

Martin Hric  
ID: 111696

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE  
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLOGIÍ

## UDP KOMUNIKÁTOR

Vypracoval: Martin Hric

Cvičiaci: Ing. Matej Janeba

Ak. rok: 2021/2022

## OBSAH

NÁVRH .....	3
Implementácia .....	3
Používateľské rozhranie: .....	3
Server .....	3
Klient .....	4
Hlavička .....	5
Typy správ .....	5
CRC .....	5
DIAGRAM .....	5
ZMENY OPROTI NÁVRHU .....	7
ANALÝZA .....	8
IMPLEMENTÁCIA .....	9
FUNKCIE .....	10
Používateľské prostredie .....	11
WIRESHARK .....	13
ZÁVER .....	14

## NÁVRH

### Implementácia

Program bude implementovaný v jazyku *Python 3*, s použitím knižníc *socket*, *queue*, *math*, *os*, *libscrc*, *threading*.

Spojenie sa nadviaže medzi mojim PC, ktorý bude vystupovať ako server a laptop ako client. Aj PC aj laptop disponujú operačným systémom Windows.

#### Používateľské rozhranie:

Na začiatku stále si vypýtam od užívateľa potrebné údaje, pokiaľ sú chybné, znova vypýtam od začiatku údaje.

Program sa spustí aj keď nevyplní správnu IP a port, bohužiaľ ale komunikácia neprebehne.

#### Server

Pri spustení programu na PC ako server, užívateľ vyberie port, na ktorom bude server figurovať. Následne čaká na inicializačnú správu. Ak neprichádza, server stále počúva a čaká.

Ak príjde inicializačný fragment, odošle klientovi naspäť správu potvrdzujúcu spojenie.

Server bude kontrolovať prijaté fragment po desiatkach, čiže ak začnú prichádzať dáta, až po 10 fragmentoch ich všetky skontroluje pomocou CRC (Cyclic redundancy check) a skontroluje ak dorazí posledný fragment (ak fragmentov je menej ako 10 alebo počet fragmentov mod 10 != 0)

Keď narazí na nejaký chybný fragment, informuje o tom klienta vypísaním, v ktorej desiatke sa nachádza a taktiež aj jeho poradie, inak pošle správu o úspešne prenesených dátach.

Po obdržaní všetkých fragmentov, ktoré boli úspešné, tieto fragmenty budú spojené a uložené ako súbor.

Čiže ARQ je Stop and Wait , po desiatkách , aby to časovo nebolo príliš obtiažne.

