekúnd správu s hlavičkou bez dát a očakáva odpoveď pokiaľ je program v štádiu nečinnosti (idle).

Dáta sú pred odoslaním zabalené od vlastnej hlavičky, ktorá sa skladá z 3 časti.

Vlastná hlavička

Dáta sú pred odoslaním zabalené od vlastnej hlavičky (Obr.1), ktorá sa skladá z 3 časti: flags, celkový počet fragmentov/číslo aktuálneho fragmentu, a kontrolná CheckSum. Celková veľkosť hlavičky je 8 bajtov a maximálna veľkosť posielaného rámca teda bude bude 1472 – 8 = 1464 bajtov.

Oproti návrhu sa líši iba tým, že som zmenšil flags a spojil aktuálny a celkový počet fragmentov.

	Octets	0								1								2								3							
Octets	Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Flags								Počet celkových fragmentov / poradové číslo aktuálneho fragmentu																							
4	32	Kontrola fragmetnu (Checksum)																															

Obr.1 hlavička a jej časti

Flags

Táto časť hlavičky obsahuje informáciu o type správy. V tabuľke nižšie(Tab.1) sú zobrazené všetky aktuálne možnosti pre typy správ. Od návrhu si flag líšia oba tým, že sú skrátené na 1 bajt.

Flags	Typ správy	
0	Žiadosť o zahájenie komunikácie	2-way-handshake
1	Potvrdenie o prijatí zahájenie komunikácie	
2	Posielanie textu	
3	Obsah súboru	
4	Názov súboru	
5	Počet nasledujúcich fragmentov	
6	Potvrdenie o úspešnom prijatí	
7	Opätovné vyžiadanie fragmentov	
8	Keep alive	
9	Odpoveď na keep alive	
10	Žiadosť na výmenu roli	
11	Informácia o odpojení klienta	
12	Oznámenie servera klientovi o zmene roli	
13	Pripravenie nové servera + port na ktorom bude nový server počúvať	
14	Zachovanie stavu roli	
15	ACK	

Tab.1 tabuľka flagov

Aktuálny a celkový počet fragmentov

Informácia o poradí posielanom rámci alebo o celkovom počte očakávaných rámcov. Tieto informácie sú dôležite aby prijímateľ vedel usporiadať prijate rámce do správneho poradia, nakoľko ich poradenie nie je garantované a taktiež bude vedieť kedy prijal všetky rámce. Príjemca vie rozlíšiť či sa jedná o poradie alebo počet vďaka flagu. Ak je flag nastavený na 5, tak číslo nám hovorí o počte fragmentov.

CheckSum

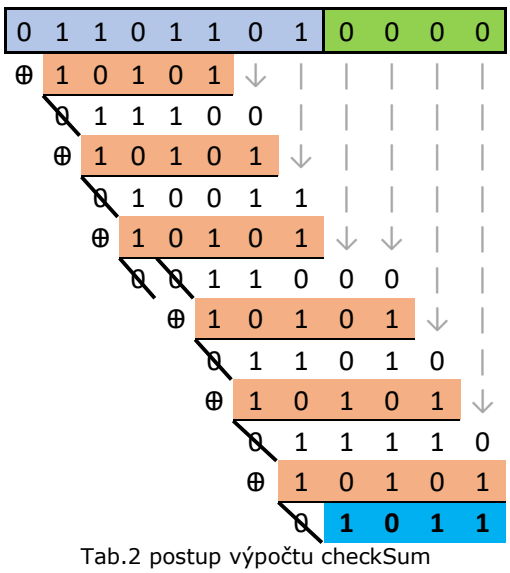
Posledná časť bude obashuje výsledok z kontroly posielaného rámca. Po prijatí si prijímateľ vypočíta vlastnú hodnotu a skontroluje si ju s doručenou. Pokiaľ by sa líšili, tak došlo k chybe pri posielaní a požiada o odosielateľa o opätovné zaslanie. Na výpočet používam funkcie *crc32* z python knižnice *binascii*. Pre túto funkciu som sa rozhodla lebo poskytuje menšiu šancu rovnakého výsledku ako jeho 2 bytový ekvivalent *crc16*. Pre porovnanie *crc32* má 2^{32} (4 294 967 296) možných výsledkov zatiaľ čo *crc16* má iba 2^{16} (65 536) možných výsledkov. Šanca, že nám *crc32* dá rovnaký výsledok je 0,000 000 000 232 831 % = 2,3 * 10^{-10} % a pri crc16 to je 0,000 015 258 789 % = 1,5 * 10^{-5} %.

