Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 812 19 Bratislava

Dokumentácia k projektu z PKS:

Komunikácia s využitím UDP protokolu

Meno: **Peter Markuš**
Študijný odbor: INFO-4
Akademický rok: 2016/2017

Obsah

Komunikácia s využitím UDP protokolu

1 Zadanie úlohy

Nad protokolom UDP (User Datagram Protocol) transportnej vrstvy sieťového modelu TCP/IP navrhnite a implementujte program, ktorý umožní komunikáciu dvoch účastníkov v sieti Ethernet, teda prenos správ ľubovoľnej dĺžky medzi počítačmi (uzlami).

Program bude pozostávať z dvoch častí – vysielacej a prijímacej. Vysielací uzol pošle správu inému uzlu v sieti. Predpokladá sa, že v sieti dochádza k stratám dát. Vysielajúca strana rozloží správu na menšie časti - fragmenty, ktoré samostatne pošle. Správa sa fragmentuje iba v prípade, ak je dlhšia ako max. veľkosť fragmentu. Veľkosť fragmentu musí mať používateľ možnosť nastaviť aj menší ako je max. prípustný pre trasportnú vrstvu.

Po prijatí správy na cieľovom uzle tento správu zobrazí. Ak je správa poslaná ako postupnosť fragmentov, najprv tieto fragmenty spojí a zobrazí pôvodnú správu.

Komunikátor musí vedieť usporiadať správy do správneho poradia, musí obsahovať kontrolu proti chybám pri komunikácii a znovuvyžiadanie rámca, vrátane pozitívneho/negatívneho potvrdenia. Pri nečinnosti komunikátor automaticky odošle paket pre udržanie spojenia. Odporúčame riešiť cez vlastne definované signalizačné správy.

Program musí mať nasledovné vlastnosti (minimálne):

1. Pri posielaní správy musí používateľovi umožniť určiť cieľovú stanicu.
2. Používateľ musí mať možnosť zvoliť si max. veľkosť fragmentu.
3. Obe komunikujúce strany musia byť schopné zobrazovať:
   - poslanú resp. prijatú správu,
   - veľkosť fragmentov správy.
4. Možnosť odoslať chybný rámec
5. Možnosť odoslať dáta zo súboru a v tom prípade ich uložiť na prijímacej strane do súboru

Zadanie sa odovzdáva:

1. Návrh riešenia
2. Predvedenie riešenia v súlade s prezentovaným návrhom

Program musí byť organizovaný tak, aby oba komunikujúce uzly mohli byť (nie súčasne) vysielačom a prijímačom správ. Pri predvedení riešenia je podmienkou hodnotenia schopnosť doimplementovať jednoduchú funkcionalitu na cvičení.

2 Analýza

2.1 Enkapsulácia a hlavičky

Počas procesu enkapsulácie si každá vrstva vytvára dátovú jednotku protokolu. Proces nastáva pri putujúcich dát sieťou od zdroja k cieľu, pričom menia svoju formu a naberajú hlavičky danej vrstvy. Každou vrstvou pridaním svojej hlavičky vznikne datagram danej vrstvy. Názov o spätnom procese s odstraňovaním hlavičiek sa hovorí o dekapsulácii.

2.2 Hlavné vlastnosti protokolu UDP z pohľadu zadania

Cez UDP protokol je možné posielať správy (datagramy) iným používateľom, ktorý sú napojený v sieti internetového protokolu (IP). Narozdiel od TCP protokolu nezaručuje, že sa príslušné pakety nestratia alebo sa nezmení poradie paketov. Práve aj kvôli tomu môže byť UDP časovo rýchlejší a efektívnejší.

2.3 Chyby v prenose a znovuvyžiadanie rámca

UDP má nezabezpečený prenos dát a môžu sa stratiť, poškodiť, duplikovať alebo predbiehať. Je preto potrebné zabezpečiť mechanizmus, ktorý ošetrí tento problém zo strany aplikácie za pomoci vlastného protokolu.

