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

Dokumentácia – Komunikácia s využitím UDP protokolu

Dávid Penťa ID: 110871

Počítačové a komunikačné siete

FIIT STU 2021/2022

Zadanie úlohy

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.

Zmeny oproti návrhu

Jediná zmena oproti návrhu bola že som zmenil typy packetov, aby som mohol rozoznať správu od súboru, a v súbore názov súboru od jeho obsahu. Tiež som pridal nové typy aby som mohol začať a ukončiť posielanie správ na udržiavanie spojenia, a vrátiť server do hlavného menu ak sa tam vrátil klient.

Fungovanie riešenia

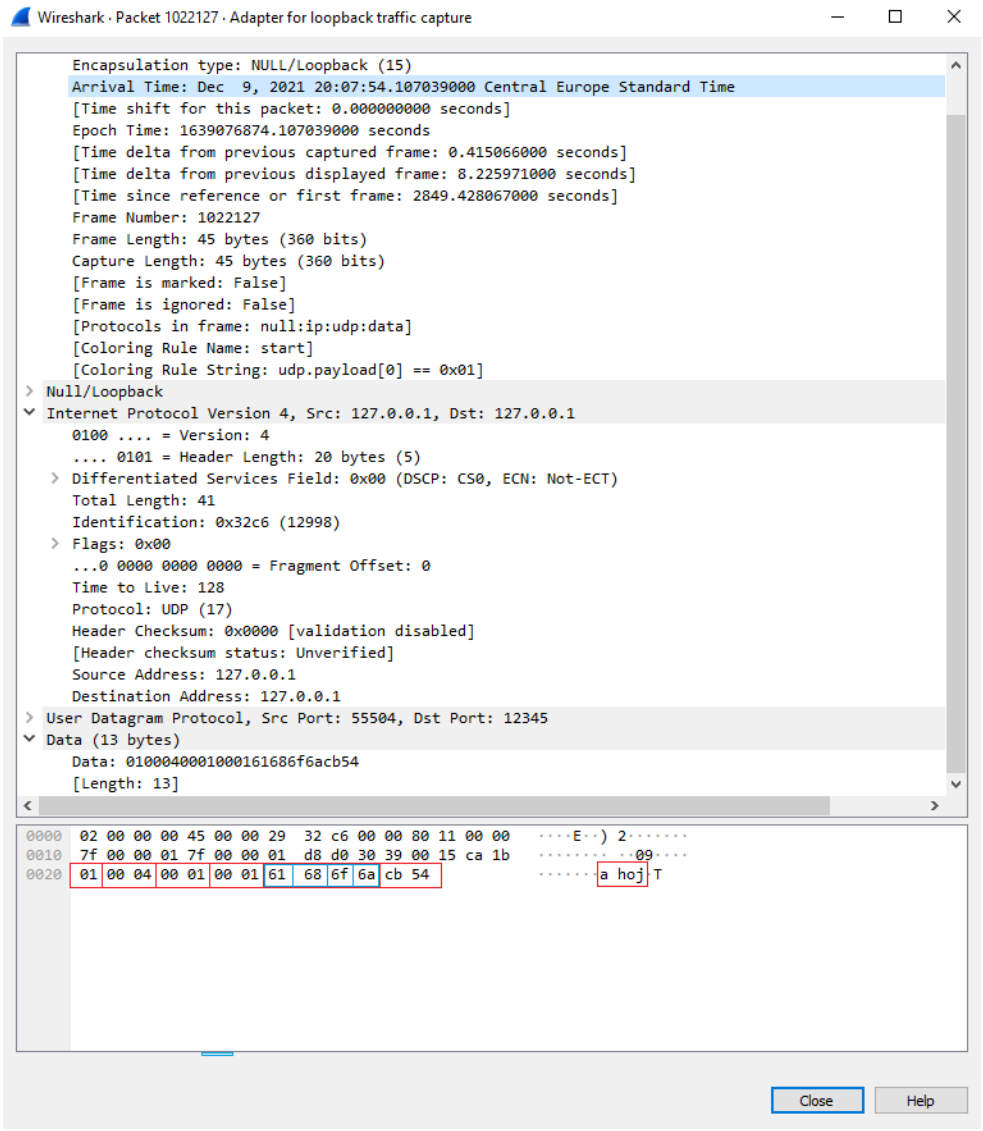
Používateľ si na začiatku vyberie, či chce byť server alebo klient. Tiež si môže vybrať program ukončiť. Ak si vyberie, že chce byť klient, musí zadať IP adresu serveru, port a . Potom ak stlačí CTRL + C, tak si môže vybrať, či sa chce vrátiť do hlavného menu, alebo chce odoslať súbor alebo správu. Ak si vyberie súbor, tak musí zadať cestu k súboru. Klient môže určiť maximálnu dĺžku fragmentu, najviac však 1463. Ak zadá príliš nízku dĺžku fragmentu, ktorá by vytvorila viac ako 65535 fragmentov, tak program automaticky vypočíta najmenšiu možnú dĺžku fragmentu tak aby sa poslalo menej ako 65535 fragmentov. Toto sa musí urobiť aby sa počet fragmentov a poradové číslo aktuálneho fragmentu zmestili do hlavičky. Po odoslaní sa používateľovi zobrazí, či bolo prijatie správy serverom úspešné. Ak server nebude odpovedať 30 sekúnd na správy udržania spojenia, tak spojenie so serverom ukončí. Ak si používateľ vyberie, že chce byť server, tak musí nastaviť port kam sa budú ukladať prijaté súbory. Ak používateľ stlačí CTRL + C, má možnosť vrátiť sa do hlavného menu. Po inicializovaní komunikácie sa vypíše správa, ako aj pri úspešnom prijatí