Crc32 sa počíta tak, že sa urči polynóm, k dátam je pripočítaných 32 núl a následne začína je používaná operácie logickej nezhody, až kým nedostane výsledok, ktorý menší ako 32 bitov- Toto číslo bude reprezentovať náš checkSum.

Polynóm použitý touto funkciou:
$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + x^0$
v binárnej sústave tento polynóm vyzerá nasledovne:

1 0000 0100 1100 0001 0001 1101 1011 0111

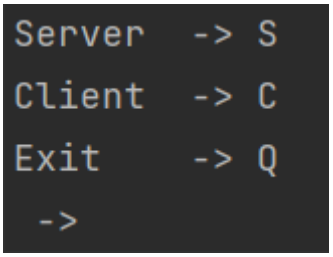
Na ukážku je na tabuľke nižšie (tab.2) vypočítaný *crc4* s polynómom $x^4 + x^2 + x^0$ = 10101 a dátami 0110 1101



Tab.2 postup výpočtu checkSum

Priebeh programu

Program začína krátkym menu (Obr.2) v ktorom sa používateľ rozhoduje, ktorú verziu spustí. Do tohto menu sa používateľ opäť môže dostať, ak bude jeho komunikácia ukončená.



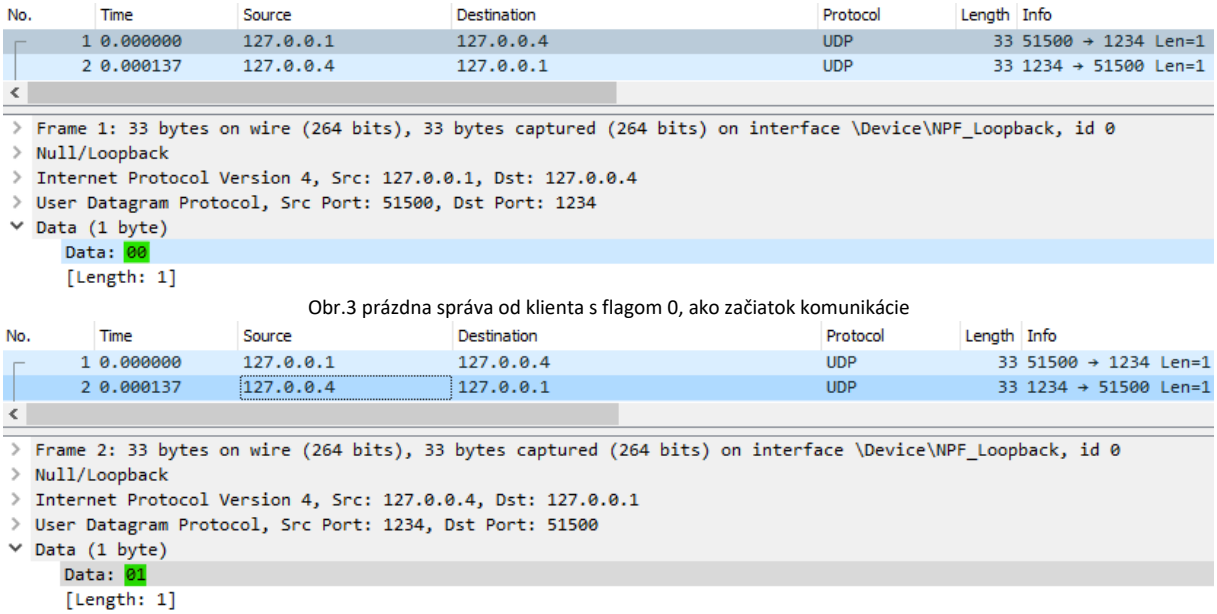
Obr.2 úvodne menu

Pre voľbu klienta (odosielateľa) musí používateľ zadať písmeno „C“ a pre server (prijímateľ) písmeno „S“, ak by chcel program ukončiť tak musí zadať písmeno „Q“.

Nadviazanie spojenia

Spojenie sa nadviaže, tak že klient pošle prázdnu správu s flagom 0 na adresu servera a očakáva odpoveď v podobe prázdnej správy s flagom 1.

Na obrázkoch nižšie môžeme vidieť počiatok komunikácie na localhost kde client s IP 127.0.0.1 poslal správu, ktorá obsahovala iba 1 byte (flag 0) (Obr.3) a ako odpoveď mu prišla od servera s IP 127.0.0.4 správa s taktiež 1 bytom (flag 1) (Obr.4)



Obr.3 prázdna správa od klienta s flagom 0, ako začiatok komunikácie

Obr.4 prázdna správa od servera s flagom 1, ako pozitívna odozva

Klient

