Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológii
8. December 2021

Počítačové a komunikačné siete

Komunikátor s využitím UDP protokolu

Roland Vdovják
id: 110912

Obsah

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 znovu vyžiadanie 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:
arpa/inet.h
netinet/in.h
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.

Odovzdáva sa:
1. Návrh riešenia
2. Predvedenie riešenia v súlade s prezentovaným návrhom

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.

Návrh riešenia

Štruktúra hlavičky

Typ	Veľkosť	Poradie	CRC	Dáta
1B	2B	2B	4B	...

Typ

Hodnota TYP určuje buď jeden alebo viac flagov, ktoré daný opisujú daný rámec. Flagy môžu byť:

Bajt [bin0]	Typ flagu (správy)
0000 0001	Nadviazanie spojenia
0000 0010	Prijatie fragmentu
0000 0100	Odmietnutie fragmentu
0000 1000	Keep alive
0001 0000	Ukončenie spojenia
0010 0000	Posielanie dát
0100 0000	Potvrdenie spojenia

Veľkosť

Veľkosť fragmentu, ktorý sa prenáša. Hovorí o reálnej veľkosti, ktorá sa prenáša. Keď má naša hlavička 9B, IP hlavička 20B, UDP hlavička 8B, tak veľkosť bude maximálne

$$1500 - 9 - 20 - 8 = 1463B$$

Veľkosť je stanovená práve takto, aby nebolo nutné fragmentovať na druhej vrstve modelu OSI.

Poradie

Poradie fragmentu.

CRC

Implementovali funkciu *crc32()* z knižnice *zlib*. Veľkosť návratovej hodnoty funkcie *crc32()* sú štyri bajty, preto v hlavičke zaberá práve 4B.

ARQ

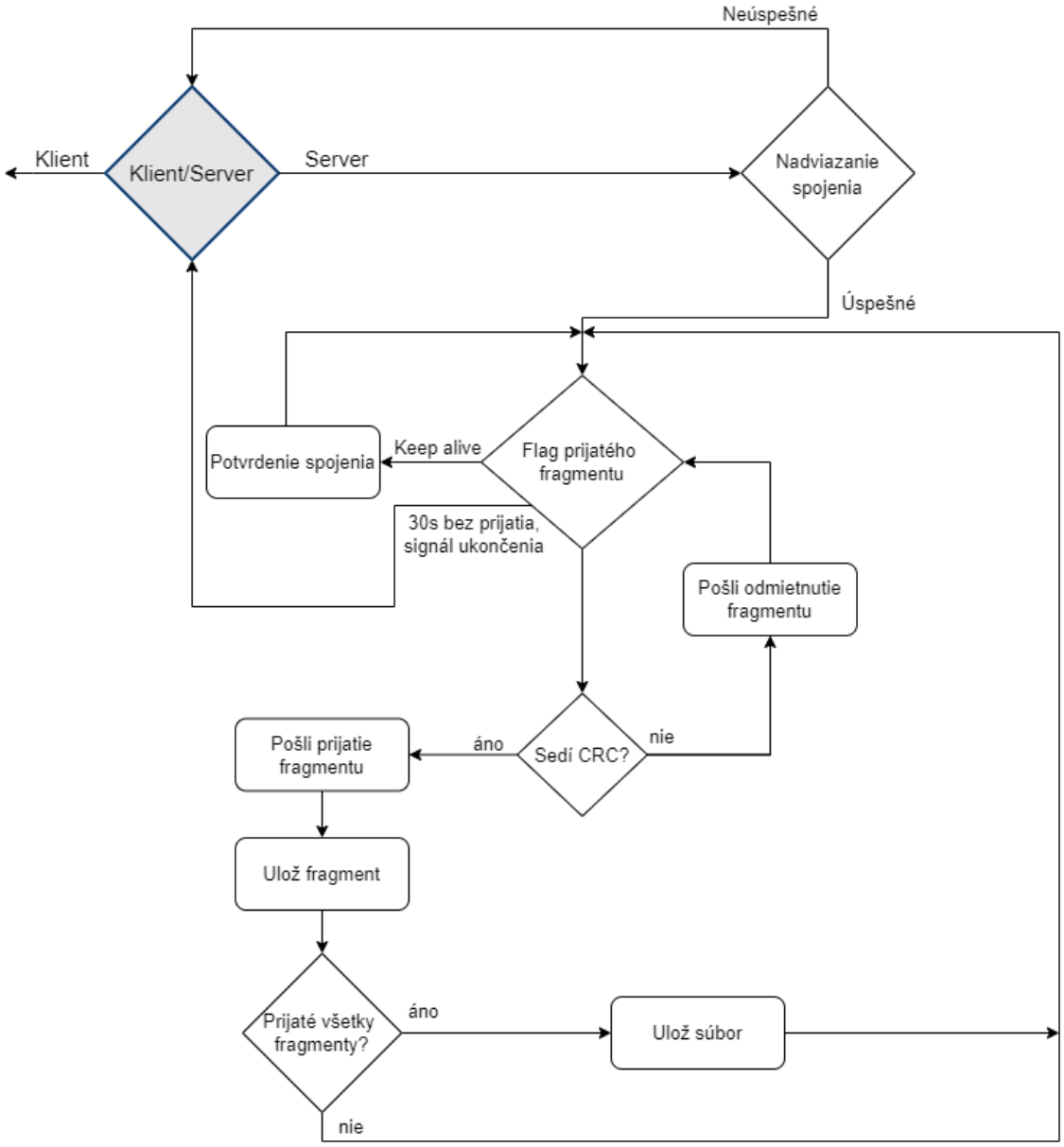
Stop and wait – táto metóda znamená, že po každom jednom fragmente čaká klient na odpoveď servera, či bol fragment prijatý alebo nie. Ak bol prijatý, tak sa pokračuje ďalším fragmentom až kým nepošleme celý súbor, ktorý odosielame. Ak nebol fragment prijatý tak klient posiela znovu rovnaký fragment, až kým nie je prijatý.

