SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta informatiky a informačných technológii

Počítačové a komunikačné siete

Zadanie 2:

Komunikácia s využitím UDP protokolu

Tomáš Socha

ID: 110896

Akademický rok 2021/2022

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 5-20s pokiaľ používateľ neukončí spojenie. Odporúčame riešiť cez vlastne definované signalizačné správy.

Program musí mať nasledovné vlastnosti (minimálne):

1.Program musí byť implementovaný v jazykoch C/C++ alebo Python s využitím knižníc na prácu s UDP socket, skompilovateľný a spustiteľný v učebniach. Odporúčame použiť python modul *socket,* C/C++ knižnice *sys/socket.h* pre linux/BSD a *winsock2.h* pre Windows. Iné knižnice a funkcie na prácu so socketmi musia byť schválené cvičiacim. V programe môžu byť použité aj knižnice na prácu s IP adresami a portami:

2. Program musí pracovať s dátami optimálne (napr. neukladať IP adresy do 4x int).

3. Pri posielaní súboru musí používateľovi umožniť určiť cieľovú IP a port.

4. Používateľ musí mať možnosť zvoliť si max. veľkosť fragmentu.

5. Obe komunikujúce strany musia byť schopné zobrazovať:

a) názov a absolútnu cestu k súboru na danom uzle,
b) veľkosť a počet fragmentov.

6. Možnosť simulovať chybu prenosu odoslaním minimálne 1 chybného fragmentu pri prenose súboru (do dátovej časti fragmentu je cielene vnesená chyba, to znamená, že prijímajúca strana deteguje chybu pri prenose).

7. Prijímajúca strana musí byť schopná oznámiť odosielateľovi správne aj nesprávne doručenie fragmentov. Pri nesprávnom doručení fragmentu vyžiada znovu poslať poškodené dáta.

8. Možnosť odoslať 2MB súbor a v tom prípade ich uložiť na prijímacej strane ako rovnaký súbor, pričom používateľ zadáva iba cestu k adresáru kde má byť uložený.

Program musí byť organizovaný tak, aby oba komunikujúce uzly mohli prepínať medzi funkciou vysielača a prijímača bez reštartu programu - program nemusí (ale môže) byť vysielač a prijímač súčasne. Pri predvedení riešenia je podmienkou hodnotenia schopnosť doimplementovať jednoduchú funkcionalitu na cvičení.

2. Zmeny oproti návrhu

Zmien oproti návrhu som spravil len minimum. Väčšinou išlo len o drobné úpravy.

a) CRC metóda

Najvýraznejšou zmenou oproti návrhu bola zmena CRC metódy. V návrhu som plánoval s použitím CRC32 hlavne kvôli tomu že ide o najznámejšiu používanú CRC metódu, no zaberala v hlavičke 4B. Na základe podnetu od cvičiaceho som ju zmenil a nahradil CRC16(xmodem). Touto zmenou som dosiahol zmenšenie veľkosti hlavičky o 2B, takže po novom už má veľkosť 6B. Tým pádom sa aj zväčšila maximálna dĺžka fragmentu o 2B. Implementáciu som prebral z knižnice *binascii*. Generujúci polynóm má v tomto prípade tvar: x^16+x^12+x^5+1

Čo predstavuje binárne tvar: 1 0001 0000 0010 0001

Výsledná hodnota crc bude mať tým pádom veľkosť 16-bitov. Postup vypočítania hodnoty CRC je nasledovný. Najskôr sa ku bitom predstavujúce dáta z konca prilepí 16 núl. Potom sa naplní posuvný register bitovým posunom doľava. Zvyšok vstupných dát sa nachádza napravo od tohto registra.

Po naplnení sa pozrieme na prvý bit v registri(zľava), a ak je jedna tak vykonáme posun vľavo, čiže prvý bit už bude mimo posuvného registra, ale ešte si ho zapamätáme. Následne vykonáme operáciu xor s polynómom a výsledok vložíme do registra. 1.bit,ten ktorý už bol mimo registra sa po aplikovaní XOR s 1 zmenil na 0, takže nám vypadol.

Novú hodnotu v posuvnom registri posúvame doľava, kým nám znovu na jeho začiatku nenachádza 1. potom opakujeme rovnaký postup(posun o 1,xor, posun).Keď sa už naľavo od posuvného registra nebudú nachádzať žiadne bity, tak v posuvnom registri sa bude nachádzať výsledná hodnota CRC.

Obrázok nižšie znázorňuje príklad výpočtu crc hodnoty pre písmeno T.

vstup:	"T" v binarnom tvare 0101 0100
vstup + 16 nul:	0101 0100 0000 0000 0000 0000
polynomial:	100 0100 0000 1000 01
	001 0000 0000 1000 0100 0000
	1 0001 0000 0010 0001
	0 0001 0000 1010 0101 0000
	1 0001 0000 0010 0001
	0 0001 1010 0111 0001
	0x 1 A 7 1

Obrázok 1 – príklad výpočtu CRC16

b) Keep-alive

Pri keep-alive som zanechal 3s interval odosielania správ, no mierne som zväčšil toleranciu z 2 na max 3 nepotvrdené správy, nastavením 10s čakacej doby na potvrdenie keep-alive správy.

c) Výmena rolí

Princíp pri výmene rolí zostáva oproti návrhu nezmenený, no pridal som miernu funkcionalitu, kedy po požiadaní zmenu rolí príde druhej strane správa, ktorú môže prijať alebo odmietnuť. Strane servera bude táto správa doručená aj keď sa nenachádza v stave počúvania, čiže sa nachádza v ponuke a nezvolil si zatiaľ žiadnu možnosť.

d) Simulácia chyby

V simulácií chyby nastala len jedna zmena oproti návrhu. Počet poslaných chybných fragmentov zostal nezmenený. V prvých 20 bol každý nepárny poslaný chybný, avšak ak bol celkový počet fragmentov menší tak bol počet chybne poslaných menší.