## Klient

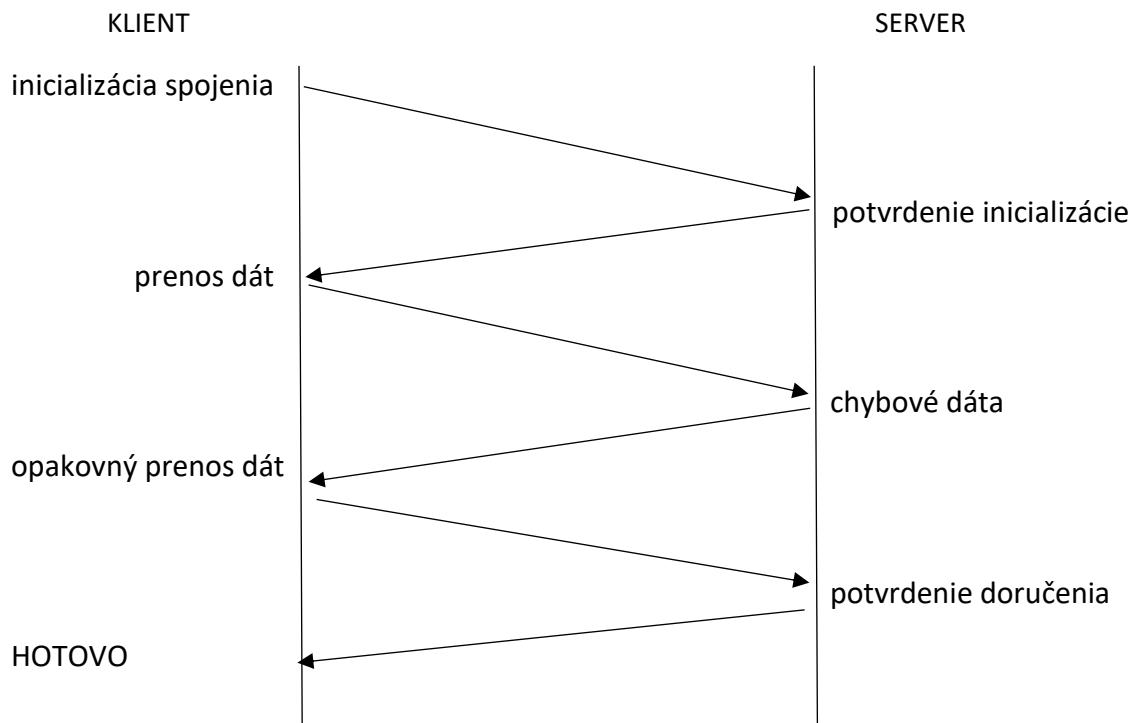
Klient musí ako input vpísať viac dát a to konkrétne IP adresu servera, port , veľkosť fragmentu, druh správy – či je to jednoduchý message alebo súbor, ak je to message tak ďalej požiadava o napísanie správy , ak je to súbor tak požiadava o cestu k súboru. Taktiež sa opýta, či sa majú dáta simulovane naschvál poškodiť.

Následne pošle inicializačnú správu serveru, po ktorej príde správa zo servera, ktorá nám hovorí o úspešnej inicializácii. Ak nepríde, bude užívateľ vyzvaný na opätovné zadanie IP adresy a portu.

Súbor rozdelí na fragment podľa špecifikovanej veľkosti, ak klient zadal väčšiu veľkosť ako je potrebná, tak sa zmení na takú veľkosť aká je veľkosť dát.

Ak po odoslaní správy alebo súboru klient po vyzvaní od servera neodošle inicializačný fragment serveru, v prípade ak chce poslať ďalšiu správu (v prípade ak nechce tak sa klient vypne) tak sa posiela tzv. “keep\_alive” správa, na zachovanie spojenia.

Po odoslaní desiatich fragmentov klient čaká na správu od servera o správnosti, ak server ohlásí chybové dáta, tak ich klient zaradí do queue a ak príde na rad tak sa znova pošlu.



Martin Hric

ID: 111696

## Hlavička

Takto vyzerá celý fragment aj s dátami.

1500 B (20B => IP, 8 B => UDP, MY HEADER = 9 B)					
TYP SPRÁVY	VEĽKOSŤ FRAGMENTU	POČET FRAGMENTOV	PORADIE FRAGMENTU	DÁTA	CRC
1 byte	2 bytes	2 bytes	2 bytes	max 1463 bytes	2 bytes

## Typy správ

DEC	BIN	Význam
1	001	Inicializácia spojenia
2	010	Odosielajúci sa fragment
3	011	Chybné dáta
4	100	Keep_alive
5	101	Správne dáta

## CRC

Na kontrolu správnosti fragementu, ktorý sa prijal sa použije CRC (cyclic redundancy check).

