



Počítačové komunikácie a siete  
**2. projekt - Varianta ZETA: Sniffer paketov**

# Obsah

## Obsah

<b>1</b>	<b>Úvod</b>	<b>2</b>
<b>2</b>	<b>Implementácia</b>	<b>2</b>
2.1	Základné informácie . . . . .	2
2.2	Main . . . . .	2
2.3	print_packet . . . . .	2
<b>3</b>	<b>Príklady spustenia</b>	<b>4</b>
<b>4</b>	<b>Testovanie</b>	<b>5</b>
4.1	TCP . . . . .	5
4.2	UDP so zadaným argumentom -p . . . . .	5
4.3	ARP so zadaným argumentom -n . . . . .	6
4.4	ICMP IPv4 . . . . .	6
4.5	ICMP IPv6 . . . . .	7
4.6	Interface, help . . . . .	7
<b>5</b>	<b>Zdroje</b>	<b>8</b>
<b>6</b>	<b>Záver</b>	<b>8</b>

# 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 byť nápomocný aj pri odhaľovaní nekalého podstrkávania paketov prípadne 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 analyzátoru výrazne uľahčila keďže jadrom celého analyzátoru 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 analyzátoru 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 potrebné hlavne pri priradovaní argumentov.

### 2.2 Main

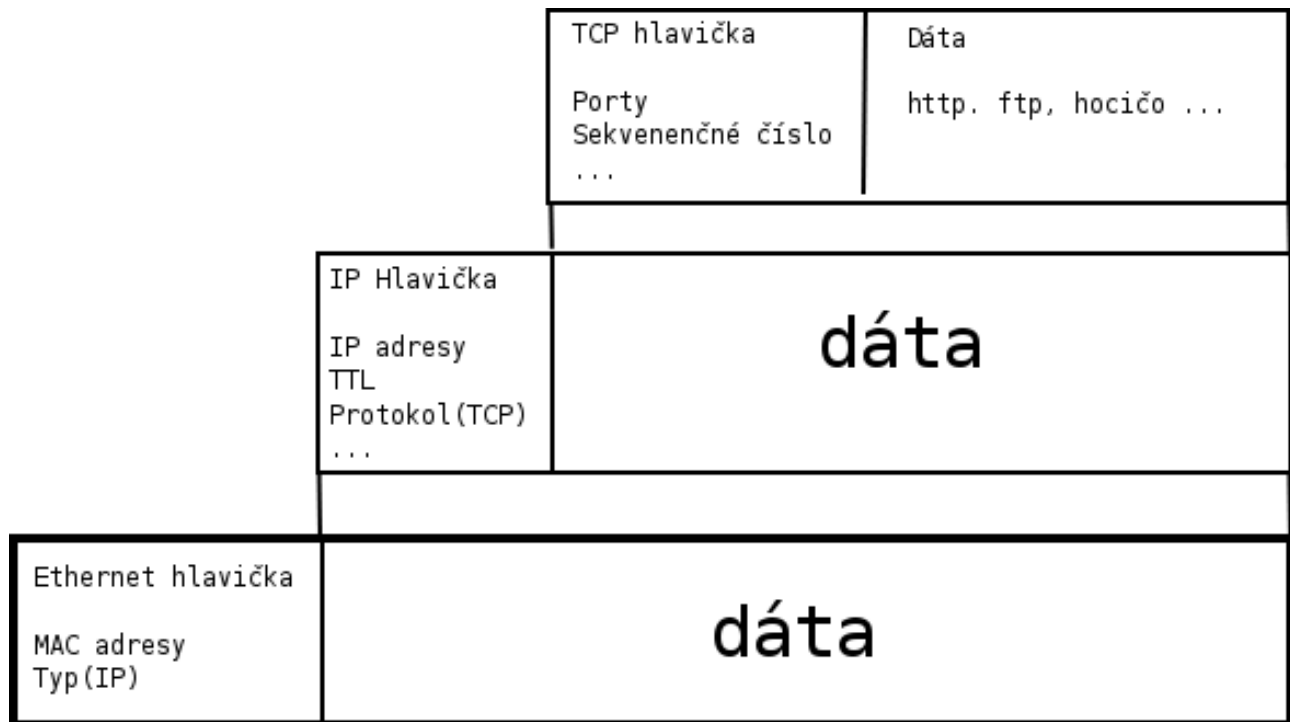
Funkcia `main` na úvod volá funkciu `ProcessArgs` na spracovanie argumentov, nasleduje nastavenie filtra pomocou funkcie `SetFilter` a samotne volania funkcií z `pcap` knižnice, ktoré sú potrebné pre sieťový analyzátor. Konkrétne sa jedna o funkcie ako:

`pcap_lookupdev` - nastaví nám sieťové rozhranie  
`pcap_lookupnet` - zistí masku a IP adresu  
`pcap_open_live` - otvorí sniffing session v promiskuidnom mode  
`pcap_datalink` - skontroluje či sme na ethernetete  
`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 potrebné štruktúry, ktoré uľahčujú neskoršiu prácu so samotným paketom či 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 čo všetko v sebe obsahuje paket, môžete vidieť na nasledujúcom obrázku.



Neskor je potrebné zistiť či sa jedná o paket typu IPv6, IPv4 alebo ARP protokol, to sa dá zistiť podľa ethernetového typu (vid obrázok), ktorý je nám známy z hlavičkového súboru `ethernet.h`. Keď už poznám akého typu je paket tak stačí iba zistiť typ protokolu na to som využil štruktúru `ip` a jej člen `ip_p` z hlavičkového súboru `ip.h`. Keď je známy typ paketu a aj protokol ostáva už iba samotný výpis dát 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 - potrebné zavolať funkciu `hexDump`, ktorá vypíše dáta zo sniffovaného 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 program 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

```
student@student-vm:~$ sudo ./ipk-sniffer -i ens33 --tcp

2021-04-24T17:09:10.617+02:00 192.168.149.129 : 47406 > 172.217.23.234 : 443, lenght 93 bytes
000000: 00 50 56 e3 fe 5c 00 0c 29 f3 4e 51 08 00 45 00 .PV..\..).NQ..E.
000010: 00 4f 64 07 40 00 40 06 bb b4 c0 a8 95 81 ac d9 .Od.@.@.....
000020: 17 ea b9 2e 01 bb 18 c2 37 d7 5a e5 9a 80 50 18 .....7.Z...P.
000030: f5 c8 1b 2f 00 00 17 03 03 00 22 52 c4 07 d4 c1 .../....."R....
000040: 70 e0 07 9b 82 2f c4 a4 70 2d 49 bc 3a b0 c4 ab p..../.p-I:....
000050: 2f d0 36 64 81 e1 db 22 f9 fc 17 2d 3d /.6d..."...=

tcp
No. Time Source Destination Protocol Length Info
1483 2021-04-24 15:09:09,600456 192.168.149.129 172.217.23.234 TLSv1.2 93 Application Data
Total Length: 79
Identification: 0x6407 (25607)
Flags: 0x40, Don't fragment
Fragment Offset: 0
Time to Live: 64
Protocol: TCP (6)
0000 00 50 56 e3 fe 5c 00 0c 29 f3 4e 51 08 00 45 00 .PV..\..).NQ..E.
0010 00 4f 64 07 40 00 40 06 bb b4 c0 a8 95 81 ac d9 .Od.@.@.....
0020 17 ea b9 2e 01 bb 18 c2 37 d7 5a e5 9a 80 50 18 .....7.Z...P.
0030 f5 c8 1b 2f 00 00 17 03 03 00 22 52 c4 07 d4 c1 .../....."R....
0040 70 e0 07 9b 82 2f c4 a4 70 2d 49 bc 3a b0 c4 ab p..../.p-I:....
0050 2f d0 36 64 81 e1 db 22 f9 fc 17 2d 3d /.6d..."...=
```

