Slovenská technická univerzita

Fakulta informatiky a informačných technológií

Ilkovičova 2, 842 16 Bratislava 4

**Komunikácia s využitím UDP protokolu**

**Počítačové a komunikačné siete**

**Autor:** Martin Družbacký

**Cvičiaci:** Ing. Kristián Košťál, PhD.

**Cvičenie:** UT 18:00

**Rok:** 2022/2023

**Obsah**

[**Zoznam obrázkov** 3](#_Toc121766254)

[**Vlastná hlavička** 4](#_Toc121766255)

[**FLAG** 4](#_Toc121766256)

[**NUMBER** 4](#_Toc121766257)

[**CRC** 4](#_Toc121766258)

[**Maximálna veľkosť pre dáta** 7](#_Toc121766259)

[**Diagram programu** 8](#_Toc121766260)

[**Flowchart** 8](#_Toc121766261)

[**Sekvenčný diagram** 9](#_Toc121766262)

[**Metóda ARQ** 10](#_Toc121766263)

[**Metóda udržania spojenia** 10](#_Toc121766264)

[**Simulácia chyby** 11](#_Toc121766265)

[**Implementácia** 11](#_Toc121766266)

[**Používateľské rozhranie** 11](#_Toc121766267)

[**Úvodné menu** 11](#_Toc121766268)

[**Server GUI** 11](#_Toc121766269)

[**Prijatie správy** 12](#_Toc121766270)

[**Prijatie súboru** 13](#_Toc121766271)

[**Klient GUI** 13](#_Toc121766272)

[**Posielanie správy** 13](#_Toc121766273)

[**Posielanie súboru** 14](#_Toc121766274)

[**Keep Alive výpis** 14](#_Toc121766275)

[**Zdrojový kód** 14](#_Toc121766276)

[**Klient** 14](#_Toc121766277)

[**Server** 16](#_Toc121766278)

[**Vlastná hlavička** 17](#_Toc121766279)

[**Prevzaté CRC** 18](#_Toc121766280)

[**Keep Alive** 19](#_Toc121766281)

[**Zachytávanie komunikácie Wiresharkom** 19](#_Toc121766282)

[**Otvorenie komunikácie** 19](#_Toc121766283)

[**Výmena strán** 20](#_Toc121766284)

[**Posielanie suboru** 20](#_Toc121766285)

[**Ukoncenie komunikacie** 21](#_Toc121766286)

[**Záver** 22](#_Toc121766287)

# **Zoznam obrázkov**

[Obrázok 1 Ukážka Hlavičky 4](#_Toc121766320)

[Obrázok 2 Počítanie CRC 6](#_Toc121766321)

[Obrázok 3 Diagram programu 8](#_Toc121766322)

[Obrázok 4 Sekvenčný diagram programu 9](#_Toc121766323)

[Obrázok 5 https://www.javatpoint.com/stop-and-wait-protocol 10](#_Toc121766324)

[Obrázok 6 Úvodné menu 11](#_Toc121766325)

[Obrázok 7 Nastavovanie servera 11](#_Toc121766326)

[Obrázok 8 Vyberanie úložiska 11](#_Toc121766327)

[Obrázok 9 Prehodenie po odosielaní 12](#_Toc121766328)

[Obrázok 10 Prijatie správ 12](#_Toc121766329)

[Obrázok 11 Prijatie súboru 13](#_Toc121766330)

[Obrázok 12 Nastavenie klienta 13](#_Toc121766331)

[Obrázok 13 Menu klienta 13](#_Toc121766332)

[Obrázok 14 Posielanie správ 13](#_Toc121766333)

[Obrázok 15 Posielanie súboru 14](#_Toc121766334)

[Obrázok 16 Úkažka keep alive 14](#_Toc121766335)

[Obrázok 17 Klása pre klienta 14](#_Toc121766336)

[Obrázok 18 Nastavenie klienta 15](#_Toc121766337)

[Obrázok 19 Klient loop 15](#_Toc121766338)

[Obrázok 20 Trieda pre server 16](#_Toc121766339)

[Obrázok 21 Nastavenie servera 16](#_Toc121766340)

[Obrázok 22 Server loop 16](#_Toc121766341)

[Obrázok 23 Zostrojenie vlastnej hlavičky 17](#_Toc121766342)

[Obrázok 24 Prevzate CRC 18](#_Toc121766343)

[Obrázok 25 Keep Alive 19](#_Toc121766344)

[Obrázok 26 Otvorenie komunikácie 19](#_Toc121766345)

