Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Počítačové a komunikačné siete

**Zadanie č.2**

**UDP komunikátor**

Akademický rok 2022/2023

**Meno:** Ján Ágh **Dátum:** 19.10.2022

**Cvičiaci:** Ing. Lukáš Mastiľak **Počet strán:** 9

**Obsah**

[**1 Stručný opis problematiky a zadania** 1](#_Toc121253513)

[**2 Navrhnuté riešenie** 2](#_Toc121253514)

[**2.1 Použité programové prostriedky** 2](#_Toc121253515)

[**2.2 Organizácia projektu** 2](#_Toc121253516)

[**2.3 Primárne vlastnosti riešenia** 3](#_Toc121253517)

[**2.3.1 Návrh hlavičky vlastného protokolu** 3](#_Toc121253518)

[**2.3.2 Opis spôsobu inicializácie a ukončenia komunikácie** 4](#_Toc121253519)

[**2.3.3 Opis metódy kontrolnej sumy (CRC)** 5](#_Toc121253520)

[**2.3.4 Opis použitej ARQ metódy** 6](#_Toc121253521)

[**2.3.5 Opis metódy na udržiavanie spojenia** 7](#_Toc121253522)

[**2.3.6 Opis spôsobu zmeny úloh** 7](#_Toc121253523)

[**2.4 Zmeny oproti návrhu** 8](#_Toc121253524)

[**3 Opis dôležitých častí programu** 9](#_Toc121253525)

[**3.1 Diagram priebehu programu** 9](#_Toc121253526)

[**3.2 Zabalenie a rozbalenie paketov** 10](#_Toc121253527)

[**3.3 Odosielanie a prijímanie dátových paketov** 11](#_Toc121253528)

[**3.4 Evaluácia prijatých signálov** 12](#_Toc121253529)

[**3.5 Systém doručovania keep-alive signálov** 13](#_Toc121253530)

[**4 Ukážka testovacieho scenáru** 14](#_Toc121253531)

# **1 Stručný opis problematiky a zadania**

Cieľom zadania je navrhnúť program umožňujúci komunikáciu dvoch zariadení v lokálnej sieti typu Ethernet s využitím vlastného protokolu nad protokolom User Datagram Protocol (UDP), ktorý pracuje na transportnej vrstve sieťového modelu TCP/IP.

Program sa skladá z dvoch častí, a to vysielacej (klient) a prijímacej (server), a umožňuje používateľovi preposielať jednoduché textové správy a súbory ľubovoľného formátu. Používateľ má taktiež možnosť zadefinovať si maximálnu veľkosť fragmentu, ktorá nesmie presiahnuť veľkosť 1500B, aby nedochádzalo k opätovnej fragmentácii na datalinkovej vrstve.

Program obsahuje kontrolu chýb pri komunikácii a znovuvyžiadanie chybných fragmentov, pričom na túto skutočnosť využíva signalizačné správy odosielané počas komunikácie. Počas doby trvania komunikácie jednotlivé strany v intervale každých 5 sekúnd odosielajú paket slúžiaci na udržiavanie spojenia druhej strane dovtedy, kým sa používateľ nerozhodne manuálne komunikáciu prerušiť alebo nevznikne nečakaná chyba.

# **2 Navrhnuté riešenie**

## **2.1 Použité programové prostriedky**

Na riešenie úlohy bol použitý programovací jazyk Python verzie 3.10.8. Riešenie bolo vyvíjané v prostredí Visual Studio Code verzie 1.72.1 a pri jeho vývoji boli využité základné objektovo orientované princípy vrátane dedenia.

## **2.2 Organizácia projektu**

Program pozostáva z niekoľkých základných súborov, ktoré sú nevyhnutné pre jeho funkčnosť. Každý zo spomínaných súborov plní istú úlohu počas vykonávania programu.

Prvým súborom je *interface.py* nachádzajúci sa v adresári *src.* Tento súbor obsahuje statickú triedu Interface, ktorá slúži na poskytovanie spätnej väzby používateľovi pomocou konzolových výpisov a taktiež akceptovanie vstupných informácií. Tvoria ju metódy *initial\_setup()* na prvotné nastavenie programu (výber možnosti klient – server), *initialize\_client()* a *initialize\_server()* na zobrazenie menu a akceptovanie vstupu patriaceho ku klientskej, resp. serverovskej časti programu a ďalšie, menej významné a pomocné metódy.

