Isa projekt 2024

Matyáš Krejza xkrejz07

2024-10-11

Contents

Isa projekt 2024

An application for obtaining network traffic statistics

Author: Matyáš Krejza xkrejz07

1.0 Basic usage

```
make
```

```
./isa-top -i eth0
```

Basic instructions

This is the minimal setup and will use the default values for arguments. You can use these options:
- -i - name of the desired interface the program should use (required)

- -s <b|p> - sorting behavior of the application. `b` will sort by bytes and `p` by the number of packets

The app was tested and developed on Fedora 41 and build was tested on Merlin using c++ version 20.

1.1 Project assignment

The goal of the project is to create a tool `isa-top`, which shows transfer speeds for IP addresses communicating on the selected interface. The tools sorts up to 10 of these connections either by the most data being sent or by the most number of packets being sent.

2.0 The basics

The bare-bones of this project is listening for packets on the selected interface. Each packet is parsed and evaluated. Then the program outputs a statistics based on this evaluation which refreshes every second.

2.1 What is a packet

We can think of a packet as a box. A box that travels from point A to point B and has all the necessary information that it might need on this journey. The box also carries information that it wants to deliver which, in most cases but not always, takes up the most size from the box size. There is a big numbers of different types of packets, but in this project, we mostly care about 2. An IPv4[1][1] packet and IPv6[2][2] packet.

2.1.1 Anatomy of a packet

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                  Example Internet Datagram Header
```

IPv4 packet header [1][1]

This is an example of a IPv4 packet header. In this project we mostly care about the fields Total length, Protocol, Source and Destination address.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Payload Length        |  Next Header  |   Hop Limit   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Destination Address                      +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPv6 packet header[2][2]

This is an example of a IPv6[2][2] packet header. Here we care about Source and destination address, Next Header and Payload Length fields. Notice that the Total length field is absent. This will be important later on.

3.0 Application design

I have decided to make the app into two main pieces. One that takes care of capturing and parsing packets and second one that sorts and prints the statistics. These pieces will run each as an independent thread and they will share an array of captured connections as follows:

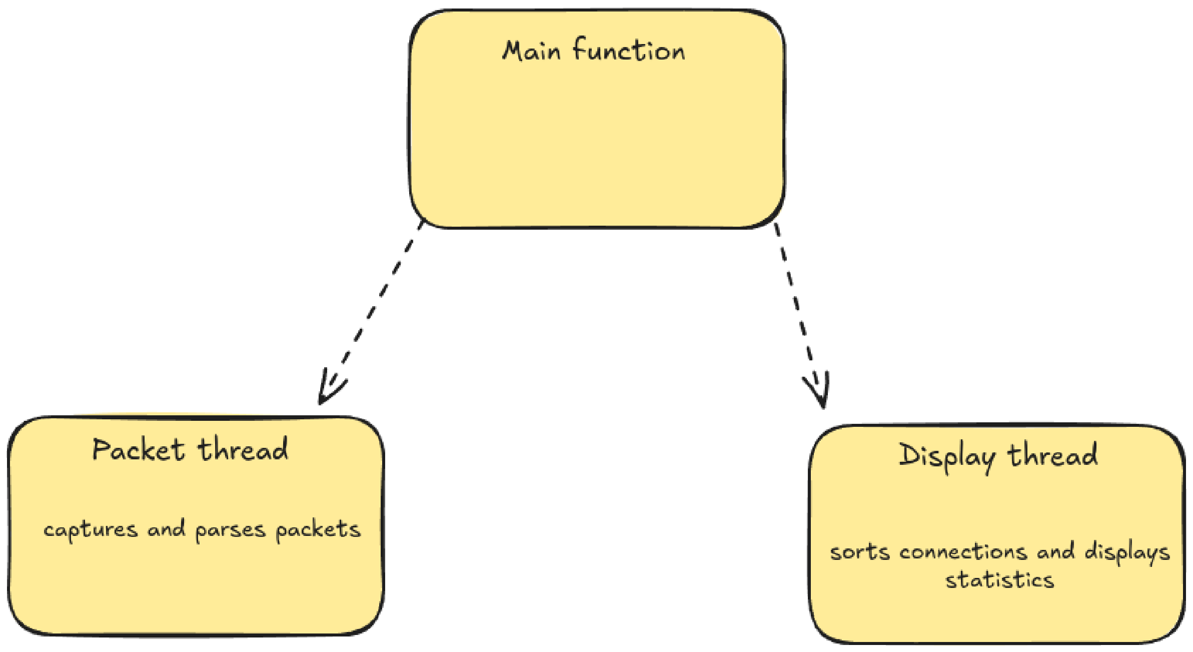

Main function
Packet thread
captures and parses packets
Display thread
sorts connections and displays statistics

Diagram of threads used in the program

The capturing of packets will be done via the **libpcap**[8][8] library and statistics will be displayed using the **ncurses**[7][7] library.

4.0 Implementation

The `main` function lives in the `isa-top.cpp` file. The rest of the project is in the files `read_packets.cpp` and `display_speeds.cpp`. The project follows the design mentioned chapter above. Each of the `cpp` files has a corresponding `hpp` file but there also is an extra `connections.hpp` file. In this file I define the `Connection` structure and a public vector `vector<Connection>` that holds the data across the two threads.

The Connection struct is defined as follows:

```
struct Connection {

string src_ip; //Source IP of the connection

string dst_ip; //Destination ip of the connection

string protocol; //Protocol used by the connection

uint16_t src_port; //Source port

uint16_t dst_port; //Destination port

uint64_t bytes_sent; //How many bytes were sent

uint64_t bytes_received; //How many bytes were received

uint64_t packets; //Total count of packets exchanged
};
```

Connection struct implementation

While I acknowledge that my use of `vector<Connection>` is a bit inefficient compared to different data structures, I decided to use it anyways, mainly because I was afraid of implementing something more exotic.

Every sniffed packet gets parsed through the function `parse_packet` which is set as a callback to the pcap library. In this function I first check if the packet is IPv4 or IPv6 because, as mentioned earlier, they have different structures and the program needs to parse them differently. After recognizing the version of the packet, the program parses source and destination address, as well as protocol information and transport header information. Based on the later two the program then sets the protocol of the packet and based on that it tries to parse the source and destination port information. The last step is to look if this is the first time seeing a packet with this connection information. If that is the case, we create a new connection struct and we save it. However if such connection already exists, we update the bytes transferred as well as number of packets transferred.

The size of each packet is determined via the libpcap library using the struct `pcap_pkthdr *header` that is provided with every packet captured. When a packet is captured, the program stores it's size by `packet_size = header->len;`

4.1 Packet direction

Since there is no reliable way to tell if the packet was sent or received by our device, eq. running this program on a server which forwards connections, we need to decide how to count if the packet was sent or received. I chose the following method. If I capture a packet with connection information that I have not seen before, eq. the destination ip is new, I count it as being **sent**. If we get a packet that was seen before, we take a look at the source ip. If the destination ip matches the destination IP we seen before, we count it as being sent.

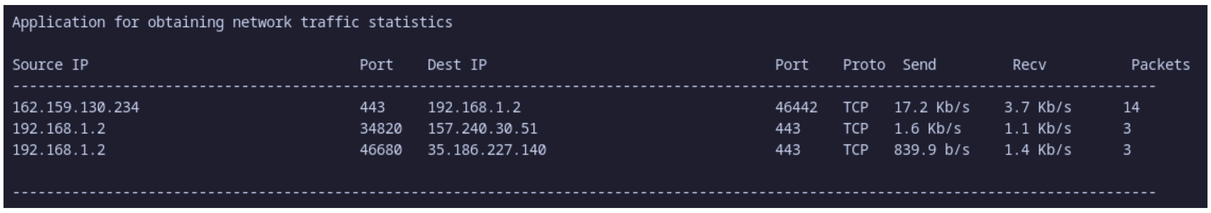
4.2 Supported protocols

The program can identify and display which protocol is used in each particular connection. The supported protocols are: TCP[3][3], UDP[4][4], ICMP[5][5] and ICMPv6[6][6].

5.0 Instructions

The app needs access to the network devices in order to function properly. Run it with elevated privileges. The program can be executed as `./isa-top -i interface`, where interface is the name of the network interface the program should listen on.

After execution, the app starts printing out statistics in a table using the ncurses library.



