BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Computer Communications and Networks

2020/2021

Packet Sniffer

Lukáš Plevač (xpleva07)

Brno, April 22, 2021

Contents

1. Introduction

This is manual for ipk-sniffer. Ipk-sniffer is packet sniffer with basic filters support. Sniffed packets is printed with destination and source port/address and payload using HexDump.

1.1. Usage

```
ipk-sniffer [-i interface_name | --interface interface_name] {-p port}
{[--tcp|-t] [--udp|-u] [--arp] [--icmp]} {-n num}
```

Square brackets represent mandatory arguments, arguments in curly brackets are optional. Specific is use of -i argument, if you do not pass value for this argument, sniffer shows all available interfaces (only what can be used).

1.2. Parameters

-i, --interface - interface from sniff packets if empty print list of interfaces
-p - if set filter by port in src or dsc.
-t, --tcp - show only TCP packets
-u, --udp - show only UDP packets
--icmp - show only ICMPv4 and ICMPv6 packets
--arp - show only ARP packets
-n - number of packets to sniff, if not set is used 1
-h, --help - print help

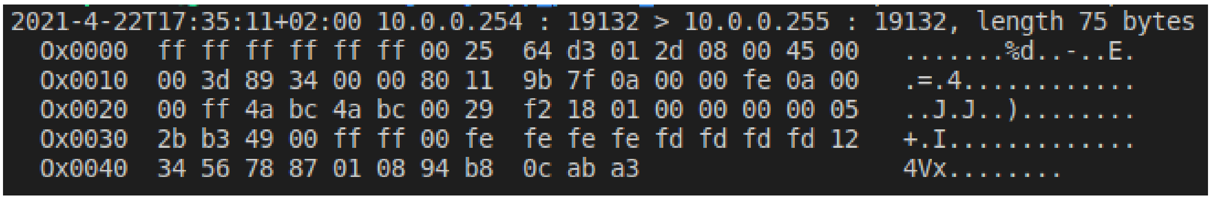
filter parameters (--tcp, --arp, --icmp, --udp) works in or mode. For example --tcp --udp means tcp or udp.

2. Output

Output of the sniffer is printed to STDOUT (normal output) or STDERR (error messages). If the sniffer ends with error, return code is non zero else return code is zero.

2.1. Format of printed packet

Output of every packet have a header. Format of header is: **{time} {src_ip} : {src_port} > {dst_ip} : {dst_port}, length {packet_size} bytes.** After header is HexDump of packet payload with heades. Format of HexDump line is: **{offset_number} {8 BYTES as HEX} {8 BYTES as HEX} {16 BYTES decoded as ASCII}**



```
2021-4-22T17:35:11+02:00 10.0.0.254 : 19132 > 10.0.0.255 : 19132, length 75 bytes
  0x0000  ff ff ff ff ff ff 00 25  64 d3 01 2d 08 00 45 00   .......%d..-..E.
  0x0010  00 3d 89 34 00 00 80 11  9b 7f 0a 00 00 fe 0a 00   .=.4............
  0x0020  00 ff 4a bc 4a bc 00 29  f2 18 01 00 00 00 00 05   ..J.J..)........
  0x0030  2b b3 49 00 ff ff 00 fe  fe fe fe fd fd fd fd 12   +.I.............
  0x0040  34 56 78 87 01 08 94 b8  0c ab a3                  4Vx........
```

Figure 1. printed packet example

3. Dependencies, build and install

The project is written in cpp with library pcap.

3. 1. Dependencies

A Successful build of a project needs few libraries, g++ compiler and Makefile support. Project needs a pcap library, netinet library, arpa library. Netinet library and arpa library are part of most linux distributions, project need it for specification of IP, IPv6, TCP and UDP headers. Pcap library must be installed, in debian base systems is in package libpcap-dev.

3.2. Build

Project is builded using g++. Can be builded automatically using **make**, or using **g++ -I ./include ./src/* -o ipk-sniffer -lpcap**

3.3. Install/Uninstall

Project after build is fully executable, but can be installed in the system using **make install**. This command moves the ipk-sniffer file from build to /usr/bin. For uninstall can be used **make uninstall**, this removes the file /usr/bin/ipk-sniffer.

4. Implementation details

4.1. files

src/main.cpp - main file of implementation. Parse cli argument and work with sniffer.cpp
src/sniffer.cpp - Implementation of sniffer - sniff packets, decode packets, support function for work with pockets and interfaces
include/sniffer.h - header file for src/sniffer.c

4.2. Classes

sniffer - Implementation of sniffer - sniff packets, decode packets, support function for work with pockets and interfaces

4.3. Data structures

All main data structures are in headers files.
struct arp_header - ARP protocol header by https://en.wikipedia.org/wiki/Address_Resolution_Protocol (April 2021)
l1_packet - pcap_pkthdr header and readed bytes from interface
l3_packet - Packet on layer 3 with layer 2 support for ARP and ETHERNET header
l4_packet - Packet on layer 4 for protocols TCP, UDP and ICMP/ICMPv6

4.4. Cli parameters parsing

Parameters are parsed using **getopt** parsing is implemented in **src/main.cpp** using case for every argument and default option for wrong parameters and -i without value

4.5. Interface open

Interface is opened when is called the constructor of **sniffer** object**.** Conscructor open interface using **PCAP_OPEN_LIVE,** if this fail constructor throw runtime exception, what must be catched in the upper function.

4.6. Packet sniffing

Packet is sniffed using method **sniffer::sniff()** this method return l1_packet. Packet is readed using function **pcap_next. sniffer::sniff()** is called in loop in main.cpp (loop in range n).

