Brno University of Technology
Faculty of Information Technology

Computer Communications and Networks

2021/2022

Project documentation

Variant ZETA: Packet Sniffer

Alina Vinogradova (xvinog00)

Brno, April 24, 2022

Table of contents

Introduction

Task description

The problem of the project was to develop an application - a network analyzer that can receive and filter packets (depending on the protocol: TCP, UDP, ICMP, ARP) on a particular device. Output data are written to the standard output (stdout) in Hex Dump format (analog is pcapng file, which is used in Wireshark). Before that, the display shows the basic values for the current packet: Packet timestamp, source and destination MAC addresses, frame length, source and destination IP addresses, source and destination ports.

Technical requirements

The program should be called as follows:

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

Where:

- `-h`: will write a help message.
- `-i` *or* `--interface interface`: just one interface on which to listen. If this parameter is not specified, or if only `-i` is specified without a value, a list of active interfaces will be printed.
- `-p port`: will filter packets on a given interface by port; if this parameter is not specified, all ports are considered; if the parameter is specified, the port can occur in both the source and destination part.
- `-t` *or* `--tcp`: will display only TCP packets.
- `-u` *or* `--udp`: will display only UDP packets.
- `--icmp`: will display only ICMPv4 and ICMPv6 packets.
- `--arp`: will display only ARP frames.
- `-n num`: specifies the number of packets to be displayed; if not specified, consider displaying only one packet (as if `-n 1`).

Also:

- Unless specific protocols are specified (or if all of them are listed at once), all protocols (i.e. all content, regardless of the protocol) are considered for printing.
- The program can be correctly terminated at any time using Ctrl+C.

Implementation

How the application works

The core of the application is the library **`pcap`**. It is a portable platform for low-level network monitoring that uses the standard pcap format. The libpcap interface supports a filtering mechanism called BPF [1], which was used in the current project. It provides a raw interface to data link layers, permitting raw link-layer packets to be sent and received. Also to work with certain protocols (work with data structures) we used headers from the **`netinet`** library.

Compile

In the root folder along with the source code there is a Makefile, which allows you to compile the program simply from command line:
`$ make`
Output executable file is **`./ipk-sniffer`** (***require root to run properly!***). Recommended usage:
`$ sudo ./ipk-sniffer ...`

Program start

The main function of the program processes the arguments, "assembles" the filter based on the prepared arguments and starts the sniffer function.
First, the **`parameters_parsing()`** function is called, divided into two parts: processing of short arguments (**`"-"`**) and processing of long arguments (**`"--"`**). After that, all arguments are checked in the same function to make sure they fit the conditions of the application.
Next, the **`filter_ctor()`** function is called, which translates the user's arguments into the appropriate **`pcap_filter`** format.
The **`pcap_t *handle`** variable is the descriptor of the packet capture endpoint. Then it is passed for processing to the **`handling_pcap()`** function, which consists of several parts:

- The variable **`device`** (which stores the values of the user parameter **`interface`**) is checked and, if it is empty, a list of all available devices is displayed.
- If the user has set the interface value, the program tries to open it with **`pcap_open_live()`**
- **`pcap_lookupnet()`** returns the subnet mask and the address of the network where packets will be listened to. The mask is further used to set the filter.
- **`pcap_compile()`** and **`pcap_setfliter()`** compile and install the program filter.

`pcap_loop()`

If the previous processing went without errors, it means that all the input data were prepared to start listening to the packets. **`pcap_loop()`** is one of three existing and one of two automatically working functions to get packets. It takes the `handle` descriptor, the number of packets we want to handle and

the packet capture call back function as parameters. [2] For each packet received, **`pcap_loop()`** calls the handler – in my implementation this is the **`got_packet()`** function.

Callback function `got_packet()`

This function is one of the most important parts of the program, along with **`handling_pcap()`**, where we "built" the network adapter, and **`pcap_loop()`**, which automatically reads packets from the adapter. It is used to collect and construct all the necessary data about the current packet, such as:

- Current timestamp
- MAC addresses for source and destination
- Length of current data frame
- IP addresses for source and destination
- Source and destination ports
- The data of the package itself

Variable declaration

For our work we need data structures from the **`netinet`** library:

- **`<netinet/udp.h>`**
- **`<netinet/tcp.h>`**
- **`<netinet/ip.h>`**
- **`<netinet/ip6.h>`**
- **`<netinet/if_ether.h>`**

`struct ip* iphdr` and **`struct ip6_hdr* ip6hdr`** are used to process IP packets. **`struct tcphdr* tcphdr`** and **`struct udphdr* udphdr`** are used to collect packet ports depending on the selected protocol.
Depending on the packet type (**`ARP, IPv4/6`**) we get the current protocol used in the packet (only in the case of **`IPv4/6`**) and the source and destination IP addresses. We can obtain current packet type by declaring a structure:

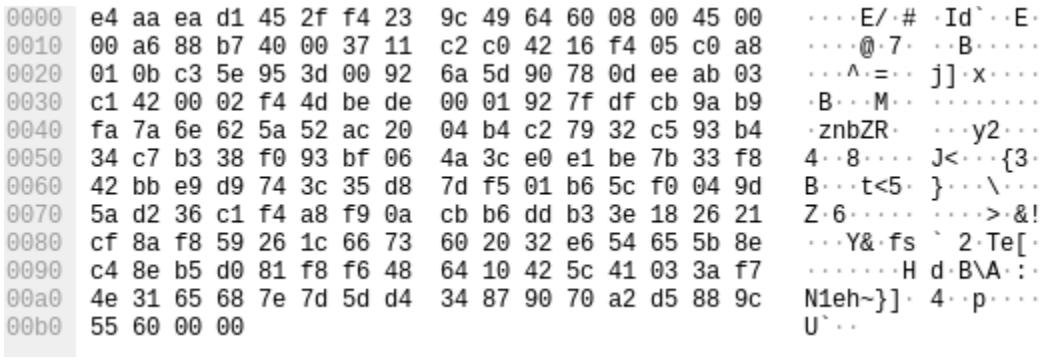
```
struct ether_header *etherhdr = (struct ether_header *)packet;
uint16_t ether_type = ntohs(etherhdr->ether_type);
```

