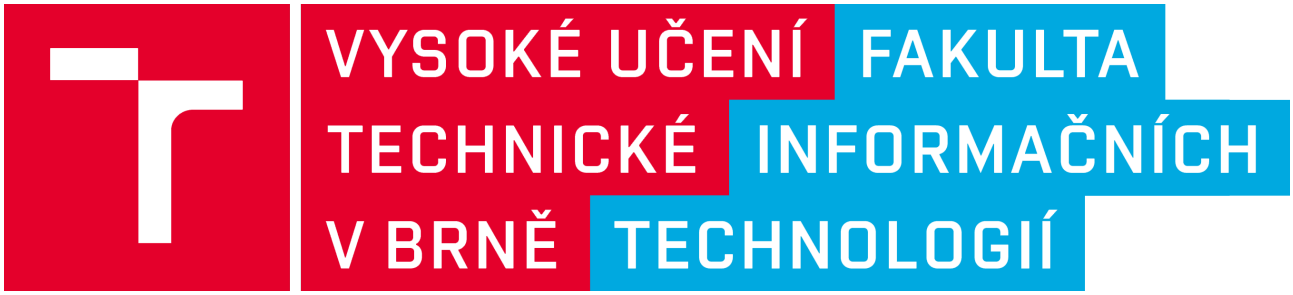


Počítačové komunikácie a siete

2. projekt - Varianta ZETA: Sniffer paketov

24. dubna 2021

Filip Brna (xbrnaf00)

Obsah

Obsah

1 Úvod

Cieľom projektu bolo implementovať Sniffer paketov, čo v preklade znamená čuchanie nad paketmi alebo teda sieťový analyzátor. Sieťový analyzátor je vhodný napríklad pre poskytovateľa internetu, správcu siete jeho účelom je zachytávanie všetkej premávky tečúcej do a z počítača pripojeného k sieti alebo môže byt nápomocný aj pri odhaľovaní nekalého podstrkávania paketov pripadne tajne odosielanie dát o ktorom užívateľ nemusí vedieť. Tento analyzátor je schopný zachytávať a filtrovať pakety nad určitým rozhraním. Program ma podporu viacerých protokolov akými sú z `IPv4 TCP, UDP, ICMP a z IPv6 ICMP`, tieto protokoly je možné vyfiltrovať podľa čísla portu, rozhrania a následne vypísať na štandardný výstup vo formáte:
čas (formát RFC3339) IP (prípadne MAC adresa) : port (zdrojový) → IP : port (cieľový), length dĺžka offset vypísaných bajtov: výpis bajtov hexa výpis bajtov ASCII.

2 Implementácia

2.1 Základné informácie

Projekt bol implementovaný v programovacom jazyku C++, pri implementácii som hlavne využil knižnicu `pcap`, ktorá implementáciu analyzátora výrazne uľahčila keďže jadrom celého analyzátora sú práve funkcie z tejto knihovne. Veľa známych programov je na tejto knižnici postavených napríklad `tcpdump, snort, wireshark` a práve `wireshark` som pri testovaní môjho analyzátora využíval. Analyzátor je v promiskuitnom móde čo znamená, že sieťová karta nám umožňuje zachytávať a sledovať aj tie pakety, ktoré nie sú určené pre ňu samotnú. Program obsahuje niekoľko globálnych premenných, ktoré sú potrebne hlavne pri priraďovaní argumentov.

