Dokumentácia vlastného protokolu nad UDP

Vladimír Jančár

25. novembra 2024

Abstrakt

Táto dokumentácia má za úlohu podrobne opísať návrh a vysvetliť implementáciu komunikačnej aplikácie s využitím vlastného UDP protokolu. Najprv opisuje konkrétne časti hlavičky protokolu a ich úlohy. Ďalej dokument približuje konkrétne metódy, využívané na spoľahlivý prenos dát pomocou jednotlivých častí navrhovanej hlavičky. Na záver rozpráva o implementácii tohto protokolu pomocou vlastnej komunikačnej aplikácie.

Obsah

1 Úvod

Tento protokol je určený na peer-to-peer komunikáciu v lokálnej Ethernet sieti a umožňuje prenos textu alebo súborov systémom Stop-and-Wait Automatic Repeat Request. Na nadviazanie spojenia využíva 3-Way Handshake, pričom stabilitu spojenia zabezbečuje keep-alive systém. Vďaka kontrolnému súčtu CRC16 a poradovým číslam packetov protokol chráni integritu prenášaných údajov a s fragmentáciou umožňuje aj posielanie väčších súborov. Hlavnou výhodou tohto protokolu je jednoduchosť, pričom nezanedbáva kľúčové vlastnosti ako detekciu chýb, spoľahlivé doručovanie správ alebo udržiavanie spojenia.

2 Návrh protokolu

Protokol je navrhutý nad UDP, no jeho cieľom je zaručovať spoľahlivý prenos dát, a preto napodobňuje funkčnosť TCP. Zabezpečuje nadviazanie spojenia medzi dvoma stranami podobne ako TCP 3-way handshake. Zvláda prenášať väšie súbory vďaka fragmentácii dát. V prípade, že overenie integrity zlyhá, poškodené dáta sa pošlú znova. Protokol umožňuje aj udržanie aktívneho spojenia medzi uzlami vďaka pravidelnej kontroly druhej strany spojenia.

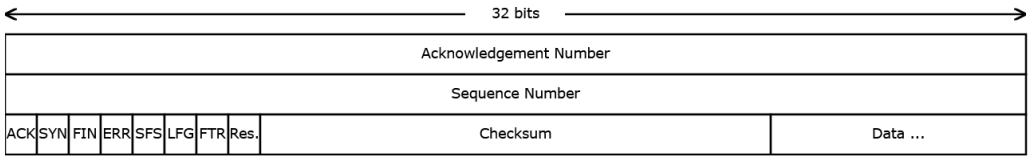
2.1 Hlavička protokolu

Hlavička vlastného protokolu je veľkosti 88 bitov a je štruktúrovaná nasledovne:

Pole	**Veľkosť**	**Popis**
Acknowledgement number	32b	Poradové číslo prijatého paketu
Sequence number	32b	Poradie paketu
ACK	1b	Acknowledgement flag
SYN	1b	Synchronization flag
FIN	1b	Finish flag
ERR	1b	Error flag
SFS	1b	Set Fragment Size flag
LFG	1b	Last Fragment flag
FTR	1b	File Transfer flag

Reserved	1b	Nevyužitý bit
Checksum	16b	Kontrolná hodnota CRC16

Diagramová reprezentácia štruktúry hlavičny je vyjadrená v obrázku 1.



Obr. 1: Štruktúra hlavičky protokolu

2.2 Popis polí

- **Acknowledgement number**: Pri potvrdení správnosti packetu toto pole obsahuje poradové číslo prijatého packetu.
- **Číslo sekvencie**: Pomáha kontrolovať správne poradie poslaných správ alebo súborov, čo je užitočné najmä pri fragmentácii. zistenie, či niektoré fragmenty nechýbajú.
- **ACK, SYN, FIN, ERR, SFS, LFG, FTR**: Riadiace flagy slúžiace na nadviazanie spojenia pomocou 3-way handshake, ukončenie spojenia a indikáciu chybných packetov, režijné správy pri prenášaní súborov.
- **Checksum**: Kontrolná hodnota pre zistenie poškodenia prenášaných údajov.

2.3 Kontroly integrity

K zabezpečeniu integrity prenášaných údajov využíva protokol 16-bitovú cyklickú kontrolu (CRC16) na výpočet kontrolného súčtu - **checksum**. Odosielateľ pred poslaním správy vypočíta jej checksum a vloží ho do hlavičky protokolu. Prijímateľ po prijatí správy vypočíta jej checksum znova a porovná ho s číslom v hlavičke protokolu. Ak sa zhodujú, správa nebola počas prenosu poškodená a prenos dát pokračuje. V prípade nezhody je odosielateľovi poslaná odpoveď s chybovým flagom ERR a poškodené dáta sa pošlú znova. Postup pri zaznamenaní poškodenia je podrobnejšie opísaný v časti 2.4.

