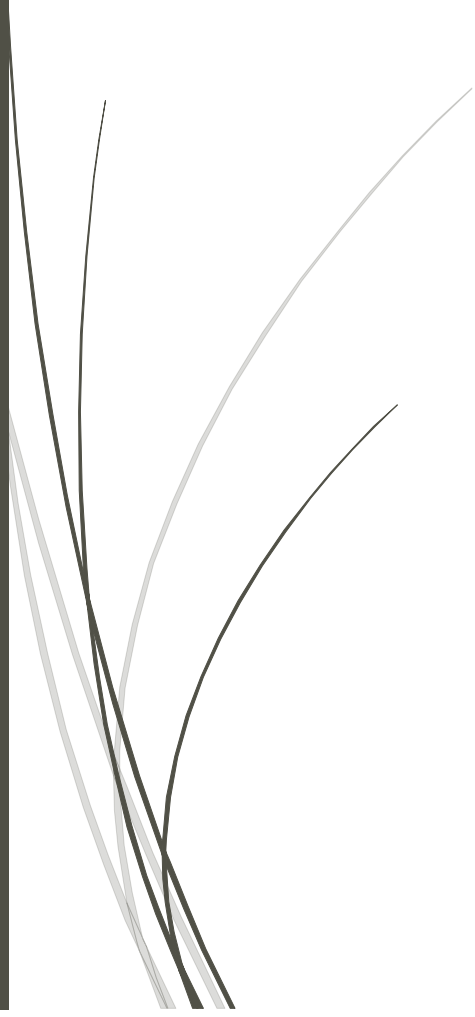
28/11/2021

Komunikácia s využitím UDP protokolu

Počítačové a komunikačné siete – zadanie 2



Samuel Hetteš, ID: 110968
STU FIIT 2021/2022

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 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: 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.

ANALÝZA

Na zabezpečenie plynulej sieťovej komunikácie boli vytvorené modely, ktoré opisujú funkcie komunikačného systému rozdelením problému na menšie časti – vrstvy, ktoré vykonávajú špecifickú úlohu. Typickým príkladom takéhoto modelu je napríklad OSI model alebo TCP/IP model, ktorý sa skladá zo 4 vrstiev:

- Aplikačná
- Transportná
- Sieťová
- Linková

Jednotlivé vrstvy medzi sebou komunikujú. V našom zadaní nás bude zaujímať hlavne transportná a aplikačná vrstva.

Aplikačná vrstva má za úlohu sprostredkovať služby aplikáciam, ako napríklad prenos správ a súborov. Definuje akýsi spôsob, akým aplikácie komunikujú so sieťou. Známe protokoly tejto vrstvy sú napríklad: HTTP, HTTPS, FTP, SSH, Telnet, TFTP a mnohé ďalšie.

Úlohou transportnej vrstvy je poskytovať prenos dát medzi koncovými užívateľmi. Typickými príkladmi protokolov tejto vrstvy sú TCP a UDP. Rozdiely medzi nimi sú hlavne v spoľahlivosti, kým TCP zabezpečuje nadviazanie spojenia, udržiavanie spojenia, bezchybný prenos dát (vyžiadanie chybných fragmentov) a ukončenie spojenia, UDP túto funkcionalitu nemá. Z toho však vyplýva, že UDP je rýchlejšie a využíva sa hlavne vtedy, keď potrebujeme zaručiť okamžitú odozvu, pričom prípadné straty dát nás až toľko nezaujímajú.

TCP

Ako som už povedal, TCP zaručuje bezchybný prenos dát. Jeho samotný cyklus pozostáva z 3 častí: nadviazanie spojenia, prenos dát a ukončenie spojenia.

- NADVIAZANIE SPOJENIA

Nadviazanie spojenia je pri TCP realizované pomocou 3-way handshaku, kde klient pošle serveru paket s nastaveným flagom SYN, ten mu odpovedá paketom s nastavenými flagmi SYN, ACK a na tento paket klient odpovedá paketom s nastaveným flagom ACK. Po uskutočnení tejto výmeny je spojenie úspešne nadviazané.

- PRENOS DÁT

Spoľahlivosť prenosu dát je pri TCP riešená viacerými mechanizmami. Jednak hlavička TCP obsahuje poradové čísla paketov pre určenie poradia daného segmentu a detekciu duplikátov, ďalej kontrolnú sumu checksum pre zistenie prípadných chýb, ktoré nastali pri prenose, taktiež využíva potvrdzovanie prijatia dát a časovače pre detekciu straty alebo oneskorenia dát.

- UKONČENIE SPOJENIA

Ukončenie spojenia pri TCP je realizované 4-way handshakom, kde jedna zo strán pošle paket s nastaveným flagom FIN, druhá strana odpovedá paketom s nastaveným flagom ACK a za ním pošle paket s nastaveným flagom FIN a prvá strana odpovedá paketom s nastaveným flagom ACK. Toto ukončenie môže byť realizované aj 3-way handshakom, kedy by druhá strana neodpovedala na FIN paket paketmi s nastavenými flagmi ACK a potom FIN, ale rovno pošle jeden paket s oboma týmito flagmi nastavenými. Okrem týchto ukončení existuje ešte aj ukončenie flagom RST.

UDP

Keďže UDP je nespolahlivý protokol, ktorý nerieši nadviazanie či ukončenie spojenia a hoci obsahuje overenie integrity pomocou checksumu, ale nerieši či dáta boli skutočne prijaté – nerieši potvrdzovanie. Pre zabezpečenie spoľahlivosti prenosu dát pomocou UDP protokolu je nutné definovať protokol nad ním, teda protokol aplikačnej vrstvy, čo je úlohou nášho zadania.

NÁVRH

V prvej časti vysvetlím typy polí, ktoré sa využívajú, následne predstavím hlavičky - typy správ a dátový paket a na záver ukážem diagram toku programu.

Typy polí

V tomto programe sa využíva viacero typov polí, z ktorých sa následne skladajú hlavičky.

