SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGII

KOMUNIKÁCIA S VYUŽITÍM UDP PROTOKOLU

POČÍTAČOVÉ A KOMUNIKAČNÉ SIETE

Meno: Michal Franczel

Cvičiaci: Ing. Pavol Helebrandt, PhD.

Šk. rok: 2019/2020

Obsah

Zadanie

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 binárneho 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.

Komunikátor 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 20-60s 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. Použité knižnice a funkcie 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 odoslať minimálne 1 chybný fragment (do 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.
8. Možnosť odoslať súbor a v tom prípade ich uložiť na prijímacej strane ako rovnaký súbor. Akceptuje sa iba ak program prenesie 2MB súbor do 60s bez chýb.

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 funkcionalitu vrstve, 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.

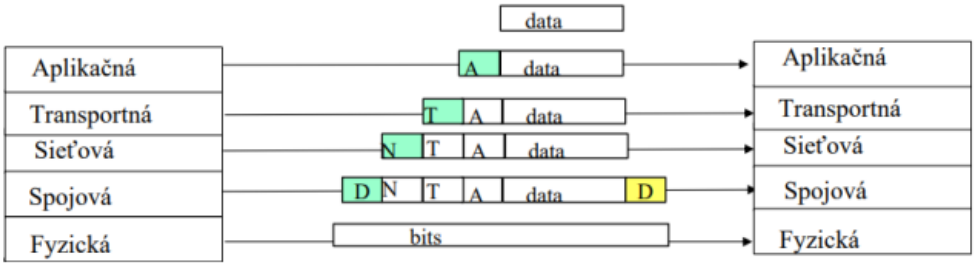
User Datagram Protocol (UDP)

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.**

Enkapsulácia dát

Jednotlivé vrstvy majú svoje špecifické údaje, ktoré sa musia odoslať spolu s dátami. Tieto údaje sa ukladajú do bloku nachádzajúceho sa pred samotnými dátami, ktorý sa volá hlavička. Vrstvy si takto pridávajú na začiatok hlavičky, každá nadchádzajúca si uloží tú hlavičku pred tú predchádzajúcu. Tento koncept postupného zabaľovania sa nazýva enkapsulácia. Protokol UDP má hlavičku o veľkosti 8B, pričom si v nej uchováva zdrojový port, cieľový port, dĺžku a kontrolný súčet, vďaka ktorému sa dá zistiť prípadná nezrovnalosť ktorá nastala pri prenose.



Obrázok 1- ilustrácia enkapsulácie, zelená označuje hlavičku pridanú v danej vrstve

Cyclic Redundancy Check

Na kontrolu správnosti prijatého fragmentu sa použije tzv. CRC. Pri odosielaní dát budú dáta delené polynómom *n*-tého stupňa , pričom na koniec dát sa pripíše zvyšok dĺžky *n.* Pri prijímaní sa dáta znovu vydelia polynómom. Ak tento raz bude zvyšok rovnaký ako je zvyšok zapísaný v prijatom fragmente, znamená to, že nenastala žiadna chyba pri prenose dát. Na kontrolu správ tu bude využitý nasledujúci polynóm[1]:

$$x^{16} + x^{15} + x^2 + 1$$

[1] https://en.wikipedia.org/wiki/Cyclic_redundancy_check

Návrh

Implementácia protokolu prebehne v jazyku *python* s použitím knižnice *socket* a knižnice *io*. Testovanie prebehne pri nadviazaní spojenia medzi počítačom s operačným systémom Linux a počítačom s operačným systémom Windows.

Druhy správ

Správa (dec)	Správa (bin)	Význam správy
1	0001	Inicializácia spojenia
2	0010	Prenos dát – hlavička odosielaných fragmentov
3	0011	Doručené dáta boli chybné
4	0100	„Keep alive“ správa
5	0101	Doručené dáta boli správne