Therefore **`ether_type`** could have 3 values (which can be recognized by application): **`ETHERTYPE_ARP, ETHERTYPE_IP, ETHERTYPE_IPV6.`** Also we can retrieve source and destination MAC addresses by sprinting 6 values from **`etherhdr->ether_shost[6]`** and **`etherhdr->ether_dhost[6]`** each. Then, having a protocol, we are able to proceed to select the protocol according to what we obtained above. I've already defined required values, where **`PROTOCOL_TCP = 6`** and **`PROTOCOL_UDP = 17`**. [3] Use switch(protocol) to select the actual protocol a packet is using. Now we can get UDP and TCP ports:
`sourcePort = ntohs(tcphdr->th_sport)` OR **`ntohs(udphdr->uh_sport)`**

And the same for **`destinationPort`**.

Timestamp constructing and displaying data

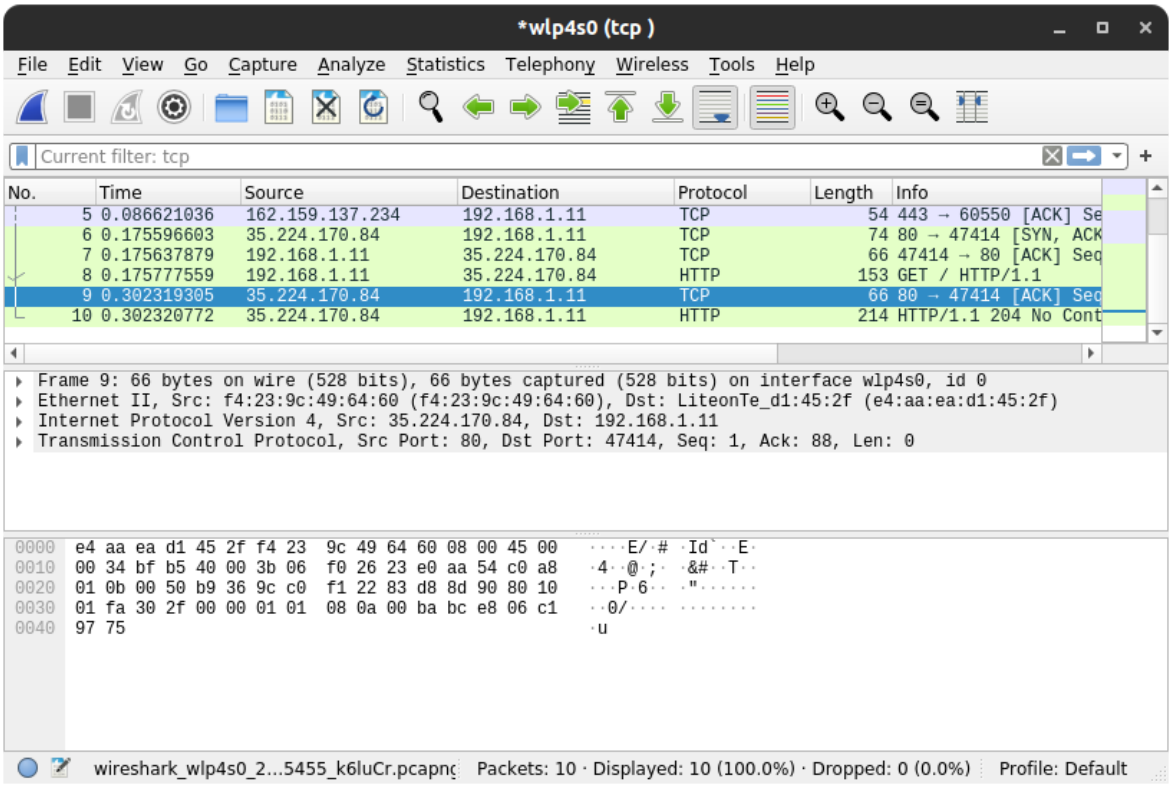
At the beginning of the function, a **`char *timestamp`** variable is created, which, with the argument **`struct pcap_pkthdr header->ts`** gets its value from the **`timestamp_ctor()`** function. The **`timeval`** structure we get as an argument carries the value **`tv_sec`**, which we can translate into milliseconds. [4] Returned timestamp is in *RFC3339* format.
After processing the packet completely (if it passed without errors), we call the **`display_packet_data()`** function and pass in it all the necessary parameters that are required for basic information about the packet. Then **`display_packet_dump()`** is called to display the Hex Dump of a packet, using **`const u_char* packet`** and **`const int len`** as parameters.