Kontrolný súčet je 16-bitová hodnota vypočítaná nasledovne:

1. Najprv si vyberieme mnohočlen, ktorý budeme používať pri výpočte hodnoty CRC. Pre CRC16 sa bežne používa:

$$P(x) = x^{16} + x^{15} + x^2 + 1$$

   (V hexadecimálnom tvare **0x8005**)

2. Nastavíme počiatočnú CRC hodnotu. Zvyčajne sa používa **0xFFFF**.

3. Postupne prechádzame vstupnými dátami. Každý byte údajov ktoré kódujeme, prejde nasledujúcimi krokmi:

   (a) XOR-ujeme byte s najvššími ôsmimi bitmi hodnoty CRC.

   (b) Pozrieme sa, či má najvyšší bit hodnotu 1. Ak áno, spravíme bitový posun hodnoty CRC doľava a XOR-ujeme ju s vybraným mnohočlenom. Ak nie, spravíme iba bitový posun CRC doľava.

   (c) Ak sme ešte neprešli všetky bity v byte, vrátime sa do kroku (a).

4. Nazáver XOR-ujeme CRC hodnotu s 0xFFFF, ako je štandardom pri CRC-16-IBM, ktorý sme sa snažili napodobniť.

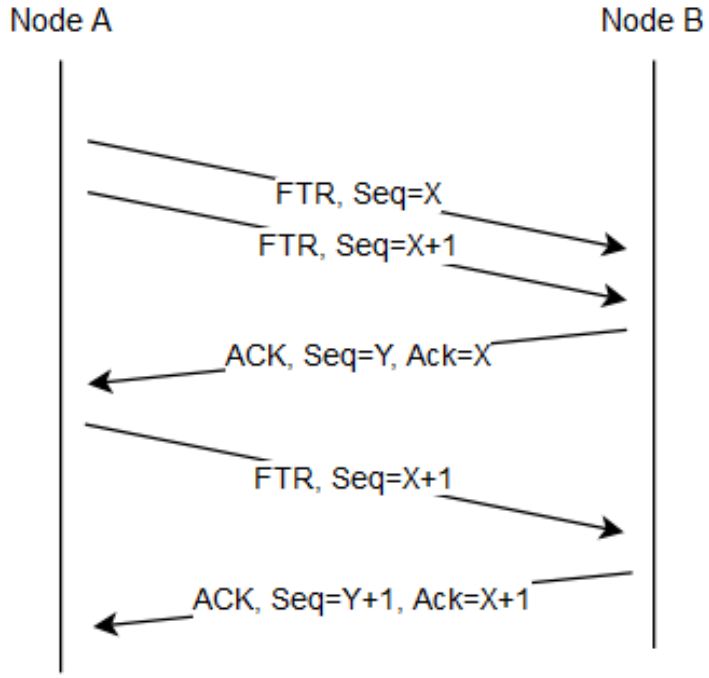
2.4 Spoľahlivý prenos údajov

Protokol zabezpečuje spoľahlivý prenos údajov pomocou vlastnej variácie ARQ (Obr. 2). Jeden thread odosiela fragmenty a druhý čaká na ACK packety. Vždy keď sa odošle fragment, pridá sa jeho sekvenčné číslo do zoznamu **unacknowledged packets**. Keď príde potvrdenie o úspešnom prijatí fragmentu (ACK packet s poradovým číslom prijatého fragmentu), sekvenčné číslo sa zo zoznamu nepotvrdených packetov odstráni. Ak ostali nejaké nepotvrdené packety v zozname, odošlú sa znova. Ak ktorákoľvek strana nedostane odpoveď na 3 po sebe idúce heartbeat packety, spojenie sa ukončí a zobrazí sa správa o neúspešnom prenose súboru. V prípade dočasného prerušenia sa prenost údajov obnoví.

2.5 Nadviazanie spojenia

Predtým, než bude možné si vymienať údaje, musia sa oba uzly presvedčiť, že druhá strana je pripravená komunikovať. K tomuto účelu slúži tzv. **3-Way Handshake** (Obr. 3), ktorý pozostáva z nasledujúcich krokov:

1. Odosielateľ žiadosti o nadviazanie spojenia pošle packet s vlajkou **SYN=1** a poradovým číslom X **(Seq=X)**.



Obr. 2: Diagram ARQ

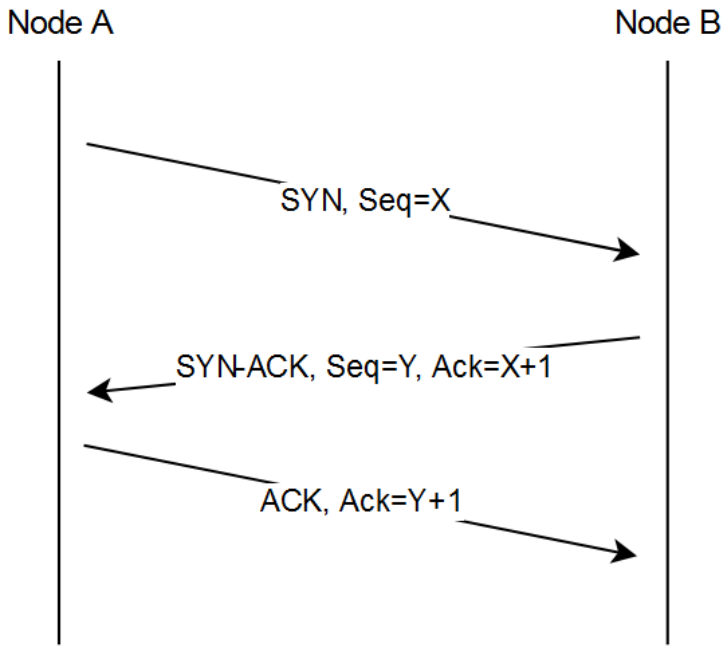
2. Keď prijímateľ dostane SYN packet, odošle odpoveď: tzv. SYN-ACK packet s vlajkami **SYN=1** a **ACK=1**. Packet bude obsahovať polia **(Seq=Y)** a **(Ack=X+1)**.

3. Nakonie iniciátor spojenia na SYN-ACK packet odpovie odoslaním ACK packetu, kde **(Ack=Y+1)**. Potom je spojenie úspešne naviazané.

Na začiatku sa oba uzly snažia byť iniciátorom, teda posielajú SYN packety. Ten, kto zaznamená SYN packet ako prvý, na neho automaticky odpovedá.

2.6 Udržanie spojenia

Pre udržanie spojenia medzi dvoma uzlami využíva protokol metódu **Keep-Alive**. Aby sa predišlo strate spojenia v prípade, že ani jedna strana neposiela údaje, vymieňajú si uzly tzv. **heartbeat packety** každých 5 sekúnd nečinnosti. Takýmto spôsobom protokol zisťuje, či sú obe strany stále pripojené. Heartbeat packety vyzerajú tak, že ich ACK flag má hodnotu 1 a ich poradové číslo je 0. Ak jedna zo strán nedostane odpoveď na 3 po sebe



Obr. 3: Diagram procesu nadviazania spojenia

odoslané heartbeat packety, spojenie sa považuje za ukončené. Vtedy nastane jedna z nasledujúcich situácií:

- Ak nedostane komunikátor odpoveď na žiaden z heartbeat packetov, vypíše informačnú správu pre používateľa o ukončení spojenia. V prípade, že sa spojenie prerušilo počas prenosu textu alebo súboru, vypíše aj to, že prenos zlyhal.
- V prípade, že spojenie nebolo prerušené manuálne jedným z používateľov, komunik8tor sa snaží opäť nadviazať kontakt pomocou 3-way handshake.
- Ak sa podarí obnoviť spojenie (komunikátor dostane odpoveď aspoň na posledný odoslaný heartbeat packet) a prerušenie nastalo počas prenosu textu alebo súboru, pokúsi sa v pokračovať v prenose tam, kde sa spojenie prerušilo.

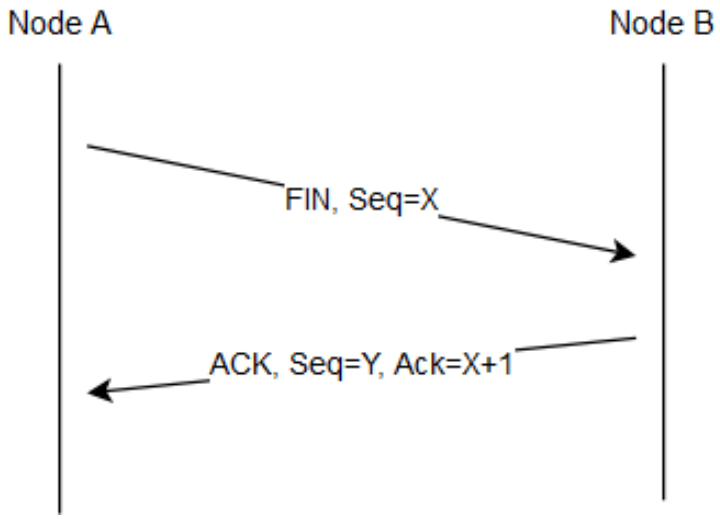
2.7 Ukončenie spojenia

V protokole prebieha ukončenie spojenia pomocou 2-way termination) (Obr. 4). Proces vyzerá nasledovne:

1. Odosielateľ pošle packet s flagom **FIN=1** a poradovým číslom (Seq=X).

2. Po prijatí FIN packetu druhý uzol odpovie odoslaním packetu s flagom **ACK=1** a rovnakým poradovým číslom. Následne odošle aj FIN packet s novým poardovým číslom (Seq=X+1).

