PKS

Nazar Biriukov, 122473

Semestrálne zadanie: Komunikácia s využitím UDP protokolu

Obsah

[Architektúra systému 3](#_Toc183464392)

[Hlavné funkcie (main.py): 3](#_Toc183464393)

[Všeobecné informácie 3](#_Toc183464394)

[Procesy 4](#_Toc183464395)

[2.1 Odosielanie a prijímanie správ 4](#_Toc183464396)

[2.2 Odosielanie a prijímanie súborov 4](#_Toc183464397)

[2.3 Mechanizmus Keep-Alive 5](#_Toc183464398)

[2.4 ARQ 5](#_Toc183464399)

[Diagramy 5](#_Toc183464400)

# Architektúra systému

Protokol je postavený na UDP a umožňuje spoľahlivý prenos údajov prostredníctvom fragmentácie, overovania integrity (CRC16) a potvrdzovania správ (ACK/NACK). Obsahuje nasledujúce moduly:

## Hlavné funkcie (main.py):

### Všeobecné informácie

Tento kód implementuje architektúru klient-server na odosielanie a prijímanie správ, udržiavanie spojenia pomocou mechanizmu keep-alive a prenos súborov medzi dvoma uzlami. Hlavnými úlohami programu sú:

1. Udržiavanie spojenia medzi uzlami pomocou správ PING/PONG.
2. Prenos správ vrátane textových údajov a súborov.
3. Dynamická zmena veľkosti fragmentov pri prenose veľkých dát**.**
4. Fragmentácia akontrolné súčty (calculation.py)
5. Prenos súborov (video.py)
6. Spracovanie hlavičiek (header.py)
7. Správy (messages.py)
8. Sieťová komunikácia (connection.py)

### Hlavné premenné

pong\_received (zoznam s jedným prvkom bool) - príznak prijatia PONG. Používa sa na potvrdenie nadviazania spojenia.

running (zoznam s prvkom bool) - riadi stav operácie programu.

connection\_status (zoznam s prvkom bool) - označuje aktuálny stav spojenia.

missed\_heartbeats (zoznam s int) - počítadlo zmeškaných správ keep-alive.

consecutive\_ack (zoznam s int) - počítadlo potvrdených správ keep-alive.

pause\_keep\_alive (zoznam s bool) - dočasná pauza v keep-alive na vykonanie iných úloh (napr. prenos súborov).

last\_frag\_end (bool) - príznak ukončenia prenosu súboru (používa sa v logike prenosu údajov).

max\_fragment\_size (celé číslo) - maximálna veľkosť fragmentu údajov v bajtoch.

### Hlavný tok programu

**1. Zadávanie údajov o pripojení**

Program požiada používateľa o zadanie nasledujúcich parametrov:

* Port na prijímanie správ.
* IP adresa vzdialeného uzla.
* Port vzdialeného uzla na odosielanie správ.
* Port na odosielanie správ.
* Maximálna veľkosť fragmentu v bajtoch.

2. **Vytváranie zásuviek**

* receive\_server - serverová zásuvka na prijímanie správ.
* client\_socket - klientsky soket na odosielanie údajov.
* Funkcia connection.create\_server(our\_receive\_port) inicializuje serverový soket. client\_socket.bind() viaže klientský soket na zadaný port.

3**. Spracovanie PONG**

Funkcia process\_pong():

* Spúšťa sa, keď je prijatá správa PONG.
* Nastaví príznaky pong\_received a connection\_status na hodnotu True.
* Odošle späť správu „PONG“, ktorá potvrdzuje, že spojenie bolo úspešne nadviazané.

4. **Vlákno na prijímanie správ**

Thread\_receive spustí funkciu messages.receive v samostatnom vlákne na spracovanie prichádzajúcich údajov. Odovzdáva sa:

* Socket (receive\_server, client\_socket).
* IP a port vzdialeného uzla.
* Odkazy na príznaky a počítadlá na kontrolu stavu spojenia.
* Vlákno je démon, čo znamená, že sa automaticky ukončí, keď sa ukončí hlavný program.