Ako prvé klient bude vyžadovať IP adresu na ktorú sa má pripojiť (IP servera) a potom si vypýta port. Pokiaľ mu budú tieto hodnoty zadané spustí sekvenciu na pripojenie k serveru. Táto sekvencia sa každé 2 sekundy pokúsi pripojiť na zadanú IP adresu a port. Pokiaľ sa jej to však nepodarí, tak po 5. pokuse ukončí proces pripájania a prejde do úvodného menu (Obr.5).

```
Server  -> S
Client  -> C
Exit    -> Q
 -> c
Starting Client:
IP: 168.172.42.4
port:1434
Trying to connect to 168.172.42.4 on port 1434
Trying to connect to 168.172.42.4 on port 1434
Trying to connect to 168.172.42.4 on port 1434
Trying to connect to 168.172.42.4 on port 1434
Trying to connect to 168.172.42.4 on port 1434
Server  -> S
Client  -> C
Exit    -> Q
 ->
```

Obr.5 (Client) Neúspešný pokus o pripojenie k serveru

Pokiaľ by sa klientovi podarilo sa k serveru, vypíše sa správa o úspechu a následne sa zobrazí aj používateľské menu (Obr.6), z ktorého si používateľ môže vybrať, ktorú akciu chce vykonať.

```
Server  -> S
Client  -> C
Exit    -> Q
 -> c
Starting Client:
IP: 192.168.1.16
port:1234
Trying to connect to 192.168.1.16 on port 1234
Successfully connected to a server.
To send message type:   MESSAGE or M
To send file type:      FILE or F:
To change frag size:    CHANGE or C
To switch roles:        SWITCH or S
To disconnect type:     QUIT or Q
 ->
```

Obr.6 (Client) používateľské menu po úspešnom pripojený na server

Používateľ môže poslať správu, ktorá sa vypíše do terminálu, súbor, zmeniť maximálnu veľkosť posielaných fragmentov, vyžiadať o zmenu rolí (výmena uzlov medzi serverom a klientom) a ukončiť spojenie.