Udržiavanie spojenia

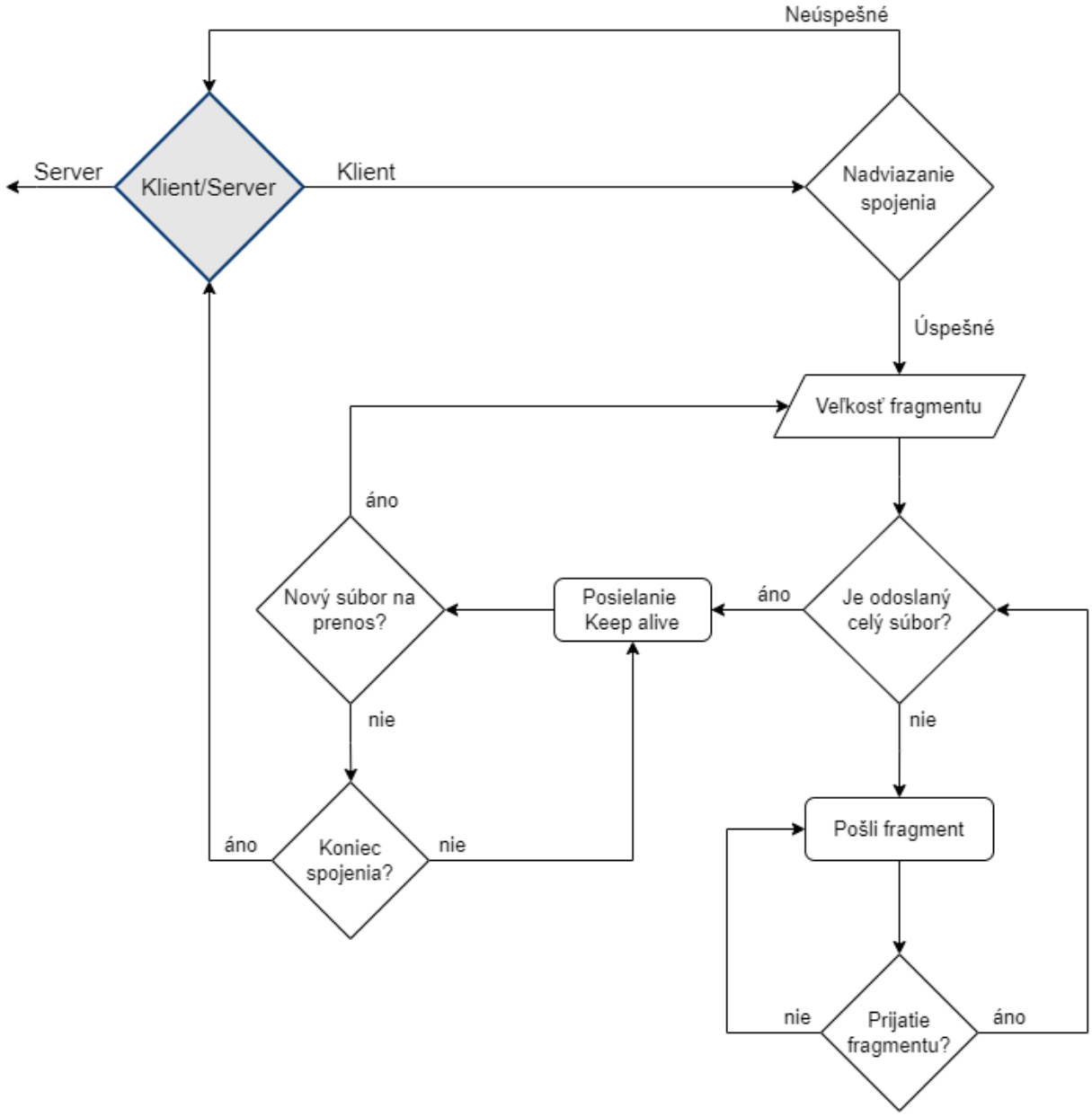
Udržiavame spojenie pomocou odosielania s typom flagu „Keep alive“. Tento fragment odosielame každých 10 sekúnd. Ak po 30 sekundách neprišiel Keep alive flag, tak sa ukončuje spojenie. Iná možnosť je ukončiť spojenie klientom po úspešnom odoslaní súboru bez potreby ďalšieho odosielania. Aby klient vedel, že je server stále dostupný, tak bude server odpovedať na správu Keep alive potvrdením spojenia, bez odpovede do niekoľkých sekúnd program ukončuje spojenie.

Diagram spracovávania

Prijímajúca strana



Odosielajúca strana



Riešenie

Zmena oproti návrhu

Zmena v hlavičke, v type (vo flagu)

Bajt [bin0]	Typ flagu (správy)
0000 0001	Nadviazanie spojenia
0000 0010	Prijatie fragmentu
0000 0100	Odmietnutie fragmentu
0000 1000	Keep alive
0001 0000	Ukončenie spojenia
0010 0000	Potvrdenie spojenia
0100 0000	Posielanie dát
0100 0001	Názov súboru
1000 0000	Posielanie textu

Okrem tejto zmeny sme nemenili návrh.