5. **Nadviazanie spojenia**

Program odošle počiatočný PING vzdialenému uzlu prostredníctvom príkazu connection.ping. Potom čaká na odpoveď PONG kontrolou pong\_received. Ak sa neprijme žiadny PONG, program počká a požiadavku zopakuje.

6. **Mechanizmus Keep-Alive**

Funkcia keep\_alive\_wrapper() obaľuje volanie connection.keep\_alive a beží v samostatnom vlákne (thread\_keep\_alive), ktoré:

* Odosiela pravidelné keep-alive správy.
* kontroluje stav spojenia.
* V prípade potreby reštartuje spojenie.

7. **Hlavná slučka programu**

Táto slučka vykonáva nasledujúce činnosti:

* Čaká na vstupný príkaz od používateľa.
* V závislosti od príkazu vykoná jednu z akcií:
* exit - ukončí program.
* file - prenesie súbor:
* Pozastaví sa v režime keep-alive (pause\_keep\_alive).
* Odošle sa signál „FILE\_RECEIVE“.
* Čaká na potvrdenie (simulované oneskorením time.sleep(2)).
* Odošle súbor prostredníctvom video.send\_file.
* size - zmena veľkosti fragmentu:
* Vyžiada si od používateľa novú veľkosť.
* Kontroluje jej platnosť.
* Odošle správu „SIZE\_CHANGE“ do vzdialeného uzla.
* Aktualizuje veľkosť lokálne prostredníctvom calculation.fragment\_change.
* Akýkoľvek iný reťazec - odošle textovú správu prostredníctvom messages.sending.
* Ak sa počas procesu vyskytnú chyby, zachytí ich blok try-except.

8. **Ukončenie programu**

Pri behu sa ukončí:

* Príznaky running a connection\_status sú nastavené na False.
* Všetky zásuvky sú uzavreté.

### Modul messages.py

uľahčuje odosielanie a prijímanie správ a fragmentov súborov cez spojenie socket UDP. Obsahuje injekciu chýb na testovanie, spracováva príkazy špecifické pre protokol a spravuje komunikáciu v sieťovej aplikácii. Nižšie je uvedené vysvetlenie kľúčových komponentov:

**Importy**

* hlavička: Poskytuje funkcie na vytváranie a analyzovanie hlavičiek paketov.
* Výpočet: Obsahuje obslužné funkcie na fragmentáciu údajov a výpočet kontrolných súčtov.
* video: Spravuje operácie súvisiace so súbormi, napríklad prijímanie a ukladanie súborov.
* os: Používa sa na overovanie adresárov.
* random: Generuje náhodnosť na simuláciu chýb.

**Funkcie**

1. sending()

Účel: Spracúva fragmentáciu správ a ich odosielanie ako paketov UDP. Podporuje vstrekovanie chýb na testovanie spoľahlivosti prenosu.

Parametre:

client\_socket: Objekt UDP socket na odosielanie dát.

message (str): Správa, ktorá sa má odoslať.

foreign\_ip (str): IP adresa prijímajúceho zariadenia.

foreign\_send\_port (int): Port na prijímajúcom zariadení.

max\_fragment\_size (int, predvolené=1024): Maximálna veľkosť každého fragmentu.

inject\_error (bool, predvolené=True): Ak je True, simuluje chyby prenosu s pravdepodobnosťou 3 %.

Ako to funguje:

Zakóduje správu do bajtov.

Fragmentuje správu na časti pomocou funkcie calculation.fragment\_data().

Pre každý fragment:

Vypočíta kontrolný súčet pomocou funkcie calculation.calculate\_checksum().

Voliteľne vloží chyby na testovanie.

Skonštruuje paket so záhlavím a fragmentom.

Odošle paket prostredníctvom client\_socket.sendto().

2. **receive()**

Účel: Nepretržite počúva prichádzajúce pakety UDP, dekóduje ich a spracováva rôzne príkazy.

Parametre:

server: Počúvajúci objekt UDP socket.

client\_socket: Objekt UDP socket na odpovedanie na správy.

foreign\_ip, foreign\_send\_port: Adresa a port komunikačného partnera.

process\_pong:

* Funkcia spätného volania na spracovanie odpovedí PONG.
* running (zoznam): Zdieľaný príznak na riadenie vlákna prijímača.
* connection\_status (zoznam): Sleduje aktívny stav spojenia.
* missed\_heartbeats (zoznam): Sleduje zmeškané signály srdcového tepu pre stav spojenia.
* consecutive\_ack, pause\_keep\_alive, last\_frag\_end: Spoločné stavové premenné na správu protokolu.