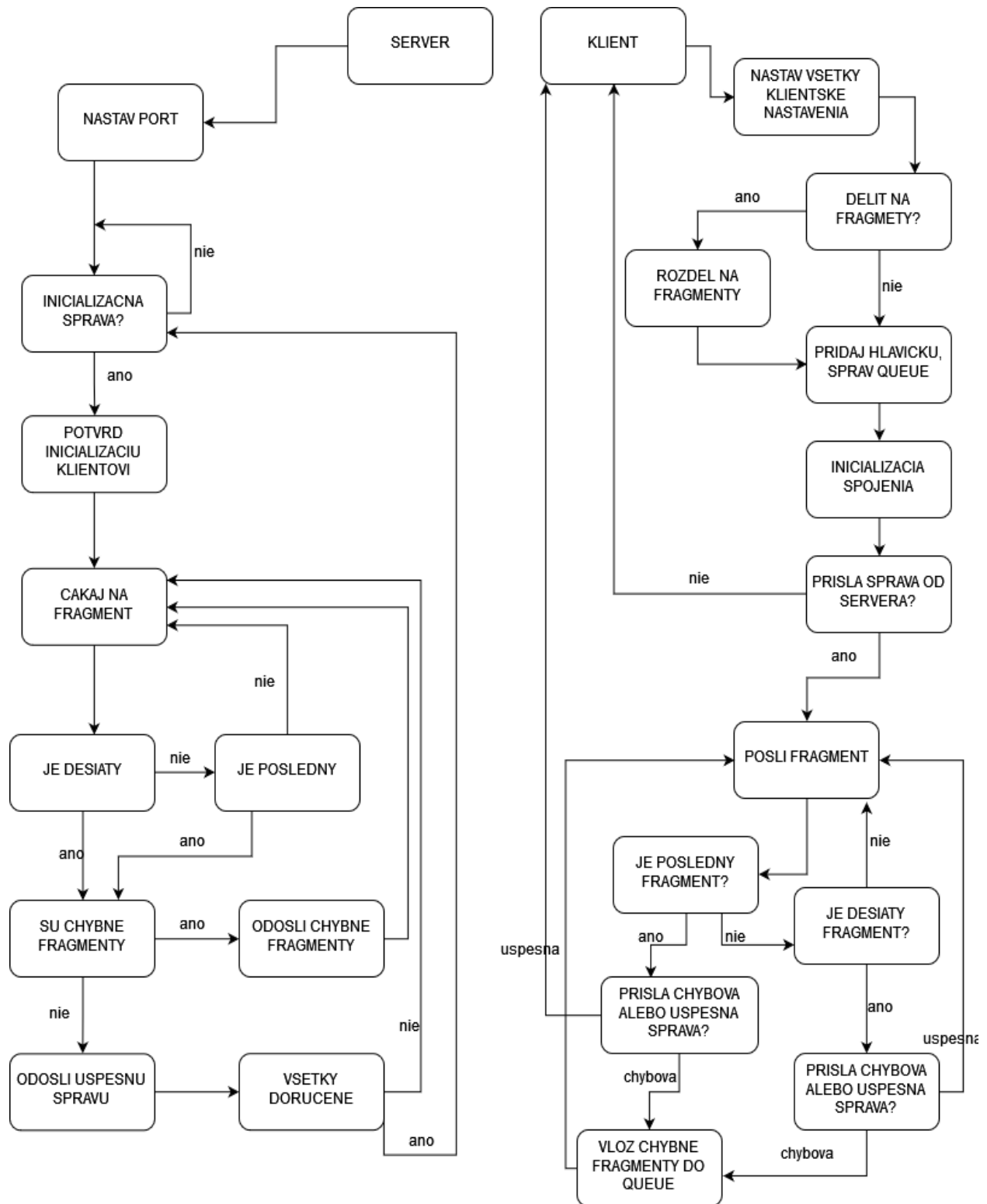
Pri odosielané dáta sú delené polynómom n- tého stupňa, pričom na koniec sa pripisuje remainder.

Pri prijímaní sa dáta znovu vydedia, ak bude zvyšok rovnaký, znamená to, že nenastala žiadna chyba.

Najčastejšie sa používa tento polynóm =>  $x^{16} + x^{12} + x^5 + 1$

CRC je omnoho presnejší ako checksum, kvôli tomu som sa ho rozhodol použiť z knižnice *libsrcrc*.

## DIAGRAM



Martin Hric  
ID: 111696

## ZMENY OPROTI NÁVRHU

Nevykonal som takmer žiadne zmeny, keďže v čase písania návrhu som teoreticky mohol písať dokumentáciu , pretože som to mal už z väčšej časti spravené.

