SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

Počítačové a komunikačné siete

Zadanie 2 - Komunikátor

Ctibor Kovalčík
AIS ID: 110829
FIIT STU
Cvičenie: Štvrtok 16:00
22.10.2021

1 Zadanie úlohy

1.1 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.

1.2 Vlastnosti

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.

2 Návrh riešenia

2.1 Programovací jazyk

Komunikátor som sa rozhodol implementovať v programovacom jazyku Python v prostredí PyCharm. Dôvodom je pre mňa príjemnejšie programovanie v tomto jazyku, ale takisto python ponúka viacero funkcií ktoré mi uľahčia prácu.

2.2 Fungovanie programu

Diagram 3

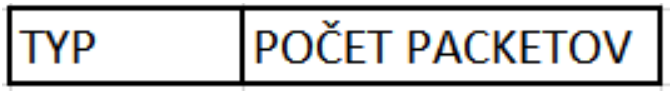
Program bude fungovať na princípe klienta a serveru. Po spustení bude na výber Server a Klient. Server si nastaví port na ktorom bude počúvať, a IP ma nastavenú na localhost (127.0.0.1). Teraz bude Server čakať na nejaké nadviazanie spojenia s klientom.

Klient po spustení programu a vybratí možnosti Klient si nastaví Ip adresu servera a port na ktorý sa chce pripojiť. Následne sa môže začať prenos dát. Klient bude mať na výber aký typ dát chce odoslať na server.

- 0 (Poslať textovú správu) -> Klient chce poslať textovú správu
- 1 (Poslať súbor) -> Klient chce poslať súbor (fotka,..) na server

2.3 Hlavička vlastného protokolu

Inicializačný packet, ktorý sa pošle vždy ešte pred samotným odosielaním dát. Obsahuje Typ správy a Počet packetov ktoré budú odoslané.



Obr. 1: Inicializačný packet

- TYP - Typ správy :
  - 0 -> textová správa (char)
  - 1 -> súbor (char)
- Počet packetov - počet packetov ktoré budú odoslané

Pred samotnými dátami ktoré chcem poslať bude táto hlavička :

1B	2B	2B	2B	
Typ	Veľkosť	Poradie	CRC	Data

Obr. 2

Hlavička obashuje 4 údaje.

- Typ -> typu char.
  - 1 -> posielanie dát (posiela Klient -> Server)
  - 2 -> Doručené dáta sú poškodené (posiela Server -> Klient)
  - 3 -> Doručené dáta sú v poriadku (posiela Server -> Klient)
  - 4 -> Udržiavanie spojenia (posiela Klient -> server)
- Veĺkosť -> Veľkosť dát ktoré sú posielané
- Poradie/Počet -> Pri inicializačnej správe -> Počet , pri posielaní packetov od Klienta->Server -> Poradie packetu.
- CRC -> na kontrolu či boli obdržane dáta na strane serveru nepoškodené

Data -> samotné dáta ktoré server/klient posiela

2.4 Veľkosť dát

Komunikátor by podľa zadania mal byť riešený tak aby nenastala fragmentácia na linkovej vrstve. Čiže maximálna veľkosť dát ktoré je možné odoslať v 1 fragmente(packete) je :
1500 B - IP_header - UDP_header - Moj_header
1500 B - 20 B - 8 B - 7 B = 1465 B

2.5 ARQ

Diagram 6
Komunikácia bude prebiehať systémom Stop-and-wait ARQ metódou, čiže klient odošle packet serveru a čaká na prijatie packetu od serveru o úspešnom/neúspešnom prenose packetu. Ak príde správa o neúspešnom prenose packetu tak Klient odošle daný packet ešte raz. Ak príde správa o úpsešnom packete, tak Klient posiela ďalší packet. Obrázok 3