CRC

Implementácia *crc* je pomocou knižnice *zlib*. Konkrétne ide o *crc32*. Polynóm určený pre túto funkciu je:

HEX: 0x1 04 C1 1D B7 BIN: 1 0000 0100 1100 0001 0001 1101 1011 0111

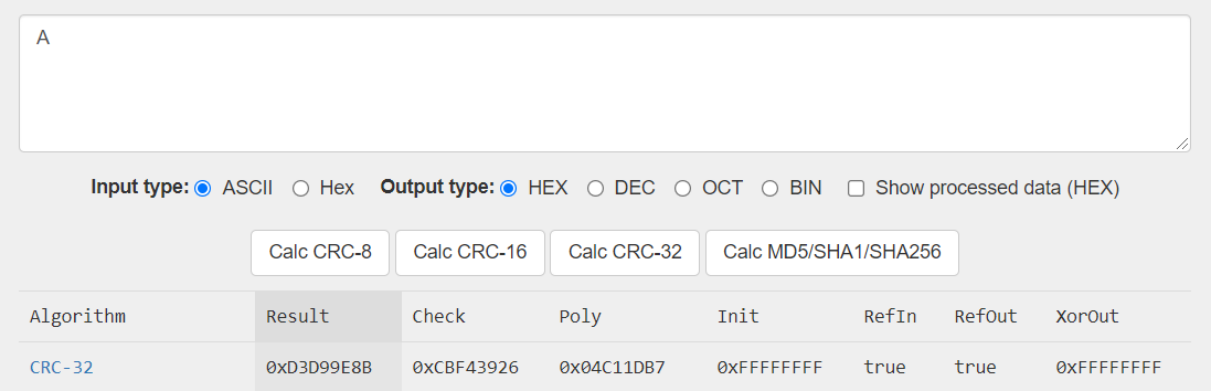
Príklad riešenia pre input „A“:

input	0100	0001								
reversed	1000	0010								
	1000	0010	0000	0000	0000	0000	0000	0000	0000	0000
0xFFFFFFFF	1111	1111	1111	1111	1111	1111	1111	1111		
XOR	0111	1101	1111	1111	1111	1111	1111	1111	0000	0000
polynom	100	0001	0011	0000	0100	0111	0110	1101	11	
XOR	0011	1100	1100	1111	1011	1000	1001	0010	1100	0000
polynom	10	0000	1001	1000	0010	0011	1011	0110	111	
XOR	0001	1100	0101	0111	1001	1011	0010	0100	0010	0000
polynom	1	0000	0100	1100	0001	0001	1101	1011	0111	
XOR	0000	1100	0001	1011	1000	1010	1111	1111	0101	0000
polynom		1000	0010	0110	0000	1000	1110	1101	1011	1
XOR		0100	0011	1101	1000	0010	0001	0010	1110	1000
polynom		100	0001	0011	0000	0100	0111	0110	1101	11
XOR		0000	0010	1110	1000	0110	0110	0100	0011	0100
0xFFFFFFFF			1111	1111	1111	1111	1111	1111	1111	1111
XOR			1101	0001	0111	1001	1001	1011	1100	1011

reversed			1101	0011	1101	1001	1001	1110	1000	1011

Tabuľka 1: Vypočítanie CRC32

Výsledok prevedený do hexadecimálneho formátu je 0xd3d99e8b. Ako kontrolu správnosti sme použili internetovú stránku (https://crccalc.com/) na výpočet crc32.



Princíp

Vstup sa premení na binárne číslo, obráti sa a pridá sa k nemu 32 núl (oranžový riadok tabuľky). Toto číslo je začiatkom XOR operácií. Prvý a posledný XOR, teda XOR z upraveného vstupu a z 32 bitového zvyšku je vypočítaný s hodnotou 0xFFFFFFFF (modré riadky). Hodnoty medzi prvým a posledným XOR-om sú počítané s polynómom. Výsledkom je 32 bitové číslo.

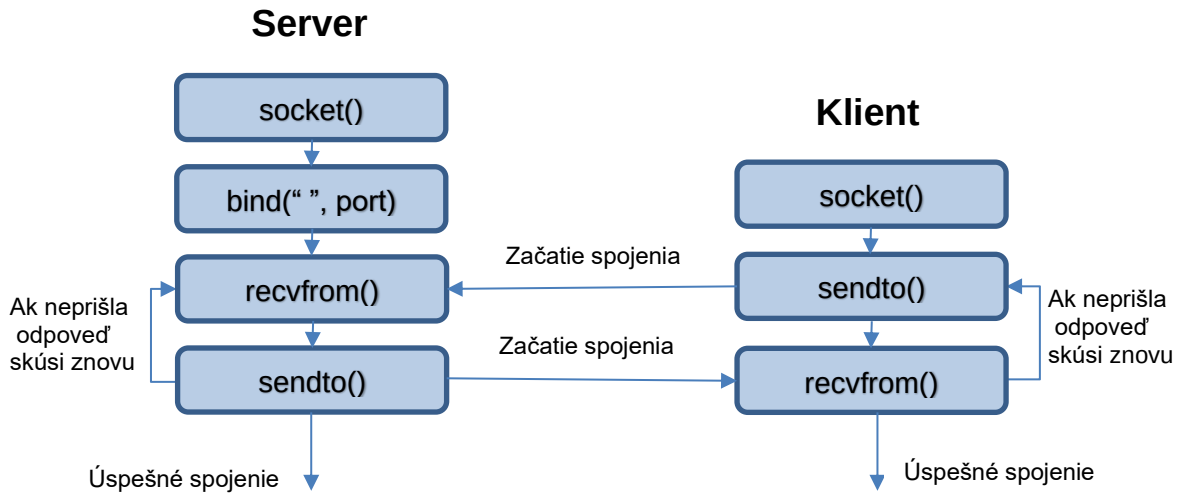
Fungovanie programu

Všetky časti programu sú dostatočne okomentované v kóde, pre prehľadnosť dokumentácie v nej spomenieme len dôležité časti kódu a logiku programu.

Spojenie Klient-Server

Používateľ by si mal najskôr vybrať rolu server, zadať port na ktorom bude počúvať (ak sa tak nestane, pripojenie nebude korektné a na strane klienta bude potrebné zadať znovu IP adresu a port). Potom na druhom zariadení používateľ vyberie rolu klient, zadá IP adresu servera

a port.