3. exit\_socket()

Účel: Odošle vzdialenému zariadeniu príkaz na ukončenie, aby sa spojenie elegantne ukončilo.

Parametre:

* client\_socket: Zásuvka použitá na odosielanie.
* foreign\_ip: IP adresa príjemcu.
* foreign\_send\_port: Port príjemcu.
* Použitie:
* Volanie funkcie sending() so správou „/exit“.

# Procesy

## 2.1 Odosielanie a prijímanie správ

**Popis:**

* Odosielateľ najskôr fragmentuje správu, ak je jej veľkosť väčšia ako maximálna povolená veľkosť fragmentu.
* Každý fragment obsahuje hlavičku s údajmi o dĺžke, poradovom čísle, príznakoch a CRC16.
* Prijímateľ dekóduje hlavičku, overuje kontrolný súčet a posiela ACK (alebo NACK).

## 2.2 Odosielanie a prijímanie súborov

**Popis:**

* Súbor sa číta v binárnom režime a delí na fragmenty.
* Každý fragment obsahuje hlavičku so základnými informáciami a je odoslaný s kontrolným súčtom.
* Ak sa prijme NACK, fragment sa znova odošle.

## 2.3 Mechanizmus Keep-Alive

**Pracovná logika**

* Kontrola stavu pauzy a pripojenia:
* Ak je pause\_keep\_alive[0] == False a spojenie je aktívne (connection\_status[0] == True), začne sa posielať HEARTBEAT.
* Ak pause\_keep\_alive[0] == True, funkcia počká 1 sekundu pred ďalšou kontrolou.
* Odosielanie HEARTBEAT
* Správa HEARTBEAT sa odošle do vzdialeného zariadenia prostredníctvom funkcie messages.sending.
* Čakanie na odpoveď (HEARTBEAT\_ACK):
* Nastaví sa časový limit zásuvky (5 sekúnd).
* Ak počas tohto času príde správa HEARTBEAT\_ACK, počítadlo missed\_heartbeats[0] sa vynuluje na 0.
* Spracovanie zmeškaných odpovedí:
* Ak nie je prijatá žiadna odpoveď, čítač missed\_heartbeats[0] sa zvýši.
* Ak sú zmeškané 3 správy HEARTBEAT\_ACK za sebou, spojenie sa považuje za prerušené:
* Odošle sa signál \exit.
* Príznaky connection\_status[0] a running[0] sú nastavené na False.
* Zásuvka sa uzavrie.
* Oneskorenie medzi iteráciami:
* Ak je aktívna pauza alebo je odoslaný HEARTBEAT, funkcia pred ďalšou kontrolou urobí oneskorenie 1 sekundu.

## 2.4 ARQ Stop And Wait

## 1. odosielanie dát s fragmentáciou:

## V module messages.py funkcia odosielania rozdelí dáta na fragmenty (pomocou calculation.fragment\_data) a odošle ich postupne.

## Pre každý fragment sa pridá hlavička, ktorá obsahuje kontrolný súčet, príznak posledného fragmentu a číslo fragmentu.

2**. Prijímanie potvrdení (ACK/NACK):**

Po prijatí fragmentu (vo funkcii receive\_file z video.py) sa vykoná kontrola kontrolného súčtu.

Ak sa kontrolný súčet nezhoduje, odošle sa NACK so žiadosťou o opätovné odoslanie poškodeného fragmentu.

Ak je fragment prijatý úspešne, je odoslaný ACK, ktorý informuje odosielateľa.