2.4 Veľkosť fragmentu

Pakety sú jednotky dát, ktoré putujú od zdroja k cieľu sieťovej komunikácie. Ak sa zo zdrojového zariadenia pošle súbor alebo veľkosť datagramu prekročí maximálnu veľkosť paketu, UDP protokol ho rozloží na viac fragmentov a pošle ich do cieľového zariadenia. Každý fragment maximálne obsahuje 1500 bajtov. Je však zmenšený kvôli dátam uložených na hlavičke.

3 Špecifikácia požiadaviek

Pre požiadavky programu je potrebné použiť programovací jazyk java, sieťové zariadenie a knižnice. Program bude využívať knižnice zamerané na štandardné grafické používateľské rozhranie a knižnicu na sieťové komunikácie:

```
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.net.SocketTimeoutException;
import java.util.ArrayList;
import java.util.Date;
```

4 Návrh riešenia

4.1 Návrh vlastného protokolu

4.1.1 Komunikácia

Na komunikáciu medzi klientom a serverom v programe je potrebné komunikovať prostredníctvom paketov odoslanými cez UDP protokol. Program bude mať možnosť dvoch volieb módu: server a klient. Budú teda pozostávať z dvoch častí: vysielacej a prijímacej. Klient vydá inicializáciu spojenia so serverom a počká na odpovede od servera. Server čaká na správy od klienta a pri doručení spätne odpovie na jeho správy na zabezpečenie úspešného poslania od klienta. V prípade že by server chcel poslať správu klientovi, v GUI si klient pre to musí nastaviť mód prijímania.

4.1.2 Rámec a pakety

Kvôli obmedzeniam protokolu nižších vrstiev je treba do zadania implementovať aj veľkosť paketu, ktorý bude zadaný používateľom. Z toho nám vyplynie, že v prípade presahu maximálnej veľkosti paketu je potrebné rámec rozložiť do fragmentov. Najmä kvôli fragmentácii sa implementuje hlavička ktorá bude zahrnutá ako súčasť dát v UDP protokole. Tá nám bude poskytovať informácie o príslušnom pakete či celom rámci. Zahrnutím tejto hlavičky bude štandardná veľkost 1500 bajtového paketu zmenšený prostredníctvom týchto zakomponovaných informácií:

ID správy – určuje typ dát správy
Poradové číslo fragmentu – pre zabezpečenie zachovaného poradia fragmentov od odosielateľa ku prijímateľovi
CRC – overuje správnosť odoslania správy (checksum)
Dĺžka správy – počet znakov správy
Dáta – prenášané dáta rámca alebo fragmentu rámca

4B	4B	4B	4B	1480B
ID správy	č. fragmentu	CRC	Dĺžka správy	Dáta

ID správy – číslo ID správy hovorí o type správy, ktorý bol odoslaný

- č. = 0 – správa je odoslaná po časovom úseku na udržanie spojenia (keepalive)
- č. = 1 – správa o znovuvyžiadanie rámca
- č. = 2 – správa o odoslaní správy od klienta
- č. = 3 – správa o odoslaní súboru

4.1.3 Fragmentácia paketov a ich poradové čísla

Pri fragmentácii sa dáta rozdelia sekvenčne do vytvorených fragmentov takým spôsobom, že časť počtu poslaných bajtov bude korešpondovať s kapacitou dát jedného fragmentu. Teda napr. pri veľkosti fragmentu 40 bajtov a dĺžke 30 bajtov zadaného reťazca v správe sa z reťazca uloži do prvého fragmetu 20 bajtov, čo je teda (veľkosť fragmentu) mínus (hlavička), a zvyšných 10 znakov sa uloží do nasledujúceho paketu. Túto informáciu o veľkosti fragmentov potrebujeme zobraziť na oboch komunikujúcich stranách spolu aj s textom správy. Prípadne na to, aby sa zabezpečilo korektné poslanie, server môže vydetekovať predbiehanie poslaných paketov. Ak sa tak stane, server pošle cez ID správu znovuvyžiadanie rámca kvôli chybne usporiadanému textu.