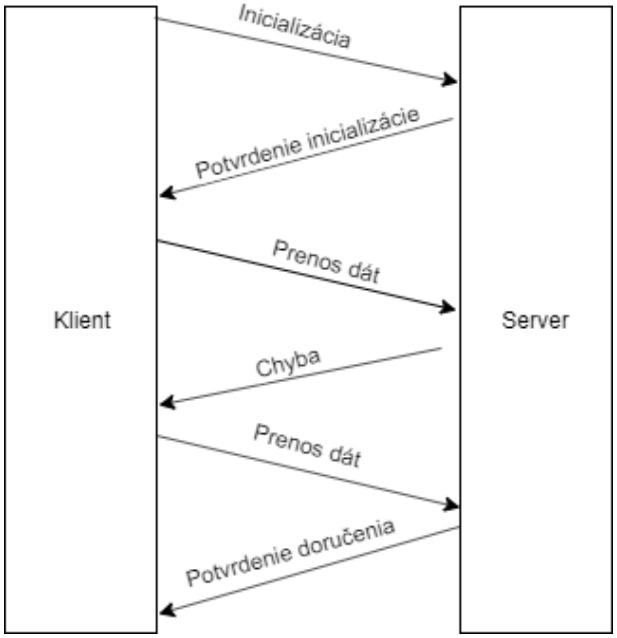
Používateľské rozhranie

Používateľské rozhranie bude implementované formou konzolovej aplikácie. V rámci aplikácie bude existovať príkaz „help“, ktorý vypíše všetky kroky, ktoré treba splniť na to, aby program prebehol úspešne. Údaje nevyhnutné na chod programu (napr. IP adresa a port v prípade klienta) budú vyžadované od užívateľa a teda ak ich nevyplní, program mu neumožní pokračovať. Po spustení užívateľ zvolí či chce figurovať v danom pripojení ako server/klient, program ho vyzve na vyplnenie povinných údajov a následne sa spustí prenos. Keď bude prenos ukončený, dostane užívateľ správu o úspešnosti prenosu.

Odosielanie a prijímanie dát

Užívateľ si bude môcť pri spustení programu vybrať, či chce figurovať v spojení ako server (prijímať dáta) alebo klient (vysielať dáta).

Ak si užívateľ vyberie možnosť server, musí mu nastaviť port. Následne začne server počúvať, či naň neprichádza správa žiadajúca o inicializáciu spojenia. Ak neprichádza, server pokračuje v počúvaní. Inak odošle správu potvrdzujúcu pripojenie. Kontrolovať prijaté fragmenty bude po prijatí desiatich fragmentov alebo po prijatí posledného fragmentu. Ak sa nájde chybný fragment, odošle správu informujúcu klienta o chybných dátach s poradiami chybných fragmentov, inak pošle správu o úspešne prenesených dátach. Po obdržaní všetkých fragmentov, ktoré boli po použití CRC vyhodnotené ako nepoškodené, budú tieto fragmenty spojené a uložené ako súbor. Následne server znova počúva.



Obrázok 2 - ilustrácia prenosu jedného chybného fragmentu

Ak si užívateľ vyberie možnosť klient, musí nastaviť IP adresu a port servera, kam chce odosielať dáta a cestu k súboru, ktorý chce odosielať. Voliteľne môže nastaviť aj maximálnu veľkosť fragmentov. 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. Inak rozdelí súbor do fragmentov (ak užívateľ zadal maximálnu veľkosť tak sa riadi podľa nej, inak sám nájde optimálnu veľkosť) a začne ich odosielať. V prípade dlhšej pasivity bude musieť byť odosielaný „keep alive“ správa a to každých 30 sekúnd, aby sa udržalo spojenie.

Po odoslaní desiatich fragmentov alebo po odoslaní posledného fragmentu bude čakať na správu potvrdzujúcu úspešné prijatie fragmentov. Ak príde chybová správa, zaradí fragmenty, ktoré boli na strane servera vyhodnotené ako chybné do fronty a teda keď prídu na rad budú znovu odoslané. Po úspešnom prenesení všetkých fragmentov sa vypíše správa o úspešnom prenesení a užívateľ môže inicializovať ďalší prenos.