[Obrázok 27 Vymenenie strán 20](#_Toc121766346)

[Obrázok 28 Posielanie súboru 20](#_Toc121766347)

[Obrázok 29 Posielanie textu 21](#_Toc121766348)

[Obrázok 30 Ukončenie komunikácie 21](#_Toc121766349)

# **Vlastná hlavička**



Obrázok Ukážka Hlavičky

## **FLAG**

Flag je typ správy, ktoré ma klient a server rozoznávať a vykonávať akcie. Môže nadobudnúť hodnotu 0-9 preto je zapísaný ako 1 char čiže nadobudne veľkosť **1B** viď [Obrázok 1 Ukážka Hlavičky](#_Vlastná_hlavička) . Flag môže nadobudnúť tieto hodnoty:

* 0 – Keep alive
* 1 – SYN určuje otvorenie spojenia
* 2 – ACK určuje potvrdenie spojenia
* 3 – Error určuje chybné prijatie fragment
* 4 – Success určuje správne prijatie fragmentu
* 5 – Text určuje posielanie textu
* 6 – File určuje posielanie súboru
* 7 – Switch určuje vymenenie strán
* 8 – End určuje ukončenie komunikácie
* 9 – Last fragment určuje ukončenie posielania fragmentov

## **NUMBER**

Poradové číslo fragmentu pre textovú správu alebo súbor. Je to potrebné kvôli zostaveniu správy v prípade ak fragmentu prídu nezoradene. Keďže očakávame väčšie množstvo fragmentov, tak potrebujeme rezervovať aspoň **3B** pre poradové číslo. Toto by malo pokryť prenos 2MB súboru po 1B. Viď [Obrázok 1 Ukážka Hlavičky](#_Vlastná_hlavička).

## **CRC**

Ako metódu kontroly som zvolil CRC (Cyclic Redundancy Check), teda kontrolu cyklickým kódom. Funguje to na princípe delenia polynómov. Dopredu sa dohodne polynóm a následne sa delí pomocou operácie XOR. CRC, ktoré používam je **CRC-16/DDS-110**, ktoré používa polynóm **X16 + X15 + X2 + 1** (0x8005).

Ak sa CRC odoslanej správy bude rovnať prijatej správe, tak vieme usúdiť , že paket prišiel správne. Implementácia tohto algoritmu bude prevzatia (<https://stackoverflow.com/questions/35205702/calculating-crc16-in-python>).

Pre moje zadanie som sa rozhodol používať CRC16 keďže jeho výpočet je o veľkosť **2B**, čo pre naše účely úplne stačí. Viď [Obrázok 1 Ukážka Hlavičky](#_Vlastná_hlavička).

Príklad počítania CRC na reťazci „A“. Viď [Obrázok 2 Počítanie CRC](#_CRC)

Obrázok, na ktorom je stôl

Automaticky generovaný popis

Obrázok Počítanie CRC

## **Maximálna veľkosť pre dáta**

Maximálnu veľkosť dát ktoré môžeme posielať bez fragmentácie zistíme pomocou

ETHERNET II payloadu (1500) od , ktorého odčítame IP hlavičku (20), UDP hlavičku (8) a mnou navrhnutú hlavičku (6).