Typy polí:

1B FLAGS – bity: [76543210]

- 7 – ANSWER: označuje žiadosť klienta o zaslanie odpovedi zo strany servera pri prenose
- 6 – START: označuje začiatok komunikácie/prenosu
- 5 – END: označuje koniec komunikácie/prenosu
- 4 – CONTROL: označuje paket viazaný s kontrolou komunikácie – pri začiatku, udržiavaní a ukončení komunikácie
- 3 – MESSAGE: označuje paket viazaný so správou
- 2 – FILE: označuje paket viazaný so súborom
- 1 – NACK: označuje negatívnu odpoveď/chybu
- 0 – ACK: označuje pozitívnu odpoveď

4B SEQUENCE NUMBER – poradové číslo paketu

4B FRAGMENT COUNT – počet fragmentov, ktoré budú posielané pri prenose

4B CHUNK SEQUENCE NUMBER – poradové číslo dávky paketov

2B DATA SIZE – veľkosť dát v dátovej časti paketu

2B FRAGMENT SIZE – veľkosť fragmentov definovaná užívateľom

2B CHUNK NACK COUNT – počet nedoručených/poškodených paketov v dávke paketov

2B CRC – cyklická kontrola redundancie

?B NACK SEQUENCE NUMBERS – poradové čísla paketov, ktoré boli nedoručené/poškodené

?B FILE NAME – názov súboru

Hlavičky - typy správ a dátový paket

Aby ste sa v rôznych typoch správ vyznali, vysvetlím ich v dvoch častiach: typy správ, ktoré riešia začiatok, udržiavanie a koniec spojenia a typy správ, ktoré riešia začiatok, priebeh a ukončenie prenosu správ/súborov.

- OTVORENIE SPOJENIA

Otvorenie spojenia začína vždy **poslaním inicializačného paketu zo strany klienta**, ktorý obsahuje pole FLAGS a pole WINDOW SIZE:

Kombinácia flagov START a CONTROL značí **žiadosť o otvorenie spojenia**.

Pokiaľ **veľkosť okna servera je rovnaká alebo väčšia ako veľkosť okna klienta**, server pošle naspäť rovnaký paket, ale s nastaveným flagom ACK:

Pokiaľ je **veľkosť okna servera menšia ako veľkosť okna klienta**, server odošle naspäť paket s nastaveným flagom NACK a jeho veľkosťou okna, ktorú si klient následne upraví:

Po obdržaní týchto paketov sa **spojenie** považuje za **otvorené**.

- UDRŽIAVANIE SPOJENIA

Udržiavanie spojenia **zabezpečuje klient** a to zasielaním paketu, ktorý obsahuje iba pole FLAGS, tentokrát je však nastavený iba flag CONTROL:

Na tento paket **server musí odpovedať** rovnakým paketom, ale s nastaveným flagom ACK:

- UKONČENIE SPOJENIA

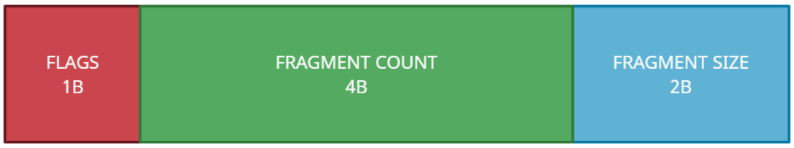
Ukončenie spojenia **môžu poslať obe strany**. Tento paket obsahuje opäť iba pole FLAGS:

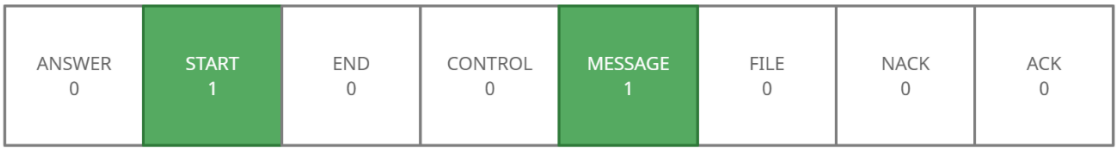
Kombinácia flagov END a CONTROL značí **ukončenie spojenia**. Na túto správu už jednotlivé **strany neposielajú odpoveď**. Jednoducho iba zašlú túto správu a ukončia spojenie.

- ZAČIATOK PRENOSU SPRÁVY/SÚBORU

Začiatok prenosu **inicializuje strana klienta** a to **zaslaním inicializačného paketu prenosu**, tento paket obsahuje polia FLAGS, FRAGMENT COUNT, FRAGMENT SIZE a pri prenose súboru aj pole FILE NAME, kde je uvedený názov súboru.

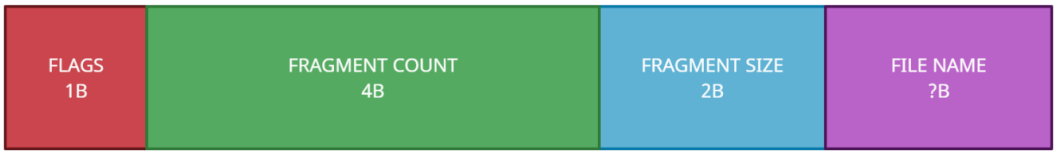
Inicializačný paket začiatku prenosu správy:

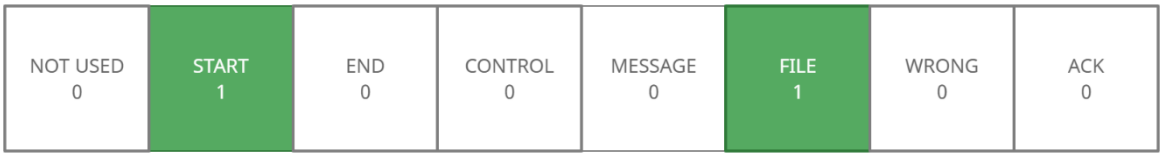




Vidíme, že tentokrát už neni spolu so START nastavený flag CONTROL, ale MESSAGE, čo značí **začiatok prenosu správy**.