Hlavička

Typ – jeden z hore uvedených typov správ (1 bajt)

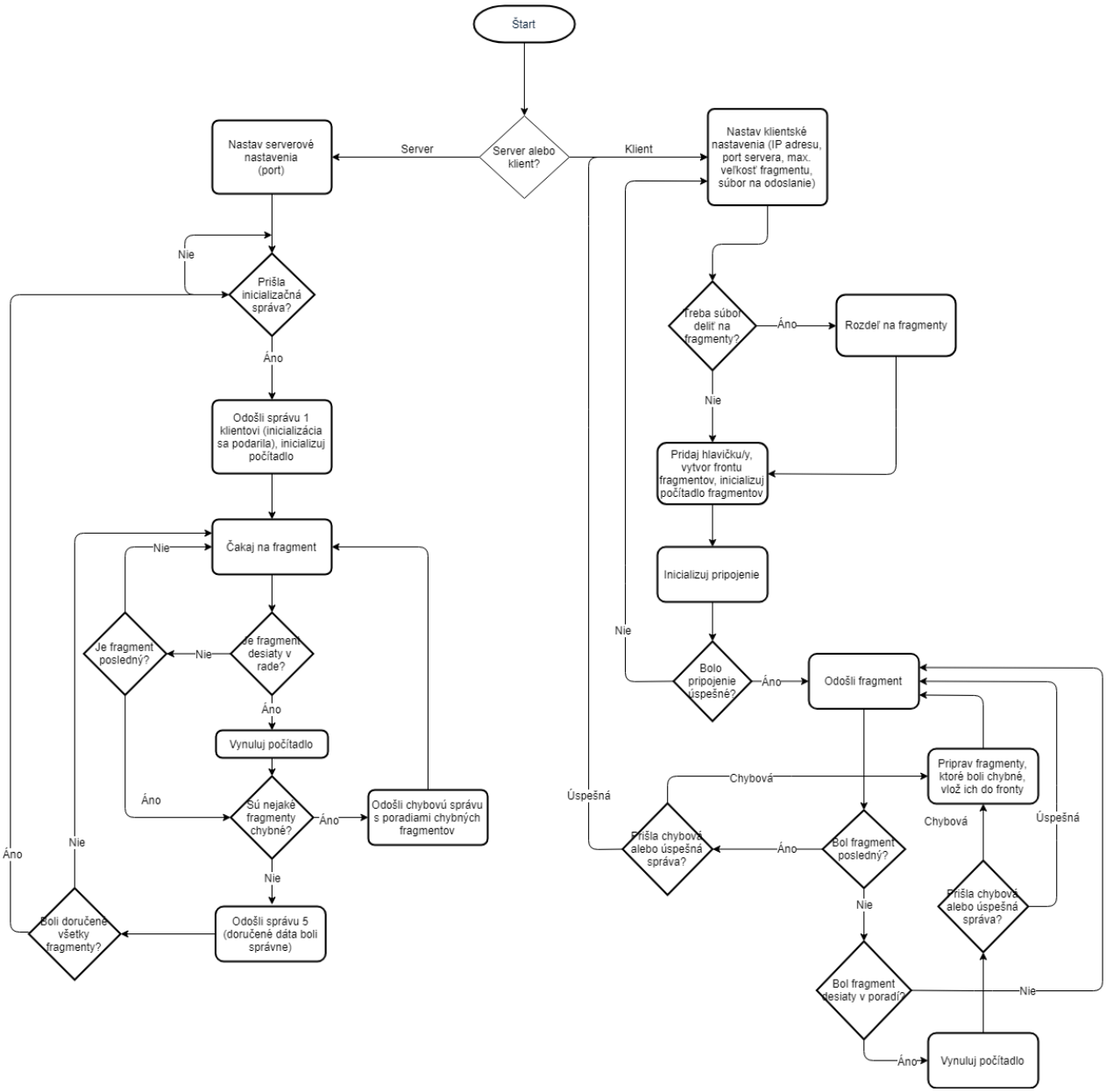
Veľkosť fragmentu – veľkosť odosielaného fragmentu (2 bajty)

Počet fragmentov – celkový počet odosielaných fragmentov (2 bajty)

Poradie fragmentu – poradie odosielaného fragmentu (2 bajty)

CRC – zvyšok po delení stanoveným polynómom

Vývojový diagram



Implementácia

Zmeny oproti návrhu

1. **Knižnice**
   Na rozdiel od návrhu bolo použitých viacero knižníc, pričom jedna spomenutá v návrhu nakoniec nebola použitá (*io*). V implementácií boli pridané nasledovné knižnice:

   PyInquirer – bola použitá na vylepšenie a skrášlenie užívateľského rozhrania, nie je štandardnou knižnicou programovacieho jazyka *Python*

   libscrc – použitá na vypočítanie CRC kódu používaného na skontrolovanie prenesenej správy

   Štandardné knižnice: *queue, math, os, threading*

2. **Maximálna veľkosť fragmentu**
   V návrhu bola uvedená maximálna veľkosť fragmentu 1472 bajtov, čo bola veľkosť fragmentu aj s hlavičkou. Maximálna veľkosť fragmentu, ktorú môže zadať užívateľ pri odosielaní dát je po odčítaní tejto hlavičky (a päty pre CRC) 1463 bajtov.

3. **Zmeny v priebehu programu**
   Na rozdiel od návrhu sa nekontroluje, či prišli všetky fragmenty až na konci prijímania fragmentov, ale v rámci aktuálne prijímanej desiatky fragmentov. Taktiež nebol implementovaný príkaz „help“, ktorý vzhľadom na spôsob realizácie užívateľského rozhrania nebol potrebný. Pridané bolo aj odosielanie prvého fragmentu, ktorý obsahuje názov súboru a celkový počet fragmentov. V prípade, že sa odosiela len textová správa, obsahuje len celkový počet fragmentov.

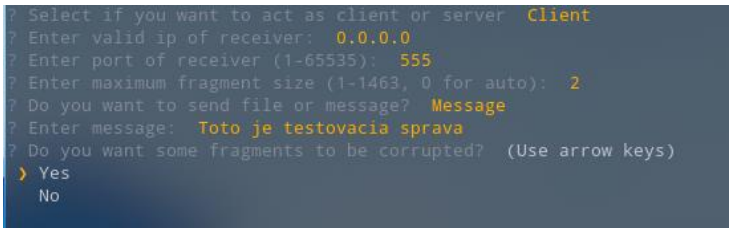
Ak si užívateľ na začiatku behu programu zvolí, že chce byť klient (a teda odosielateľ), zobrazí sa mu menu, v ktorom musí nastaviť všetky nastavenia, vrátane IP, portu, maximálnej veľkosti fragmentov, textu správy/cestu k súboru a taktiež si užívateľ môže zvoliť, či chce odoslať aj chybné fragmenty. 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.

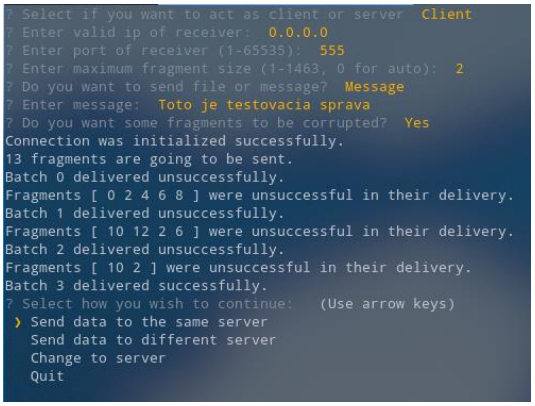
Po úspešnom odoslaní všetkých fragmentoch môže užívateľ znovu odoslať ďalšiu správu na rovnaký server, pričom nemusí byť potrebná ďalšia inicializácia spojenia. Pokiaľ je užívateľ v menu, je každých 25 sekúnd odosielaná správa na udržanie spojenia.

Užívateľské rozhranie

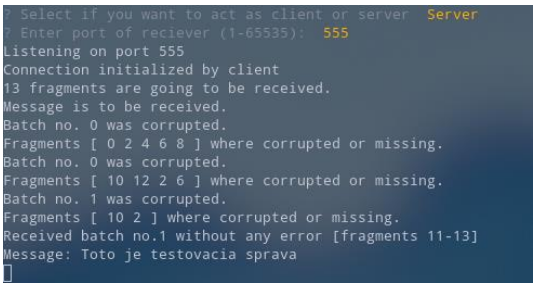
Po zvolení možnosti „Client“, užívateľ postupne začne vypĺňovať informácie nutné na chod programu:



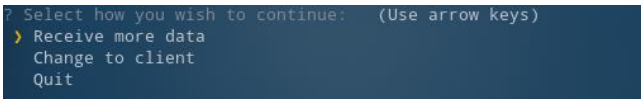
Najprv musí nastaviť IP adresu a port prijímateľa, pričom tieto polia sú povinné. Ďalším údajom je následne maximálna veľkosť fragmentu, pri ktorej sa ale dá zadať aj nula, ktorá reprezentuje automatické vypočítanie maximálnej veľkosti fragmentu. Ďalej má užívateľ možnosť si zvoliť, či chce odoslať správu alebo súbor a po výbere jednej z týchto možností si program vyžiada buď správu alebo cestu k súboru, ktorý chce užívateľ odoslať. Poslednou možnosťou je možnosť odosielania chybných fragmentov. Pri afirmatívnej odpovedi sa pozmení obsah maximálne desiatich fragmentov. Po zapísaní všetkých údajov sa začne odosielať súbor.



Pri odosielaní sa zobrazí správa o úspešnej inicializácií spojenia, koľko fragmentov bude odoslaných a následne sa začnú vypisovať odosielané fragmenty. Ak sa odosiela súbor, vypíše sa aj cesta k súboru. V prípade, že fragment nepríde na druhú stranu v takom stave, ako by mal, tak sa vypíšu aj fragmenty, ktorých odoslanie prebehlo neúspešne. Na konci odosielania sa vypíše ďalšie menu, kde má užívateľ možnosť vypnúť program, poslať dáta na rovnaký server (bez nutnosti opätovnej inicializácie spojenia), poslať dáta na iný server alebo zmeniť sa na server. Pokiaľ sa užívateľ nachádza v tomto menu, je prijímateľovi periodicky odosielaná správa na udržanie spojenia.



V prípade, že si na začiatku programu zvolí užívateľ možnosť „Server“, program si od neho vypýta len port, na ktorom bude počúvať, či neprišla inicializačná správa. Ak príde, vypíše sa koľko fragmentov bude prijatých a či bude prijatá správa alebo súbor. Následne program vypisuje prijaté desiatky fragmentov, pričom ak nastala chyba pri ich prijatí, vypíše, pri ktorých fragmentoch nastala chyba. Nakoniec sa vypíše správa alebo cesta ku prijatému súboru. Následne program čaká a ak je prijatá správa udržujúca spojenie, vypíše, že bola prijatá. Ak spojenie nie je udržované, vypíše sa nasledovné menu, kde si užívateľ môže vybrať, či chce pokračovať v prijímaní dát, či sa chce zmeniť na odosielateľa alebo či chce vypnúť program:



Hlavné použité funkcie

make_fragments(message, fragment_size) – rozdelí buď súbor alebo správu vo forme *bytearray* na fragmenty danej veľkosti. V prípade, že je veľkosť v argumente nulová, vypočíta najvyššiu možnú veľkosť fragmentu.

keep_alive(e, ip, port, sock) – pošle správu o udržaní spojenia každých 25 sekúnd pokiaľ nie je vlákno na ktorom beží ukončené.

display_end_menu() – zobrazí koncové menu odosielateľa. Táto funkcia zároveň vytvára a uzatvára vlákno, na ktorom prebieha funkcia keep_alive

send(ip, fragment_size, port, message, path, sock_et=0) – funkcia odosielajúca dáta na prijímateľovi

start_client() – funkcia, ktorá zobrazí odosielateľovi základné menu na nastavenie odosielania. Na konci volá funkciu *send()*