2.6 Kontrola poškodených packetov

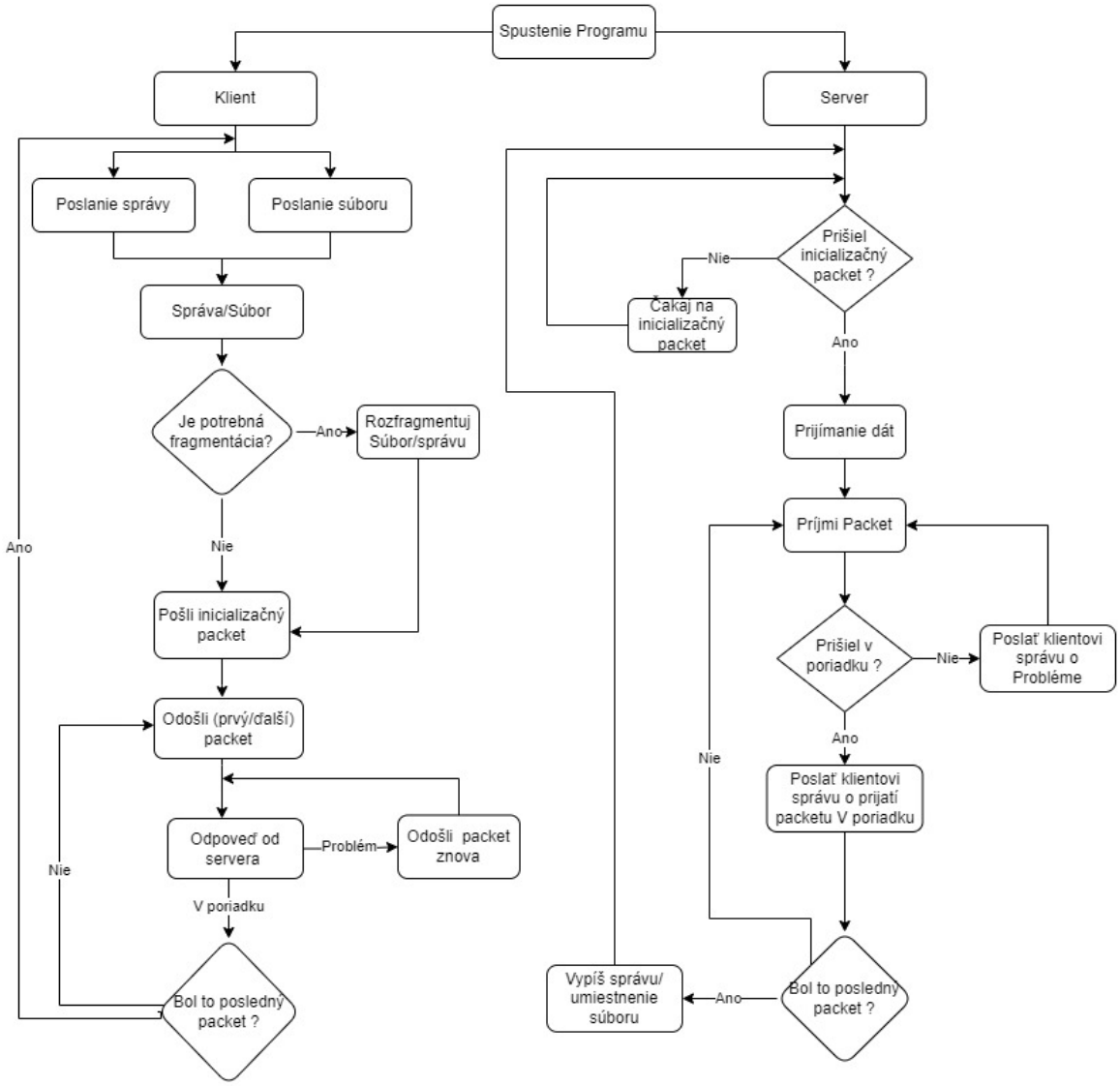
Diagram 4
Server po prijatí packetu od Klienta musí skontrolovať či sú dáta v poriadku a niesu poškodené. To zabezpečím v programe tým, že Klient pošle CRC v hlavičke a Server vďaka tomu CRC dokáže zistiť, či śu dáta poškodené alebo nie. Ak sú tak pošle správu Klientovi o neúspešnom prenose dát a ten mu ich odošle znova. Ak dáta niesu poškodené tak odošle Klientovi správu o úspešnom prenose dát.

Na výpočet crc použijem funckiu pythonu z knižnice binsacii : binascii.crc_hqx("Dáta",0). Táto funkcia vytvorí checksum o veľkosti 2B

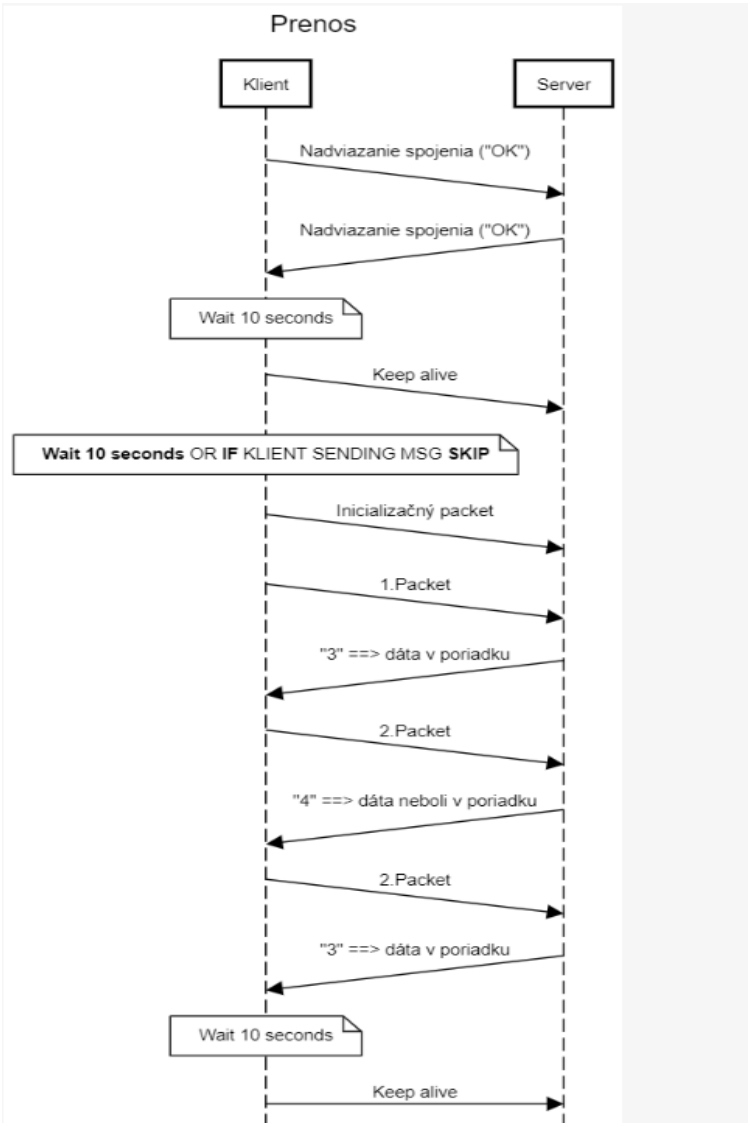
2.7 Udržiavanie Spojenia

Diagram 5
Udržiavanie spojenia medzi Klientom a Serverom budem riešiť cez thread a funkciu **keep_alive (client_socket,server_address)** ktorej parametre budú client_socket a adresa kde má posielať packety čiže adresa servera (IP,port). Tento thread sa spustí hneď potom ako sa nadviaže spojenie medzi Serverom a Klientom. Klient bude posielať každých 10 sekúnd keep alive packet a server po jeho obržďaní si zresetuje svoju životnosť vždy na 60s. V momente ak používateľ bude chcieť odoslať správu tak sa tento thread vypne a používateľ môže odosielať dáta. Thread posiela typ správy s obsahom "4" čo pre server znamená, že ak príde packet s takýmto obsahom tak resetne svoj timer (timeout) na 60 sekúnd. Ak by server nedostal žiaden packet (či už keep alive alebo dátový) po dobe dlhšiu ako 60s ,vypne sa. Hneď po odoslaní posledného packetu od Klienta->Server tak sa opäť spustí thread s keep alive funkciou.