Dvojica súborov *client.py* a *server.py* v adresári *src* obsahujú triedy s konkrétnymi implementáciami príslušných častí programu – klientskej časti v triede *Client* a serverovskej časti v triede *Server*.

Súbor *general.py* a v ňom sídliaca trieda *General* predstavuje rodičovskú triedu pre *Client* aj *Server* a obsahuje dôležité metódy využívané oboma týmito triedami, ako napríklad inicializácia a rušenie spojenia, odosielanie a analýza paketov a mnoho ďalších.

V súbore *signal\_types.py* je umiestnená enumeračná trieda *Signals.py* so zadefinovanými konštantami predstavujúcimi typy signálov, ktoré dokážu jednotlivé strany pogramu prijímať, úspešne analyzovať a odosielať.

Posledným a zároveň najdôležitejším súborom je *main.py*, tvoriaci jadro celého programu a spájajúci zvyšné triedy a súbory do jedného celku. Ide o vstupný bod programu a v ňom sa vytvoria príslušné objekty tried *Server* a *Client*.

Text

Description automatically generated with medium confidence

*Obr. 2.1 Adresárová štruktúra implementácie*

## **2.3 Primárne vlastnosti riešenia**

Nadchádzajúca podkapitola vo veľkej miere obsahuje identické informácie, aké sa nachádzajú aj v dokumente návrhu programu, s menšími úpravami. Vykonané zmeny oproti návrhu nie sú príliš rozsiahleho charakteru.

### **2.3.1 Návrh hlavičky vlastného protokolu**

Súčasťou zadania je aj návrh vlastného protokolu, ktorý bude pracovať nad protokolom UDP. Keďže protokol UDP neodosiela žiadne potvrdenia týkajúce sa prijatia dát (a týmpádom druhá strana komunikácie nemá žiadny spôsob dozvedieť sa, či dáta prišli v poriadku a v správnom poradí), úlohou navrhnutého protokolu bude vyriešiť aj túto problematiku. Štruktúra protokolu je opísaná nižšie:

**1.** **Typ paketu** – informácia, o aký druh paketu sa jedná:

* **0** – paket *SYN*, označujúci začiatok nadväzovania spojenia
* **1** – paket *SYN+ACK,* odpoveď druhej strany na požiadavku spojenia
* **2 –** paket *ACK*, odpoveď na viacero požiadaviek (definitívne potvrdenie spojenia, odpoveď na keep-alive pakety, odpoveď na prijaté fragmenty…)
* **3** – paket *keep-alive* na overovanie konektivity druhej strany
* **4** – paket *FIN* na ukončenie spojenia
* **5** – paket *FIN+ACK* na odpoveď k požiadavke ukončenia spojenia
* **6** – paket slúžiaci na opätovné vyžiadanie dát
* **7** – paket na inicializáciu odosielania dát (obsahuje typ alebo názov s príponou súboru)
* **8 ­–** paket na odoslanie súboru alebo textovej správy
* **9** – paket na oznámenie zmeny rolí (z klienta sa stane server a naopak)

**2. Identifikácia prenosu** – každý process prenosu má vlastné identifikačné číslo

**3. Poradové číslo** – slúži na správne zoskupenie fragmentov na strane prijímateľa

**4. Dáta** – prenášané údaje

**5. CRC** – kontrolný súčet na detekciu chýb v prenášanom pakete

Shape, rectangle

Description automatically generated

*Obr. 2.1 Znázornenie hlavičky protokolu*

Dôležitou poznámkou je fakt, že okrem signálov typu 7 a 8 sa všetky druhy signálov prenášajú v pakete samostatne (paket pri ich odosielaní neobsahuje žiadny iný údaj, iba číslo signálu uvedené v prvej časti hlavičke protokolu).

### **2.3.2 Opis spôsobu inicializácie a ukončenia komunikácie**

Pred začiatkom každej komunikácie / prenosu je z pohľadu zadania dôležité nadviazať spoľahlivé spojenie s prijímacou stranou. Keďže protokol User Datagram Protocol (UDP) pracuje nespoľahlivo (bez spojenia a overovania), o nadviazanie a rušenie spojenia sa stará vlastný prokotol, navrhnutý v predchádzajúcej podkapitole, pomocou špeciálnych druhov paketov uvedených v časti “Typ paketu”.