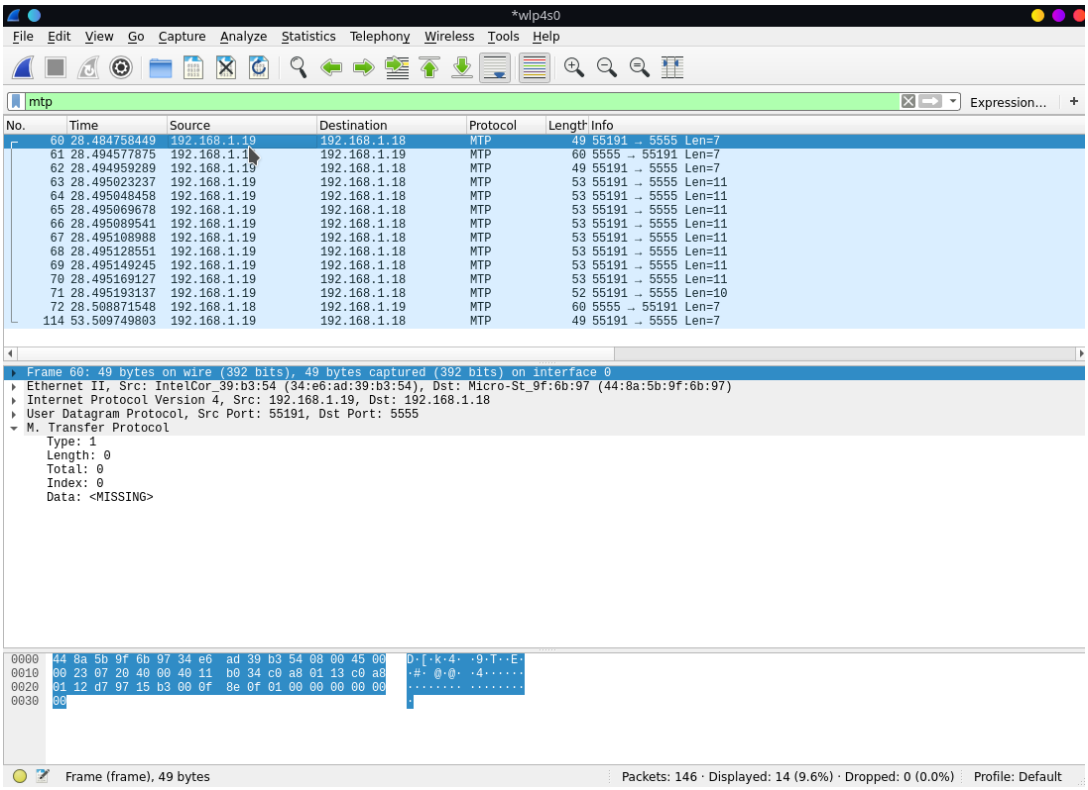
start_server() – základná funkcia prijímateľa, ktorá vykonáva prijímanie súboru/správy a vypisovanie.

Použité dátové štruktúry: *queue, bytearray, dictionary*

Na tvorbu hlavičiek bola použitá funkcia *to_bytes()*, ktorá zapíše hodnotu do argumentom určeného počtu bajtov.

Fragmenty zachytené aplikáciou Wireshark

Následné zachytenie fragmentov prebehlo pri prenose krátkej správy s nastavením veľkosti fragmentov na 2 bajty. Na zobrazenie obsahu fragmentov ako sú viditeľné nižšie bol použitý vlastný doplnok ku Wiresharku.