1500-20-8-6=**1466**

Čiže maximálna veľkosť bez toho aby packet bol rozfragmentovaný je **1466B**

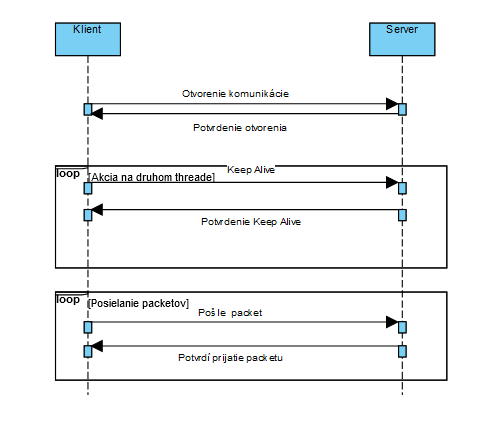
# **Diagram programu**

## **Flowchart**



Obrázok Diagram programu

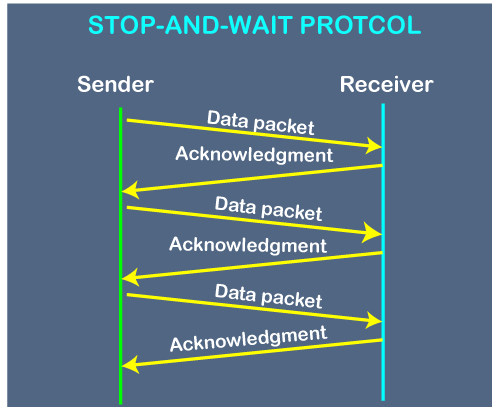
## **Sekvenčný diagram**



Obrázok Sekvenčný diagram programu

# **Metóda ARQ**

Ako metódu ARQ som sa rozhodol použiť Stop&Wait prístup. Funguje to na jednoduchom princípe. Klient pošle 1 packet a čaká na potvrdenie od serveru či packet prišiel správne. Ak prišiel správne tak odpovedá že packet prišiel v poriadku pokiaľ prišiel poškodený tak si znovu vyžiada ten packet. Tento spôsob môže prísť ako trochu neefektívny, ale vie zabezpečiť prenos všetkých packetov správne.



Obrázok https://www.javatpoint.com/stop-and-wait-protocol

# **Metóda udržania spojenia**

Po nadviazaní spojenia bude Klient posielať každých 5 sekúnd keep alive packet a bude čakať na odpoveď od Servera. Ak odpoveď nepríde 5krát po sebe tak na 6krát bude spojenie ukončené. Ak príde tak sa toto počítadlo vynuluje a cyklus pokračuje. Riešenie bude použitie druhého threadu, ktorý bude posielať KA packety. Je to kvôli tomu aby nevznikol konflikt s hlavným programom. Keď sa bude posielať text alebo súbor tak bude KA dočasne vypnute a po odoslaní sa znovu zapne.

# **Simulácia chyby**

Simuláciu chybu prezentujem v mojom kóde ako chybné CRC. Prebieha to tak že počas posielania si vyberiem 1 packet a pokazím mu CRC tak že jeho hodnotu navýšim o 1. Keď príde tento packet na server tak bude vypočítaná CRC hodnota na základe dám a keďže nebude zhodovať tak server tento packet na novo vyžiada.

# **Implementácia**

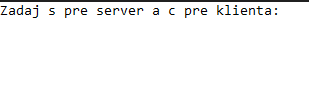
Pre implementáciu tohto zadania som sa rozhodol využiť python keďže je veľmi priateľský na túto problematiku. K vyriešeniu budem potrebovať tieto knižnice:

* socket – na vytvorenie klient/server
* os – zistovanie absolutnej cesty odoslaného súboru
* time – kvôli časovaniu odosielania packetov
* threading – využitie viacerých thredov

## **Používateľské rozhranie**

Používateľské rozhranie je implementované konzolovou formou. Používateľ zadáva textové vstupy na ovládanie.

### **Úvodné menu**



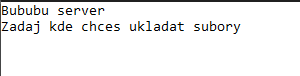
Obrázok Úvodné menu

### **Server GUI**

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok Nastavovanie servera



Obrázok Vyberanie úložiska



Obrázok Prehodenie po odosielaní

### **Prijatie správy**

Obrázok, na ktorom je stôl

Automaticky generovaný popis

Obrázok Prijatie správ

Tento príklad ukazuje prijatie packetu aj s chybou. Na výstupe sa zobrazí poradové číslo packetu a či bol chybný alebo bez chyby. Ako posledné zobrazí počet prijatých packetov a celkovú prijatú veľkosť. Čiže hlavička + data.

### **Prijatie súboru**

Obrázok, na ktorom je text, stôl

Automaticky generovaný popis

Obrázok Prijatie súboru

Prijatie súboru ma skoro rovnaké výpisy ako prijatie správy. Akurát ma navyše vypísanie absolútnej cesty k súboru a veľkosť daného súboru.

### **Klient GUI**

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok Nastavenie klienta

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok Menu klienta

### **Posielanie správy**

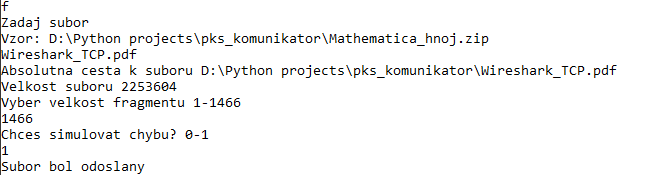
Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok Posielanie správ

Používateľ si zadá text, ktorý chce posielať vyberie veľkosť fragmentu, ktorou chce text posielať a v poslednom rade si vyberie či chce simulovať chybu.

