

BRNO UNIVERSITY OF TECHNOLOGY  
FACULTY OF INFORMATION TECHNOLOGY

Computer Communications and Networks – Project 2  
Network Services Scanner

# Contents

[illegible]

# 1 Introduction

The aim of this project was to create a simple TCP and UDP scanner. The program scans the specified IP address and ports and outputs the status of the ports (open, filtered, closed). Packets must be sent using BSD sockets.

## 1.1 TCP SYN scanner

A TCP scanner is the basic form of scanning, and a SYN scanner is one of its forms. This type of scanning is known as a "half-open" scanner because it never opens the entire connection [3]. It works by sending a packet with the SYN flag and waiting for a response. If no response is received, the port is marked as filtered; if an RST response is received, the port is marked as closed. If a response with SYN-ACK flags arrives, the port is open. The advantage of this scanner is that the first packet that arrives clearly indicates whether the port is open or closed, and that the TCP protocol ensures reliable packet delivery.

## 1.2 UDP scanner

The ICMP packet reception method is used. A UDP packet is sent to a given port, and if it is not open, the system responds with an ICMP packet of type 3 and code 3 (port unreachable); otherwise, the port is considered open. However, if the port is blocked by a firewall, for example, it is also marked as open. Another disadvantage of the UDP scanner is that it does not guarantee reliable packet delivery, so it is necessary to send it several times before the port is marked as open/closed. [1].

# 2 Implementation

The implementation is divided into several parts. First, the given arguments must be processed, then the filter is set, the packet is sent, and finally the received packet is processed.

## 2.1 Processing arguments

I originally wanted to process the parameters using the `getopt()` function, but this function only supports single-character options, which is not suitable for `-pt` and `-pu`. The `getopt long` function is also not suitable because it only supports `--option(2 hyphens)` options, which again is not suitable.<sup>(1)</sup> I therefore created my own function to process the parameters. The required ports are stored in an array. If the ports are specified as a range, the first and last ports are stored in an array and the range flag is set. The hostname is processed using the `getaddrinfo()` function, which, when set correctly, can also obtain the IP version (IPv4 or IPv6).

## 2.2 Packet capture settings

The disadvantage of so-called raw packets is that some operating systems (FreeBSD, new versions of Windows) do not support receiving these packets via clipboards by default [2], which is why the `pcap.h` library is used. Another advantage of this library is that it allows you to set the required packet parameters. So the source and destination IP addresses, source port, packet type, and all others are filtered out. As a result, the only thing that passes is the response from the given host on the specified port.

## 2.3 Sending a packet

Before sending a packet, its IP header must be filled in. The `ip.h` library is used for this, where the structure for the given IP header is already predefined. Next, the TCP or UDP header must be filled in, and for this

---

<sup>1</sup> For more information, see: <https://stackoverflow.com/questions/14634553/c-using-getopt-to-parse-multiple-arguments>

use the `tcp.h` and `udp.h` libraries. For the `tcp` header, it is also necessary to calculate the checksum and set the SYN flag.

Using the `setsockopt()` function with the necessary parameters, we inform the system that we are only adding information at the network layer and that the system must add the rest itself. Then we just use the `sendto()` function to send the packet.

## 2.4 Processing of the received TCP packet

The `pcap_loop()` function is used to capture TCP packets. One of its parameters is to call the given function when the requested packet arrives. This packet is then processed as described in section 1.1. If the response does not arrive within 1 second, the wait is interrupted by the SIGALARM signal. One more packet is sent, and if no response is received after two attempts, the port is considered filtered.

## 2.5 Processing of the received ICMP packet

The `pcap_loop()` function is also used to capture TCP packets, and the packet is processed as described in section 1.2. UDP and ICMP packets do not have a mechanism for reliable packet delivery, so if no response in the form of an ICMP packet is received within 1 second, another UDP packet is sent. If no response is received even then, the port is considered open.

## 2.6 IP version 6 support

As mentioned in section 2.1, the `getaddinfo()` function determines whether it is IP version 6. In this case, the `ip6.h` library is required to create the IP header, and the entire IP header is then twice the size, meaning that the necessary flags (SYN, ACK, RST, ICMP type and code) are located in different places.

When sending a UDP raw packet with IPv6 addresses, an ICMPv6 response is expected, and if the port is unavailable, it is not type 3 and code 3 as stated in the assignment, but type 1 and code 4<sup>2</sup>.