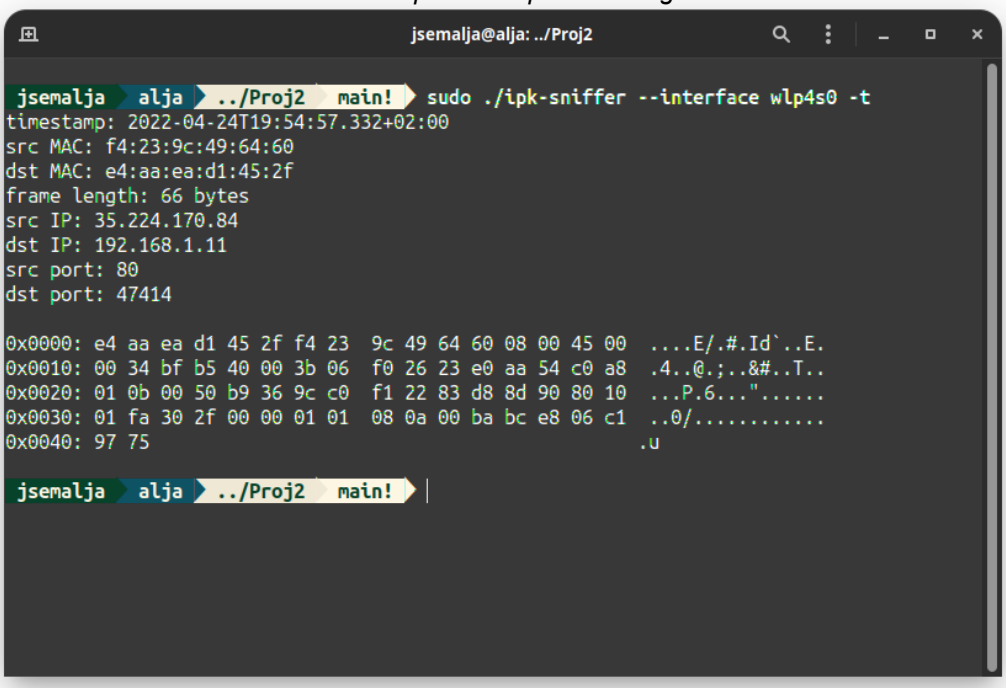
Pic: example of HexDump data

Sample outputs

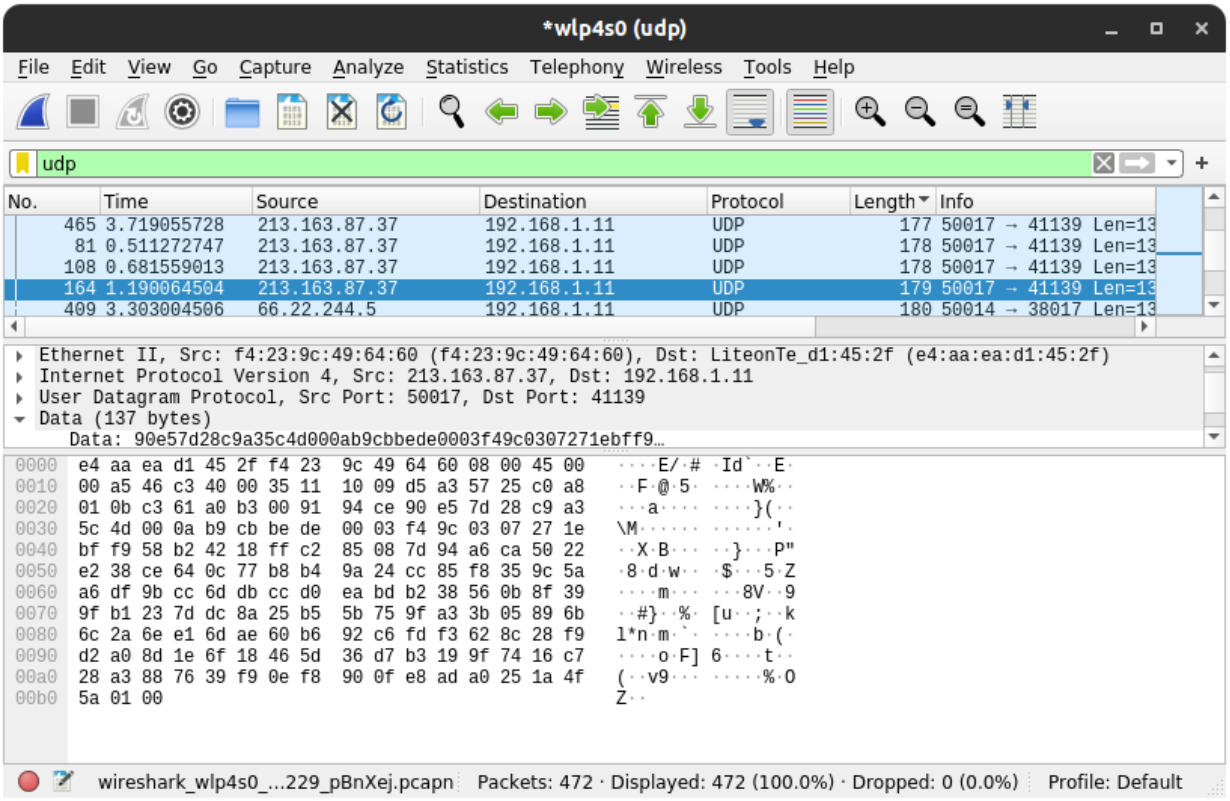
Below there are simple outputs examples using Wireshark to compare received packets.



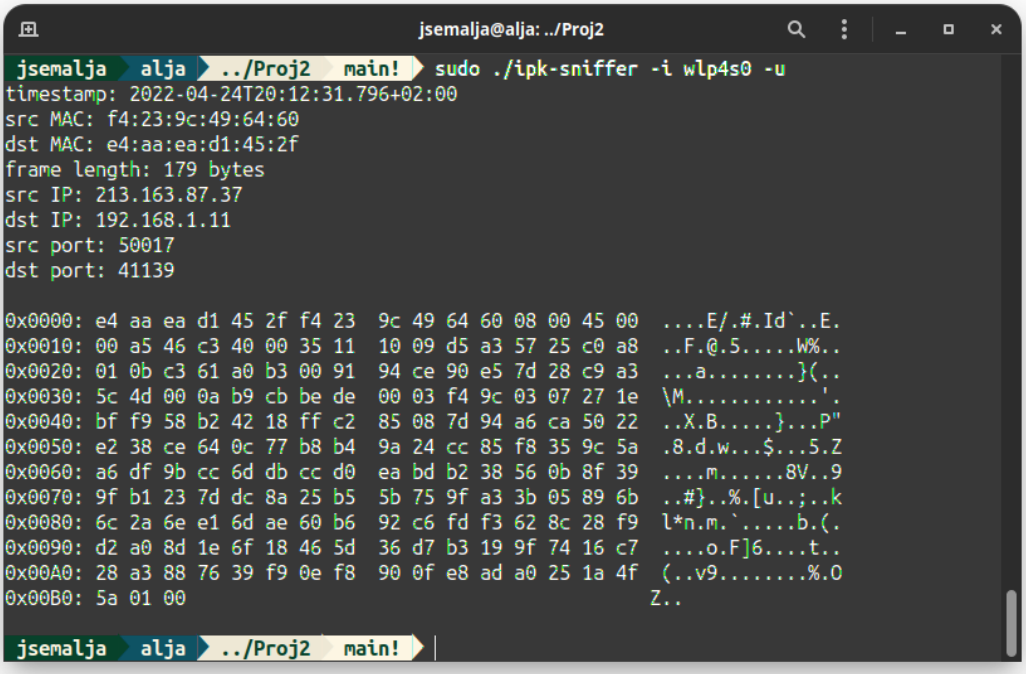
Pic: one TCP packet captured using Wireshark



