

Brno University of Technology
Faculty of Information Technology



Computer Communications and Networks

2021/2022

Project documentation

Variant ZETA: Packet Sniffer

Contents

1.	Introduction	3
2.	Usage	3
3.	Implementation	4
4.	Testing	7
	4.1. Transmission Control Protocol	7
	4.2. User Datagram Protocol	9
	4.3. Internet Control Message Protocol version 6	10
	4.4. Address Resolution Protocol	12
5.	References	14

Introduction

The main goal of the project was to create a program capable of intercepting packets on the network and printing them in a special format.

A packet sniffer — also known as a packet analyzer, protocol analyzer or network analyzer — is a piece of hardware or software used to monitor network traffic. Sniffers work by examining streams of data packets that flow between computers on a network as well as between networked computers and the larger Internet.

Usage

Build the program with:

make

Run the program:

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

The program needs root access to the interface to be capable of reading the packets.

The arguments can be in any order.

- interface says what interface should the program use for sniffing. If the argument *-i* is omitted, then the program prints out the available interfaces.
- port filters the packets depending on the given port number. If not specified, all ports are considered.

- num states how many packets we wish to be printed out. If not specified, consider displaying only one packet.
- if *-t* or *--tcp* outputs only TCP packets.
- if *-u* or *--udp* outputs only UDP packets.
- if *--icmp* outputs only ICMPv4 and ICMPv6 packets
- if *--arp* outputs only ARP frames

If specific protocols are not specified, they are all considered for printing.

Example:

```
sudo ./ipk-sniffer -i eth0 -p 23 --tcp -n 2
```

```
sudo ./ipk-sniffer -i eth0 --udp
```

```
sudo ./ipk-sniffer -i eth0 -n 10
```

```
sudo ./ipk-sniffer -i
```

Implementation

The sniffer is written in C language with additional libraries like *pcap.h* and others.

Two structures have been developed. *Packet* is used to store information about packets. Structure *Arguments* is used to correctly process options entered by the user.

First, the *handle_args* function takes command line arguments and fills the structure with the *arguments*. For arguments with optional parameters, I used the macro `OPTIONAL_ARGUMENT_IS_PRESENT`, which I was inspired by on this [page](#). This makes the code more readable and helps save an optional argument parameter, if present.

Next, it was necessary to get a list with all available interfaces for further connection. The *pcap_findalldevs* function helped me with this. After that we need to get interface source IP and netmask using *pcap_lookupnet*. Otherwise, an error will be printed and the program will exit with number 1.

The *pcap_open_live* function will then open the interface for sniffing and *handle_protocols* will start processing the protocols entered by the user, if requested. Next, the *pcap_compile* compiles protocols into a filter program. And already *pcap_setfilter* sets the necessary protocol filters.

After that, the program starts sniffing the requested interface thanks to the *pcap_loop* function. And the *got_packet* is a call back function that parses each captured packet. It initializes the *Packet* structure, defines the protocol and calls the appropriate function to set the header data.

handle_arp - Stores the source and destination IP address (Address Resolution Protocol).

Functions *handle_ip* and *handle_ipv6* work with Internet Protocol and Internet Protocol version 6, respectively. The *inet_ntop* function helps process addresses and store them.

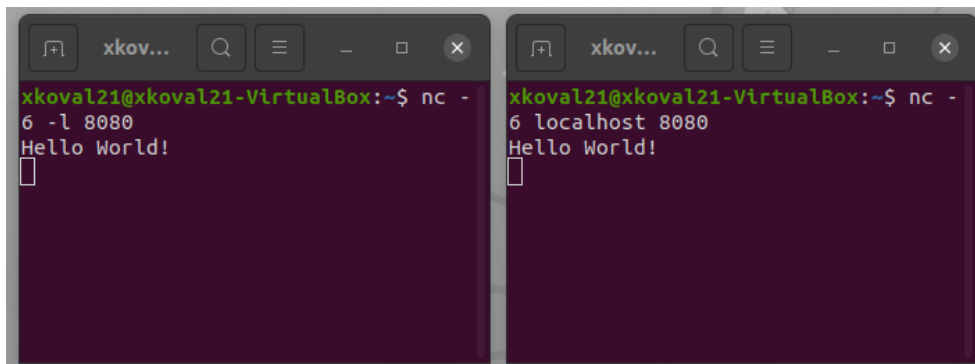
After the header data is filled in, the program prepares the time in a special format for timestamp and, using the two functions *handle_packet* and *handle_hex_dump*, displays information about the packet. Finally, *pcap_freecode* releases the specified allocated memory, and *pcap_close* closes the files associated with the packet capture channel.

