Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológii

Daniel Sawa

PKS

Zadanie 2: Komunikácia s využitím UDP protokolu

2022/2023

Piatok 8:00

Ing. Miroslav Bahleda, PhD

Obsah

Zadanie ... 3
Návrh ... 4
Hlavička vlastného protokolu ... 5
Flag – ... 5
CRC – ... 6
Poradové číslo – ... 6
Počet fragmentov – ... 6
Veľkosť jedného fragmentu ... 6
Opis ARQ ... 7
Metódy pre udržanie spojenia ... 7
Diagram spracovávania komunikácie na oboch uzloch ... 8
Popis jednotlivých častí zdrojového kódu ... 9
Client ... 9
Server ... 9
Implementácia návrhu ... 10
Sender ... 10
Reciever ... 11
Komunikácia ... 11
Zmena oproti návrhu ... 12
Hlavička ... 12
Flag – ... 13
CRC ... 13
Funkcie na výpočet CRC ... 14
Veľkosť jedného fragmentu ... 15
Udržiavanie spojenia ... 15
Fragmentovanie ... 15
Zdrojový kód ... 16
Knižnice ... 16
Globálne premenné ... 16
Classy a funkcie ... 16
Zhodnotenie ... 18

Daniel Sawa 116288

Zadanie

Zadanie 2: **Komunikácia s využitím UDP protokolu**

Zadanie úlohy

Navrhnite a implementujte program s použitím vlastného protokolu nad protokolom UDP (User Datagram Protocol) transportnej vrstvy sieťového modelu TCP/IP. Program umožní komunikáciu dvoch účastníkov v lokálnej sieti Ethernet, teda prenos textových správ a ľubovoľného súboru medzi počítačmi (uzlami).

Program bude pozostávať z dvoch častí – vysielacej a prijímacej. Vysielací uzol pošle súbor inému uzlu v sieti. Predpokladá sa, že v sieti dochádza k stratám dát. Ak je posielaný súbor väčší, ako používateľom definovaná max. veľkosť fragmentu, vysielajúca strana rozloží súbor na menšie časti - fragmenty, ktoré pošle samostatne. Maximálnu veľkosť fragmentu musí mať používateľ možnosť nastaviť takú, aby neboli znova fragmentované na linkovej vrstve.

Ak je súbor poslaný ako postupnosť fragmentov, cieľový uzol vypíše správu o prijatí fragmentu s jeho poradím a či bol prenesený bez chýb. Po prijatí celého súboru na cieľovom uzle tento zobrazí správu o jeho prijatí a absolútnu cestu, kam bol prijatý súbor uložený.

Program musí obsahovať kontrolu chýb pri komunikácii a znovuvyžiadanie chybných fragmentov, vrátane pozitívneho aj negatívneho potvrdenia. Po zapnutí programu, komunikátor automaticky odosiela paket pre udržanie spojenia každých 5s pokiaľ používateľ neukončí spojenie ručne. Odporúčame riešiť cez vlastne definované signalizačné správy a samostatný thread.

Daniel Sawa 116288

Návrh

Cieľ zadania je navrhnúť a implementovať program s použitím vlastného protokolu nad protokolom UDP (User Datagram Protocol) transportnej vrstvy sieťového modelu TCP/IP. Implementujeme nad UDP keďže UDP je protokol bez spojenia, bez potvrdenia a nespoľahlivý.

UDP nedokáže:

• nezriaďuje spojenie pred prenosom dát

• nepotvrdzuje prijaté dáta

• nedeteguje straty

• nie je možnosť požadovať opakovanie prenosu dát

• negarantuje doručenie dát

• nezaručuje, že dáta sú prijímané v rovnakom poradí ako boli vyslané

• nemá mechanizmus na riadenie toku dát medzi koncovými uzlami resp. na riadenie zahltenia

TCP je spoľahlivý a dokáže to potvrdzovanie príjmu, časovačmi, číslovanie bajtov, riadenie toku a zahltenia, pohyblivé.