**3. časový limit a opätovné odoslanie:**

* **Odosielateľ čaká na potvrdenie (ACK/NACK) určitý čas (settimeout). Ak sa v tomto čase neprijme žiadne potvrdenie, paket sa odošle znova.**
* **Výhody metódy Stop-and-Wait ARQ:**
* **Jednoduchosť implementácie: jednoduchá implementácia a ladenie.**
* **Spoľahlivosť: zabezpečuje správne údaje aj v prípade chýb alebo strát.**
* **Nevýhody metódy Stop-and-Wait ARQ:**
* **Nízka výkonnosť: odosielateľ čaká na potvrdenie každého paketu, čo vedie k neefektívnemu využitiu kanála s vysokým oneskorením.**
* **Nevhodné pre vysokorýchlostné siete: oneskorenia medzi odosielaním paketov výrazne znižujú priepustnosť**

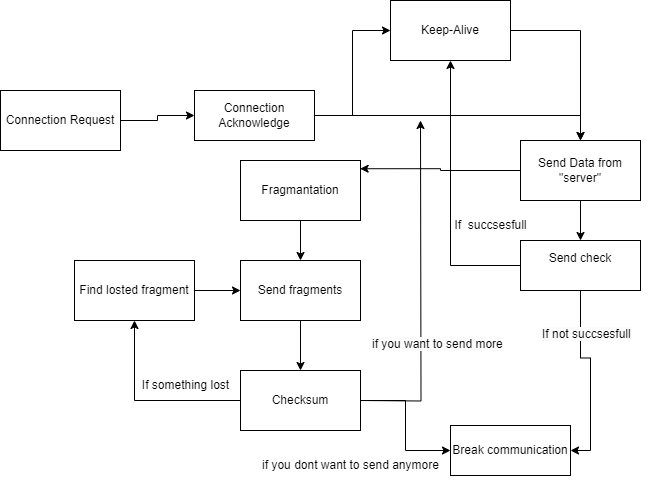
## 2.5 CRC 16

Kroky výpočtu CRC-16

Inicializácia:

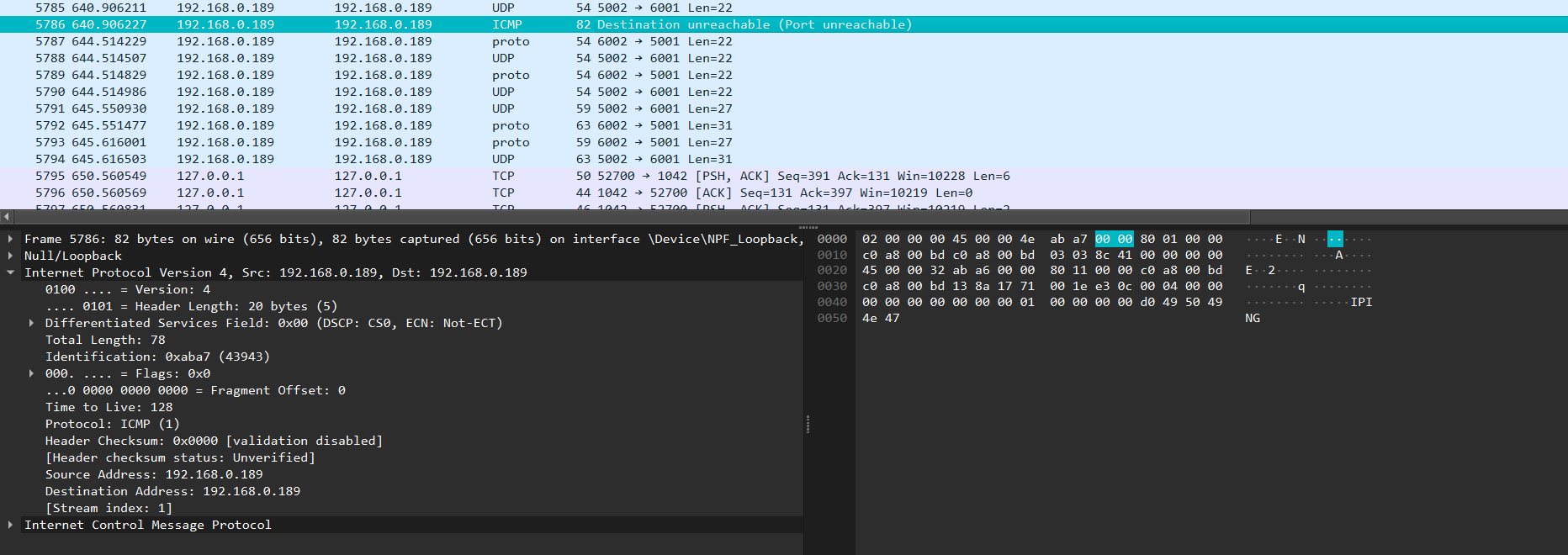
* Definuje sa polynóm použitý v algoritme (napr. 0x1021 pre CRC-16-CCITT).
* Nastaví sa počiatočná hodnota registra CRC (zvyčajne 0xFFFF alebo 0x0000).
* Spracovanie údajov bit po bite:
* Každý bajt údajov sa spracuje bitovo, počnúc vysokým bitom.
* Vysoký bit registra XOR sa XORuje s vysokým bitom aktuálneho bajtu.
* Register sa posunie doľava (o jeden bit).
* Ak je po posune doľava vysoký bit registra 1, na register sa aplikuje XOR s polynómom.
* Opakujte pre všetky bajty:
* Proces sa opakuje pre všetky bajty v údajoch.
* Celková hodnota:
* Po spracovaní všetkých bajtov sú údaje v registri celkovým kontrolným súčtom CRC-16