4.1.4 CRC

Okrem problému o neusporiadaných paketov je potrebné skontrolovať na základe vyhodnotenia CRC (checksumu), či prijaté správy boli nepoškodené, alebo sa nestratili. Ak sa tak stane, server pošle klientovi cez ID správu znovuvyžiadanie rámca číslom 1. V opačných prípadoch vyšle ID správu č. 2 o správnom odoslaní správy.
Na možnosť odoslania chybného rámca prepíšem hodnoty v CRC tak, aby nesedeli s klientovou druhou stranou.
Taktiež sa môže stať, že server v danom momente bude vypnutý alebo nebude prijímať žiadne správy. V takom prípade bude pomocou TimeoutExceptionu v jave na strane odosielateľa zabezpečená informovanosť o nedetekovanom príjme.

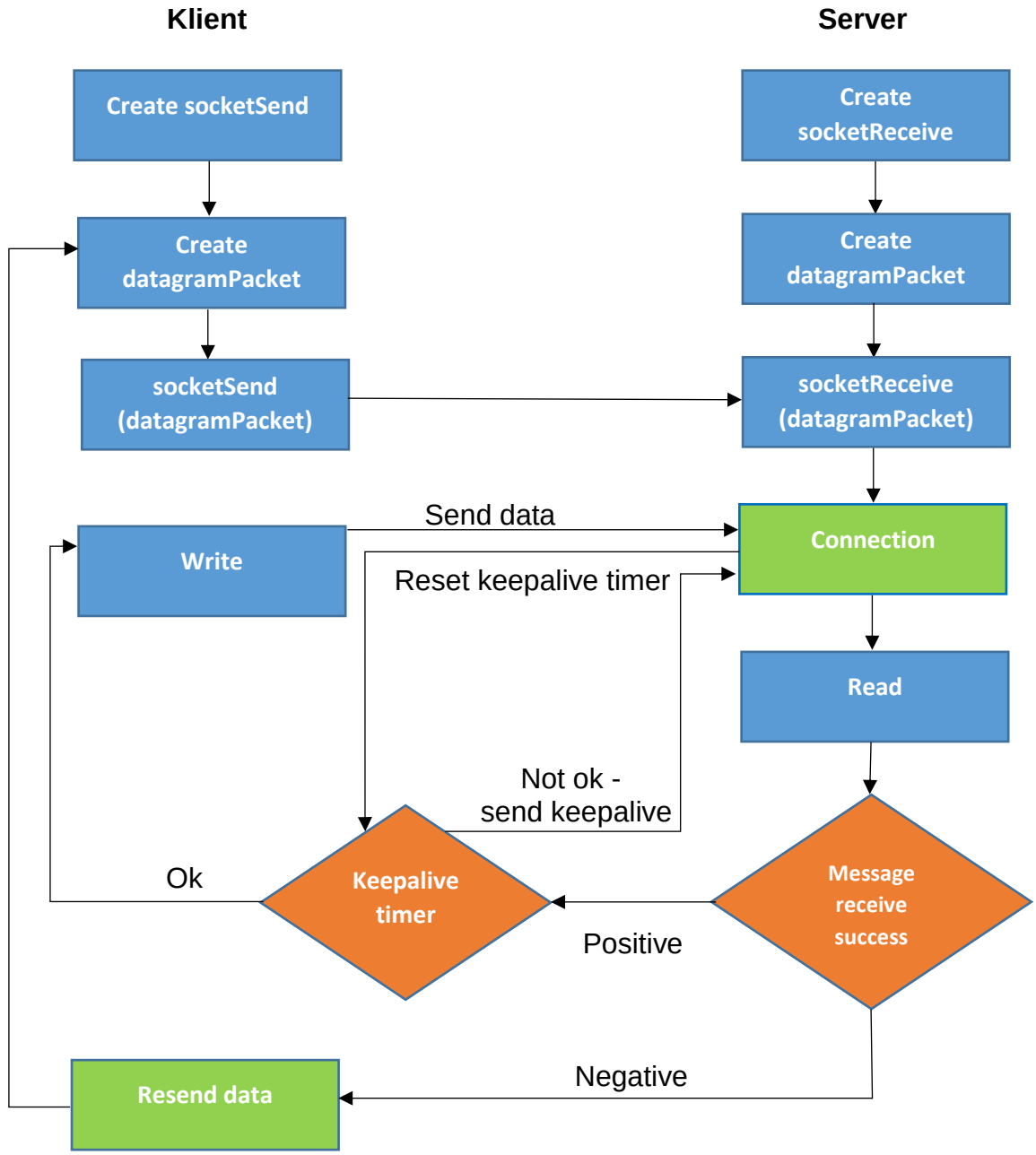
4.1.5 Keepalive

Kontrola spojenia je dôležitá na udržanie spojenia medzi klientom a serverom. Fungovať bude na princípe keepalive signálu, pri ktorom sa bude spúšťať hneď, ako server potvrdí spojenie. Za každým odoslaním sa timer vynuluje. Keď timer dosiahne určitý čas na udržanie kontaktu, potom klient odošle keepalive signál serveru. Môže to byť aj dátovo prázdny paket, kedže nám ide len o ID správu č. 0 v tomto prípade. Pri tomto odoslaní sa bude timer resetovať.

