Computer Communications and Networks
Project: packet sniffer

Denis Karev

April 2022

1 Introduction

This is a manual for the packet sniffer application, developed as a part of the Computer Communications and Networks course. It can be used for monitoring network traffic on the specific adapter in promiscuous mode[1]. This application supports ARP, ICMP, TCP and UDP packets.

2 Building project

The provided Makefile allows to build the whole project. As the application is written in C, it uses `dotnet build` command to create binaries in the `bin\Release\netcoreapp3.1` folder.

3 Running project

You can use the provided shell script `ipk-sniffer` to run the project. It should be compiled before execution and binaries must be located in the `bin\Release\netcoreapp3.1` folder.
Application supports the following parameters:

1. `-i` or `--interface` – sets the network interface name to sniff packets on. If not set, the program will print the list of available interfaces and exit.
2. `-p` – sets the port number to be monitored. It includes both, source and destination ports. If not set, the program will monitor all the ports.
3. `-t` or `--tcp` – if set, program will monitor TCP packets.
4. `-u` or `--udp` – if set, program will monitor TCP packets.
5. `--icmp` – if set, program will monitor ICMP packets.
6. `--arp` – if set, program will monitor ARP packets.
7. `-n` – amount of packets to sniff, defaults to 1.

If no protocol was specified, the sniffer will monitor all supported protocols.

[1] Network controller mode, which allows to analyze all packets in the network

4 Implementation details

The program consists of 3 classes (`Program`, `Sniffer` and `PacketData`) and a `Settings` structure.

4.1 Start-up

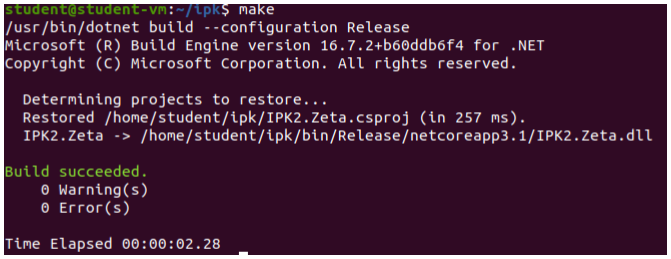
On start-up, the application will parse command-line arguments and load their values to the `Settings` structure, which will be then passed to the `Sniffer` constructor.
The `Sniffer` object will start capturing packets only after `StartCapture()` method is executed.

4.2 Sniffing

Sniffing is performed by the `Sniffer` class. This class uses the SharpPCAP library for analyzing packets. At the beginning, before capturing, it will setup the filter (port and protocols) using the `BuildFilter` method. After that, packets will be analyzed in the `InterfaceOnOnPacketArrival()` method. This event handler uses three methods: `TryReadTransportData()`, `TryReadIcmpData()` and `TryReadArpData()` to determine the packet type (these methods try to extract the packet as an `EthernetPacket`, `IcmpV4Packet`, `IcmpV6Packet` or `ArpPacket`) and extract the required data.
Data is stored in the `PacketData` record, which implements `ToString()` method for printing output in the required format.
At the end, `Sniffer` safely stops capturing and disposes used interface object.

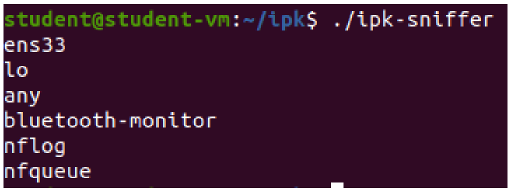
5 Testing

The application was tested on the provided virtual machine.



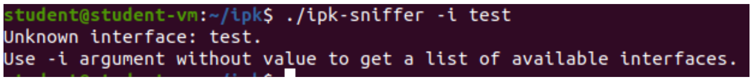
```
student@student-vm:~/ipk$ make
/usr/bin/dotnet build --configuration Release
Microsoft (R) Build Engine version 16.7.2+b60ddb6f4 for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

  Determining projects to restore...
  Restored /home/student/ipk/IPK2.Zeta.csproj (in 257 ms).
  IPK2.Zeta -> /home/student/ipk/bin/Release/netcoreapp3.1/IPK2.Zeta.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:02.28
```

Figure 1: Makefile output



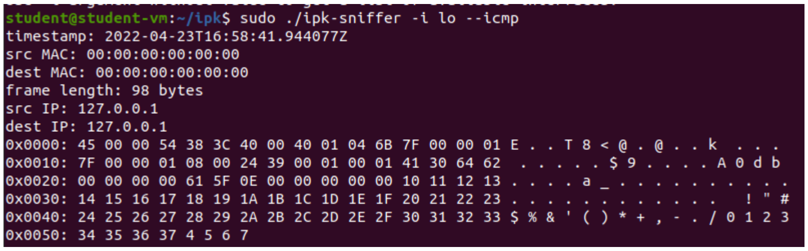
```
student@student-vm:~/ipk$ ./ipk-sniffer
ens33
lo
any
bluetooth-monitor
nflog
nfqueue
```

Figure 2: Running program without interface parameter



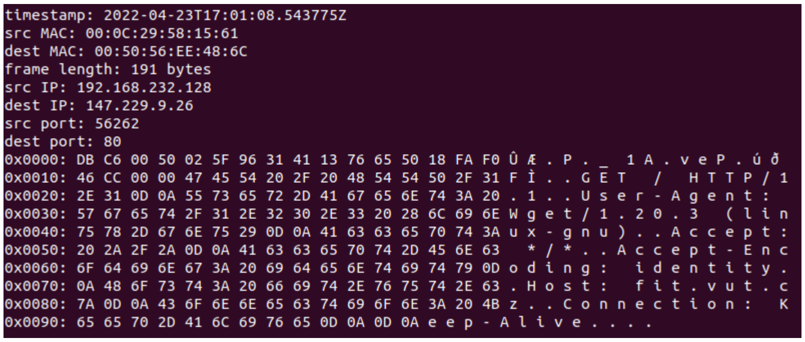
```
student@student-vm:~/ipk$ ./ipk-sniffer -i test
Unknown interface: test.
Use -i argument without value to get a list of available interfaces.
```