Server

Ako prvé server bude vyžadovať IP adresu a port na ktorom bude počúvať. Pokiaľ mu budú tieto hodnoty zadané spustí sa počúvanie kde server čaká na žiadosť o zahájenie komunikácie od klienta. (Obr.5)

```
Server  -> S
Client  -> C
Exit    -> Q
 -> s
Starting Server:
IP: 192.168.1.17
Port:1423
('192.168.1.17', 1423)
```

Obr.5 (Server) server čaká na klienta

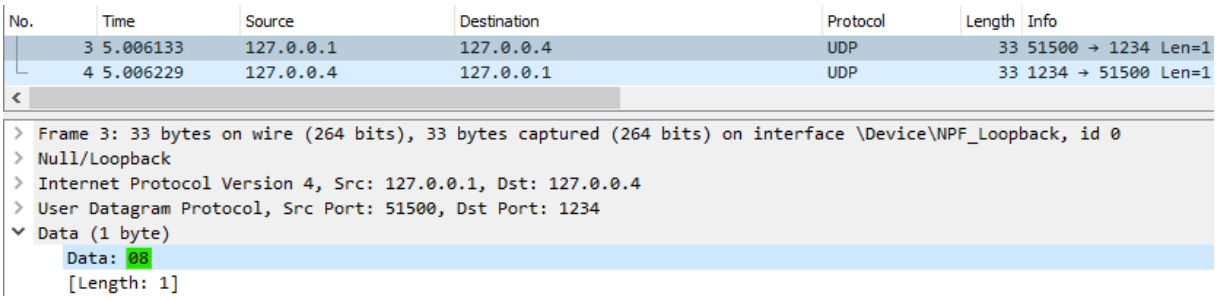
Ak sa na server podarí pripojiť, server vypíše sa správu o úspešnom nadviazaní komunikácie a ďalej čaká na ďalšie kroky používateľa (Obr.6). Ak používateľ nepošle žiadnu správu, server obdrží a pošle späť odozvu na správu o udržaní spojenia (keep alive).

```
Server  -> S
Client  -> C
Exit    -> Q
 -> s
Starting Server:
IP: 192.168.1.17
Port:1234
('192.168.1.17', 1234)
Client IP: 192.168.1.16 and port: 53778
```

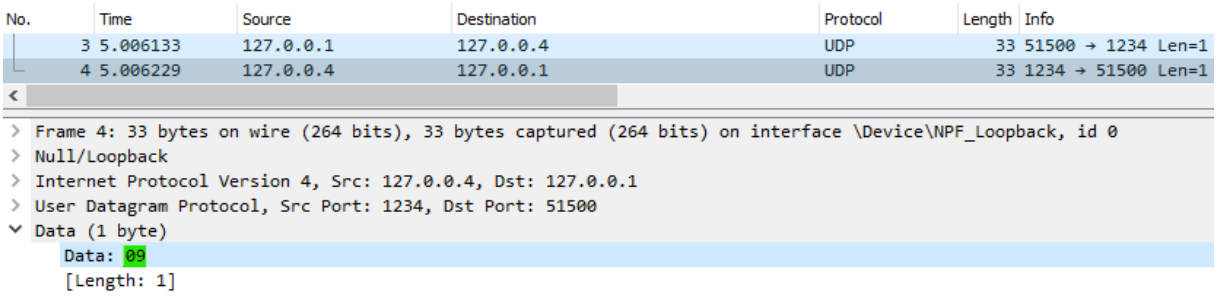
Obr.6 (Server) server vypísal informácie o pripojenom klientovi a očakáva ďalšie správy

Keep alive

Na udržanie spojenia sa používať metódu keep alive, kde klient počas nečinnosti v časových intervaloch o dĺžke 5 sekúnd posiela správy bez dát s *flagon* 8 (Keep alive). Na obrázkoch z Wireshark je zachytený keep alive request (Obr.7) a replay (Obr.8).