všetkých fragmentov komunikácie. Všetky pakety ktoré prišli, budú skontrolované pomocou CRC-16-CCITT ich poradového čísla. Počet paketov o ktoré prišli poškodené sa vypíše. Ak klient 30 sekúnd nepošle žiadnu správu, tak server ukončí spojenie.

Navrhnutá štruktúra hlavičky

1 8	1 8	1 8	1 8	1 8	1 8	1 8	1 8	1 8
Typ	Veľkosť dát		Poradie fragmentu		Počet všetkých fragmentov		Dáta	Checksum

Štruktúra hlavičky vo Wiresharku



Typ – 1 bajt

0. Typ: Správa pre udržiavanie spojenia
1. Typ: Začiatok správy
2. Typ: Koniec správy
3. Typ: Chyba – Dáta ktoré prišli boli chybné
4. Typ: Obsah správy alebo súboru v už začatej komunikácii
5. Typ: Potvrdenie že dáta boli prijaté
6. Typ: Zastavenie posielania správ pre udržiavanie spojenia
7. Typ: Obnovenie posielania správ pre udržiavanie spojenia
8. Typ: Klient ukončil spojenie
9. Typ: Začiatok súboru
10. Typ: Koniec názvu súboru a začiatok obsahu súboru

Veľkosť dát – 2 bajty

Veľkosť dát momentálneho fragmentu v bajtoch.

Poradie fragmentu – 2 bajty

Poradové číslo momentálneho fragmentu.

Počet všetkých fragmentov – 2 bajty

Celkový počet všetkých fragmentov na ktoré bol súbor rozdelený.

Checksum – 2 bajty

K dátam sa na koniec pripoja dva bajty, získané pomocou CRC-16-CCITT, na kontrolu či nedošlo k poškodeniu dát počas prenosu.

Metóda kontrolnej sumy

Ako metódu kontrolnej sumy som si zvolil kontrolu cyklickým kódom (CRC), a to konkrétne štandard CRC-16-CCITT.

Dáta: Hi!

Dáta v binárnej forme: 01001000 01101001 00100001

K dátam sa pridajú dva prázdne bajty.

Dáta v binárnej forme: 01001000 01101001 00100001 00000000 00000000

Polynóm vytvorený z dát: $x^{38} + x^{35} + x^{30} + x^{29} + x^{27} + x^{24} + x^{21} + x^{16}$

Polynóm vytvorený z dát sa vydelí polynómom zo štandardu CRC-16-CCITT, $x^{16} + x^{12} + x^5 + 1$, podľa aritmetiky konečného poľa.