# 3 Testing

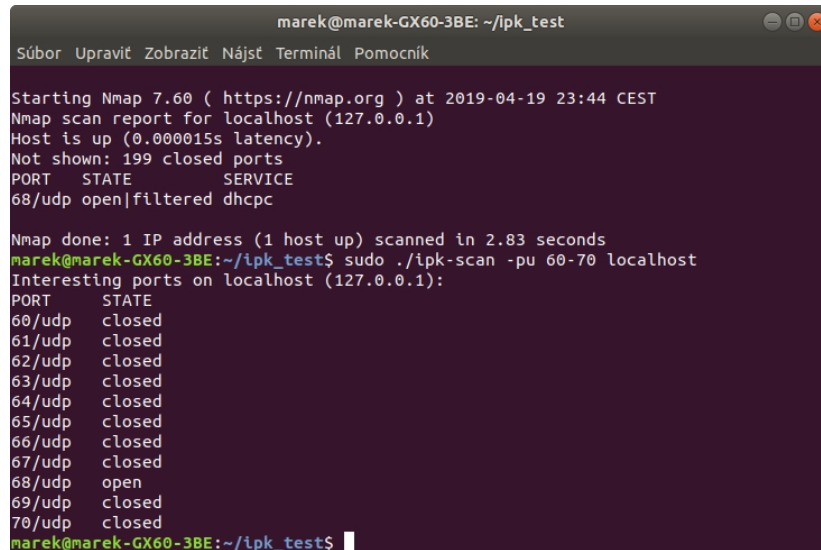
Testing was performed in two ways. Comparison of the output with the reference and analysis of sent and received packets, which I used mainly to detect errors when the required response did not arrive. Ultimately, this helped me detect errors even when the output was the same as the reference.

---

<sup>2</sup> For more information, see: [https://en.wikipedia.org/wiki/Internet\\_Control\\_Message\\_Protocol\\_for\\_IPv6](https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol_for_IPv6)

### 3.1 Comparison of output with reference

To test which ports are open and which are closed, I used the bash command `nmap`, which lists all open ports in the specified range in the output.



```
marek@marek-GX60-3BE: ~/ipk_test
Súbor Upraviť Zobrazíť Nájst Terminál Pomocník

Starting Nmap 7.60 ( https://nmap.org ) at 2019-04-19 23:44 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000015s latency).
Not shown: 199 closed ports
PORT      STATE      SERVICE
68/udp    open|filtered  dhcpd

Nmap done: 1 IP address (1 host up) scanned in 2.83 seconds
marek@marek-GX60-3BE:~/ipk_test$ sudo ./ipk-scan -pu 60-70 localhost
Interesting ports on localhost (127.0.0.1):
PORT      STATE
60/udp    closed
61/udp    closed
62/udp    closed
63/udp    closed
64/udp    closed
65/udp    closed
66/udp    closed
67/udp    closed
68/udp    open
69/udp    closed
70/udp    closed
marek@marek-GX60-3BE:~/ipk_test$
```

Figure 1: Using the `nmap` command as a UDP scanner and comparison with my scanner

### 3.2 Packet analysis

To analyze sent and received packets, I used the `wireshark` tool, which, when set up correctly, can check flags (SYN, ACK, RST...), checksums, and many other things on all layers (not just the network layer).