4.1.6 Odoslanie správy a súboru

Identifikácia odosielania obyčajnej správy bude cez číslo ID správy 2. Dáta zo súboru budú posielané tak, ako pri klasickej správe, no po prijatí na základe čísla ID správy 3 server dostane pokyn na uloženie dát do súboru, ktorého názov bude taký ako aj zo strany klienta. Názov súboru sa bude prenášať na prvom fragmente dát a dáta budú začínať až na druhom fragmente rámca. Funkcionalita týchto typov správ bude implementovaná cez tlačidlá v GUI.

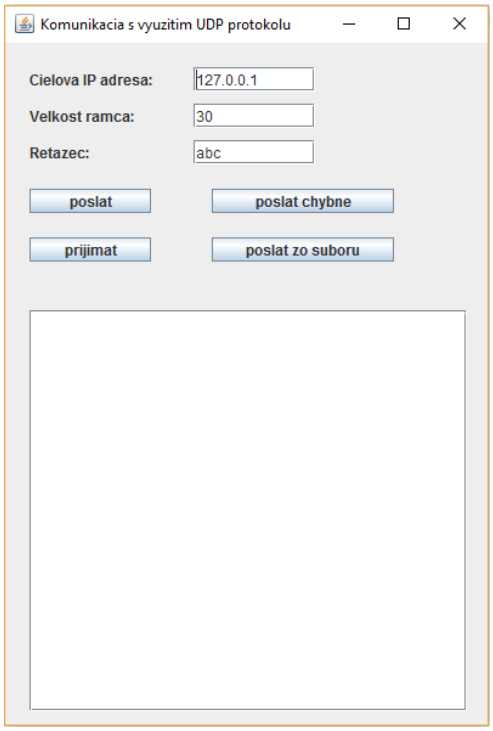
4.2 Vývojový diagram



4.3 Používateľské rozhranie

Implementácia bude realizovaná v jazyku Java.

Používateľ si v prípade posielania zvolí cieľovú IP adresu s ktorou bude komunikovať, veľkosť rámca a reťazec ktorý predstavuje obsah správy. V prípade stavu prijímania sa klikne na tlačidlo prijimat. Keď používateľ prijíma, nebude mu umožnené posielanie. Preto sa vytvorí ďalšie tlačidlo stopPrijmu, pričom všetky ostatné už nebudú k dispozícii. Okrem poslaných správ sa budú zobrazovať aj veľkosti paketov v textovom okienku.