Testing

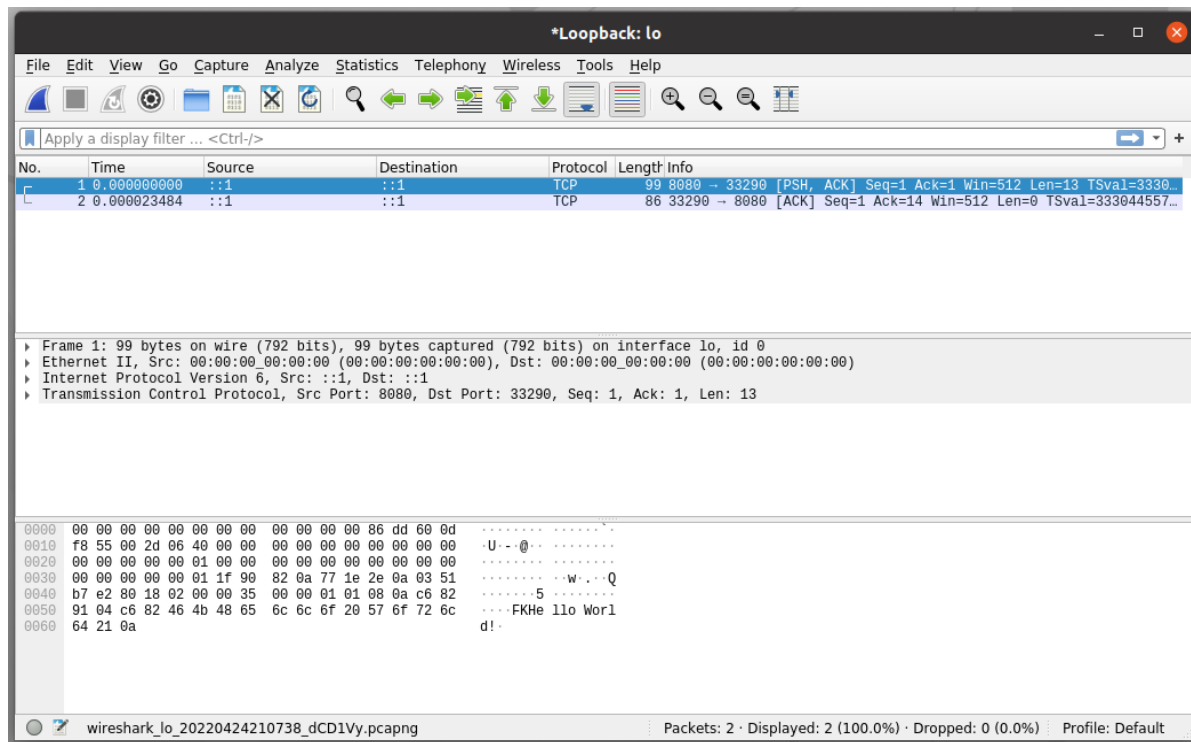
Tested with a VirtualBox virtual machine. On Ubuntu version 22.04 using the Wireshark traffic analyzer.