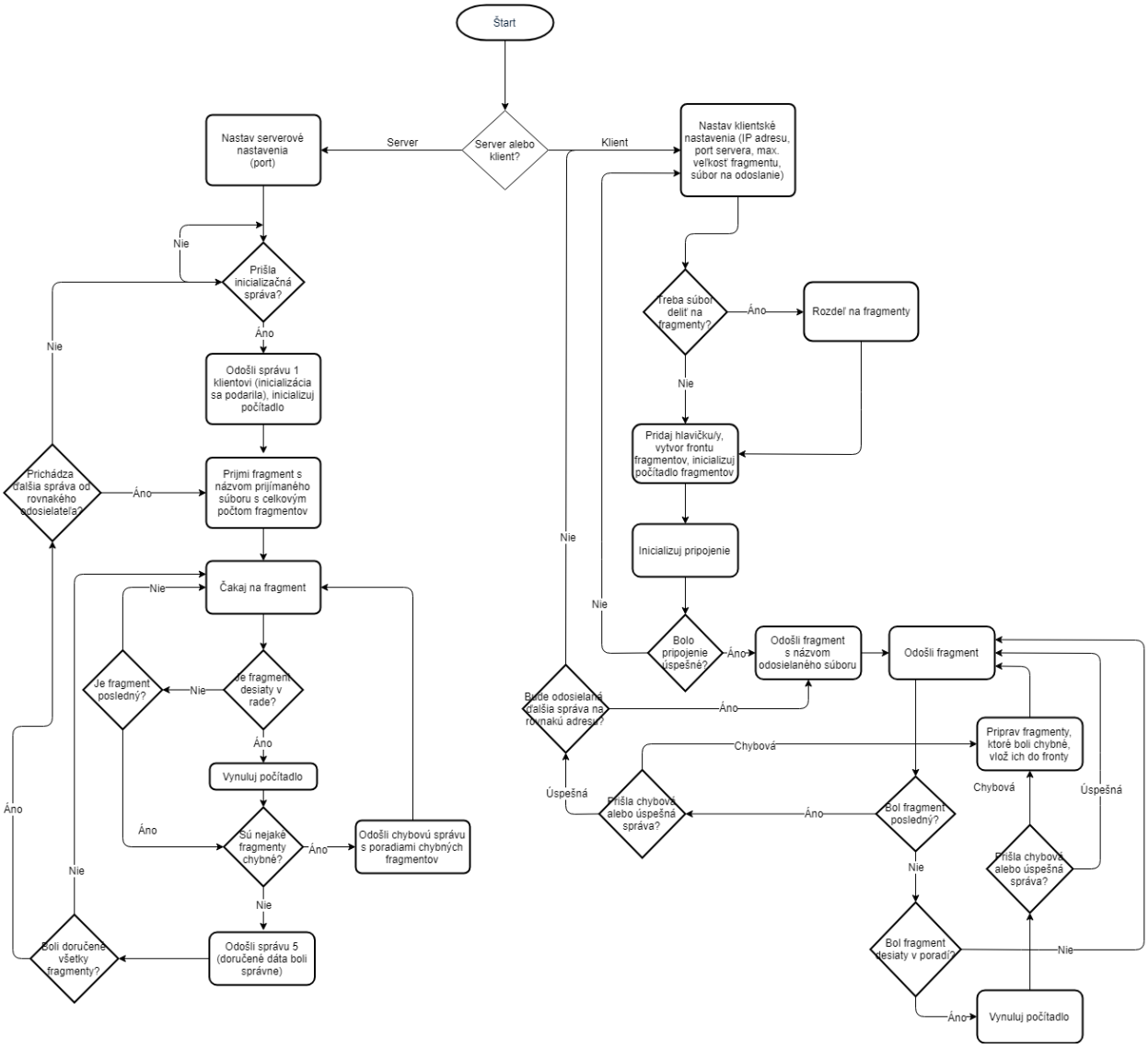
Fragment s dátami vyzerá nasledovne:

Ako vidieť na nižšie uvedenom obrázku, odpoveď prijímateľa na nesprávne doručené fragmenty obsahuje indexy piatich zle doručených fragmentov:

Splnené scenáre

1. Program je implementovaný v jazyku Python
2. Pri odosielaní je umožnené zadať IP adresu a port
3. Pri odosielaní si užívateľ môže zvoliť max. veľkosť fragmentu
4. Odosielateľ aj prijímateľ zobrazujú absolútnu cestu k súboru, veľkosť a počet fragmentov
5. Odosielateľ môže odoslať 10 chybných fragmentov
6. Odosielateľovi je oznamované správne/nesprávne doručenie fragmentov každých 10 fragmentov, pričom sa vypíšu indexy nesprávne doručených fragmentov
7. Odosielanie súborov a ich následné ukladanie
8. Prenášanie súboru menšieho ako jeden fragment
9. Prerušenie odosielania správ pre udržanie spojenia

Finálny vývojový diagram



V porovnaní s návrhom je podobný, no bolo pridané odosielanie prvého fragmentu, ktorý zabezpečuje prenos názvu odosielaného súboru. Pokiaľ nie je odosielaný súbor, ale iba správa, má tento fragment prázdnu dátovú časť. Vtedy spĺňa funkciu informátora o počte odosielaných fragmentov. Taktiež tam bolo pridané znázornenie toho, že nemusí byť inicializované spojenie v prípade, ak je odosielaný súbor tomu istému príjemcovi.

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 WiFi pripojením. 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.

Zdroje

1. Materiály k zadaniu dostupné v AIS
2. Prednášky dostupné v AIS
3. https://en.wikipedia.org/wiki/Cyclic_redundancy_check