5 Implementácia

5.1 Zmeny oproti návrhu

~~V prípade že by server chcel poslať správu klientovi, v GUI si klient pre to musí nastaviť mód prijímania.~~ Žiadny mód nebude potrebný pretože pre každú stranu stačí zapnúť prijímanie správ. Tak bude každá strana odosielateľom a zároveň aj prijímateľom.

~~4B~~	~~4B~~	~~4B~~	~~4B~~	~~1480B~~
~~ID správy~~	~~č. fragmentu~~	~~CRC~~	~~Dĺžka správy~~	~~Dáta~~

8B	1B	4B	4B	4B	1479B
CRC	ID správy	poradové č. fragmentu	veľkosť paketu	počet paketov	Dáta

ID správy – číslo ID správy hovorí o type správy, ktorý bol odoslaný

- ~~č. = 0 – správa je odoslaná po časovom úseku na udržanie spojenia (keepalive)~~
- ~~č. = 1 – správa o znovuvyžiadanie rámca~~
- č. = 2 – správa o odoslaní správy od klienta
- č. = 3 – správa o odoslaní súboru

~~Aby sa zabezpečilo korektné poslanie, server môže vydetekovať predbiehanie poslaných paketov. Ak sa tak stane, server pošle cez ID správu znovuvyžiadanie rámca kvôli chybne usporiadanému textu.~~ Znovuvyžiadanie rámca kvôli chybne usporiadanému textu sa nestane, pretože server si usporadúva pakety sám vždy bezchybne.

~~Dáta zo súboru budú posielané tak, ako pri klasickej správe, no po prijatí na základe čísla ID správy 3 server dostane pokyn na uloženie dát do súboru, ktorého názov bude taký ako aj zo strany klienta. Názov súboru sa bude prenášať na prvom fragmente dát a dáta budú začínať až na druhom fragmente rámca.~~ (najprv som myslel, že bude stačiť posielanie textových súborov tak, že prečítam všetky znaky, a vložím ich do reťazca (teda správy). To zrejme stačiť nebude, preto som umožnil posielať súbory cez bajty pre možnosť posielania rôznych druhov formátov. Názov súboru od odosielateľa sa však prenášať nebude na druhú stranu.

5.2 Použité knižnice a funkcie

GUI

```
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
```

Threads

```
import java.io.IOException;
import java.net.InetAddress;
```

Address

```
import java.net.InetAddress;
import java.net.UnknownHostException;
```

Convertor

```
import java.nio.ByteBuffer;
```

Packet

```
import java.util.Random;
import java.util.zip.CRC32;
import java.util.zip.Checksum;
```

Receiver

```
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.SocketTimeoutException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Arrays;
import java.util.zip.CRC32;
import java.util.zip.Checksum;
```

Sender

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.SocketTimeoutException;
import java.nio.file.Files;
import java.nio.file.Paths;
```

5.3 Organizácia programu

Aplikácia sa zapne triedou **Start**, ktorá vytvorí grafické používateľské rozhranie. Každé tlačidlo v ňom vytvorí thread s argumentami IP adresy, veľkosť rámca, reťazec a číslo ktoré identifikuje druh správy. Argumenty tak putujú do triedy **Threads**, kde pomocou switchu rozčleníme všetky druhy správ do jednotlivých case-ov. Case 0 až 1 vykonajú poslanie správy vytvorením triedy Sender. Číslo 1 sa bude líšiť tým, že dá vedieť triede Sender aby sa jednalo o chybné poslanie správy. Pri každom poslaní sa spustí/vynuluje timer signálu **keepalive**, ktorý sa posiela serveru na zistenie udržaného spojenia. Vytvorí sa taktiež cez triedu Sender stým rozdielom, že nebude obsahovať žiadne dáta. Paket teda bude prázdny a cez while cyklus ho bude pravidelne každých 30 sekúnd posielať, aby na neho server “nezabudol”.

Klient:

Pri posielaní správy cez **Sender** sa najprv vytvorí ackSocket, ktorý bude čakať maximálne 2 sekundy na server pre potvrdenie príjmu jednotlivých paketov. Následne sa vytvorí počet potrebných paketov do pola, pričom každý bude obsahovať svoju veľkosť dáť. To bude nápomocné v nasledujúcom for cykle, v ktorom sa jednotlivé pakety budú sekvenčne posielať.