TCP časovače:

• zriadenie spojenia (connection establishment timer)

• oneskorenie ACK (delayed ACK timer)

• testovanie nulového okna (persistence timer)

• opakovanie prenosu (retransmission timer)

• Fin_Wait_2 časovač

• Time_Wait časovač (2MSL timer)

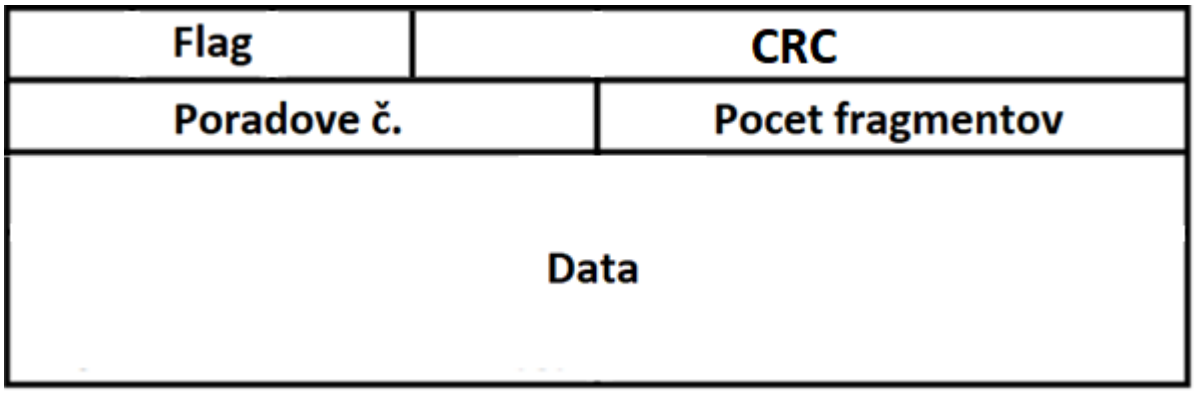
• test živosti spojenia (keepalive timer)

Náš program doplní TCP funkcionalitu do UDP s cieľom dosiahnutia rovnakej spoľahlivosti pri posielaní dát.

Daniel Sawa 116288

Hlavička vlastného protokolu

Pri návrhu našej hlavičky sa budeme inšpirovať TCP hlavičkou.



Flag – 1 B

CRC – 3 B

Poradové č. – 2 B

Počet fragmentov – 2 B

Flag –

bude obsahovať 1 B ktorý určí nasledujúce akcie v programe.

0 – SYN (pokus o nadviazanie spojenia)

1 – DATA (prenos dát)

2 – ACK (potvrdenie prijatia dát)

3 - NACK (potvrdenie prijatia dát ale pri kontrole CRC sa našla chyba)

4 - FIN (ukončenie spojenia)

5 – KA (Keep-Alive)

6 – SW (zmena clinet >< server)

Daniel Sawa 116288

CRC –

CRC slúži na kontrolu dát či nedošlo ku chybe pri prenose (ak nesedia bity ktoré prišli).

CRC funguje tak že dáta pred odoslaním vydelíme štandardizovaným polynómom a zvyšok po delení odošleme v CRC časti hlavičky. Po prijatí dát a zvyšku prijaté dáta delíme zvyškom a ak prišli dáta bez chyby očakávame zvyšok 0.

Ak dáta prišli bez chyby posielame naspäť nulu.

Zvolil som Pretty Good Privacy (PGP) polynóm

$$x^{24} + x^{23} + x^{18} + x^{17} + x^{14} + x^{11} + x^{10} + x^7 + x^6 + x^5 + x^4 + x^3 + x + 1$$

Keďže sme si zvolili deliteľa polynom dĺžky 24 bitov tak v hlavičke vyhradíme 3 bytes pre CRC kedže najväčší možný zbytok po delení je 24 bitové číslo. (24 bite/8 = 3 bytes)

Poradové číslo –

použijeme 2 B keďže pošleme najväčší súbor ktorý bude mať veľkosť 2MB čo je približne 2,100,000 B, a každý fragment bude obsahovať približne 1400 B.

2,100,000 / 1,400 = **1500 fragmentov**

Maximálne poradové číslo by malo byť približne 1500.

