BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

IPK – 2nd Assignment

Packet Sniffer Manual

April 25, 2021

Marek Filip

Contents

1 Abstract

This manual was created as a documentation for the Packet Sniffer assignment of the BUT FIT IPK 2021 summer course. The original assignment details can be found here [9] (BUT FIT internal).

The program is a network analyzer that catches and filters packets on a specific interface. Depending on the provided arguments, either *TCP*, *UDP*, *ICMP* or *ARP* number of packets is sniffed and details about them, including byte data, is printed out to the ‘stdout‘ in a hex/ascii format.

2 Implementation details

2.1 Handling of arguments

When the program gets executed, the first thing that gets handled are the program arguments.

Parsing of arguments is done via the `System.CommandLine` [7] C# library. The options are defined as `Option` objects belonging to the root command. The root command object has a function *handler* where the actual semantical handling of arguments happen. The handler starts handling when the root command is invoked.

When the user enters port number that is less than zero, the program informs the user about the erroneous behaviour. Port with number `-1` internally refers to all possible ports. Similarly, when the user enters number of packets that is less or equal to zero or when there is no interface to listen to, the user gets notified again.

To get the *help* option, user needs to call the program with one of these options: `-?  -h --help`. The help screen is also shown when invalid program argument is present. The help screen is internally constructed from the defined argument options' descriptions.

2.2 Listing interfaces

When no error happens during the parsing of arguments, the program flow splits in two possible paths. Either it starts capturing network packets or it lists all of the possible interfaces that can be listened to.

This subsection talks about the listing of the interfaces. Before I start the explanation I must disclose that all of the work that is done with the network interfaces is thanks to the *SharpPcap* [5] C# library.

To list a device I get the list of all interfaces from the library and loop over it to list all of them out to the console.

2.3 Capturing packets

The other possible program path is to capture the packets and output relevant information they contain. The outputted information contains:

- Used filter
- time in RFC3339 format [4],
- source address and port,
- destination address and port,
- number of bytes transmitted,
- transmitted bytes in hexadecimal and ASCII format splitted by chunks of 16 bytes.

When the selected interface exists doesn't exist the user is notified via error message.

Firstly, the interface is opened in a promiscuous mode with a timeout of 1000 milliseconds. Then the filter, constructed from the program arguments, is set. After that the interface starts capturing the filtered packets and prints out information about each one of them. The program is now in loop, until the desired number of packets (default: 1) is captured or the user exits out of the program in a different way, e.g. `ctrl+c`.

The TCP and UDP packets are listed with both their IP addresses and ports. ICMP packets with their IP addresses but without ports. ARP packets are listed with their hardware addresses instead and without ports.

At the end, the general statistics provided by the interface is outputted.

2.4 Problems along the way

C# dotnet could not find `libcap.so` in my Fedora distribution. After an hour of debugging and using `LD_DEBUG` on the reference machine, I found out I mistook `libcap.so` for `libpcap.so` and quickly resolved the issue with a proper installation on my Fedora machine.

Reading an array of bytes splitted by chunks of fixed length was a difficult concept for me in C#. That's why I seeked the answer online and stumbled upon the Code Project hub. Bunty Swapnil asked the question and Matt T Heffron provided the final code I use with a tweak in my own code base. The original answer can be found here [2].

2.5 Code documentation

XML-styled comments [10] to document the structures, classes and their methods.

2.6 Used external tools

When I start to work in a new programming language I usually try to find some good CLI analyzers and formatters to help me make sure the code is statically well-formed. For code analysis and formatting of the C# source files, the `Roslynator` [8] tool is perfect.

Its commands are also defined in Makefile as targets to unify the development process. Regarding Makefile, I made `prepend-header` target, that prepends my custom header description to every C# source file. Needless to say, the work I did making the commands largely exceeded the work needed to define these headers by hand. It was a nice learning experience.

3 Testing

3.1 Unit testing framework

Since I am new to C# I chose xUnit [6] as the open-source framework. The library uses two kinds of tests, **facts** and **theories**. Facts are tests that are always *true*, they do not have any parameters and should test data that doesn't change. Theories are tests that are only true for some specific data, they should test data that does change and they do come with parameters.

I mostly use theories in my tests. The implemented tests primarily cover functions listed in `Utils.cs` which concern data operations.

3.2 Manual tests with Wireshark

The other form of testing is manual. When the program has began to output sensible data, *Wireshark* [1] helped me make sure that they were correct. By cross-referencing the outputted results with the Wireshark packet capturing I check if the data are identical.

To get the compared data in a reasonably quick way (making sure I am looking at the correct packet), I use filter searching (input field at top of the application) together with the timestamp checking to ensure data identity.

I conducted the manual testing on both local and reference machine. I could not find any visible error when analyzing the output.

4 Summary

During the assignment, when I needed some explanation of how packet capturing works internally, I drew it from the manual here [3]. When actually implementing the packet capturing, the beginning code was inspired by the examples listed in the SharpPcap library's repository here [5].

I have enjoyed this assignment. It has taught me lessons in networking, especially ARP and ICMP protocols and the packet traffic in general. I think the difficulty is adequate. If I am to complain about something, that would be all of the scrutinous details regarding documentation and citation, but I can see the benefits even if I do not like it.

References

1. COMBS, Gerald. *Wireshark* [online]. 2021 [visited on 2021-04-25]. Available from: `https://www.wireshark.org/`.
2. HEFFRON, Matt T. *Code Project: For those who code* [online]. 2019-11 [visited on 2021-04-25]. Available from: `https://www.codeproject.com/Answers/684747/i-want-to-read-1-kb-data-each-time-from-byte-array#answer2`.
3. JACOBSON, Van, et al. *pcap(3) - Linux User's Manual* [online]. 2021 [visited on 2021-04-24]. Available from: `https://www.tcpdump.org/pcap3_man.html`.
4. KLYNE, G., et al. *Date and Time on the Internet: Timestamps* [online]. [N.d.] [visited on 2021-04-25]. Available from: `https://tools.ietf.org/html/rfc3339`.
5. MORGAN, Chris, at al. *sharppcap* [online]. GitHub [visited on 2021-04-24]. Available from: `https://github.com/chmorgan/sharppcap`.
6. NEWKIRK, James, Brad Wilson. *xUnit.net* [online]. 2021 [visited on 2021-04-24]. Available from: `https://xunit.net/`.
7. PIHRT, Josef. *command-line-api* [online]. GitHub [visited on 2021-04-25]. Available from: `https://github.com/dotnet/command-line-api`.
8. SEQUEIRA, Jon. *roslynator* [online]. GitHub [visited on 2021-04-25]. Available from: `https://github.com/JosefPihrt/Roslynator`.
9. VESELÝ, Vladimír. *Varianta ZETA: Sniffer paketů* [online]. 2021-02 [visited on 2021-04-25]. Available from: `https://wis.fit.vutbr.cz/FIT/st/course-sl.php.cs?id=727797&item=83874`.
10. WAGNER, Bill, et al. *Document your C# code with XML comments* [online]. 2021-01 [visited on 2021-04-24]. Available from: `https://docs.microsoft.com/en-us/dotnet/csharp/codedoc`.