```
82
83          for(j = 0; j < pocetPacketov; j++) {
84              do {
85                  data = Packet.vytvorPacket(poradoveCislo, packety[j], retazec, idMsg, offset, pocetPacketov);
86                  GUI.textFinal.append("velkost " + (j+1) + ". packetu: " + packety[j] + "\n");
87                  datagramPacket = new DatagramPacket(data, data.length, adresa, 9002);
88                  socketSend.send(datagramPacket);
89
90                  ackSocket.receive(ackPacket); // wait for acknowledgment
91                  respCode = ackPacket.getData()[0];
92
93                  if (idMsg == 1) // prvykrat som poslal chybne
94                      idMsg = 0;  // od teraz uz posielam spravne
95
96              } while (respCode == 0);    // ked bol NACK, tak cyklus pokracuje, lebo chceme poslat spravny ramec
97
98              poradoveCislo++;
99              offset += packety[j];
100         }
```

Premenná offset slúži na správne rozdelenie celej správy do paketov. idMsg ak je 1, tak v triede **Packet** sa zaručí nesprávne vypočítaný checksum.

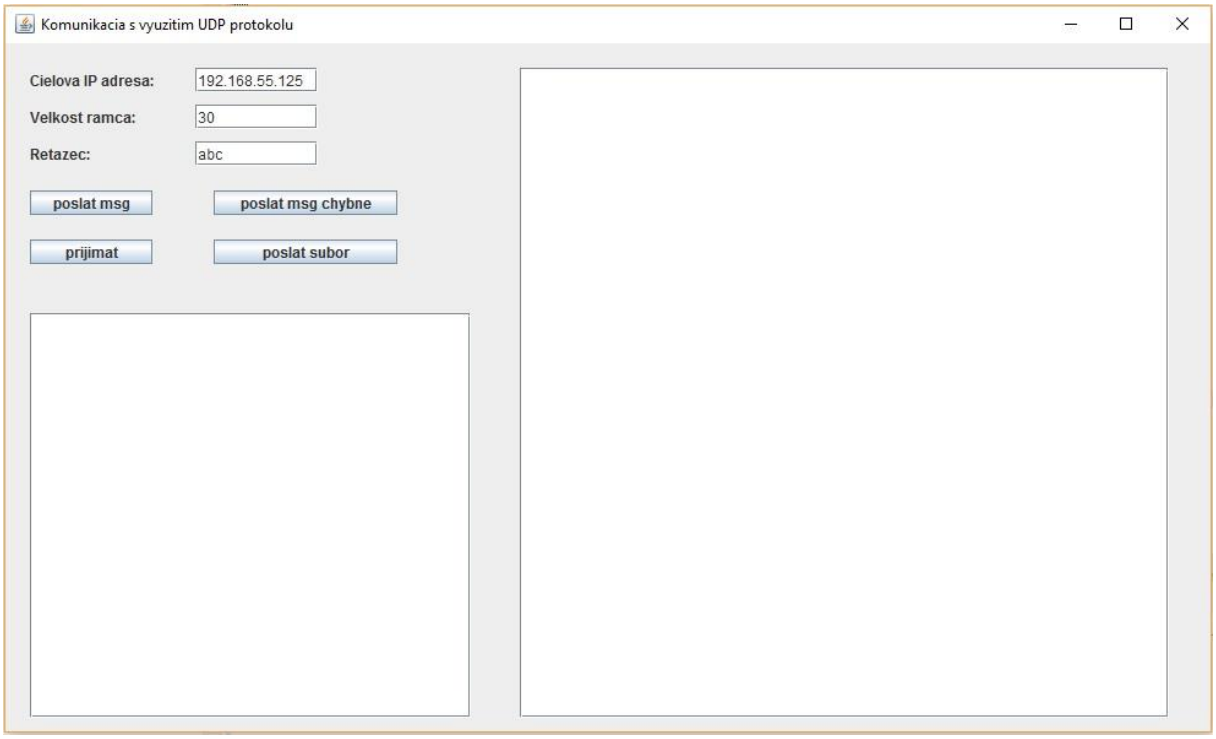
Server:
Na prijímanie správ sa v Threads vytvorí trieda Receiver pomocou case-u s číslom 3. V ňom sa vytvorí socketReceive, ktorý má maximálny timeout 60 sekúnd. Následne v cykle while bude dekódovať bajty do pomocných premenných pomocou triedy **Convertor**. Ak prišiel keepalive signál v 60 sekundovom intervale, tak preskočí všetky operácie s dátami