S **2 B** vieme reprezentovať hodnoty od **0** po **65,536**

Počet fragmentov –

bude taktiež 2 B keďže maximálny počet fragmentov bude 1500.

Veľkosť jedného fragmentu

Maximálne sa da poslať 1500 B pred tým ako dôjde ku fragmentácií na linkovej vrstve.

IP hlavička = 20 B

UDP hlavička = 8 B

Naša hlavička = 8 B

Veľkosť prenesených dát bude maximálne 1,500 – 20 – 8 – 8 = **1,464 B**

Daniel Sawa 116288

Opis ARQ

Zvolili sme si STOP-AND-WAIT ARQ protocol ktorý funguje nasledovne:

Po odoslaní jedného rámcu, odosielateľ čaká na potvrdenie (ACK) od prijímateľa pred tým ako pošle nasledujúci rámec.

Ak potvrdenie (ACK) nepríde do určitého času tak odosielateľ znova odošle rámec.

Čas na poslanie ACK som si vybral 1 sekunda (kedže Keep-Alive je 5 sekund).

Metódy pre udržanie spojenia

Po naviazaní spojenia budeme chcieť vedieť či spojenie ma byť udržané alebo ukončené. Spojenie môže byť ukončené správou s falgom FIN alebo tým že vyprší čas od posledného prijatia správy s flagom Keep-Alive.

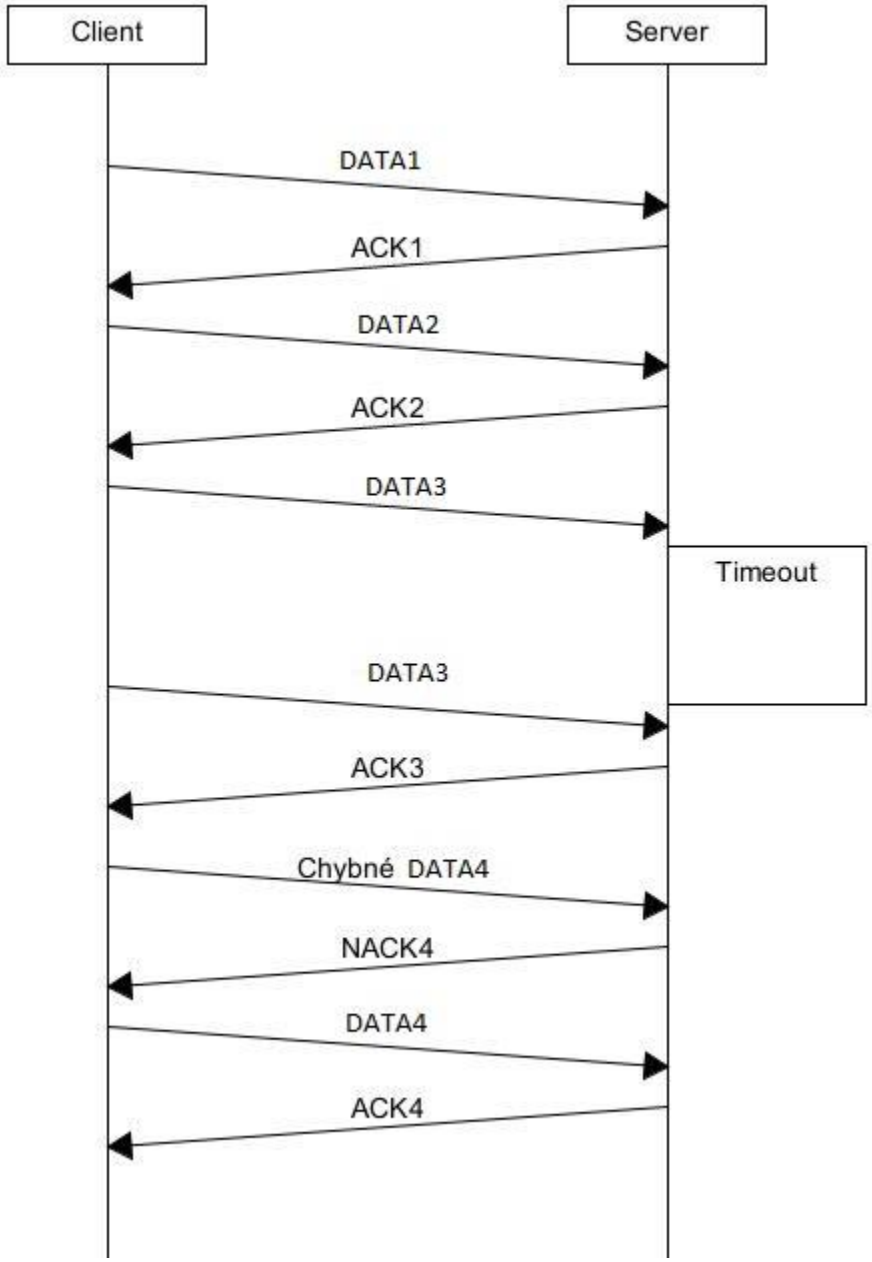
Keby nepríde sprava s flagom Keep-Alive tak by toto spojenie trvalo do nekonečna (dokým by sme nevypli celý program).