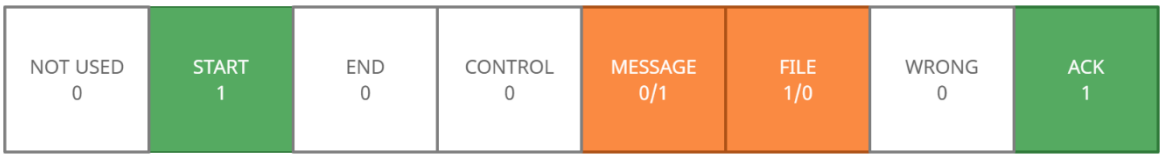
Inicializačný paket začiatku prenosu súboru:





Pri prenose súboru nie je spolu s flagom START nastavený flag MESSAGE, ale FILE, čo značí **začiatok prenosu súboru**.

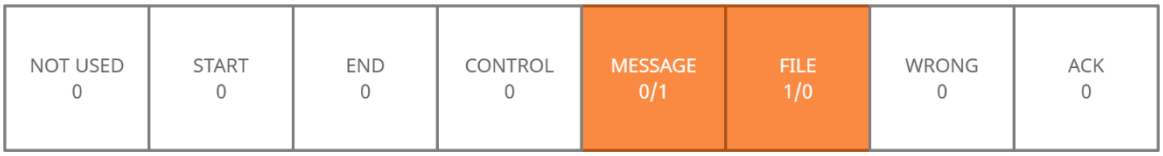
Na tieto inicializačné pakety **server musí odpovedať** zaslaním rovnakého paketu, avšak iba s polom FLAGS a nastaveným flagom ACK:



Po prijatí odpovede zo strany servera, **začne klient posielať dáta**.

- DÁTOVÝ PAKET

Tento paket obsahuje **najväčšiu hlavičku** s poliami FLAGS, SEQUENCE NUMBER, DATA SIZE a CRC (9B):



V poli FLAGS je **pri prenose dátového paketu nastavený iba flag** MESSAGE/FILE v závislosti od toho, čo sa prenáša.

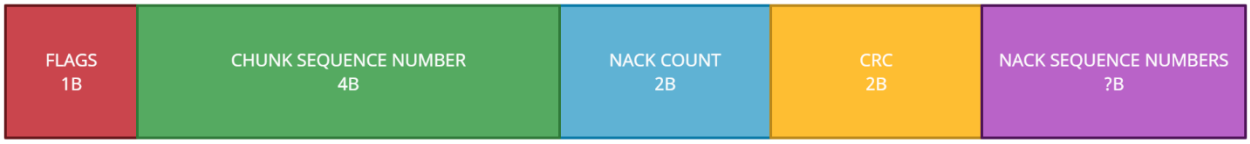
Server **neodpovedá na jednotlivé dátové pakety**, ale **odpovedá až keď mu prídu všetky pakety v jednej dávke** (chunku) v závislosti od veľkosti okna.

Odpoveď môže mať dve podoby. Buď všetky pakety prišli v poriadku alebo niektoré boli poškodené/neprišli.

Kladné potvrdenie:

Server týmto **potvrdzuje prijatie všetkých paketov v dávke** s daným poradovým číslom.

Negatívne potvrdenie:



Server týmto **oznamuje** klientovi, **že v dávke s daným poradovým číslom mu neprišlo** NACK_COUNT paketov. Z dôvodu dôležitosti korektného doručenia tejto správy sa tu nachádza aj pole CRC. Klient následne v ďalšej dávke pošle pakety, ktoré neboli doručené.

V prípade, že **server neodošle žiadnu odpoveď po zaslaní dávky**, klient pošle serveru paket, ktorý obsahuje iba pole FLAGS a nastavený flag ANSWER, čo značí **žiadosť klienta o zaslanie odpovede**:

- UKONČENIE PRENOSU SPRÁVY/MESSAGE

Ukončenie prenosu ak všetko prebehlo v poriadku **inicializuje strana servera** a to zaslaním paketu, ktorý obsahuje iba pole FLAGS:

Ak jedna **strana neodpovedá, môžu obe strany ukončiť spojenie** zaslaním paketu s polom FLAGS:

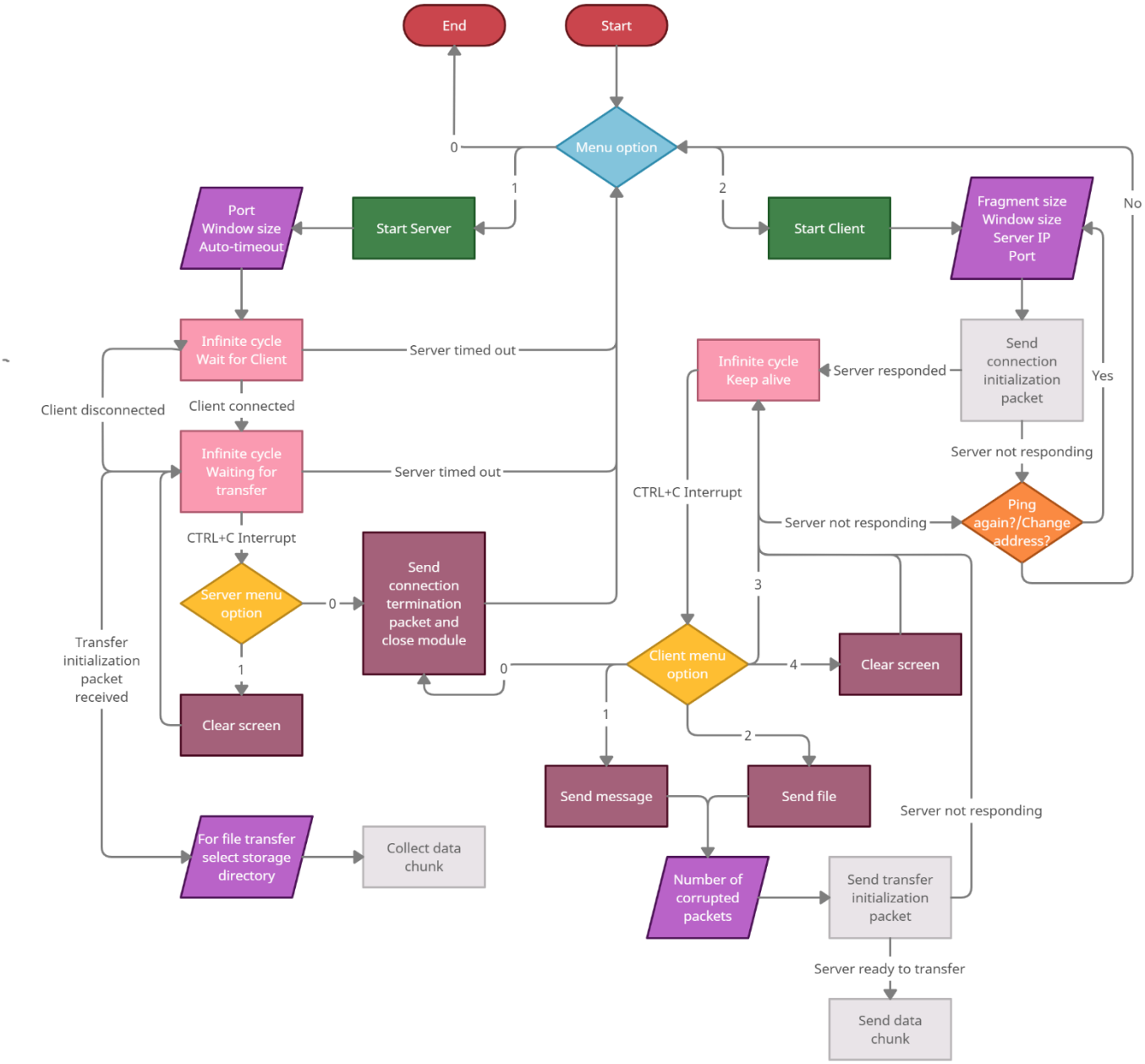
V tomto prípade je okrem flagu END nastavený aj flag NACK, čo značí, že **komunikácia neprebehla korektne**, ale bola predčasne ukončená.

Na ukončenie prenosu **druhá strana neodpovedá**.

Diagram toku programu

- ZÁKLADNÝ TOK PROGRAMU

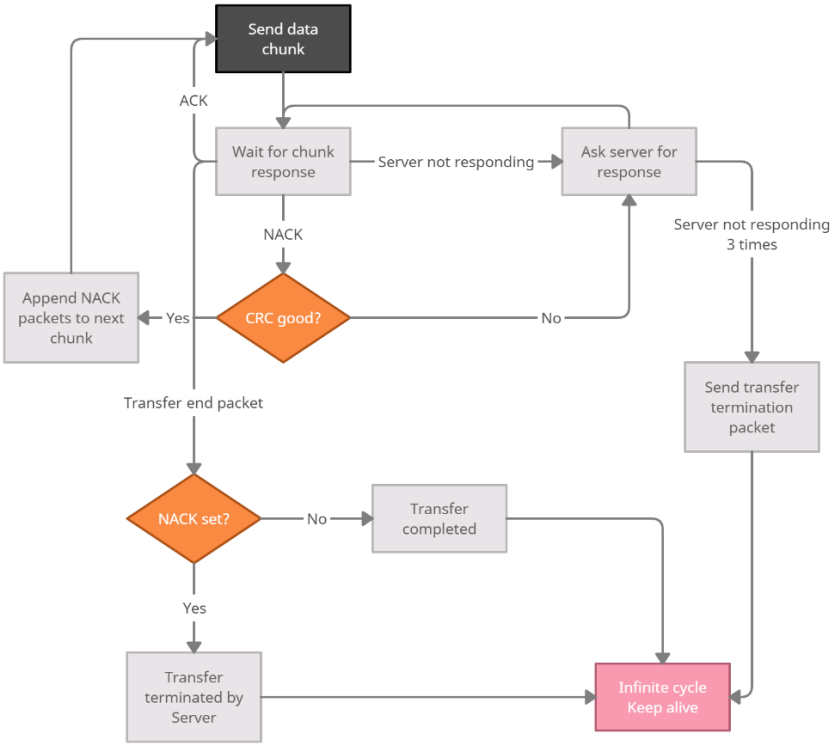
Prvá časť diagramu ukazuje spustenie programu, výber modulu, otvorenie, udržiavanie, ukončenia spojenia a jednotlivé možnosti menu pre dané moduly. Taktiež je ešte načrnutý začiatok prenosu správy/súboru.



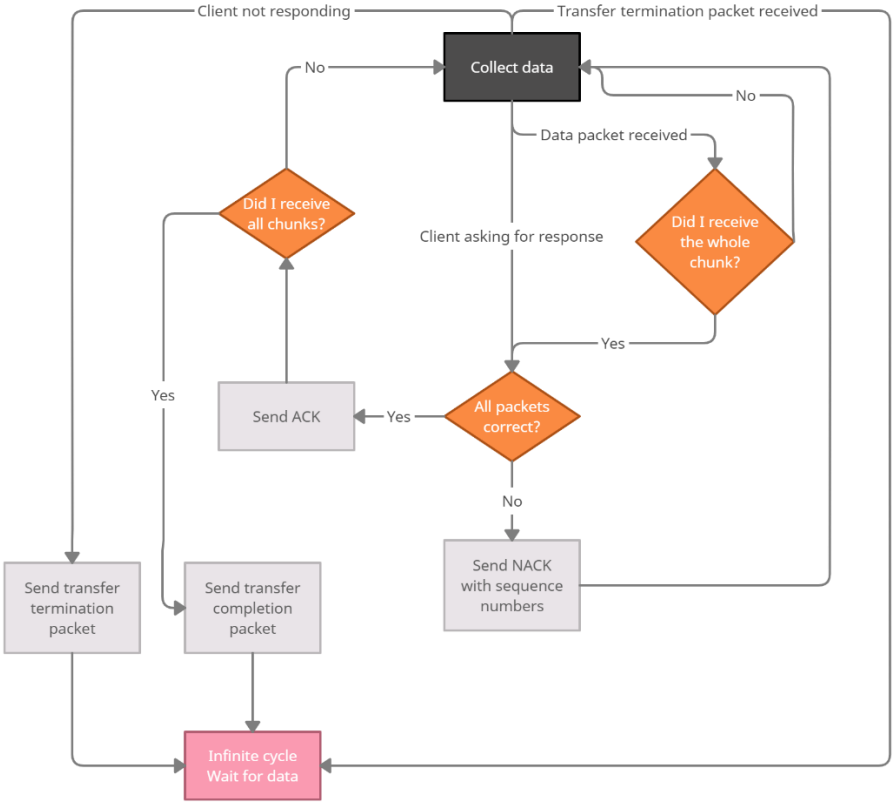
V diagrame vyššie na začiatku prenosu strana klient ešte zadáva správu alebo cestu k súboru, ktorý sa má poslať.