$x^{38} + x^{35} + x^{30} + x^{29} + x^{27} + x^{24} + x^{21} + x^{16} \div x^{16} + x^{12} + x^5 + 1 = x^{13} + x^{12} + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + 1$

Výsledný polynóm v binárnej sústave: 00110001 11111101

Ten sa pridá k pôvodným dátam, ktoré sa takto pošlú: 01001000 01101001 00100001 00110001 11111101

Kontrola prebieha tak že polynóm vytvorený z poslanej správy, $x^{38} + x^{35} + x^{30} + x^{29} + x^{27} + x^{24} + x^{21} + x^{16} + x^{13} + x^{12} + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + 1$, musí byť deliteľný podľa aritmetiky konečného poľa bez zvyšku, polynómom zo štandardu CRC-16-CCITT, $x^{16} + x^{12} + x^5 + 1$.

Ak dôjde k zmene dát, zvyšok bude pravdepodobne iný ako 0.

Na výpočet CRC-16-CCITT som informácie zo stránky https://www.lammertbies.nl/comm/info/crc-calculation a časť kódu zo stránky https://stackoverflow.com/questions/35205702/calculating-crc16-in-python.

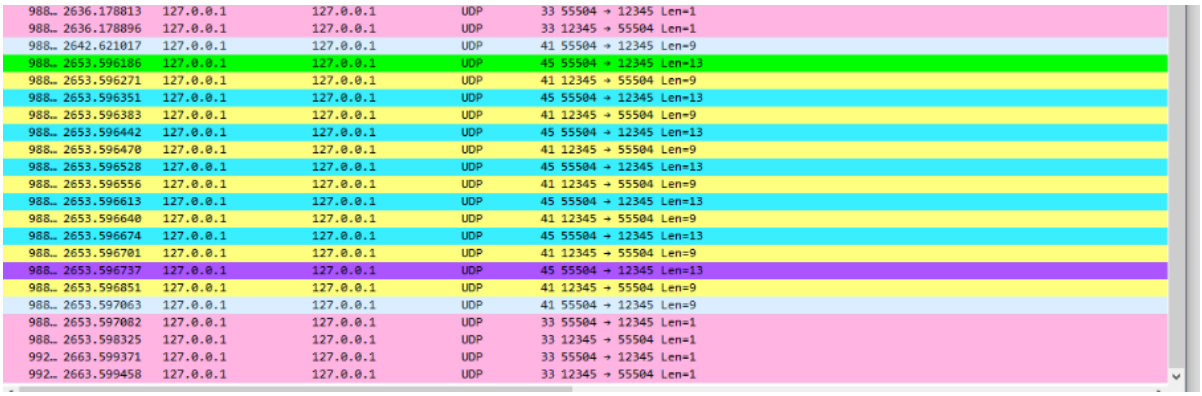
Kód ktorým počítam CRC-16-CCITT:

```
def crc16(data):
    crc = 0x0000
    for i in range(0, len(data)):
        crc ^= data[i] << 8
        for j in range(0, 8):
            if (crc & 0x8000) > 0:
                crc = (crc << 1) ^ 0x1021
            else:
                crc = crc << 1
    return crc & 0xFFFF
```