Tak ako TCP protokol budeme používať časovače pre udržane spojenia. Čo znamená že periodicky (v rovnakých časových intervaloch) budeme posielať správu serveru s flagom KA (Keep-Alive) aby sme udržali spojenie.

Keep-Alive správu budeme posielať každých 5 sekúnd.

Daniel Sawa 116288

Diagram spracovávania komunikácie na oboch uzloch



Znázornenie posielania dát s potvrdením

bez Keep-alive a vytvorenia a ukončenia komunikácie **pre prehľadnosť**.

Daniel Sawa 116288

Popis jednotlivých častí zdrojového kódu

Program budeme písať v jazyku Python, komunikáciu medzi zariadeniami budeme uskutočňovať pomocou knižnice socket.

Program sa bude skladať z tried Server a Client ktoré budú mať svoje funkcie a premenné (keby chceme mať viacero Clientov ktorý by posielali správy na Server).

Client a Server budú mať funkciu na:

- vytvorenia spojenia cez 3 way handshake (SYN,SYN ACK,ACK) a ukončenie spojenia (FIN,FIN ACK, ACK)

- sledovanie počtu a ktoré rámce boli úspešne doručene bez chyby.

- pri zmene role na opačnú rolu (client -> server , server -> client) sa odošle ramec s flagom SW

Client

bude mať funkciu na :

- fragmentovanie (aby sme náš súbor rozdelili na 1464 B časti)

- CRC kde sa vytvorí zvyšok po delení dát.

- Keep-Alive funkciu

- vyhradenie bajtov podľa hlavičky

- odosielanie rámca

- časovač ktorý bude očakávať potvrdenie prijatia rámca (ACK)

Server

bude mať funkciu na :

- CRC kde sa skontroluje či sa nevyskytujú chyby. Ak sa nachádza chyba tak sa odošle NACK

- odosielanie ACK ak prišiel rámec bez chýb

- ak prídu všetky rámce tak ukončíme komunikáciu

Daniel Sawa 116288

Implementácia návrhu

Možnosti sa vždy nachádzaju pred „ -> “ šípkou

Na začiatku ma používateľ možnosť rozhodnúť sa či bude:

S – sender

R-reciever

Sender

Po vybratí programu sender treba zadať ip adresu a port kde sa budú súbory alebo správy odosielať.

Následne vyberieme maximálnu veľkosť fragmentu, maximálne sa da poslať fragment veľkosti 1452 bajtov.

Následne zadáme „q“ na ukončenie programu, „swap“ na zmenu rolí, cestu súboru ktorú chceme poslať alebo „správa: tu napíšeme svoju správu“. (vypíšu sa nám aj súbory ktoré sa nachádzajú v súbore kde beží program).