### 4.2 UDP so zadaným argumentom -p

```
student@student-vm:~$ sudo ./ipk-sniffer -i ens33 --udp -p 53

2021-04-24T17:19:00.505+02:00 192.168.149.129 : 39543 > 192.168.149.2 : 53, lenght 109 bytes
000000: 00 50 56 e3 fe 5c 00 0c 29 f3 4e 51 08 00 45 00 .PV..\..).NQ..E.
000010: 00 5f 48 2d 40 00 40 11 46 8c c0 a8 95 81 c0 a8 ._H-@.@.F.....
000020: 95 02 9a 77 00 35 00 4b ac 31 28 11 01 00 00 01 ...w.5.K.1(....
000030: 00 00 00 00 00 01 16 72 32 2d 2d 2d 73 6e 2d 78 .....r2---sn-x
000040: 6f 78 67 62 76 75 78 61 2d 63 75 6e 73 0b 67 6f oxgbvuxa-cuns.go
000050: 6f 67 6c 65 76 69 64 65 6f 03 63 6f 6d 00 00 01 oglevideo.com...
000060: 00 01 00 00 29 02 00 00 00 00 00 00 00 00 00 ....).....

No. Time Source Destination Protocol Length Info
62 2021-04-24 15:18:59,420918 192.168.149.129 192.168.149.2 DNS 109 Standard query 0x2811 A r2---sn-x
Time to Live: 64
Protocol: UDP (17)
Header Checksum: 0x468c [validation disabled]
[Header checksum status: Unverified]
Source Address: 192.168.149.129
Destination Address: 192.168.149.2
> User Datagram Protocol, Src Port: 39543, Dst Port: 53
> Domain Name System (query)
0000 00 50 56 e3 fe 5c 00 0c 29 f3 4e 51 08 00 45 00 .PV..\..).NQ..E.
0010 00 5f 48 2d 40 00 40 11 46 8c c0 a8 95 81 c0 a8 ._H-@.@.F.....
0020 95 02 9a 77 00 35 00 4b 16 ff 28 11 01 00 00 01 ...w.5.K.1(....
0030 00 00 00 00 00 01 16 72 32 2d 2d 2d 73 6e 2d 78 .....r2---sn-x
0040 6f 78 67 62 76 75 78 61 2d 63 75 6e 73 0b 67 6f oxgbvuxa-cuns.go
0050 6f 67 6c 65 76 69 64 65 6f 03 63 6f 6d 00 00 01 oglevideo.com...
0060 00 01 00 00 29 02 00 00 00 00 00 00 00 00 00 ....).....
```

### 4.3 ARP so zadaným argumentom -n

```
$ sudo ./ipk-sniffer -i eth0 --arp -n 2

2021-04-24T15:29:04.624+02:00 0:15:5d:44:aa:a0 > ff:ff:ff:ff:ff:ff, lenght 58 bytes
000000:  ff ff ff ff ff ff 00 15 5d 44 aa a0 08 06 00 01  ....]D.....
000010:  08 00 06 04 00 01 00 15 5d 44 aa a0 ac 11 c7 de  ....]D.....
000020:  00 00 00 00 00 00 ac 11 c0 01 00 00 00 00 00 00  ....
000030:  00 00 00 00 00 00 00 00 00 00  ....

2021-04-24T15:29:04.624+02:00 0:15:5d:a1:22:9b > 0:15:5d:44:aa:a0, lenght 42 bytes
000000:  00 15 5d 44 aa a0 00 15 5d a1 22 9b 08 06 00 01  ..]D....].".
000010:  08 00 06 04 00 02 00 15 5d a1 22 9b ac 11 c0 01  ....].".
000020:  00 15 5d 44 aa a0 ac 11 c7 de  ..]D.....

87 2021-04-24 15:29:06,157051  Microsof_44:aa:a0  Broadcast  ARP  58 Who has 172.17.192.1? Tell 172.17.
88 2021-04-24 15:29:06,157083  Microsof_a1:22:9b  Microsof_44:aa:a0  ARP  42 172.17.192.1 is at 00:15:5d:a1:22:
< >

> Frame 88: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{C8B23A40-D44C-4AB7-B795-78A0E783DAA0}
v Ethernet II, Src: Microsof_a1:22:9b (00:15:5d:a1:22:9b), Dst: Microsof_44:aa:a0 (00:15:5d:44:aa:a0)
  > Destination: Microsof_44:aa:a0 (00:15:5d:44:aa:a0)
  > Source: Microsof_a1:22:9b (00:15:5d:a1:22:9b)
  Type: ARP (0x0806)
> Address Resolution Protocol (reply)
< >

0000 00 15 5d 44 aa a0 00 15 5d a1 22 9b 08 06 00 01  ..]D....].".
0010 08 00 06 04 00 02 00 15 5d a1 22 9b ac 11 c0 01  ....].".
0020 00 15 5d 44 aa a0 ac 11 c7 de  ..]D....

> Frame 87: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface \Device\NPF_{C8B23A40-D44C-4AB7-B795-78A0E783DAA0}
v Ethernet II, Src: Microsof_44:aa:a0 (00:15:5d:44:aa:a0), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  > Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  > Source: Microsof_44:aa:a0 (00:15:5d:44:aa:a0)
  Type: ARP (0x0806)
< >

0000 ff ff ff ff ff ff 00 15 5d 44 aa a0 08 06 00 01  ....]D....
0010 08 00 06 04 00 01 00 15 5d 44 aa a0 ac 11 c7 de  ....]D....
0020 00 00 00 00 00 00 ac 11 c0 01 00 00 00 00 00 00  ....
0030 00 00 00 00 00 00 00 00 00 00  ....
```