Obr.7 prázdna správa od klienta s flagom 8, keep alive request

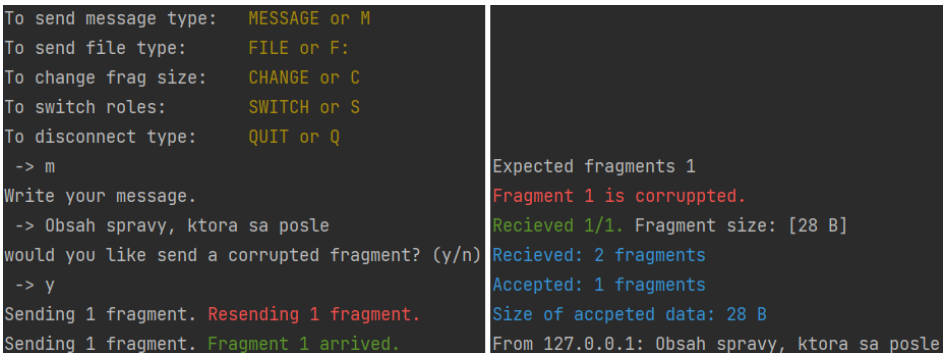


Obr.8 prázdna správa od klienta s flagom 9, keep alive replay

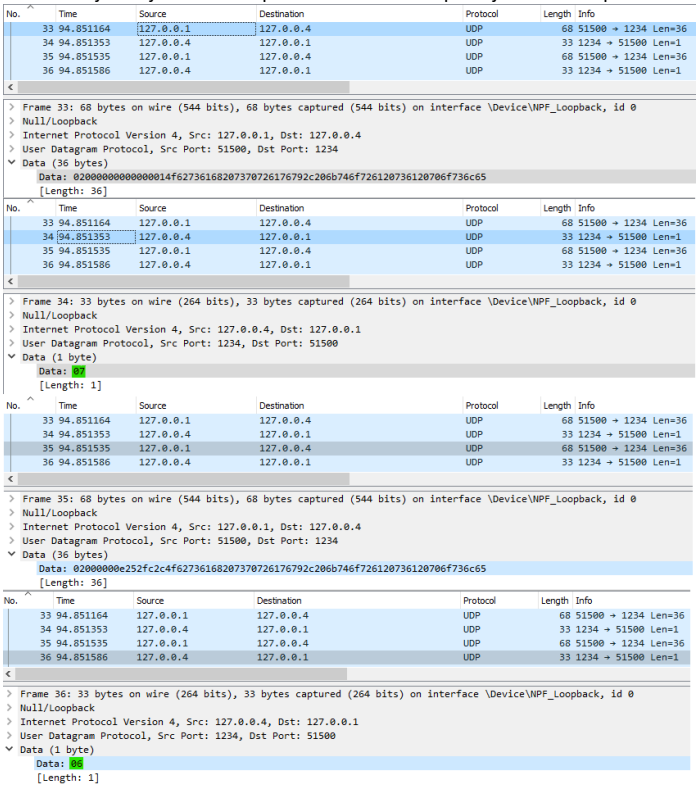
ARQ

ARQ metóda nám dohliada na to aby boli všetky rámce doručené. Pokiaľ by nejaký rámce chýbal alebo by nebol doručený v čas, tak si príjemca vyžiada o jeho opätovne zaslanie.

Názorná ukážka na snímkach z Wiresharku (Obr.10) a kódu (Obr.9) nižšie.



Obr.9 Na ľavej časti je terminál z pohľadu klienta a na pravej časti zase z pohlaď server

