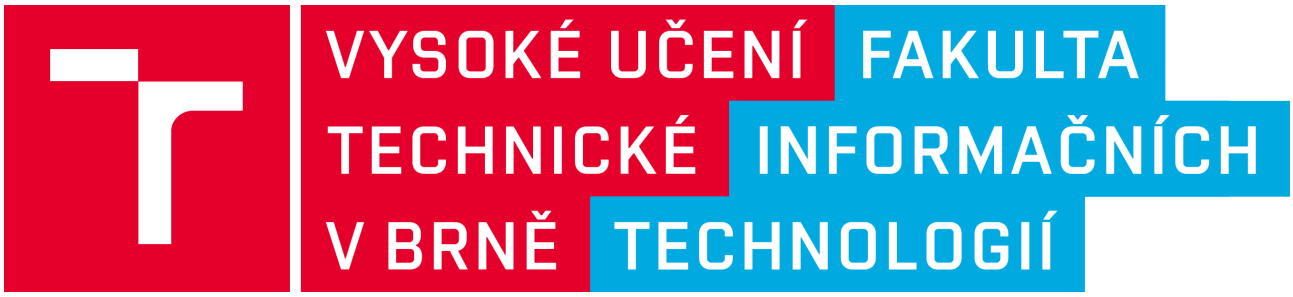
Vysoké učení technické v Brně
Fakulta informačních technologií



Počítačové komunikace a sítě
2. Projekt
Varianta [ZETA] : Sniffer paketů

Martin Bublavý (xbubla02)

10. apríla 2022

Obsah

1 Úloha

Cieľom bolo vytvoriť sieťový analyzátor, ktorý zachytáva a filtruje prichádzajúce a odchádzajúce **TCP, UDP, ICMP** pakety a **ARP** rámce na danom sieťovom rozhraní a vypisuje informácie o nich. [4]

2 Spustenie programu

Program je možné spustiť cez terminál na operačných systémoch linux. Ku kompilácií bol použitý prekladač `gcc 9.4.0`. K programu je taktiež pribalený súbor **makefile**, ktorý program zostaví pomocou príkazu `make` (testované na verzii `GNU Make 4.2.1`).

2.1 Argumenty

Program je spustitelný v tvare:

```
./ipk-sniffer [-i rozhranie | --interface rozhranie] [-p port] [--tcp|-t]
              [--udp|-u] [--arp] [--icmp] [-n num] [-h]
```

- **`-i | --interface <rozhranie>` -** Špecifikuje rozhranie, na ktorom sa dané pakety budú zachytávať. Ak nie je použitý alebo špecifikovaný, tak sa vypíšu všetky aktívne rozhrania a program končí s hodnotou **0**.
- **`-p <port>` -** Voliteľný parameter, ktorý filtruje TCP a UDP packety podľa daného portu.
- **`-t | --tcp` -** Voliteľný parameter, budú zachytávané len TCP packety (ICMP a ARP budú nadalej zachytávané).
- **`-u | --udp` -** Voliteľný parameter, budú zachytávané len UDP packety (ICMP a ARP budú nadalej zachytávané).
- **`--arp` -** Voliteľný parameter, budú zachytávané len ARP rámce (TCP a UDP budú nadalej zachytávané).
- **`--icmp` -** Voliteľný parameter, budú zachytávané len ICMPv4 a ICMPv6 packety (TCP a UDP budú nadalej zachytávané).
- **`-n <num>` -** Voliteľný parameter, špecifikuje koľko packetov má program pred skončením zachytiť. Ak nie je špecifikovaný, uvažuje sa 1 packet.
- **`-h` -** Vypíše nápovedu.

Ak nie je ani jeden filtrovací parameter špecifikovaný, zobrazujú sa všetky packety (TCP, UDP, ICMP, ARP). V prípade chybného argumentu program vypíše nápovedu a končí s návratovým kódom **21**.

3 Implementácia

Program bol implementovaný v jazyku C v súbore `./ipk-sniffer.c`.