Jediné, čo bolo zmenené je, že prv som mal 2 osobitné zdrojové kódy pre server a klient, teraz som ich ale parsol do jedného main.py kódu, kvôli Vášmu emailu, aby nám vedel klient a server zmeniť role.

Čo som ale nezmenil, čo ste spomínali bola veľkosť v hlavičke všetkých fragmentov, keďže ak je malý nastavený fragment, tak potom 2MB súbor to neodošle pretože to preteká.

## ANALÝZA

V rámci sieťovej komunikácie existuje veľké množstvo protokolov, ktoré musia spolupracovať, aby sme vďaka nim dokázali preniesť dáta na iné zariadenia v sieti. Na vyjadrenie vzťahov medzi jednotlivými protokolmi sa spopularizovali dva modely, ktorými sú model TCP/IP a referenčný model OSI. Oba modely sa skladajú z vrstiev, pričom každá vrstva poskytuje svoju funkcionálnu vrstvu, ktorá je jej nadradená.

### **V modeli TCP/IP je päť vrstiev:**

1. Aplikačná vrstva
2. Transportná vrstva
3. Sieťová vrstva
4. Linková vrstva
5. Fyzická vrstva

Aplikačná vrstva je najvyššou vrstvou a slúži na komunikáciu medzi koncovými aplikáciami. Medzi príklady protokolov aplikačnej vrstvy patria napr. HTTP, FTP a SMTP.

Transportná vrstva slúži na komunikáciu medzi hostami, ktorí sú adresovaní jednotlivými portmi. Tu sú najpoužívanejšími protokolmi TCP a UDP.

### **Transmission Control Protocol (TCP)**

TCP je spojovo orientovaný protokol, ktorý sa v porovnaní s UDP vyznačuje svojou spoľahlivosťou. To znamená, že sa používa pri prenose dát, kde je potrebná bezstratovosť, keďže sa uisťuje, či transfer dát prebehol úspešne. Na začiatku spojenia sa využíva tzv. three-way-handshake na zabezpečenie spoľahlivej komunikácie. Vďaka horeuvedeným vlastnostiam sa ale tento protokol stáva pomalším a nevhodným napríklad na streamovanie filmov/seriálov, kde sa vyžaduje rýchlosť.

### **User Datagram Protocol (UDP)**

UDP nie je spojovo orientovaný protokol a teda nezriaďuje spojenie pred prenosom dát. Vyznačuje sa svojou rýchlosťou, ktorú nadobúda práve vďaka tomu, že nedeteguje straty a nežiada o opätovné zaslanie dát v prípade ak nejaké nastanú. Taktiež podporuje broadcast a multicast, keďže nemusí nadväzovať spojenie pred samotným prenosom ako TCP. Jedna správa v UDP sa nazýva datagram a jeden datagram môže mať maximálnu veľkosť 65 506 B po zohľadnení veľkosti hlavičky IP (20B) a veľkosti hlavičky UDP (8B). Avšak na to, aby sa správa nedelila aj na linkovej vrstve, musí mať 1500 bajtov, čo je po zohľadnení IP hlavičky a UDP hlavičky 1472 bajtov. Najväčší fragment teda môže mať maximálnu veľkosť 1472 bajtov aj s hlavičkou. Nerátam moju hlavičku, čiže ešte -9 B.



## IMPLEMENTÁCIA

Ak si užívateľ na začiatku behu programu zvolí, že chce byť klient (a teda odosielateľ), zobrazí sa mu menu, v ktorom musí nastaviť všetky nastavenia, vrátane IP, portu, maximálnej veľkosti fragmentov, textu správy/cestu k súboru a taktiež si užívateľ môže zvoliť, či chce odoslať aj chybné fragmenty. Správa alebo prípadný súbor sa následne rozdelí na fragmenty pomocou metódy `make_fragments()`, kde sa mu pridá hlavička a päta obsahujúca CRC kód.