3. Keď prvý odosielateľ prijme FIN packet, definitívne sa ukončuje spojenie oboch uzlov.



Obr. 4: Diagram procesu ukončenia spojenia

3 Implementácia

Na implementáciu protokolu bola v jayzku Python[1] vytvorená komunikačná aplikácia. Tá umožnuje nadviazanie spojenia dvoch uzlov pomocou nášho vlastného protokolu. Pri spustení používateľ zadá IP adresu druhého uzla, port na prijímanie správ a port na odosielanie. Keďže ide o peer-to-peer komunikáciu, po úspešnom spojení si obe strany môžu vymieňať správy až do ukončenia spojenia.[2]. To môže nastať keď jeden z používateľov napíše "**/disconnect**", alebo jednoducho vypne aplikáciu. Správanie uzla je zobrazené diagramom v Obr. 5

[1]Python v3.10.12 s použitím knižníc **socket** a **threading**.
[2]Vďaka vláknam môžu oba uzly správy aj prijímať, aj posielať

3.1 Fragmentácia

Implementácia podporuje fragmentovaný prenos údajov s maximálnou veľkosťou fragmentu definovanou používateľom. Každý fragment správy nesie informácie o svojej pozícii v kompletnej správe a je očíslovaný rovnakým sekvenčným číslom. Postup prenosu dát pri fragmentácii:

1. Veľká správa sa rozdelí na niekoľko menších fragmentov, pričom každá z nich obsahuje časť pôvodnej správy má rovnaké poradové číslo (sequence number v hlavičke protokolu). Každý fragment má osobitné fragmentové číslo, ktoré označuje jeho poradie medzi zvyšnými fragmentami.[3]

2. Prvý odoslaný packet je špeciálny. Ide o režijný packet so sekvenčným číslo 0 a vlajkami ACK=1 a FTR=1. Obsahuje v dátovej časti celkový počet očakávaných fragmentov a meno prenášaného súboru. Ostatné packety s fragmentami súboru sú označené vlajkou FTR=1

3. Prijímateľ postupne dostane všetky fragmenty, pričom podľa poradového čísla rozpozná, že patria k tej istej správe a v akom poradí ich následne treba poskladať. Posledný fragmentový packet súboru je označený vlajkou LFG=1 (Last fragment). Pred poskladaním sa každý fragment skontroluje (vypočíta sa checksum pomocou CRC16). Ak je fragment poškodený, odošle sa naspäť hlásenie o chybe (ERR packet) a fragment sa získa od odosielateľa znova.

4. Nakoniec sa všetky fragmenty zoradia podľa fragmentových čísel a spoja sa do výslednej správy.

3.2 Príkazy a používateľský vstup

Program podporuje tri príkazy:

- **/setfragsize** $< size >$ pre nastavenie veľkosti fragmentov. Po zadaní tohoto príkazu sa odošle druhému uzlu packet s vlajkou SFS=1 a požadovanou veľkosťou fragmentov.

- **/disconnect** pre manuálne ukončenie spojenia (inicializáciu 2-way termination), teda odoslanie FIN packetu.

[3]Maximálna veľkosť fragmentu je okrem používateľského vstupu ohraničená aj maximálnou veľkosťou UDP packetov (2^{16} bitov).

- **/send** $< file >$ pre odoslanie súboru. Pri prijímaní súboru je používateľovi daná možnost uložiť ho v akomkoľvek priečinku (zadaním cesty k nemu). Ak zadaná cesta nie je valídna alebo bol vstup prázdny, súbor sa uloží do priečinka predvoleného systémom.

3.3 Vytváranie a riešenie chýb

Protokol zahŕňa mechanizmus na detekciu a požiadanie o opätovné odoslanie stratených alebo poškodených packetov. Pre ich otestovanie sa v implementácii jeden z packetov vždy poškodí tak, že sa jeho checksum zmení na neplatnú hodnotu. Metódy riešenia problémových packetov sú rozoberané v častiach 2.3, 2.4 a 2.6. Hlavná myšlienka je, že akékoľvek poškodené alebo stratené packety sa opäť vyžiadajú od odosielateľa, a duplikované packety sú ignorované.

3.4 Ukážka programu a analýza wiresharkom

Pre analýzu vo wiresharku sme vytvorili vlastný lua skript. Beh programu a vyzerá nasledovne:

1. Obr. 6: Používateľský vstup pri spustení aplikácie a začatie 3-Way handshake, nižšie sú zobrazené SYN a ACK packety vo wiresharku.