Figure 3: Running program with a wrong -i parameter value



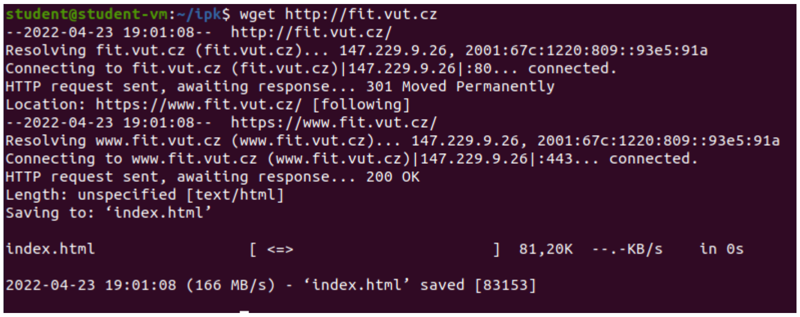
```
student@student-vm:~/ipk$ sudo ./ipk-sniffer -i lo --icmp
timestamp: 2022-04-23T16:58:41.944077Z
src MAC: 00:00:00:00:00:00
dest MAC: 00:00:00:00:00:00
frame length: 98 bytes
src IP: 127.0.0.1
dest IP: 127.0.0.1
0x0000: 45 00 00 54 38 3C 40 00 40 01 04 6B 7F 00 00 01 E . . T 8 < @ . @ . . k  . . .
0x0010: 7F 00 00 01 08 00 24 39 00 01 00 01 41 30 64 62  . . . . . . $ 9 . . . . A 0 d b
0x0020: 00 00 00 00 61 5F 0E 00 00 00 00 00 10 11 12 13 . . . . a _ . . . . . . . . . .
0x0030: 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 . . . . . . . . . . . .   ! " #
0x0040: 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 $ % & ' ( ) * + , - . / 0 1 2 3
0x0050: 34 35 36 37 4 5 6 7
```

Figure 4: Catching an ICMP packet using lo interface. Packet was sent by ping localhost command.



```
timestamp: 2022-04-23T17:01:08.543775Z
src MAC: 00:0C:29:58:15:61
dest MAC: 00:50:56:EE:48:6C
frame length: 191 bytes
src IP: 192.168.232.128
dest IP: 147.229.9.26
src port: 56262
dest port: 80
0x0000: DB C6 00 50 02 5F 96 31 41 13 76 65 50 18 FA F0 Û Æ . P . _  1 A . v e P . ú ð
0x0010: 46 CC 00 00 47 45 54 20 2F 20 48 54 54 50 2F 31 F Ì . . G E T   /   H T T P / 1
0x0020: 2E 31 0D 0A 55 73 65 72 2D 41 67 65 6E 74 3A 20 . 1 . . U s e r - A g e n t :
0x0030: 57 67 65 74 2F 31 2E 32 30 2E 33 20 28 6C 69 6E W g e t / 1 . 2 0 . 3   ( l i n
0x0040: 75 78 2D 67 6E 75 29 0D 0A 41 63 63 65 70 74 3A u x - g n u ) . . A c c e p t :
0x0050: 20 2A 2F 2A 0D 0A 41 63 63 65 70 74 2D 45 6E 63   * / * . . A c c e p t - E n c
0x0060: 6F 64 69 6E 67 3A 20 69 64 65 6E 74 69 74 79 0D o d i n g :   i d e n t i t y .
0x0070: 0A 48 6F 73 74 3A 20 66 69 74 2E 76 75 74 2E 63 . H o s t :   f i t . v u t . c
0x0080: 7A 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 4B z . . C o n n e c t i o n :   K
0x0090: 65 65 70 2D 41 6C 69 76 65 0D 0A 0D 0A e e p - A l i v e . . . .
```

Figure 5: Catching several TCP packets using ess33 interface. Packet was sent by running wget fit.vut.cz



```
student@student-vm:~/ipk$ wget http://fit.vut.cz
--2022-04-23 19:01:08--  http://fit.vut.cz/
Resolving fit.vut.cz (fit.vut.cz)... 147.229.9.26, 2001:67c:1220:809::93e5:91a
Connecting to fit.vut.cz (fit.vut.cz)|147.229.9.26|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://www.fit.vut.cz/ [following]
--2022-04-23 19:01:08--  https://www.fit.vut.cz/
Resolving www.fit.vut.cz (www.fit.vut.cz)... 147.229.9.26, 2001:67c:1220:809::93e5:91a
Connecting to www.fit.vut.cz (www.fit.vut.cz)|147.229.9.26|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: ‘index.html’

index.html                 [ <=>                      ]  81,20K  --.-KB/s    in 0s

2022-04-23 19:01:08 (166 MB/s) - ‘index.html’ saved [83153]
```

Figure 6: Wget, which received the packet, displayed on the previous screenshot

6 Bibliography

1. SharpPCAP authors. *SharpPCAP readme.*
   `https://github.com/dotpcap/sharppcap/`

2. The Tcpdump group. *PCAP MAN page.*
   `https://www.tcpdump.org/manpages/pcap.3pcap.html`