Obrázok 1: Počietočné spojenie Klient-Server

Na signál dobrého spojenia musí klient poslať prázdnu hlavičku len s flagom začiatku spojenia a server mu vráti rovnakú prázdnu hlavičku len s flagom. Server ďalej počúva prichádzajúce rámce a správa sa podľa diagramu. Každé rozhodnutie v diagrame reprezentuje číslo flagu v hlavičke. Keďže nie je pri posielaní textu alebo dát špecifikovaný flag posledného fragmentu, ktorý klient odosiela, server zisťuje, či je odoslané všetko vďaka kombinácií po sebe idúcich fragmentov s flagom Posielam text alebo Posielam dáta a Keep Alive.

Klient

Klient sa pripája na server pomocou zadania IP serveru a portu, požívateľ zadá po výbere role klient. Po úspešnom spojení sa začína posielať hlavička s flagom Keep alive. Po spojení má na výber z troch možností:

- Poslať súbor [1]
  - Program si vypýta cestu ku súboru a veľkosť jedného fragmentu, pošle počiatočný rámec s meno súboru. Následne posiela fragmenty súboru. Po odoslaní vypíše informáciu o poslaní.
- Poslať text [2]
  - Program očakáva textový input. Implementovaný je multiline string. Na ukončenie inputu textu je potrebné zadať na riadok len znak „$“. Potom si program vypýta veľkosť fragmentu a posiela text.
- Vymeniť rolu/Ukončiť [3]
  - Zastaví vlákno s KeepAlive, pošle rámec s flagom na ukončenie spojenia, ukončí spojenie a presmeruje používateľa na výber Server/Klient/Ukončiť.

Server

Po vybratí role server je potrebné zadať port na ktorom bude server počúvať akonáhle sa na tento port pripojí klient. Po spojení s klientom čaká na input ktorý rozhoduje o:

- Pokračovať
    - Server je pripravený na počúvanie
- Vymeniť rolu/Ukončiť [0]
    - Odošle rámec s s flagom na ukončenie spojenia, zatvorí socket a presmeruje používateľa na výber Server/Klient/Ukončiť.
- Pokračovať ako server [1]
    - Čaká na pripojenie klienta, port zostáva rovnaký

Ak server pokračuje, je pripravený prijímať rámce. Po každom prijatom rámci vyvodí nejakú akciu. Po prenesení súboru alebo textu má používateľ na výber možnosti spomínané vyššie.

Keep Alive

Implementácia posielania rámcov s keep alive je v samostatnom vlákne v klientovi. Na strane servera sú prijímané rámce s keep alive rovnako ako všetky ostatné. Ak na server príde keep alive, server odpovedá „Connection confirmed“. Ak klient nedostane odpoveď do desiatich sekúnd, pošle znovu. Toto opakuje trikrát. Ak po tridsiatich sekundách neprišla odpoveď, klient vyhodnotí, že server neodpovedá a naviguje klienta na ukončenie.

Možnosti ukončenia spojenia

Program sa môže ukončiť viacerými spôsobmi, nie všetky sú korektné, ale sú ošetrené.

- Ukončenie zo strany klienta- odoslaný „Connection end“ flagu
- Ukončenie zo strany servera- odoslaný „Connnection end“ flagu
- Odpojenie klienta- bez odoslania „Connection end“ flagu
    - Program s klientom nečakane skončí, odpojí sa zo siete, prestane posielať Keep Alive správy.
- Odpojenie servera- bez odoslaný „Connection end“ flagu
    - Program so serverom nečakane skončí, odpojí sa zo siete.

Ukončovací flag má hodnotu 16 a je pomenovaný ako „Connection end“, ak tento pragment prijme server alebo klient, používateľ dostane o tom informáciu a je navigovaný ako pokračovať.

Posielanie

Na posielanie je určená jedna funkcia, ktorá ako parameter berie kľúč, ktorý hovorí o tom, či sa posiela text alebo súbor. Hlavným rozdielom medzi posielaním textu a súboru je tvorba počiatočných premenných. Pri posielaní súboru je potrebné zistiť zo vstupu o aký súbor sa

jedná, zistiť jeho veľkosť, dáta uložiť do premennej vo forme bajtov. Ďalej názov bez cesty, ktorý sa rovno pošle na server a stanoviť flag fragmentu, ktorý sa bude odosielať pri posielaní dát.

Ukážka kódu:

```
# Posiela sa subor
if f_type == '1':
    f_path = input("Zadaj cestu k suboru.\n")
    f = open(f_path, 'rb')
    f_name = (os.path.basename(f_path)).encode('utf-8')

    # odoslanie mena suboru
    h_type = 65
    h_crc = crc(f_name)
    h_frag_num = 0
    h_size = len(f_name)
    frag_full = h_type.to_bytes(1, byteorder='big') + h_size.to_bytes(2,
byteorder='big') + h_frag_num.to_bytes(2, byteorder='big') +
h_crc.to_bytes(4, byteorder='big') + f_name
    print("Posielam meno suboru")
    client_s.sendto(frag_full, server_address)  # Odoslanie

    size = os.path.getsize(f_path)
    data = f.read()
    h_type = 64

# Posiela sa textova sprava
if f_type == '2':
    print("Zadaj text spravy.\n")
    text_msg = load_text_msg()
    data = text_msg.encode('utf-8')
    size = len(data)
    h_type = 128
```

Premenné s predponou „*h_*“ hovoria o hlavičke. *F_type* je typ súboru- teda text alebo súbor.

Po tejto počiatočnej časti program zisťuje veľkosť fragmentu od používateľa, táto veľkosť je obmedzená. Môže vyť v intervale *<1, 1463>* a zároveň počet fragmentov musí byť menší ako 65535. Dôvodom je veľkosť hodnoty Poradie v hlavičke, táto veľkosť je ohraničená na 2B. Ak by po zadaní veľkosti fragmentu bol počet fragmentov väčší ako 65534, tak musí používateľ zadať veľkosť znovu, používateľ je inštruovaný minimálnou potrebnou veľkosťou.