2.2 Main

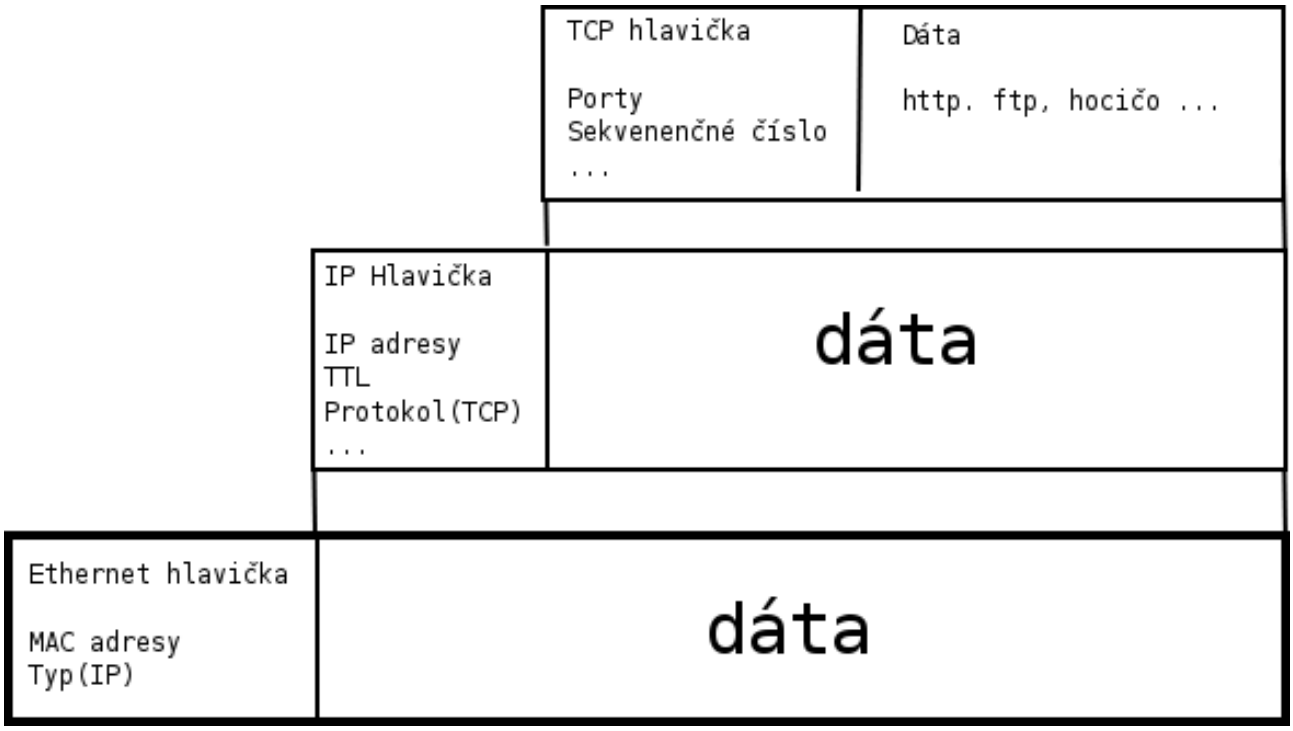
Funkcia main na úvod vola funkciu `ProcessArgs` na spracovanie argumentov, nasleduje nastavenie filtra pomocou funkcie `SetFilter` a samotne volania funkcii z pcap knižnice, ktoré sú potrebne pre sieťový analyzátor. Konkrétne sa jedna o funkcie ako:

`pcap_lookupdev` - nastaví nám sieťové rozhranie
`pcap_lookupnet` - zisti masku a IP adresu
`pcap_open_live` - otvorí sniffing session v promiskuidnom mode
`pcap_datalink` - skontroluje ci sme na ethernete
`pcap_compile, pcap_setfilter` - skompiluje a nastaví filter
`pcap_loop` - funkcia na zachytávanie paketov
`pcap_close` - uvoľnenie zdrojov

2.3 print_packet

Funkcia v ktorej prebieha hlavná práca s paketom a protokolom. Jej účelom je rozdelenie podľa paketov, protokolov a následne výpis paketu v správnom formáte. Na začiatku sú deklarované všetky potrebne štruktúry, ktoré uľahčujú neskoršiu prácu so samotným paketom ci už ide o analýzu ethernetovej hlavičky alebo hlavičky jednotlivých protokolov. Nasleduje inicializovanie

premennej time pomocou funkcie `now_rfc3999`, ktorá vracia aktuálny UTC čas v potrebnom formáte RFC3339, čas je typu reťazec. Ukážka co vštko v sebe obsahuje paket, možete vidieť na nasledujúcom obrázku.