Sedner nastavujeme po nastavení recievera keďže Sender posiela prvú správu

```
Sender / Reciever
--------------------------------------------------------------------------------
s/r -> s
Zvolte ip adresu a port (localhost:  127.0.0.1 20001 )     reciever dava len 0 a port
--------------------------------------------------------------------------------
 -> 127.0.0.1 20001
['127.0.0.1', 20001]
Zvolte max velkost fragmentu v B (mensie rovne ako 1452 na lan)
--------------------------------------------------------------------------------
 -> 1452
Zvolte subor ktory budete odosielat alebo zadajte absolutnu cestu
--------------------------------------------------------------------------------
2_zadanie_PKS_ZS2223.pdf
7_2020_Subnetting.pdf
client.py
flags.py
main.py
server.py
SNW.jpg
text.txt
Whitney Shafer - All My Exes Live In Texas.mp3
--------------------------------------------------------------------------------
Na ukoncenie komunikacie napise "q" alebo "swap"
q/swap/cesta/sprava -> text.txt
```

Daniel Sawa 116288

Reciever

Po vybraní programu reciever treba zadať 0 a port na ktorom bude komunikácia prebiehať.

Následne vyberieme cestu kde chceme uložiť súbor alebo napíšeme „swap“ alebo stlačíme enter ak chceme prijímať správy.

Po tomto sa spustí komunikácia. Nastavujeme vždy reciever prvý keďže sender posiela prvú správu

```
Sender / Reciever
--------------------------------------------------------------------------------
s/r -> r
Zvolte ip adresu a port (localhost:  127.0.0.1 20001 )     reciever dava len 0 a port
--------------------------------------------------------------------------------
 -> 0 20001
['0', 20001]
Zvolte cestu kde subor ulozite alebo swap alebo stlacte enter pre prijimanie sprav
--------------------------------------------------------------------------------
 -> C:\Users\dsawa\Desktop\algoritmy\Algoritmy\FIIT 2021\projekt iua\2022-2023\PKS zadani 2\folder pre kopirovanie
```

Komunikácia

Na začiatku komunikácie pošle Sender SYN následne sa mu vráti SYN od Recievera a Sender pošle ACK (3 way handshake). Po tomto začneme odosielať fragmenty s dátami.

Každý raz čo prídu DATA bez chyby odošleme ACK s čislom DATA ktoré potvrdzujeme.