Pic: the same TCP packet captured using application



Pic: one UDP packet captured using Wireshark



Pic: the same UDP packet captured using application

Testing

For tests I used **nc**, **ping**, **ping6.**

Summary

To test using **netcat** I needed two command line sessions: first one is for listening and second one is for sending some data packets. [5] In tests cases #1-4 for examination **netcat** was used with parameters such as:

```
$ nc -6 [-l | localhost] 8080
```

Where command with the argument **-l** is the “server” and the other one with **localhost** is a “client”. The task of the application will be to "intercept" packets between this server and the client. The same rules apply to packets with the UDP protocol.

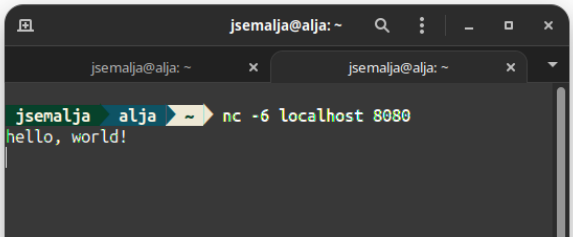
```
$ nc -u -6 [-l | localhost] 8080
```

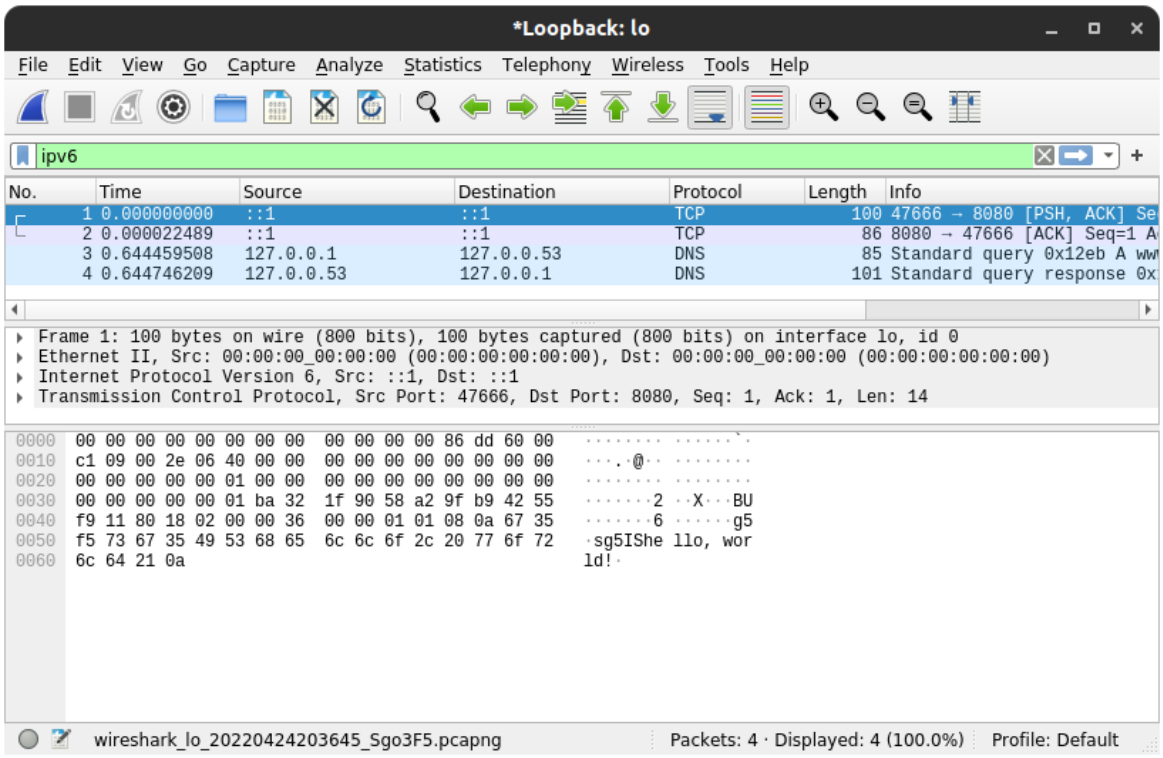
Second function I used for testing is **ping** (and **ping6**). **ping** sending an **ICMP ECHO_REQUEST** to network hosts. [6]