Neskor je potrebne zistiť ci sa jedna paket typu IPv6, IPv4 alebo ARP protokol, to sa dá zistiť podľa ethernetového typu(vid obrazok), ktorý je nám známy z hlavičkového súboru `ethernet.h`. Keď už poznám akého typu je paket tak stačilo iba zistiť typ protokolu na to som využil štruktúru `ip` a jej člen `ip_p` z hlavicového súboru `ip.h`. Keď je známy typ paketu a aj protokol ostáva už iba samotný výpis dať v nasledovnom poradí:
čas zdrojová IP (prípadne MAC adresa) zistená z pomocnej štruktúry príslušného protokolu (`tcphdr, udphdr, icmphdr, arphdr`) : port(iba v prípade TCP, UDP protokolov) → cieľová IP : port (obdobne ako u zdroja), length dĺžka offset vypísaných bajtov: výpis bajtov hexa výpis bajtov ASCII.
IP - pri IPv4 je vypísaná pomocou funkcie `inet_ntoa` (konvertuje Internetové číslo v IN do ASCII) z `inet.h`, pri IPv6 je to `inet6_ntoa`.
Port - vypísaný pomocou funkcie `inet_ntohs` (anglicky popis - "Convert a 16-bit value from network-byte order to host-byte order") z `in.h`.
MAC adresa - vypísaná pri ARP protokole pomocou `ether_ntoa` z `ether.h`.
Hexa výpis bajtov - potrebne zavolať funkciu `hexDump`, ktorá vypíše dáta zo sniffovaneho paketu na štandardný výstup.