Transmission Control Protocol

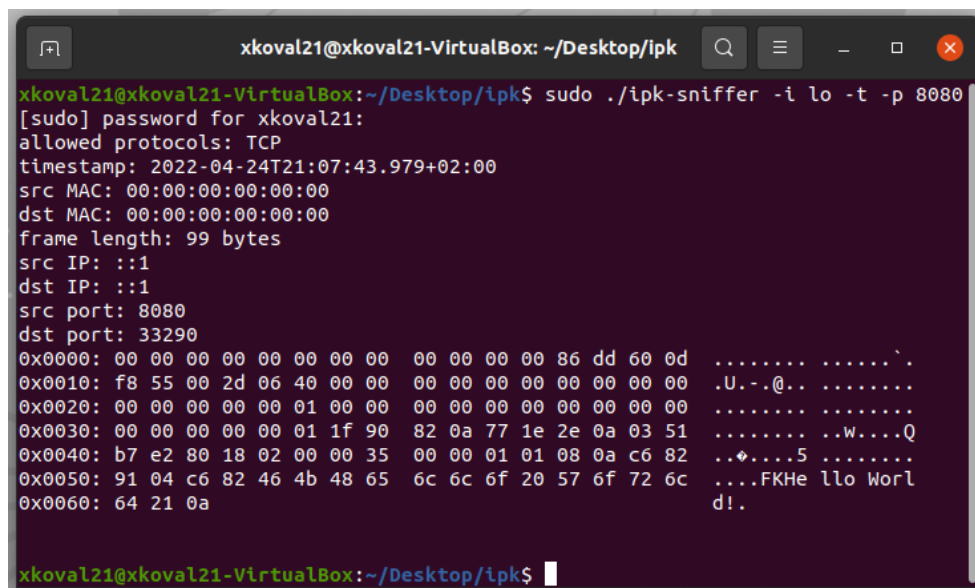
Two terminals were used for testing. Two terminals were used for testing. The first terminal used the `nc -l 8080` command. And in the second terminal `nc -6 localhost 8080`. 8080 is a port, so we can check that the `-p` operator works correctly. Now we send the text to the terminal and it is displayed in another terminal. This means that the package has been sent.



In Wireshark, through the lo interface, we can see two packets that have been sent:

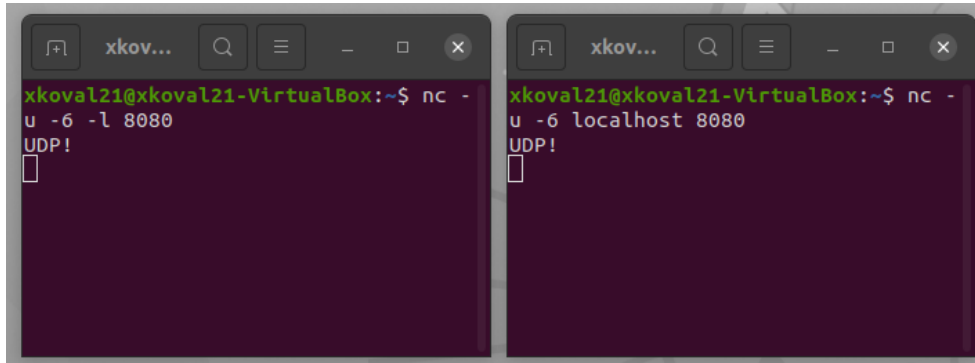


Compare this with the data from the ipk-sniffer program:

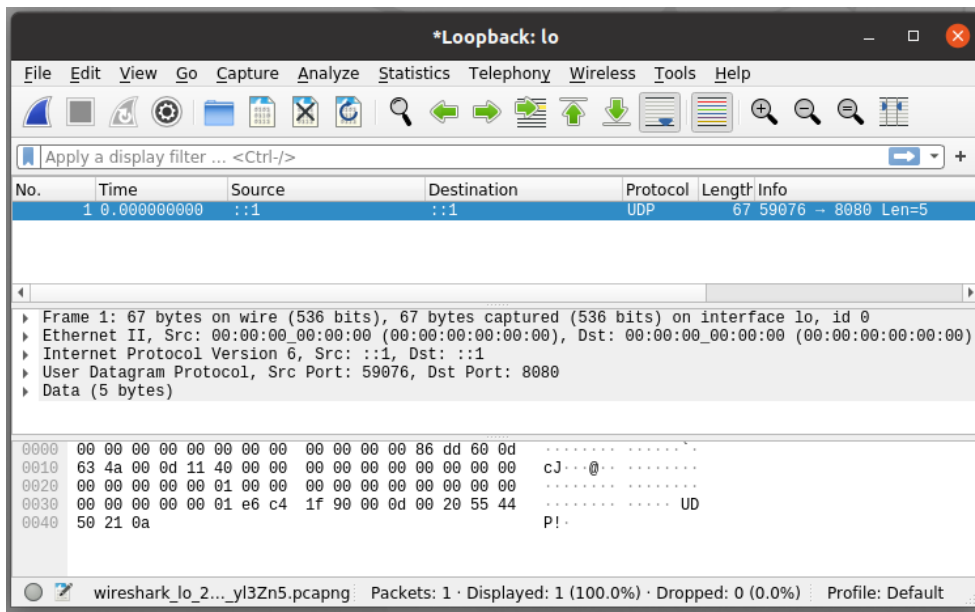


User Datagram Protocol

We also used two terminals, but added the -u operator.