3.1 Postup funkčnosti

3.1.1 Načítanie argumentov

Zaisťuje funkcia **arg_parse**, ktorá je volaná z funkcie **main** hneď po začatí programu. Funkcia používa knižnicu `getopt.h`, ktorá zaisťuje kontrolu a načítanie argumentov v krátkom a dlhom tvare. Informácie o použitých argumentoch a ich hodnoty ukladá do štruktúry `my_args`. Pri načítaní argumentov sa taktiež kontroluje platnosť portu (či leží v medziach) a taktiež chyby vo formáte portu a čísla v argumente -n. V prípade neočakávanej chyby pri spracovávaní argumentov vypíše nápovedu a vráti návratový kód **21**.

```
typedef struct{
    int port;         //-p <port>
    int packet_num;   //-n <num>
    bool tcp;         //tcp flag
    bool udp;         //udp flag
    bool arp;         //arp flag
    bool icmp;        //icmp flag
}my_args;
```

Obr. 1: Štruktúra my_args

3.1.2 Zachytávanie paketov

Ak spracovanie argumentov prešlo bez chyby tak funkcia **main** zavolá buď funkciu **show_active_interfaces()** (ak chýbal argument -i alebo nemal hodnotu), inak funckiu **sniff_packets(char *interface, my_args *args)**. Funkcia `show_active_interfaces()` získa zoznam všetkých aktívnych sieťových rozhraní pomocou funkcie `pcap_findalldevs(pcap_if_t **alldevsp, char *errbuf)`. Ak funkcia zlyhá, program sa ukončí s hodnotou **22** a vypíše chybovú správu. Inak vypíše zoznam rozhraní a končí s hodnotou **0**.

Funkcia `sniff_packets` najprv zistí masku podsiete a ip adresu pomocou funkcie `pcap_lookupnet`, ktorú môžeme následne priradiť zariadeniu. Ak funkcia zlyhá, program vypíše chybovú správu a končí s hodnotou **22**. Ďalej funkcia otvorí rozhranie zadané používateľom v argumente `--interface` na zachytávanie paketov pomocou funkcie `pcap_open_live` [1]. Ak funkcia zlyhá, program vypíše chybovú správu a končí s hodnotou **22**. Následne program zostaví filter paketov podľa užívatelských argumentov na vstupe. Ak neboli špecifikované žiadne argumenty, tak sa filter nastaví na základnú hodnotu, kde spracováva všetky pakety (UDP, TCP, ARP, ICMP). Tento filter je následne predaný funkcii `pcap_compile` aj so získanou ip adresou. Táto funkcia skompiluje tento adaptér , ktorý je následne aplikovaný pomocou funkcie `pcap_setfilter`. Ak niektorá z týchto funkcií zlyhá, tak program vypíše chybovú správu a končí s hodnotou **22**. Tento adaptér je následne predaný funkcii `pcap_loop`, ktorá ho uvedie do nekonečného cyklu, v ktorom sa zachytávajú pakety. Táto funkcia taktiež volá funkciu `process_packet`, ktorá spracuje každý zachytený paket. Tento cyklus sa vykonáva kým sa nespracuje taký počet paketov, aký užívateľ zadal na vstupe v argumente `-n`. Ak nebol zadaný, uvažuje sa počet 1.

3.1.3 Spracovanie paketov

Spracovávanie paketov sa vykonáva pomocou funkcie **`process_packet`**, ktorá je volaná pre každý zachytený paket. Ako prvé sa zavolá funkcia **`get_timestamp`**, ktorá zistí aktuálny čas s presnosťou na milisekundy a vráti ho vo forme stringu. Potom funkcia získa z paketu ethernetovu hlavičku pomocou štruktúry `struct ether_header`, ktorá je použitá na určenie typu packetu (IPv6, ARP, IPv4). Ak sa jedná o ARP tak volá funkciu **`process_arp`**, inak získa protokol. Ak sa jedná o protokol **TCP**, volá funkciu **`process_tcp`**. Obdobne pre **UDP** a **ICMP** volá funkcie **`process_udp`** a **`process_icmp`**. Všetky informácie o paketoch sa ukladajú do štruktúry `struct packet_info`, ktorú uvedené funkcie vracajú. Štruktúra `struct ether_header` taktiež poskytuje dĺžku rámcu a MAC adresu príjemcu a odosielateľa.

```
struct packet_info{
    int packet_type;                //(UDP, TCP, ARP, ICMP)

    char src_ip[50];                //Source IP from ip_header
    char dst_ip[50];                //Destination IP from ip_header
    u_int8_t *src_mac;              //Source MAC address from ethernet header
    u_int8_t *dst_mac;              //Destination MAC address from ethernet header
    u_int32_t frame_len;            //Frame lenght
    u_int16_t src_port;             //Source port
    u_int16_t dst_port;             //Destination port

    //ICMP
    u_int8_t icmp_type;             //ICMP type
    u_int8_t icmp_code;             //ICMP code

    //ARP
    u_int16_t arp_format_hw;        //Format of hardware address
    u_int16_t arp_format_pro;       //Format of protocol address
    u_int8_t *arphw_sender_mac;     //Sender hardware address
    u_int8_t *arphw_target_mac;     //Target hardware address
    char arp_sender_ip[16];         //Sender IP address
    char arp_target_ip[16];         //Target IP address
    unsigned char arp_len_hw;       //Lenght of hardware address
    unsigned char arp_len_pro;      //Lenght of protocol address
    uint16_t arp_opcode;            //ARP opcode
};
```