2. Obr. 7: Výmena správ medzi uzlami.

3. Obr. 8: Nastavovanie veľkosti fragmentov a poslaný SFS packet.

4. Obr. 9: Posielanie súboru zobrazeneé na uzle odosielateľa a nižšie na uzle prijímateľa. Môžeme si všimnúť, že prijímateľ zaznamenal jednu chybu pri prijímaní súboru (invalid checksum), kvôli ktorej poslal späť ERR packet s poškodeným packetom. Po prijatí súboru sa objavila možnosť vybrať miesto uloženia. Pri prázdnom vstupe sa súbor uložil na systémom predurčené miesto.

5. Obr. 10: Súborový packet (FTR flag) a ACK packet, ktorý potvrdzuje jeho správne prijatie.

6. Obr. 11: ERR packet pre poškodený packet.

7. Obr. 12: Ukončenie spojenia používateľom a FIN packet.

4 Záver

Úspešne sme navrhli a implementovali vlastný protokol na jednoduchý, no spoľahlivý prenos údajov spôsobom *Stop-and-Wait ARQ*. Protokol umožňuje fragmentáciu správ, kontrolu integrity pomocou CRC-16 a zabezpečuje spoľahlivé spojenie vďaka Keep-Alive systému a nadviazaním spojenia cez 2-way handshake. Prenášané packety sú viditeľné prostredníctvom programu Wireshark a sú v nej rozoznateľné polia nášho vlastného protokolu.

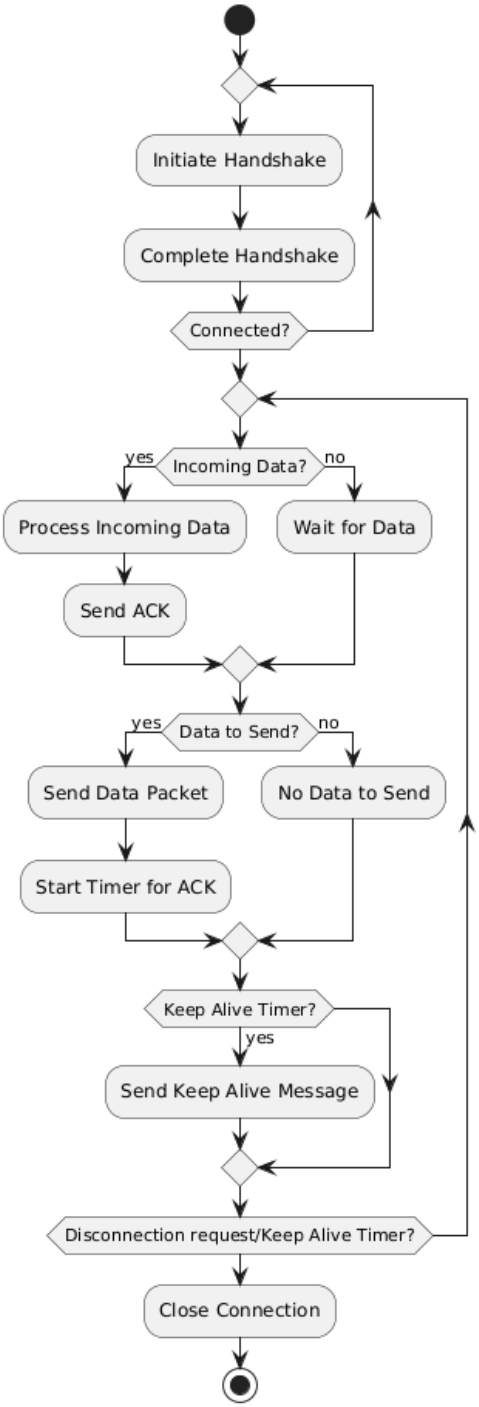
5 Zmeny oproti pôvodnému návrhu

- Zmena spôsobu ukončenia spojenia. Namiesto 4-way termination sa používa 2-way termination.
- Zjednodušenie ARQ systému a pridanie diagramu. Prechod zo S&W na vlastnú variáciu.
- Zjednodušenie vytvárania chýb.
- Pridanie príkazov /setfragsize, /send a úprava /disconnect.
- Prijímanie správ a prijímanie súborov sa deje v rozdielnych threadoch.
- Používateľ si môže vybrať, kam chce prijatý súbor uložiť.
- Prenášané správy a prenášané súbory majú osobitné sequence numbers, pričom packety určené na prenos súborov sú označené flagom FTR (File transfer).
- Z hlavičky protokolu boli z redundantnosti odstránené fragmentové číslo, počet fragmentov a dĺžka údajov.
- Do hlavičky boli pridané vlajky LFG (Last fragment), SFS (Set fragment size), FTR (File transfer).
- V hlavičke