4.7. Packet parse

Pocket parse is implemented in **main.cpp** using support methods **sniffer::l3_decode** and **sniffer::l4_decode**. After packet is sniffed, packet is immediately converted to **l3_packet** using **sniffer::l3_decode** (if is only l2 protocol in packet, then is ether_hdr only set), and then converted to **l4_packet** using **sniffer::l4_decode** (if last packet protocol work in < 4 layer, then nothing happens). Layers decoding is done using a recasing pointer on the packet as protocol header. After read this header (usually here is the protocol type of the next header), can be readed next header, if it exists, using the same method but recating pointer after the last header. Header reading order for supported protocols:

ethernet -> ip -> tcp
ethernet -> ipv6 -> tcp
ethernet -> ip -> udp
ethernet -> ipv6 -> udp
ethernet -> arp
ethernet -> ip -> icmp
ethernet -> ip6 -> icmpv6

From headers are read IP addresses, ports, protocols types and mac addresses, but in headers is much more information, which can be used in future versions of sniffer.

4.8. Filter packets

Packets in a project can be filtered by protocol or port. Port filtering is done using **sniffer::set_filter()** what call **pcap_compile()** and **pcap_setfilter().** Filter for port is constructed from this string **port: {port_num}.** Protocol filtering is not implemented in sniffer.cpp, must be done in main.cpp, using simple if for protocol number from method **sniffer::get_protocol()** this function return protocol number, readed from **l3_packet** scruct headers.

4.9. Print packets

Packets after sniff, parse and filter are printed to STDOUT. Printing is done in main.cpp using **printf()** function. Firstly is printed pseudo header, with time when packet be sniffed, src and dst address from methods **sniffer::get_src()** and **sniffer::get_dst(),** if protocol is UDP or TCP src and dst ports from methods **sniffer::get_src_port()** and **sniffer::get_dst_port(),** as last in header is printed length of packet header form **l1_packet.** Then is printed body of packet using **sniffer::hex_dump()** this printed packet as bytes in HEX and on end of line prints ASCII od this bytes.

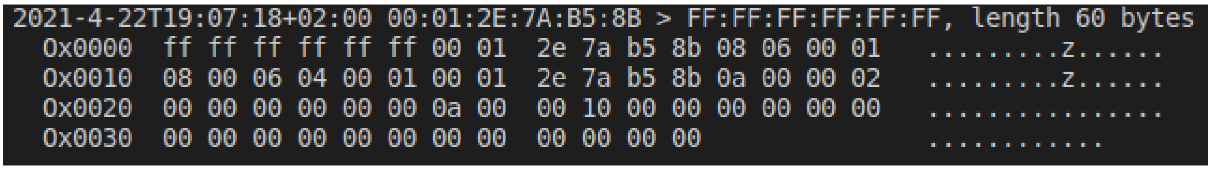
4.10. Close interface

interface is automatically closed when the **sniffer** destructor is called. Destructor use for close interface and free all resources function **pcap_close()**

5. Testing

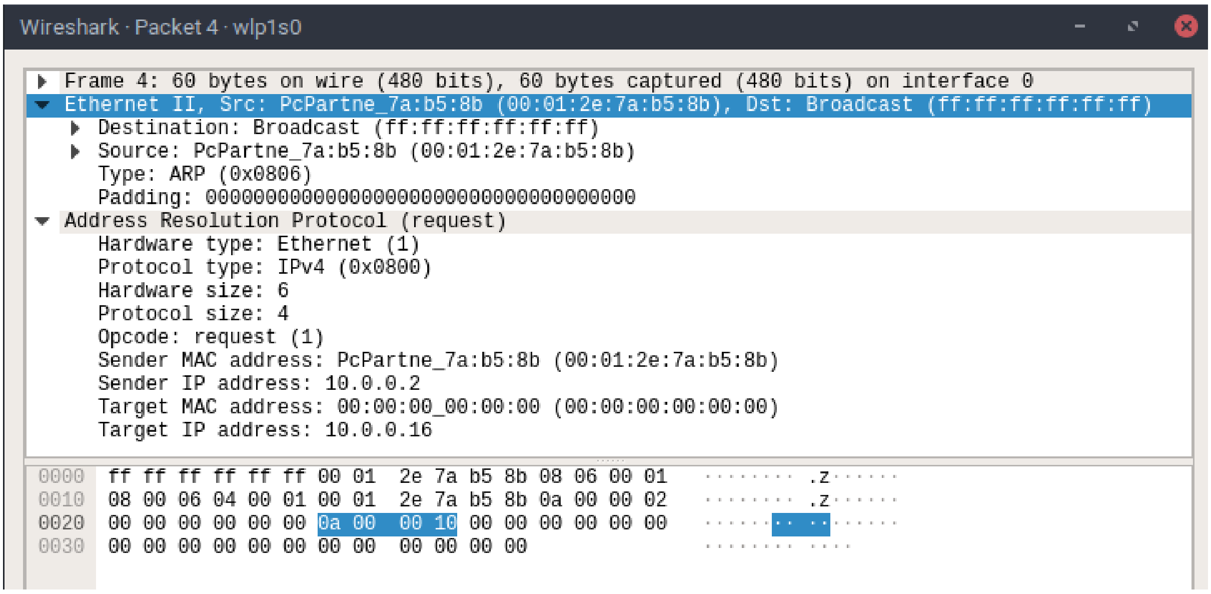
Application is tested on a referencial machine (Ubuntu 20.04.2 LTS) and development machine (Arch Linux) and output is compared with Wireshark 2.6.10.