### 4.4 ICMP IPv4

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

2021-04-24T15:50:00.144+02:00 172.17.199.222 > 172.17.192.1, lenght 42 bytes
000000:  00 15 5d a1 22 9b 00 15 5d 44 aa a0 08 00 45 00  ..]."....]D....E.
000010:  00 1c b4 68 00 00 40 01 e6 75 ac 11 c7 de ac 11  ...h..@..u.....
000020:  c0 01 08 00 e7 5f 10 9f 00 01  ...._....

No.      Time                Source                Destination            Protocol  Length  Info
392 2021-04-24 15:50:00,744267  172.17.199.222      172.17.192.1          ICMP      42      Echo (ping) request id=0x109f, seq=1
< >

> Frame 392: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{C8B23A40-D44C-4AB7-B795-78A0E783DAA0}
v Ethernet II, Src: Microsof_44:aa:a0 (00:15:5d:44:aa:a0), Dst: Microsof_a1:22:9b (00:15:5d:a1:22:9b)
  > Destination: Microsof_a1:22:9b (00:15:5d:a1:22:9b)
  > Source: Microsof_44:aa:a0 (00:15:5d:44:aa:a0)
  Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 172.17.199.222, Dst: 172.17.192.1
< >

0000 00 15 5d a1 22 9b 00 15 5d 44 aa a0 08 00 45 00  ..]."....]D....E.
0010 00 1c b4 68 00 00 40 01 e6 75 ac 11 c7 de ac 11  ...h..@..u.....
0020 c0 01 08 00 e7 5f 10 9f 00 01  ...._....
```

## 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
000000: 33 33 ff e5 8f 41 00 15 5d 44 aa a0 86 dd 60 00 33...A..]D....`.
000010: 00 00 00 20 3a ff fe 80 00 00 00 00 00 02 15 ... :.....
000020: 5d ff fe 44 aa a0 ff 02 00 00 00 00 00 00 00 00 ]..D.....
000030: 00 01 ff e5 8f 41 87 00 a0 5f 00 00 00 00 fe 80 .....A..._.....
000040: 00 00 00 00 00 00 9a fc 11 ff fe e5 8f 41 01 01 .....A..
000050: 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 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>:      (povinný argument)
                    rozhranie je voliteľný parameter,
                    ak nebude definované vypíše zoznam aktívnych rozhraní
-p <číslo portu>:    nastav číslo 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 konkrétny protokol špecifikovaný, vypisujú sa všetky
-n <číslo>:          číslo - počet rozhraní, ktoré sa zobrazia, ak nie je argument zadany tak sa zobrazí jeden
```

\*V poslednom riadku pri výpise nápovery je chyba, namiesto "počet rozhraní" ma byť "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 znalosti ohľadom pcap knižnice a sieťovom analyzátoe 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](https://dox.ipxe.org/ipv6_8c.html#a54a82d98c20b9b1ccc276df64cf36971)

Zdroj funkcie `now_rfc3339` :

— otázka : 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` :

— otázka : 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 analyzátoru, naučil som sa pracovať aj s nim. Všetky nadobudnuté znalosti pre prácu s protokolmi, paketmi, Wiresharkom, pcap knižnicou vnímam do budúcnosti ako veľké plus či už v profesionálnom alebo súkromnom živote.