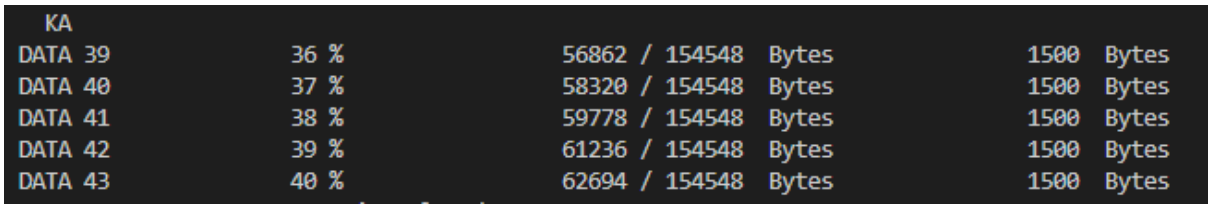
Ak prídu chybné dáta alebo nesprávne číslo dát pošleme NACK s číslom DATA ktoré chceme.

```
 SYN
------------------> Conection created <------------------
We will send 106 fragments
 ACK 1          0 %          1458 / 152136  Bytes          1500  Bytes
 ACK 2          1 %          2916 / 152136  Bytes          1500  Bytes
 ACK 3          2 %          4374 / 152136  Bytes          1500  Bytes
NACK 4
 ACK 4          3 %          5832 / 152136  Bytes          1500  Bytes
NACK 5
 ACK 5          4 %          7290 / 152136  Bytes          1500  Bytes
 ACK 6          5 %          8748 / 152136  Bytes          1500  Bytes
 ACK 7          6 %          10206 / 152136  Bytes          1500  Bytes
 ACK 8          7 %          11664 / 152136  Bytes          1500  Bytes
 ACK 9          8 %          13122 / 152136  Bytes          1500  Bytes
NACK 10
 ACK 10          9 %          14580 / 152136  Bytes          1500  Bytes
 ACK 11          10 %          16038 / 152136  Bytes          1500  Bytes
 ACK 12          11 %          17496 / 152136  Bytes          1500  Bytes
 ACK 13          12 %          18954 / 152136  Bytes          1500  Bytes
 ACK 14          13 %          20412 / 152136  Bytes          1500  Bytes
 ACK 15          14 %          21870 / 152136  Bytes          1500  Bytes
 ACK 16          15 %          23328 / 152136  Bytes          1500  Bytes
 ACK 17          16 %          24786 / 152136  Bytes          1500  Bytes
NACK 18
 ACK 18          16 %          26244 / 152136  Bytes          1500  Bytes
```

```
 SYN
 ACK
------------------> Conection created <------------------
DATA 1          0 %          1458 / 154548  Bytes          1500  Bytes
DATA 2          1 %          2916 / 154548  Bytes          1500  Bytes
DATA 3          2 %          4374 / 154548  Bytes          1500  Bytes
DATA 4          3 %          5832 / 154548  Bytes          1500  Bytes
DATA 5          4 %          7290 / 154548  Bytes          1500  Bytes
DATA 6          5 %          8748 / 154548  Bytes          1500  Bytes
DATA 7          6 %          10206 / 154548  Bytes          1500  Bytes
DATA 8          7 %          11664 / 154548  Bytes          1500  Bytes
DATA 9          8 %          13122 / 154548  Bytes          1500  Bytes
DATA 10          9 %          14580 / 154548  Bytes          1500  Bytes
DATA 11          10 %          16038 / 154548  Bytes          1500  Bytes
DATA 12          11 %          17496 / 154548  Bytes          1500  Bytes
DATA 13          12 %          18954 / 154548  Bytes          1500  Bytes
DATA 14          13 %          20412 / 154548  Bytes          1500  Bytes
DATA 15          14 %          21870 / 154548  Bytes          1500  Bytes
DATA 16          15 %          23328 / 154548  Bytes          1500  Bytes
DATA 17          16 %          24786 / 154548  Bytes          1500  Bytes
DATA 18          16 %          26244 / 154548  Bytes          1500  Bytes
DATA 19          17 %          27702 / 154548  Bytes          1500  Bytes
DATA 20          18 %          29160 / 154548  Bytes          1500  Bytes
DATA 21          19 %          30618 / 154548  Bytes          1500  Bytes
DATA 22          20 %          32076 / 154548  Bytes          1500  Bytes
```

Môže nastať situácia kedy nepríde ACK od recievera tak sender po dvoch sekundách pošle DATA znova

Popri odosielaní dat sa na vlastnom threade odosiela aj KA čo je keep alive.

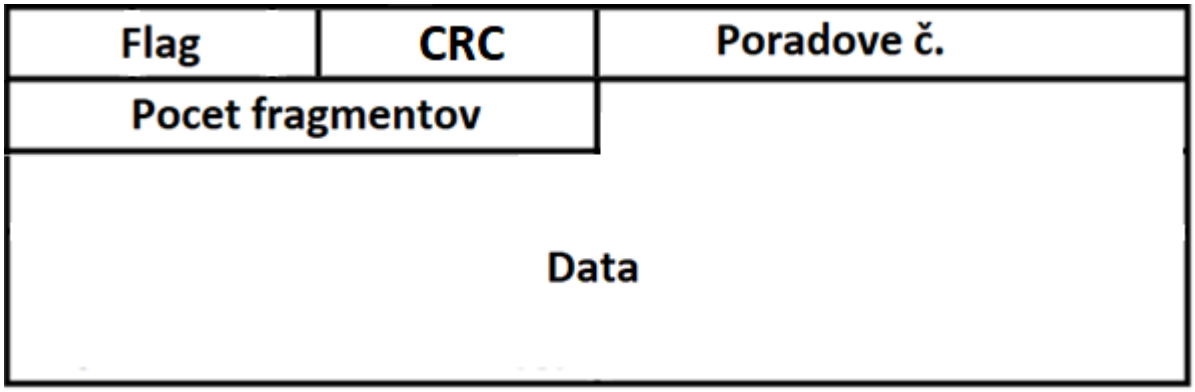


Po ukončení každej komunikácie a aj pri swape nám vypíše menu na výber Sender a Reciever.