5.1 ARP



```
2021-4-22T19:07:18+02:00 00:01:2E:7A:B5:8B > FF:FF:FF:FF:FF:FF, length 60 bytes
  0x0000  ff ff ff ff ff ff 00 01  2e 7a b5 8b 08 06 00 01   .........z......
  0x0010  08 00 06 04 00 01 00 01  2e 7a b5 8b 0a 00 00 02   .........z......
  0x0020  00 00 00 00 00 00 0a 00  00 10 00 00 00 00 00 00   ................
  0x0030  00 00 00 00 00 00 00 00  00 00 00 00               ............
```

figure 2. ARP packet in ipk-sniffer

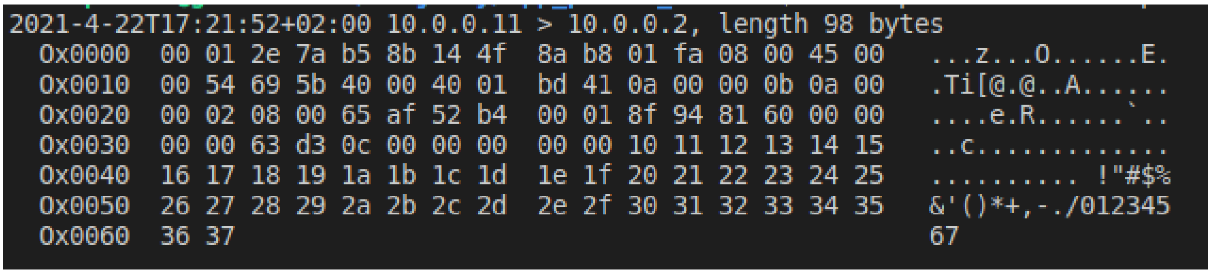


```
▸ Frame 4: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▾ Ethernet II, Src: PcPartne_7a:b5:8b (00:01:2e:7a:b5:8b), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
   ▸ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
   ▸ Source: PcPartne_7a:b5:8b (00:01:2e:7a:b5:8b)
     Type: ARP (0x0806)
     Padding: 000000000000000000000000000000000000
▾ Address Resolution Protocol (request)
     Hardware type: Ethernet (1)
     Protocol type: IPv4 (0x0800)
     Hardware size: 6
     Protocol size: 4
     Opcode: request (1)
     Sender MAC address: PcPartne_7a:b5:8b (00:01:2e:7a:b5:8b)
     Sender IP address: 10.0.0.2
     Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
     Target IP address: 10.0.0.16

0000  ff ff ff ff ff ff 00 01  2e 7a b5 8b 08 06 00 01   ········ ·z······
0010  08 00 06 04 00 01 00 01  2e 7a b5 8b 0a 00 00 02   ········ ·z······
0020  00 00 00 00 00 00 0a 00  00 10 00 00 00 00 00 00   ········ ········
0030  00 00 00 00 00 00 00 00  00 00 00 00               ········ ····
```

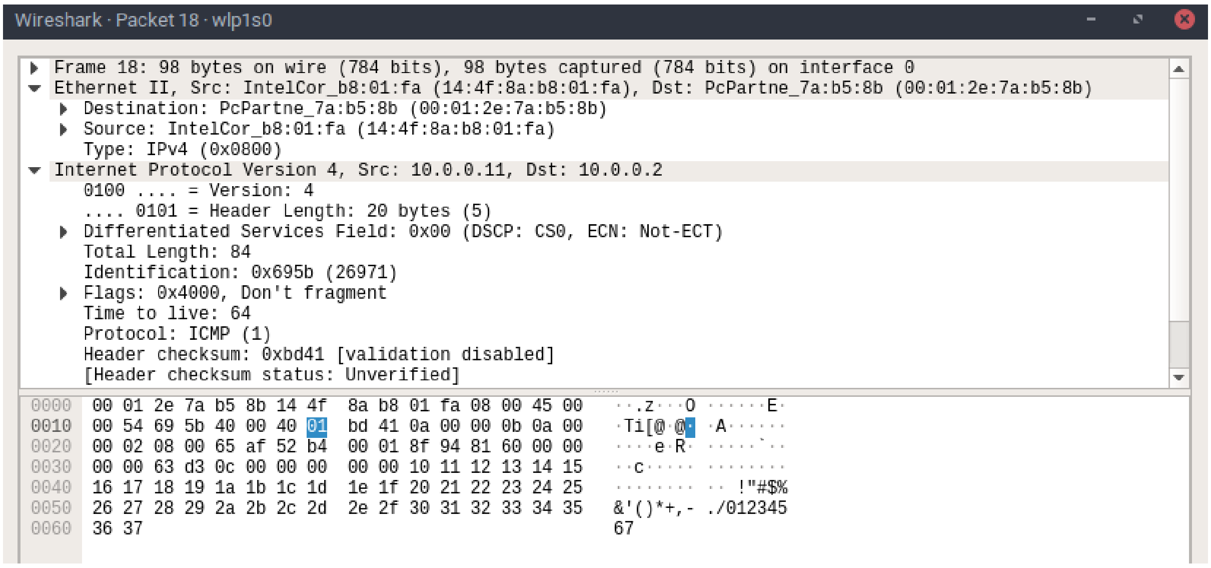
figure 3. ARP packet in Wireshark

5.2. ICMP



```
2021-4-22T17:21:52+02:00 10.0.0.11 > 10.0.0.2, length 98 bytes
  0x0000  00 01 2e 7a b5 8b 14 4f  8a b8 01 fa 08 00 45 00   ...z...O......E.
  0x0010  00 54 69 5b 40 00 40 01  bd 41 0a 00 00 0b 0a 00   .Ti[@.@..A......
  0x0020  00 02 08 00 65 af 52 b4  00 01 8f 94 81 60 00 00   ....e.R......`..
  0x0030  00 00 63 d3 0c 00 00 00  00 00 10 11 12 13 14 15   ..c.............
  0x0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   .......... !"#$%
  0x0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,-./012345
  0x0060  36 37                                              67
```