V prípade neposlania druhého a posledného fragmentu, som sa rozhodol simulovať stratu doručenia len na 2. fragmente. Chybu som simuloval tým, že som z dátovej časti odstránil posledný byte.

e) Ukončenie spojenia

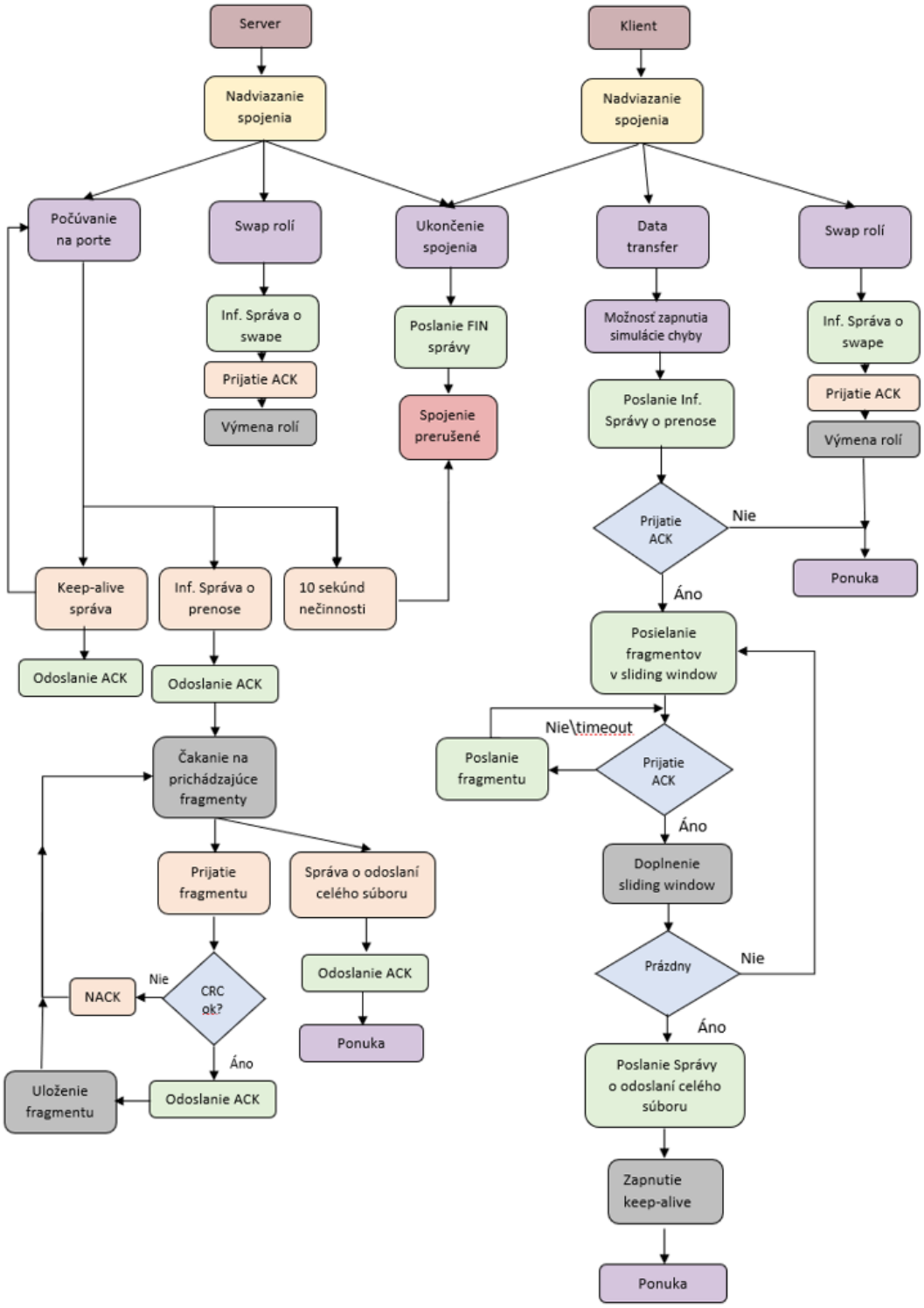
Princíp ukončenia spojenia taktiež zostane oproti návrhu nezmenený, kde sa spojenie ukončí poslaním FIN správy. Ale implementoval som skončenie aj druhej strany po tom, ako mu bude oznámené, že spojenie bolo ukončené. Takže po zvolení ukončenia spojenia sa ktorejkoľvek strane sa vypnú oba koncové uzly.

f) ARQ

Zvolená selective ARQ metóda zostala oproti návrhu nezmenená.

3. Diagram spracovávania komunikácie

Diagram nižšie zobrazuje priebeh spracovania komunikácie. Diagram sa skladá z 2 častí: časť servera a klienta. V prípade serveru je po nadviazaní spojenia možné prijať dáta aj keď sa nachádza v ponuke. A to takým spôsobom, že mu príde správa z druhého koncového uzla, ktorú bude môcť potvrdiť alebo prijať.



Obrázok 2- Diagram spracovávania komunikácie

4. Opis častí zdrojového kódu

Ako implementačné prostredie som si zvolil jazyk Python vo verzii 3.9 hlavne kvôli jednoduchej práci s reťazcami znakov a poľami.

a) Použité knižnice

- **os ->** funkciu _exit() – pre ukončenie programu
- **random ->** funkciu random() – na vygenerovanie náhodného čísla pri simulácii chyby.
- **socket ->** funkciu *socket()* **-** na vytváranie socketov
- **struct ->** funkcie *pack()* a *unpack()* – na vytvorenie bajtového objektu, a jeho rozbalenie
- **time ->** funkciu *sleep()* – vo funkcii keep-alive na pravidelne posielanie Keep-alive správ
- **threading->** funkciu *Thread()* – pre vytvorenie threadov pre Keep-alive a na dostávanie ack správ od prijímateľa pri posielaní dát.
- **Binascii ->** funkciu *crc_hqx()* – pre výpočet crc hodnoty

```
import os
import random
import socket
from struct import pack, unpack
from time import sleep
import threading
import binascii
```

Obrázok 3 – použité knižnice v zdrojovom kóde

b) Globálne premenné

- ***FORMAT* ->** premenná slúžiaca na uloženie formátu dekódovania správ
- ***keep_alive_thread* ->** premenná slúži na uloženie vytvoreného threadu pre keep-alive
- ***keep_alive_status* ->** premenná slúžiaca na ukončenie while cyklu v keep-alive threade
- ***server_pocuvanie_bg_status* ->** premenná slúži na ukončenie while cyklu v threade pre počúvanie na pozadí
- ***server_bg_thread* ->** premenná slúži na uloženie vytvoreného threadu pre počúvanie na pozadí

```
FORMAT = "utf-8"

keep_alive_thread = None
keep_alive_status = False

server_pocuvanie_bg_status = False
server_bg_thread = None
```

