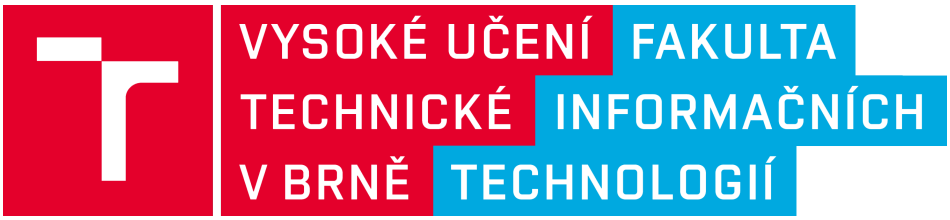



# Implementačná dokumentácia k druhému IPK projektu: Sniffer packetov

Veronika Molnárová, xmolna08

23.4.2022

# Obsah

# 1 Úvod

Cieľom projektu bolo vytvoriť jednoduchý packet sniffer implementovaný v jazyku C. Programu je možné špecifikovať dané rozhranie, port, vybraný typ packetov a počet packetov, ktoré má program vyhladať a ich informácie vypísať na štandarný výstup. V prípade nastania vnútornej chyby je vypísaná chybová hláška a program ukonečený. Beh programu je tiež možné ukončiť SIGINT signálom, ktorý preruší beh programu.

# 2 Základ problematiky

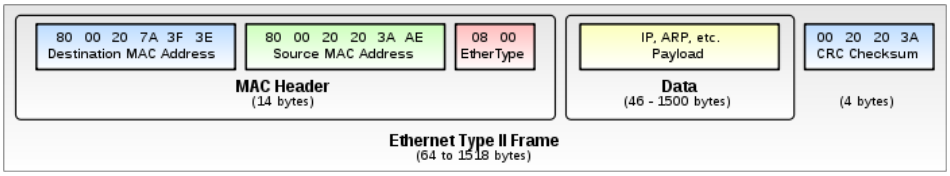
V našom projekte bolo možné špecifikovať nasledujúce typy packetov, ktoré mohol program odchytávať:

- ARP packet (sieťová vrstva)
- IMCP packet (sieťová vrstva)
- TCP packet (transportná vrstva)
- UDP packet (transportná vrstva)

Pričom program podporuje IPv4 aj IPv6 protokol.

Zakladný koncept projektu pozostáva z odchytávania jednotlivých packetov prichádzajúcich na dané rozhranie, ich následnej fitrácie podľa špecifikácie typov packetov na odchytávanie a neposledne získavania enkapsulovaných dát z odchyteného packetu a vypísania jeho obsahu na štandardný výstup.

Odchytený datagram získavame ako frame vrstvy sieťového rozhrania. Môžeme ho vidieť na nasledujúcom obrázku.

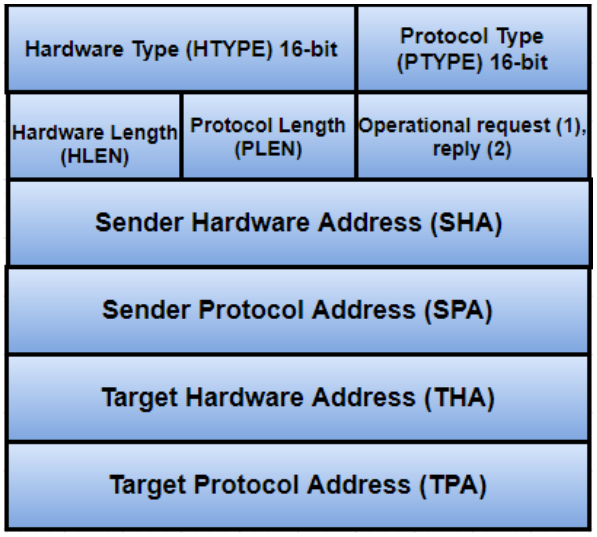



Z daného framu je potrebné oddelenie jeho hlavičky, z ktorej sa vyčíta MAC adresa zdrojového aj cieľového zariadenia. Posledná získaná informácia z ethernetovej hlavičky je ethernetový typ, podľa ktorého dokážeme rozlíšiť, aký typ packetu sa nachádza v payloade daného frame-u.

