Computer Communications and Networks

2020/2021

Packet sniffer

Šesták Pavel(xsesta07)

Brno, April 24, 2021

Contents

1 Introduction

This is manual for TUI utility ipk-sniffer.

1.1 Usage

./ipk-sniffer [-i <i> | –interface <i>] {-p <p>} {[–tcp|-t] [–udp|-u] [–arp] [–icmp] } {-n num}
Square brackets represent mandatory arguments, arguments in curly brackets are optional. In case of some kind of error during executing program non zero return value is returned.

1.1.1 Parameters

-i <i> specify interface to listen. If argument is not specified program print all interfaces.
<i> represent interface name: eth0, wlo0, any, etc.
-p <p> filters traffic only on specif port (included source and target).
-t|- -tcp filters just TCP packets.
-u|- -udp filters just UDP packets.
- -icmp filters just ICMP packets.
- -arp filters just ARP packets.
-n <n> Specify number of packets to show, default value is 1.

1.2 Output

Result of sniffer is send to standard output. Output of sniffer corresponds with Wireshark convence.

1.2.1 Format of packet output

timestamp source_IP : source_port > destination_IP : destination_port, length length_in_bytes bytes_offset: bytes_in_hexa bytes_in_ascii.

```
2021-04-16T22:57:27.56+02:00 192.168.0.16 : 42586 > 69.171.250.15 : 443, length
98 bytes
0x0000: c8 d1 2a 8c ce 48 70 77  81 2b e7 0b 08 00 45 00  ..*..Hpw.+....E.
0x0010: 00 54 f7 25 40 00 40 06  43 0b c0 a8 00 10 45 ab  .T.%@.@.C.....E.
0x0020: fa 0f a6 5a 01 bb b9 87  4b 99 82 8c 78 fb 80 18  ...Z....K...x...
0x0030: 01 f5 25 77 00 00 01 01  08 0a 16 c7 a7 43 9b ff  ..%w.........C..
0x0040: 52 22 17 03 03 00 1b c2  4b 7a a2 9c e8 5a 66 b5  R"......Kz...Zf.
0x0050: 4e 61 06 2d 91 cc b8 ac  91 14 85 df c3 6b 2b d3  Na.-.........k+.
0x0060: e1 a3                                            ..
```

Figure 1: Example of output

2 Dependencies and build

This sniffer was implemented in c++ with library pcap.

2.1 Dependencies

For successfully build you need to install two libraries pcap and boost.
On Debian machine with apt repository software can be downloaded packages: libboost-all-dev and libpcap-dev.

2.2 Build

This program was build with g++ compiler, pcap library need to be linked.
Build command: g++ -o ipk-sniffer ipk-sniffer.cpp ipk-sniffer.hpp -lpcap

3 Implementation

3.1 Constants and global variables

layer_2_header_length is unsigned integer and represent length of header on layer 2, which is variable for different protocols.
flags 8bit integer which contains which protocols will be filtered. Every protocol has binary round number, and checking for current argument is implemented with masking (bit and)
packet_show integer which represent number of packets to show, its initialized to default value of argument n, and by argument n can be modified.
packet_show integer with port to filter, can be initialized by specifying port argument.
interface pointer to string which contains name of interface, must be specified by argument interface.
TCPFLAG is set to 1 and its used to set tcp flag to global variable flags.
UDPFLAG is set to 2 and its used to set udp flag to global variable flags.
ICMPFLAG is set to 4 and its used to set icmp flag to global variable flags.
ARPFLAG is set to 8 and its used to set arp flag to global variable flags.
TCP_PROTOCOL is set to 0x06, which is specified by IANA port list table.
UDP_PROTOCOL is set to 0x11, which is specified by IANA port list table.
ICMP_PROTOCOL is set to 0x01, which is specified by IANA port list table.
ETHER_TYPE_IPV4 is set to 0x800, its type of frame on link layer.
ETHER_TYPE_IPV6 is set to 0x86dd, its type of frame on link layer.
ETHER_TYPE_ARP is set to 0x0806, its type of frame on link layer.

3.2 Parsing arguments

In main is called function argumentParse. Function has two parameters argument count and array of arguments. Arguments are compared by if-tree with function compare.

3.3 Opening connection

In main is called function openConnection. Function has parameter which specify interface name. It's called function pcap_open_live on specified interface. Function return handler to sniffing. Depends on arguments is compiled filter to port or specified protocols by functions pcap_compile and pcap_setfilter. It's set length of link layer head and handler is returned to main.

3.4 Handle packets

3.4.1 Process header

In main is called function pcap_loop, we specify handler which we get when we opened connection, number of processed packets is set and its specified function to handle each packet. For each packet we need to decapsulate it. First 14-16bytes (depends on protocol) is reserved for link layer head. From this layer we need get type(IPV4,IPV6,ARP), for comparison we used macros ETHER_TYPE. Depends on protocol we get source and destination addresses. Specify on IPV4 communication. For decoding IPV4 addresses function inet_ntoa is used. For decoding IPV6 addresses inet_ntop function is used. For UDP and TCP communication we need to get ports from transport layer. Network byte order is different from host byte order, for display correct port numbers and other information we need to use function to correct this byte order. For short (16bit) we use function ntohs and for integer(32bit) we use ntohl. In header we need to print timestamp and packet length, this information we get from lowest layer.

3.4.2 Process data

After we print packet header we need to print raw data. Each row contains 16bytes of packet. First we print offset of packet in hexadecimal format. After offset information is printed 16 bytes in hexadecimal format. last column contains same data in ASCII format(Each non printable char is replaced by dot). For printing output in hexadecimal format was used library boost.

3.5 Close connection

After we print the required number of packets we can close connection by calling function pcap_close.

4 Testing

Application was build and test on Ubuntu 20.04.2 LTS and compared with Wireshark 3.2.3 (Git v3.2.3 packaged as 3.2.3-1).

4.1 Link layer type: Ethernet II

These tests were performed at the interface wlo1.

4.1.1 ARP packet

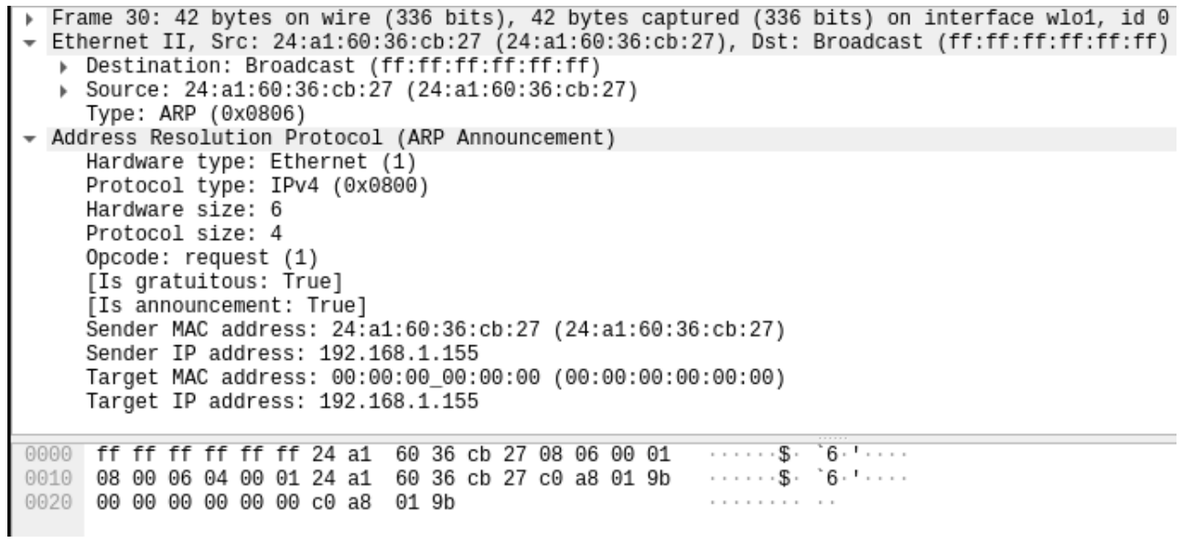

Figure 2: Packet detail in Wireshark

```
sestakp@ubuntu:~/Desktop/FIT/4SEM/IPK/Sniffer$ sudo ./ipk-sniffer -i wlo1 --arp
2021-04-24T15:56:46.161+02:00 24:A1:60:36:CB:27 > FF:FF:FF:FF:FF:FF, length 42 bytes
0x0000: ff ff ff ff ff ff 24 a1  60 36 cb 27 08 06 00 01  ......$. `6.'....
0x0010: 08 00 06 04 00 01 24 a1  60 36 cb 27 c0 a8 01 9b  ......$. `6.'....
0x0020: 00 00 00 00 00 00 c0 a8  01 9b                    ........ ..
```

Figure 3: Packet detail in ipk-sniffer

4.1.2 TCP packet

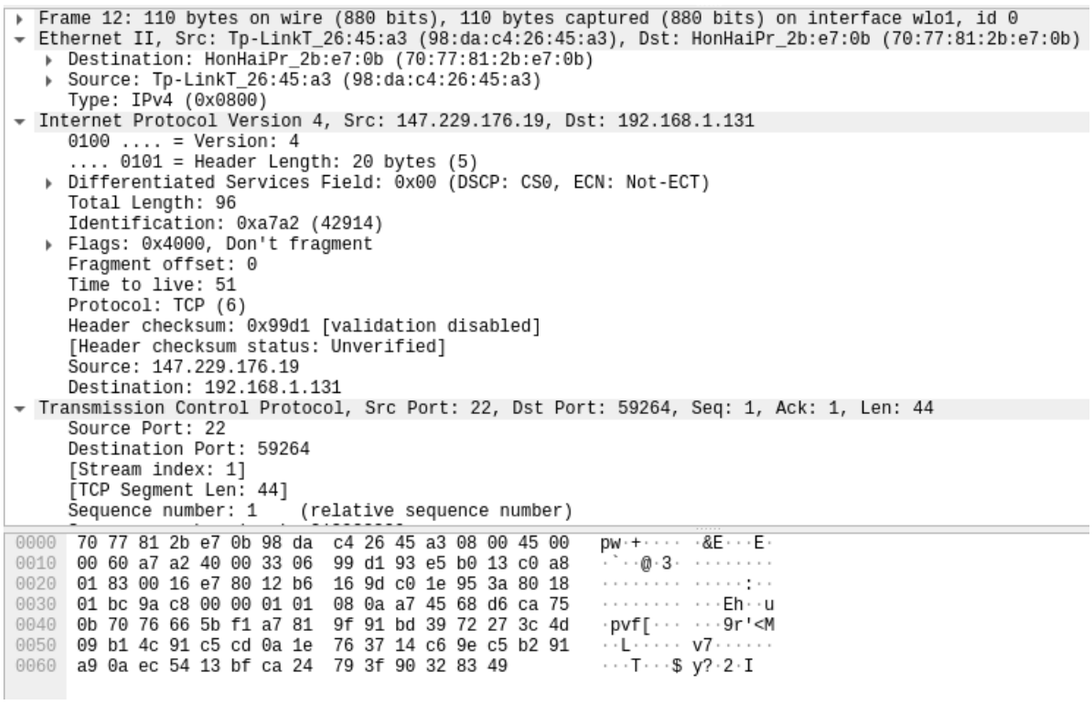

Figure 4: Packet detail in Wireshark

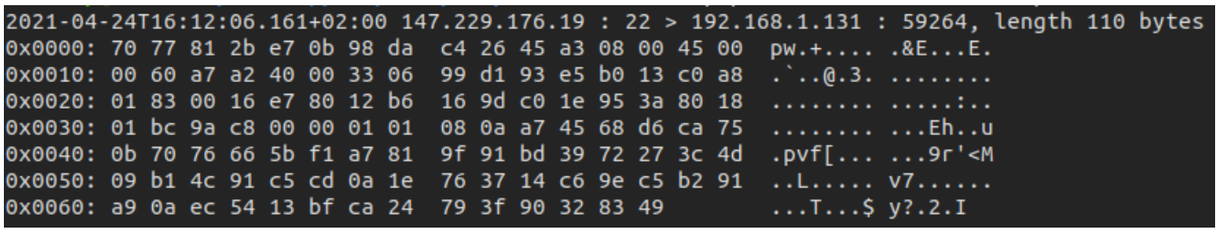

Figure 5: Packet detail in ipk-sniffer

4.1.3 UDP packet

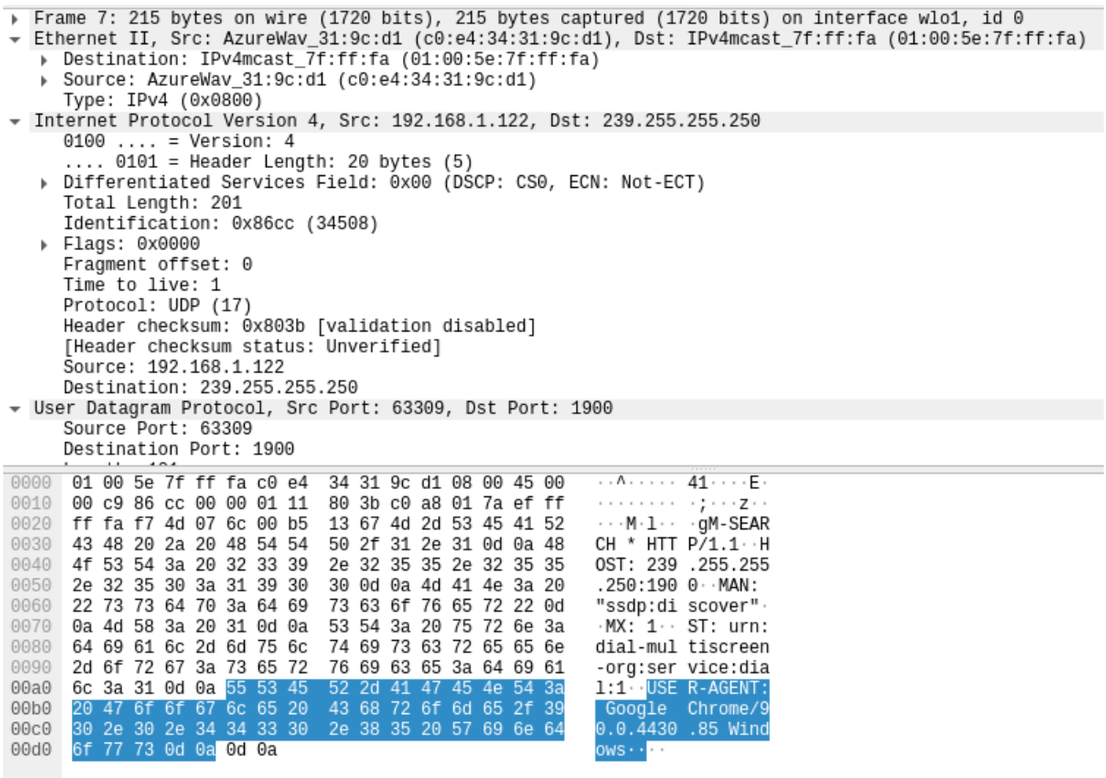

Figure 6: Packet detail in Wireshark

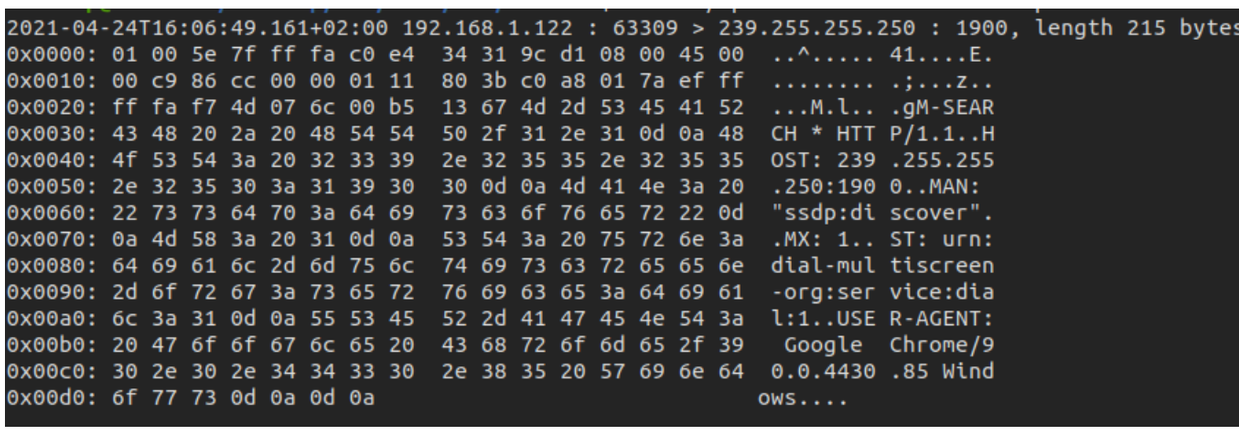

Figure 7: Packet detail in ipk-sniffer

4.1.4 ICMP packet

```
▸ Frame 39: 120 bytes on wire (960 bits), 120 bytes captured (960 bits) on interface wlo1, id 0
▾ Ethernet II, Src: HonHaiPr_2b:e7:0b (70:77:81:2b:e7:0b), Dst: Tp-LinkT_26:45:a3 (98:da:c4:26:45:a3)
  ▸ Destination: Tp-LinkT_26:45:a3 (98:da:c4:26:45:a3)
  ▸ Source: HonHaiPr_2b:e7:0b (70:77:81:2b:e7:0b)
    Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 192.168.1.131, Dst: 192.168.1.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)
    Total Length: 106
    Identification: 0xf8de (63710)
  ▸ Flags: 0x0000
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0xfd1f [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.1.131
    Destination: 192.168.1.1
▸ Internet Control Message Protocol
0000  98 da c4 26 45 a3 70 77  81 2b e7 0b 08 00 45 c0   ···&E·pw ·+····E·
0010  00 6a f8 de 00 00 40 01  fd 1f c0 a8 01 83 c0 a8   ·j····@· ········
0020  01 01 03 03 81 1d 00 00  00 00 45 00 00 4e 00 00   ········ ··E··N··
0030  40 00 40 11 b6 ca c0 a8  01 01 c0 a8 01 83 c0 cc   @·@····· ········
0040  00 89 00 3a 17 2f 64 e7  00 00 00 01 00 00 00 00   ···:·/d· ········
0050  00 00 20 43 4b 41 41 41  41 41 41 41 41 41 41 41   ·· CKAAA AAAAAAAA
0060  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41   AAAAAAAA AAAAAAAA
0070  41 41 41 00 00 21 00 01                           AAA··!··
```

Figure 8: Packet detail in Wireshark

```
sestakp@ubuntu:~/Desktop/FIT/4SEM/IPK/Sniffer$ sudo ./ipk-sniffer -i wlo1 --icmp
2021-04-24T15:59:23.161+02:00 192.168.1.131 > 192.168.1.1, length 120 bytes
0x0000: 98 da c4 26 45 a3 70 77  81 2b e7 0b 08 00 45 c0  ...&E.pw .+....E.
0x0010: 00 6a f8 de 00 00 40 01  fd 1f c0 a8 01 83 c0 a8  .j....@. ........
0x0020: 01 01 03 03 81 1d 00 00  00 00 45 00 00 4e 00 00  ........ ..E..N..
0x0030: 40 00 40 11 b6 ca c0 a8  01 01 c0 a8 01 83 c0 cc  @.@..... ........
0x0040: 00 89 00 3a 17 2f 64 e7  00 00 00 01 00 00 00 00  ...:./d. ........
0x0050: 00 00 20 43 4b 41 41 41  41 41 41 41 41 41 41 41  .. CKAAA AAAAAAAA
0x0060: 41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41  AAAAAAAA AAAAAAAA
0x0070: 41 41 41 00 00 21 00 01                          AAA..!..
```

Figure 9: Packet detail in ipk-sniffer

4.2 Link layer type: linux cooked capture

These tests were performed at the interface any.

4.2.1 ARP packet

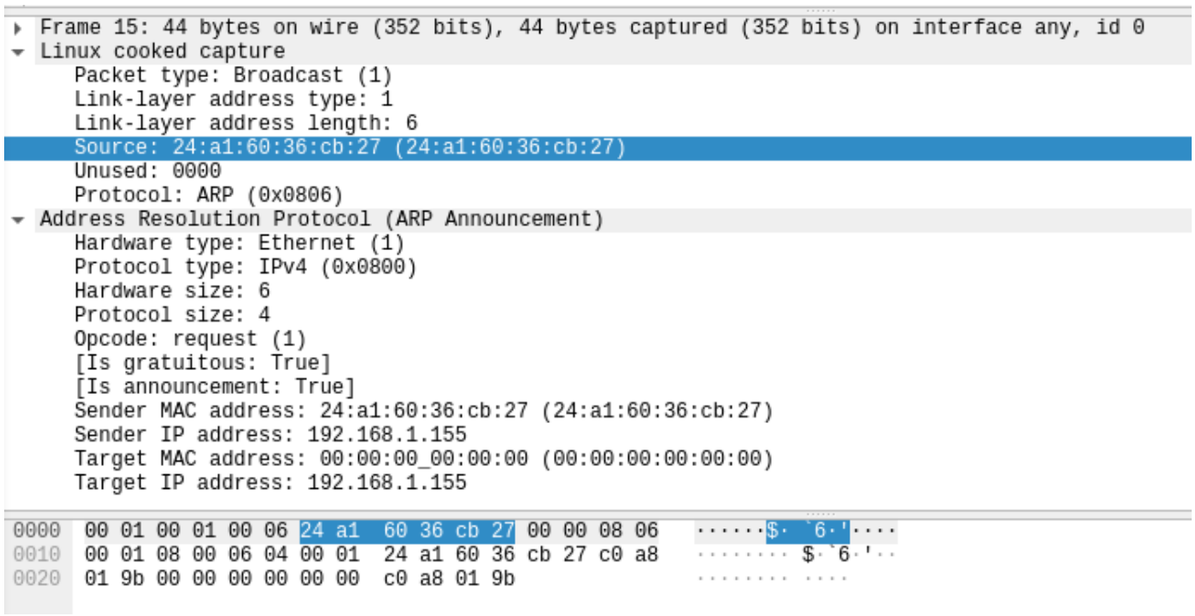

Figure 10: Packet detail in Wireshark

```
2021-04-24T16:29:47.161+02:00 24:A1:60:36:CB:27 > 00:00:00:00:00:00, length 44 bytes
0x0000: 00 01 00 01 00 06 24 a1  60 36 cb 27 00 00 08 06  ......$. `6.'....
0x0010: 00 01 08 00 06 04 00 01  24 a1 60 36 cb 27 c0 a8  ........ $.`6.'..
0x0020: 01 9b 00 00 00 00 00 00  c0 a8 01 9b              ........ ....
```

Figure 11: Packet detail in ipk-sniffer

4.2.2 TCP packet

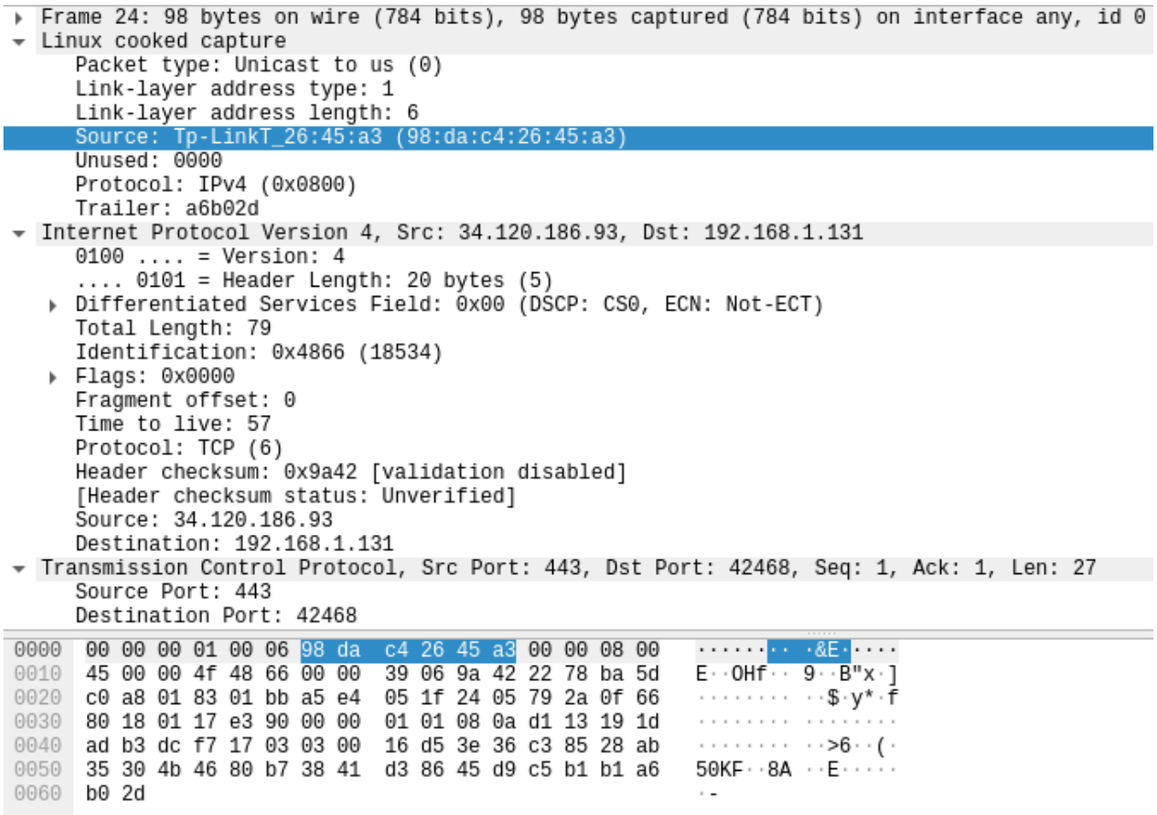

Figure 12: Packet detail in Wireshark

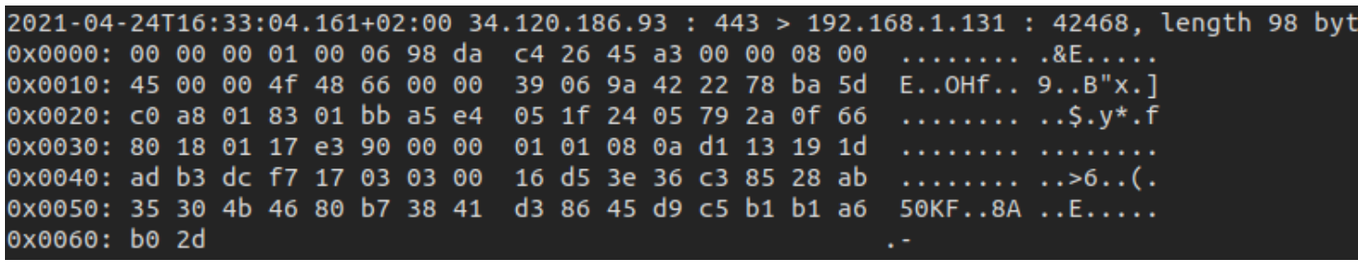

Figure 13: Packet detail in ipk-sniffer

4.2.3 UDP packet

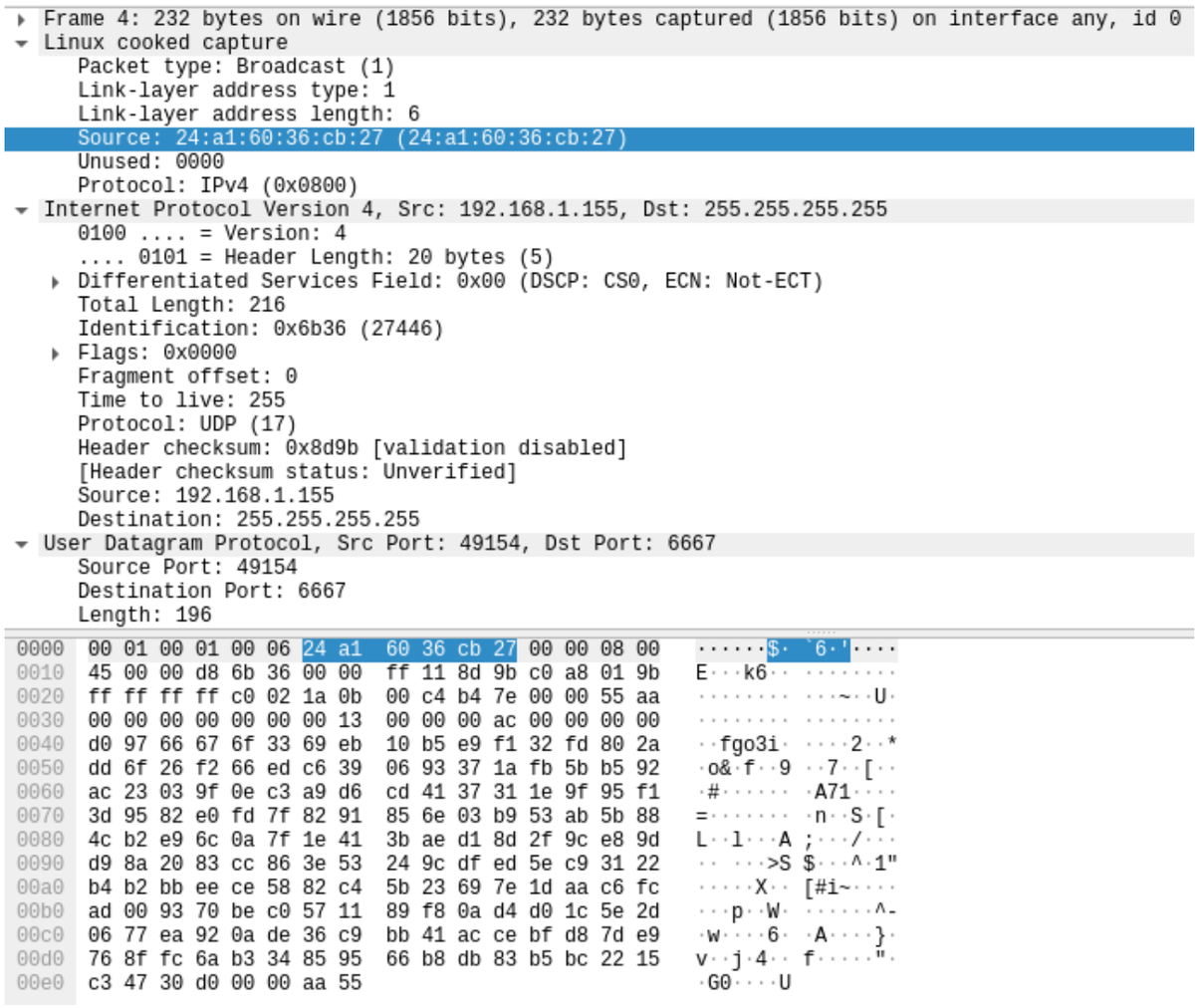

Figure 14: Packet detail in Wireshark

```
2021-04-24T16:34:35.161+02:00 192.168.1.155 : 49154 > 255.255.255.255 : 6667, length 232 bytes
0x0000: 00 01 00 01 00 06 24 a1  60 36 cb 27 00 00 08 00  ......$. `6.'....
0x0010: 45 00 00 d8 6b 36 00 00  ff 11 8d 9b c0 a8 01 9b  E...k6.. ........
0x0020: ff ff ff ff c0 02 1a 0b  00 c4 b4 7e 00 00 55 aa  ........ ...~..U.
0x0030: 00 00 00 00 00 00 00 13  00 00 00 ac 00 00 00 00  ........ ........
0x0040: d0 97 66 67 6f 33 69 eb  10 b5 e9 f1 32 fd 80 2a  ..fgo3i. ....2..*
0x0050: dd 6f 26 f2 66 ed c6 39  06 93 37 1a fb 5b b5 92  .o&.f..9 ..7..[..
0x0060: ac 23 03 9f 0e c3 a9 d6  cd 41 37 31 1e 9f 95 f1  .#...... .A71....
0x0070: 3d 95 82 e0 fd 7f 82 91  85 6e 03 b9 53 ab 5b 88  =....... .n..S.[.
0x0080: 4c b2 e9 6c 0a 7f 1e 41  3b ae d1 8d 2f 9c e8 9d  L..l...A ;.../...
0x0090: d9 8a 20 83 cc 86 3e 53  24 9c df ed 5e c9 31 22  .. ...>S $...^.1"
0x00a0: b4 b2 bb ee ce 58 82 c4  5b 23 69 7e 1d aa c6 fc  .....X.. [#i~....
0x00b0: ad 00 93 70 be c0 57 11  89 f8 0a d4 d0 1c 5e 2d  ...p..W. ......^-
0x00c0: 06 77 ea 92 0a de 36 c9  bb 41 ac ce bf d8 7d e9  .w....6. .A....}.
0x00d0: 76 8f fc 6a b3 34 85 95  66 b8 db 83 b5 bc 22 15  v..j.4.. f.....".
0x00e0: c3 47 30 d0 00 00 aa 55                           .G0....U
```

Figure 15: Packet detail in ipk-sniffer

4.2.4 ICMP packet

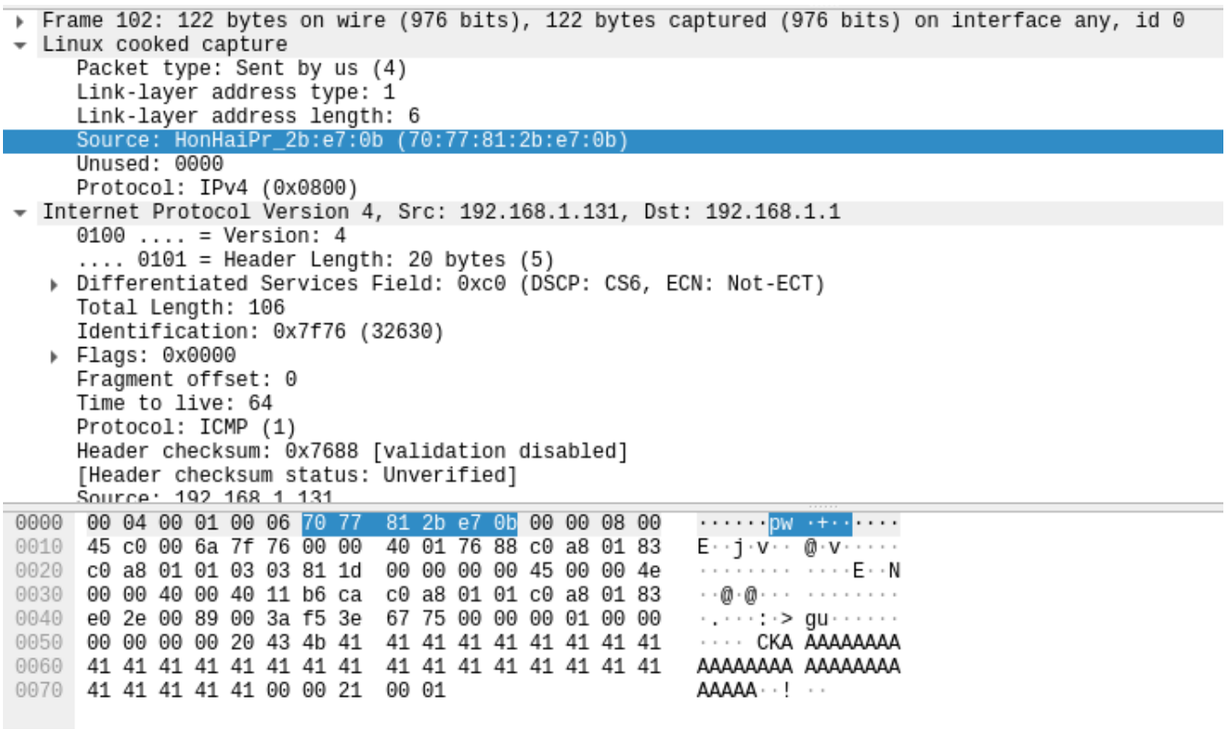

Figure 16: Packet detail in Wireshark

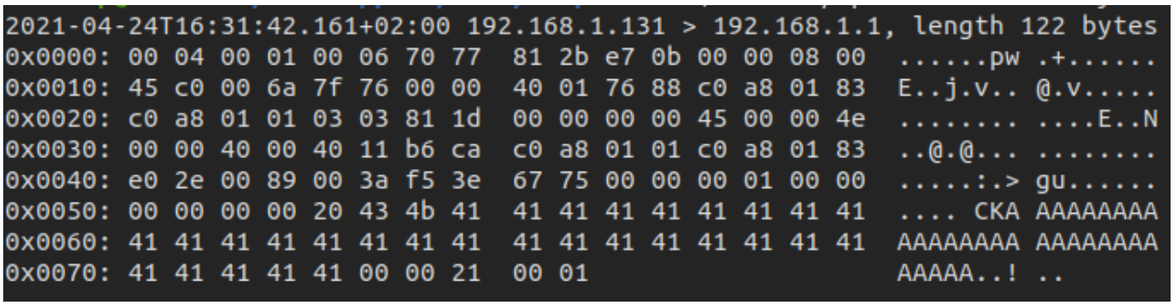

Figure 17: Packet detail in ipk-sniffer

5 Conclusion

Basic features are successfully implemented but there is lot of things to improve. List of packet information can be enritched, graphic frontend can be added for easier work with sniffer.