Obr. 2: Štruktúra packet_info

3.1.4 Spracovanie IP

Spracovanie IP hlavičiek [3] sa vykonáva v každej zo spomenutých funkcií [3.1.3]. Spracovanie prebieha pomocou štruktúr `struct ip` alebo `struct ip6_hdr` (ak sa jedná o IPv6). Tieto štruktúry poskytujú informácie o zdrojovej IP adrese, cieľovej IP adrese a dĺžke hlavičky (20 - 60 bajtov = IPv4). Dĺžka je v štruktúre uložená ako počet 32 bitových úsekov, preto je treba vynásobiť **4** aby sme získali veľkosť v bajtoch (`dĺžka = ip_header->ip_hl * 4`). Pre IPv6 toto zisťovať netreba keďže má konštantnú dĺžku (40 bajtov). Získané IP adresy za ukladajú v štruktúre `packet_info` [3.1.3].

```
if(ipv6){
    struct ip6_hdr *ip_header = (struct ip6_hdr*)(p_body + sizeof(struct ether_header));
    header_len = 40;
}
else{
    struct ip *ip_header = (struct ip*)(p_body + sizeof(struct ether_header));
    header_len = ip_header->ip_hl * 4;
}
```

3.1.5 Spracovanie ARP

Spracováva funckia **`process_arp`**. ARP nepoužíva IPv6, teda pre vytvorenie ARP štrukúr sa využíva dĺžka IPv4 [3.1.4].
Pre ARP sa využívajú 2 štruktúry:

- `struct arphdr` - Obsahuje formát a dĺžku hardvérovej a protokolovej adresy
- `struct ether_arp` - Obsahuje harvérové a protokolové IP adresy odosielateľa a príjemcu

Informácie z týchto štruktúr sa ukladajú do štruktúry `packet_info` [3.1.3].

```
struct ip *ip_header = (struct ip*)(p_body + sizeof(struct ether_header));
struct arphdr *arp_header = (struct arphdr*)(p_body + (ip_header->ip_hl * 4) + sizeof(struct ether_header));
struct ether_arp *etharp = (struct ether_arp*)(p_body + (ip_header->ip_hl * 4) + sizeof(struct ether_header));
```

Pre ARP sa zisťuje:

- Formát harvérovej a protokolovej adresy
- Dĺžka hardvérovej a protokolovej adresy
- Operačný kód (request, reply)
- Hardvérová adresa odosielateľa a prijemcu
- IP adresa odosielateľa a príjemcu

3.1.6 Spracovanie ICMP

Spracováva funkcia **`process_icmp`**. Spracovanie IP je popísané v sekcii 3.1.4. ICMP podobne ako ARP nevyužíva porty, teda okrem IP adries sa zisťuje len typ a kód ICMP paketu. Podľa typu a kódu môžeme zistiť riadiacu správu. [5]

Tieto informácie sa získavajú pomocou štruktúry `struct icmphdr` a ukladajú do štruktúry `packet_info` [3.1.3].

```
struct icmphdr *icmp_header = (struct icmphdr*)(p_body + header_len + sizeof(struct ether_header));
info.icmp_code = icmp_header->code;
info.icmp_type = icmp_header->type;
```