O začiatok prvotnej inicializácie komunikácie sa stará odosielateľ (klient), ktorý sa snaží nadviazať spojenie so serverom formou *three-way handshake*. Úrobí tak odoslaním inicializačného paketu typu 0 – *SYN.* Akonáhle server identifikuje klientovu požiadavku, odošle mu späť paket typu 1 – *SYN+ACK* na potvrdenie akceptovania požiadavky spojenia (*ACK*) spolu s vlastnou inicializačnou požiadavkou (*SYN*). Po doručení uvedeného paketu sa komunikácia na strane klienta považuje už za nadviazanú, je však potrebné odoslať na stranu servera ešte jeden paket typu 2 – *ACK* na potvrdenia prijatia predchádzajúceho paketu. Komunikácia sa považuje za nadviazanú aj na strane servera až po prijatí tohto paketu. Po úspešnom uskutočnení *three-way handshake* je možné začať prenos dát odoslaním paketu 7.

Diagram

Description automatically generated

*Obr. 2.2 Sekvenčný diagram nadviazania spojenia*

Po úspešnom prenose dát sa klient môže rozhodnúť oficiálne prerušiť komunikáciu. Začiatok prerušenia komunikácie odštartuje odoslaním paketu typu 4 – *FIN*, z ktorého sa server dozvie o snahe ukončiť spojenie. Úlohou servera je odoslať odpoveď vo forme paketu typu 5 – *FIN+ACK*, v ktorom oboznámi klienta o prijatí úvodného paketu typu 4 – *FIN* (*ACK*) a zároveň doručí vlastnú žiadosť o prerušenie komunikácie (*FIN*). Akonáhle klient prevezme spomínaný paket, považuje komunikáciu za úspešne prerušenú a v poslednom kroku odošle na server paket typu 2 – *ACK* na potvrdenie prijatia predchádzajúceho paketu. Server považuje spojenie za úspešne ukončené až po prijatí tohto potvrdzovacieho paketu.

Diagram, schematic

Description automatically generated

*Obr. 2.3 Sekvenčný diagram prerušenia spojenia*

### **2.3.3 Opis metódy kontrolnej sumy (CRC)**

Výpočet aj evaluácia správnosti kontrolného súčtu pri každom odoslanom a prijatom pakete sa realizuje pomocou metódy *crc32()* pochádzajúcej z knižnice *zlib*. Uvedená metóda ako argument akceptuje dáta spolu s navrhnutou hlavičkou a vráti 32-bitovú celočíselnú hodnotu. Táto hodnota sa v uvedenom formáte vloží do hlavičky protokolu, paket dorazí na cieľovú adresu a tam sa opätovne uskutoční výpočet sumy s cieľom zistiť, či údaje dorazili bezchybne. Chybovosť sa overuje porovnaním nového výpočtu s hodnotou uloženou v hlavičke, pričom ak sa zhodujú, program pokračuje v ďalšej analýze paketu a odošle späť paket typu 2, ale ak nastane nezhoda indikujúca chybu, prijatý paket sa odstráni a odošle sa paket typu 5 na vyžiadanie opätovného odoslania údajov.

Uvedená metóda počas výpočtu kontrolného súčtu kombinuje dve operácie – binárny posun vľavo a následne operáciu XOR výsledku s binárnou hodnotou 1 0000 0100 1100 0001 0001 1101 1011 0111. Krátky opis samotného algoritmu by vyzeral nasledovne:

* *Obráť vstupnú hodnotu a priraď na jej koniec celkovo 32 núl*
* *Vykonaj operáciu XOR vstupu s hodnotou 0xFFFFFFFF*
* *Ak je prvý bit výsledku rovný 1, vykonaj XOR s vyššie písaným binárnym polynómom*
* *Vykonaj binárny posun celého výsledko vľavo a posuň číslo vpravo*
* *Opakuj kroky 3 a 4, dokým prvých 8 bitov výsledku nebude nulových*
* *Vykonaj operáciu XOR vstupu s hodnotou 0xFFFFFFFF a obráť výsledok*

### **2.3.4 Opis použitej ARQ metódy**

ARQ (Automatic Repeat Quest) je rozsiahla skupina metód detekcie a korekcie chýb využívaných pri prenose dát. Pre zaistenie spoľahlivého prenosu dát po nespoľahlivom kanáli používa signály na potvrdenie príjmu. Úlohou príjemcu je indikovať odosielateľovi úspešné a bezchybné prijatie paketu odoslaním kladného potvrdenia, pričom ak odosielateľ neobrdží uvedené potvrdenie do doby uplynutia zadefinovaného časového intervalu, odošle daný paket opäť. Počet znovuodoslaní paketov je obmedzený a ak sa prekročí maximálny počet pokusov, nastane chyba v komunikácii a spojenie sa považuje za prerušené.