Po rozbalení daného payloadu môžeme získať nasledujúce packety:

## 2.1 ARP packet

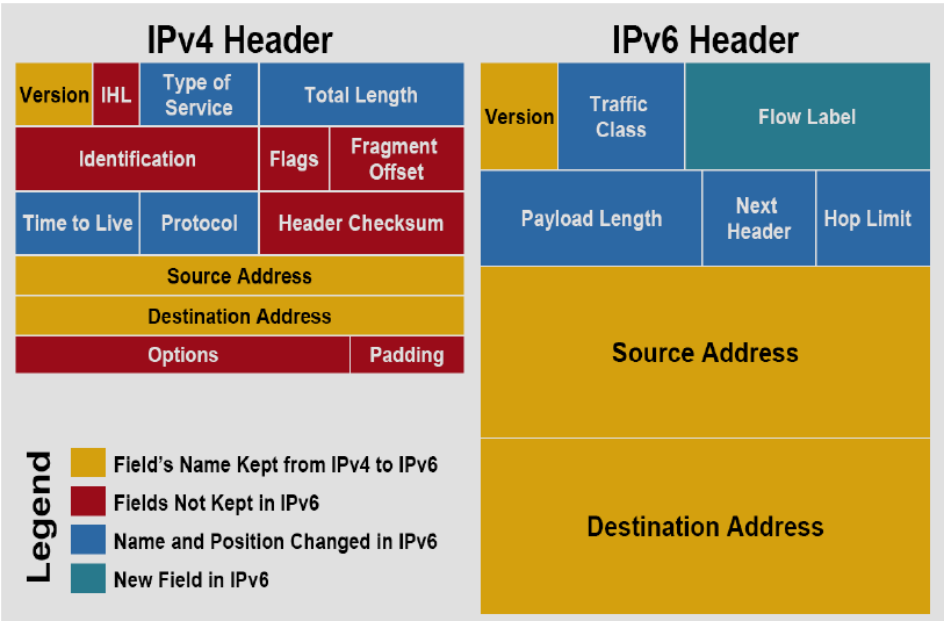
Štruktúra ARP packetu vyzerá nasledovne:




Tieto druhy packetov sa požívajú napr. pri identifikácií pripojených zariadení v lokálnej sieti, kedy po zadaní IP adresy sa zariadenie snaží túto adresu pingnúť čakajúc, či niektoré zo zariadení odpovie. Z daného packetu teda vieme zistiť IP adresy cieľového aj zdrojového zariadenia.

## 2.2 IP packet

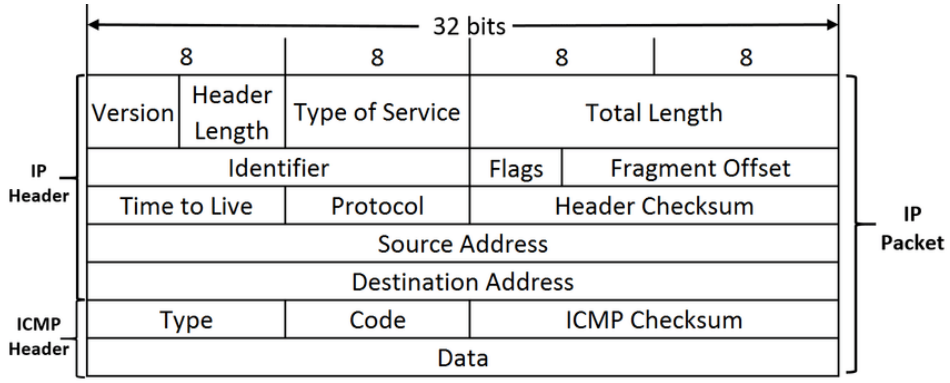
Pri IP packetoch môžeme naraziť na ich IPv4 alebo IPv6 verziu, ktorých hlavičky sa líšia.




IPv4 packet má premennú veľkosť packetu, zatial čo IPv6 packet ju ma presne danú. Z daných hlavičiek si vieme skontrolovať správnosť verzie, získať dané IP adresy zdrojového aj cieľového zariadenia aj typ IP packetu podľa daného protokolu(next header pri IPv6). Pri IPv4 packete je potrebné zistiť aj veľkosť danej hlavičky z IHL(počet dvojslov), aby sme vedeli povedať, kde začína payload. Podľa protokolu IP packetu možeme rozlíšiť ICMP, UDP a TCP datagramy.