Metóda fungovania ARQ

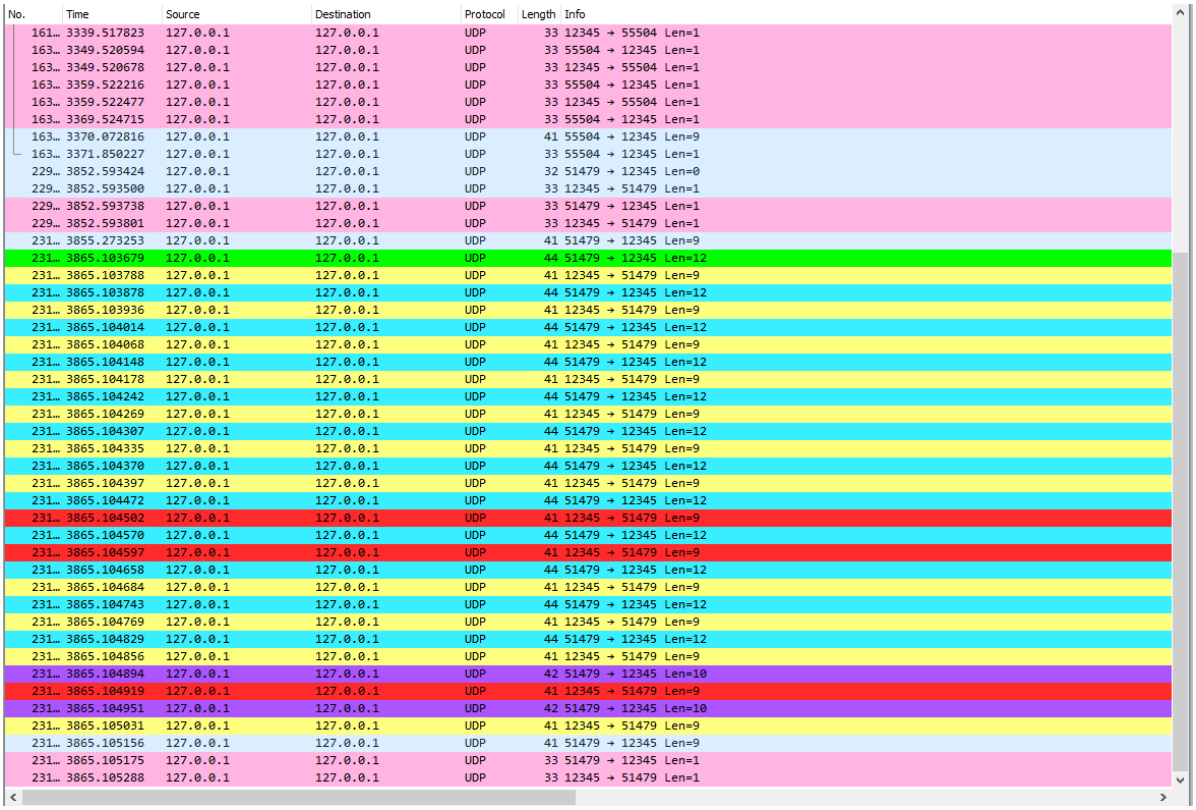
Ako metódu ARQ (Automatic Repeat reQuest) som si zvolil stop and wait. Takže server bude na každý packet v ktorom nebola nájdená chyba odpovedať packetom typu 5. Ak v pakete ktorý prišiel server chybu nájde, tak odpovie paketom typu 3. Klient každú packet odosiela až dokým nedostane odpoveď paketom typu 5.

Komunikácia bez chýb



988…	2636.178813	127.0.0.1	127.0.0.1	UDP	33 55504 → 12345 Len=1
988…	2636.178896	127.0.0.1	127.0.0.1	UDP	33 12345 → 55504 Len=1
988…	2642.621017	127.0.0.1	127.0.0.1	UDP	41 55504 → 12345 Len=9
988…	2653.596186	127.0.0.1	127.0.0.1	UDP	45 55504 → 12345 Len=13
988…	2653.596271	127.0.0.1	127.0.0.1	UDP	41 12345 → 55504 Len=9
988…	2653.596351	127.0.0.1	127.0.0.1	UDP	45 55504 → 12345 Len=13
988…	2653.596383	127.0.0.1	127.0.0.1	UDP	41 12345 → 55504 Len=9
988…	2653.596442	127.0.0.1	127.0.0.1	UDP	45 55504 → 12345 Len=13
988…	2653.596470	127.0.0.1	127.0.0.1	UDP	41 12345 → 55504 Len=9
988…	2653.596528	127.0.0.1	127.0.0.1	UDP	45 55504 → 12345 Len=13
988…	2653.596556	127.0.0.1	127.0.0.1	UDP	41 12345 → 55504 Len=9
988…	2653.596613	127.0.0.1	127.0.0.1	UDP	45 55504 → 12345 Len=13
988…	2653.596640	127.0.0.1	127.0.0.1	UDP	41 12345 → 55504 Len=9
988…	2653.596674	127.0.0.1	127.0.0.1	UDP	45 55504 → 12345 Len=13
988…	2653.596701	127.0.0.1	127.0.0.1	UDP	41 12345 → 55504 Len=9
988…	2653.596737	127.0.0.1	127.0.0.1	UDP	45 55504 → 12345 Len=13
988…	2653.596851	127.0.0.1	127.0.0.1	UDP	41 12345 → 55504 Len=9
988…	2653.597063	127.0.0.1	127.0.0.1	UDP	41 55504 → 12345 Len=9
988…	2653.597082	127.0.0.1	127.0.0.1	UDP	33 55504 → 12345 Len=1
988…	2653.598325	127.0.0.1	127.0.0.1	UDP	33 12345 → 55504 Len=1
992…	2663.599371	127.0.0.1	127.0.0.1	UDP	33 55504 → 12345 Len=1
992…	2663.599458	127.0.0.1	127.0.0.1	UDP	33 12345 → 55504 Len=1