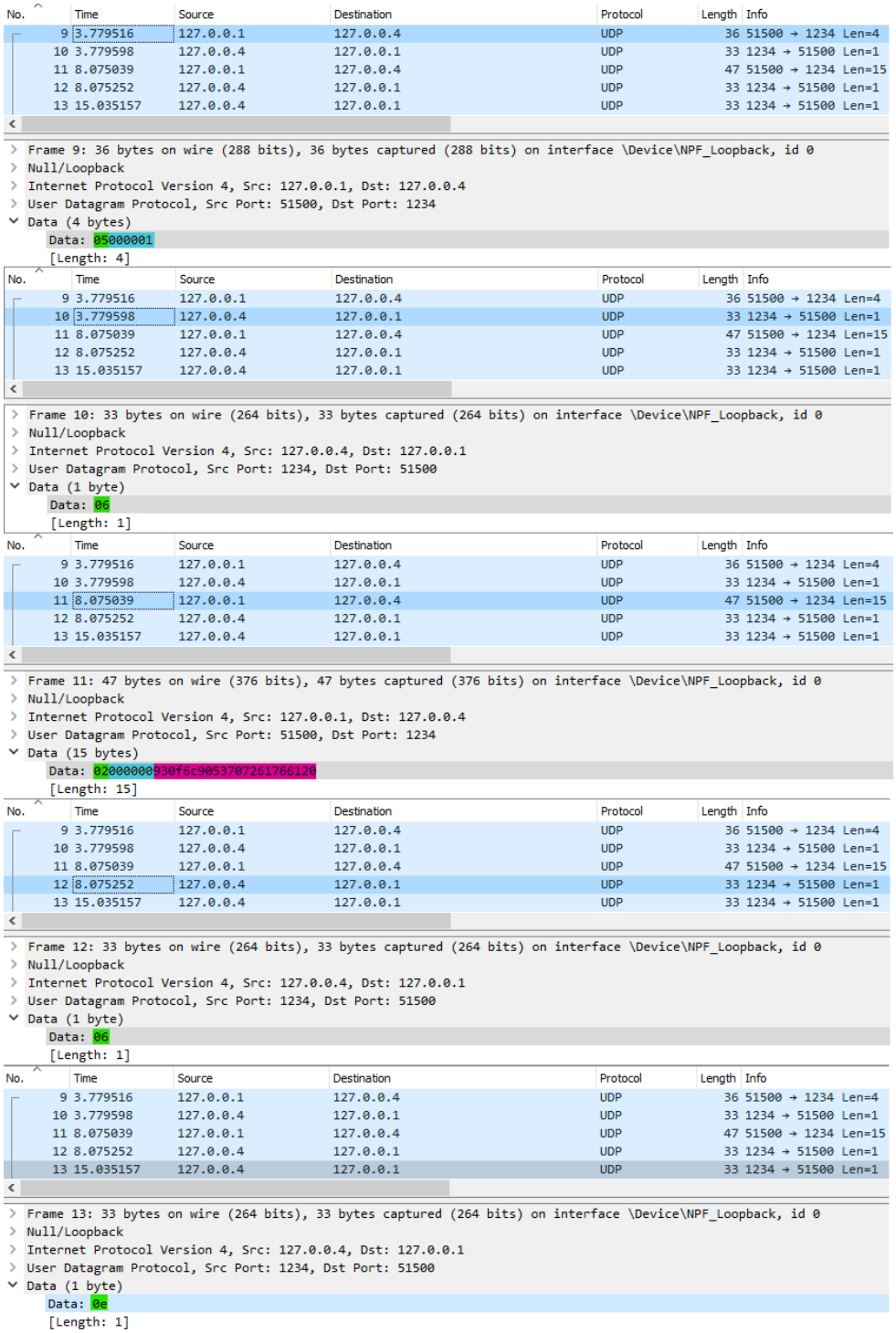


Obr.10 Opätovné poslanie (flag 7) servera o packet a jeho následne podvedenie (flag 6)

V mojom riešení využijem metódu Stop & Wait, kde odosielateľ pošle 1 rámcov a čaká na správu o jeho prijatí. Pokiaľ by dostal správu o chybovom prijatí, tak pošle opäť ten istý rámcov. ARQ sa líši od návrhu v tom, že neposielam n fragmentov ale iba 1.