Diagram

Description automatically generatedKomunikátor pre overovanie správnosti odosielania dát využíva ARQ metódu *Stop and Wait*, ktorej princíp je veľmi jednoduchý. Odosielateľ posiela pakety po jednom, pričom po každom odoslanom pakete očakáva signál kladného potvrdenia zo strany príjemcu. Nasledujúci paket je odoslaný až po prijatí spomenutého potvrdenia. Na druhej strane, od prijímateľa sa očakáva, že po prijatí paketu toto potvrdenie odošle. Samozrejme, ak prijatý paket obsahuje chybné údaje (zistené napr. tým, že vypočítaný kontrolný súčet sa nerovná súčtu uvedeného v hlavičke), namiesto jednoduchého potvrdenia prijatia sa odosielateľovi doručí paket typu 6, slúžiaci na znovuvyžiadanie predchádzajúceho paketu.

*Obr. 2.4 Sekvenčný diagram ARQ metódy Stop and Wait*

### **2.3.5 Opis metódy na udržiavanie spojenia**

Odosielanie paketov typu 3 (na overovanie konektivity druhého komunikujúceho zariadenia) sa uskutočňuje v odlišnom vlákne ako primárna komunikácia, s cieľom predchádzať kolíziám. Uvedené pakety odosiela vysielací účastník komunikácie prijímacúcemu v pravidelných intervaloch dĺžky 5 sekúnd, pričom po odoslaní sa očakáva aj odpoveď prichádzajúca z opačnej strany vo forme paketu typu 2 – *ACK*. Ak odpoveď nedorazí do piatich sekúnd, odošle sa ešte ďalšie dve pakety typu 3 (tiež v intervaloch 5 sekúnd) a ak ani na tieto žiadosti nepríde odpoveď, spojenie sa považuje za nefunkčné a automaticky sa preruší. Týmto by mali byť ošetrené situácie, kedy sa prvý paket typu 2 – *ACK* podarilo odoslať, ale nepodarilo sa ho doručiť (nedojde k automatickému ukončeniu spojenia).

Ak klient na odoslané pakety typu 3 nedostane odpoveď, považuje druhú stranu na nedosiahnuteľnú a okamžite preruší spojenie (nevyužije sa oficiálne prerušenie kombináciou paketov *FIN, FIN+ACK, ACK*, keďže server by na ne neodpovedal).

Rovnako, v prípade, ak server nedostane v očakávanej dobe tri po sebe nasledujúce pakety typu 3 zo strany klienta, považuje klienta za odpojeného a prestane počúvať, pričom dôležitý je fakt, že k prerušeniu počúvania dojde iba v prípadoch, keď bol klient reálne k serveru pripojený (ak server iba čaká na inicializáciu spojenia zo strany klienta, neprestáva počúvať).

### **2.3.6 Opis spôsobu zmeny úloh**

Súčasťou programu je aj možnosť automatickej výmeny úloh medzi klientom a serverom. Proces výmeny úloh musí byť inicializovaný na strane klienta, pričom po jeho skončení sa z klienta stane server a zo servera klient. Proces prebieha nasledovne:

* Používateľ na strane klienta v menu vyberie možnosť *Change mode.*
* Na server sa odošle paket typu 3 – *SWICH* a očakáva sa odpoveď.
* Zo strany servera dorazí paket typu 4 – *FIN* a troma krokmi sa oficiálne ukončí spojenie
* Následne sa vykonávanie programu na oboch stranách vráti do *while* cyklu v *main.py*, kde sa uloží nová hodnota do premennej *entity\_type*, podľa ktorej sa v novej iterácii cyklu zvolí správna možnosť (v tomto prípade zmena úloh).
* Dôležité údaje, ako IP adresa a port servera sú neustále uložené v globálnych premenných, kvôli čomu ich po výmene úloh používateľ nebude musieť manuálne zadávať.
* Po inicializácií dvoch nový entít (klienta a servera) dojde k ich automatickému spojeniu využitím *three-way handshake.*
* Účastníci komunikácie sú pripravení na prenos dát.

Základnou úlohou inicializačnej strany je doručiť druhému účastníkovi paket typu 9 – oznámenie o zmene rolí, pričom rozdiel oproti iným druhom komunikácie je, že sa tu neočakáva odpoveď druhej strany vo forme paketu typu 2 – *ACK*. Po odoslaní paketu si klient automaticky zmení svoju rolu, pričom akonáhle druhej strane dorazí paket typu 9, aj ona vykoná rovnaký krok.