Komunikácia s chybami



No.	Time	Source	Destination	Protocol	Length	Info
161…	3339.517823	127.0.0.1	127.0.0.1	UDP	33	12345 → 55504 Len=1
163…	3349.520594	127.0.0.1	127.0.0.1	UDP	33	55504 → 12345 Len=1
163…	3349.520678	127.0.0.1	127.0.0.1	UDP	33	12345 → 55504 Len=1
163…	3359.522216	127.0.0.1	127.0.0.1	UDP	33	55504 → 12345 Len=1
163…	3359.522477	127.0.0.1	127.0.0.1	UDP	33	12345 → 55504 Len=1
163…	3369.524715	127.0.0.1	127.0.0.1	UDP	33	55504 → 12345 Len=1
163…	3370.072816	127.0.0.1	127.0.0.1	UDP	41	55504 → 12345 Len=9
163…	3371.850227	127.0.0.1	127.0.0.1	UDP	33	55504 → 12345 Len=1
229…	3852.593424	127.0.0.1	127.0.0.1	UDP	32	51479 → 12345 Len=0
229…	3852.593500	127.0.0.1	127.0.0.1	UDP	33	12345 → 51479 Len=1
229…	3852.593738	127.0.0.1	127.0.0.1	UDP	33	51479 → 12345 Len=1
229…	3852.593801	127.0.0.1	127.0.0.1	UDP	33	12345 → 51479 Len=1
231…	3855.273253	127.0.0.1	127.0.0.1	UDP	41	51479 → 12345 Len=9
231…	3865.103679	127.0.0.1	127.0.0.1	UDP	44	51479 → 12345 Len=12
231…	3865.103788	127.0.0.1	127.0.0.1	UDP	41	12345 → 51479 Len=9
231…	3865.103878	127.0.0.1	127.0.0.1	UDP	44	51479 → 12345 Len=12
231…	3865.103936	127.0.0.1	127.0.0.1	UDP	41	12345 → 51479 Len=9
231…	3865.104014	127.0.0.1	127.0.0.1	UDP	44	51479 → 12345 Len=12
231…	3865.104068	127.0.0.1	127.0.0.1	UDP	41	12345 → 51479 Len=9
231…	3865.104148	127.0.0.1	127.0.0.1	UDP	44	51479 → 12345 Len=12
231…	3865.104178	127.0.0.1	127.0.0.1	UDP	41	12345 → 51479 Len=9
231…	3865.104242	127.0.0.1	127.0.0.1	UDP	44	51479 → 12345 Len=12
231…	3865.104269	127.0.0.1	127.0.0.1	UDP	41	12345 → 51479 Len=9
231…	3865.104307	127.0.0.1	127.0.0.1	UDP	44	51479 → 12345 Len=12
231…	3865.104335	127.0.0.1	127.0.0.1	UDP	41	12345 → 51479 Len=9
231…	3865.104370	127.0.0.1	127.0.0.1	UDP	44	51479 → 12345 Len=12
231…	3865.104397	127.0.0.1	127.0.0.1	UDP	41	12345 → 51479 Len=9
231…	3865.104472	127.0.0.1	127.0.0.1	UDP	44	51479 → 12345 Len=12
231…	3865.104502	127.0.0.1	127.0.0.1	UDP	41	12345 → 51479 Len=9
231…	3865.104570	127.0.0.1	127.0.0.1	UDP	44	51479 → 12345 Len=12
231…	3865.104597	127.0.0.1	127.0.0.1	UDP	41	12345 → 51479 Len=9
231…	3865.104658	127.0.0.1	127.0.0.1	UDP	44	51479 → 12345 Len=12
231…	3865.104684	127.0.0.1	127.0.0.1	UDP	41	12345 → 51479 Len=9
231…	3865.104743	127.0.0.1	127.0.0.1	UDP	44	51479 → 12345 Len=12
231…	3865.104769	127.0.0.1	127.0.0.1	UDP	41	12345 → 51479 Len=9
231…	3865.104829	127.0.0.1	127.0.0.1	UDP	44	51479 → 12345 Len=12
231…	3865.104856	127.0.0.1	127.0.0.1	UDP	41	12345 → 51479 Len=9
231…	3865.104894	127.0.0.1	127.0.0.1	UDP	42	51479 → 12345 Len=10
231…	3865.104919	127.0.0.1	127.0.0.1	UDP	41	12345 → 51479 Len=9
231…	3865.104951	127.0.0.1	127.0.0.1	UDP	42	51479 → 12345 Len=10
231…	3865.105031	127.0.0.1	127.0.0.1	UDP	41	12345 → 51479 Len=9
231…	3865.105156	127.0.0.1	127.0.0.1	UDP	41	51479 → 12345 Len=9
231…	3865.105175	127.0.0.1	127.0.0.1	UDP	33	51479 → 12345 Len=1
231…	3865.105288	127.0.0.1	127.0.0.1	UDP	33	12345 → 51479 Len=1

