
# Dokumentace ke projektu do IPK

## Varianta ZETA: Sniffer paketů

Maxim Plička(xplick04)

24. dubna 2022

# Obsah

**1 Úvod** **2**

**2 Návrh aplikace** **2**
2.1 Zpracování argumentů . . . . . 2
2.2 Připojení na rozhraní . . . . . 2
2.3 Zpracovávání paketů . . . . . 2
2.4 Tisk dat . . . . . 2

**3 Popis implementace** **2**
3.1 Hlavní funkce programu . . . . . 2
3.2 Zpracovávání paketů a tisk dat . . . . . 3

**4 Testování** **4**
4.1 UDP . . . . . 4
4.2 ICMPv4 . . . . . 5
4.3 ICMPv6 . . . . . 6

# 1 Úvod

Sniffer paketů je program, který se používá pro zachycení jednotlivých paketů. Pracuje tak, že postupně zpracovává data jednotlivých paketů a na základě toho vypisuje informace o nich. Dále může také pracovat s různými filtry. Tyto filtry následně specifikují typ/typy paketů, které chce uživatel zpracovat.

# 2 Návrh aplikace

Aplikace se dělí na čtyři základní části:

## 2.1 Zpracování argumentů

V této části program zpracuje argumenty a pošle je do hlavního cyklu programu. Dále se rozhodne mezi výpisem jednotlivých připojených rozhraní a připojením na dané rozhraní.

## 2.2 Připojení na rozhraní

Program musí prvně otestovat, že rohraní existuje a také že se k němu dá připojit. Následně se aplikuje zadaný filtr a začne zpracovávání jednotlivých paketů.

## 2.3 Zpracovávání paketů

U paketů se prvně vezme ethernet hlavička, v které se rozhodne zda se jedná o IPv4, IPv6 a nebo ARP paket. Následně u IPv4 a IPv6 rozděluje mezi další tři kategorie TCP, UDP a ICMP.

## 2.4 Tisk dat

Tisk samotných dat jsem chtěl původně udělat až po zpracování celého paketu, ale nakonec jsem se rozhodl tisknout data již během zpracovávání, kde k nim byl snadný přístup.

# 3 Popis implementace

## 3.1 Hlavní funkce programu

Na začátku se zavolá funkce `argParse()`. Následně se pomocí funkce `getopt_long()` zpracují argumenty předané z **argc** a **argv**. Funkce `getopt_long()` se oproti `getopt()` liší v tom, že umí zpracovávat i dlouhé varianty argumentů specifikované v **long_options[]**. Souběžně se zpracováváním argumentů se tvoří řetězec **filtr[]**, který se využije při pozdější aplikaci filtru na pakety.

Hlavní část programu dále vyhodnotí argumenty. V případě, že uživatel nespecifikoval žádná rozhraní (**interfaceFlag** se rovná nule) se zavolá funkce `printInterface()`. V této funkci se následně zavolá `pcap_findalldevs()`, která do proměnné **devs** uloží vázaný list všech dostupných rozhraní. Poté pomocí cyklu prochází list a během toho vypisuje jejich název. Po vytisknutí všech rohraní se program ukončí.

V případě že uživatel zadal specifické rozhraní se v hlavní části zavolají následující funkce:

- `pcap_lookupnet()` - získá číslo sítě a masku z **interface[]**
- `pcap_open_live()` - otevře **interface[]**, z kterého chce uživatel načítat pakety
- `pcap_datalink()` - kontroluje, zda se jedná o ethernet paket
- `pcap_compile()` - kompiluje **filtr[]**
- `pcap_setfilter()` - aplikuje **filtr[]** na přijímané pakety
- `pcap_loop()` - hlavní cyklus, kde se volá callback funkce `parsePacket()`
- `pcap_close()` - zavře **interface[]**
- `pcap_freecode()` - uvolní program načtený ve **fp**, který byl využit u kompilace

## 3.2 Zpracovávání paketů a tisk dat

Funkce `parsePacket()` vypíše čas, MAC adresu odesílatele/příjemce a velikost rámce. Čas se vypisuje zavoláním pomocné funkce `printTimeStamp()`, kde se čas z ethernet hlavičky předělá na požadovaný formát. Poté se pomocí knihovny **ethernet.h** rozhoduje, zda se jedná o IPv4, IPv6 nebo ARP paket. V této funkci se dále volají funkce `parseIPv4()` a `parseIPv6()`. V případě ARP se jen tisknou data.