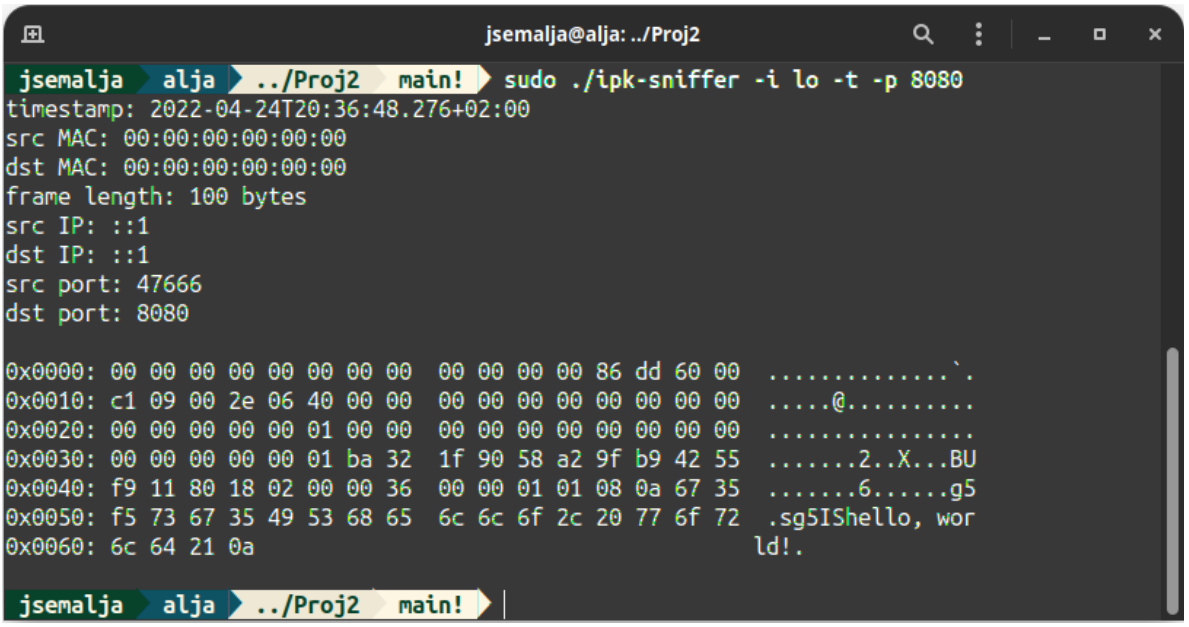
`netcat` testing

Test case #1: tcp packet with port filter using IPv6

Connection is OK at port 8080.
Wireshark also can see this packet.



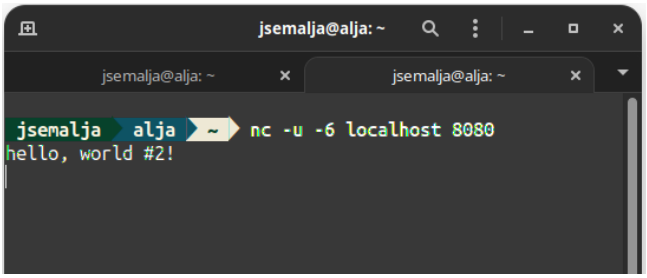


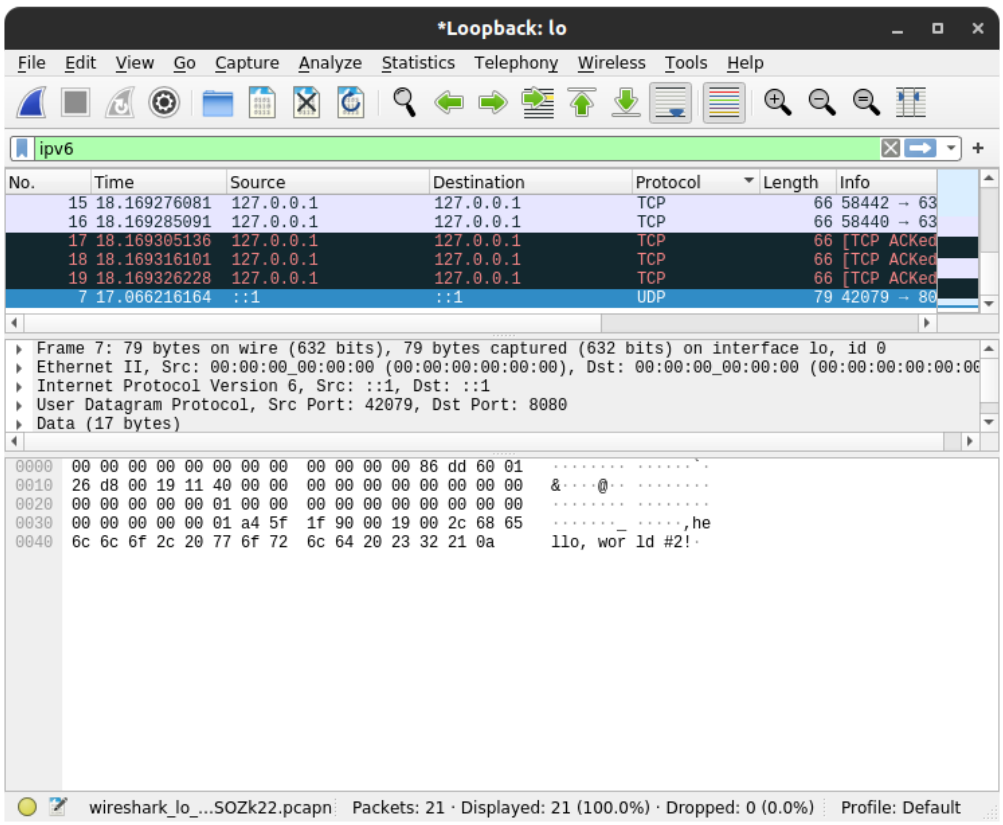


Application as well shows the received packet.