Diagram pokračuje na ďalšej strane, kde je ukázaný tok programu pri prenose.

- TOK PROGRAMU PRI PRENOSE – KLIENT

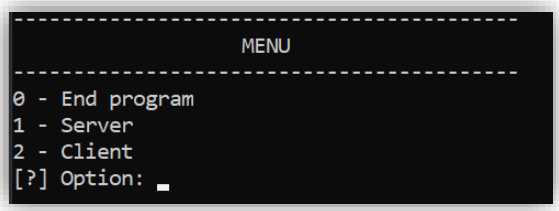


- TOK PROGRAMU PRI PRENOSE – SERVER



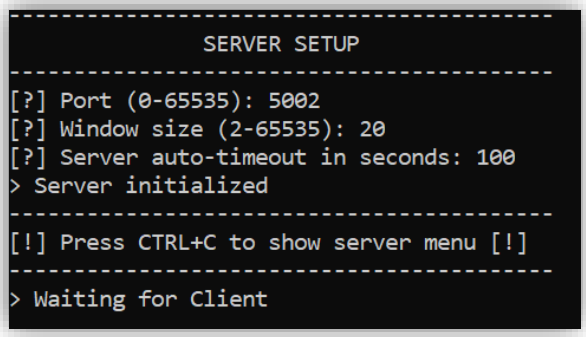
POUŽÍVATEĽSKÉ ROZHRANIE

Pri spustení programu sa používateľovi zobrazí hlavné menu a čaká sa na vybratie možnosti: modul server, modul klient alebo ukončenie programu.

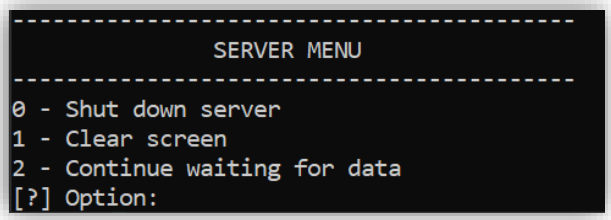


Server GUI

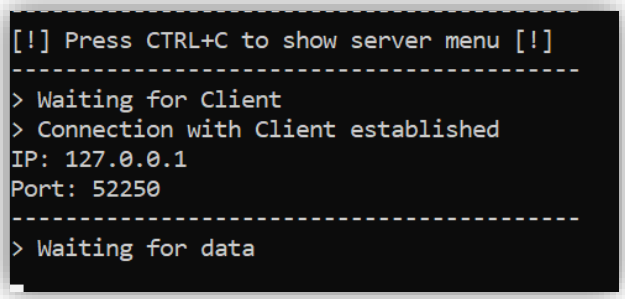
Pri vybratí modulu Server si program od používateľa vypýta vstupné informácie - port, veľkosť okna, ktorá označuje maximálny počet paketov, ktoré server môže obdržať v jednej dávke a časový limit automatického ukončenia servera ak neobdrží žiadne pakety:



Vypíše sa správa o inicializácii servera a čaká sa na klienta. Používateľ má možnosť prerušiť prijímanie dát a zobraziť menu servera stlačením kláves CTRL+C, kde má možnosť ukončiť server, vyčistiť plochu (konzolu) alebo pokračovať v prijímaní dát:



Po pripojení klienta sa vypíše správa o jeho IP adrese a porte:



Po prijatí žiadosti o prenos sa vypíše správa, že sa budú prenášať dáta a informácie spojené s daným prenosom. Ak ide o súbor, tak sa od používateľa vyžiada adresár, kde má byť súbor uložený:

```
----------------------------------------
> File transfer incoming
> File name: archive.zip
[?] Storage directory path: .
> Number of fragments: 519
> Max size of a fragment: 1463
```

Následne sa budú vypisovať informácie o prijatých paketoch a dávkach. Na záver sa vypíše súhrná informácie o prenose:

```
> Packet #517 ACK, Data size: 1463
> Packet #518 ACK, Data size: 1250
> Packet #488 NACK
> Packet #489 NACK
> Packet #490 NACK
----------------------------------------
> Chunk #30 total ACK: 8
> Chunk #30 total NACK: 3
----------------------------------------
> Packet #488 ACK, Data size: 1463
> Packet #489 ACK, Data size: 1463
> Packet #490 ACK, Data size: 1463
> Chunk #31 total ACK: 3
> Chunk #31 total NACK: 0
----------------------------------------
> Transfer completed
> All packets transferred
> Total ACK: 519
> Total NACK: 55
> Total data size: 759084 bytes
----------------------------------------
> File name: archive.zip
> Absolute path: /home/kesuera/python/archive.zip
----------------------------------------
```