Ve funkci `parseIPv4()` se pomocí knihovny **ip.h** získá IP odesílatele a příjemce, které se vypíší. Následně se rozhodne, zda se jedná o TCP, UDP a nebo ICMP paket.

Ve funkci `parseIPv6()` se druhy paketů zpracují stejně jako v `parseIPv4()`. Rozdíl je ve využití knihovny **ip6.h** a téke při porovnávání ICMP paketu. Zde se musí porovnávat s jiným číslem, jelikož ICMP má rozdílný kód v IPv4 a IPv6.

Ve funkcích `parseTCP()` a `parseUDP()` se vytiskne port odesílatele a příjemce. Nakonec se ve funkci `parsePacket()` pomocí funkcí `printPayload()` a `print_hex_ascii_line()` vypíší data.

V připadě ICMP paketu se dál nic nevypisuje

# 4 Testování

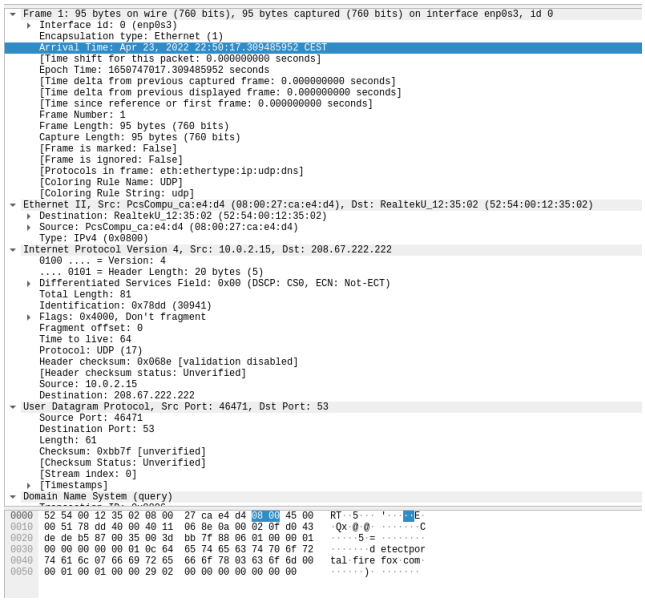
V následujících testech se budou porovnávat výstupy z Wiresharku a implementovaného **ipk-sniffer**. Pro příklad je přiloženo pár testů.

## 4.1 UDP

```
student@student-vm:~/IPK$ sudo ./ipk-sniffer -i enp0s3
timestamp: 2022-04-23T22:50:17.309+02:00
src MAC: 08:00:27:ca:e4:d4
dst MAC: 52:54:00:12:35:02
frame length: 95 bytes
src IP: 10.0.2.15
dst IP: 208.67.222.222
src port: 46471
dst port: 53
------------------------------------------------------------

0x0000   52 54 00 12 35 02 08 00  27 ca e4 d4 08 00 45 00    RT..5...'.....E.
0x0010   00 51 78 dd 40 00 40 11  06 8e 0a 00 02 0f d0 43    .Qx.@.@........C
0x0020   de de b5 87 00 35 00 3d  bb 7f 88 06 01 00 00 01    .....5.=........
0x0030   00 00 00 00 00 01 0c 64  65 74 65 63 74 70 6f 72    .......detectpor
0x0040   74 61 6c 07 66 69 72 65  66 6f 78 03 63 6f 6d 00    tal.firefox.com.
0x0050   00 01 00 01 00 00 29 02  00 00 00 00 00 00 00       ......).........
```

Obrázek 1: ipk-sniffer




Obrázek 2: WireShark

## 4.2 ICMPv4

```
student@student-vm:~/IPK$ sudo ./ipk-sniffer -i lo
timestamp: 2022-04-23T22:38:54.878+02:00
src MAC: 00:00:00:00:00:00
dst MAC: 00:00:00:00:00:00
frame length: 98 bytes
src IP: 127.0.0.1
dst IP: 127.0.0.1
----------------------------------------------------------------

0x0000   00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00    ..............E.
0x0010   00 54 ab 7a 40 00 40 01  91 2c 7f 00 00 01 7f 00    .T.z@.@..,......
0x0020   00 01 08 00 07 f0 00 10  00 01 de 63 64 62 00 00    ...........cdb..
0x0030   00 00 e1 65 0d 00 00 00  00 00 10 11 12 13 14 15    ...e............
0x0040   16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25    .......... !"#$%
0x0050   26 27 28 29                                        &'()
```