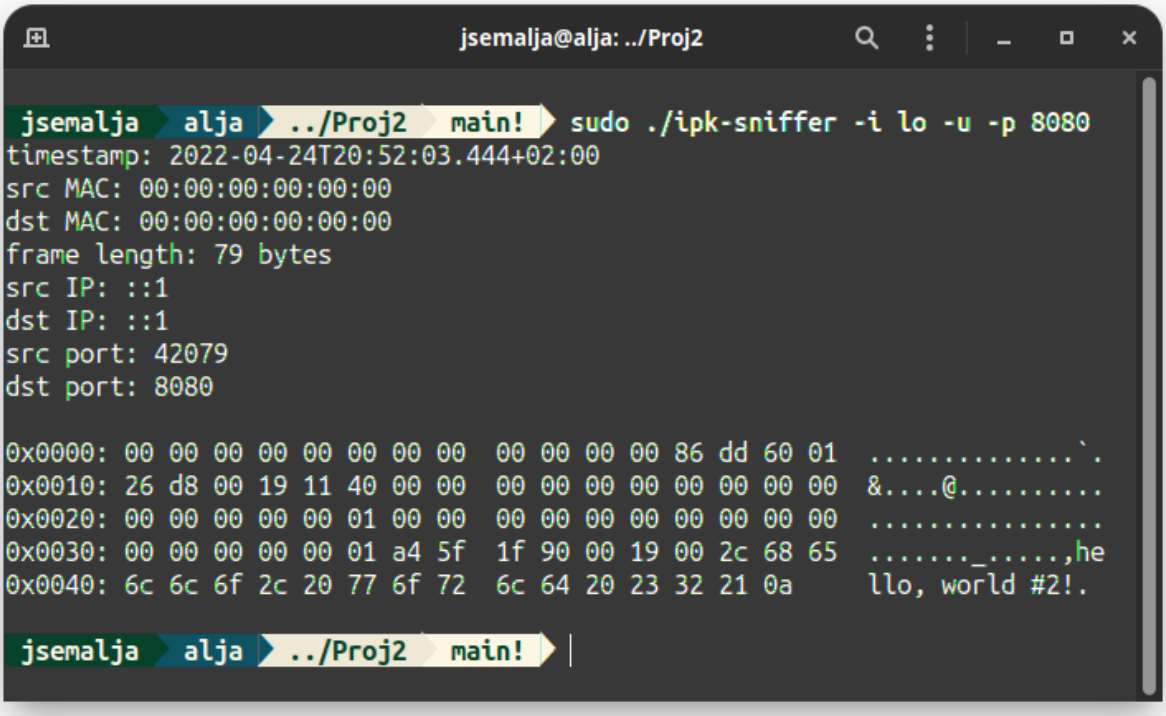
TEST CASE #1 IS OK

Test case #2: udp packet with port filter using IPv6





Connection is OK, packet transmission is OK.

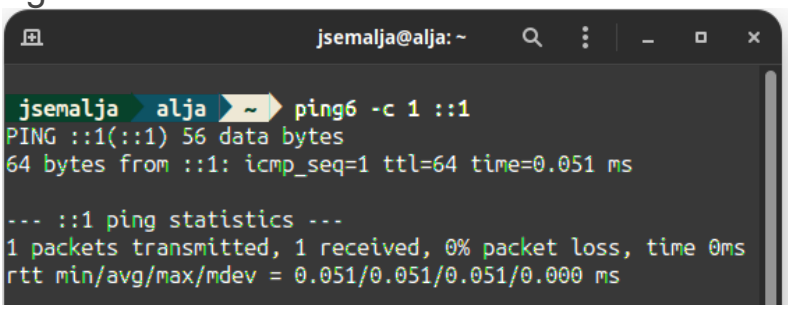


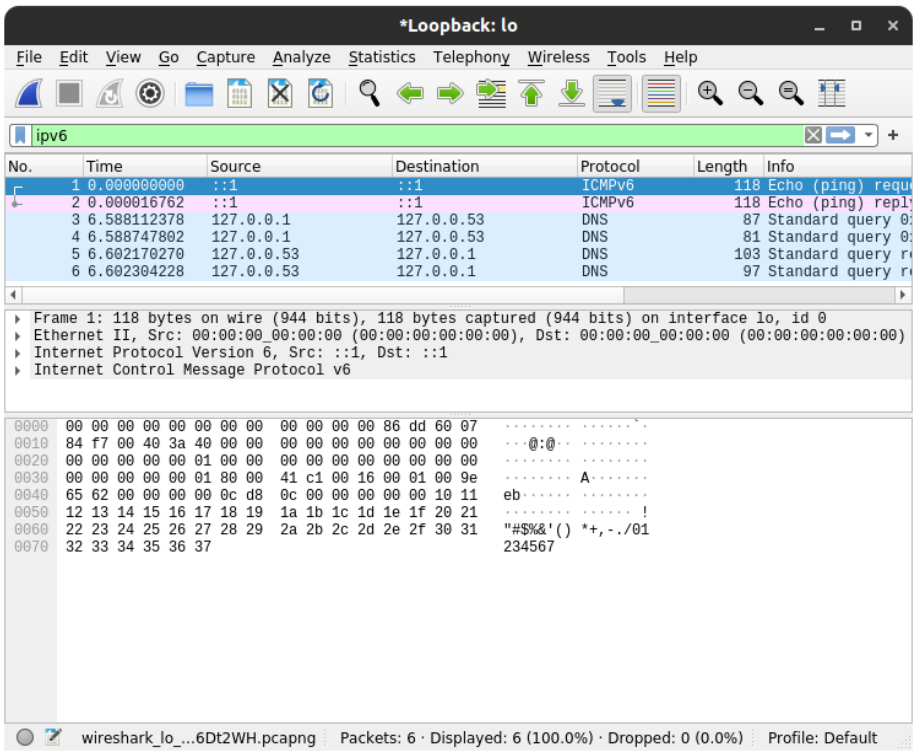
Application received a packet.