3.1.7 Spracovanie TCP

Spracováva funckia **`process_tcp`**. Spracovanie IP je popísané v sekcii 3.1.4. Na rozdiel od ARP a ICMP, TCP a UDP používa porty. Po získaní IP adries z IP hlavičky [3] sa získavajú porty zo štruktúry `[struct tcphdr]` pomocou funkcie **ntohs**. Informácie sa ukladajú do štruktúry `packet_info` [3.1.3].

```
struct tcphdr *tcp_header = (struct tcphdr*)(p_body + header_len + sizeof(struct ether_header));

info.src_port = ntohs(tcp_header->th_sport);
info.dst_port = ntohs(tcp_header->th_dport);
```

3.1.8 Spracovanie UDP

Spracováva funckia **process_udp**. Získavanie informácií prebieha obdobne ako pre TCP. Rozdiel spočíva v tom, že miesto štruktúry pre TCP sa použije štruktúra `[struct udphdr]`. Informácie sa ukladajú do štruktúry `packet_info` [3.1.3].

```
struct udphdr *udp_header = (struct udphdr*)(p_body + header_len + sizeof(struct ether_header));

info.src_port = ntohs(udp_header->uh_sport);
info.dst_port = ntohs(udp_header->uh_dport);
```

3.2 Výpis výstupu

Zabezpečené funkciami **print_info** a **print_data**.

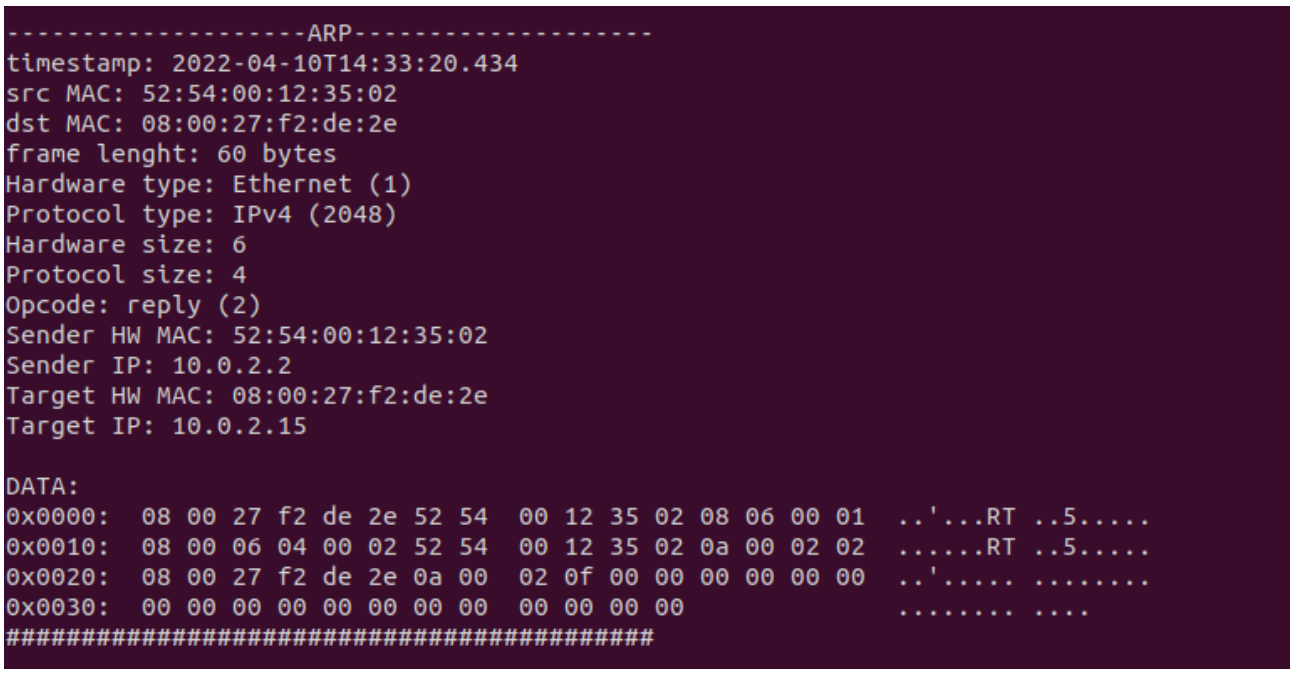
Funkcia **print_info** sa volá ako prvá a zabezpečuje výpis informácií o samotnom pakete zo štruktúry `packet_info` [3.1.3]. Výpis sa mení na základe typu paketu (TCP, UDP, ICMP, ARP), teda vypíše len informácie, ktoré sú pre daný typ platné. Funkcia taktiež využíva funkciu **print_mac_addr**, ktorá berie MAC adresu a vypíše ju podľa štandartu IEEE 802. Na prvom riadku výpisu je označenie typu paketu.