Ukážka z Wireshakru

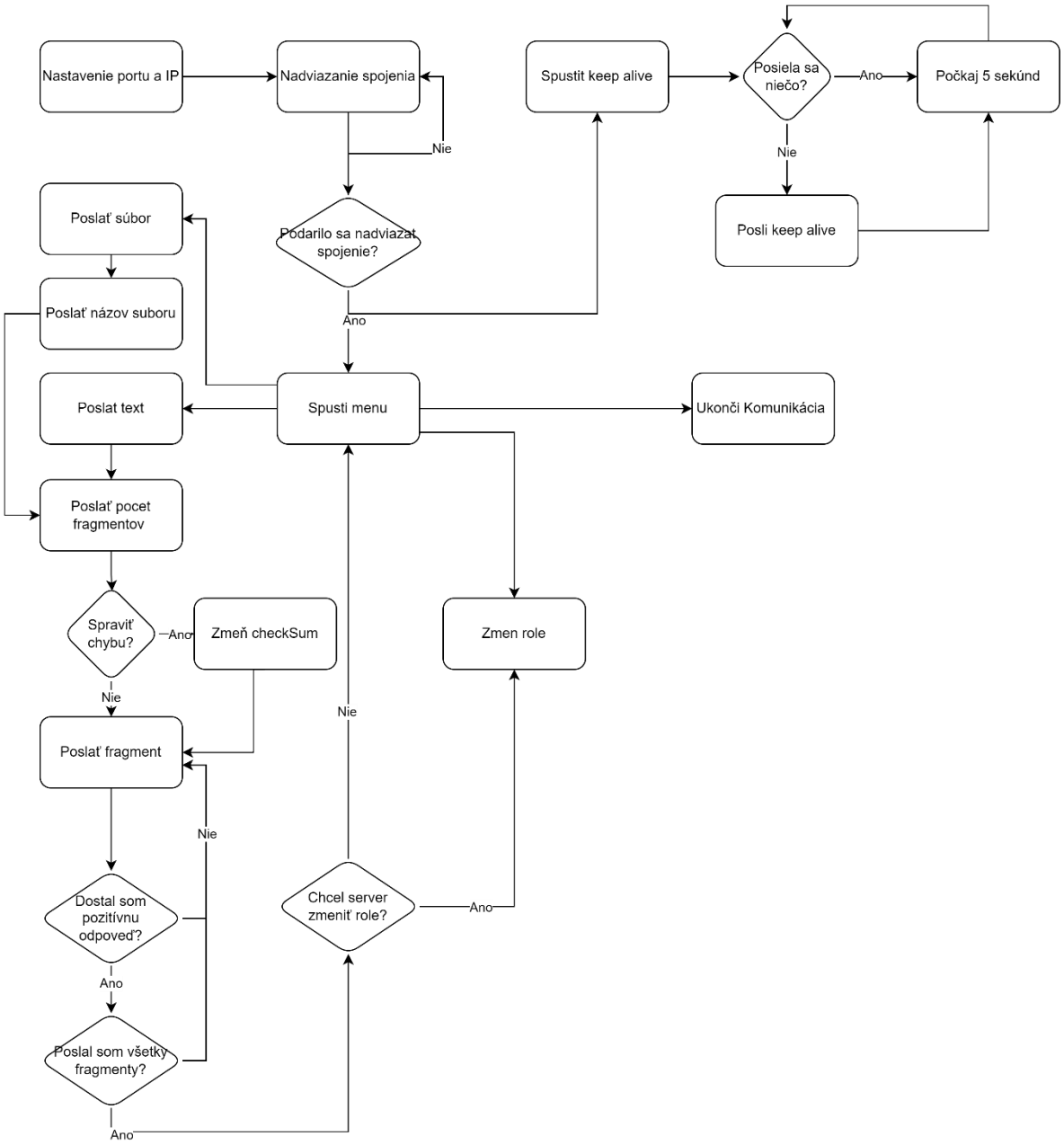
Ukážka ako vyzerá posielanie správy cez zachytené packety na lokálnej sieti. Posledný packet je odpoveď serveru, že nežiada o zmenu roli.