Diagram

Description automatically generated

*Obr. 2.5 Sekvenčný diagram výmeny úloh*

## **2.4 Zmeny oproti návrhu**

V porovnaní s originálnym návrhom programu a jeho funkčnosti nastalo niekoľko zmien, žiadna z nich však nie je príliš rozsiahla. Zoznam zmien sa nachádza nižšie:

* Zmena veľkosti položky *Typ paketu* z 2 na 1 Byte v hlavičke
* Odstránenie položky *Veľkosť paketu* z hlavičky (dá sa vypočítať na strane servera)
* Pakety typu 3 – *keep-alive* sú odosielané len zo strany klienta na stranu servera

Pri prvej z uvedených možností bolo cieľom znížiť počet prenášaných dát, pretože údaje obsiahnuté v tejto položke hlavičky (číselné hodnoty od 0 po 9, predstavujúce typ signálu) je relálne možné prenášať (zakódovať) aj v 1 Byte.

Výsledkom druhej zmeny bolo taktiež zníženie veľkosti prenášaných dát, pretože samotnú veľkosť paketu nie je potrebné odosielať na server (tento údaj je možné jednoducho určiť aj na strane servera výpočtom na základe prenesených dát).

Posledná, tretia zmena, týkajúca sa *keep-alive* paketov bola dôležitá z logických dôvodov. Po uskutočnení spojenia stačí, ak bude klient odosielať *keep-alive* pakety na stranu servera a server na ich prijatie reagovať odoslaním paketu typu 2 – *ACK*, pretože už v tomto prípade dokážu obe strany reagovať na prerušenie komunikácie (server dokáže prerušiť spojenie, ak sa mu v očakávanom časovom intervale nedoručí paket typu 3, a klient dokáže taktiež predčasne ukončiť komunikáciu, ak mu za vopred prednastavený časový interval po odoslaní *keep-alive* paketu nedorazí zo strany servera paket typu 2 – *ACK*).

# **3 Opis dôležitých častí programu**

Nasledujúca kapitola sa venuje examinácii kľúčových súčastí programu, ako napríklad zabalenie a rozbalenie paketov, evaluácia signálov, doručovanie *keep-alive* signálov a odosielanie a prijímanie dátových paketov.

## **3.1 Diagram priebehu programu**

Diagram

Description automatically generated

*Obr. 3.1 Diagram inicializácie oboch entít v programe*

## **3.2 Zabalenie a rozbalenie paketov**

Pod pojmami zabalenie a rozbalenie sa rozumie prevod údajov obsiahnutych v pakete do takého tvaru, v akom sa dokážu efektívne preniesť cez sieť od klienta k serveru (v jazyku Python prevod bežných dátových typov na typ *bytes* alebo *bytearray*) a následne konvertovanie týchto údajov späť do ich pôvodného tvaru.

Konvertovanie paketu na byty a jeho následné odoslanie rieši metóda *send\_packet()* triedy General, ktorá na vstupe akceptuje typ odosielaného paketu, adresu prijímateľa, zoznam údajov odosielaných v pakete a taktiež informáciu, či sa v náhodných paketoch majú pred odoslaním vygenerovať chyby, alebo nie (posledné dve vymenované parametre sú dobrovoľné). Ak ide o jednoduchý signál (napr. *ACK, SYN, FIN, keep-alive...*), v pakete sa okrem zakódovanej hodnoty tohto signálu neprenášajú žiadne iné informácie. Ak sa ale jedná o pakety typu *DATA\_INIT* alebo *DATA*, paket obsahuje všetky údaje uvedené v hlavičke, pričom hodnota *crc* sa vypočíta priamo v tejto metóde.

Text

Description automatically generated

*Obr. 3.2 Metóda send\_packet() triedy General*

Na strane servera rieši prevod prijatých byteov späť do ich originálneho tvaru metóda *decode\_data\_packet()*. Ako to už z jej názvu vyplýva, používa sa len na prevod paketov typu *DATA* alebo *DATA\_INIT*, pretože, ako už bolo vyššie spomenuté, v prípade paketov iných typov sa prenáša iba typ signálu a jeho hodnota sa dekóduje priamo na mieste, kde sa využíva.