Funkcia **print_data** sa stará o výpis samotných dát paketu v hexadecimálnej a čitelnej podobe. Ak sa znak vypísať nedá, vypíše sa bodka. Výpis prebieha po riadku s pomocou viacerých vnorených `for` cyklov. Každý riadok obsahuje počet dovtedy vypísaných bajtov, 16 bajtov dát v hexadecimálnej podobe (1 dvojica = 1 bajt) a čiteľnú podobou dát na danom riadku. Výpis jednoho paketu je ukončený riadkom, ktorý obsahuje len znak '#'.

```
--------------------TCP--------------------
timestamp: 2022-04-09T22:12:52.900
src MAC: 08:00:27:f2:de:2e
dst MAC: 52:54:00:12:35:02
frame lenght: 54 bytes
src IP: 10.0.2.15
dst IP: 34.107.221.82
src port: 41836
dst port: 80

DATA:
0x0000:  52 54 00 12 35 02 08 00  27 f2 de 2e 08 00 45 00  RT..5... '.....E.
0x0010:  00 28 18 ac 40 00 40 06  16 58 0a 00 02 0f 22 6b  .(..@.@. .X...."k
0x0020:  dd 52 a3 6c 00 50 94 7f  95 8c 00 9d 3a de 50 10  .R.l.P.. ....:.P.
0x0030:  fa 14 0b e7 00 00                                 ......
##########################################
```

Obr. 3: Príklad výstupu TCP



Obr. 4: Príklad výstupu ARP

4 Použité knižnice

4.1 Esenciálne pre prácu v jazyku C

- **<stdio.h>, <stdlib.h>, <string.h>, <stdbool.h> –** Práca s reťazcami, malloc ...
- **<ctype.h> –** Použité kvoli funkcii **isprintf** (overenie tlačiteľnosti znaku)
- **<getopt.h> –** Spracovanie argumentov na vstupe
- **<time.h> –** Určenie času zachytenia packetu

4.2 Práca s packetmi a sieťovými prvkami [2]

- **<pcap/pcap.h> -** Zachytávanie a filtrovanie packetov
- **<netinet/ip_icmp.h> -** Práca s paketmi typu ICMPv4 a ICMPv6
- **<netinet/udp.h> -** Práca s protokolom UDP (štruktúra hlavičky)
- **<netinet/tcp.h> -** Práca s protokolom TCP (štruktúra hlavičky)
- **<netinet/ether.h> -** Štruktúra ethernetovej hlavičky a práca s ňou
- **<netinet/ip6.h> -** Štruktúra IPv6 hlavičky
- **<netinet/ip.h> -** Štruktúra IPv4 hlavičky
- **<arpa/inet.h> -** Funkcie na prevod IPv4 a IPv6 adries do binárnej formy a naspäť

5 Návratové hodnoty

- **0 -** Program skončil bez chyby.
- **99 -** Vnútorná chyba programu (malloc).
- **21 -** Chyba argumentu na vstupe.
- **22 -** Chyba pri zlyhaní funkcie z knižnice `PCAP`.