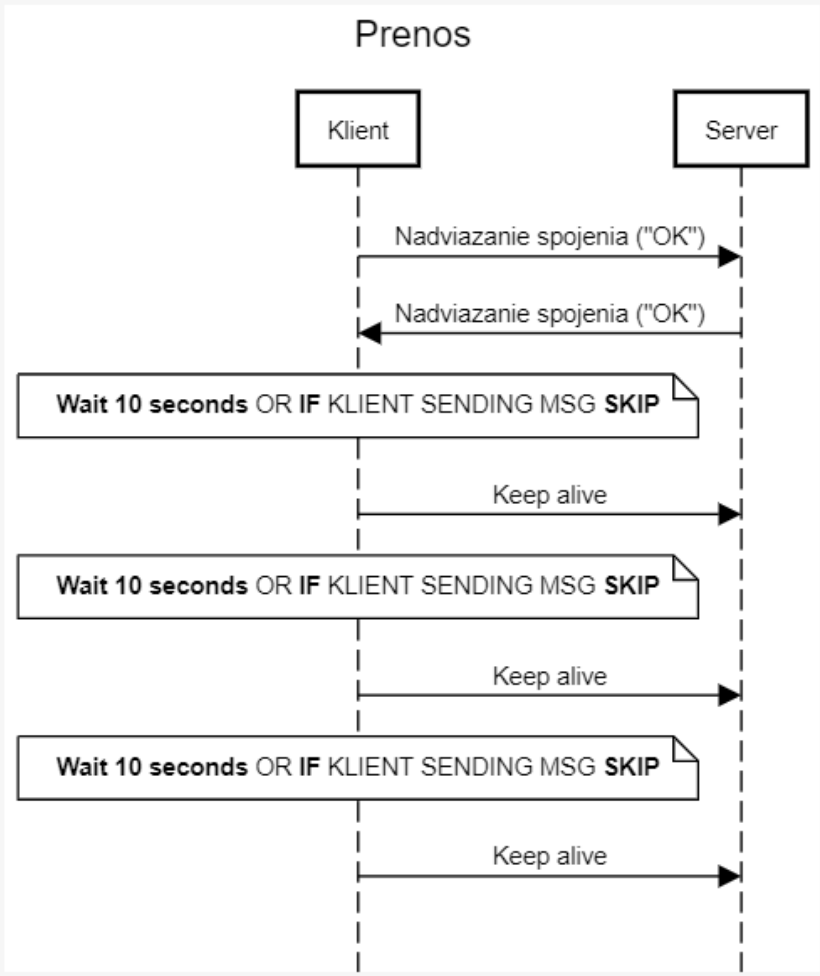
2.8 Diagram



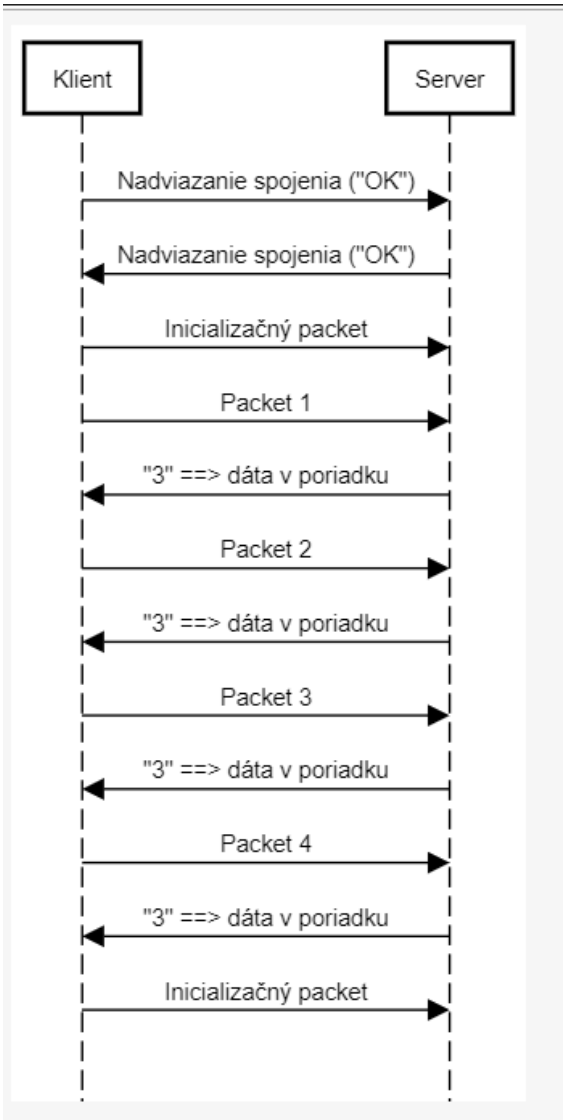
Obr. 3: Diagram programu



Obr. 4: Prenos 2 packetov + keep alive



Obr. 5: Keep Alive



Obr. 6: Prenos 4 packetov + začiatok novej správy (inicializačný packet)

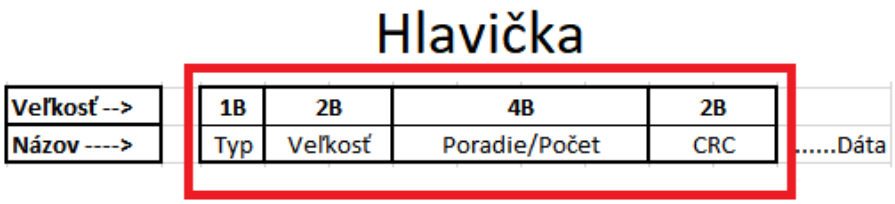
3 Riešenie

3.1 Zmeny oproti Návrhu

Pri samotnej implementácie komunikátora prebehli zmeny oproti Návrhu. Tieto zmeny boli podmienené konzultáciami k samotnému Návrhu ale aj samotným programovaním a zistením nejakých chýb v Návrhu.