Data from Wireshark:



Data from ipk-sniffer:

```
xkoval21@xkoval21-VirtualBox: ~/Desktop/ipk
xkoval21@xkoval21-VirtualBox:~/Desktop/ipk$ sudo ./ipk-sniffer -i lo --udp
allowed protocols: UDP
timestamp: 2022-04-24T21:19:55.18+02:00
src MAC: 00:00:00:00:00:00
dst MAC: 00:00:00:00:00:00
frame length: 67 bytes
src IP: ::1
dst IP: ::1
src port: 59076
dst port: 8080
0x0000: 00 00 00 00 00 00 00 00 00 00 00 00 86 dd 60 0d .....`..
0x0010: 63 4a 00 0d 11 40 00 00 00 00 00 00 00 00 00 00 cJ...@..
0x0020: 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 .....
0x0030: 00 00 00 00 00 01 e6 c4 1f 90 00 0d 00 20 55 44 ..... UD
0x0040: 50 21 0a ..... P!..

xkoval21@xkoval21-VirtualBox:~/Desktop/ipk$
```

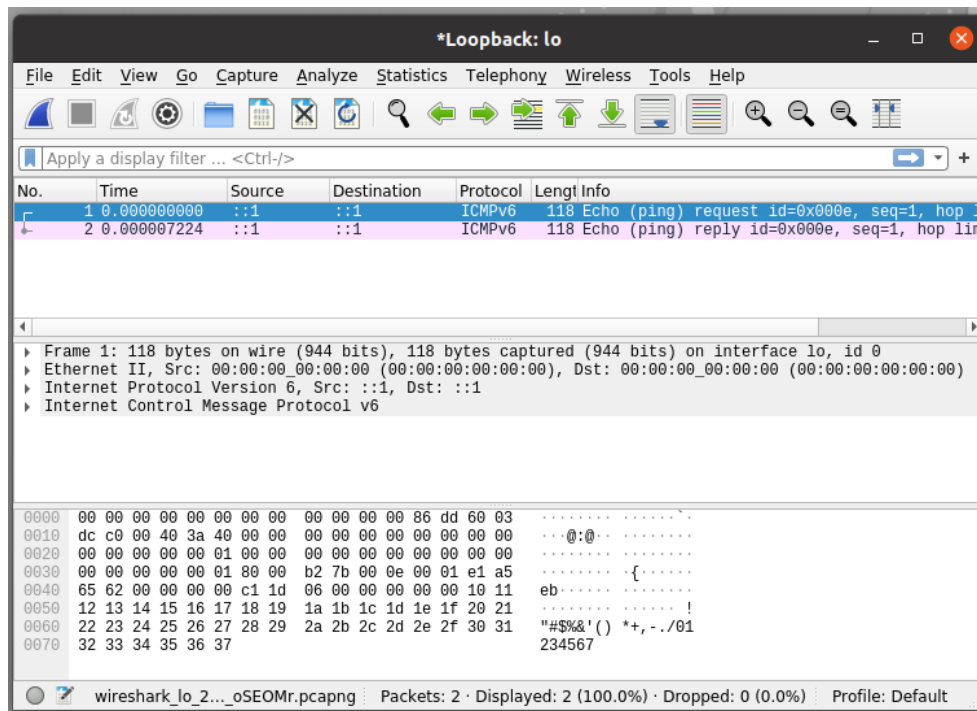
Internet Control Message Protocol version 6

In the terminal, entered `ping6 -c 1 ::1` and sent one packet.