Po nastavovaní sa inicializuje spojenie so serverom. Pošle sa inicializačná správa, na ktorú server odpovie rovnakou inicializačnou správou. Následne sa vyšle správa s názvom súboru (ak nejaký je) a celkovým počtom fragmentov. Potom sa už začnú odosielať jednotlivé fragmenty správy. Pokiaľ je na serveri prijatý nesprávny fragment (čo je zistené pomocou porovnania CRC dátovej časti s CRC kódom uloženým v päte fragmentu), alebo sa nejaký fragment niekde stratí, odosielateľ prijme správu s ich indexami a znovu ich pridá do fronty na odoslanie. Na potvrdzujúcu správu čaká vždy po každých desiatich odoslaných fragmentoch.

Po úspešnom odoslaní všetkých fragmentoch môže užívateľ znovu odoslať ďalšiu správu na rovnaký server, pričom nemusí byť potrebná ďalšia inicializácia spojenia. Pokiaľ je užívateľ v menu, je každých 25 sekúnd odosielaná správa na udržanie spojenia.

Martin Hric  
ID: 111696

## FUNKCIE

`make_fragments()` – rozdelí buď súbor alebo správu vo forme bytearray na fragment danej veľkosti. Ak je 0, vypočíta potrebnú veľkosť fragment. Funkcia pre klienta.

`keep_alive()` – posiela správu s typom "4" o udržaní spojenia, pokiaľ nie je thread ukončený.

`end_menu()` – koncové menu klienta, vytvára thread pre `keep_alive`.

`send()` – hlavná funkcia klienta pre odosielanie fragmentov a čakanie na ARQ stop and wait 10, pre potvrdenie

`parser()` – funkcia pre server, slúži len na to aby sa mi jednoduchšie pristupovalo k dátam vo fragmente.

`start_client()` – inicializuje premenné a nastavuje IP, port..

`start_server()` - hlavná funkcia servera

`check_ip()` – overuje IP

`get_ip_address()` – kvôli tomu, že posielam po ethernet a `socket.gethostbyname(socket.gethostname())` funguje pre loopback, nie ethernet.

`main()` – zapínam celý program a vyberám tam, či je to klient alebo server.

Globálna premenná `FAILED_COUNT` mi určuje, koľko Bytov má byť poškodených.

Martin Hric

ID: 111696

## Používateľské prostredie

SERVER:

```
Command Prompt - python main.py
Davka c.1962 prijata bez chyb
Davka c.1963 prijata bez chyb
Davka c.1964 prijata bez chyb
Davka c.1965 prijata bez chyb
Davka c.1966 prijata bez chyb
Davka c.1967 prijata bez chyb
Davka c.1968 prijata bez chyb
Davka c.1969 prijata bez chyb
Davka c.1970 prijata bez chyb
Davka c.1971 prijata bez chyb
Davka c.1972 prijata bez chyb
Davka c.1973 prijata bez chyb
Davka c.1974 prijata bez chyb
Davka c.1975 prijata bez chyb
Davka c.1976 prijata bez chyb
Davka c.1977 prijata bez chyb
Davka c.1978 prijata bez chyb
Davka c.1979 prijata bez chyb
Davka c.1980 prijata bez chyb
Davka c.1981 prijata bez chyb
Davka c.1982 prijata bez chyb
Davka c.1983 prijata bez chyb
Davka c.1984 prijata bez chyb
Davka c.1985 prijata bez chyb
Davka c.1986 prijata bez chyb
Davka c.1986 prijata bez chyb
Cesta k suboru: C:\Users\marti\Desktop\PKS\zadanie2\adm2021.zip
Spojenie je udrziavane klientom
Spojenie je udrziavane klientom
```