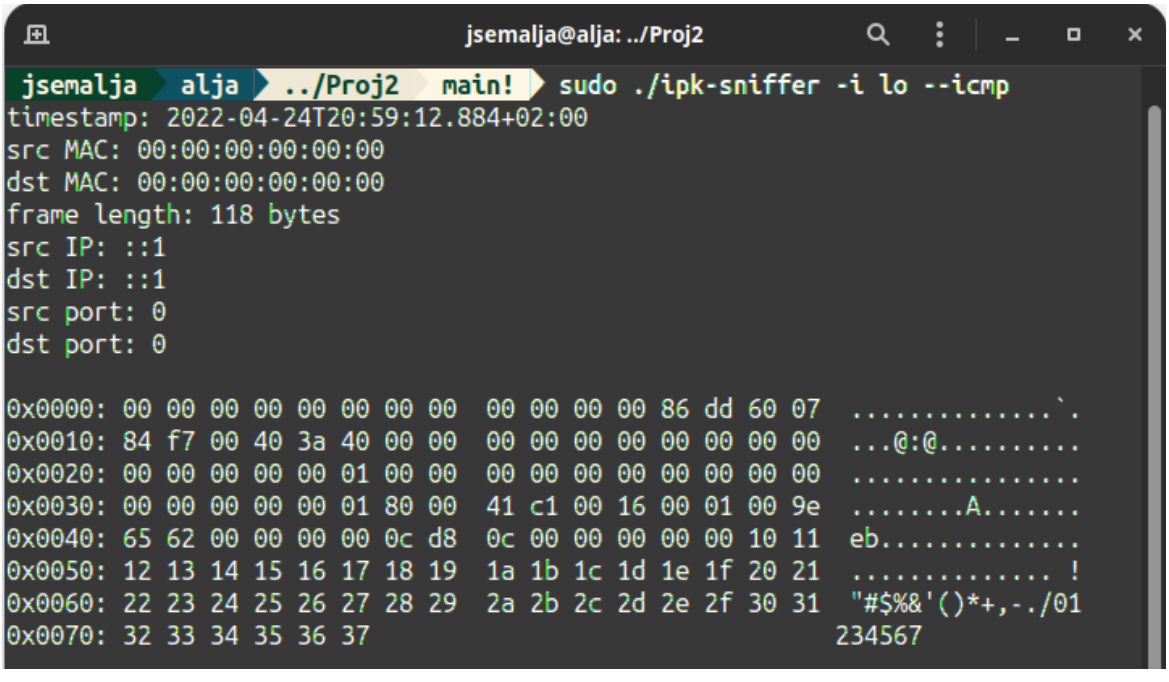
TEST CASE #2 IS OK

`ping` testing

Test case #3: ICMPv6 packet receiving



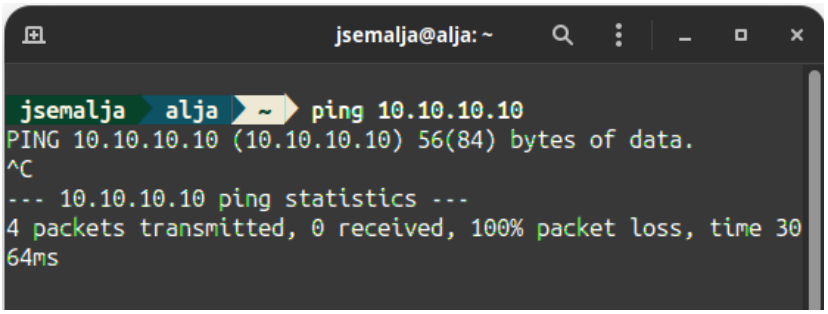


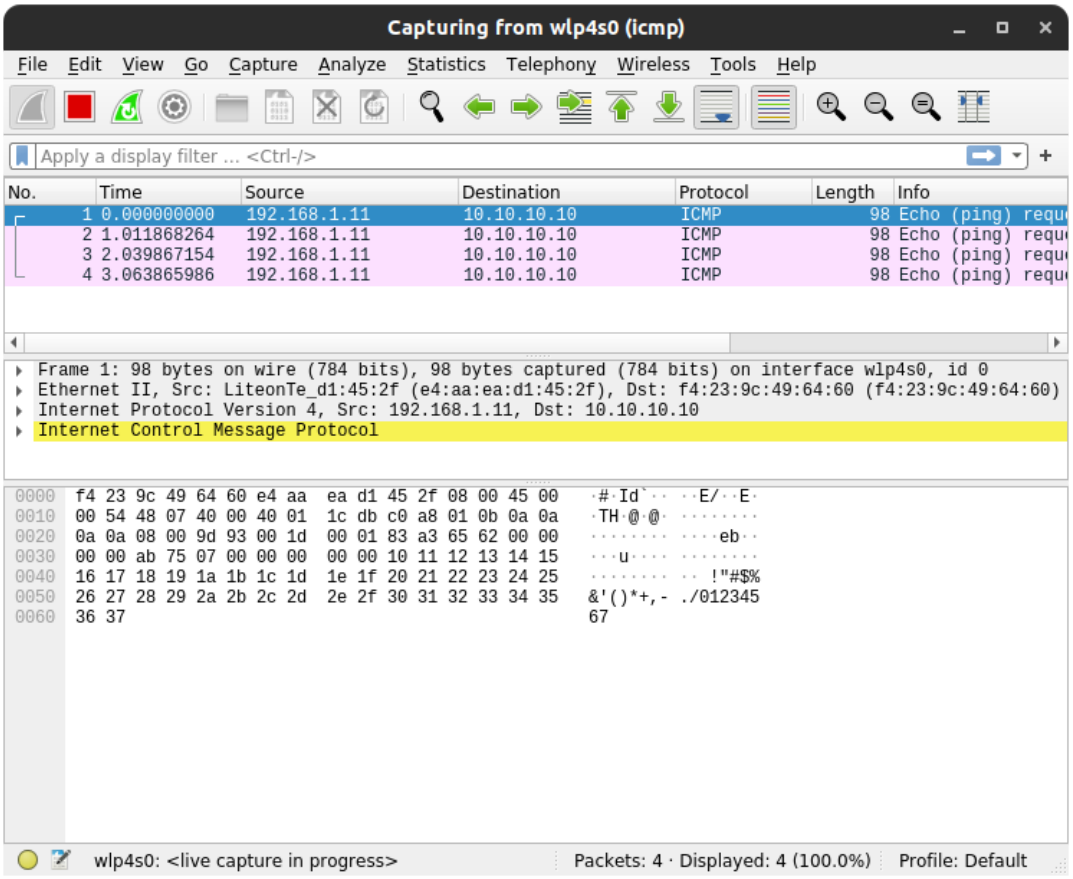


As well as wireshark, application also can see and receive packet and its data.