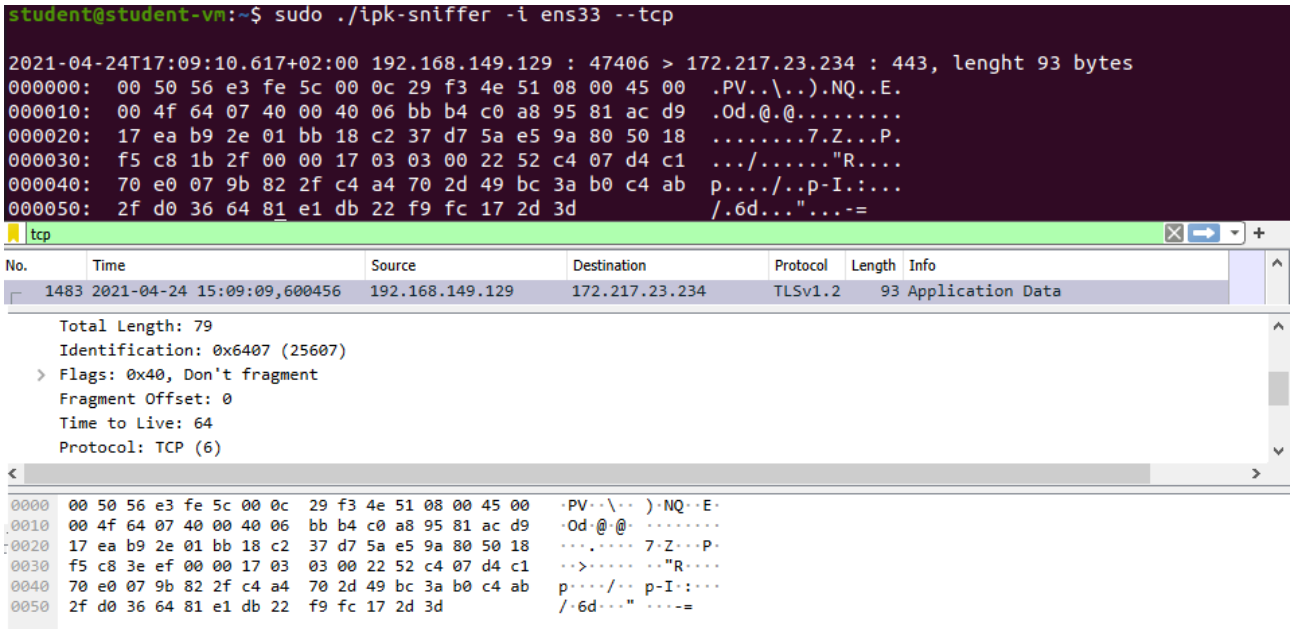
3 Príklady spustenia

sudo ./ipk-sniffer -i eth0 -p 23 –tcp -n 2
sudo ./ipk-sniffer -i eth0 –udp
sudo ./ipk-sniffer -i eth0 -n 10
sudo ./ipk-sniffer -i eth0 -p 22 –tcp –udp –icmp –arp .... je rovnaké ako:
sudo ./ipk-sniffer -i eth0 -p 22
sudo ./ipk-sniffer -i eth0

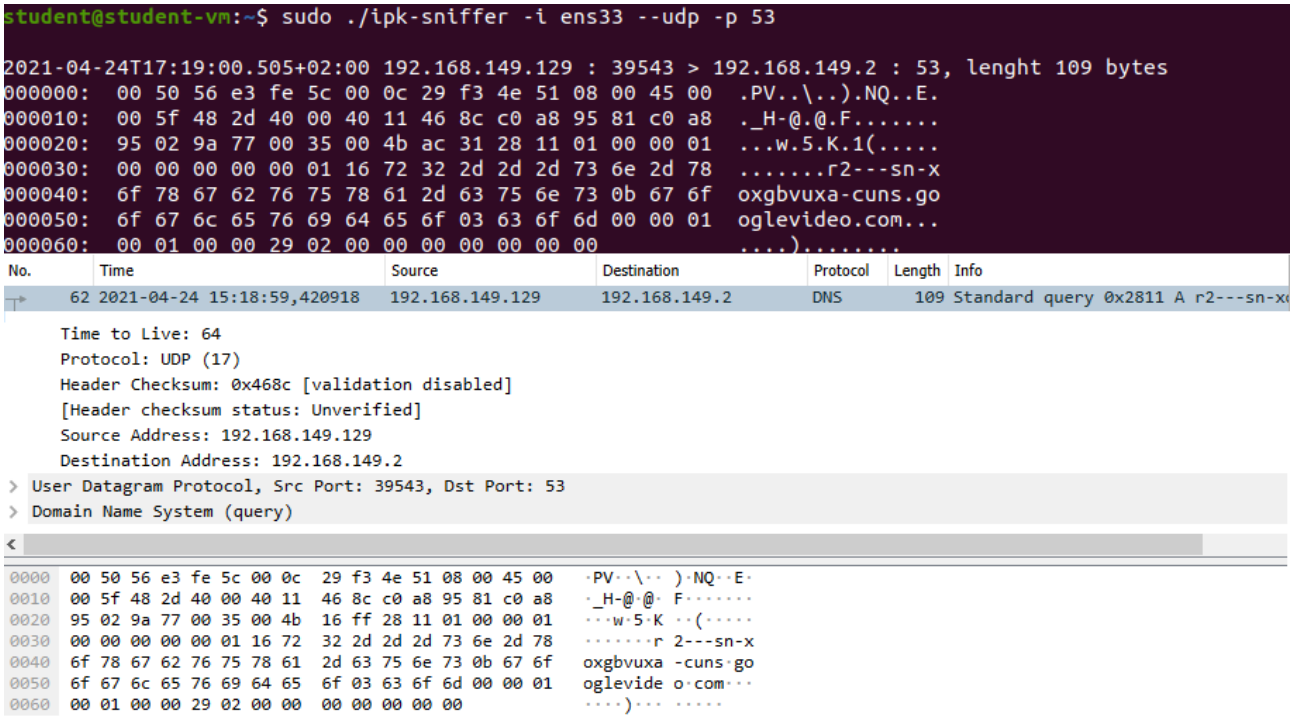
4 Testovanie

Snímky obrazoviek boli robené tak aby boli zachytené všetky potrebné informácie či už samotný pogram s parametrami a jeho výstupom alebo dáta pre kontrolu z programu Wireshark. Na testovanie som používal ako referenčný virtualny stroj tak aj lokálne WSL na mojom počítači.