### 2.2.1 ICMP packet

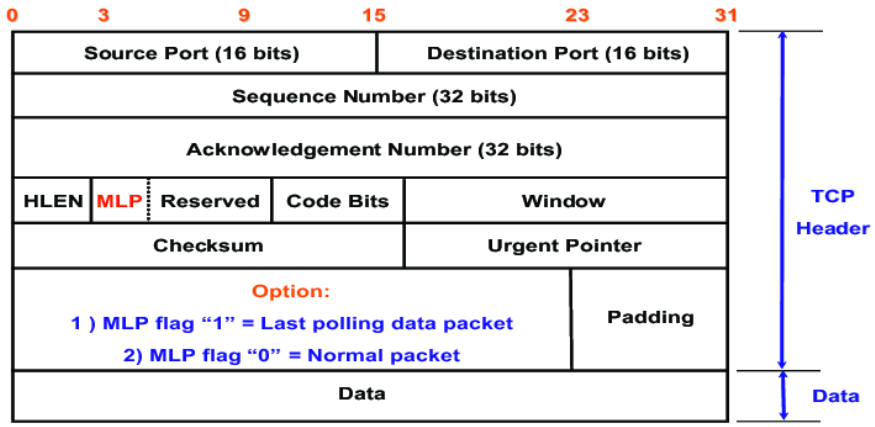
Ako môžeme vidieť z obrázku, ICMP packet obsahuje dalšiu hlavičku hneď za IP hlavičkou.




Tieto packety majú jednoduchú štruktúru a sú využívané napr. na pingovanie.

### 2.2.2 TCP packet

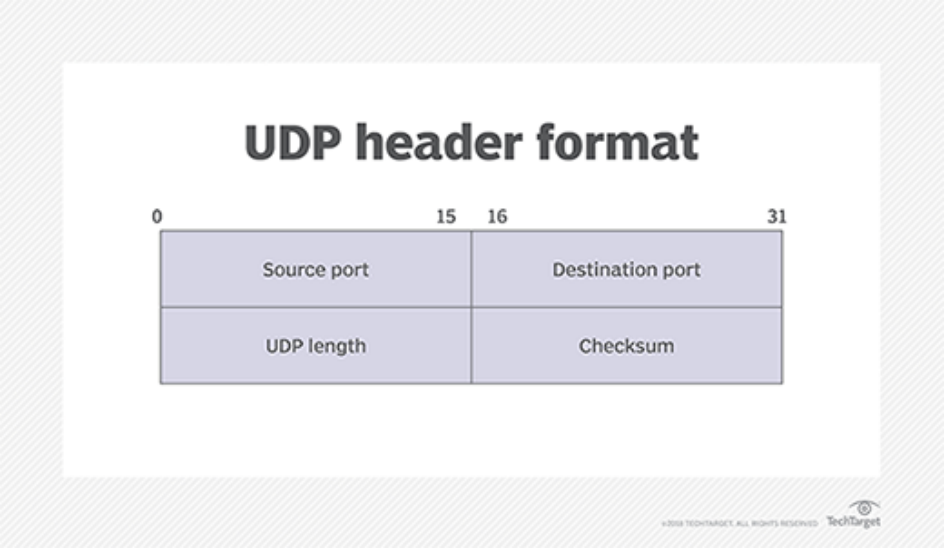
Podsledná informácia, ktorú z datagramov potrebujeme získať, sú dané porty, na ktorých prebieha komunikácia.




Tak tomu je pri TCP datagramoch, ktoré dokážu prenášať správy z jedného zariadenia na druhé. Ich hlavičky nemajú pevnú dĺžku a sú značne obsiahle.

### 2.2.3 UDP packet

Druhý spôsob posielania správ je pomocou UDP datagramov. Tie sú značne jednoduchšie, keďže ponúkajú menej spoľahlivú komunikáciu medzi zariadeniami.




Obsahujú zdrojový a cieľový port, na ktorých prebieha daná komunikácia.

# 3 Implementácia projektu

Pri implementácií projektu bolo využitých množstvo knižních obsahujúcich predovšetkým jednotlivé štruktúry pre hlavičky datagramov. Program vieme rozdeliť do niekoľkých segmentov.

## 3.1 Parsovanie argumentov

Pre prasovanie argumentov programu bola použitá štruktúra option a funkcia getopt_long() z knižnice getopt.h. Danej funkcií sú predané jednotlivé možnosti dlhých a krátkych argumentov a flag, či obsahujú aj hodnotu. Funkcia vracia daný znak podla práve načitaného argumentu spoločne s hodnotou argumentu v štruktúre optarg. V prípade neznámeho argumentu je vypísaná nápoveda užívateľovi.