Obrázok 4 – použité globálne premenné

c) Funkcie klienta

Funkcie pre klienta sa nachádzajú v sekcii klient. Táto sekcia je označená komentárom.

client_start()

Funkcia slúžiaca na nadviazanie spojenia ponocou odoslania syn a prijatia ack správy. Timeout je nastavený na 60s, po ktorých dôjde ku vypnutiu programu

client_main()

Hlavná funkcia ktorá zabezpečuje zobrazenie ponuky a vykonanie príslušnej operácie(zavolanie funkcie) podľa výberu.

client_prenos_dat()

Funkcia zavolaná pri výbere možnosti prenosu dát. Zabezpečuje celý prenos jedného súboru alebo správy. Najskôr však žiada od používateľa zadanie cesty k prenášanému súboru, zadania veľkosti fragmentu a či chce simulovať chybu.

selective_arq_prenos()

Funkcia zavolaná funkciou client_prenos_dat() na zabezpečenie posielania fragmentov pomocou selective arq metódy. Dáta sú najskôr rozdelené do fragmentov podľa max. zadanej veľkosti. Následne sa naplní sliding window indexami prvých packetov. Potom sa vytvorí nový thread na počúvanie prijatých ack/nack správ a začnú sa posielať fragmenty a dopĺňať sliding window novými indexami.

pocuvanie_ack_packetov()

Zavolaná funkciou selective_arq_prenos() vytvorením nového threadu. Slúži na prijímanie odpovedí od servera. Dostávanie ACK/NACK odpovedí na doručené fragmenty, či prišli v poriadku alebo došlo ku zmene dát pri prenose. Pre ktoré fragmenty bolo doručené ACK tak tie sa vyberú zo sliding window. Pre ktoré bolo doručené NACK sú vložené do listu pre nesprávne doručené fragmenty, a bude ich potrebné znovu poslať. Timeout je nastavený na 2 sekundy, kedy ak za ten čas nepríde žiadna správa, tak sú fragmenty v sliding window poslané znovu.

keep_alive()

Slúži na posielanie keep-alive správ a prijímanie odpovedí. Spustená funkciou client_main() v novom threade. Pri 3 neodpovedaní na keep-alive správu dôjde ku vypnutiu klienta pretože server už nepočúva. V prípade tvrdého vypnutia servera sa hneď vypne aj klient.

nacitanie_suboru()

Zabezpečuje správne načítanie súboru zo zvolenej absolútnej adresy.

d) Funkcie servera

Funkcie pre server sa nachádzajú v sekcii server. Táto sekcia je označená komentárom

server_start()

Slúži na korektné nadviazanie spojenia. Po prijatí SYN správy odošle ACK .

server_main()

Hlavná funkcia servera. Zabezpečuje zobrazenie ponuky a vykonanie príslušnej operácie(zavolanie funkcie) podľa výberu. Taktiež vytvára nový thread pre počúvanie na pozadí.

server_pocuvanie_bg()

Slúži na počúvanie prichádzajúcich správ od klienta na pozadí, čiže v čase keď sa server nachádza v ponuke a nebola ešte zvolená žiadna možnosť. Beží v ďalšom threade.

server_pocuvanie_fg()

Slúži na počúvanie po vybratí možnosti počúvanie z ponuky. Po prijatí správy od servera podľa jej typu vhodne zareaguje.

server_prijimanie_fragmentov()

Funkcia zabezpečuje prijímanie fragmentov a ich kontrolu, či prišli neporušené. Taktiež odosiela odpovede serveru pre každý prijatý fragment. Najskôr sa vytvorí prázdny list, do ktorého sú postupne vkladané správne doručené fragmenty na indexy príslušného čísla fragmentu. Po prijatí správy o doručení všetkých fragmentov vráti dáta do funkcie ktorá ju zavolala.

rozbalenie_hlavicky()

Zabezpečí rozbalenie hlavičky každého doručeného fragmentu.

ulozenie_suboru()

Uloží prenesený súbor na používateľom zadanú adresu.

e) Spoločné funkcie

Ide o funkcie ktoré používa prijímajúca, ako aj odosielajúca strana. Nájdeme ich v sekcii spoločné.

main()

Hlavná funkcia celého programu. Zobrazí začiatočnú ponuku výberu roly. Je možné si zvoliť byť ako prijímač, alebo vysielač. Následne podľa výberu sa spustí príslušná funkcia buď servera alebo klienta pre nadviazanie komunikácie.

vymena_roli()

Zavolá sa po zvolení výmeny roli z ponuky. Zabezpečí výmenu rolí odoslanim inf. Správy o swape a následnom potvrdení od druhej strany.

ukoncenie_programu()

Spustená po zvolení danej možnosti z ponuky. Odošle inf. Správu o ukončení spojenia a vypne daný program

5. Používateľské prostredie

Navrhnutý program využíva používateľské prostredie ktoré vyžaduje interakciu s používateľom. Po spustení programu sa vypíše hlavička obsahujúca názov programu a meno autora. Taktiež sa vypíše požiadavka na zvolenie funkcionality programu, či pôjde o vysielač alebo prijímač.

```
-------------------------------------
|  UDP komunikátor                  |
|  autor: Tomáš Socha               |
-------------------------------------

vyber funkcionalitu programu:
-----------------
Prijímač - p
Vysielač - v
-----------------
zvoľ 1 možnosť:
```

Obrázok 5 – základné menu po spustení

Po zvolení konkrétnej funkcionality a zadaní portu a IP adresy sa zobrazí ponuka.

```
Klient prihlaseny na server ('127.0.0.1', 2525)

Ponuka:
----------------
Prenos dát  - p
Výmena rolí - v
Ukončenie   - q
----------------
Vyber 1 možnosť:
```

Obrázok 6 – Ponuka po prihlásení klienta

```
[NOVÉ SPOJENIE] ('127.0.0.1', 65211) úspešne pripojené.

Ponuka:
----------------
Počúvanie   - p
Výmena rolí - v
Ukončenie   - q
----------------
Vyber 1 možnosť:
```

Obrázok 7 – Ponuka po prihlásení servera

Po zvolení možnosti prenosu treba vyplniť požadované polia, vypíše sa sumár prenášaných dát a následne prebehne samotný prenos dát.