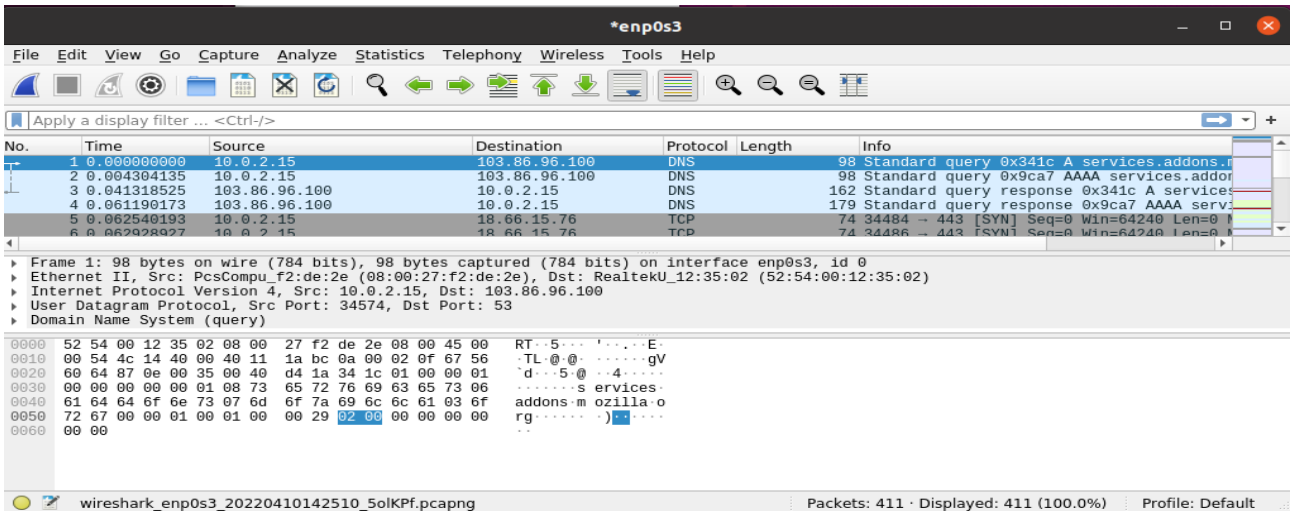
6 Testovanie

Testovanie prebiehalo pomocou nástroja **Wireshark** na virtuálnom stroji so systémom `Ubuntu 20.04.4 LTS`. Nástroj **Wireshark** a program **ipk-sniffer** boli súbežne spustené a následne sa buď otvoril prehliadač a porovnával ten istý paket alebo paket odoslaný pomocou príkazov `ping / ping -6, curl / curl -6`.

```
--------------------UDP--------------------
timestamp: 2022-04-10T14:25:33.890
src MAC: 08:00:27:f2:de:2e
dst MAC: 52:54:00:12:35:02
frame lenght: 98 bytes
src IP: 10.0.2.15
dst IP: 103.86.96.100
src port: 34574
dst port: 53

DATA:
0x0000:  52 54 00 12 35 02 08 00  27 f2 de 2e 08 00 45 00  RT..5... '.....E.
0x0010:  00 54 4c 14 40 00 40 11  1a bc 0a 00 02 0f 67 56  .TL.@.@. ......gV
0x0020:  60 64 87 0e 00 35 00 40  d4 1a 34 1c 01 00 00 01  `d...5.@ ..4.....
0x0030:  00 00 00 00 00 01 08 73  65 72 76 69 63 65 73 06  .......s ervices.
0x0040:  61 64 64 6f 6e 73 07 6d  6f 7a 69 6c 6c 61 03 6f  addons.m ozilla.o
0x0050:  72 67 00 00 01 00 01 00  00 29 02 00 00 00 00 00  rg...... .)......
0x0060:  00 00                                            ..
##########################################
```