Po tomto vchádza program do cyklu odosielania, každým opakovaním sa vytvorí hlavička a odreže potrebná časť dát, všetky údaje sa spoja a pošlú. Po poslaní očakávajú fragment hovoriaci o tom, či fragment prešiel bez problémov alebo bol odmietnutý. Podľa toho pokračuje ďalej alebo znovu odosiela pôvodne odmietnutý fragment.

Po odoslaní celého súboru alebo textu vypíše, že súbor alebo text bol odoslaný a jeho veľkosť.

Chyba v dátovej časti

Z dôvodu nutnosti posielať chybu v dátovej Časti sme sa rozhodli posielať prvý resp. nultý bajt rámca ako samé nuly. Pôvodný bajt si odložíme do premennej *original_byte*. Výber toho, ktoré rámce budú týmto ovplyvnené nastavujeme v globálnej premennej. Posielame chybných pár rámcov v prvej desiatke odoslaných. Potenciálne vylepšenie sú jednoduché, len meníme obsah globálnej premennej- poľa.

Po odoslaní chyby príde na príjem klienta flag rámca „Fragment redjected“. Po zaznamenaní tohto flagu vrátime hodnotu bajtu z premennej *original_byte* do nultého bajtu dátovej časti.

Nultý index je stanovený z dôvodu posielania minimálne jedného bajtu pri posielaní súboru alebo textu.

Ukážka kódu:

```
# Posielanie chyb
if sended and h_frag_num in wrong_fragments:
    original_byte = frag_data[0]
    frag_data[0] = 0
```

Premenná *frag_data* obsahuje fragment dát daného rámca. *Sended* hovorí o tom, či bol predchádzajúci fragment odoslaný. *Wrong_fragments* je globálna premenná typu poľa. A *h_frag_num* je momentálne číslo rámca ktorý sa posiela.

Prijímanie

Server prijíma rámce a podľa hodnoty flagu sa rozhoduje, akú akciu vykoná. Samotný try exept blok pozostáva z viacerých if blokov.

Ukážka kódu:

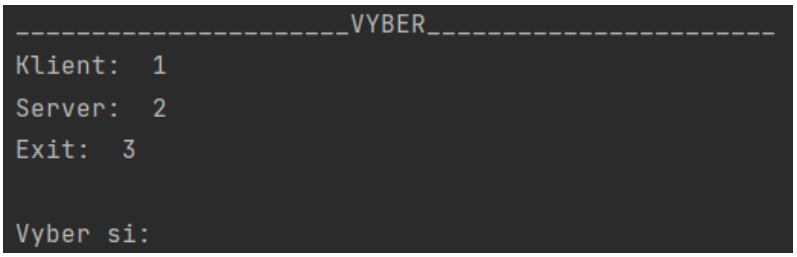
```
# Vypis flagu
if type_int != 8:
    ...

# Ukoncenie spojenia
if type_int == 16:
    ...

# KEEP ALIVE + odpoved
if type_int == 8:
    ...

# Meno suboru
if type_int == 65:
    ...

# Kontrola crc
if type_int == 64 or type_int == 128:
    ...
```

```
# Odoslanie celeho suboru
if type_int == 8 and (last_type_int == 64 or last_type_int == 128):
    ...
```

Užívateľské rozhranie

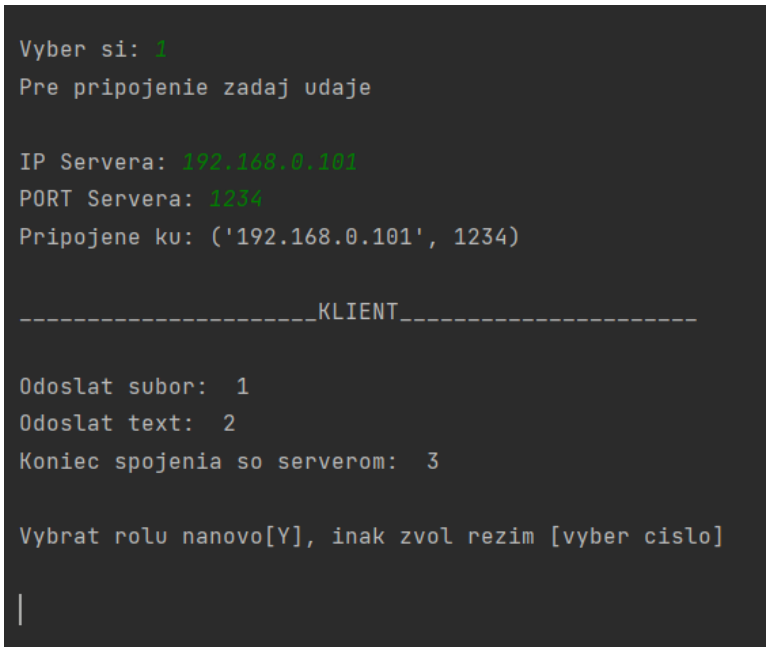
Používateľ pracuje s programom cez terminál. Zadáva na vstup rôzne čísla alebo údaje, podľa ktorých program rozhoduje, čo bude robiť.