## 3.2 Získanie a filtácia packetov

V prípade nezadanie rozhrania sú užívateľovi na výstup vypísané všetky aktívne rozhrania a program je ukončený. Pre tento účel je využitá funkcia pcap_findalldevs() z knižnice pcap.h, ktorá vracia viazaný zoznam nájdených rozhraní.

V prípade, že je zadané rozhranie sa môže nadviazať spojenie s daným rozhraním a začať filtrácia a získavanie packetov.
Prvotne sa pokúsi získať maska na dané rozhranie pomocou pcap_lookupnet() a nadviaže sa spojenie s daným rozhraním pomocou pcap_open_live(). Takto získame tzv. handle, pomocou ktorého sa uskutočnuje komunikácia s daným rozhraním. Jednotlivé packety sa odchytávajú pomocou funkcie pcap_loop(), ktorému sa špecifikuje počet packetov, ktorý sa má odchytiť a callback funkcia volaná na jednotlivé packety. Po ukončení práce s packetmi sa handle uzavrie pre správne ukončenie komunikácie s rozhraním.

Pre filtráciu packetov sa vytára štruktúra bpf_program, ktorá vzniká na základe filtrujúceho výrazu. Ten sa kompiluje cez funkciu pcap_compile() a aplikuje na daný handle pomocou pcap_setfilter().

### 3.3 Rozbaľovanie packetov

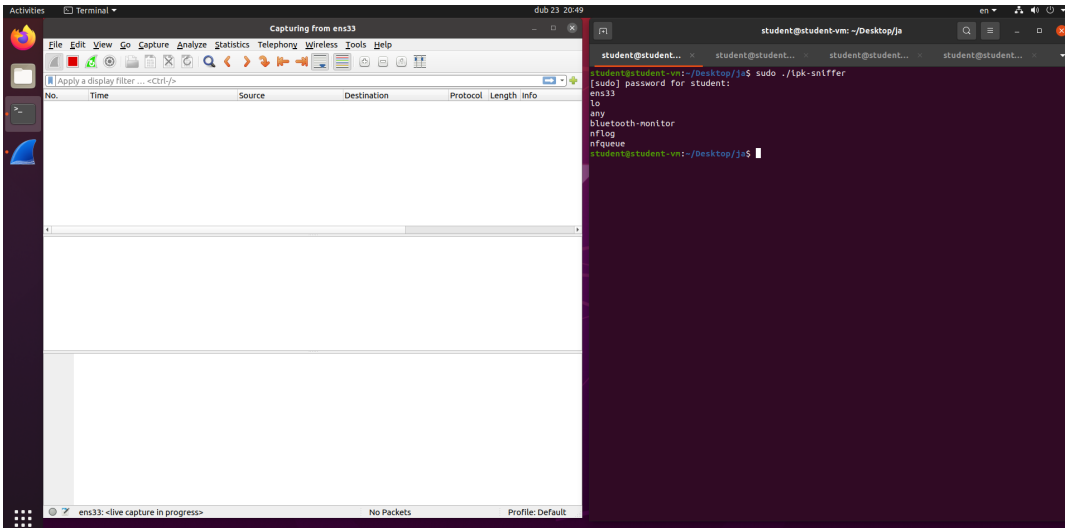
Rozbaľovanie získaného datagramu prebieha postupným odstraňovaním hlavičiek, získavaním informácií z nich a zvyšné dáta sú posielané na ďalšiu úroveň odstraňovania hlavičiek. Sú využívané štruktúry ether_header, ip6_hdr, ip, ether_arp a tcphdr.