Zmena v hlavičke

Hlavička po novom zmenila veľkosť zo 7B na 9B kvôli tomu aby sa rozmedzie hodnôt v poli Poradie/Počet zvýšil a to zabezpečí aby bolo možné odoslať aj viac packetov naraz (pod 1 inicializačným packetom) Ale Pole **Typ** môže mať iné hodnoty oproti tomu, čo



Obr. 7

bolo v Návrhu.

- Typ -> veľkosť 1B
  - 0 -> Doručené dáta sú poškodené (posiela Server -> Klient)
  - 1 -> Doručené dáta sú v poriadku (posiela Server -> Klient
  - 2 -> Udržiavanie spojenia (posiela Klient -> server)
  - 3 -> Posielanie textovej správy
  - 4 -> Posielanie súboru
  - 5 -> Inicializačný packet
  - 6 -> Hello Server / Hello Client -> nadviazanie spojenia
- Veĺkosť -> Veľkosť dát ktoré sú posielané
- Poradie/Počet ->
  Pri inicializačnej správe = Počet
  Pri posielaní packetov od Klienta->Server a naopak = Poradie packetu.
- CRC -> na kontrolu či boli obdržane dáta na strane serveru nepoškodené

Data -> samotné dáta ktoré server/klient posiela

Zmena v inicializačnom packete

Inicializačný packet ako bol predstavený v Návrhu som odstránil ale pred samotným odosielaním dát sa vždy musí poslať nejaké info o packetoch ktoré sa budú odosielať a to prebehne tým že sa pošle moja hlavička a Typ sa bude rovnať **5**. Tým sa zabezpečí aj kontrola prostredníctvom checksumu (CRC). Do Pola Poradie/Počet sa zapíše počet packetov ktoré je potrebné odoslať.

3.2 Vlastná Hlavička

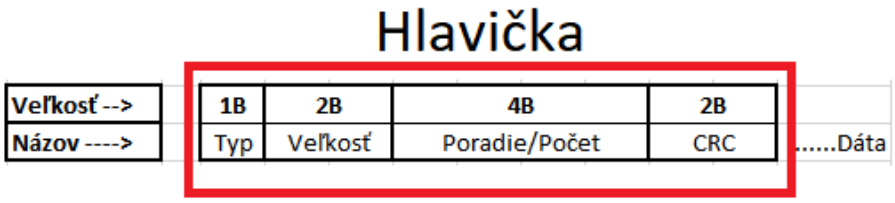
Vlastná hlavička ktorú som implementoval na UDP komunikáciu má veľkosť 9 Bytov. Skladá sa z 4 polí.
Typ, reprezentuje o aký typ packetu ide. V implementácií využívam viacero typov packetov ktoré posielam :

- 0 -> Doručené dáta sú poškodené (posiela Server -> Klient)
- 1 -> Doručené dáta sú v poriadku (posiela Server -> Klient
- 2 -> Udržiavanie spojenia (posiela Klient -> server)
- 3 -> Posielanie textovej správy
- 4 -> Posielanie súboru
- 5 -> Inicializačný packet
- 6 -> Hello Server / Hello Client -> nadviazanie spojenia

Veľkosť, reprezentuje veľkosť dát ktoré tento packet obsahuje

Poradie/Počet, reprezentuje poradie packetu alebo počet v závislosti od typu packetu. Ak sa jedná o inicializačný packet, tak toto pole predstavuje Počet packetov ktoré bude klient serveru odosielať ak je to packet s fragmentom správy/súboru tak toto pole predstabuje poradie packetu

CRC, reprezentuje Checksum o veľkosti 16bitov (2B).
používaná funkcia : binascii.crc_hqx(data , 0). Do tejto funkcie posielam hlavicku bez Checksumu + samotné dáta ktoré posielam. Týmto vytvorím checksum ktorý vložím na posledné miesto v hlavičke a za týmto už pôjdu dáta.



Obr. 8

3.3 Diagram fungovania programu

Na ďalšej strane je znázornený diagram fungovania programu 9.

- **Spustenie programu**
  Po spustení programu je možnosť si vybrať čo má reprezentovať spustený program. Na výber sú 2 možnosti. Server a Klient

- **Server**
  Po vybratí možnosti **Server** nasleduje konfigurácia servera.

  Používateľ musí nastaviť port na ktorom má server počúvať.

  Po nastavení portu Server čaká na správu Hello Server od Klienta. Ak táto správa nepríde do 60s alebo príde poškodená správa tak spojenie nebolo nadviazané a Program ponúkne používateľovi opäť výber možnosti ako pri spustení programu.

  Ak však správa od klienta prišla do 60s tak spojenie bolo nadviazané a Server bude čakať na inicializačný packet.

  Po prijatí inciliazičného packetu sa skontroluje checksum ak všetko sedí tak Server odošle klientovi správu o úspešnom prijatí packetu a čaká na prijatie prvého packetu s dátami.

  Po prijatí packetu skontroluje checksum a ak by to nesedelo odošle Klientovi packet s typom "0> čiže doručené dáta neboli v poriadku. Avšak ak je kontrola chýb (Checksum) v poriadku, tak odošle Klientovi packet s typom "1", čiže doručené dáta boli v poriadku, a ak ešte neboli prijaté všetky packety (v incializačnom packete sa posiela možstvo packetov ktoré budú poslané) tak čaká na ďalší, ale ak to už bol posledný packet tak vypíše správu ktorú obdržal. Následne ponúkne uživateľovi menu v ktorom môže server vypnúť, zmeniť rolu alebo pokračovať (Server bude pripravený príjmať dáta)

- **Klient**
  Po vybratí možnosti **Klient** nasleduje konfigurácia klienta.