Martin Hric

ID: 111696

KLIENT:

```
Command Prompt - python main.py
Microsoft Windows [Version 10.0.19042.1348]
(c) Microsoft Corporation. All rights reserved.

C:\Users\marti>cd Desktop

C:\Users\marti\Desktop>cd PKS

C:\Users\marti\Desktop\PKS>cd zadanie2

C:\Users\marti\Desktop\PKS\zadanie2>python main.py
CLIENT(0) alebo SERVER(1):0
INPUT IP ADDRESS WHERE TO SEND DATA: 192.168.0.59
INPUT SERVER PORT(5000-65535): 8000
INPUT FRAGMENT SIZE(1-1463) || 0 for automatic: 0
INPUT TYPE NUMBER -> MESSAGE(0) OR FILE(1): 0
TYPE THE MESSAGE ->zdravim druhy pocitac
DO YOU WANT DATA TO BE CORRUPTED? YES(1) OR NO(2): 2
Spojenie bolo vytvorene
velkost fragmentu nastavena na: 21
1 fragmentov bude poslanych
Davka c.0 uspesne dorucena
Ako si zelate pokracovat? ZMENIT ROLU(0) , POSLAT DATA na rovnaky server(1) , ZMENIT SERVER(2) , UKONCIT(3):
```

Martin Hric  
ID: 111696

## WIRESHARK

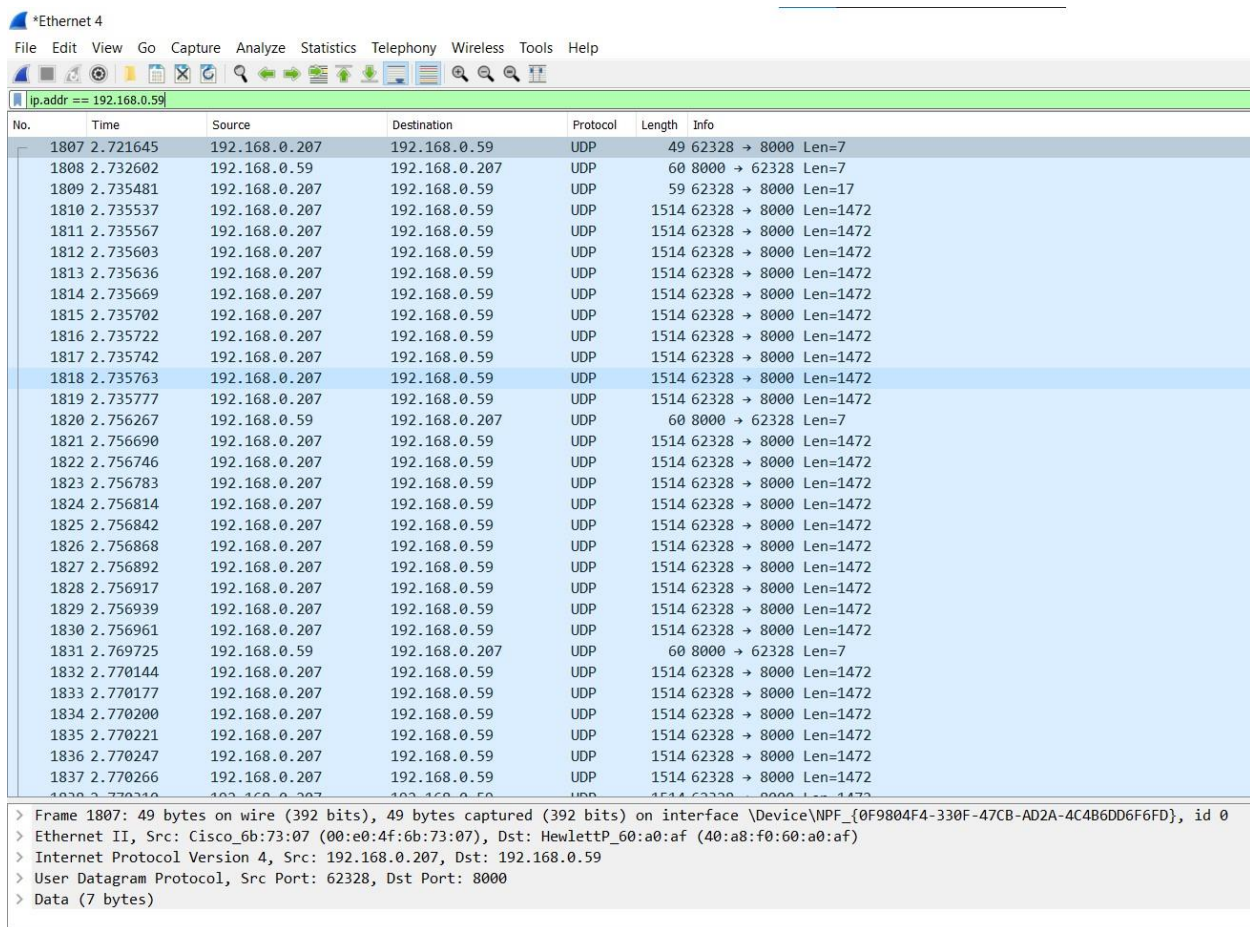
Takto vyzerá komunikácia vo wiresharku, pri posielaní súboru.