```
Application for obtaining network traffic statistics

Source IP                                 Port    Dest IP                                   Port    Proto  Send        Recv        Packets
---------------------------------------------------------------------------------------------------------------------------------------
162.159.130.234                           443     192.168.1.2                               46442   TCP    17.2 Kb/s   3.7 Kb/s    14
192.168.1.2                               34820   157.240.30.51                             443     TCP    1.6 Kb/s    1.1 Kb/s    3
192.168.1.2                               46680   35.186.227.140                            443     TCP    839.9 b/s   1.4 Kb/s    3

---------------------------------------------------------------------------------------------------------------------------------------
```

An example of the program's output

You can see the `Source Ip` and `Source Port`, `Destination Ip` and `Destination Port`. There is also the detected `Protocol`, information about `Send` and `Received` data and the total number of packets.

6.0 Testing

When testing the application, I have used a number of approaches.

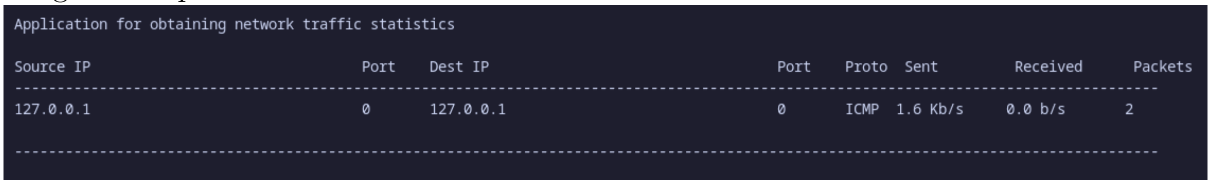
6.1 Ping command

The `ping` utility is built in almost every operating system. It allows the user to check whether or not a service is responding. We can ping `localhost` and make our app listen on the `lo` interface to filter out any unwanted traffic.

Command:

```
ping 0.0.0.0
PING 0.0.0.0 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.045 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.043 ms
...
```

Program output:



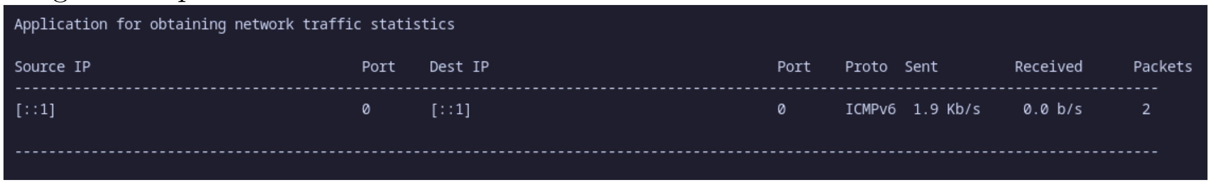
```
Application for obtaining network traffic statistics

Source IP                                 Port    Dest IP                                   Port    Proto  Sent        Received    Packets
---------------------------------------------------------------------------------------------------------------------------------------
127.0.0.1                                 0       127.0.0.1                                 0       ICMP   1.6 Kb/s    0.0 b/s     2

---------------------------------------------------------------------------------------------------------------------------------------
```

This can also be used to test the IPv6 support.

Command:

```
ping6 localhost
PING localhost (::1) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=0.061 ms
64 bytes from localhost (::1): icmp_seq=2 ttl=64 time=0.045 ms
...
```

Program output:



```
Application for obtaining network traffic statistics

Source IP                                 Port    Dest IP                                   Port    Proto  Sent        Received    Packets
---------------------------------------------------------------------------------------------------------------------------------------
[::1]                                     0       [::1]                                     0       ICMPv6  1.9 Kb/s    0.0 b/s     2