Metóda pre udržanie spojenia

Spojenie sa udržiava tak, že ak neprebieha iná komunikácia, tak klient každých 10 sekúnd odošle paket typu 0 a očakáva tiež odpoveď od serveru paketom typu 0. Server na každý prijatý paket typu 0, odpovie zaslaním paketu, ktorý je tiež typu 0. Ak neprebieha iná komunikácia a jedna strana 30 sekúnd nedostane paket typu 0, spojenie medzi uzlami sa ukončí. Paket typu 0 v hlavičke obsahuje len svoj typ.

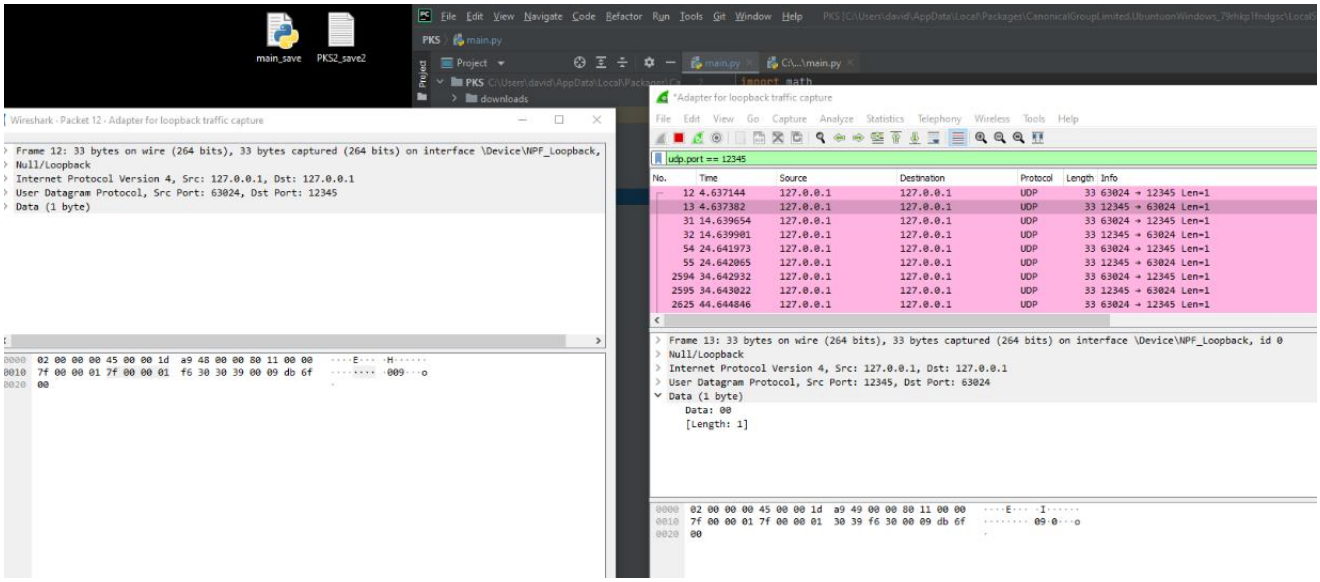