  Používateľ musí nastaviť IP adresu servera a port na ktorý sa budú dáta(packety) posileať.

  Po nastavení sa pošle packet "Hello Server"Serveru a čaká sa na odpoveď ak príde odpoveď od servera "Hello Client", tak spojenie bolo úspešné nadviazané .

  Ak by správa neprišla do 60s alebo ak by prišla poškodená správa tak Klientovi sa nepodarilo úspešne nenadviazať spojenie so serverom a používateľovi sa poskytne

menu výberu ako na začiatku. Ak správa prišla do 60s tak spojenie bolo nadviazané a Klient pošle Serveru inicializačný packet.

Inicializačný packet obsahuje hlavičku + data. V hlavičke je pole typ = "5ä poradie neznačí v tomto prípade poradie packetu ale počet packetov ktoré klient odošle na server. Po poslaní inicializačného packetu klient čaká na odpoveď od servera. Server v prípade ak inicliazačný packet prišiel v proeiadku odošle packet s typom = "1"čiže packet bol úspešne prijatý.

Následne klient začne po 1 posielať packety s dátami. Za každým odoslaným packetom čaká na odpoveď od servera. V prípade ak odpoveď od servera pride s typom v hlavičke = "0"tak ten istý packet s dátami odošle na server ešte raz. Ak príde správa s typom v hlavičke packetu = "1", tak dáta boli úspešne poslané a klient odošle v poradí ďalší packet.

3.4 ARQ

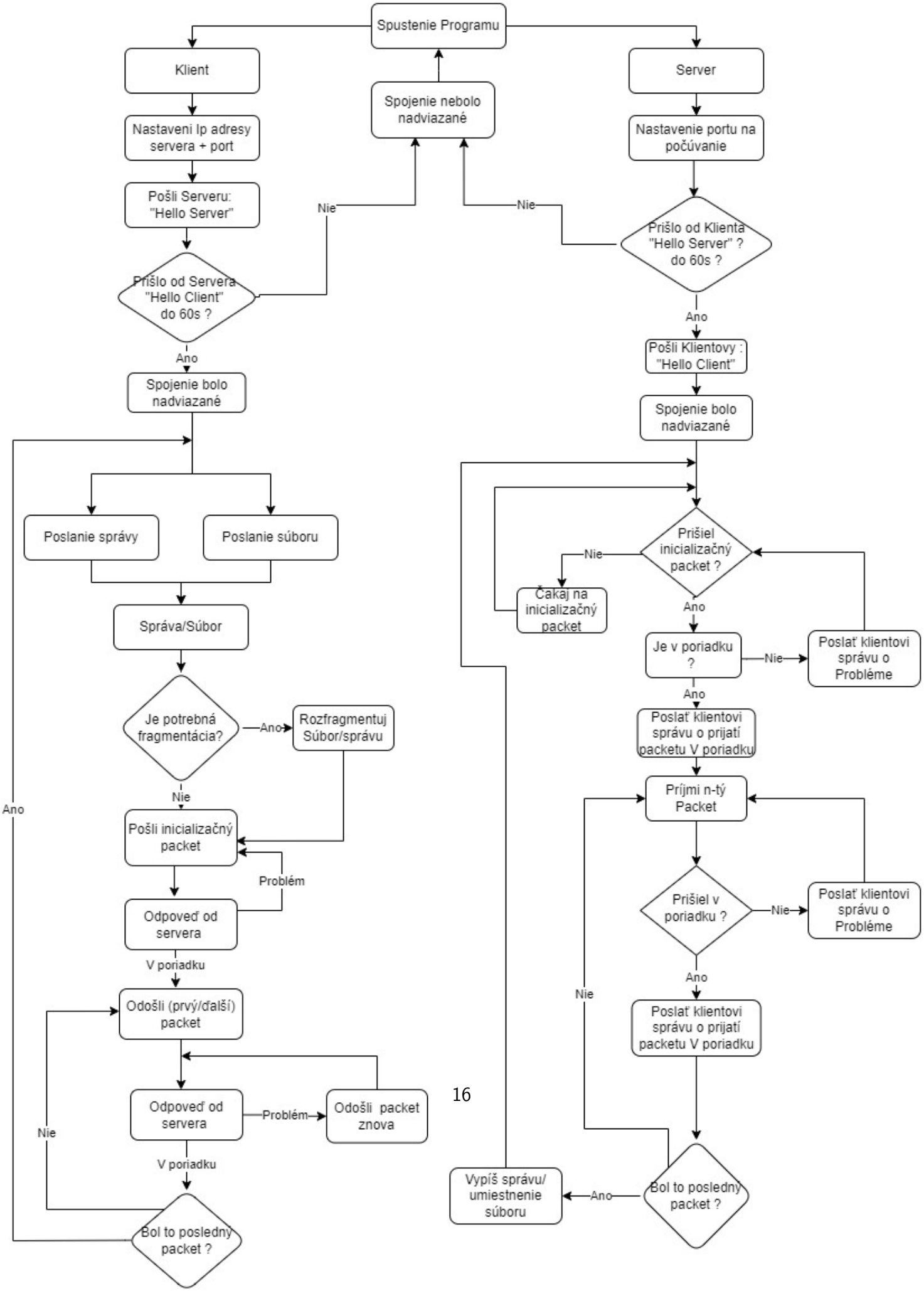
Program som realizoval Stop-and-Wait ARQ metódou. Za každým packetom ktorý pošle klient na server je očakávaný packet od servera s správou o prijatí packetu. Program funguje podľa tohto diagramu 9 a bližšie vizualizované posielanie jednotlivých packetoch je zobrazené na diagramoch 10 11. Vizualizáciu v prostredí Wireshark je možné vidieť na obrázkoch 12 a 13