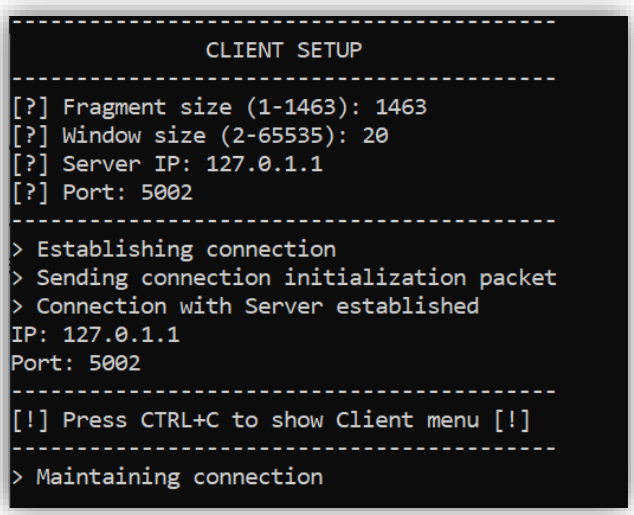
Výpis pri prenose správy:

```
> Message transfer incoming
> Number of fragments: 1
> Max size of a fragment: 1463
----------------------------------------
> Packet #0 ACK, Data size: 20
> Chunk #0 total ACK: 1
> Chunk #0 total NACK: 0
----------------------------------------
> Transfer completed
> All packets transferred
> Total ACK: 1
> Total NACK: 0
> Total data size: 20 bytes
----------------------------------------
> Message: Ahoj, pridi na kavu.
----------------------------------------
> Waiting for data
```

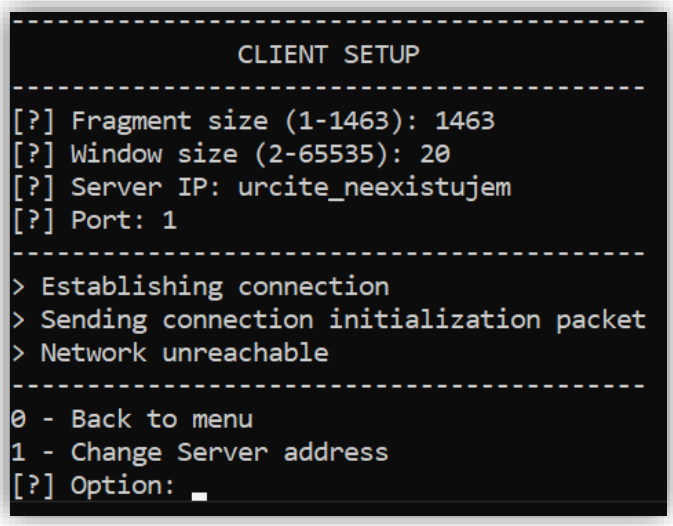
Ak sa klient odpojí alebo vyprší časový limit pripojenia, vypíše sa o tejto skutočnosti správa. V prípade vypršania časového limitu sa program vráti naspäť do hlavného menu, v prípade odpojenia klienta sa čaká na znovunapojenie klienta až do vypršania časového limitu.

Klient GUI

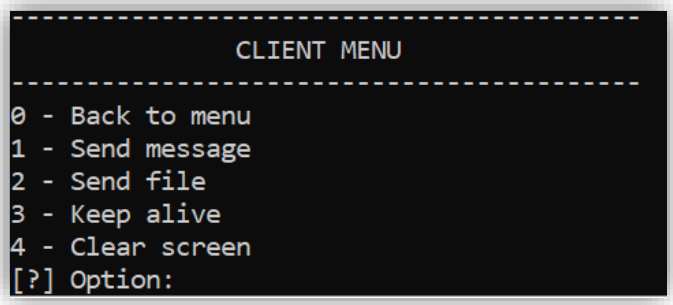
Po vybratí modulu Klient si program od používateľa vypýta vstupné informácie – veľkosť fragmentov, veľkosť okna, IP adresa servera a jeho port. V prípade korektného pripojenia na server sa vypíše správa:



V prípade, že je sieť nedostupná alebo nedostane žiadnu odpoveď, môže si vybrať možnosť zmeniť adresu servera alebo sa vrátiť späť do hlavného menu:

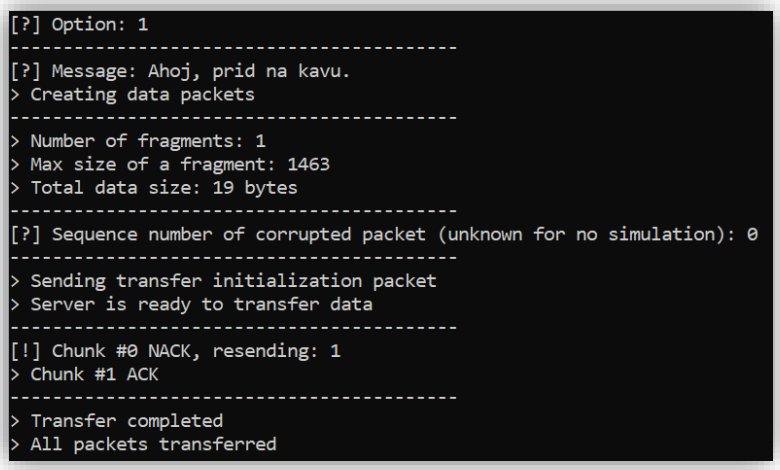


Po pripojení na server program udržiava spojenie. To môže používateľ opäť narušiť stlačením kláves CTRL+C, čím sa zobrazí menu klienta, kde má možnosť vrátiť sa do hlavného menu (ukončiť spojenie), poslať správu, poslať súbor, pokračovať v udržiavaní spojenia alebo vyčistiť plochu (konzolu):