Najprv musíme zmeniť pri Recieverovi či chceme swap spravu alebo cestu, potom môžeme nastaviť Sendera.

Zmena oproti návrhu

Hlavička



Flag – 1 B

CRC – 1 B

Poradové č. – 2 B

Počet fragmentov – 2 B

Daniel Sawa 116288

Flag –

bude obsahovať 1 B ktorý určí nasledujúce akcie v programe.

0 – SYN (pokus o nadviazanie spojenia)

1 – DATA (prenos dát)

2 – ACK (potvrdenie prijatia dát)

3 - NACK (potvrdenie prijatia dát ale pri kontrole CRC sa našla chyba)

4 - FIN (ukončenie spojenia)

5 – KA (Keep-Alive)

6 – SW (zmena clinet >< server)

7 – MSG (Posielanie textu)

Oproti návrhu som pridal Flag na posielanie textu keďže pracujeme inak s DATA ako s MSG.

CRC –

Po skúšaní kratších polynómov sme dostali rovnako dobré výsledky pri kontrole bitov v dátach tak som si zvoil polynom CRC-3-GSM.

$$x^3 + x + 1$$

Tým pádom budeme potrebovať len jeden bajt pri prenose zvyšku po delení.

Daniel Sawa 116288

Funkcie na výpočet CRC

```
def xor(a, b):
    result = []
    for i in range(1, len(b)):
        if a[i] == b[i]:
            result.append('0')
        else:
            result.append('1')
    return ''.join(result)


def mod2div(divident, divisor):
    pick = len(divisor)
    tmp = divident[0 : pick]

    while pick < len(divident):

        if tmp[0] == '1':
            tmp = xor(divisor, tmp) + divident[pick]

        else:
            tmp = xor('0'*pick, tmp) + divident[pick]

        pick += 1
    if tmp[0] == '1':
        tmp = xor(divisor, tmp)
    else:
        tmp = xor('0'*pick, tmp)
    checkword = tmp
    return checkword


def encodeData(data, key):
    data_bits = "".join([access_bit(data,i) for i in range(len(data)*8)])
    data_bits = data_bits[::-1]
    l_key = len(key)

    appended_data = data_bits + '0'*(l_key-1)
    appended_data = create_error(appended_data) # <------------------------
    remainder = mod2div(appended_data, key)

    return remainder
```

```
def decodeData(data, crc, key):
    data_bits = "".join([access_bit(data,i) for i in range(len(data)*8)])
    data_bits = data_bits[::-1]
    appended_data = data_bits + crc
    remainder = mod2div(appended_data, key)

    return remainder
```

Daniel Sawa 116288

Veľkosť jedného fragmentu

Po testovaní na LAN sieti som sa dozvedel že ip hlavička má veľkosť 34 bajtov takže môžeme odosielať 1452 bajtov dát.

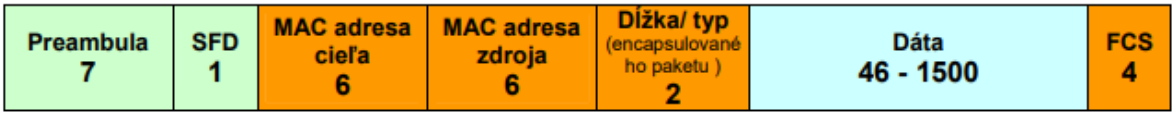
IP hlavička = 34 B

UDP hlavička = 8 B

Naša hlavička = 6 B

Veľkosť prenesených dát bude maximálne 1,500 – 34 – 8 – 6 = **1,452 B**

Ukážka štruktúry rámca Ethernet II



Udržiavanie spojenia

Spojenie sa môže ukončiť len tým že vyprší keep alive na jednom z programov a následne vyprší aj na druhom. Keep alive vyprší tým spôsobom že sa prestanú medzi sebou odosielať KA správy. FIN flag len oznamuje že bol odoslaný posledný rámec.