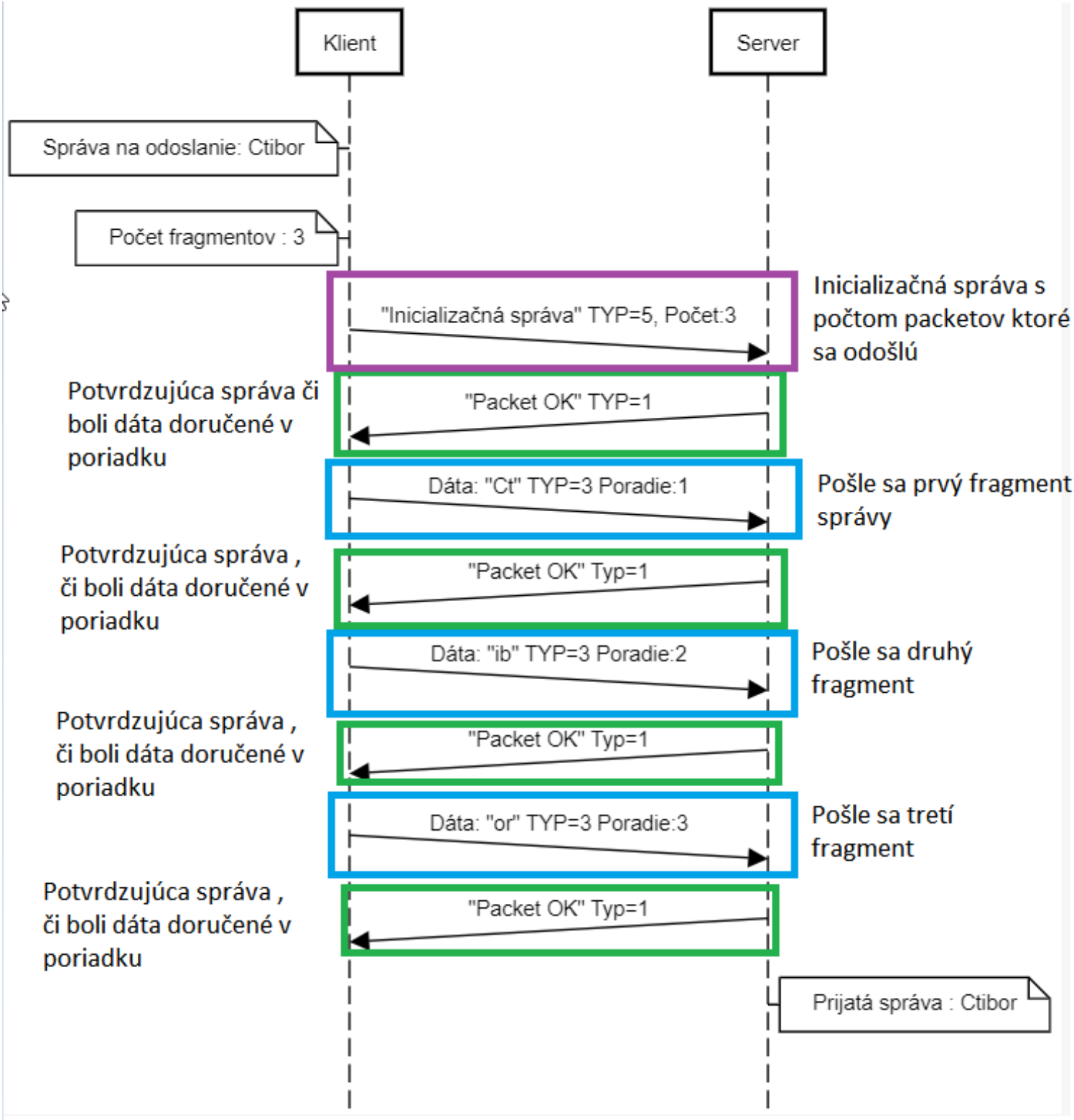
3.5 Simulácia chýb

Simuláciu chýb som v program riešil spôsobom , že používateľ si zadá percentuálnu pravdepodobnosť chyby a ak tá pravdepodobnosť výjde tak sa CRC vypočítané s hlavičky + dát pred odoslaním na server zmení.

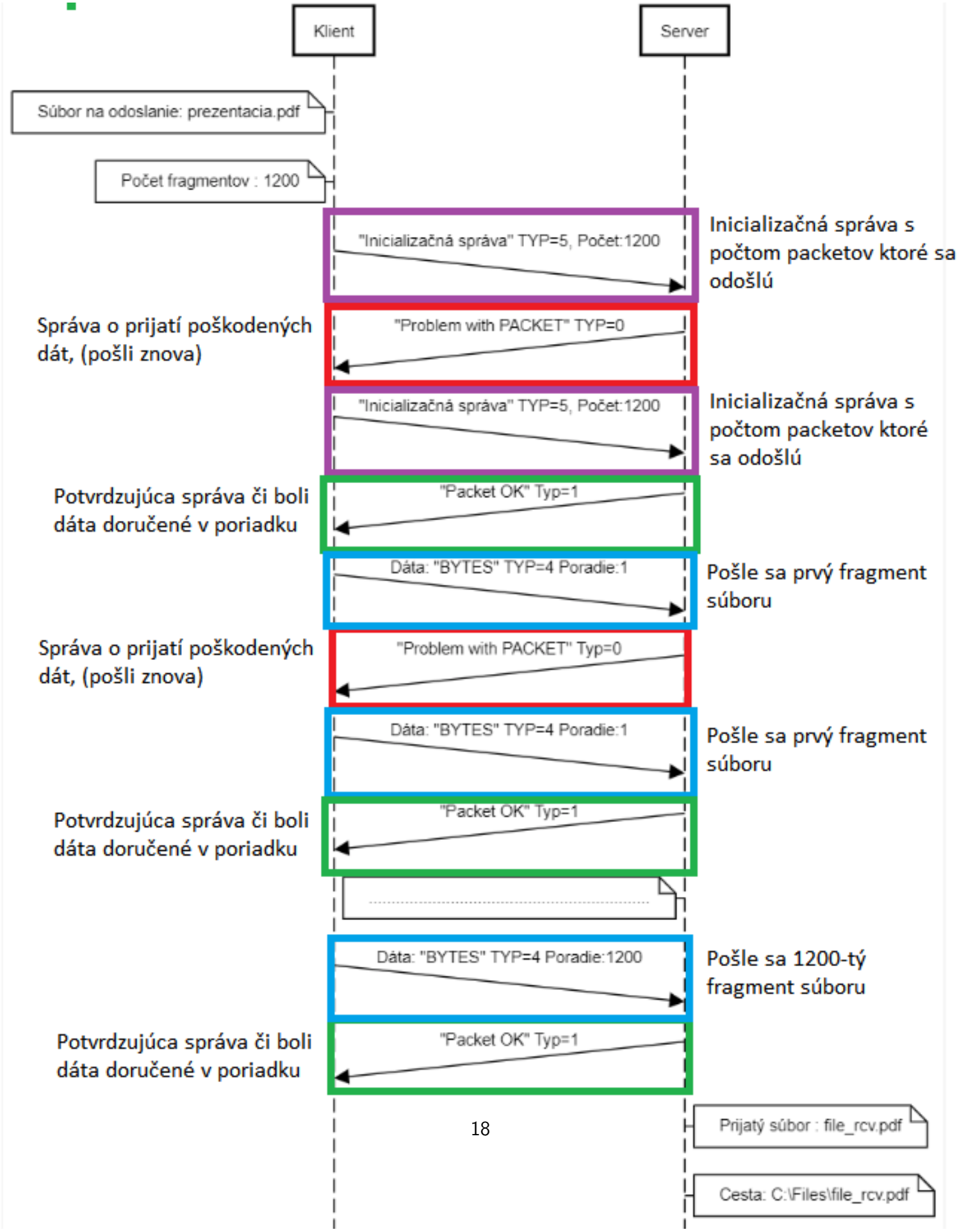
3.6 Keep Alive

Funkcia Keep alive zabezpečí aby sa spojenie nevyplo/nezrušilo po dlhšej dobe. Keep alive funguje v programe tým že používateľ po odoslaní správy môže zapnúť túto funckiu, ktorá kaŹdých 10s pošle na server udržiavaci packet aby server nezrušil spojenie. Ak používateľ bude chcieť poslať nejakú správu/súbor na server tak táto funkcia ktorá sa vykonáva v threade zanikne.

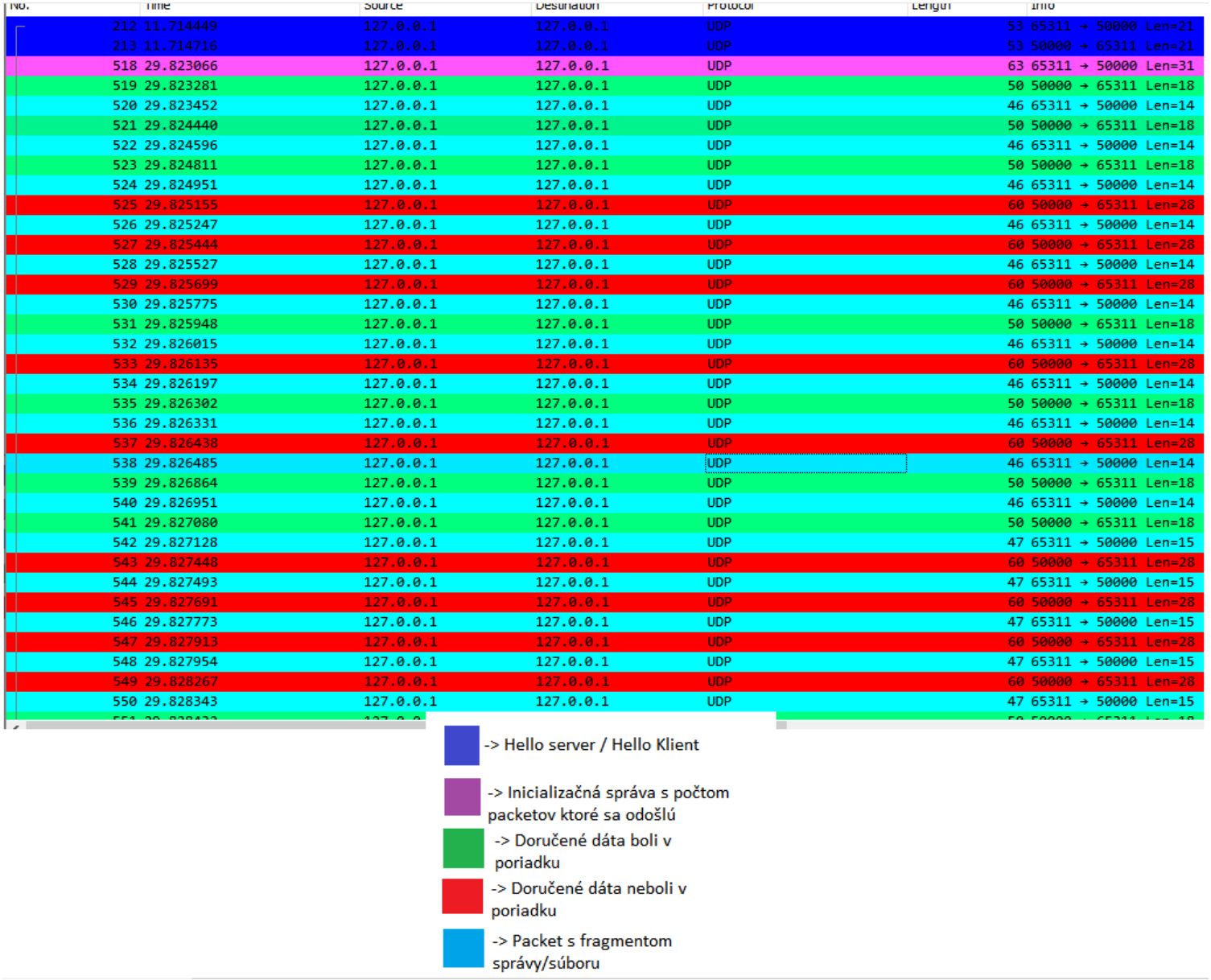

Spustenie Programu
Klient
Server
Nastaveni Ip adresy servera + port
Pošli Serveru: "Hello Server"
Prišlo od Servera "Hello Client" do 60s ?
Nie
Spojenie nebolo nadviazané
Ano
Spojenie bolo nadviazané
Nastavenie portu na počúvanie
Prišlo od Klienta "Hello Server" ? do 60s ?
Nie
Ano
Pošli Klientovy : "Hello Client"
Spojenie bolo nadviazané
Poslanie správy
Poslanie súboru
Správa/Súbor
Je potrebná fragmentácia?
Ano
Rozfragmentuj Súbor/správu
Nie
Pošli inicializačný packet
Problém
Odpoveď od servera
V poriadku
Odošli (prvý/ďalší) packet
Odpoveď od servera
Problém
Odošli packet znova
V poriadku
Bol to posledný packet ?
Nie
Ano
Prišiel inicializačný packet ?
Nie
Čakaj na inicializačný packet
Ano
Je v poriadku ?
Nie
Poslať klientovi správu o Probléme
Ano
Poslať klientovi správu o prijatí packetu V poriadku
Príjmi n-tý Packet
Prišiel v poriadku ?
Nie
Poslať klientovi správu o Probléme
Ano
Poslať klientovi správu o prijatí packetu V poriadku
Bol to posledný packet ?
Ano
Vypíš správu/ umiestnenie súboru
Nie



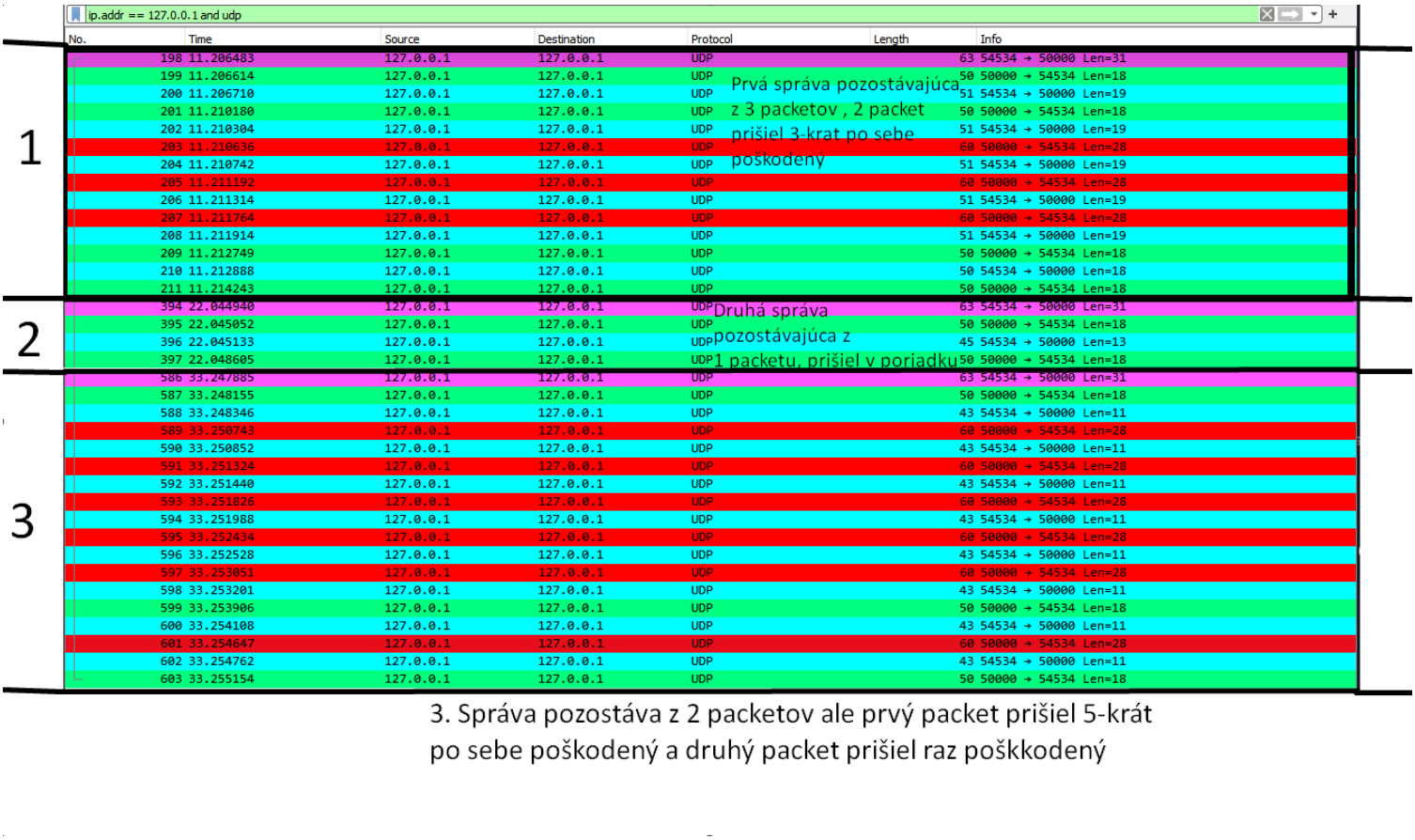
Obr. 10: Znázornenie prenosu dát



Obr. 11: Znázornenie prenosu dát



Obr. 12: Znázornenie prenosu vo Wireshark



Obr. 13: Znázornenie prenosu vo Wireshark