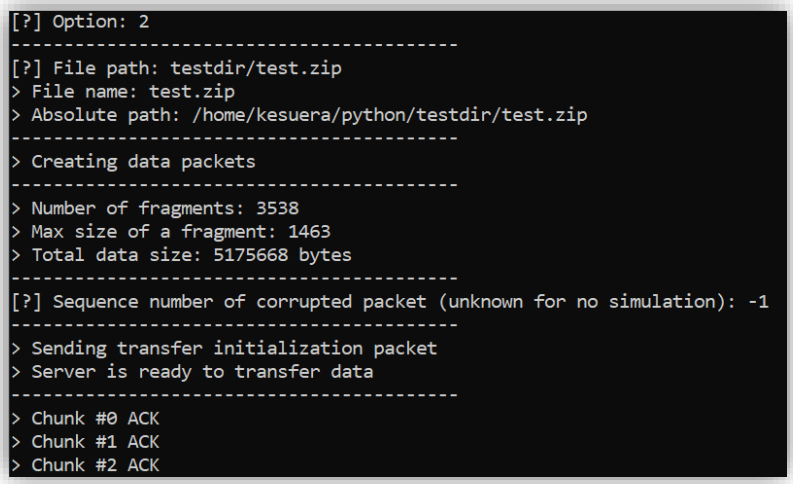


Pri vybratí možnosti poslať správu ďalej používateľ zadá text správy, pri vybratí možnosti poslať súbor zadáva cestu k súboru (obsahuje aj kontrolu či existuje), nakoniec zadáva poradové číslo paketu, ktorý sa má poslať poškodený. Následne ak server potvrdí začiatok prenosu vypisujú sa informácie o odpovediach servera a na záver správa o ukončení prenosu.

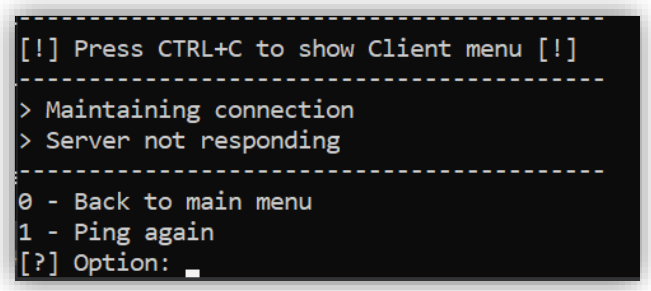
Prenos správy:



Prenos súboru:



V prípade, že pri udržiavaní spojenia server prestane odpovedať, vypíše sa správa a program sa používateľa spýta či má server pingnúť ešte raz alebo sa vrátiť do hlavného menu:



Toto bol základný opis používateľského rozhrania. Okrem týchto informačných správ obsahuje program ešte aj upozornenia, ak je zadaný vstup nekorektný a upozornenia ak dôjde k chybe pri prenose.

IMPLEMENTÁCIA

V tejto časti opíšem implementačné prostredie, ARQ metódu, ktorú využívam v tomto programe, metódu CRC, maximálnu veľkosť fragmentu a zdrojový kód.

Zmeny oproti návrhu

Jediná zmena, ktorá sa v programe udiala je tá, že používateľ si nevolí počet paketov, ktoré chce poslať poškodené, ale volí si jeden konkrétny, ktorý chce poslať poškodený.

Implementačné prostredie

Pre vyriešenie daného problému som si zvolil programovací jazyk Python, konkrétne verziu Python 3.8.10. Tento jazyk som si vybral z toho dôvodu, že Python má jednoduchú syntax, ľahko zrozumiteľné knižnice a rieši alokáciu a uvoľnovanie pamäte.

ARQ metóda

Na zabezpečenie plynulého a rýchleho prenosu som si vybral metódu **Selective-repeat ARQ**, ktorú som trochu upravil – server neposiela odpoveď na každý paket, ale až po obdržaní celej dávky paketov.

Výhoda tejto metódy je, že klient nemusí čakať na potvrdenie jednotlivých paketov, ale môže poslať celú dávku paketov na základe veľkosti okna. Až po poslaní celej dávky čaká na potvrdenie. Server v tomto prípade zbiera pakety

až pokým neobdrží celú dávku. V prípade, že celá dávka príde v poriadku, pošle potvrdenie obdržania celej dávky paketov. V prípade, že niektorý z paketov príde poškodený/nepríde, tak neodmieta všetky nasledujúce pakety, ale prijíma ich a ukladá do buffera. Až po obdržaní celej dávky pošle naspäť negatívne potvrdenie spolu s poradovými číslami paketov, ktoré boli poškodené.

Problém môže nastať vtedy, ak sa niektorý z paketov stratí. V takom prípade server nevie, že klient už odoslal celú dávku. Server stále očakáva pakety, kým klient očakáva potvrdenie. Preto má klient nastavený časovač dokedy čaká na odpoveď servera. Po vypršaní tohto času pošle serveru paket so žiadosťou o odpoveď. Server po prijatí tejto žiadosti skontroluje, ktoré pakety mu chýbajú a pošle klientovi odpoveď.

Klient ukončuje prenos ak nedostane odpoveď na 3 žiadosti o odpoveď. Server ukončuje prenos po vypršaní časovaču. Ak všetko prebehlo v poriadku a server obdržal všetky pakety, pošle ukončovací paket prenosu klientovi.

CRC metóda

V programe využívam CRC-16 CCIT metódu s predefinovanou vyhľadávacou tabuľkou, ktorá obsahuje 256 hodnôt.

Keďže metóda je 16-bitová, tak výsledné číslo bude mať veľkosť 2B. Počiatočnou hodnotou v tejto implementácii je *0xFFFF*. Potom prichádza cyklus. V tomto cykle *sa hodnota CRC doplní o 8 núl na konci* a spraví sa *XOR* operácia *s hodnotou v tabuľke* na pozícii, ktorá sa vypočíta tak, že *hodnota CRC sa posunie doprava o 8 miest* – skráti sa na 8 bitov a spraví sa *XOR* operácia *s daným dátovým bytom* (keďže máme 256 hodnôt tak potrebujeme 8 bitové číslo). Na konci cyklu sa spraví *AND operácia výslednej hodnoty a hodnoty 0xFFFF*, aby výsledné číslo bolo 16-bitové.

```
crc = 0xFFFF
for byte in data:
    crc = (crc << 8) ^ table[(crc >> 8) ^ byte]
    crc &= 0xFFFF
return crc
```

Zdroj:
https://gist.github.com/oysstu/68072c44c02879a2abf94ef350d1c7c6#gistcomment-3943460

Maximálna veľkosť fragmentu

Na základe viacerých zdrojov som zistil, že odporúčaná maximálna veľkosť fragmentu, aby nenastala fragmentácia na linkovej vrstve je 1500B. Keďže najväčšia hlavička, ktorá sa využíva je hlavička dátového paketu (9B), tak po odpočítaní veľkosti IP hlavičky (20B), UDP hlavičky (8B) a mojej hlavičky (9B), nám zostáva veľkosť **1463**, čo je maximálna veľkosť fragmentu (dátovej časti), ktorú používateľ môže nastaviť.