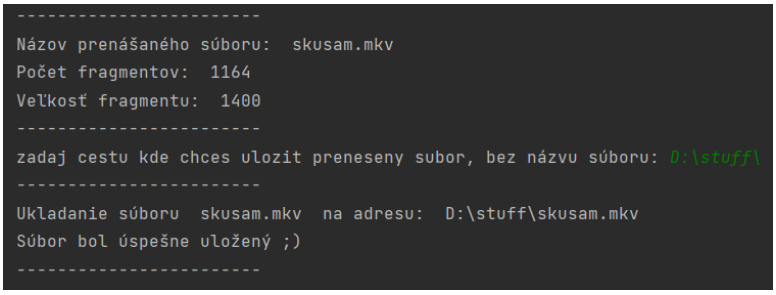
```
Zvoľ čo chceš prenášať:
-------------
Súbor  - f
Správu - m
-------------
zadaj: f
Zvolený súbor
Zadaj názov súboru aj s jeho absolútnou cestou:D:\skusam.mkv
Zadaj max. veľkosť fragmentu:1400
Chceš zapnúť simuláciu chyby? a/n:a

------------------------
Prenášaný súbor:  skusam.mkv
Absolútna cesta:  D:\skusam.mkv
Počet fragmentov:  1164
Max. veľkosť fragmentu:  1400
------------------------
```

Obrázok 8 - príprava na prenos súboru

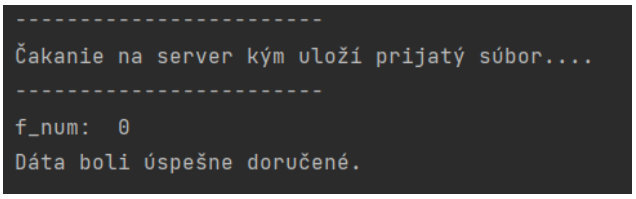
```
f_num:  1
NACK
f_num:  3
NACK
f_num:  4
ACK
f_num:  5
NACK
f_num:  6
ACK
f_num:  7
NACK
f_num:  8
ACK
f_num:  9
NACK
f_num:  10
ACK
```

Obrázok 9 – ukážka časti prenosu súboru

Po skončení prenosu je na strane prijímača potrebné zadať absolútnu adresu, kde sa má doručený súbor uložiť. Neplatí to v prípade, že bola doručená správa, tá je len vypísaná na obrazovku. Na prijímajúcej strane sa vypíše inf. Správa



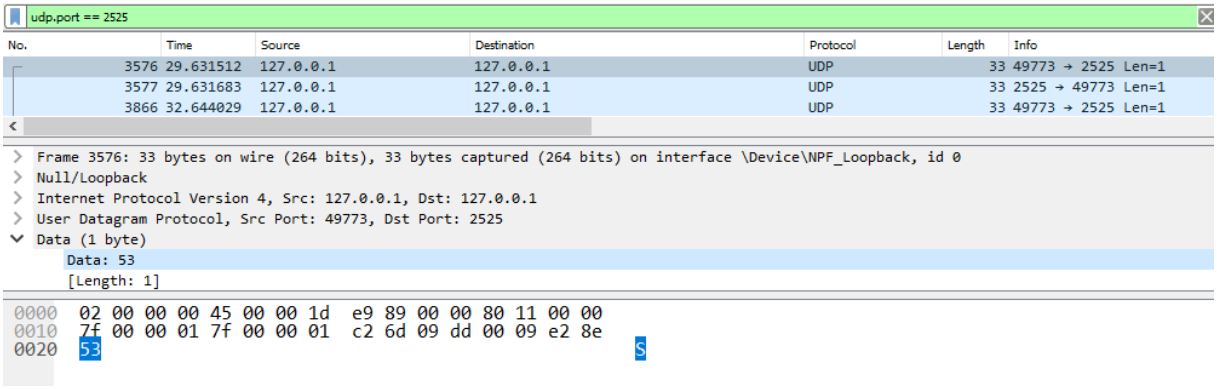
Obrázok 10 - sumárny výpis po prijatí súboru na strane prijímača



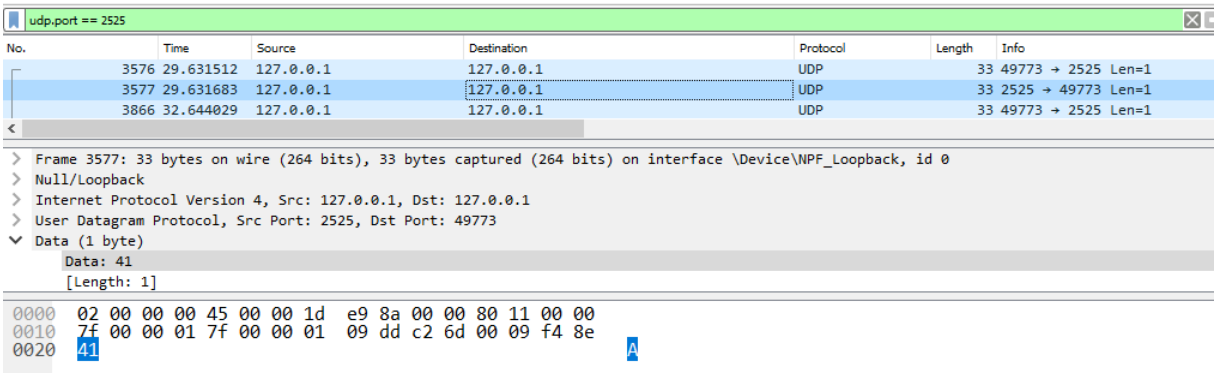
Obrázok 11 – Oznámenie po doručení súboru na odosielajúcej strane