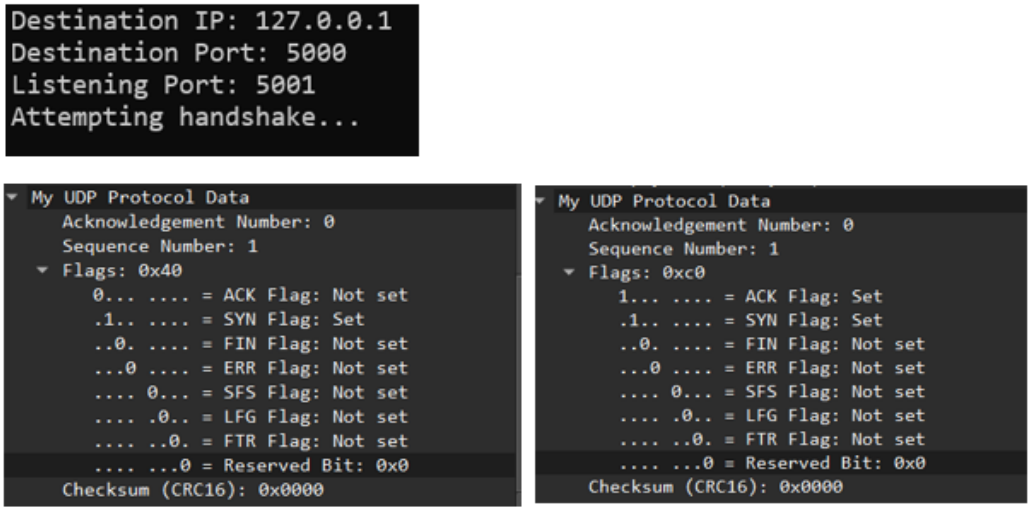
6 Zdroje

- `https://www.planttext.com/`
- `https://app.diagrams.net/`

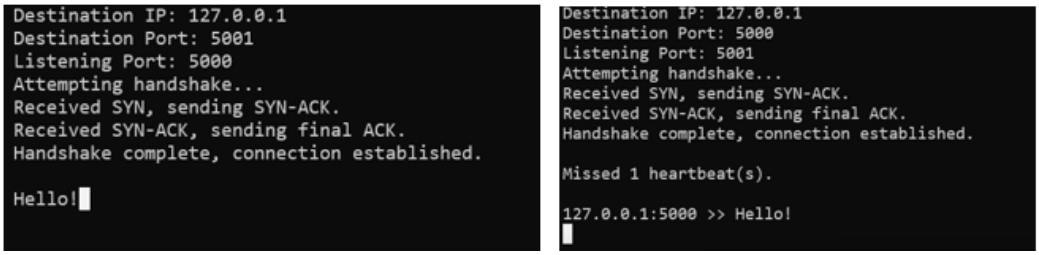
- https://protocol-designer.app/protocols/9d5267ac-fe04-441b-849c-d9c5c5564e9
- https://en.wikipedia.org/wiki/Transmission_Control_Protocol
- https://en.wikipedia.org/wiki/Cyclic_redundancy_check
- STEINMETZ, Ralf – WEHRLE, Klaus. 2005. Peer-to-Peer Systems and Applications. Berlin, 2005. 476 s. ISBN 978-3-540-29192-0



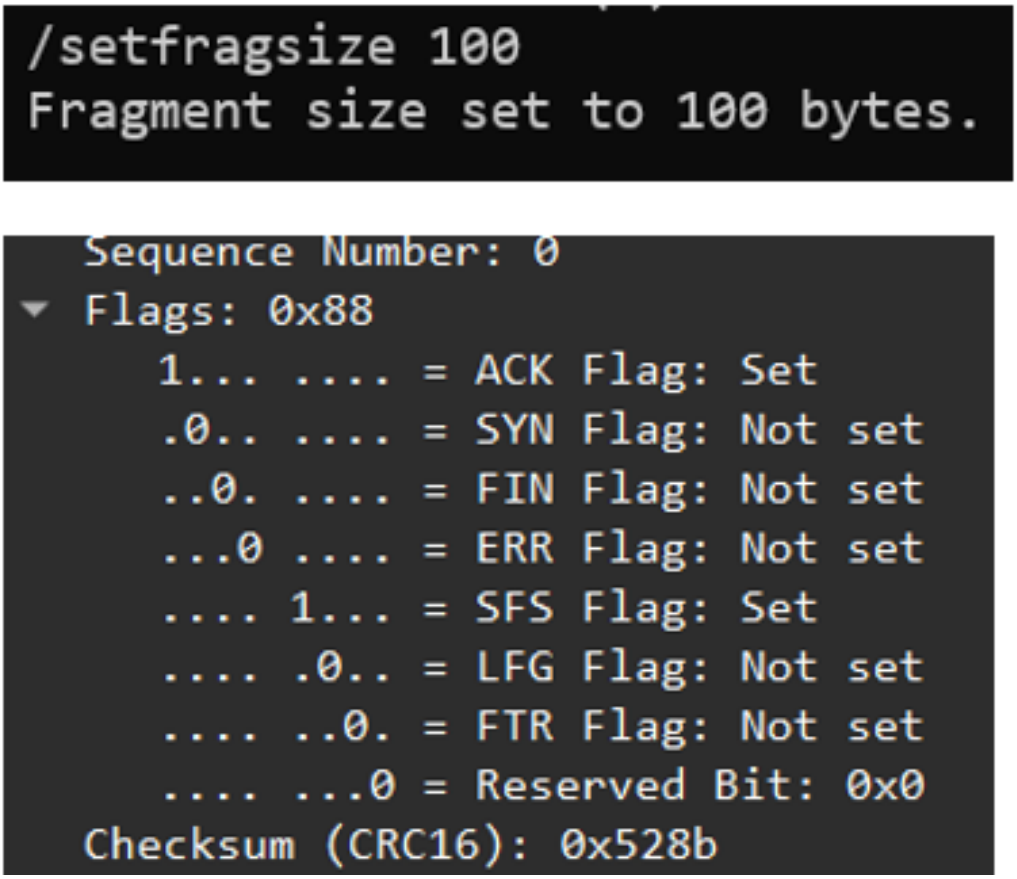
Obr. 5: Diagram opisujúci správanie uzla.