# Diagrammyprotocol.drawio.pngy

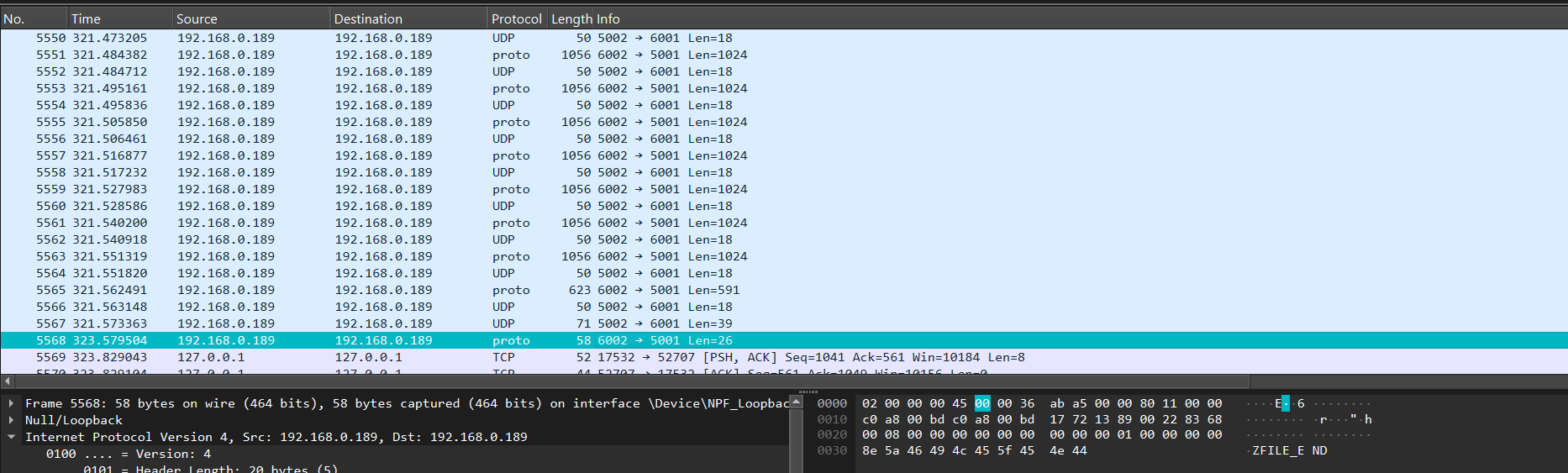


# 

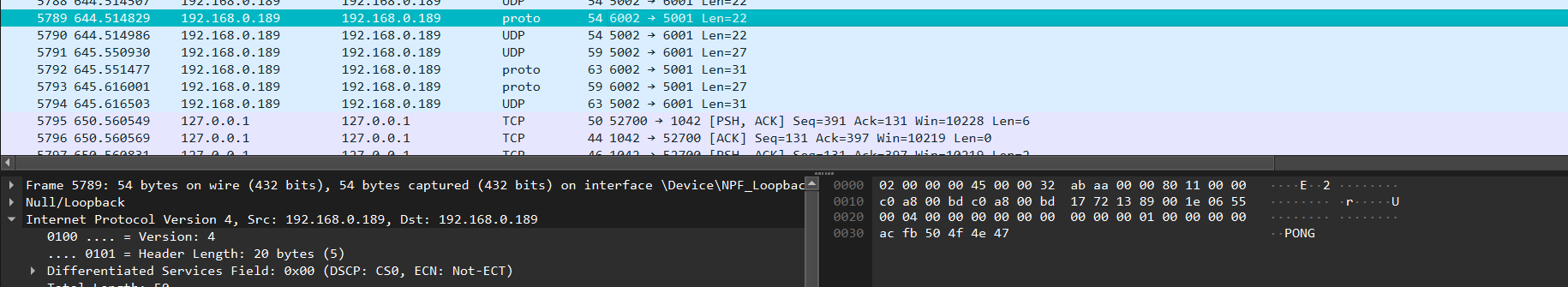
# WireShark