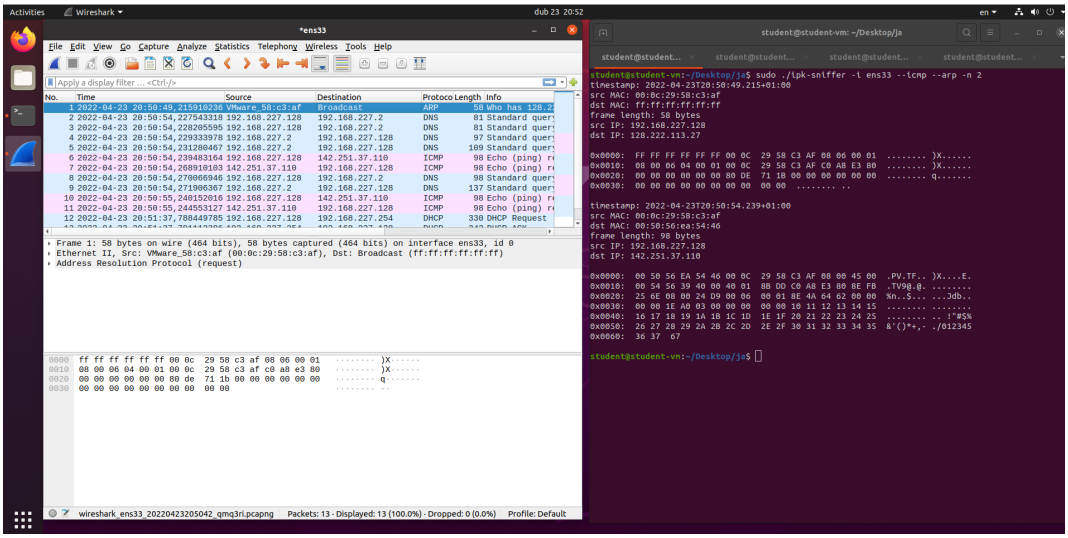
Jednotlivé adresy sú často uložené s kódovaním big endian a v prípade vypisovania je ich potreba previesť pre účely jazyka C funkciou ntohs(). Spoločne s počiatočným datagramom je tiež získaná hlavička s časom odoslania packetu a jeho veľkosťou v bytoch.