Obrázek 3: ipk-sniffer

```
Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface lo, id 0
    Interface id: 0 (lo)
    Encapsulation type: Ethernet (1)
    Arrival Time: Apr 23, 2022 22:38:54.878067623 CEST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1650746334.878067623 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 98 bytes (784 bits)
    Capture Length: 98 bytes (784 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:icmp:data]
    [Coloring Rule Name: ICMP]
    [Coloring Rule String: icmp || icmpv6]
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Destination: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Source: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 84
    Identification: 0xab7a (43898)
    Flags: 0x4000, Don't fragment
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0x912c [validation disabled]
    [Header checksum status: Unverified]
    Source: 127.0.0.1
    Destination: 127.0.0.1
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x07f0 [correct]
    [Checksum Status: Good]
    Identifier (BE): 16 (0x0010)
    Identifier (LE): 4096 (0x1000)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    [Response frame: 2]
    Timestamp from icmp data: Apr 23, 2022 22:38:54.000000000 CEST
    [Timestamp from icmp data (relative): 0.878067623 seconds]
    Data (48 bytes)
        Data: e1650d0000000000101112131415161718191a1b1c1d1e1f…
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ·····E·
0010  00 54 ab 7a 40 00 40 01  91 2c 7f 00 00 01 7f 00   ·T·z@·@· ·,·····
0020  00 01 08 00 07 f0 00 10  00 01 de 63 64 62 00 00   ·····cdb··
0030  00 00 e1 65 0d 00 00 00  00 00 10 11 12 13 14 15   ···e····
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ·· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                             67
```

Obrázek 4: WireShark

## 4.3 ICMPv6

```
student@student-vm:~/IPK$ sudo ./ipk-sniffer -i lo
timestamp: 2022-04-23T22:40:36.519+02:00
src MAC: 00:00:00:00:00:00
dst MAC: 00:00:00:00:00:00
frame length: 80 bytes
src IP: 127.0.0.1
dst IP: 127.0.0.53
src port: 48992
dst port: 53
---------------------------------------------------------------

0x0000   00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00    ..............E.
0x0010   00 42 4c 8e 40 00 40 11  ef e6 7f 00 00 01 7f 00    .BL.@.@.........
0x0020   00 35 bf 60 00 35 00 2e  fe 75 54 2e 01 20 00 01    .5.`.5...uT.. ..
0x0030   00 00 00 00 00 01 09 6c  6f 63 61 6c 68 6f 73 74    .......localhost
0x0040   00 00                                              ..
```

Obrázek 5: ipk-sniffer

```
▾ Frame 2: 108 bytes on wire (864 bits), 108 bytes captured (864 bits) on interface lo, id 0
  ▸ Interface id: 0 (lo)
    Encapsulation type: Ethernet (1)
    Arrival Time: Apr 23, 2022 22:40:36.519431742 CEST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1650746436.519431742 seconds
    [Time delta from previous captured frame: 0.000128596 seconds]
    [Time delta from previous displayed frame: 0.000128596 seconds]
    [Time since reference or first frame: 0.000128596 seconds]
    Frame Number: 2
    Frame Length: 108 bytes (864 bits)
    Capture Length: 108 bytes (864 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:udp:dns]
    [Coloring Rule Name: UDP]
    [Coloring Rule String: udp]
▾ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
  ▸ Destination: 00:00:00_00:00:00 (00:00:00:00:00:00)
  ▸ Source: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 127.0.0.53, Dst: 127.0.0.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 94
    Identification: 0x50e6 (20710)
  ▸ Flags: 0x4000, Don't fragment
    Fragment offset: 0
    Time to live: 64
    Protocol: UDP (17)
    Header checksum: 0xeb72 [validation disabled]
    [Header checksum status: Unverified]
    Source: 127.0.0.53
    Destination: 127.0.0.1
▾ User Datagram Protocol, Src Port: 53, Dst Port: 48992
    Source Port: 53
    Destination Port: 48992
    Length: 74
    Checksum: 0xfe91 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
  ▸ [Timestamps]

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E·
0010  00 5e 50 e6 40 00 40 11  eb 72 7f 00 00 35 7f 00   ·^P·@·@· ·r···5··
0020  00 01 00 35 bf 60 00 4a  fe 91 54 2e 81 80 00 01   ···5·`·J ··T.····
0030  00 01 00 00 00 01 09 6c  6f 63 61 6c 68 6f 73 74   ·······l ocalhost
0040  00 00 1c 00 01 c0 0c 00  1c 00 01 00 00 00 00 00   ........ ........
0050  10 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0060  01 00 00 29 ff d6 00 00  00 00 00 00               ···)···· ····
```

Obrázek 6: WireShark