Text

Description automatically generated

*Obr. 3.3 Metóda decode\_data\_packet() triedy General*

## **3.3 Odosielanie a prijímanie dátových paketov**

Odosielanie dátových paketov sa z logických dôvodov uskutočňuje iba na strane klienta, preto sa skupina metód *create\_transfer\_header(), create\_fragments()* a *transfer\_data()* nachádza iba v triede Client.

Úlohou prvej z menovaných metód je vytvorenie paketu typu 7 – *DATA\_INIT* so všetkými potrebnými údajmi. Pri tomto pakete sa do políčka *poradové číslo* uloží počet fragmentov, ktoré má server očakávať, a do *dáta* sa vkladá v prípade textovej správy slovo „text“ a v prípade súboru názov aj s príponou.

Text

Description automatically generated

*Obr. 3.4 Metóda create\_transfer\_header() triedy Client*

Druhá metóda *create\_fragments()* rieši rozdelenie prenášaných dát v prípade, ak sú väčšie, ako používateľom nastavená maximálna veľkosť fragmentu. Vrámci nej sa vytvoria jednotlivé pakety, pričom ku každému fragmentu dát je už v tomto prípade priložené aj identifikačné číslo paketu. Po dokončení fragmentácie metóda vracia zoznam paketov pripravených na prenos po sieti.

Text

Description automatically generated

*Obr. 3.5 Metóda create\_fragments() triedy Client*

Posledná metóda *transfer\_data()* sa stará o samotný proces odosielania pripravených dát, pričom dokáže reagovať aj na problémy v prenose. Ak sa paket nesprávne doručí, zo servera sa vráti správa typu 6 – *ERROR* a klient došle identický paket znova. Rovnaký výsledok nastane, ak sa zo strany servera nevráti správa typu 2 – *ACK* potvrdzujúca správnosť prijatia paketu – po uplynutí časového intervalu sa paket automaticky odošle znova.

Text

Description automatically generated

*Obr. 3.6 Metóda transfer\_data() triedy Client*

Prijímanie dátových paketov sa uskutočňuje na strane servera vrámci metódy *receive\_data()*. Server z prvotného paketu *DATA\_INIT* pozná presný počet očakávaných dátových paketov a po prijatí každého z nich si vypočíta vlastné *crc*, ktoré porovnáva s *crc* uloženým v pakete. Ak sa nezhodujú, vyžiada si daný paket opäť, v opačnom prípade pripojí jeho obsah k premennej *received*.

Text

Description automatically generated

*Obr. 3.7 Metóda receive\_data() triedy Server*

## **3.4 Evaluácia prijatých signálov**

Na automatické určenie správnej odpovede pre každý prijatý paket bola zhotovená metóda *eval\_sig()* triedy General. Štruktúra metódy je veľmi jednoduchá. V prvom kroku sa za zistí hodnota signálu konvertovaním z byteov na typ integer. Následne sa už iba vyberie správny typ odpovede na daný signál (na každý signál sa odpovedá nejakým preddefinovaným spôsobom, napr. na signál typu 0 – SYN sa ako odpoveď očakáva signál typu 1 – *SYN+ACK*, na signál typu 6 – *ERROR* sa odpovie znovuodoslaním predchádzajúceho paketu a signálom typu 8 – *DATA* a podobne) a táto hodnota sa vráti späť.

Text

Description automatically generated

*Obr. 3.8 Metóda eval\_sig() triedy General*

## **3.5 Systém doručovania keep-alive signálov**

Akonáhle sa vytvorí spojenie medzi klientom a serverom, klientska strana inicializuje odosielanie paketov typu 3 – *keep-alive* a doručuje ich na server v pravidelnom intervale 5 sekúnd, pričom server na ich príchod reaguje spätným odoslaním paketu typu 2 – *ACK*. Dôležitou poznámkou je, že odosielanie *keep-alive* paketov prebieha iba v prípadoch, keď sa práve neuskutočňuje žiadny prenos dát medzi koncovými uzlami (počas prenosu dát je pozastavené a odosielanie sa obnoví v okamihu dokončenia prenosu dát). Systém *keep-alive* paketov využíva multithreading na klientskej strane z dôvodu, aby používateľ dokázal interagovať s programom a nebol obmedzovaný formou zablokovania programu.

Text

Description automatically generated

*Obr. 3.9 Metódy init\_keepalive() a keep\_connection() triedy Client*

# **4 Ukážka testovacieho scenáru**

Nasledujúca kapitola sa venuje examinácii