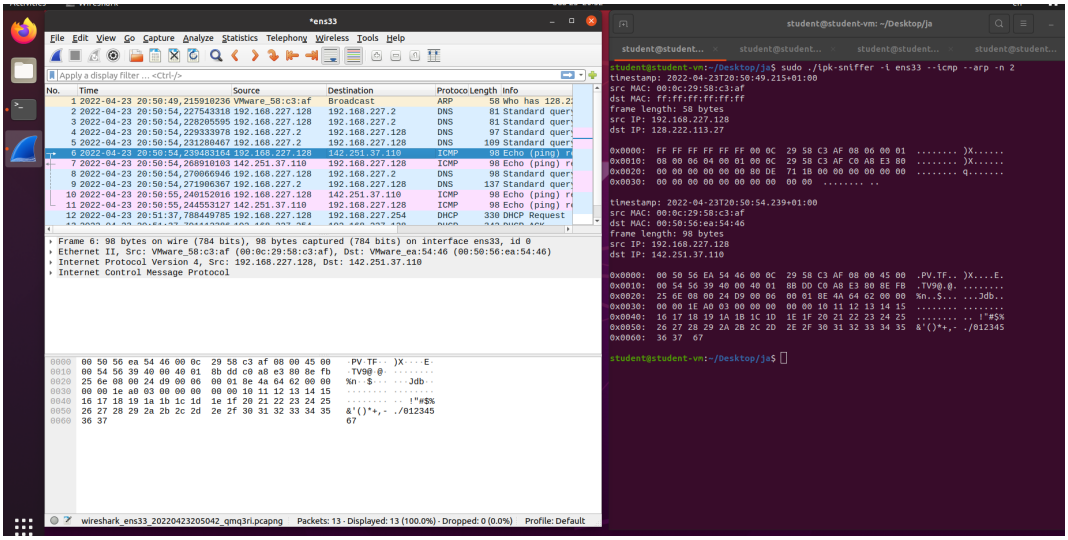
## 4 Testovanie

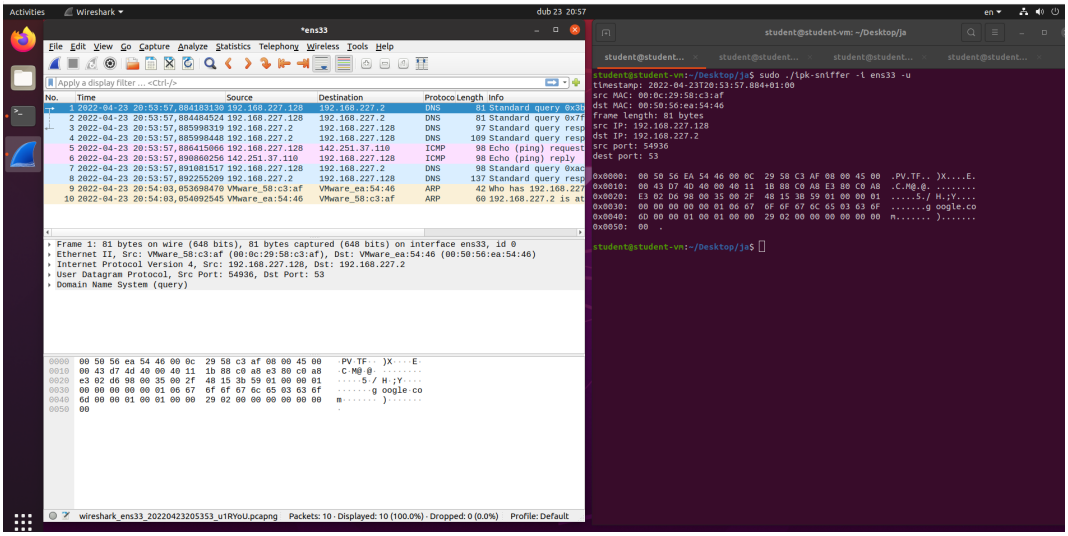
Pri testovaní sme využívali nástroj Wireshark, ktorý dokáže odchytávať jednotlivé packety na danom rozhraní a zobraziť ich obsah. Testovanie prebiehalo na referenčnom virtuálnom stroji, ktorý nám bol poskytnutý pre účely projektu. Spočívalo predovšetkým v porovnávaní informácií získaných pomocou nášho packet snifferu s informáciami z Wiresharku.

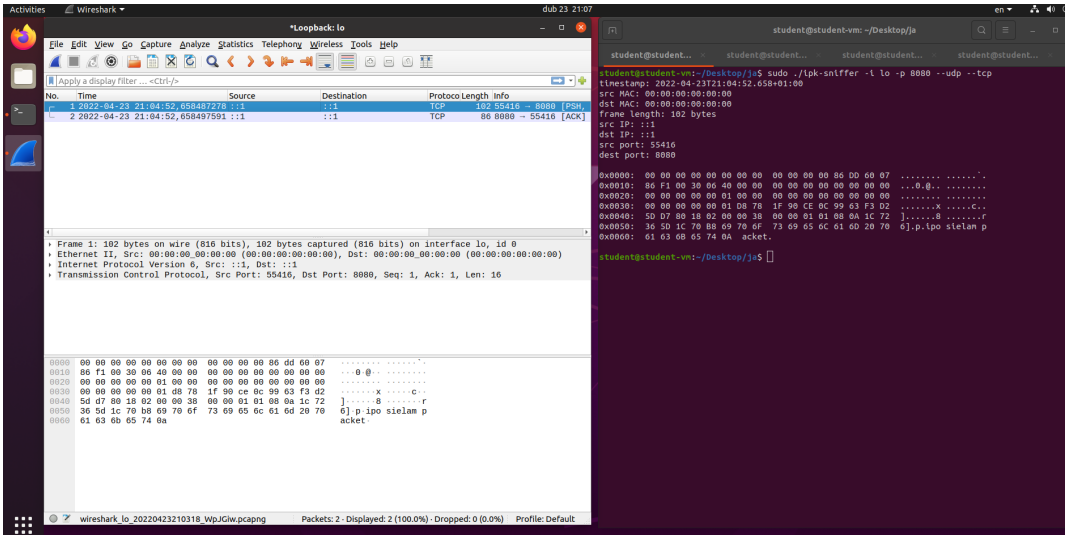
Následne je priložených pár screenshotov z testovania odchytávania packetov. Môžeme vidieť vedľa seba porovnávané informácie ako timestamp packetu, jeho velkosť v bytoch, MAC a IP adresy a ak sú prítomné tak aj jednotlivé porty. Ďalej vidíme obsah celých frame-ov a v prehľade môžeme vidieť aj úspešnú filtráciu packetov, kedy sú ignorované nesprávne typy packetov.









# 5 Použitá literatúra

- Použité stránky pri tvorení programu: https://www.tcpdump.org/pcap.html

https://www.devdungeon.com/content/using-libpcap-c
https://www.binarytides.com/category/general/

- Študentský stream od Tedro:
  https://www.youtube.com/watch?v=--tR2XqAZm0

- Riešenia problémov počas implementácie a referenčné stránky:
  https://stackoverflow.com/
  https://www.wikipedia.org/
  https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml
  https://docs.huihoo.com/doxygen/linux/kernel/3.7/structipv6hdr.html
  https://sites.uclouvain.be/SystInfo/usr/include/netinet/ip.h.html