

## IPK – Project 2

Adrián Kálazi  
xkalaz00@stud.fit.vutbr.cz

April 25, 2021

# Contents

<b>1</b>	<b>Implementation</b>	<b>3</b>
1.1	Argument parsing . . . . .	3
1.2	Sniffer - core . . . . .	3
1.3	Sniffer - packet examination . . . . .	4
<b>2</b>	<b>Testing</b>	<b>4</b>
<b>3</b>	<b>Bibliography</b>	<b>5</b>

## Prologue

Our task was to create a packet sniffer which would effectively catch, identify and dump the content of packets. The capture could optionally be restricted to a specific interface, port or a specific protocol.

Protocol could be one of:

- TCP
- UDP
- ICMP
- ARP

The application should also support basic functionality displayed by similar applications (see section 2).

## 1 Implementation

The application was implemented in C++ with the use of the libpcap packet capture library. We used the documentation of PCAP<sup>1</sup> and linux manpages. Error handling is mostly done with C++ exceptions, although some C functions were handled without them. Doxygen comments are located in the header files.

### 1.1 Argument parsing

Command line arguments parsing is done by the `Config` class. The parsing is done manually, without the use of `getopt`. Resulting options are stored in public member variables of the `Config` instance.

The program can also exit without error nor any sniffing in only two cases:

- `-h` | `--help` - prints help and exits
- `-i` | `--interface` - prints interfaces and exits

### 1.2 Sniffer - core

The core functionality of the sniffer is implemented in the `Sniffer` class. On initialization, argument parsing is done and a `Config` instance is stored in a member variable for config access.

After initialization, the `run` method is executed, which progresses as follows:

1. The `pcap` handler is initialized.
2. A packet filter program is compiled from the enabled options in `Config` and applied to the `pcap` handle.
3. A loop is performed, in which each captured packet is examined in the `packet_callback` static method (given as a callback to `pcap_loop`)
4. Cleanup of the compiled filter program and `pcap` handle is performed.

---

<sup>1</sup><https://www.tcpdump.org/manpages/pcap.3pcap.html>

### 1.3 Sniffer - packet examination

Packet examination is done in the `packet_callback` static method as stated in section 1.2. The packet is unpacked in layers, depending on the data from previous layers.

1. First, The data-link layer is examined depending on the result of `pcap_datalink()`, supported header types are `DLT_EN10MB` (ethernet frame), `DLT_LINUX_SLL` and `DLT_LINUX_SLL2` (Linux "cooked" capture encapsulation). The header is then stripped.
2. Then, the EtherType is a deciding factor, with `ETHERTYPE_IP`, `ETHERTYPE_IPV6` and `ETHERTYPE_ARP` supported. This header is also stripped after examining this layer.
3. After that, the transport layer protocol is determined, with `IPPROTO_TCP`, `IPPROTO_UDP`, `IPPROTO_ICMPV6` and `IPPROTO_ICMP` as valid options.
4. The necessary data collected in the previous steps is then put to stdout in the expected format.

## 2 Testing

The most notable open-source application with similar base functionality is probably wireshark<sup>2</sup>. We tested our application by sending and comparing sample packets created with easily identifiable data.

---

<sup>2</sup><https://www.wireshark.org/>

### **3 Bibliography**