### **Posielanie súboru**



Obrázok Posielanie súboru

Používateľ zadá názov (ak sa súbor nachádza v priečinku) alebo cestu k danému súboru. Potom si zadá veľkosť fragmentu po ktorom to chce posielať a v poslednom rade či chce simulovať chybu v prenose. Taktiež zobrazuje veľkosť a absolútnu cestu k súboru.

### **Keep Alive výpis**

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok Úkažka keep alive

Ukážka ako bol server dostupný a následne odpojený. Po piatich neúspešných pokusoch bude klient vypnutý.

## **Zdrojový kód**

### **Klient**

Klientska časť pozostáva z triedy Client, nastavenia klienta a klient loop

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok Klása pre klienta

Obrázok, na ktorom je text, vnútri, snímka obrazovky

Automaticky generovaný popis

Obrázok Nastavenie klienta

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok Klient loop

Z klienta posielam serveru packet pre otvorenie komunikácie a následne čakám na odpoveď. Ak náhodou odpoveď nepríde alebo nastane timeout tak to skúsim znovu. Keďže connection timeout môže nastať pri výmene strán.

### **Server**

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok Trieda pre server

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok Nastavenie servera

Obrázok, na ktorom je text, osoba, snímka obrazovky

Automaticky generovaný popis

Obrázok Server loop

Na strane servera musím využívať try except aby som si ošetril či sa už náhodou nebindujem na už nabindovaný socket.

### **Vlastná hlavička**

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok Zostrojenie vlastnej hlavičky

### **Prevzaté CRC**

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok Prevzate CRC

### **Keep Alive**

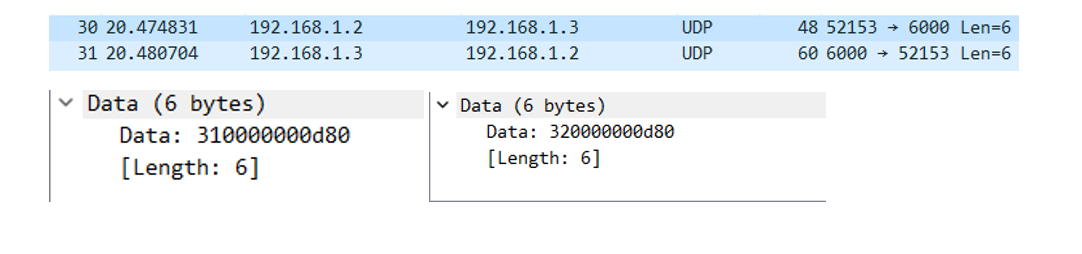
Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok Keep Alive

# **Zachytávanie komunikácie Wiresharkom**

### **Otvorenie komunikácie**



Obrázok Otvorenie komunikácie

### **Výmena strán**

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok Vymenenie strán

### **Posielanie suboru**

Obrázok, na ktorom je stôl

Automaticky generovaný popis

Obrázok Posielanie súboru

Veľkosť posielaného fragmentu bola nastavená na maximálnych 1466B avšak keď posielam názov súboru tak nefragmentujem daný packet. Pri ostatných packetoch už, ale fragmentáciu uplatňujem.

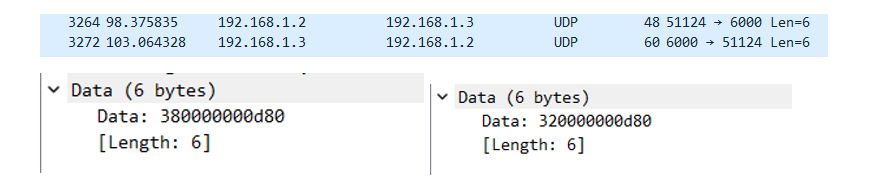
**Posielanie textu**

Obrázok, na ktorom je stôl

Automaticky generovaný popis

Obrázok Posielanie textu

### **Ukoncenie komunikacie**



Obrázok Ukončenie komunikácie

# **Záver**

Toto zadanie bolo celkom zaujímavé, ktoré však skôr testovalo programátorské schopnosti oproti tým sieťarským. To si myslím že je celkom zahodený potenciál. Avšak pri mojom riešení by sa dalo spraviť niekoľko vylepšení. Napríklad by sa dala implementovať ARQ metóda selective ARQ, ktorá by data posiela po chunkovach a stačilo by posielať jedno ACK alebo NACK. Týmto by sa proces prenášania dát zrýchlil keďže by sa nemuselo potvrdzovať prijatie každého jedného packetu ako je to pri stop&wait ARQ metóde. V mojom zadaní spĺňam všetky požadované body a myslím si že moje riešenie je na dobrej úrovní.