Fragmentovanie

Súbor rovnako ako text bude rozdelený na rovnaké veľkosti bajtov, veľkosť je zvolená používateľom.

Súbor rozdeľujeme pomocou funkcie read() v ktorej sa nachádza veľkosť bajtov a text rozdeľujeme po násobkoch zvolenej veľkosti následne zakódujeme a odošleme.

Daniel Sawa 116288

Zdrojový kód

Knižnice:

```
import socket
import os
import math
import threading
import time
import random
import ntpath
from flags import flags
```

Socket – nám slúži na komunikáciu na sieti.

OS -na vypisanie súborov v priečinku a zistenie velkosti súboru.

Math - na matematické funkcie.

Threading – na vytvorenie threadu

Time – na meranie času

Random – random

Ntpath – na vypisanie nazvu súboru z cesty

Flags – je to súbor kde je dict s našími Flagmi (ACK,SYN....)

Globálne premenné

Podstatná globálna premenná je len key kde sa nachádza náš CRC polynóm vo forme bitov.

Ostatné sú

Cesta súboru ktorý odosieláme/prijímame

Maximálna veľkosť fragmentu

Role, kde bude rola Sender/Reciever

IP a Port

Classy a funkcie

Celý program sa delí na classy Sender a Reciever.

Obidva classy maju funkciu `def start(self):`

Sender – pošle prvý správu a vykoná sa 3 way handshake (SYN,SYN,ACK) následne sa spustí keep alive thread a funkcia send_message alebo send_file.

Reciever- prvý prijíma správu a tiež vykoná handshake následne sa spustí keep alive thread a vykoná funkcia recieve_file.

Daniel Sawa 116288

Obidva maju funkcie

:`def close(self):`

Ktorá vypne thread a ukončí komunikáciu vtedy ked vyprší Keep alive čas.

`def keep_alive(self):`

Odpočítava časa je vo vlastnom threade.

`def header(self,byte,sum_frag,frag,crc,flag):`

Vytvorí a vráti 6 bajtovú hlavičku pomocou dát čo do nej vložíme.

`def handshake(self,address):`

Vykoná 4 way handshake a ukončí život tejto klasy (na konci kódu sa nachádza while loop ktorý vykoná role.start() takže takto udržujeme spojenie). Takže za každým ked odošléme správu alebo text tak sa nám vytvorí nový Sender/Reciever.

Funkcie ktoré ma iba Sender

`def swap(self):`

Slúži na výmenu rolí.

`def send_message(self,address):` a `def send_file(self,address):`

Odosielanie suboru alebo textu

Funkcie ktoré ma iba Reciever

`def recieve_file(self,address):`

Okrem toho su funkcie:

`def sender_funkcia():`

Kde zvolíme veľkosť fragmentu v bajtoch a cestu súboru ktorý budeme odosielať, ukončenie programu, swap alebo správu.

Daniel Sawa 116288

```
def reciever_funkcia():
```

Kde zvolíme cestu swap alebo správu.

```
def bitstring_to_bytes(s):
```

Meníme bity ktoré sú vo forme stringu na bajty.

```
def access_bit(data, num):
```

Meníme bajty na bity

```
def create_error(data):
```

Meníme náhodne bity na hodnotu 1.

```
def encodeData(data, key):
```

Vypočítavame zvyšok po delení polynómom (CRC)

```
def decodeData(data, crc, key):
```

Vypočítavame zvyšok po delení zvyškom (CRC)

Zhodnotenie

Náš program nám umožňuje odosielať a prijímať správy a súbory s informáciami o tom koľko % a koľko bajtov sa prenieslo.

Môžeme si zvoliť a meniť role ktoré sú odosielateľ a prijímateľ dát.

Môžeme si zvoliť veľkosť fragmentovania dát.

Náš program nás informuje o stave jednotlivých rámcov, napríklad či boli doručené správne a ktorý v poradí bol odoslaný.