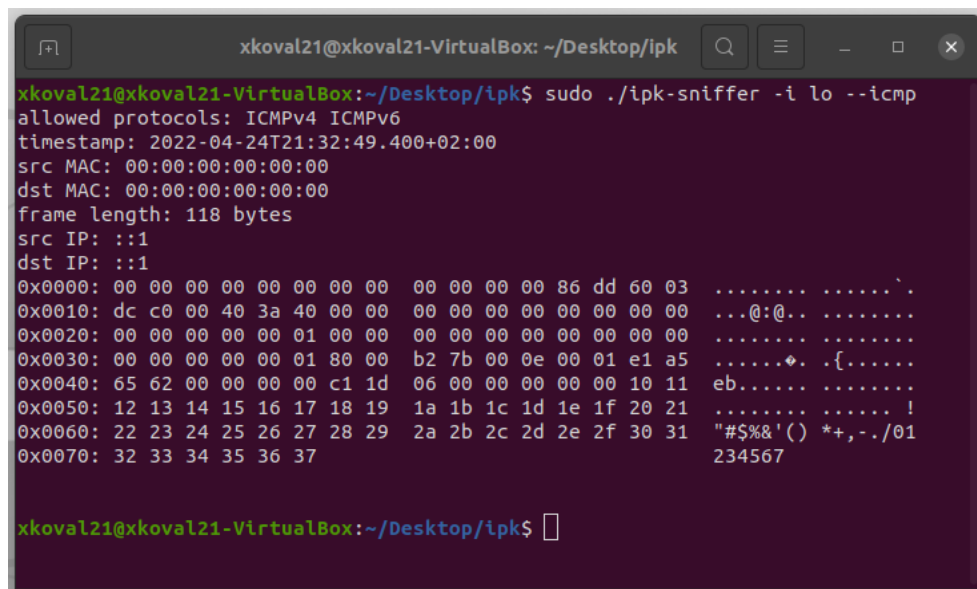
```
xkoval21@xkoval21-VirtualBox: ~
xkoval21@xkoval21-VirtualBox:~$ ping6 -c 1 ::1
PING ::1(::1) 56 data bytes
64 bytes from ::1: icmp_seq=1 ttl=64 time=0.021 ms

--- ::1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.021/0.021/0.021/0.000 ms
xkoval21@xkoval21-VirtualBox:~$
```

Data from Wireshark:



Data from ipk-sniffer:



Address Resolution Protocol

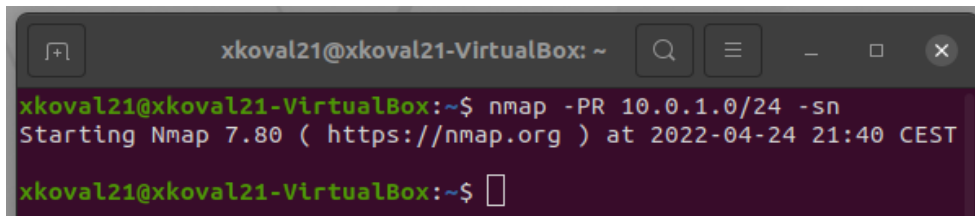
For another interface - enp0s3.



```
xkoval21@xkoval21-VirtualBox: ~/Desktop/ipk
xkoval21@xkoval21-VirtualBox:~/Desktop/ipk$ sudo ./ipk-sniffer -i
Option -i without parameter.

enp0s3
lo
any
bluetooth-monitor
nflog
nfqueue
xkoval21@xkoval21-VirtualBox:~/Desktop/ipk$
```

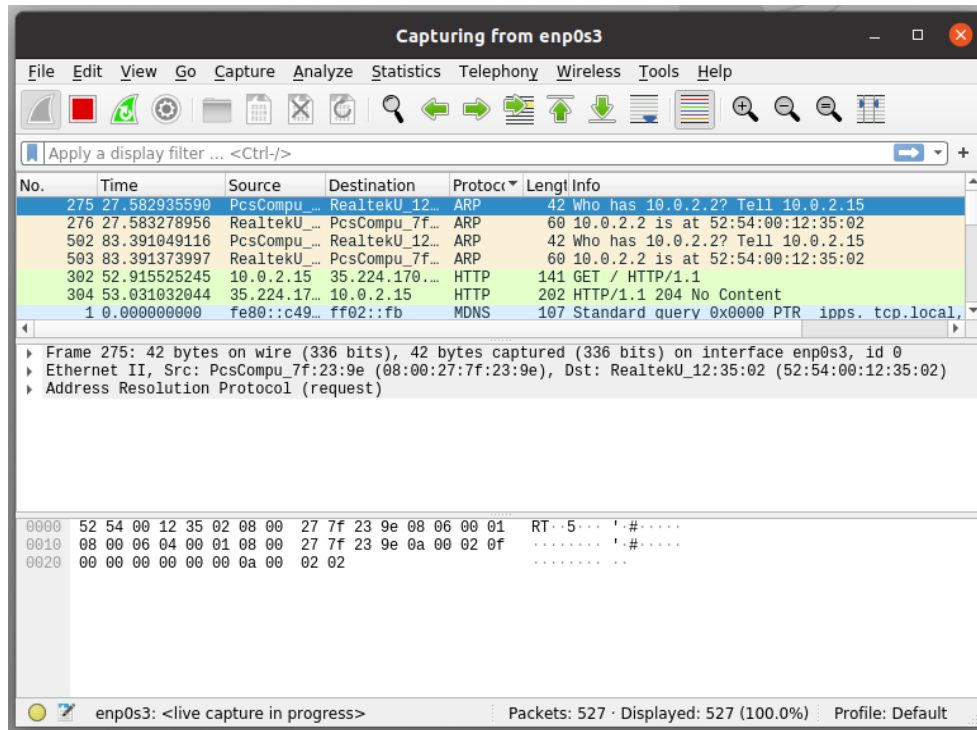
We used the nmap command and called it with the following parameters *-PR 10.0.1.0/24 -sn*.