6. Komunikácia vo Wireshark-u

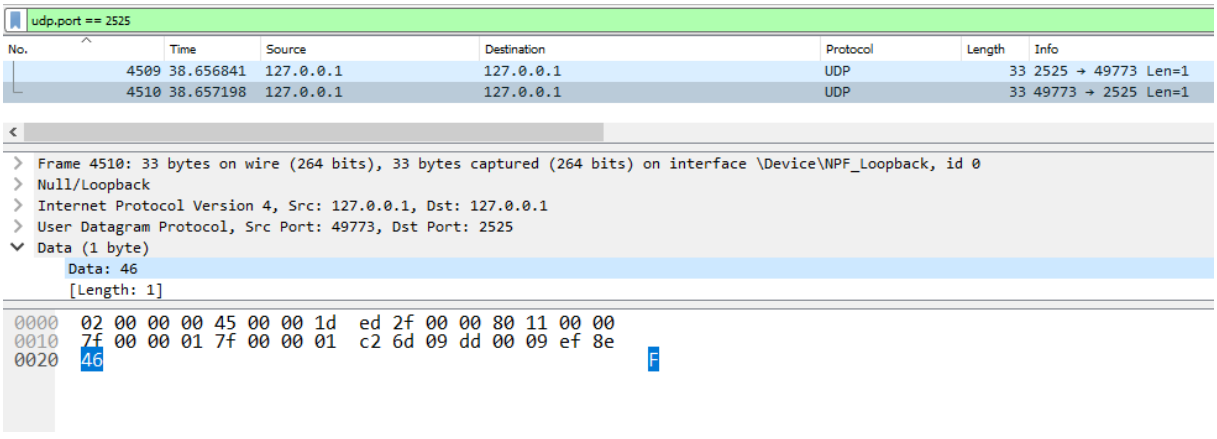
a) Nadviazanie a ukončenie spojenia



Obrázok 12 – poslanie SYN správy z vysielača



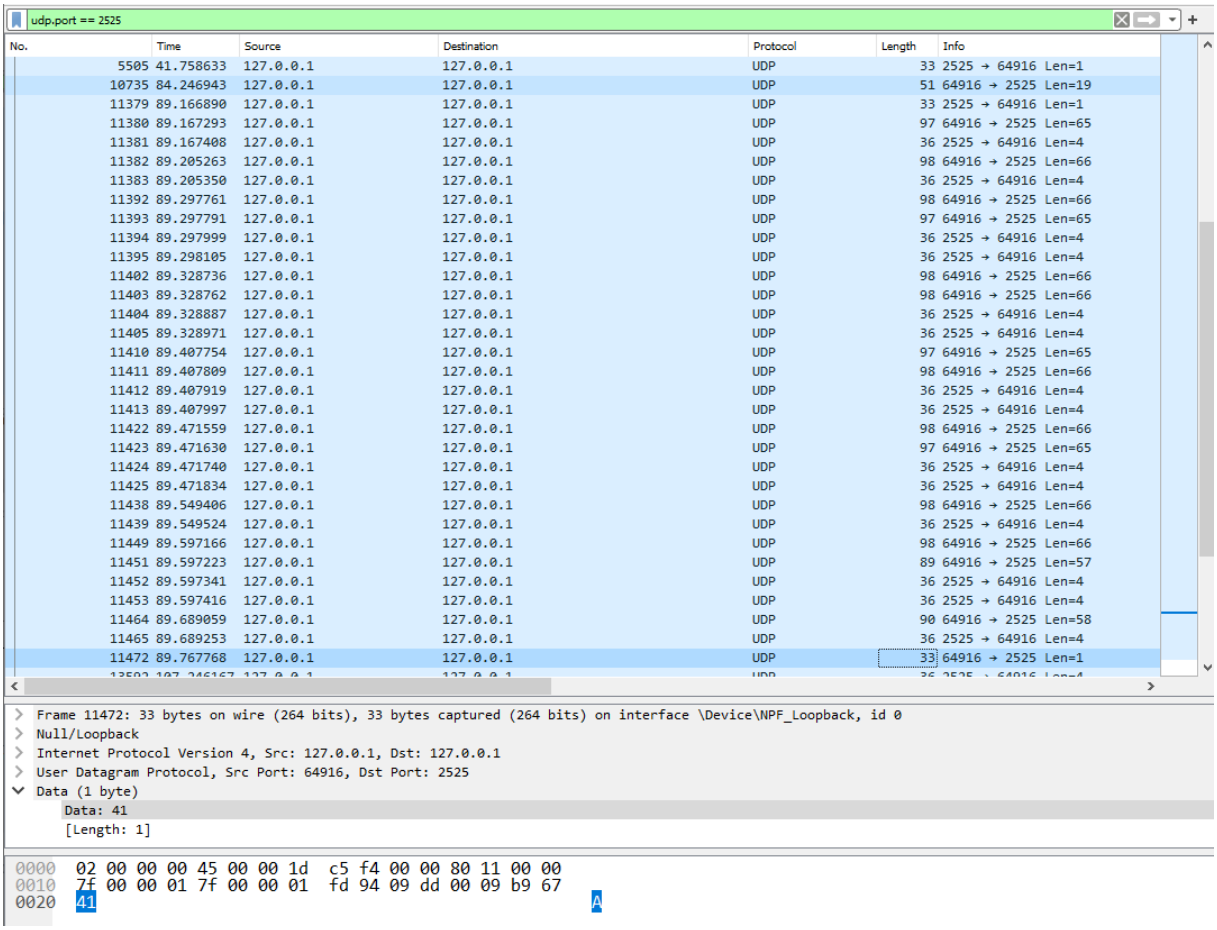
Obrázok 13 – Prijímač potvrdí prijatie SYN správy odoslaním ACK



Obrázok 14- Ukončenie spojenia poslaním FIN správy

b) Posielanie správ

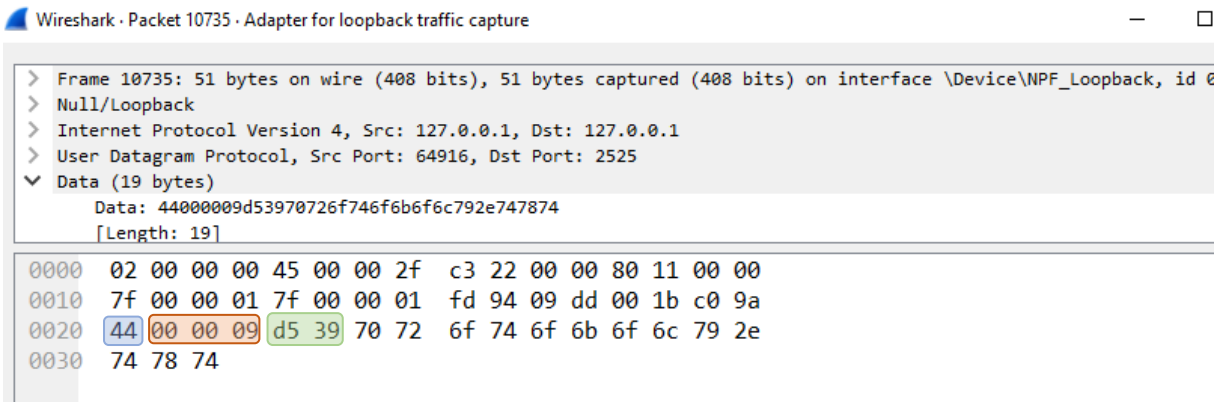
Obrázok nižšie ukazuje celý priebeh prenosu jedného súboru. Prenos začne poslaním inf. Správy o prenose ako je zobrazené v obrázku 16 , jej potvrdením a následným odosielaním jednotlivých fragmentov.