figure 4. ICMP packet in ipk-sniffer



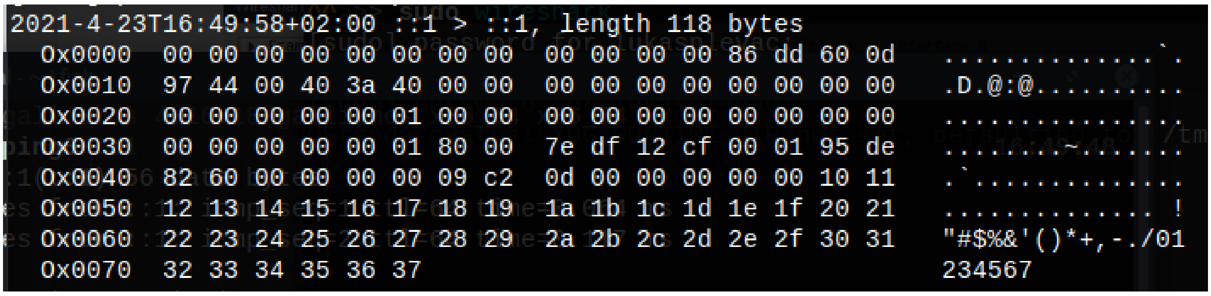
Wireshark · Packet 18 · wlp1s0

```
▸ Frame 18: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▾ Ethernet II, Src: IntelCor_b8:01:fa (14:4f:8a:b8:01:fa), Dst: PcPartne_7a:b5:8b (00:01:2e:7a:b5:8b)
  ▸ Destination: PcPartne_7a:b5:8b (00:01:2e:7a:b5:8b)
  ▸ Source: IntelCor_b8:01:fa (14:4f:8a:b8:01:fa)
    Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 10.0.0.11, Dst: 10.0.0.2
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 84
    Identification: 0x695b (26971)
  ▸ Flags: 0x4000, Don't fragment
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0xbd41 [validation disabled]
    [Header checksum status: Unverified]

0000  00 01 2e 7a b5 8b 14 4f  8a b8 01 fa 08 00 45 00   ··.z···O ······E·
0010  00 54 69 5b 40 00 40 01  bd 41 0a 00 00 0b 0a 00   ·Ti[@·@· ·A······
0020  00 02 08 00 65 af 52 b4  00 01 8f 94 81 60 00 00   ····e·R· ·····`··
0030  00 00 63 d3 0c 00 00 00  00 00 10 11 12 13 14 15   ··c····· ········
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
```

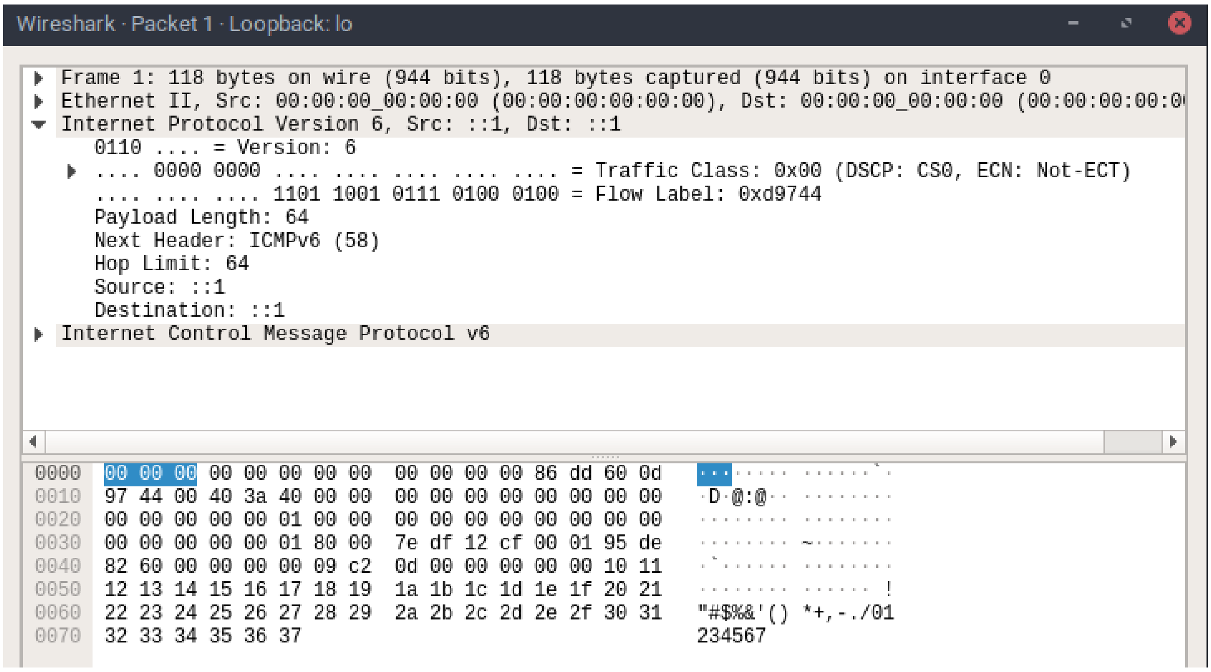
figure 5. ICMP packet in Wireshark

5.3. ICMPv6



```
2021-4-23T16:49:58+02:00 ::1 > ::1, length 118 bytes
 0x0000  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 0d  ..............`.
 0x0010  97 44 00 40 3a 40 00 00  00 00 00 00 00 00 00 00  .D.@:@..........
 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  7e df 12 cf 00 01 95 de  ........~.......
 0x0040  82 60 00 00 00 00 09 c2  0d 00 00 00 00 00 10 11  .`..............
 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
```