Menu



Obrázok 2: Hlavné menu

Role



Obrázok 3: Výber role Klient

```
Vyber si: 2
_____________________SERVER_____________________
Cakanie na klienta...
Zadaj port: 1234
Pripojene ku: ('192.168.0.104', 63592)


Pokracovat [Enter], Vymena roli alebo koniec programu [0], Spustenie servera na novo [1]


Server je pripraveny.
```

Obrázok 4: Výber role Server

Posielanie súboru

```
1
Zadaj cestu k suboru.
foto.jpg
Posielam meno suboru
Velkost jedneho fragmentu: 1463
Posielam ramec cislo: 1
Flag ramca: Fragment recieved
Posielam ramec cislo: 2
Flag ramca: Fragment redjected
Posielam znovu
Posielam ramec cislo: 2
Flag ramca: Fragment recieved
```

Obrázok 5: Posielanie súboru na strane Klienta

```
Posielam ramec cislo: 2768
Flag ramca: Fragment recieved
Posielam ramec cislo: 2769
Flag ramca: Fragment recieved
Poslany subor: C:\Users\Roland\PycharmProjects\Komunikator\venv\foto.jpg
Velkost: 4050422 B
_____________________KLIENT_____________________

Odoslat subor:  1
Odoslat text:  2
Koniec spojenia so serverom:  3

Vybrat rolu nanovo[Y], inak zvol rezim [vyber cislo]
```

Obrázok 6: Odoslanie všetkých rámcov na strane Klienta

```
Flag fragmentu:  Sending data
Fragment cislo 2769 o velkosti 838 sedi


Sprava bola odoslana cela.
Zadaj cestu, kde sa ulozi subor. Pre domovsky priecinok [Enter]

Ulozene v C:\Users\rolan\PycharmProjects\Komunikator\venv\foto.jpg
Pocet fragmentov: 2769
Velkost: 4050422 B

Pokracovat [Enter], Vymena roli alebo koniec programu [0], Spustenie servera na novo [1]


Server je pripraveny.
```

Obrázok 7: Odoslanie všetkých rámcov na strane Servera

Posielanie textu

```
2
Zadaj text spravy.

Posielam text ktory bude v dokumentacii.
Je iplementovany multiline string takze ide aj toto

:)

Ukoncovaci znak je $ na prazdnom riadku.
Dekódované a Kódované je to pomocou UTF-8, takže ide aj interpunkcia.
$
Velkost jedneho fragmentu: 20
Posielam ramec cislo: 1
Flag ramca: Fragment recieved
Posielam ramec cislo: 2
Flag ramca: Fragment redjected
Posielam znovu
Posielam ramec cislo: 2
Flag ramca: Fragment recieved
```

Obrázok 8: Posielanie textu ma strane Klienta

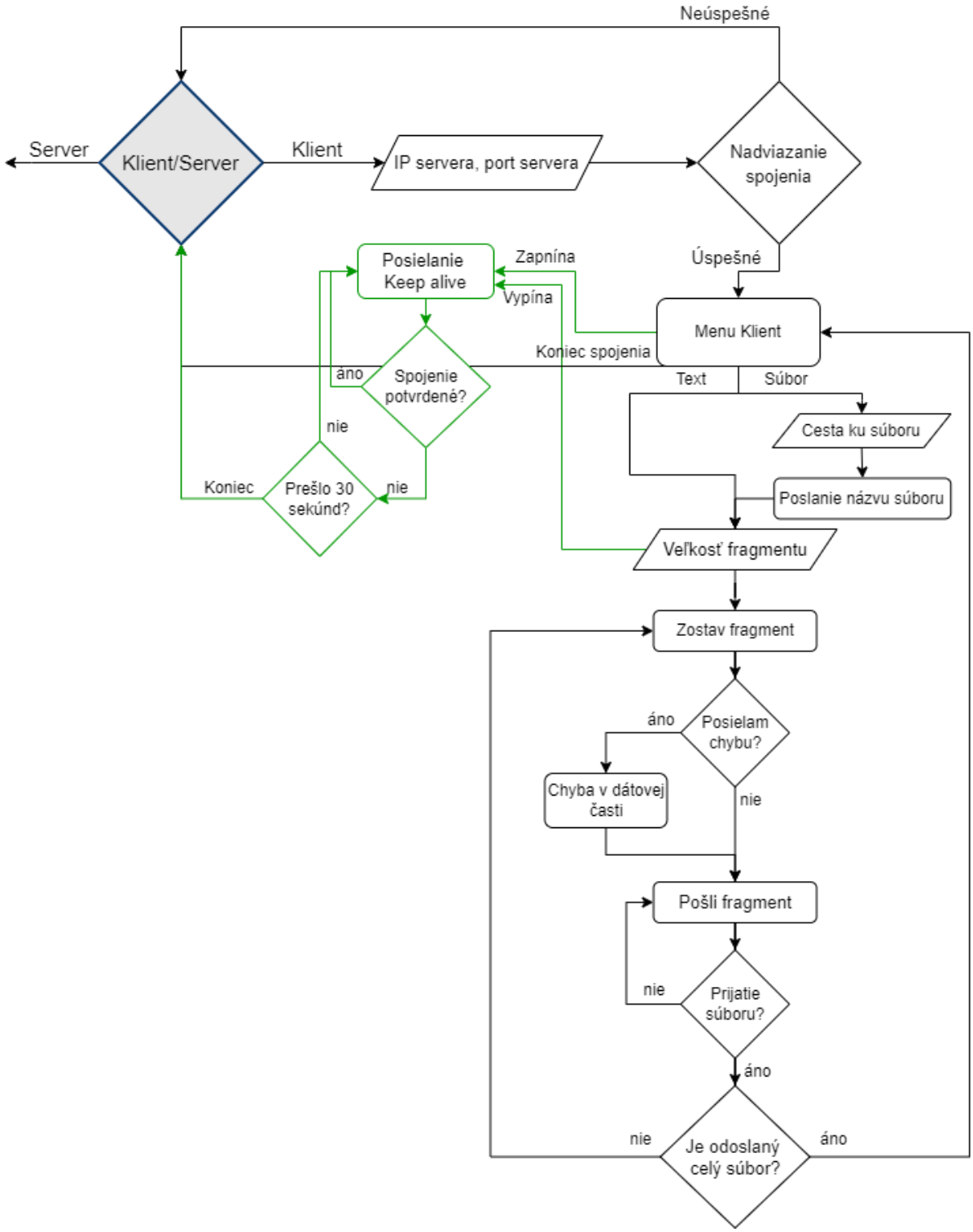
```
Flag fragmentu:  Sending text
Fragment cislo 11 o velkosti 20 sedi

Flag fragmentu:  Sending text
Fragment cislo 12 o velkosti 3 sedi


Sprava bola odoslana cela.
Text spravy:

Posielam text ktory bude v dokumentacii.
Je iplementovany multiline string takze ide aj toto

:)

Ukoncovaci znak je $ na prazdnom riadku.
Dekódované a Kódované je to pomocou UTF-8, takže ide aj interpunkcia.


Pokracovat [Enter], Vymena roli alebo koniec programu [0], Spustenie servera na novo [1]
```