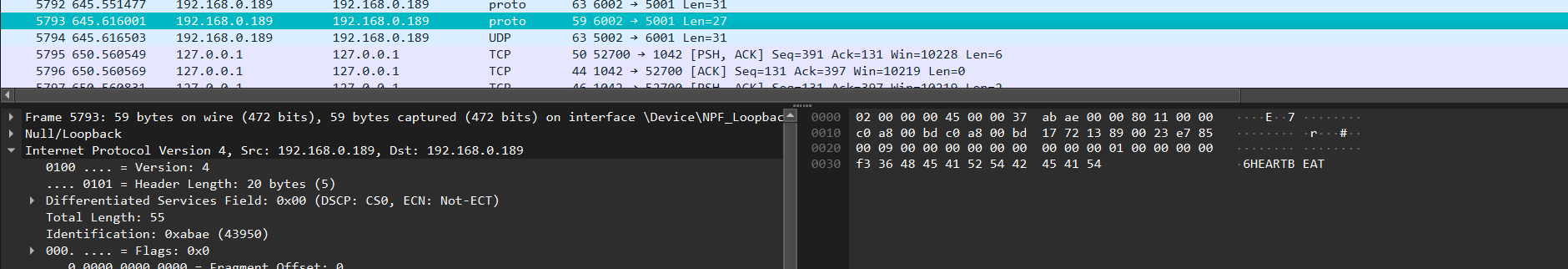
PING

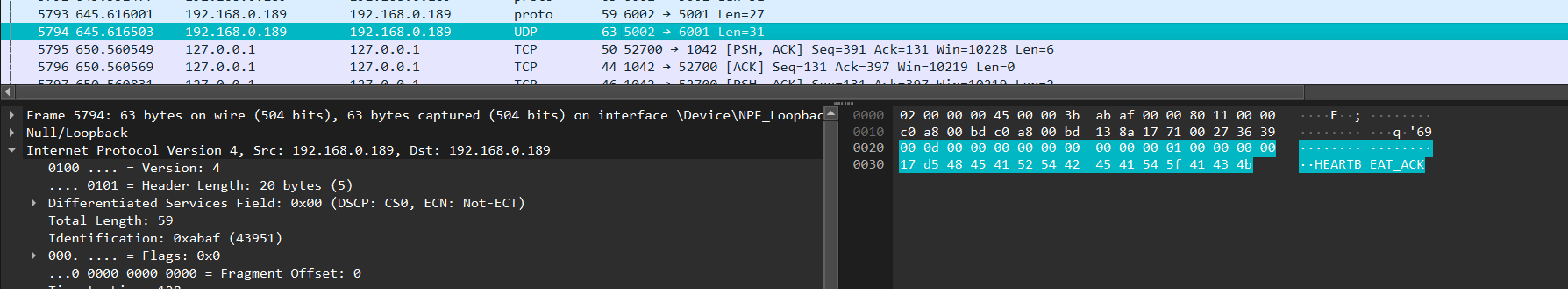


Dokončite odosielanie súboru



PONG



HEARTBEAT

HEARTBEAT\_ACK