figure 6. ICMPv6 in ipk-sniffer



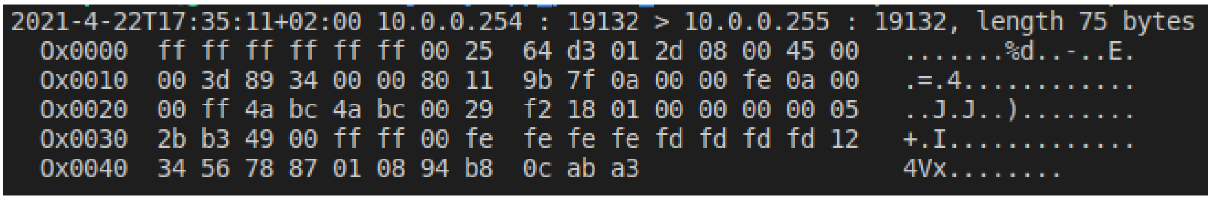
Wireshark · Packet 1 · Loopback: lo

```
▸ Frame 1: 118 bytes on wire (944 bits), 118 bytes captured (944 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:0
▾ Internet Protocol Version 6, Src: ::1, Dst: ::1
     0110 .... = Version: 6
   ▸ .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
     .... .... .... 1101 1001 0111 0100 0100 = Flow Label: 0xd9744
     Payload Length: 64
     Next Header: ICMPv6 (58)
     Hop Limit: 64
     Source: ::1
     Destination: ::1
▸ Internet Control Message Protocol v6
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 0d   ········ ······`·
0010  97 44 00 40 3a 40 00 00  00 00 00 00 00 00 00 00   ·D·@:@·· ········
0020  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00   ········ ········
0030  00 00 00 00 00 01 80 00  7e df 12 cf 00 01 95 de   ········ ~·······
0040  82 60 00 00 00 00 09 c2  0d 00 00 00 00 00 10 11   ·`······ ········
0050  12 13 14 15 16 17 18 19  1a 1b 1c 1d 1e 1f 20 21   ········ ······ !
0060  22 23 24 25 26 27 28 29  2a 2b 2c 2d 2e 2f 30 31   "#$%&'() *+,-./01
0070  32 33 34 35 36 37                                  234567
```

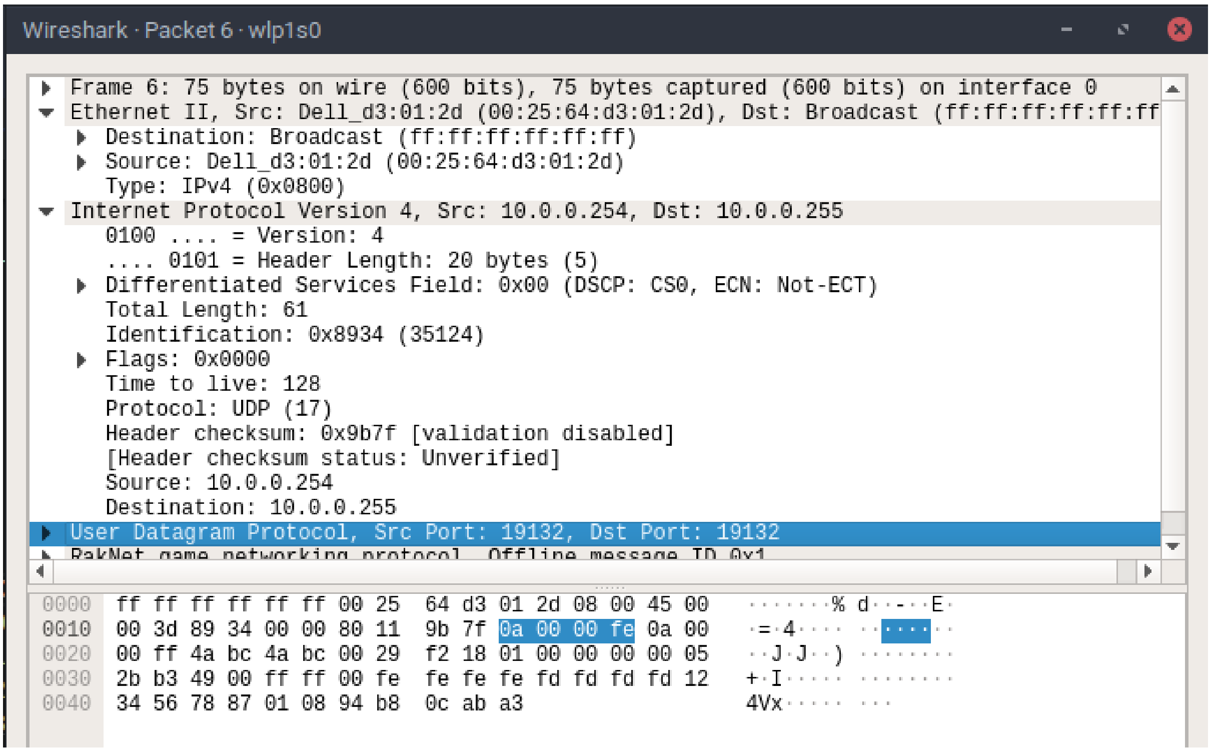
figure 7. ICMPv6 in Wireshark

5.4. UDP on IPV4



```
2021-4-22T17:35:11+02:00 10.0.0.254 : 19132 > 10.0.0.255 : 19132, length 75 bytes
  0x0000  ff ff ff ff ff ff 00 25  64 d3 01 2d 08 00 45 00   .......%d..-..E.
  0x0010  00 3d 89 34 00 00 80 11  9b 7f 0a 00 00 fe 0a 00   .=.4............
  0x0020  00 ff 4a bc 4a bc 00 29  f2 18 01 00 00 00 00 05   ..J.J..)........
  0x0030  2b b3 49 00 ff ff 00 fe  fe fe fe fd fd fd fd 12   +.I.............
  0x0040  34 56 78 87 01 08 94 b8  0c ab a3                  4Vx........
```

figure 8. UDP packet in ipk-sniffer

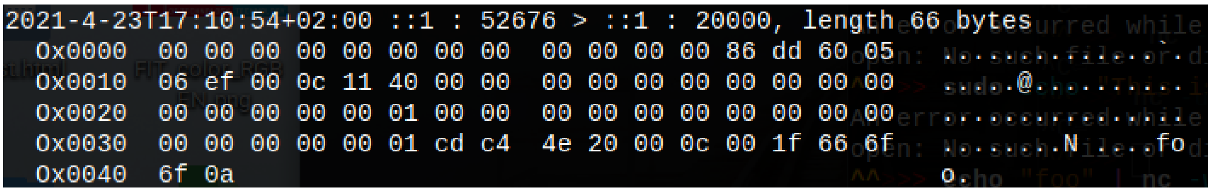


```
▶ Frame 6: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface 0
▼ Ethernet II, Src: Dell_d3:01:2d (00:25:64:d3:01:2d), Dst: Broadcast (ff:ff:ff:ff:ff:ff
   ▶ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
   ▶ Source: Dell_d3:01:2d (00:25:64:d3:01:2d)
     Type: IPv4 (0x0800)
▼ Internet Protocol Version 4, Src: 10.0.0.254, Dst: 10.0.0.255
     0100 .... = Version: 4
     .... 0101 = Header Length: 20 bytes (5)
   ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
     Total Length: 61
     Identification: 0x8934 (35124)
   ▶ Flags: 0x0000
     Time to live: 128
     Protocol: UDP (17)
     Header checksum: 0x9b7f [validation disabled]
     [Header checksum status: Unverified]
     Source: 10.0.0.254
     Destination: 10.0.0.255
▶ User Datagram Protocol, Src Port: 19132, Dst Port: 19132
▶ RakNet game networking protocol, Offline message ID 0x1

0000  ff ff ff ff ff ff 00 25  64 d3 01 2d 08 00 45 00   ·······% d··-··E·
0010  00 3d 89 34 00 00 80 11  9b 7f 0a 00 00 fe 0a 00   ·=·4···· ········
0020  00 ff 4a bc 4a bc 00 29  f2 18 01 00 00 00 00 05   ··J·J··) ········
0030  2b b3 49 00 ff ff 00 fe  fe fe fe fd fd fd fd 12   +·I····· ········
0040  34 56 78 87 01 08 94 b8  0c ab a3                  4Vx····· ···
```

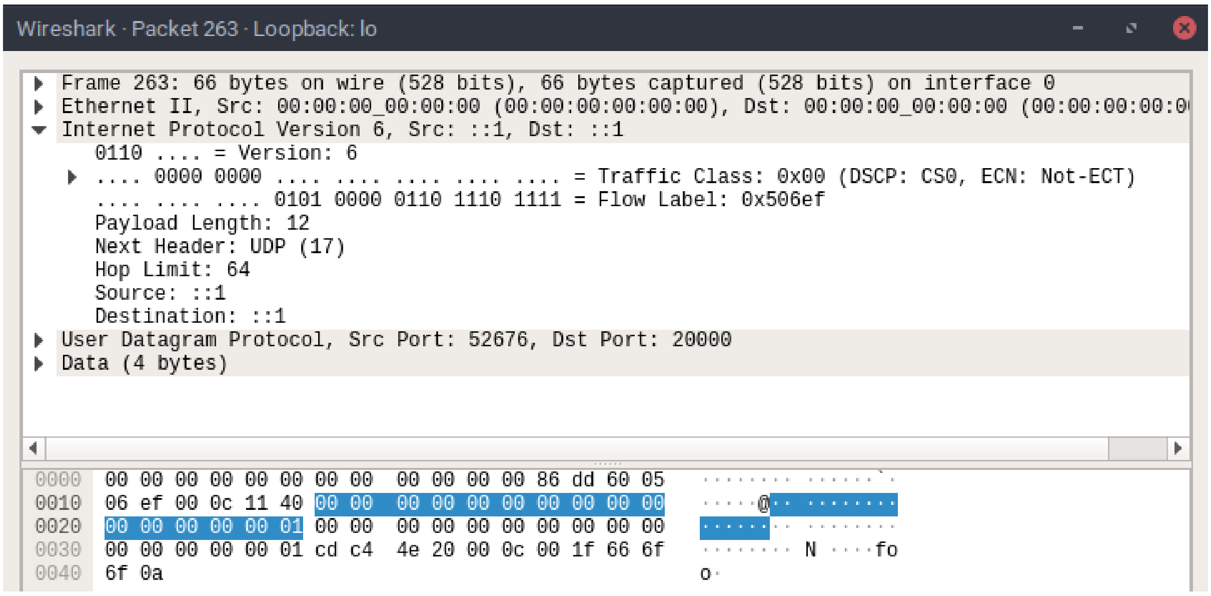
figure 9. UDP packet in WIreshark

5.5. UDP on IPV6



```
2021-4-23T17:10:54+02:00 ::1 : 52676 > ::1 : 20000, length 66 bytes
  0x0000  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 05  ..............`.
  0x0010  06 ef 00 0c 11 40 00 00  00 00 00 00 00 00 00 00  ......@.........
  0x0020  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00  ................
  0x0030  00 00 00 00 00 01 cd c4  4e 20 00 0c 00 1f 66 6f  ........N.....fo
  0x0040  6f 0a                                             o.
```

figure 10. UDP packet in ipk-sniffer

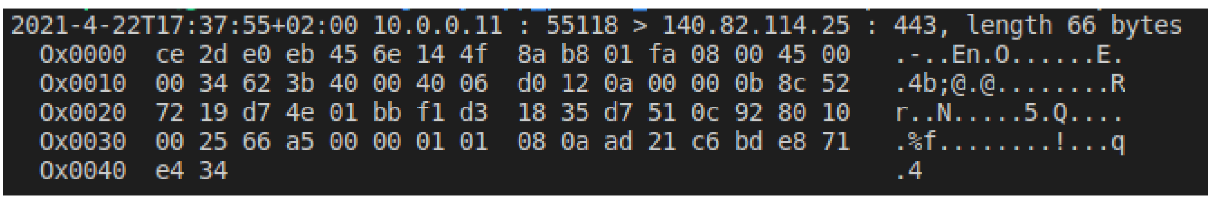


```
▶ Frame 263: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:0
▼ Internet Protocol Version 6, Src: ::1, Dst: ::1
     0110 .... = Version: 6
   ▶ .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
     .... .... .... 0101 0000 0110 1110 1111 = Flow Label: 0x506ef
     Payload Length: 12
     Next Header: UDP (17)
     Hop Limit: 64
     Source: ::1
     Destination: ::1
▶ User Datagram Protocol, Src Port: 52676, Dst Port: 20000
▶ Data (4 bytes)

0000  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 05   ········ ······`·
0010  06 ef 00 0c 11 40 00 00  00 00 00 00 00 00 00 00   ·····@·· ········
0020  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00   ········ ········
0030  00 00 00 00 00 01 cd c4  4e 20 00 0c 00 1f 66 6f   ········ N ····fo
0040  6f 0a                                              o·
```

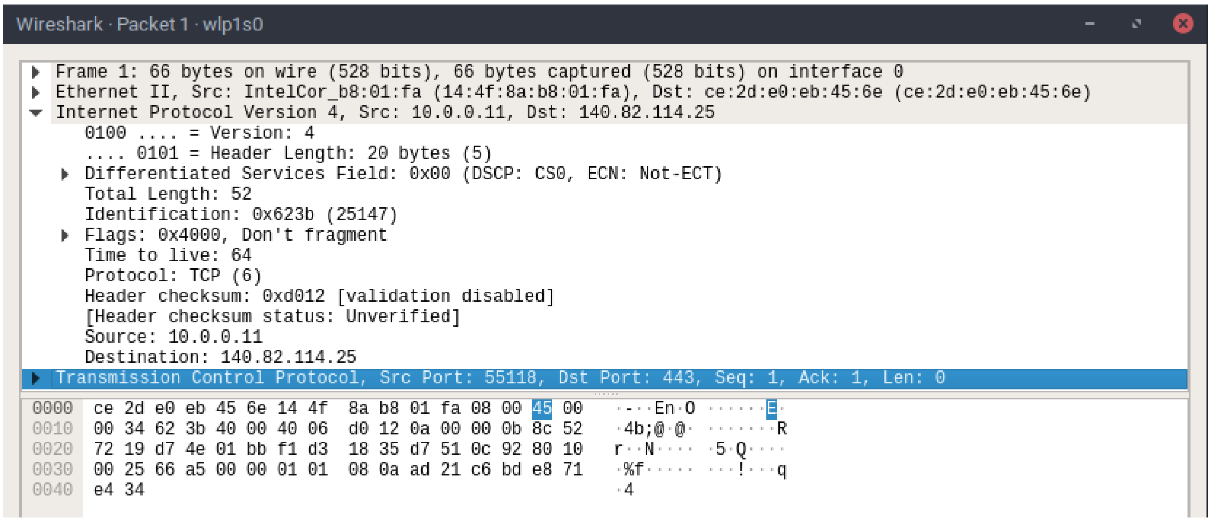
figure 11. UDP packet in WIreshark

5.6. TCP on IPV4



```
2021-4-22T17:37:55+02:00 10.0.0.11 : 55118 > 140.82.114.25 : 443, length 66 bytes
  0x0000  ce 2d e0 eb 45 6e 14 4f  8a b8 01 fa 08 00 45 00   .-..En.O......E.
  0x0010  00 34 62 3b 40 00 40 06  d0 12 0a 00 00 0b 8c 52   .4b;@.@........R
  0x0020  72 19 d7 4e 01 bb f1 d3  18 35 d7 51 0c 92 80 10   r..N.....5.Q....
  0x0030  00 25 66 a5 00 00 01 01  08 0a ad 21 c6 bd e8 71   .%f........!...q
  0x0040  e4 34                                              .4
```

figure 12. TCP packet in ipk-sniffer

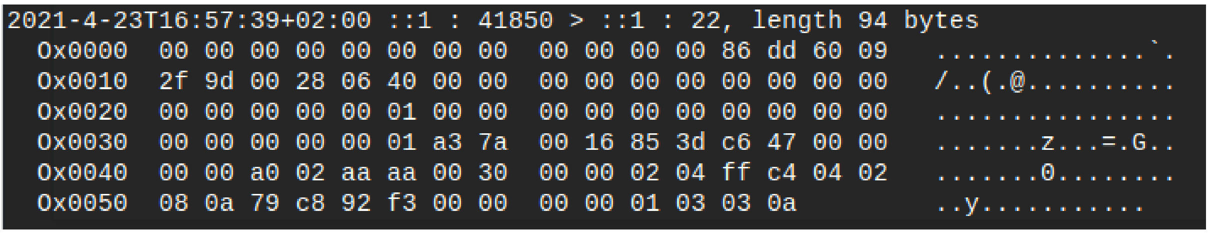


```
▶ Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▶ Ethernet II, Src: IntelCor_b8:01:fa (14:4f:8a:b8:01:fa), Dst: ce:2d:e0:eb:45:6e (ce:2d:e0:eb:45:6e)
▼ Internet Protocol Version 4, Src: 10.0.0.11, Dst: 140.82.114.25
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 52
    Identification: 0x623b (25147)
  ▶ Flags: 0x4000, Don't fragment
    Time to live: 64
    Protocol: TCP (6)
    Header checksum: 0xd012 [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.0.0.11
    Destination: 140.82.114.25
▶ Transmission Control Protocol, Src Port: 55118, Dst Port: 443, Seq: 1, Ack: 1, Len: 0

0000  ce 2d e0 eb 45 6e 14 4f  8a b8 01 fa 08 00 45 00   ·-··En·O ······E·
0010  00 34 62 3b 40 00 40 06  d0 12 0a 00 00 0b 8c 52   ·4b;@·@· ·······R
0020  72 19 d7 4e 01 bb f1 d3  18 35 d7 51 0c 92 80 10   r··N···· ·5·Q····
0030  00 25 66 a5 00 00 01 01  08 0a ad 21 c6 bd e8 71   ·%f····· ···!···q
0040  e4 34                                              ·4
```

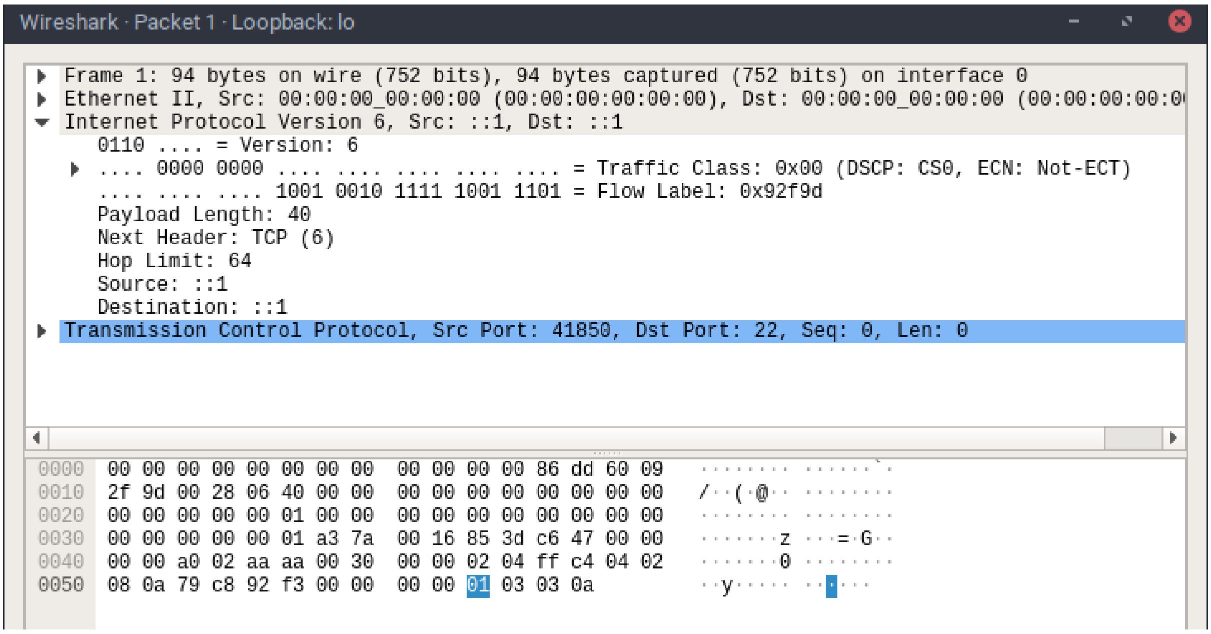
figure 13. TCP packet in Wireshark

5.7. TCP on IPV6



```
2021-4-23T16:57:39+02:00 ::1 : 41850 > ::1 : 22, length 94 bytes
  0x0000  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 09   ..............`.
  0x0010  2f 9d 00 28 06 40 00 00  00 00 00 00 00 00 00 00   /..(.@..........
  0x0020  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00   ................
  0x0030  00 00 00 00 00 01 a3 7a  00 16 85 3d c6 47 00 00   .......z...=.G..
  0x0040  00 00 a0 02 aa aa 00 30  00 00 02 04 ff c4 04 02   .......0........
  0x0050  08 0a 79 c8 92 f3 00 00  00 00 01 03 03 0a         ..y...........
```

figure 14. TCP packet in ipk-sniffer



```
▶ Frame 1: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface 0
▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:0
▼ Internet Protocol Version 6, Src: ::1, Dst: ::1
     0110 .... = Version: 6
   ▶ .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
     .... .... .... 1001 0010 1111 1001 1101 = Flow Label: 0x92f9d
     Payload Length: 40
     Next Header: TCP (6)
     Hop Limit: 64
     Source: ::1
     Destination: ::1
▶ Transmission Control Protocol, Src Port: 41850, Dst Port: 22, Seq: 0, Len: 0
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 09   ········ ······`·
0010  2f 9d 00 28 06 40 00 00  00 00 00 00 00 00 00 00   /··(·@·· ········
0020  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00   ········ ········
0030  00 00 00 00 00 01 a3 7a  00 16 85 3d c6 47 00 00   ·······z ···=·G··
0040  00 00 a0 02 aa aa 00 30  00 00 02 04 ff c4 04 02   ·······0 ········
0050  08 0a 79 c8 92 f3 00 00  00 00 01 03 03 0a         ··y····· ······
```

figure 15. TCP packet in Wireshark

6. Conclusion

Basic specified function is implemented in project and project working correctly, but project showing only a few information about packets like port, address and length. Should be more useful to show more decoded information from methods **sniffer::l3_decode** and **sniffer::l4_decode**, or implement GUI for better work with sniffer.