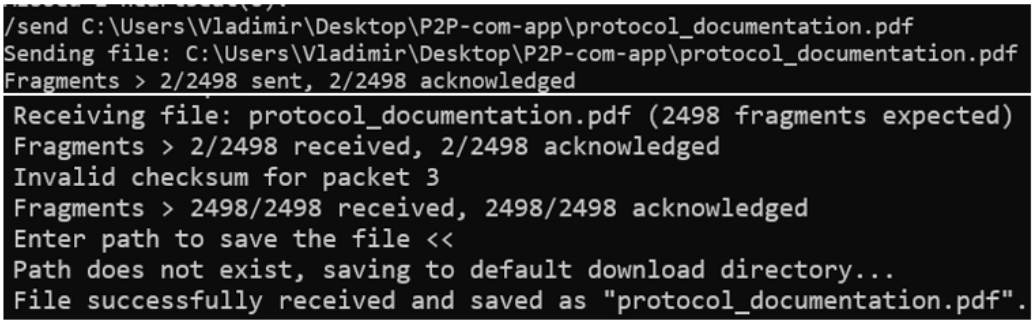
Obr. 6: Používateľský vstup a vymenené handshake packety



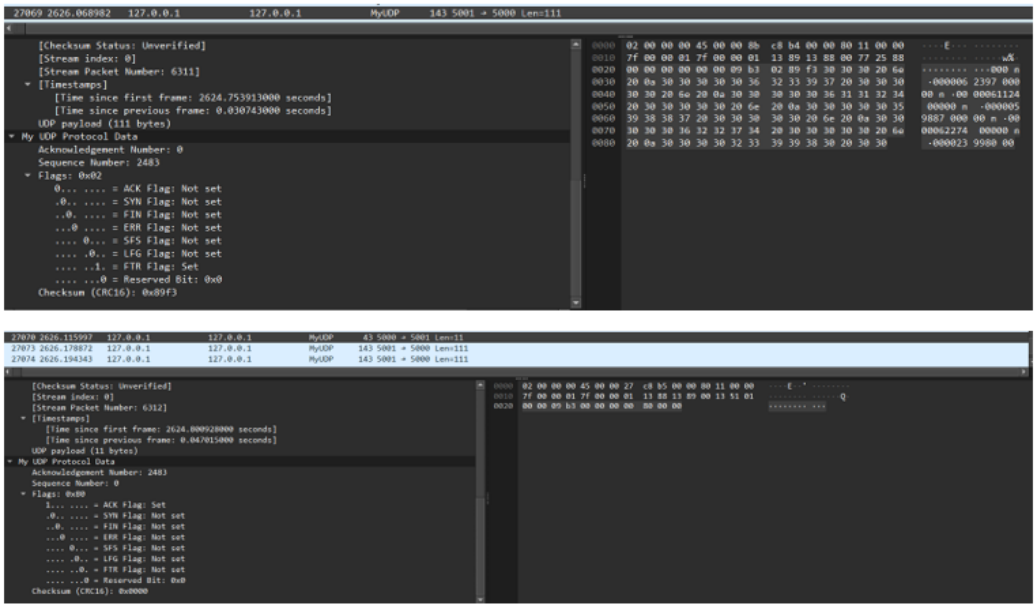
Obr. 7: Posielanie správ



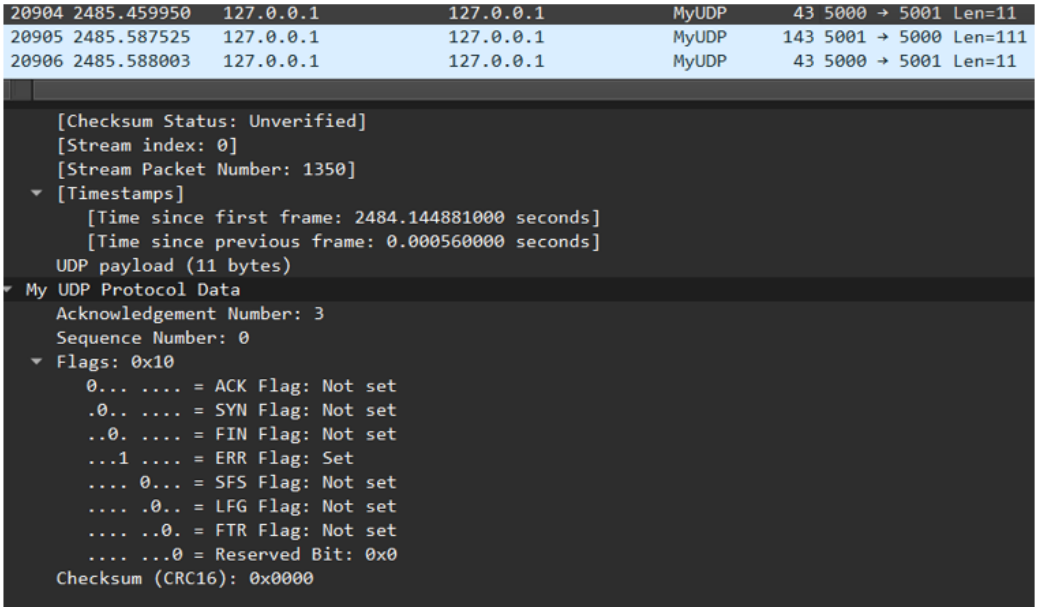