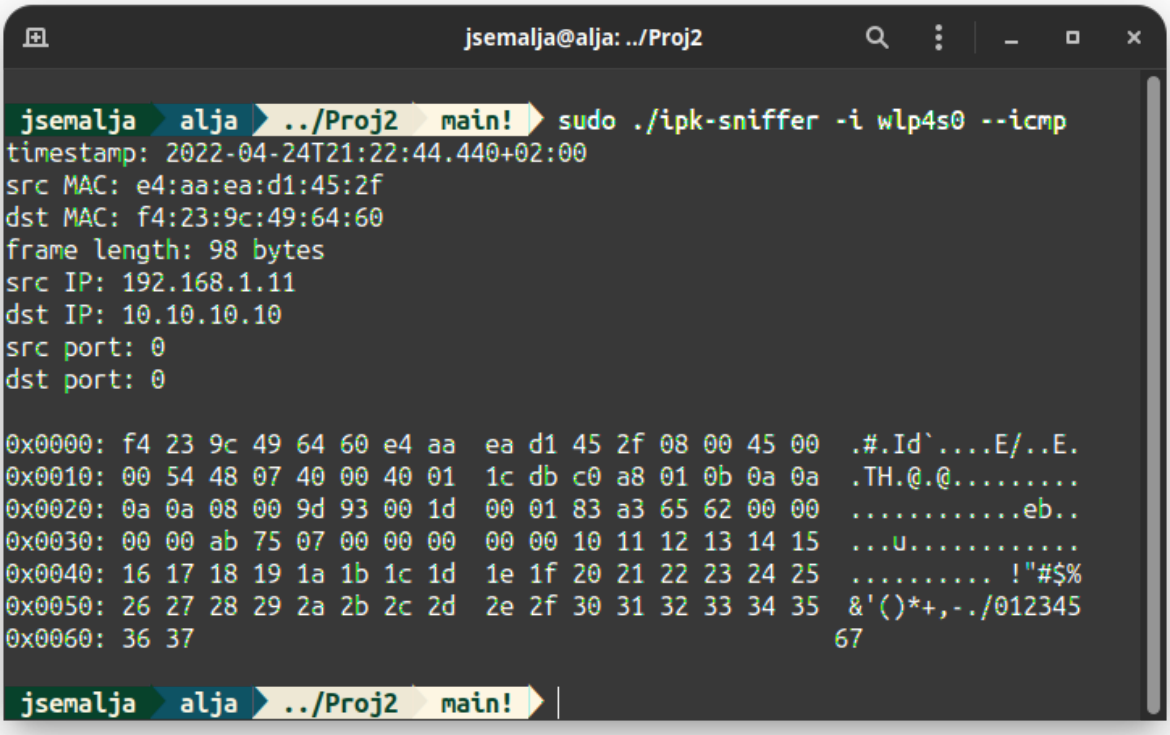
TEST CASE #3 IS OK

Test case #4: ICMPv4 packet receiving





As we can see in Wireshark, data from ping was successfully transmitted.



The same packet received by application.

TEST CASE #4 IS OK

List of sources

1. Berkeley Packet Filter - *Wikipedia* [online]. Available from: Berkeley Packet Filter - Wikipedia
2. *Home | TCPDUMP & LIBPCAP* [online]. Copyright © 2010 [cit. 24.04.2022]. Available from: pcap_loop(3PCAP) man page | TCPDUMP & LIBPCAP
3. *Internet Assigned Numbers Authority* [online]. Available from: Protocol Numbers
4. <sys/time.h>. *The Open Group Publications Catalog* [online]. Copyright © 1997 The Open Group [cit. 24.04.2022]. Available from: <sys/time.h>
5. *Ubidots Help Center* [online]. Available from: Sending TCP/UDP packets using Netcat | Ubidots Help Center
6. ping6(8) - Linux man page. *Linux Documentation* [online]. Available from: ping6(8) - Linux man page
7. EtherType - *Wikipedia* [online]. Available from: EtherType - Wikipedia
8. Develop a Packet Sniffer with Libpcap - vichargrave.github.io. *vichargrave.github.io* [online]. Copyright © 2022 Vic Hargrave. Powered by [cit. 24.04.2022]. Available from: Develop a Packet Sniffer with Libpcap
9. Programming with pcap | TCPDUMP & LIBPCAP. *Home | TCPDUMP & LIBPCAP* [online]. Copyright © 2010 [cit. 24.04.2022]. Available from: Programming with pcap | TCPDUMP & LIBPCAP
10. C - pcap_findalldevs to display all interfaces is stuck in an infinite loop - Stack Overflow. *Stack Overflow - Where Developers Learn, Share, & Build Careers* [online]. Available from: C - pcap_findalldevs to display all interfaces is stuck in an infinite loop - Stack Overflow
11. c - how to get hexdump of a structure data - Stack Overflow. *Stack Overflow - Where Developers Learn, Share, & Build Careers* [online]. Available from: how to get hexdump of a structure data - Stack Overflow
12. time - I'm trying to build an RFC3339 timestamp in C. How do I get the timezone offset? - Stack Overflow. *Stack Overflow - Where Developers Learn, Share, & Build Careers* [online]. Available from: I'm trying to build an RFC3339 timestamp in C. How do I get the timezone offset? - Stack Overflow