Obrázok 15- zobrazenie celého prenosu súboru

i) Poslanie inf. Správy o prenose dát

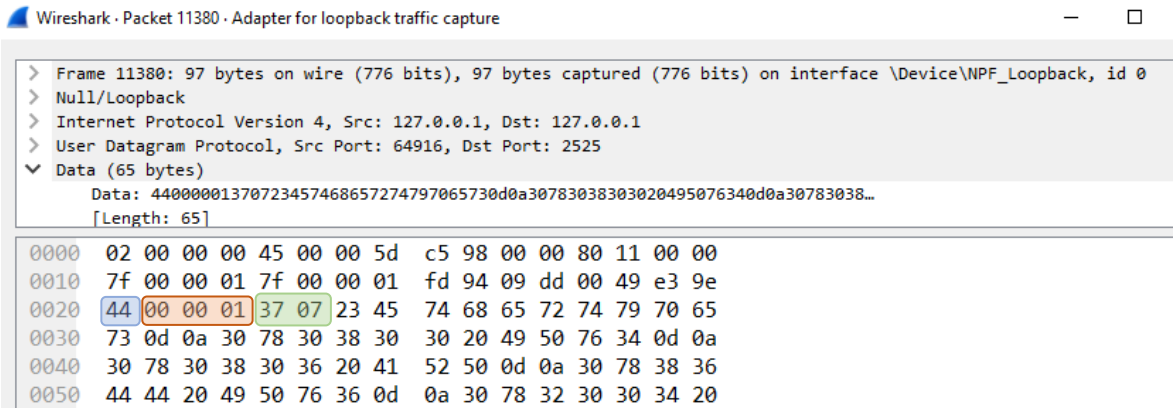
Na obrázku nižšie je zachytená inf. Správa o prichádzajúcom prenose. Prvý byte je typ správy, ďalšie 3 počet fragmentov, potom crc hodnota a nasleduje názov súboru.



Obrázok 16- Odoslanie inf. Správy o prenose

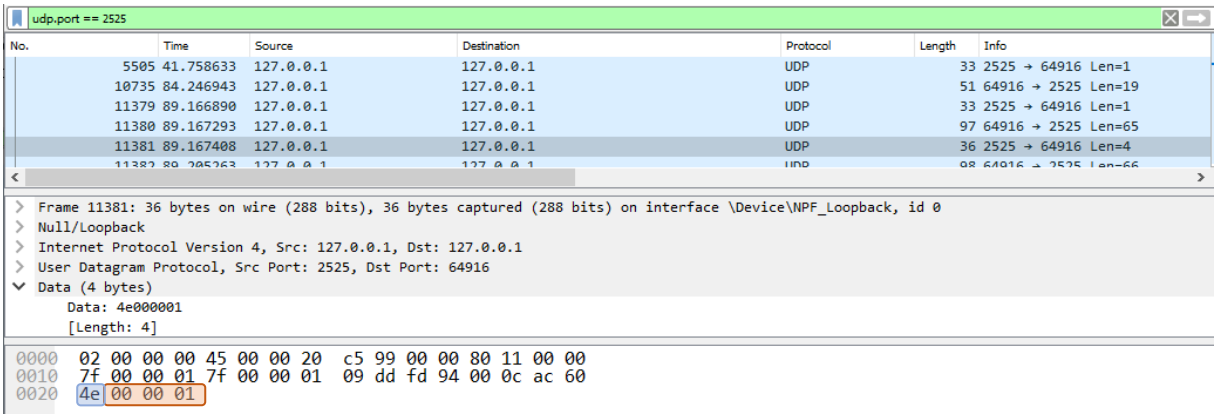
ii) Posielanie jednotlivých fragmentov

Následne sú postupne posielané fragmenty. Jednotlivé časti hlavičky sú opäť vyznačené farebne.



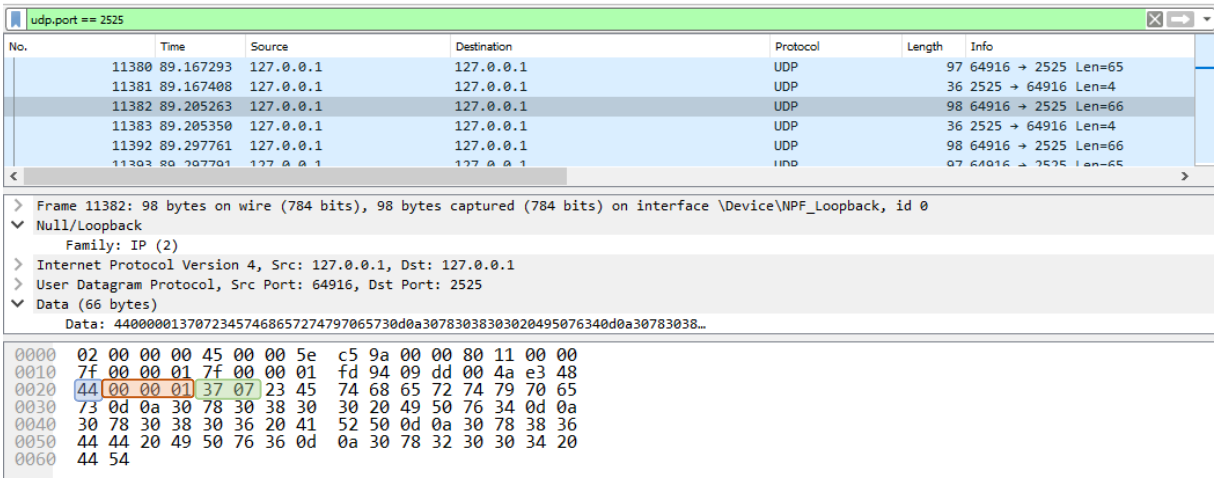
Obrázok 17 – Odoslanie 1. fragmentu

1.fragment bol poškodený, tak server odošle NACK pre daný fragment.