Simulácia chyby

Simuláciu chyby som zabezpečil takým spôsobom, že používateľ má možnosť vybrať si jeden konkrétny paket, ktorý chce poslať poškodený. V tomto pakete nahradím prvý dátový byte znakom 'X' alebo znakom 'Y' ak prvý byte je znak 'X'.

Zdrojový kód

Na vyriešenie problému som využil procedulárne programovanie. Jediná objektovo-orientovaná vec v programe sú triedy Server a Client, ktoré neobsahujú žiadne metódy, ale iba uložené parametre. Samotné funkcie v programe možno rozčleniť do 3 skupín: funkcie patriace modulu Server, funkcie patriace modulu Klient a funkcie zdieľané oboma modulmi.

Pri spustení programu sa spustí hlavné menu a na základe voľby užívateľa sa ďalej spustí modul Server, modul Klient alebo sa ukončí program.

- KNIŽNICE
  - *socket* – práca so soketmi
  - *time* – časovanie odosielania paketov pre udžiavanie komunikácie
  - *struct* – práca s binárnymi dátami
  - *os* – systémové volanie pre vyčistenie plochy (konzoly)

- TRIEDA SERVER

Pri spustení servera sa v triede Server ukladajú informácie o sokete servera, veľkosti okna a po napojení klienta aj jeho IP adresa a port.

```
# Server
class Server:
    sock = None  # socket
    window_size = None  # window size defined by user
    client_address = None  # (client ip, port)
```

- TRIEDA CLIENT

Pri spustení klienta sa v triede Client ukladajú informácie o sokete klienta, adrese servera, veľkosti fragmentov a veľkosti okna (tá sa môže zmeniť ak veľkosť okna servera je menšia).

```
# Client
class Client:
    sock = None  # socket
    server_address = None  # (server ip, port)
    frag_size = None  # fragment size defined by user
    window_size = None  # window size defined by user
```

- FUNKCIE MODULU SERVER
  - *start_server()* – zabezpečuje spustenie modulu server, načítanie vstupných informácií a zároveň rieši prijímanie a odpovedanie na pakety súvisiace s otvorením, udržiavaním a ukončením komunikácie (odpojí server po vypršaní časového limitu), ako aj začiatok prenosu a menu servera
  - *process_transfer()* – spustená po prijatí inicializačného paketu začiatku prenosu a rieši prijímanie paketov a odpovedanie klientovi pri prenose a takisto rieši ukončenie prenosu

- FUNKCIE MODULU KLIENT
  - *start_client()* – zabezpečuje spustenie modulu klient, načítanie vstupných informácií, volá funkcie pre otvorenie, udržiavanie a ukončenie komunikácie a nachádza sa tu menu klienta, odkiaľ na základe požiadavky volá funkcie pre zabezpečenie prenosu
  - *control_connection(client, start, end)* – rieši otvorenie, udržiavanie a ukončenie komunikácie. V prípade, že je spustená s nastaveným parametrom *start*, tak odošle inicializačný paket pre otvorenie komunikácie a prijme odpoveď od servera. Ak je spustená s nastaveným parametrom *end*, tak odošle paket pre ukončenie spojenia. Pokiaľ nie je spustená so žiadnym z týchto parametrov, tak funkcia udržiava spojenie – každých 10 sekúnd odošle paket pre udržiavanie spojenia a 5 sekúnd čaká na odpoveď, v prípade, že server neodpovedá, opýta sa či ho má pingnúť znova alebo ukončiť spojenie
  - *send_data(data, data_type, file_name, client)* – spúšťa sa po voľbe užívateľa poslať správu/súbor, zabezpečuje zaslanie inicializačného paketu začiatku prenosu, zavolá funkciu pre fragmentovanie dát, vytvorenie dátových paketov a riadi priebeh prenosu – posiela pakety v dávkach, zbiera odpovede
  - *fragment_data(data, frag_size)* – zabezpečuje fragmentáciu dát na základe veľkosti fragmentov, vráti list s fragmentovanými dátami

- *create_data_packet(data, flags, seq_num, crc)* – vytvorí balík binárnych dát (dátový paket)

- ZDIEĽANÉ FUNKCIE

- *crc16(data)* – vráti vypočítanú hodnotu CRC pre dané dáta
- *get_{flag_name}_flag(flags)* – 8 funkcií, ktoré vytiahnu z daných flagov, konkrétny bit flagu, ktorý chceme zistiť, či je nastavený a vráti True alebo False

SPLNENÁ FUNKCIONALITA

- 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. ✅
- 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. ✅
- Program musí byť implementovaný v jazykoch C/C++ alebo Python s využitím knižníc na prácu s UDP socket, skompilovateľný a spustiteľný v učebniach. ✅
- Program musí pracovať s dátami optimálne (napr. neukladať IP adresy do 4x int). ✅
- Pri posielaní súboru musí používateľovi umožniť určiť cieľovú IP a port. ✅
- Používateľ musí mať možnosť zvoliť si max. veľkosť fragmentu. ✅
- Obe komunikujúce strany musia byť schopné zobrazovať:
  - a. názov a absolútnu cestu k súboru na danom uzle, ✅
  - b. veľkosť a počet fragmentov. ✅
- 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). ☑

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