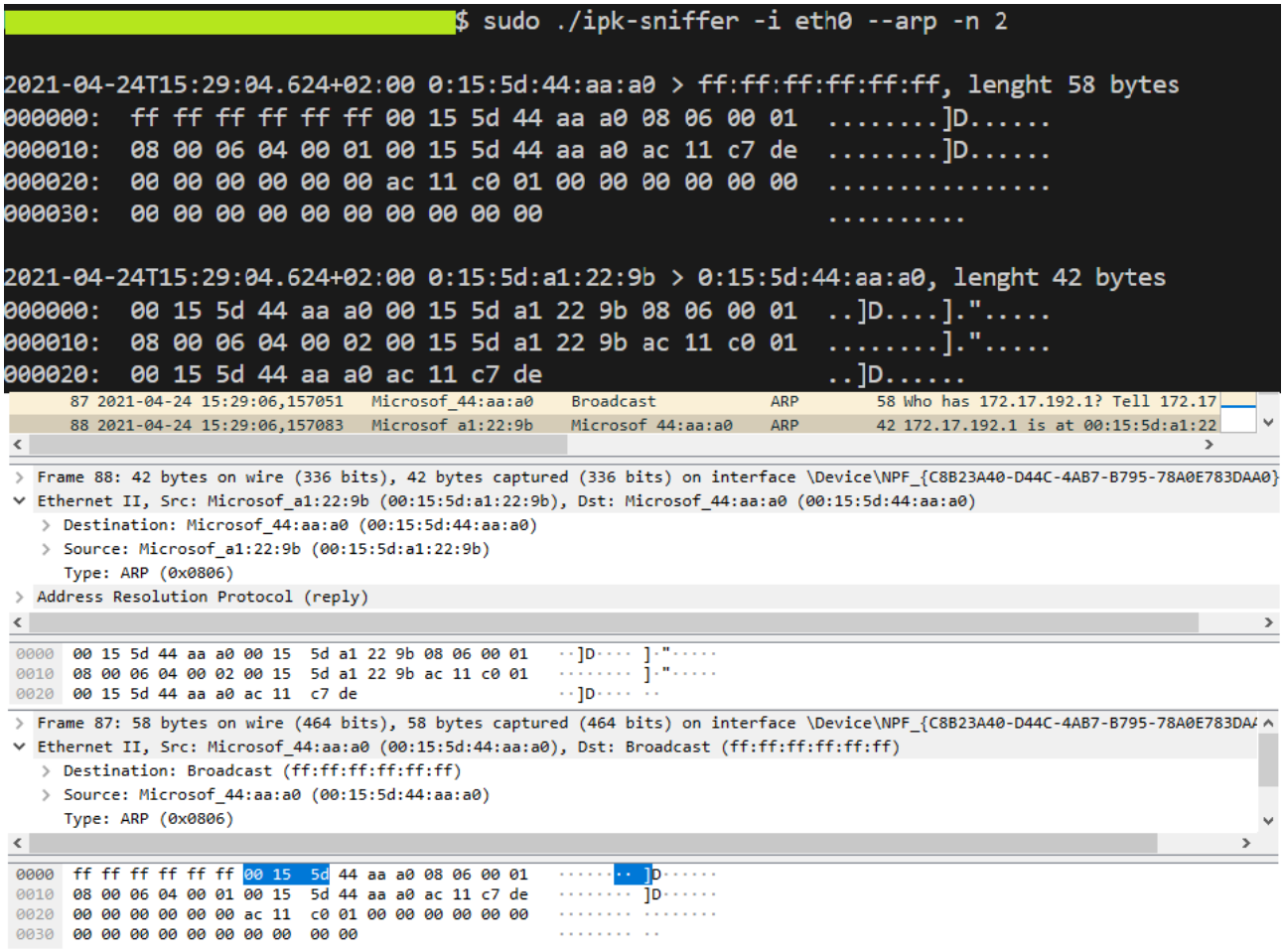
4.1 TCP



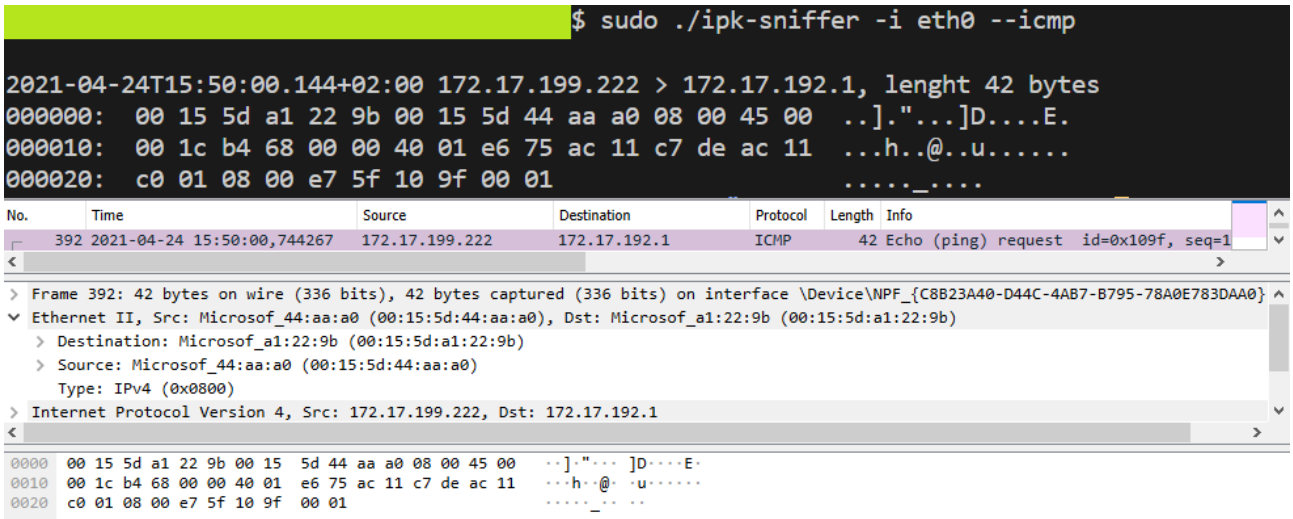
4.2 UDP so zadaným argumentom -p



4.3 ARP so zadaným argumentom -n



4.4 ICMP IPv4



4.5 ICMP IPv6

```
$ sudo ./ipk-sniffer -i eth0 --icmp

2021-04-24T15:37:00.784+02:00 fe80::215:5dff:fe44:aaa0 > ff02::1:ffe5:8f41, lenght 86 bytes
0x0000:  33 33 ff e5 8f 41 00 15 5d 44 aa a0 86 dd 60 00  33...A..]D....`.
0x0010:  00 00 00 20 3a ff fe 80 00 00 00 00 00 00 02 15  ... :...........
0x0020:  5d ff fe 44 aa a0 ff 02 00 00 00 00 00 00 00 00  ]..D............
0x0030:  00 01 ff e5 8f 41 87 00 a0 5f 00 00 00 00 fe 80  .....A..._......
0x0040:  00 00 00 00 00 00 9a fc 11 ff fe e5 8f 41 01 01  .............A..
0x0050:  00 15 5d 44 aa a0                               ..]D..
```

```
187 2021-04-24 15:37:01,459663   fe80::215:5dff:fe44… ff02::1:ffe5:8f41   ICMPv6   86 Neighbor Solicitation for fe80::9afc:
Frame 187: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface \Device\NPF_{C8B23A40-D44C-4AB7-B795-78A0E783DAA0}
Ethernet II, Src: Microsof_44:aa:a0 (00:15:5d:44:aa:a0), Dst: IPv6mcast_ff:e5:8f:41 (33:33:ff:e5:8f:41)
  Destination: IPv6mcast_ff:e5:8f:41 (33:33:ff:e5:8f:41)
  Source: Microsof_44:aa:a0 (00:15:5d:44:aa:a0)
  Type: IPv6 (0x86dd)
```

```
0000  33 33 ff e5 8f 41 00 15  5d 44 aa a0 86 dd 60 00   33···A·· ]D····`·
0010  00 00 00 20 3a ff fe 80  00 00 00 00 00 00 02 15   ··· :··· ········
0020  5d ff fe 44 aa a0 ff 02  00 00 00 00 00 00 00 00   ]··D···· ········
0030  00 01 ff e5 8f 41 87 00  a0 5f 00 00 00 00 fe 80   ·····A·· ·_······
0040  00 00 00 00 00 00 9a fc  11 ff fe e5 8f 41 01 01   ········ ·····A··
0050  00 15 5d 44 aa a0                                  ··]D··
```