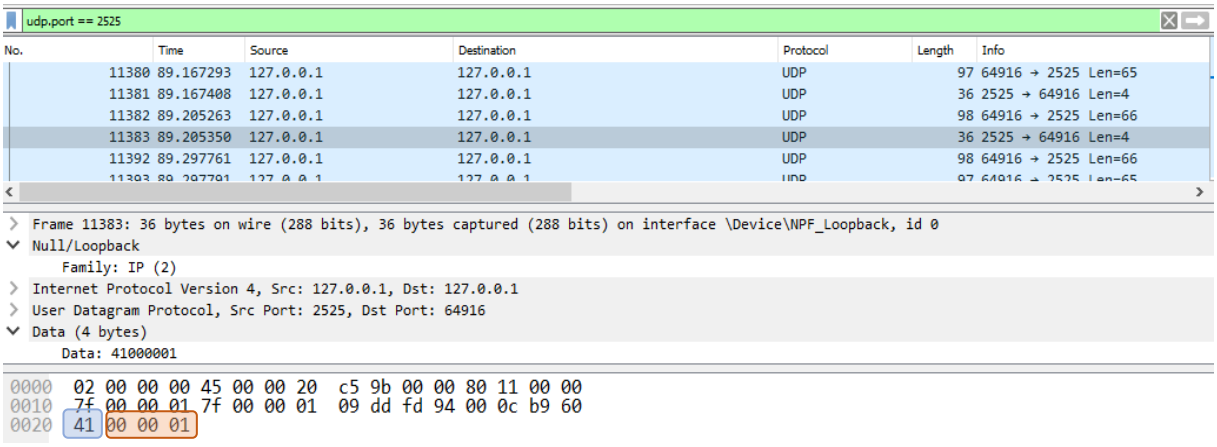
Obrázok 18 – NACK pre 1.fragment

Klient neskôr znovu odošle daný fragment ktorý bol poškodený počas prenosu.



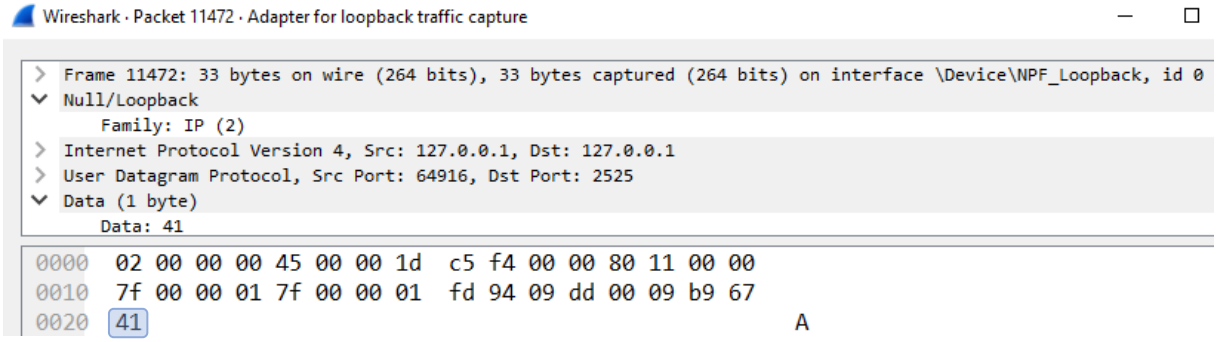
Obrázok 19 – Znovu poslanie 1.fragmentu

Fragment bol doručený v poriadku. Server odošle ACK pre daný fragment.