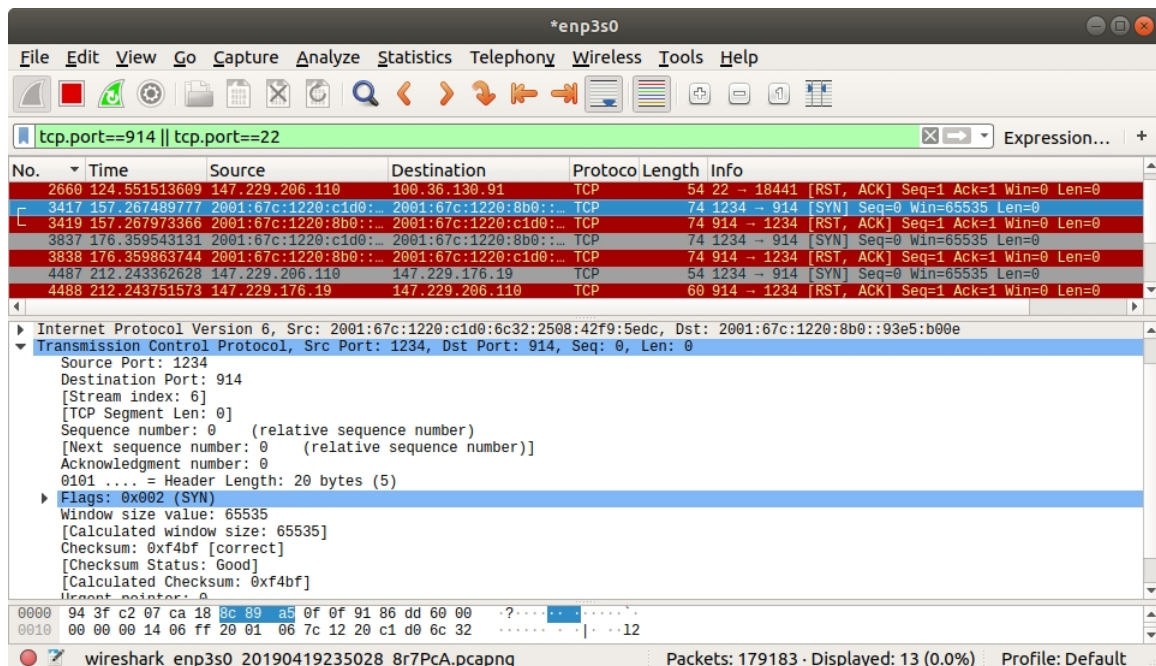


Figure 2: Wireshark analysis of a sent TCP SYN packet IPv6

### 3.2.1 Analyzing packets triggered by the nmap command

In Wireshark, I finally analyzed the packets generated by the `nmap` command. I found that scanning TCP and UDP ports works exactly the same way as with this scanner. The method is described in sections 1.1 and 1.2.

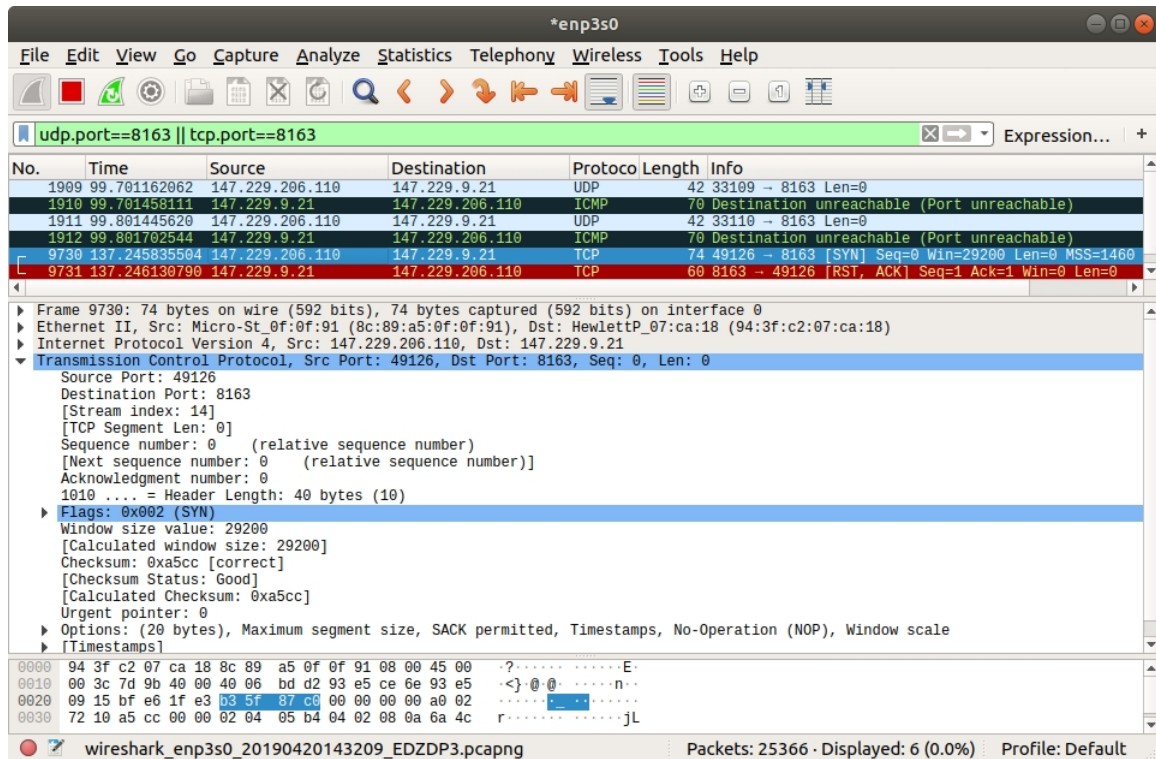


Figure 3: Analysis of a TCP packet sent by the nmap command

## References

- [1] Ajiratech: Port scanner. March 15, 2017. Available from:  
\_ [https://en.wikipedia.org/wiki/Port\\_scanner](https://en.wikipedia.org/wiki/Port_scanner)
- [2] Czokl: Simple TCP/UDP scanner in C++. 2018-4-21. Available from: <https://www.security-portal.cz/clanky/jednoduch%C3%BD-tcpudp-scanner-v-c>
- [3] Hegy: Port scanning. 2011-5-30. Available from: [https://cs.wikipedia.org/wiki/Skenov%C3%A1n%C3%AD\\_port%C5%AF](https://cs.wikipedia.org/wiki/Skenov%C3%A1n%C3%AD_port%C5%AF)