4.6 Interface, help

```
student@student-vm:~$ sudo ./ipk-sniffer -i
1: ens33
2: lo
3: any
4: bluetooth-monitor
5: nflog
student@student-vm:~$ sudo ./ipk-sniffer --help
-i <rozhranie>:           (povinny argument)
                          rozhranie je volitelny parameter,
                          ak nebude definovane vypise zoznam aktivnych rozhrani
-p <cislo portu>:         nastav cislo portu
-t alebo --tcp:           zobrazuje iba TCP pakety
-u alebo --udp:           zobrazuje iba UDP pakety
--icmp:                   zobrazuje iba ICMPv4 a ICMPv6 pakety
--arp:                    zobrazuje iba ARP pakety
-------------------------------------------------------------
ak nie je konkretny protokol specifikovany, vypisuju sa vsetky
-n <cislo>:               cislo - pocet rozhrani, ktore sa zobrazia, ak nie je argument zadany tak sa zobrazi jeden
```

*V poslednom riadku pri výpise nápovedy je chyba, namiesto "počet rozhraní"ma byt "počet paketov"(v kóde opravené).

5 Zdroje

Informácie ohľadom toho ako sieťový analyzátor pracuje som získaval z nasledujúcich zdrojov, taktiež sa tu nachádzajú aj zdroje odkiaľ som čerpal niektoré zdrojové kódy. Základne informácie a ynalosti ohľadom pcap knižnice a sieťovom analyzátore som nadobúdal na týchto stránkach:
— autor : Oliver Kindernay
— link: https://linuxos.sk/clanok/packet-capturing-s-libpcap-1/
— link: https://linuxos.sk/clanok/packet-capturing-s-libpcap-2/
Zdroj funkcie `inet6_ntoa` :
— link: https://dox.ipxe.org/ipv6_8c.html#a54a82d98c20b9b1ccc276df64cf36971
Zdroj funkcie `now_rfc3339` :
— otazka : C++ RFC3339 timestamp with milliseconds using std::chrono
— odpoved, autor : sebastian, datum - Jan 23 '19 at 10:30
— link: https://stackoverflow.com/questions/54325137/c-rfc3339-timestamp-with-milliseconds
Zdroj funkcie `hexDump` :
— otazka : how to get hexdump of a structure data
— odpoved, autor : paxdiablo, datum - Oct 15 '11 at 6:08
— link: https://stackoverflow.com/questions/7775991/how-to-get-hexdump-of-a-structure-data

6 Záver

Projekt hodnotím pozitívne, pretože som si rozšíril informácie ohľadom internetových protokolov. Ďalej som sa naučil pracovať s hlavičkami jednotlivých paketov a odlíšiť ich od iných dát z paketu. Keďže som pri projekte aktívne využíval open-source program Wireshark na testovanie môjho analyzátora, naučil som sa pracovať aj s nim. Všetky nadobudnuté znalosti pre prácu s protokolmi, paketmi, Wiresharkom, pcap knižnicou vnímam do budúcnosti ako veľké plus ci už v profesionálnom alebo súkromnom živote.