Obr. 8: Nastavenie veľkosti fragmentov



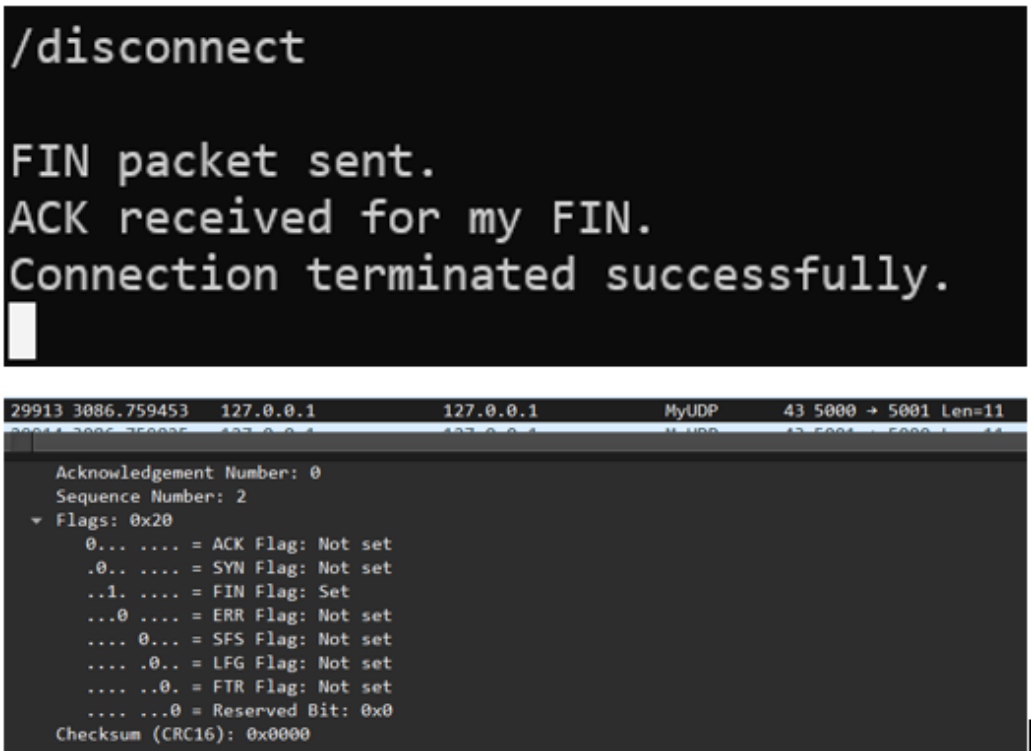
Obr. 9: Posielanie súboru



Obr. 10: FTR a ACK packety pri posielaní súboru



Obr. 11: Error packet



Obr. 12: Ukončenie spojenia