Obrázok 9: Posielanie textu na strane Servera

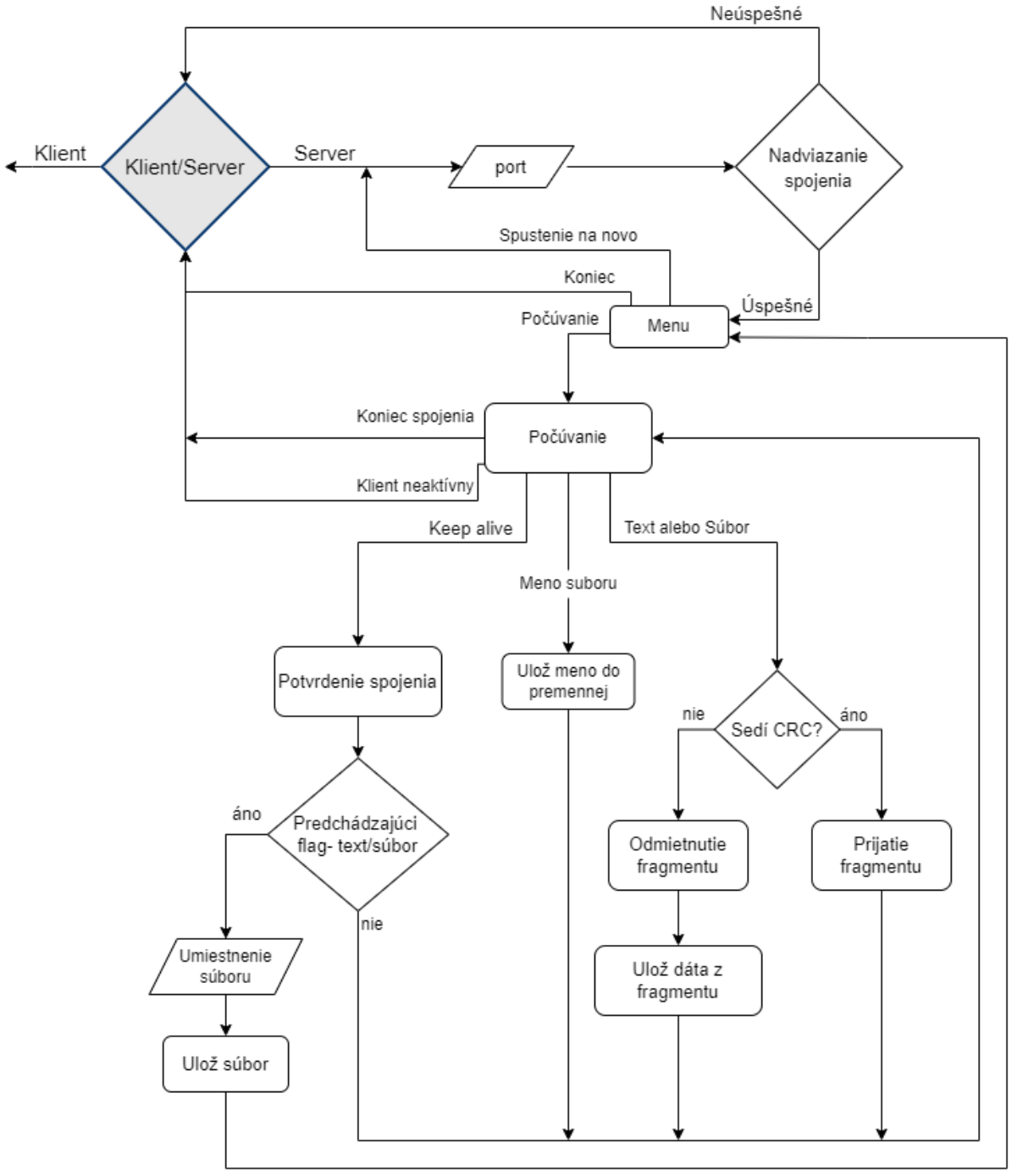
Diagramy

Diagramy zostali oproti návrhu logicky rovnaké, no pridali sme niekoľko potrebných funkcií, vstupov, výstupov. Pri konzultácií sme dostali podnet na vylepšenie a spresnenie diagramov. Vylepšené diagramy:

Zelená časť diagramu reprezentuje samostatné vlákno.



Obrázok 10: Diagram Klient



Obrázok 11: Diagram Server

Wireshark

Pri testovaní a debugovaní programu sme používali aj program Wireshark. Posielali sme rámce cez router na druhé zariadenie. Záznam z wiresharku sme vyfiltrovali aby nám ukazoval komunikáciu medzi IP adresami týchto zariadení. Záznam vyzerá takto. Ako legendu ku farbám používame rozšírenú tabuľku s flagmi.