---------------------------------------------------------------------------------------------------------------------------------------
```

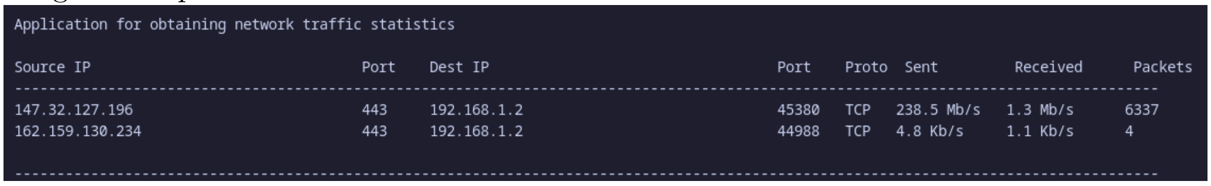
6.2 Wget command

Wget is also a very common command that is used to download files from the internet from the terminal. I used this command to check if the app counts sent/received data correctly.

Command:

```
wget https://download.fedoraproject.org/pub/fedora/linux/releases/41/Workstation/x86_
Fedora-Workstation-L  10% [=======>        ]  245.44M   27.57MB/s
```

Program output:



```
Application for obtaining network traffic statistics

Source IP                               Port    Dest IP                                 Port    Proto  Sent         Received      Packets
----------------------------------------------------------------------------------------------------------------------------------------
147.32.127.196                          443     192.168.1.2                             45380   TCP    238.5 Mb/s   1.3 Mb/s      6337
162.159.130.234                         443     192.168.1.2                             44988   TCP    4.8 Kb/s     1.1 Kb/s      4

----------------------------------------------------------------------------------------------------------------------------------------
```

We can see that the wget command show transfer speed of 27.57MB which corresponds to 231.3Mbit. The app shows 238.5Mbit. I consider this to be OK, because the wget command transfer speed is refreshed every 0.1s where as my app refreshes every 1 second so its almost impossible that these two apps would display speed of the same group of packets.

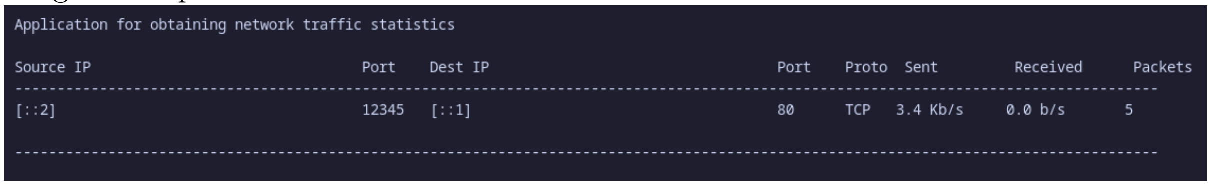
6.3 My own tester

Testing via built in commands was OK, but I wanted more ways to test the IPv6 support. I decided to write my own simple tester on the Go language that would create and send packets and send them to localhost. With my own program, I can control the number of packets per second I can send and also their size.

Command:

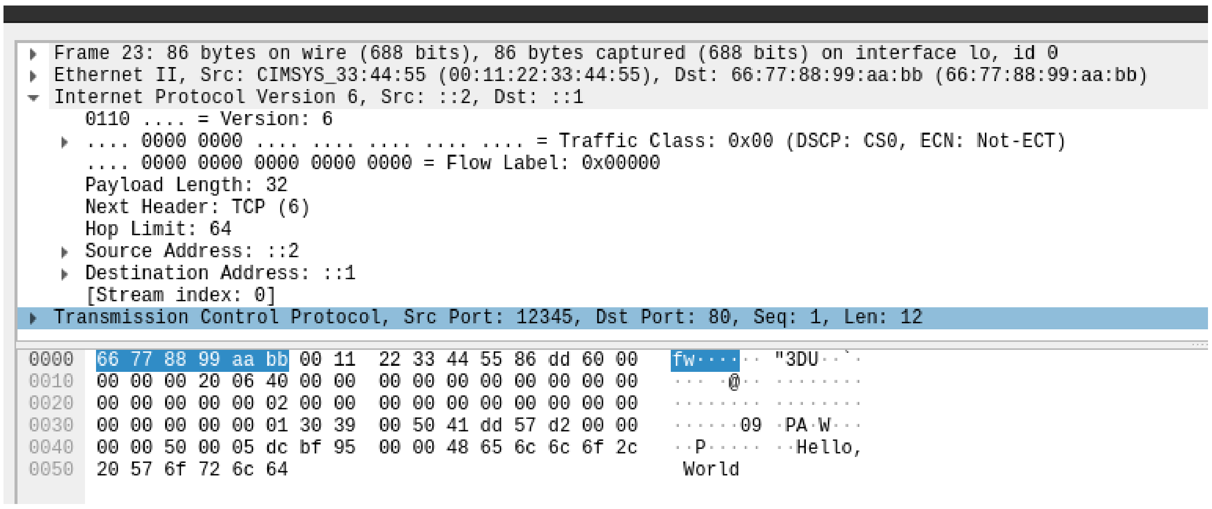
```
   sudo go run main.go
2024/11/18 20:24:18 Sent an IPv6 packet.
2024/11/18 20:24:18 Sent an IPv6 packet.
2024/11/18 20:24:18 Sent an IPv6 packet.
2024/11/18 20:24:19 Sent an IPv6 packet.
2024/11/18 20:24:19 Sent an IPv6 packet.
2024/11/18 20:24:19 Sent an IPv6 packet.
2024/11/18 20:24:19 Sent an IPv6 packet.
2024/11/18 20:24:19 Sent an IPv6 packet.
...
```

Program output:



```
Application for obtaining network traffic statistics

Source IP                               Port    Dest IP                                 Port    Proto  Sent         Received      Packets
----------------------------------------------------------------------------------------------------------------------------------------
[::2]                                   12345   [::1]                                   80      TCP    3.4 Kb/s     0.0 b/s       5

----------------------------------------------------------------------------------------------------------------------------------------
```

Here I've sent an IPv6 packet every 0.2s seconds which equals to 5 packets a second. We can see the same number in the program output. Size of each packet was 688 bits which is 3444 bits per second. Here is one such packet captured in wireshark:



```
▸ Frame 23: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface lo, id 0
▸ Ethernet II, Src: CIMSYS_33:44:55 (00:11:22:33:44:55), Dst: 66:77:88:99:aa:bb (66:77:88:99:aa:bb)
▾ Internet Protocol Version 6, Src: ::2, Dst: ::1
     0110 .... = Version: 6
   ▸ .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
     .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
     Payload Length: 32
     Next Header: TCP (6)
     Hop Limit: 64
   ▸ Source Address: ::2
   ▸ Destination Address: ::1
     [Stream index: 0]
▸ Transmission Control Protocol, Src Port: 12345, Dst Port: 80, Seq: 1, Len: 12

0000  66 77 88 99 aa bb 00 11  22 33 44 55 86 dd 60 00   fw······ "3DU··`·
0010  00 00 00 20 06 40 00 00  00 00 00 00 00 00 00 00   ··· ·@·· ········
0020  00 00 00 00 00 02 00 00  00 00 00 00 00 00 00 00   ········ ········
0030  00 00 00 00 00 01 30 39  00 50 41 dd 57 d2 00 00   ······09 ·PA·W···
0040  00 00 50 00 05 dc bf 95  00 00 48 65 6c 6c 6f 2c   ··P····· ··Hello,
0050  20 57 6f 72 6c 64                                   World
```

Figure 1: Wireshark

References

[1] *RFC 791: Internet Protocol.* (n.d.). IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc79
[2] *RFC 2460: Internet Protocol, Version 6 (IPv6) Specification.* (n.d.). IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc2460
[3] *RFC 793: Transmission Control Protocol.* (n.d.). IETF Datatracker. https://datatracker.ietf.org/do
[4] *RFC 768: User Datagram Protocol.* (n.d.). IETF Datatracker. https://datatracker.ietf.org/doc/html
[5] *RFC 792: Internet Control Message Protocol.* (n.d.). IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc792
[6] *RFC 4443: Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification.* (n.d.). IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc4443
[7] *NCURSES – New Curses.* (n.d.). https://invisible-island.net/ncurses/
[8] LIBPCAP | TCPDUMP & LIBPCAP_. (n.d.). https://www.tcpdump.org/