Prvý packet je od klienta na server inicializovanie spojenia.

Druhý packet je od servera klientovi potvrdenie inicializácie.

Tretí packet je informovanie servera, koľko fragmentov bude poslaných a dĺžka fragmentu.

Potom sa už len posielajú fragmenty s dátami, a po každej 10 vidíme, že od servera prichádza správa o informovaní, či boli správne alebo nie.



\*Ethernet 4

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

ip.addr == 192.168.0.59

No.	Time	Source	Destination	Protocol	Length	Info
1807	2.721645	192.168.0.207	192.168.0.59	UDP	49	62328 → 8000 Len=7
1808	2.732602	192.168.0.59	192.168.0.207	UDP	60	8000 → 62328 Len=7
1809	2.735481	192.168.0.207	192.168.0.59	UDP	59	62328 → 8000 Len=17
1810	2.735537	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1811	2.735567	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1812	2.735603	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1813	2.735636	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1814	2.735669	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1815	2.735702	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1816	2.735722	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1817	2.735742	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1818	2.735763	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1819	2.735777	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1820	2.756267	192.168.0.59	192.168.0.207	UDP	60	8000 → 62328 Len=7
1821	2.756690	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1822	2.756746	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1823	2.756783	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1824	2.756814	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1825	2.756842	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1826	2.756868	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1827	2.756892	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1828	2.756917	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1829	2.756939	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1830	2.756961	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1831	2.769725	192.168.0.59	192.168.0.207	UDP	60	8000 → 62328 Len=7
1832	2.770144	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1833	2.770177	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1834	2.770200	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1835	2.770221	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1836	2.770247	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1837	2.770266	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472
1838	2.770286	192.168.0.207	192.168.0.59	UDP	1514	62328 → 8000 Len=1472

> Frame 1807: 49 bytes on wire (392 bits), 49 bytes captured (392 bits) on interface \Device\NPF\_{0F9804F4-330F-47CB-AD2A-4C4B6DD6F6FD}, id 0

> Ethernet II, Src: Cisco\_6b:73:07 (00:e0:4f:6b:73:07), Dst: HewlettP\_60:a0:af (40:a8:f0:60:a0:af)

> Internet Protocol Version 4, Src: 192.168.0.207, Dst: 192.168.0.59

> User Datagram Protocol, Src Port: 62328, Dst Port: 8000

> Data (7 bytes)

## ZÁVER

Tento program slúži na spracovanie vstupného súboru, prípadné delenie na fragmenty a odoslanie fragmentov spolu s novo vytvorenými hlavičkami cez sieť na iný počítač.

Ten prijíma fragmenty, skontroluje ich a znovu ich spojí do jednotného súboru.

Testovanie prebehlo medzi dvoma počítačmi pripojenými cez switch

Počas testovania boli odosielané a prijímané súbory do sedemdesiat megabajtov, pričom jedinou limitáciou pri odosielaní súborov bola hlavička, keďže počet fragmentov aj index fragmentu bol ukladaný do dvoch bajtov.

Taktiež boli skúšané rôzne kombinácie rôzne veľkých textových správ, ktoré boli odosielané s rôznymi počtami chybných fragmentov.