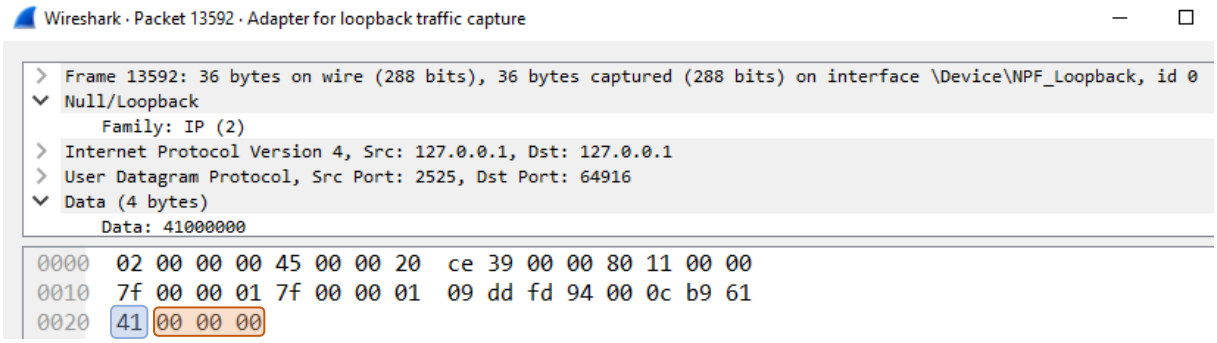


Obrázok 20 – ACK pre 1.fragment

iii) Poslanie potvrdzujúcej správy o prenose všetkých fragmentov

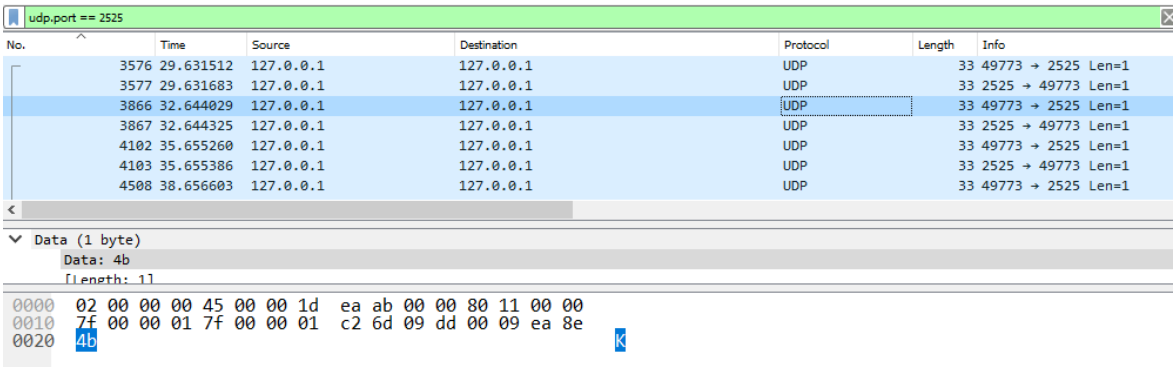


Obrázok 21 – odoslanie ACK správy od klienta, že boli doručené všetky fragmenty

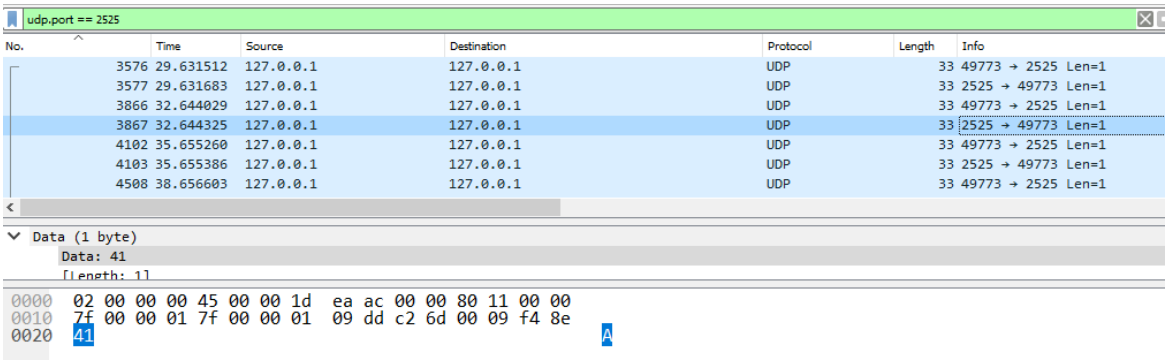


Obrázok 22 - Potvrdenie servera o doručení celého súboru

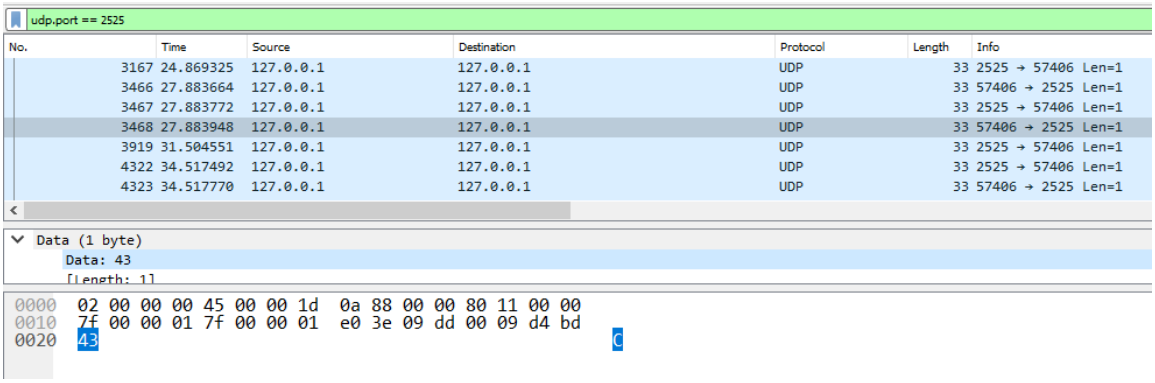
c) Keep-alive správy, výmena rolí



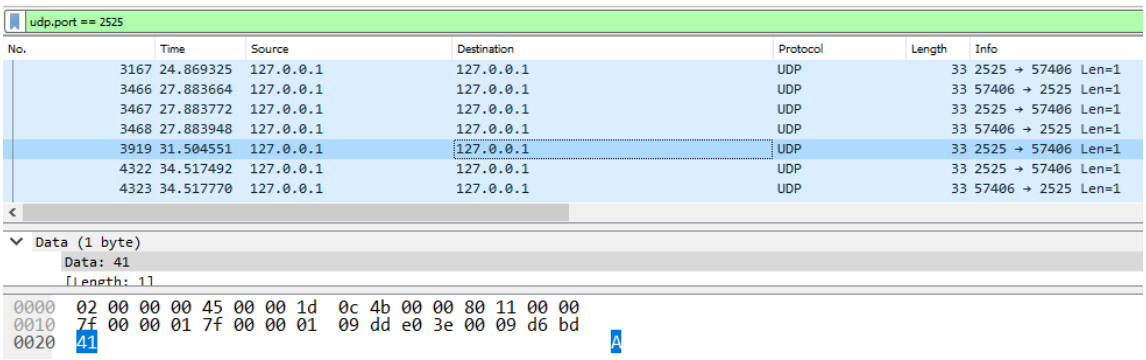
Obrázok 23- Klient odošle keep-alive správu



Obrázok 24- Server potvrdí doručenie keep-alive správy odoslaním ACK správy



Obrázok 25- Poslanie žiadosti o výmenu rolí



Obrázok 26 – Akceptovanie žiadosti o výmenu rolí

7. Zhodnotenie a záver

Toto 2. zadanie mi prišlo omnoho zaujímavejšie ako prvé, pretože sme mali sami niečo navrhnúť a následne zrealizovať a taktiež výsledný program má pre mňa väčšiu hodnotu. Vďaka nemu sa mi ozrejmili veci ohľadom princípu fungovania odosielania dát medzi zariadeniami. Naučil som sa ako zrealizovať takýto prenos pomocou jazyku python. Taktiež tu bol môj prvý kontakt s Threadmi kde som ich aj reálne použil v programe. Taktiež je toto môj prvý semester čo programujem v pythone, tak mi to pomohlo sa posunúť ďalej v jeho znalosti a schopnosti v ňom programovať.

Samozrejme nezaobišlo sa to bez ťažkostí a nad písaním kódu a debugovaním som strávil mnoho nielen večerov. No keď sa spätne pozriem na svoju prácou, tak si myslím že s ňou môžem byť celkom spokojný a tie strávené hodiny určite stáli za to.