No.	Time	Source	Destination	Protocol	Length	Info
1299	48.088780	192.168.0.104	192.168.0.101	UDP	51	61300 → 1234 Len=9
1300	48.107965	192.168.0.101	192.168.0.104	UDP	60	1234 → 61300 Len=9
1331	49.117923	192.168.0.104	192.168.0.101	UDP	51	61300 → 1234 Len=9
1513	56.095407	192.168.0.101	192.168.0.104	UDP	60	1234 → 61300 Len=9
1650	61.103628	192.168.0.104	192.168.0.101	UDP	51	61300 → 1234 Len=9
1651	61.141457	192.168.0.101	192.168.0.104	UDP	60	1234 → 61300 Len=9
1805	66.143111	192.168.0.104	192.168.0.101	UDP	51	61300 → 1234 Len=9
1806	66.159390	192.168.0.101	192.168.0.104	UDP	60	1234 → 61300 Len=9
1923	71.166417	192.168.0.104	192.168.0.101	UDP	51	61300 → 1234 Len=9
1927	71.279517	192.168.0.101	192.168.0.104	UDP	60	1234 → 61300 Len=9
2041	76.280256	192.168.0.104	192.168.0.101	UDP	51	61300 → 1234 Len=9
2042	76.297147	192.168.0.101	192.168.0.104	UDP	60	1234 → 61300 Len=9
2091	78.221151	192.168.0.104	192.168.0.101	UDP	71	61300 → 1234 Len=29
2093	78.253966	192.168.0.101	192.168.0.104	UDP	60	1234 → 61300 Len=9
2094	78.254064	192.168.0.104	192.168.0.101	UDP	71	61300 → 1234 Len=29
2095	78.256084	192.168.0.101	192.168.0.104	UDP	60	1234 → 61300 Len=9
2096	78.256140	192.168.0.104	192.168.0.101	UDP	71	61300 → 1234 Len=29
2097	78.258252	192.168.0.101	192.168.0.104	UDP	60	1234 → 61300 Len=9
2098	78.258303	192.168.0.104	192.168.0.101	UDP	58	61300 → 1234 Len=16
2099	78.260407	192.168.0.101	192.168.0.104	UDP	60	1234 → 61300 Len=9
2100	78.260458	192.168.0.104	192.168.0.101	UDP	58	61300 → 1234 Len=16
2101	78.262478	192.168.0.101	192.168.0.104	UDP	60	1234 → 61300 Len=9
2102	78.262520	192.168.0.104	192.168.0.101	UDP	51	61300 → 1234 Len=9
2104	78.264849	192.168.0.101	192.168.0.104	UDP	60	1234 → 61300 Len=9
2173	81.299840	192.168.0.104	192.168.0.101	UDP	51	61300 → 1234 Len=9
2216	83.073649	192.168.0.101	192.168.0.104	UDP	60	1234 → 61300 Len=9
2260	84.486079	192.168.0.104	192.168.0.101	UDP	51	61300 → 1234 Len=9
2449	91.665949	192.168.0.101	192.168.0.104	UDP	60	1234 → 61300 Len=9

Obrázok 12: Wireshark záznam- prenášaný text

Bajt [bin0]	Typ flagu (správy)	Farba
0000 0001	Nadviazanie spojenia	
0000 0010	Prijatie fragmentu	
0000 0100	Odmietnutie fragmentu	
0000 1000	Keep alive	
0001 0000	Ukončenie spojenia	
0010 0000	Potvrdenie spojenia	
0100 0000	Posielanie dát	
0100 0001	Názov súboru	
1000 0000	Posielanie textu	

No.	Time	Source	Destination	Protocol	Length	Info
612	23.977032	192.168.0.104	192.168.0.101	UDP	51	61389 → 1234 Len=9
633	24.056073	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
670	25.070766	192.168.0.104	192.168.0.101	UDP	51	61389 → 1234 Len=9
941	35.073434	192.168.0.104	192.168.0.101	UDP	51	61389 → 1234 Len=9
1076	38.232569	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1077	38.232569	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1203	43.244458	192.168.0.104	192.168.0.101	UDP	51	61389 → 1234 Len=9
1205	43.307728	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1221	43.943382	192.168.0.104	192.168.0.101	UDP	59	61389 → 1234 Len=17
1308	46.960427	192.168.0.104	192.168.0.101	UDP	1514	61389 → 1234 Len=1472
1312	46.995357	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1313	46.995483	192.168.0.104	192.168.0.101	UDP	1514	61389 → 1234 Len=1472
1314	46.998089	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1315	46.998190	192.168.0.104	192.168.0.101	UDP	1514	61389 → 1234 Len=1472
1316	47.000656	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1317	47.000731	192.168.0.104	192.168.0.101	UDP	1514	61389 → 1234 Len=1472
1318	47.003588	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1319	47.003677	192.168.0.104	192.168.0.101	UDP	1514	61389 → 1234 Len=1472
1320	47.006357	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1321	47.006445	192.168.0.104	192.168.0.101	UDP	1514	61389 → 1234 Len=1472
1323	47.008832	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1324	47.008920	192.168.0.104	192.168.0.101	UDP	1514	61389 → 1234 Len=1472
1325	47.014857	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1326	47.015214	192.168.0.104	192.168.0.101	UDP	1514	61389 → 1234 Len=1472
1327	47.017549	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1328	47.017908	192.168.0.104	192.168.0.101	UDP	1514	61389 → 1234 Len=1472
1330	47.020140	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1331	47.020474	192.168.0.104	192.168.0.101	UDP	1514	61389 → 1234 Len=1472
1332	47.022747	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1333	47.023084	192.168.0.104	192.168.0.101	UDP	1514	61389 → 1234 Len=1472
1334	47.025577	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1335	47.025916	192.168.0.104	192.168.0.101	UDP	1514	61389 → 1234 Len=1472
1336	47.028898	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1337	47.029222	192.168.0.104	192.168.0.101	UDP	1514	61389 → 1234 Len=1472
1338	47.031962	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1339	47.032295	192.168.0.104	192.168.0.101	UDP	1514	61389 → 1234 Len=1472
1340	47.035157	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1341	47.035461	192.168.0.104	192.168.0.101	UDP	1514	61389 → 1234 Len=1472
1342	47.039419	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9
1343	47.039778	192.168.0.104	192.168.0.101	UDP	1514	61389 → 1234 Len=1472
1344	47.044885	192.168.0.101	192.168.0.104	UDP	60	1234 → 61389 Len=9

Obrázok 13: Wireshark záznam- prenášaný súbor