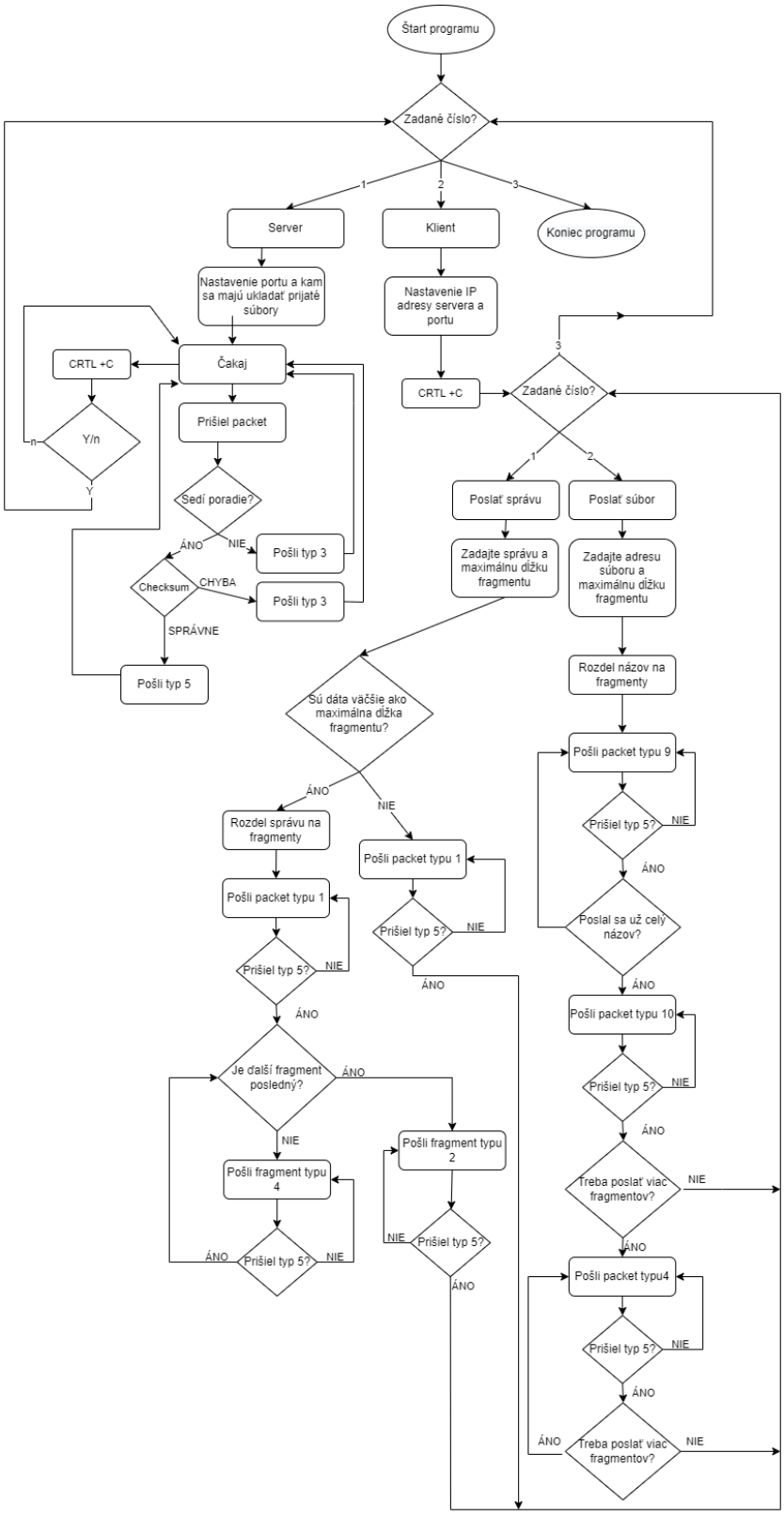


Diagram spracovávania



Voľba implementačného prostredia

Môj program som naprogramoval v programovacom jazyku Python 3 v IDE PyCharm od JetBrains. Je to pretože sa v ňom pracuje jednoduchšie s dátami ako v jazykoch C a C++, a má veľké množstvo funkcií, ktoré tieto jazyky nemajú. Používam modul socket na prácu s UDP protokolom. Na testovanie používam virtuálny stroj Linux Ubuntu ale aj prenos medzi dvomi fyzickými počítačmi.