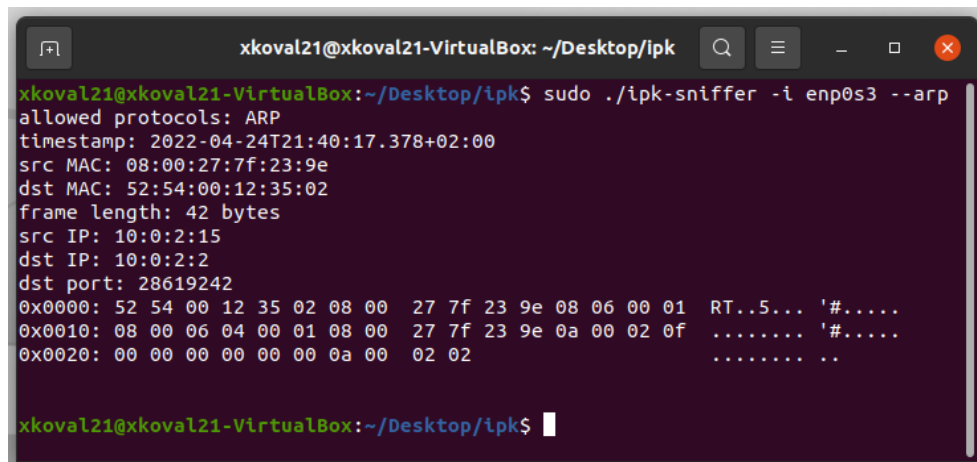
```
xkoval21@xkoval21-VirtualBox: ~
xkoval21@xkoval21-VirtualBox:~$ nmap -PR 10.0.1.0/24 -sn
Starting Nmap 7.80 ( https://nmap.org ) at 2022-04-24 21:40 CEST

xkoval21@xkoval21-VirtualBox:~$
```

Data from Wireshark:



Data from ipk-sniffer:



References

1. ether_header Struct Reference. *Propox* [online].
http://www.propox.com/download/edunet_doc/all/html/structether_header.html#_details
2. How to manually send an ARP request - Super User. [online].
<https://superuser.com/questions/940635/how-to-manually-send-an-arp-request>
3. Simplest Codings: Create your own packet sniffer in C. [online].
<http://simplestcodings.blogspot.com/2010/10/create-your-own-packet-sniffer-in-c.html>
4. Develop a Packet Sniffer with Libpcap. [online].
<https://vichargrave.github.io/programming/develop-a-packet-sniffer-with-libpcap/>
5. How to code a Packet Sniffer in C with Libpcap on Linux - BinaryTides. [online].
<https://www.binarytides.com/packet-sniffer-code-c-libpcap-linux-sockets/>
6. inet_ntop(3) — Linux manual page. *Michael Kerrisk - man7.org* [online].
https://man7.org/linux/man-pages/man3/inet_ntop.3.html
7. Carstens T. *Programming with pcap* [online]. 2002.
<https://www.tcpdump.org/pcap.html>
8. Optional arguments with getopt_long(3). [online].
<https://cfengine.com/blog/2021/optional-arguments-with-getopt-long/>