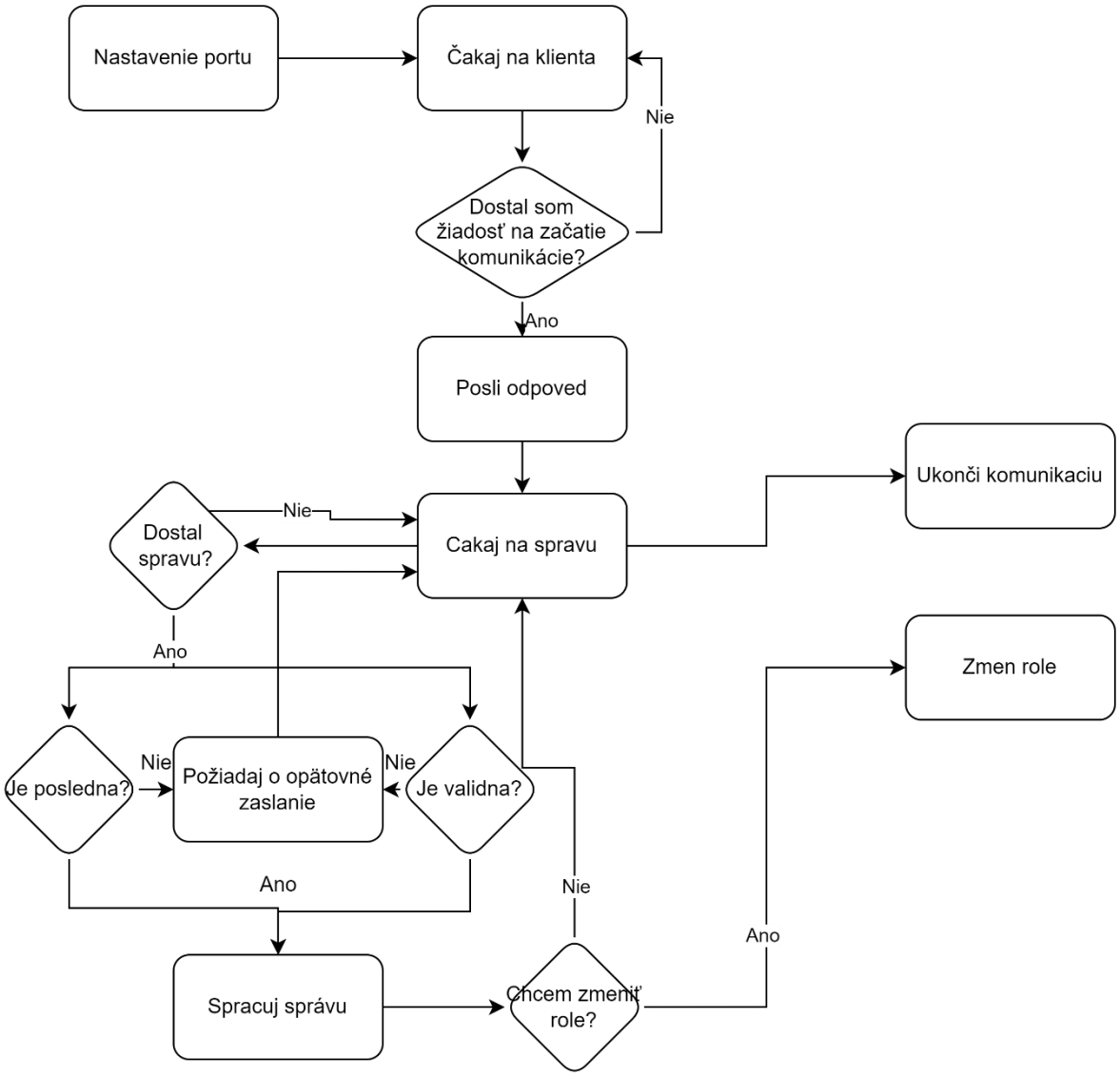
Obr.11 komunikácia pri posielaní správy, zelena = flag, modra = počet / poradové číslo fragmentu a bordova = dáta

Diagram

Klient (Odosielateľ)



Prijímateľ (Server)



Záver

Vďaka tomu zadaniu som mohol vidieť ako funguje posielanie dát a taktiež som si mohol rozšíriť svoje programovacie znalosti v jazyku Python.