```
else if (dataLength >= 0 && dataLength < 65507) {
    //String value = new String(datagramPacket.getData(), Packet.HEADER_SIZE, dataLength);
    GUI.text.append("Velkost " + packetNumber + ". paketu: " + datagramPacket.getLength() + ", dlzka dat: " + dataLength + "\n");

    sentChecksum = Convertor.BytesToLong(datagramPacket.getData(), 0, 8);
    checksum = new CRC32();
    checksum.update(buffer, 8, 13 + dataLength);        // 1 + 4 + 4 + 4 + dlzka
    reciChecksum = checksum.getValue();
    if (reciChecksum == sentChecksum)
        responseBuffer[0] = 1;      // 1 = ACK
    else {
        GUI.text.append("Bolo to chybne poslane, posielam znovuvyziadanie ramca\n");
        responseBuffer[0] = 0;      // 0 = NACK
    }

    if (msg == null && idMsg < 2)       // spajam spravu dokopy
        msg = new FullMessage(amount);
    if(idMsg < 2) {
        String value = new String(datagramPacket.getData(), Packet.HEADER_SIZE, dataLength);
        msg.insertNewFragment(packetNumber, value);
    }

    else if(mss == null)                // spajam bajty pre subor
        mss = new ByteArrayOutputStream();
    if(idMsg == 2) {
        byte[] value = Arrays.copyOfRange(datagramPacket.getData(), Packet.HEADER_SIZE, dataLength+Packet.HEADER_SIZE);
        mss.write(value);
    }
```

Najprv sa zobrazí veľkosť daného paketu ktorý sa práve poslal, porovná sa checksum na overenie správneho poslania. Ak sa práve vypočítaný checksum nerovná s checksumom uloženým v hlavičke príslušného paketu, pošle sa NACK pre znovuvyžiadanie rámca. Ak sa jedná o poslanie správy, tak pomocou triedy **FullMessage** sa budú spájať všetky stringy paketov do jedného stringu, ktorý sa na konci vypíše. Ak sa jedná o poslanie súboru, spájajú sa bajty pomocou funkcie ByteArrayOutputStream a na konci cyklu sa pomocou získaných bajtov zapíše súbor do miesta C:\poslane\posted.
Predtým sa však ešte posiela paket ACK/NACK správy, ktorá povie klientovi, či teda má poslať rámec ešte raz.

5.4 Používateľská príručka – GUI

Ľavá strana predstavuje text pre odosielateľa, pravá strana text pre prijímateľa. Pre posielanie súboru je potrebné zadať do položky reťazec cestu s menom súboru, ktorý sa bude prenášať.



6 Zhodnotenie

Implementáciu som zrealizoval v programovacom jazyku java. Bol vybraný práve za veľké množstvo funkčných mechanizmov. Umožňoval procesy spracovávania paketov jednoduchým spôsobom a množstvo funkcií nebolo treba implementovať ručne. Nevýhodou javy však je, že niektoré operácie sa vykonávaju pomalšie ako v jazyku C, C++ či C#.

Posielanie súboru spolu s jeho názvom nebolo neimplementované z časových dôvodov**.**