Obr. 5: Zachytenie UDP paketu pomocou ipk-sniffer



Obr. 6: Zachytenie UDP paketu pomocou wireshark

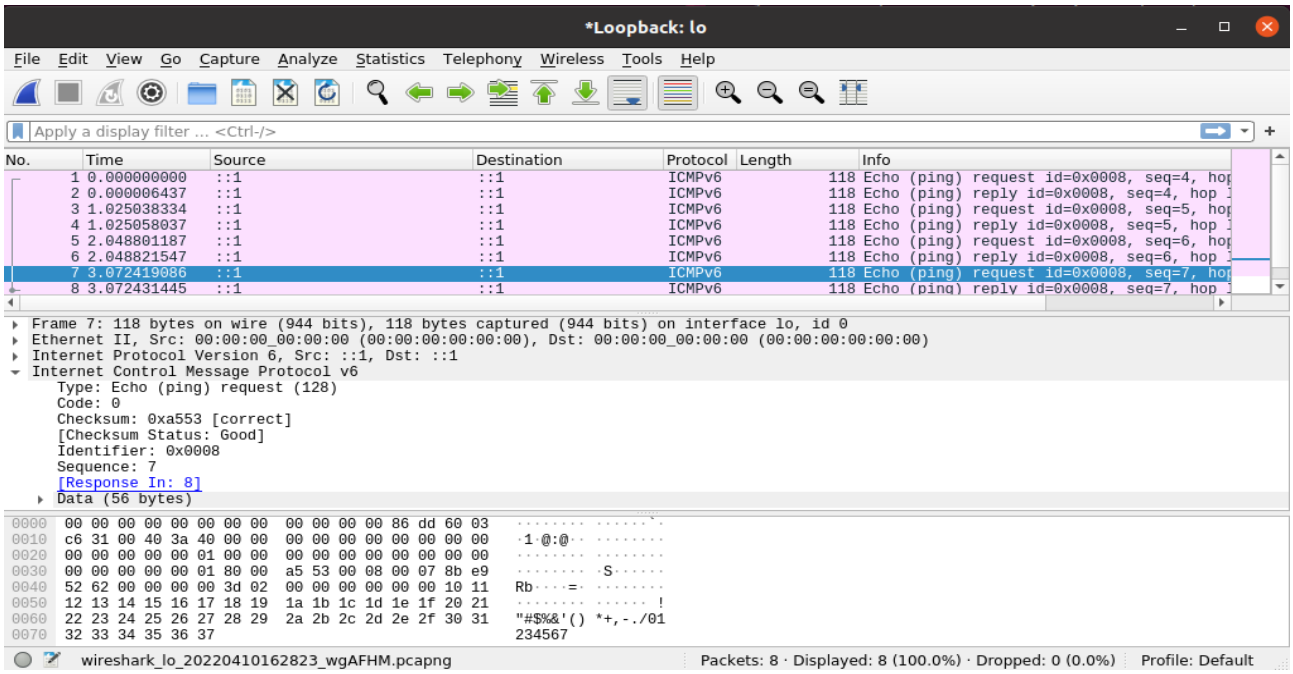
```
xbubla02@xbubla02:~/Desktop$ ping -6 localhost
PING localhost(ip6-localhost (::1)) 56 data bytes
64 bytes from ip6-localhost (::1): icmp_seq=1 ttl=64 time=0.013 ms
64 bytes from ip6-localhost (::1): icmp_seq=2 ttl=64 time=0.069 ms
64 bytes from ip6-localhost (::1): icmp_seq=3 ttl=64 time=0.068 ms
```

Obr. 7: Príkaz ping -6 pre testovanie ICMPv6

```
--------------------ICMP--------------------
timestamp: 2022-04-10T16:28:27.0
src MAC: 00:00:00:00:00:00
dst MAC: 00:00:00:00:00:00
frame lenght: 118 bytes
src IP: ::1
dst IP: ::1
Type: 128
Code: 0

DATA:
0x0000:  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 03  ........ ......`.
0x0010:  c6 31 00 40 3a 40 00 00  00 00 00 00 00 00 00 00  .1.@:@.. ........
0x0020:  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  ........ ........
0x0030:  00 00 00 00 00 01 80 00  a5 53 00 08 00 07 8b e9  ........ .S......
0x0040:  52 62 00 00 00 00 3d 02  00 00 00 00 00 00 10 11  Rb....=. ........
0x0050:  12 13 14 15 16 17 18 19  1a 1b 1c 1d 1e 1f 20 21  ........ ...... !
0x0060:  22 23 24 25 26 27 28 29  2a 2b 2c 2d 2e 2f 30 31  "#$%&'() *+,-./01
0x0070:  32 33 34 35 36 37                                 234567
##########################################
```

Obr. 8: Zachytenie ICMPv6 paketu pomocou ipk-sniffer



Obr. 9: Zachytenie ICMPv6 paketu pomocou wireshark

7 Zdroje

Použitá Literatúra

[1] PROGRAMMING WITH PCAP. [online], [vid. 2022-04-02]. Dostupné z: https://www.tcpdump.org/pcap.html

[2] Structs in netinet. [online], [vid. 2022-04-01]. Dostupné z: https://unix.superglobalmegacorp.com/Net2/netinet.s.html

[3] TCP/IP and tcpdump Pocket Reference Guide. [online], [vid. 2022-04-02]. Dostupné z: https://www.garykessler.net/library/tcpip_prg_GKA.pdf

[4] AG, P.: IT Explained: Packet Sniffing. [online], rev. 2020, [vid. 2022-04-07]. Dostupné z: https://www.paessler.com/it-explained/packet-sniffing

[5] Wikipedia: Internet Control Message Protocol. [online]. Dostupné z: https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol