Politecnico di Torino

Master’s degree in computer engineering

October 2023

Sniffnet

A cross-platform network monitoring tool developed in Rust.

Candidate:
Giuliano Bellini

Relator:
Giovanni Malnati

Un pensiero va a Giacomo, Martina, mamma e papà
(ed anche al Tatolo, ovviamente)

Abstract

One of the peculiar facets of the modern world is to be characterized by the constant and ubiquitous presence of **Internet connectivity**.
In such a context, the activity of **network traffic monitoring** is assuming increasing relevance and is at the foundation of different processes: from detecting potential cybersecurity attacks to troubleshooting usability issues or performing lawful interception.

During the Master's degree at the Polytechnic University of Turin, I had the chance to be involved in an academic project to develop a **network analyzer**, namely software to gather incoming and outgoing data from a computer device and able to help inspect the nature of the exchanged traffic.

The tool went much further than the initial plans, as I fell in love with the development process to the point of working on the application full-time for almost a year straight. What was born as a toy project is today called **Sniffnet** and is one of the most popular network monitoring tools on GitHub, the renowned code hosting platform for version control and collaboration.

Sniffnet is a **cross-platform** application, compatible with all the major operating systems, that stands out for its **ease of use** in allowing whoever to grasp a comprehensive, high-level view of their network activity.
A further distinctive feature of the tool is to be entirely written in **Rust**, a modern programming language to build efficient and reliable software.

The dissertation aims at describing not only the **development process** of the application but also the management activities behind proper software **documentation**, **maintenance**, and **distribution**.

Last but not least, the disquisition includes the main lessons learned during the **GitHub Accelerator**, a program to pioneer new ways for developers to sustainably work on open-source, which Sniffnet had the luck of being part of.

Table of contents

List of figures

List of code snippets

Acronyms and abbreviations

API — Application Programming Interface

ARPANET — Advanced Research Projects Agency Network

AS — Autonomous System

ASCII — American Standard Code for Information Interchange

AWS — Amazon Web Services

CEO — Chief Executive Officer

CI/CD — Continuous Integration and Continuous Delivery

CLI — Command Line Interface

CNAME — Canonical Name

COSMIC — Computer Operating System Main Interface Components

CPU — Central Processing Unit

CSS — Cascading Style Sheets

CSV — Comma-Separated Values

CTO — Chief Technology Officer

DNS — Domain Name Service

DPI — Deep Packet Inspection

FOSS — Free and Open Source Software

FQDN — Fully Qualified Domain Name

GH — GitHub

GIMP — GNU Image Manipulation Program

GNOME — GNU Network Object Model Environment

GNU — GNU’s Not Unix!

GTK — GIMP ToolKit

GUI — Graphical User Interface

HCI — Human-Computer Interaction

HTML — HyperText Markup Language

IANA — Internet Assigned Numbers Authority

ICMP — Internet Control Message Protocol

IP — Internet Protocol

IT — Information Technology

JS — JavaScript

JSON — JavaScript Object Notation

MAC — Media Access Control

MMDB — MaxMind Data Base

OS — Operating System

OSPO — Open Source Program Office

OSS — Open Source Software

PCAP — Packet Capture

PID — Process Identifier

PR — Pull Request

SEO — Search Engine Optimization

SSL — Secure Sockets Layer

SVG — Scalable Vector Graphics

TCP — Transmission Control Protocol

TOML — Tom’s Obvious, Minimal Language

UDP — User Datagram Protocol

UI — User Interface

URL — Uniform Resource Locator

UX — User Experience

VM — Virtual Machine

1. Introduction

Before diving into the Sniffnet development process, I think it's necessary to have a more complete view of the scenario in which it was conceived.

For this reason, the next sections will introduce the subject of **computer networks** from a theoretical point of view, with a focus on some of the already existing **network monitoring tools**, and will present **Rust**, the programming language that has been chosen for the development of Sniffnet.

1.1. Computer Networks

Nowadays people are connected to the Internet almost all-day long: from reading their favorite news website early in the morning to chatting with their friends, navigating social media, or watching the latest TV series on Netflix before going to sleep.
All these activities, which are now taken for granted, are possible thanks to a silent and well-organized **underlying infrastructure** in charge of transporting chunks of information from all around the globe.
The underlying infrastructures which do all the "magic" and allow us to be permanently online are the computer networks.

A **computer network** is a system made of two or more computing devices (often referred to as "nodes") interconnected one with the other transmitting and sharing information [1].

In the late 1960s, the U.S. Department of Defense provided funding for the development of the first packet-switched operational network, known as **ARPANET**.
ARPANET connected the first four computers between four different American Universities in 1969 and managed to link 23 different nodes in 1971.

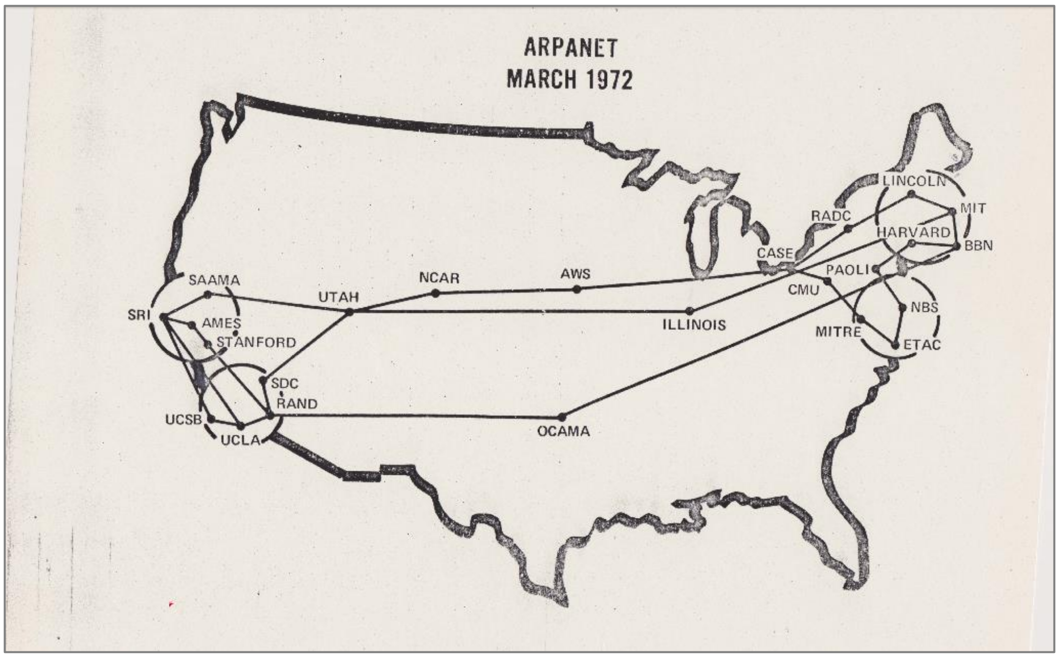


Figure 1.1 *- Representation of the nodes connected by ARPANET, the first public packet-switched computer network.*

Today, we have advanced much from that basic network prototype and we have come up with the **Internet**: a network of networks that connects billions of devices worldwide and which is the center of the contemporary era.

1.1.1. Nodes, links, and communication protocols

The basic blocks at the foundation of a computer network are nodes, links, and communication protocols.

A **node** can be a device of any kind: a computer, a smartphone, a smart TV, a smartwatch, or even a server or router.

A **link** is responsible to connect nodes with each other and to transport information according to the rules defined by a communication protocol.
Links can be of two main types:

- Wired, such as coaxial cables and optical fibers
- Wireless, such as communication satellites and cellular networks

A **communication protocol** is a set of rules that must be followed by all involved nodes to exchange data in a properly structured way.
One of the most used standards nowadays is **TCP/IP** (Transmission Control Protocol / Internet Protocol), which is a suite of communication protocols [2].
The TCP/IP protocol suite serves as an abstraction layer between the routing infrastructure and internet applications.
TCP/IP rules end-to-end communications specifying how data should be divided into packets, addressed, transferred, routed, and received at the destination.
TCP/IP defines four layers, each of which consists of specific protocols and is responsible for a given functionality:

- **Data Link** layer: defines the protocols and hardware required to connect a host to a physical network and to deliver data across it. This layer is restricted to the physical layer boundary which is frequently determined by a router or other similar equipment.
- **Network** layer: also known as the Internet layer, deals with network packets to be transmitted across network boundaries.
- **Transport** layer: responsible for creating and managing end-to-end communications services for applications across the network.
- **Application** layer: defines how high-level applications can access the network to transfer data.

1.1.2. Network packets

Information is exchanged across the network as **formatted units of data**, commonly referred to as "packets".

A packet is a small portion of a larger amount of data and consists of **user data** and **control information**.
The former, also known as **payload**, corresponds to the actual data being exchanged.
The latter, most known as **header**, is a sort of label used to provide details about the packet's content, its origin, and its destination; a network packet usually has more than one header, each attached by a specific protocol and carrying different kinds of auxiliary information [3].

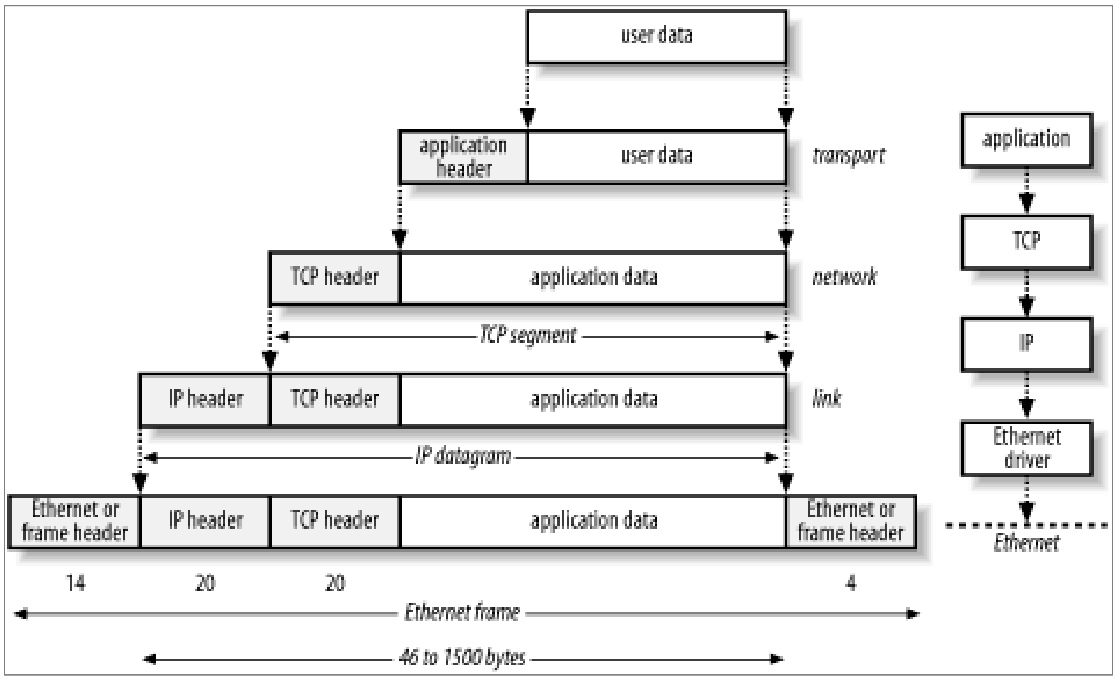


***Figure 1.2** - The typical structure of a network packet.*

The best **path** is then chosen for each packet to get to its destination, according to a **routing algorithm**.
Different packets, part of the same message, aren't forced to follow the same path and the network becomes more effective as a result.
In this way, packets can also be routed around an issue to ensure that the complete message is delivered if a piece of network hardware malfunctions while a message is being sent.

Everything we send or receive via the Internet is exchanged as a long series of packets: every web page we visit, every file we download, and every picture we upload on our social media is shared as groups of transmitting units which are the network packets.

1.2. Network monitoring tools

A packet-switched network like the Internet lends itself very well to being monitored through tools called **packet sniffers** or packet analyzers.
A packet sniffer is a utility in charge of gathering, logging, and monitoring in real-time the transmitted data.
With packet sniffing we refer to the activity of detecting and observing a flow of packets across the network [4].

Packet sniffing has **many practical applications**: network troubleshooting, detection of intrusions, statistical analysis of the data in transit, identification of suspect contents, network usage monitoring, reverse engineering of proprietary protocols, and lots of other purposes, including malicious use.
One of the most common applications is for the monitoring of **bandwidth** and traffic, to examine whether a service is using abnormally high resources.
Networks often incur in issues that need **troubleshooting**: in these scenarios, sniffers are useful for network administrators to find out where the root of the problems resides.
Packet analyzers may also be involved in **penetration testing**. A penetration test is an authorized and simulated cyber-attack, in which a sniffer could be helpful to expose the possible weaknesses of the network's defense system.
While there are several legitimate applications for packet sniffers, hackers frequently use them as well. Cyber-criminals can effectively spy on a network through packet sniffing, and in some situations, they can steal sensitive data like usernames and passwords.

There exist two main types of packet sniffers [5]:

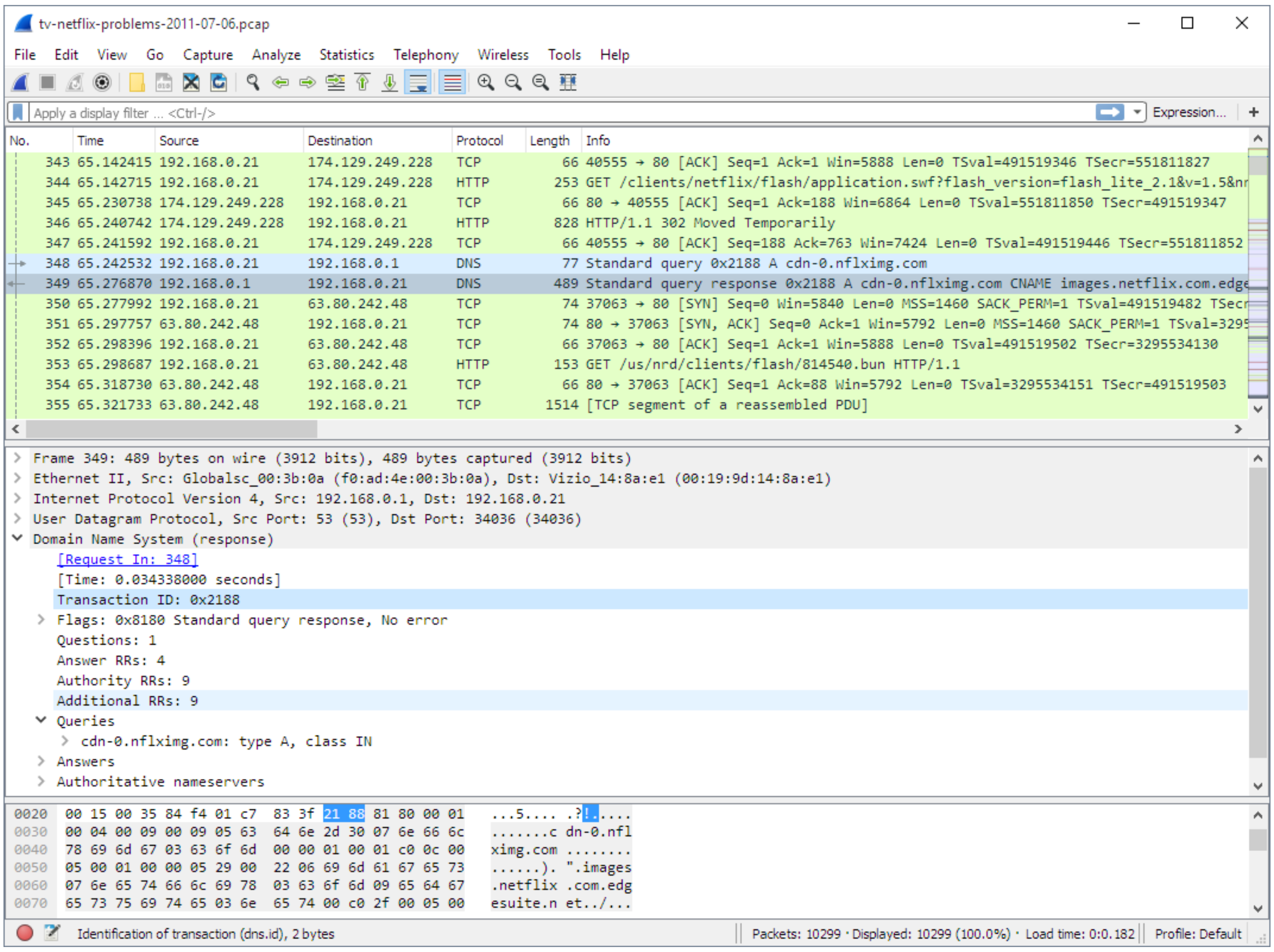
- **Hardware** packet sniffers, which are designed to be manually plugged into a network, directly store the intercepted packets or forward them to a collector for further analysis.
- **Software** packet sniffers, which are pieces of software able to configure a network interface in promiscuous mode, capturing in this way all the traffic flowing through it.

The great majority of the packet sniffers available today are software-based, including the application subject of this thesis and the tools mentioned in the next section.
As anticipated, software sniffers use a dedicated network adapter put in **promiscuous mode**, analyzing each packet, and writing to disk the relevant information for further inspection.
Other devices in the same network are not aware of nor affected by this kind of activity.

1.2.1. Wireshark

When talking about network monitoring tools, it's impossible not to mention **Wireshark** [6].
Wireshark is by far the most popular network protocol analyzer and it's the standard across many commercial and non-profit organizations.
Wireshark, formerly known as Ethereal, is a free and open-source tool that allows to examine the details of traffic at a variety of levels, ranging from connection-level information to the bits that make up a single packet.

Three panels are commonly used by Wireshark to present information.
Individual packets are presented one per line in the top panel.
The middle panel of the tool provides further information about any single frame that is selected in the top panel.
The **raw frame** is then shown in Wireshark's bottom pane, with a representation in hexadecimal on the left and the matching ASCII values on the right.


tv-netflix-problems-2011-07-06.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.   Time      Source          Destination     Protocol Length Info
  343 65.142415 192.168.0.21    174.129.249.228 TCP        66 40555 → 80 [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSval=491519346 TSecr=551811827
  344 65.142715 192.168.0.21    174.129.249.228 HTTP      253 GET /clients/netflix/flash/application.swf?flash_version=flash_lite_2.1&v=1.5&nr
  345 65.230738 174.129.249.228 192.168.0.21    TCP        66 80 → 40555 [ACK] Seq=1 Ack=188 Win=6864 Len=0 TSval=551811850 TSecr=491519347
  346 65.240742 174.129.249.228 192.168.0.21    HTTP      828 HTTP/1.1 302 Moved Temporarily
  347 65.241592 192.168.0.21    174.129.249.228 TCP        66 40555 → 80 [ACK] Seq=188 Ack=763 Win=7424 Len=0 TSval=491519446 TSecr=551811852
  348 65.242532 192.168.0.21    192.168.0.1     DNS        77 Standard query 0x2188 A cdn-0.nflximg.com
  349 65.276870 192.168.0.1     192.168.0.21    DNS       489 Standard query response 0x2188 A cdn-0.nflximg.com CNAME images.netflix.com.edge
  350 65.277992 192.168.0.21    63.80.242.48    TCP        74 37063 → 80 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=491519482 TSecr
  351 65.297757 63.80.242.48    192.168.0.21    TCP        74 80 → 37063 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=3295
  352 65.298396 192.168.0.21    63.80.242.48    TCP        66 37063 → 80 [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSval=491519502 TSecr=3295534130
  353 65.298687 192.168.0.21    63.80.242.48    HTTP      153 GET /us/nrd/clients/flash/814540.bun HTTP/1.1
  354 65.318730 63.80.242.48    192.168.0.21    TCP        66 80 → 37063 [ACK] Seq=1 Ack=88 Win=5792 Len=0 TSval=3295534151 TSecr=491519503
  355 65.321733 63.80.242.48    192.168.0.21    TCP      1514 [TCP segment of a reassembled PDU]
```

```
> Frame 349: 489 bytes on wire (3912 bits), 489 bytes captured (3912 bits)
> Ethernet II, Src: Globalsc_00:3b:0a (f0:ad:4e:00:3b:0a), Dst: Vizio_14:8a:e1 (00:19:9d:14:8a:e1)
> Internet Protocol Version 4, Src: 192.168.0.1, Dst: 192.168.0.21
> User Datagram Protocol, Src Port: 53 (53), Dst Port: 34036 (34036)
v Domain Name System (response)
    [Request In: 348]
    [Time: 0.034338000 seconds]
    Transaction ID: 0x2188
  > Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 4
    Authority RRs: 9
    Additional RRs: 9
  v Queries
    > cdn-0.nflximg.com: type A, class IN
  > Answers
  > Authoritative nameservers
```

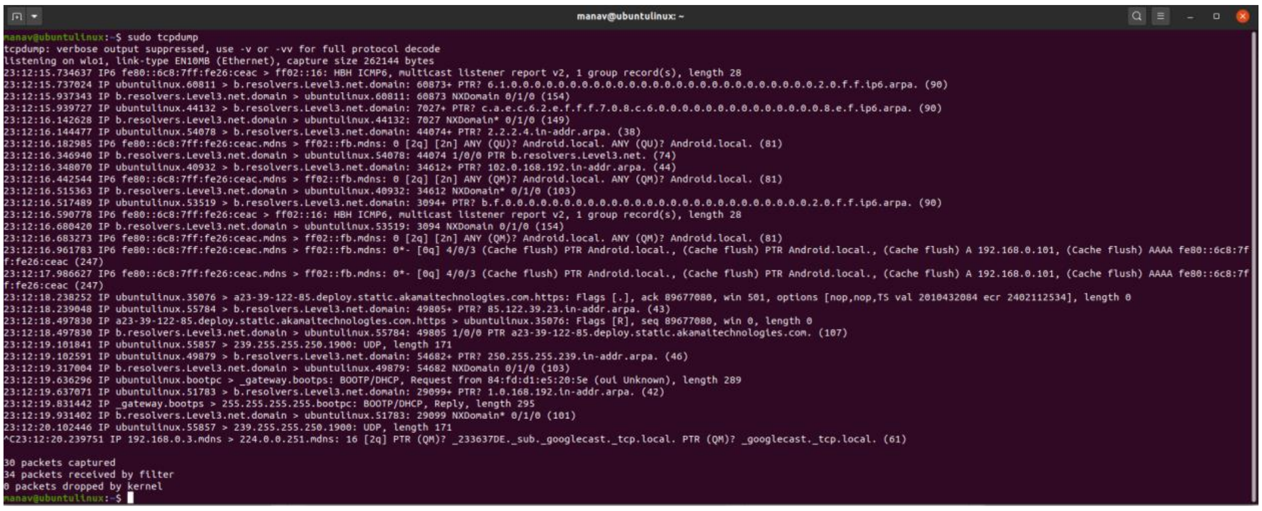
```
0020  00 15 00 35 84 f4 01 c7  83 3f 21 88 81 80 00 01   ...5.... .?!.....
0030  00 04 00 09 00 09 05 63  64 6e 2d 30 07 6e 66 6c   .......c dn-0.nfl
0040  78 69 6d 67 03 63 6f 6d  00 00 01 00 01 c0 0c 00   ximg.com ........
0050  05 00 01 00 00 05 29 00  22 06 69 6d 61 67 65 73   ......). ".images
0060  07 6e 65 74 66 6c 69 78  03 63 6f 6d 09 65 64 67   .netflix .com.edg
0070  65 73 75 69 74 65 03 6e  65 74 00 c0 2f 00 05 00   esuite.n et../...
```

```
Identification of transaction (dns.id), 2 bytes    Packets: 10299 · Displayed: 10299 (100.0%) · Load time: 0:0.182    Profile: Default
```

***Figure 1.3** - Wireshark default view.*

1.2.2. Tcpdump

Another worth-to-mention network analyzer is **Tcpdump** [7], a powerful tool that, instead of coming with a graphical user interface, is in the form of a **command-line** tool.



```
manav@ubuntulinux:~$ sudo tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlo1, link-type EN10MB (Ethernet), capture size 262144 bytes
23:12:15.734637 IP6 fe80::6c8:7ff:fe26:ceac > ff02::16: HBH ICMP6, multicast listener report v2, 1 group record(s), length 28
23:12:15.737024 IP ubuntulinux.60811 > b.resolvers.Level3.net.domain: 60873+ PTR? 6.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.f.f.ip6.arpa. (90)
23:12:15.937343 IP b.resolvers.Level3.net.domain > ubuntulinux.60811: 60873 NXDomain 0/1/0 (154)
23:12:15.939727 IP ubuntulinux.44132 > b.resolvers.Level3.net.domain: 7027+ PTR? c.a.e.c.6.2.e.f.f.f.7.0.8.c.6.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.e.f.ip6.arpa. (90)
23:12:16.142628 IP b.resolvers.Level3.net.domain > ubuntulinux.44132: 7027 NXDomain* 0/1/0 (149)
23:12:16.144477 IP ubuntulinux.54078 > b.resolvers.Level3.net.domain: 44074+ PTR? 2.2.2.4.in-addr.arpa. (38)
23:12:16.182985 IP6 fe80::6c8:7ff:fe26:ceac.mdns > ff02::fb.mdns: 0 [2q] [2n] ANY (QU)? Android.local. ANY (QU)? Android.local. (81)
23:12:16.346940 IP b.resolvers.Level3.net.domain > ubuntulinux.54078: 44074 1/0/0 PTR b.resolvers.Level3.net. (74)
23:12:16.348070 IP ubuntulinux.40932 > b.resolvers.Level3.net.domain: 34612+ PTR? 102.0.168.192.in-addr.arpa. (44)
23:12:16.442544 IP6 fe80::6c8:7ff:fe26:ceac.mdns > ff02::fb.mdns: 0 [2q] [2n] ANY (QM)? Android.local. ANY (QM)? Android.local. (81)
23:12:16.515363 IP b.resolvers.Level3.net.domain > ubuntulinux.40932: 34612 NXDomain* 0/1/0 (103)
23:12:16.517489 IP ubuntulinux.53519 > b.resolvers.Level3.net.domain: 3094+ PTR? b.f.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.f.f.ip6.arpa. (90)
23:12:16.590778 IP6 fe80::6c8:7ff:fe26:ceac > ff02::16: HBH ICMP6, multicast listener report v2, 1 group record(s), length 28
23:12:16.680420 IP b.resolvers.Level3.net.domain > ubuntulinux.53519: 3094 NXDomain 0/1/0 (154)
23:12:16.683273 IP6 fe80::6c8:7ff:fe26:ceac.mdns > ff02::fb.mdns: 0 [2q] [2n] ANY (QM)? Android.local. ANY (QM)? Android.local. (81)
23:12:16.961783 IP6 fe80::6c8:7ff:fe26:ceac.mdns > ff02::fb.mdns: 0*- [0q] 4/0/3 (Cache flush) PTR Android.local., (Cache flush) PTR Android.local., (Cache flush) A 192.168.0.101, (Cache flush) AAAA fe80::6c8:7f
f:fe26:ceac (247)
23:12:17.986627 IP6 fe80::6c8:7ff:fe26:ceac.mdns > ff02::fb.mdns: 0*- [0q] 4/0/3 (Cache flush) PTR Android.local., (Cache flush) PTR Android.local., (Cache flush) A 192.168.0.101, (Cache flush) AAAA fe80::6c8:7f
f:fe26:ceac (247)
23:12:18.238252 IP ubuntulinux.35076 > a23-39-122-85.deploy.static.akamaitechnologies.com.https: Flags [.], ack 89677080, win 501, options [nop,nop,TS val 2010432084 ecr 2402112534], length 0
23:12:18.239048 IP ubuntulinux.55784 > b.resolvers.Level3.net.domain: 49805+ PTR? 85.122.39.23.in-addr.arpa. (43)
23:12:18.497830 IP a23-39-122-85.deploy.static.akamaitechnologies.com.https > ubuntulinux.35076: Flags [R], seq 89677080, win 0, length 0
23:12:18.497830 IP b.resolvers.Level3.net.domain > ubuntulinux.55784: 49805 1/0/0 PTR a23-39-122-85.deploy.static.akamaitechnologies.com. (107)
23:12:19.101841 IP ubuntulinux.55857 > 239.255.255.250.1900: UDP, length 171
23:12:19.102591 IP ubuntulinux.49879 > b.resolvers.Level3.net.domain: 54682+ PTR? 250.255.255.239.in-addr.arpa. (46)
23:12:19.317004 IP b.resolvers.Level3.net.domain > ubuntulinux.49879: 54682 NXDomain 0/1/0 (103)
23:12:19.636296 IP ubuntulinux.bootpc > _gateway.bootps: BOOTP/DHCP, Request from 84:fd:d1:e5:20:5e (oui Unknown), length 289
23:12:19.637071 IP ubuntulinux.51783 > b.resolvers.Level3.net.domain: 29099+ PTR? 1.0.168.192.in-addr.arpa. (42)
23:12:19.831442 IP _gateway.bootps > 255.255.255.255.bootpc: BOOTP/DHCP, Reply, length 295
23:12:19.931402 IP b.resolvers.Level3.net.domain > ubuntulinux.51783: 29099 NXDomain* 0/1/0 (101)
23:12:20.102446 IP ubuntulinux.55857 > 239.255.255.250.1900: UDP, length 171
^C23:12:20.239751 IP 192.168.0.3.mdns > 224.0.0.251.mdns: 16 [2q] PTR (QM)? _233637DE._sub._googlecast._tcp.local. PTR (QM)? _googlecast._tcp.local. (61)

30 packets captured
34 packets received by filter
0 packets dropped by kernel
manav@ubuntulinux:~$
```

***Figure 1.4** - Tcpdump, a command-line network monitoring tool.*

As default, if launched without additional options, Tcpdump prints out the summary of the captured packets.
If further parameters are passed to Tcpdump via the command line, it'll be able to log the intercepted packets in an output file or filter traffic according to some user-defined rules.

The major limitation of this tool is that it's not as immediate to use as a GUI application, but in some cases, it can also be considered a point of strength: being a CLI means to be more portable, making it possible for network administrators to access devices even from remote locations.

1.3. The Rust programming language

Rust is the programming language used to develop Sniffnet in its entirety, from its business logic aspects to its graphical user interface.
It's not so common to use Rust also for frontend development: GUI libraries written in Rust are not many and most of the existing ones are not in their stable release yet (including Iced, the library I decided to use).

Many reasons brought me to this choice; I'll try to summarize them in the next paragraphs, introducing the language and its main distinctive features.

Figure 1.5 *- Logo of the Rust programming language.*

1.3.1. History

Rust is a modern, general-purpose programming language originally designed by Mozilla Research employee **Graydon Hoare** in 2006.
As part of the ongoing development of the Servo experimental browser engine, Mozilla started financing the project in 2009 and officially announced it one year later. [8]

The language rapidly evolved between 2010 and 2014: its type-system underwent major changes with the release of classes, and traits were added as a means of inheritance shortly after.
The first stable release of the language, Rust 1.0, was then published in May 2015.

Some months before the stable release, Andrew Binstock, the editor-in-chief of Dr. Dobb's Journal, commented on Rust saying that it was "*widely viewed as a remarkably elegant language*" and that it had good chances of becoming the **successor of C++**.

In February 2021, the **Rust Foundation** was born as a joint effort of Google, Microsoft, Huawei, AWS, and Mozilla.
Two months later, Google announced its support for Rust within the Android project, together with C++.

As of today, the chat platform Discord uses Rust to speed up its system, Dropbox uses it to sync files, and Cloudflare uses it to process about 20% of the total internet traffic, just to name a few practical applications of the language.

To highlight Rust's adoption even more, we just need to consider that the language has been elected as the **most loved** programming language by the annual Stack Overflow developer survey [9] for seven years straight, from 2016 to 2022.

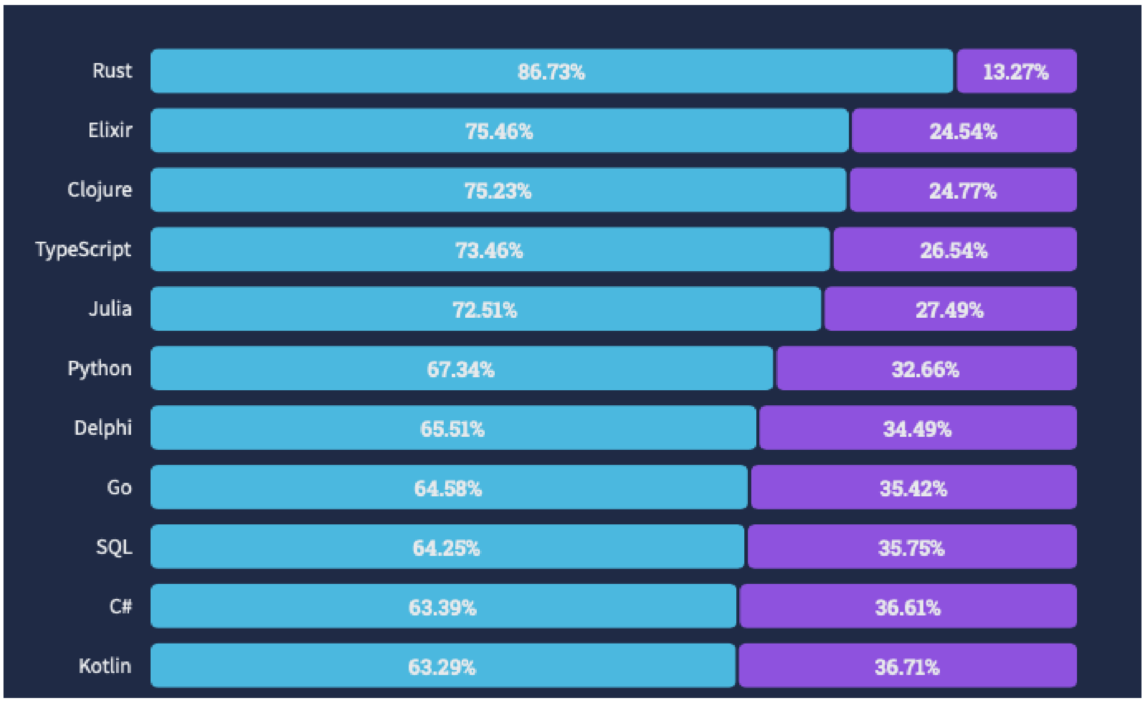


Figure 1.6 *- Rust as the most loved programming language of 2022, with 87% of developers saying they want to keep using it. Source: Stack Overflow 2022 Annual Developer Survey. [9]*

1.3.2. Main features

One of Rust's most interesting features is its management of memory and the **absence of a garbage collector**.
Many programming languages, including Java, use a garbage collector to ensure that unnecessary items will disappear from memory sooner or later at a certain point in time.
Garbage collectors are often appreciated because they don't leave the burden of managing memory to the programmer, but they have different drawbacks.
One of the disadvantages is the possibility of memory leaks: situations in which there are objects allocated in the heap that are no longer used, but the garbage collector is unable to deallocate them; memory leaks should be avoided because they block memory resources and degrade system performance over time.

Instead, Rust guarantees **memory safety** by leveraging the concept of **ownership**: memory space is owned by variables and can be temporarily borrowed by other variables.
A part of Rust's compiler, the **borrow checker**, ensures that references are always valid.

The borrow checker can detect where data needs to be initialized and where it needs to be released (or dropped, in Rust terms), by monitoring where variables are utilized throughout the program and by adhering to a set of criteria. [10]

Another interesting feature of the language is the possibility of having **zero-cost abstractions**.
This characteristic means that using higher-level programming concepts does not come up with a run-time cost, only affecting the compile-time.
Rust compiler is capable of translating statements to the most optimized form of assembly code possible, hence not encumbering the program's execution speed or its memory usage.

Rust is a **strongly typed** language, requiring each variable to be statically typed: there are fewer potential ways for the program to fail at runtime when more checks happen at compile-time.
Rust makes things easier for users thanks to a powerful **type-inference** system, which automatically detects the type of an expression and consequently allows to work with fewer or no type annotations at all.

Rust also provides an advanced **pattern matching** feature, to give more control and flexibility over the program's execution flow.
Rust's pattern syntax allows to match against literals, named variables, ranges of values and permits the destructuring of enums, structs, and tuples.
The **match** condition, part of the language, takes care that a pattern covers every possible value, making it impossible to compile a program otherwise.

What further distinguishes Rust from most programming languages is its capability to obtain **fearless concurrency**.
We speak about concurrent programming when different pieces of a program execute independently.
Concurrent programming is becoming more and more relevant as many computers nowadays have multiple processing units, but at the same time it has historically been error-prone due to its difficult management.
Moreover, errors related to concurrency are often difficult to reproduce, since it's not trivial to recreate the same exact circumstances when two or more threads are running in parallel.
Many concurrency issues in Rust are compile-time errors rather than run-time faults thanks to the concept of ownership and strict type checking.
As a result, programmers can fix their programs while developing rather than after they have already been shipped to production and are able to write code that is free of subtle bugs.

1.3.3. The state of GUIs in Rust

Graphical User Interfaces are intuitive, visual front-ends for interacting with programs.
Because of its expressiveness and high-level abstractions, Rust is theoretically a good candidate for creating sophisticated and complex user interfaces.
Unfortunately, there isn't much agreement on the ideal abstractions to use, and most of the existing GUI libraries for Rust have not reached version 1.x yet. [11]

Most of the times, programmers tend to use Rust just to implement their application’s backend, while they are more inclined to use well-established frameworks such as React, Angular, or Vue to develop their front-ends.

This is made possible by frameworks like **Tauri**:
“Tauri is a framework for building tiny, blazingly fast binaries for all major desktop platforms.
Developers can integrate any front-end that compiles to HTML, JS, and CSS for building their user interface.
The backend of the application is a Rust-sourced binary with an API that the front-end can interact with”. [12]

While this is of course a reasonable solution, since most of the application’s logical complexity often resides in its back-end, I didn’t like the idea of losing Rust stability and safety guarantees while implementing my user interface.
For this reason, I ended up searching for a library that could satisfy my needs, allowing me to **entirely develop my application in Rust**.

Despite, as previously anticipated, most of the available options are not so mature yet, libraries like Dioxus, Egui, and Iced are actively maintained, rapidly evolving, and complete enough for my use case.
After having considered all the possibilities, I decided to use **Iced**, a cross-platform GUI library for Rust which will be better discussed later.

2. The idea

How was Sniffnet born?
What are the ideas behind it?
What makes this application unique?
What are its purposes?

This chapter will answer such questions, trying to reconstruct the **stream of ideas** at the foundation of the application's conception.

"Let your ideas and thoughts give you inspiration. All creativity comes from your imagination: you first imagine and then you create" - Catherine Pulsifer.

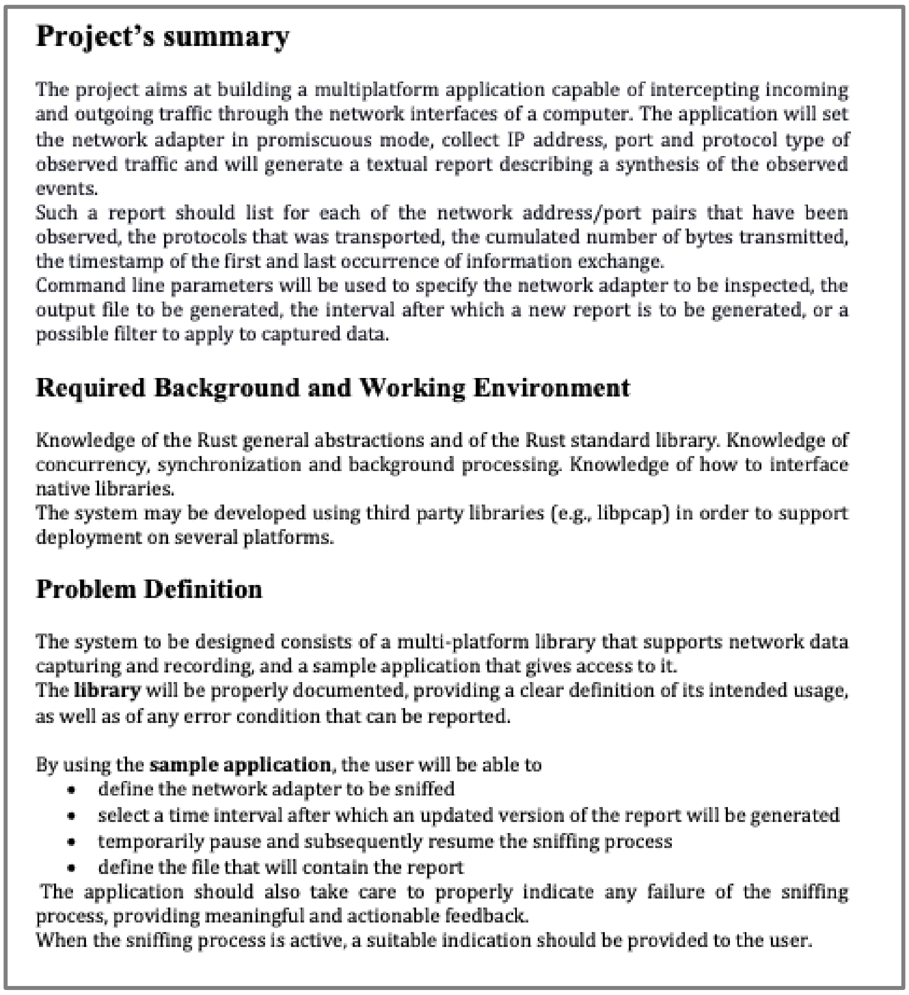
2.1. How it started

Sniffnet, originally simply called *packet_sniffer*, was born as an academic project in the scope of the course of System and Device Programming, which I had the luck to follow during the first year of my Master's degree at the Polytechnic University of Turin.

The **System and Device Programming** course accounts for 10 CFUs and is split into two modules: one about the design principles of operating systems and the other describing the interfaces for system programming and resource management.
The latter is held by Professor Malnati, the relator of the thesis, who introduced us to the **Rust** programming language.
Rust was chosen over C++ for the first time since the establishment of the course, and I consider it a very lucky coincidence, since one of the main reasons I kept developing Sniffnet, and one of the most likely motivations for its success, is exactly because it's written in Rust.

The professors of System and Device Programming propose every year **optional projects** on advanced topics about either of the two parts of the course.
One of the available projects was about developing a command line interface with the functionalities of a simple **network analyzer** and this option immediately caught my interest.
I've always wondered how analyzers such as Wireshark work under the hood, and developing a networking tool was the best way to finally discover it.
After having talked to my friend and colleague **Cristiano Canepari**, we agreed that it would have been cool to spend some time on this project during the summer, so we decided to dive into it.

More precisely, the chosen project's goal was to develop a command line tool able to intercept network data flowing through the interfaces of a computer and generate a textual output summarizing the collected information.
Such a program was required to receive different parameters from the command line to let the user choose a defined network adapter or specify filters to apply to the observed traffic.



Project's summary

The project aims at building a multiplatform application capable of intercepting incoming and outgoing traffic through the network interfaces of a computer. The application will set the network adapter in promiscuous mode, collect IP address, port and protocol type of observed traffic and will generate a textual report describing a synthesis of the observed events.
Such a report should list for each of the network address/port pairs that have been observed, the protocols that was transported, the cumulated number of bytes transmitted, the timestamp of the first and last occurrence of information exchange.
Command line parameters will be used to specify the network adapter to be inspected, the output file to be generated, the interval after which a new report is to be generated, or a possible filter to apply to captured data.

Required Background and Working Environment

Knowledge of the Rust general abstractions and of the Rust standard library. Knowledge of concurrency, synchronization and background processing. Knowledge of how to interface native libraries.
The system may be developed using third party libraries (e.g., libpcap) in order to support deployment on several platforms.

Problem Definition

The system to be designed consists of a multi-platform library that supports network data capturing and recording, and a sample application that gives access to it.
The **library** will be properly documented, providing a clear definition of its intended usage, as well as of any error condition that can be reported.

By using the **sample application**, the user will be able to

- define the network adapter to be sniffed
- select a time interval after which an updated version of the report will be generated
- temporarily pause and subsequently resume the sniffing process
- define the file that will contain the report

The application should also take care to properly indicate any failure of the sniffing process, providing meaningful and actionable feedback.
When the sniffing process is active, a suitable indication should be provided to the user.

Figure 2.1 *- The original outline for the network analyzer optional project.*

The first commit on the GitHub repository of the project is dated back to the 5th of August 2022, and it took Cristiano and me about two weeks of programming to produce the **first functional version** of the program, which was originally published on crates.io (the Rust package registry) on August the 17th.
We spent the next few days applying some more minor improvements and patches until we finally **presented the project** to Professor Malnati during the first week of September.

After the project was discussed, we were basically done with it and could move on, but I wasn't fully satisfied with the outcome and decided to **keep developing it on my own**.
I had a lot of fun implementing the first version of the command line application and I saw in it some potential: *"If this project is so fascinating to me, maybe other people could find it interesting as well"* I thought.

2.2. The desire for a modern, simple, and intuitive tool

2.2.1. The unsatisfaction with the project outcome

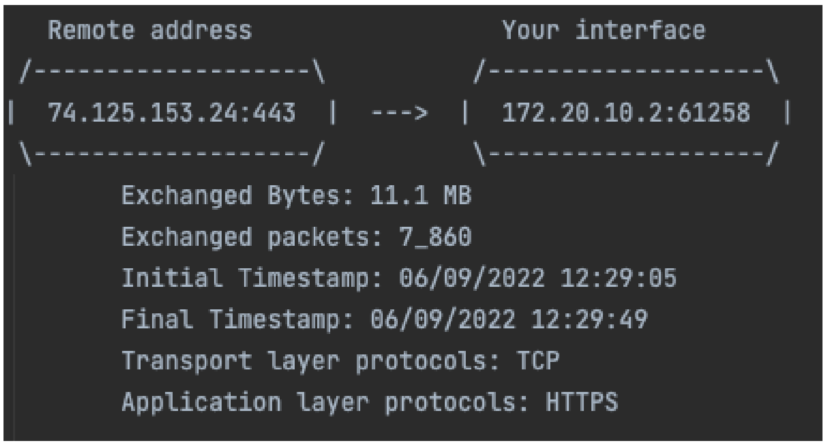
My partial unsatisfaction with the delivered project and the feeling it was still incomplete mainly derive from the fact that, despite the tool features being indeed interesting, in practice it was **boring and cumbersome to use**.

If, for example, a user wanted to inspect the network adapter 'en0', filtering data by IP version 4 and TCP transport protocol, this command had to be used:

```
$ sniffnet –a en0 –n IPv4 –t TCP
```

Code snippet 2.1 *- Command line instruction to launch the first version of the app with some arguments.*

It's not comfortable for an average end user to open a terminal interface and type the desired arguments to pass to the program and, even if it was, the information was presented in an unfriendly way, in the form of a long textual report that didn't make details easy to be grasped.



```
  Remote address                  Your interface
 /-------------------\           /-------------------\
|  74.125.153.24:443  |  --->  |  172.20.10.2:61258  |
 \-------------------/           \-------------------/
        Exchanged Bytes: 11.1 MB
        Exchanged packets: 7_860
        Initial Timestamp: 06/09/2022 12:29:05
        Final Timestamp: 06/09/2022 12:29:49
        Transport layer protocols: TCP
        Application layer protocols: HTTPS
```

Figure 2.2 *- A fragment of the original textual report generated by the program.*

Being a technical guy with a strong passion for computer science, I was able to partially appreciate what the tool did produce and gain some interesting information from its output, but I was conscious that the whole process had a lot of room for improvement. Data available from the intercepted network packets are extremely valuable and carry a lot of particulars, but those data **must be presented in a meaningful way** in order to be better analyzed and to get the most out of them.
Using a raw textual file to summarize such a rich collection of data was definitely not the most appropriate choice for a tool I wanted to use directly, without involving external data parsers.

2.2.2. The drive toward a more comfortably usable tool

Basically, I was interested in exploring the collected network packets in an **immediate**, **comfortable**, and **visually pleasing** way.
Since I had some spare time available in the following months, I decided it was the perfect occasion to build a tool to satisfy my curiosity while gathering network statistics.

During that period, I also had the chance to follow the Human-Computer **Interaction** (HCI) course held by Professor Luigi De Russis, which gave me a solid background about the concepts of usability and user-centered design.
One of the goals of the HCI course was to explain how to design suitable interfaces and interactions so that people can use a given technology with pleasure rather than frustration.
Particularly interesting was to learn that considering the users' needs, wants, and limitations is a crucial phase during the development of a system, even if it's often neglected.
Adopting such an approach can bring several different benefits to the resulting system, in terms of:

- **usability**, the extent to which the system can be used to achieve goals with effectiveness, efficiency, and satisfaction in a specific context of use.
- **usefulness**, the system's ability to provide functionalities that users really need.
- **learnability**, the overall ease of use of the system.

Driven by those principles, I wanted to expand and improve Sniffnet to create an application that wasn't just functional but also comfortably usable and effectively used.
One of the first steps in this direction has been thinking of a possible **graphical representation** of the already available data.
For this purpose, I decided to produce an SVG picture of the observed traffic rate in addition to the textual report.
Even if it was a minor change with respect to the original version, I felt like it was a revolution: I was finally able to *see and touch* what I just imagined during the previous weeks of work on the project.

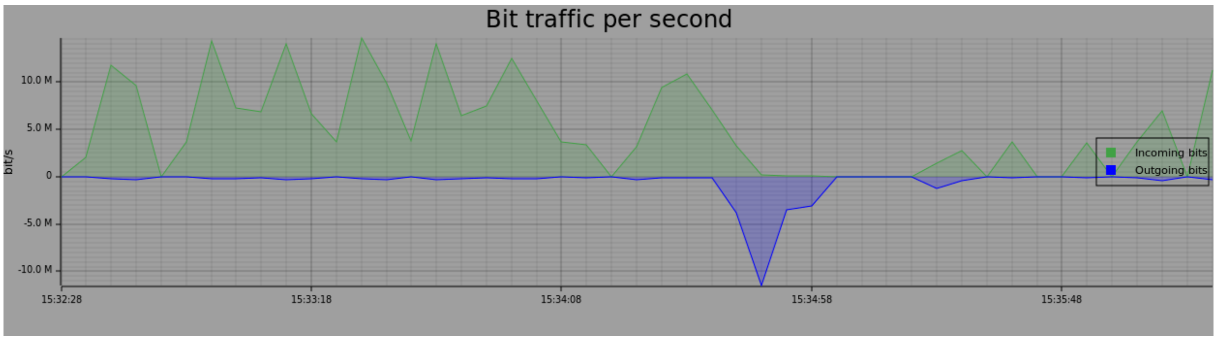


***Figure 2.3** – SVG chart about the traffic rate, generated as output of the ongoing analysis.*

The excitement I had from this small improvement made me think about the limitations of having structured the program as a command line interface.
Arrived at that point, it’s been natural for me to start considering a complete restructuring of the application, consisting of the development of a whole **graphical user interface**, which would have made it possible to obtain a significant enhancement of the program’s expressiveness and flexibility.
A graphical user interface, in addition to allowing a more immediate and natural representation, is also more suitable to build an application in which the user is free to move, act, and perform more complex interactions.

In the following are reported the very first raw and basic **sketches** of what I had in mind, which will later become the GUI for the first stable release of Sniffnet:

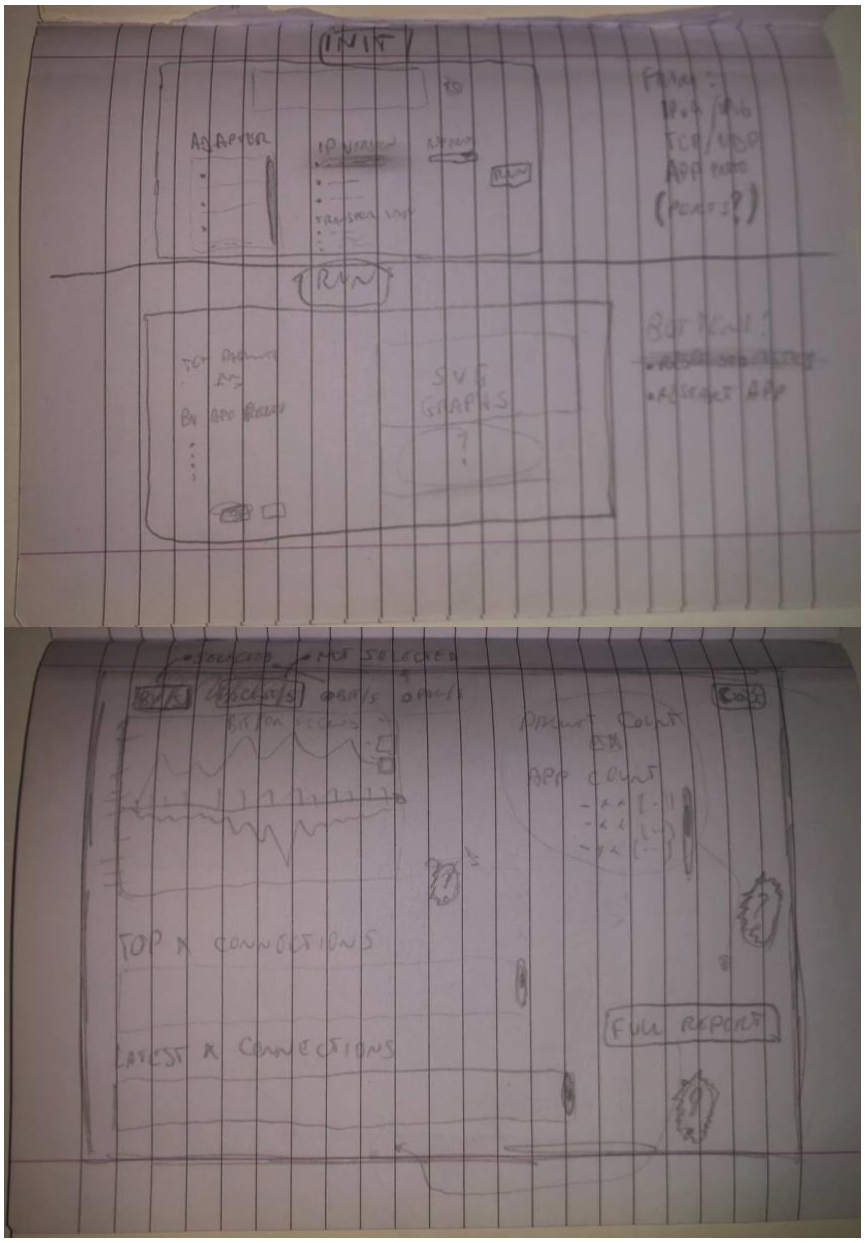


Figure 2.4 *- Elementary sketches representing the initial idea around Sniffnet GUI.*

2.2.3. Purposes, peculiarities, and target users

Having in mind a possible visual representation, the problem of defining **what the tool wanted to achieve in practice** was still open.

One element that played an important role in defining Sniffnet's purposes is that nowadays a considerable fraction of the Internet traffic exchanged is **encrypted**: Google reports that **95%** of traffic to its search engine is encrypted since 2018. [13]
The same data from Google also state that just four years prior, in 2014, the amount of encrypted traffic was just 50%.

Encryption is a modern way of protecting electronic information; it consists in converting the data to protect into an unintelligible form – called **ciphertext** – such that it can only be translated back into an interpretable form – referred to as plaintext – with the use of a key. [14]
Depending on the scenario, different kinds of encryption are usually applied:

- encryption **in transit**, to protect data from an end user and a third-party server (for example in the case of an e-commerce website).
- **end-to-end** encryption, to make confidential the information exchanged between two or more end users (such as in the case of instant messaging services).
- encryption **at rest**, to preserve the secrecy of information when not in transit (used for instance by hard disks to protect the stored data).

Putting in place these practices is important to avoid that **unintended recipients** can intercept and manipulate stored information or messages traveling across the network: the possible consequences deriving from access to private information can span from identity theft to financial fraud or personal harm.

How is encryption related to the development of Sniffnet?
As anticipated, the considerable growth in the adoption of cryptography techniques makes most of the network packets' content not understandable.
Therefore, performing **Deep Packet Inspection** (or DPI – the practice of analyzing a packet's content) didn't seem a viable solution.
What appeared more meaningful was to monitor and categorize traffic at a **higher-level**: instead of inspecting every single packet, it's convenient to analyze the flow of data by adopting a **connection-oriented** approach.

A connection is what identifies a link between two endpoints involved in a network data exchange, and it's usually characterized by 5 parameters (usually referred to as **5-tuple**): source and destination IP addresses, source and destination transport ports, and transport protocol in use.

This information alone is enough to determine several characteristics about the observed stream of data, such as the host geolocation, domain name, and administrative entity behind it.

Another key aspect in defining the application behavior is that I had in mind to create a tool that could be **easily understood and used by everyone**, not just network experts and technical people.
My ultimate goal was to implement a software based on two main pillars:

- **satisfy the curiosity** of those people for whom other network monitoring tools are not comfortable to be used.
- provide **technical features** that more experienced users could find valuable.

Most network analyzers available consist of advanced tools aimed at troubleshooting network issues and they don’t fit well with the **average user's need** for software that is **simple**, **quick**, and **painless** to use.

Guided by these intuitions, I ended up designing an application with higher-level features, such as real-time charts, custom notifications, and a fresh look, that could be appreciated by **tech enthusiasts** as well as the **general public**.

3. The development process

This chapter is thought to be the core of the thesis, as it describes the **implementation** of the app from a technical point of view, delving into the programming strategies adopted and the code's most relevant modules.

The latest version of the application is made of almost 20 thousand lines of code and many changes were introduced from the previous versions: despite it's not possible to dive into every single detail, the chapter will try to cover all the **main code-related aspects**.

The dissertation is organized in **chronological order**, with each subchapter being dedicated to the functionalities introduced in specific versions of the software.

3.1. From the beginning up to version 0.5: the command line interface

3.1.1. Command line options

As previously stated, the application initially consisted of a **CLI** producing a textual report about the observed network traffic, as required by the academic project specifications.

Different options were available to setup parameters useful for the analysis and to customize the configurations.
Such options came in the form of **command line arguments** to be supplied by the users via the terminal interface and they were parsed by Sniffnet with *clap*.
Clap [15] is a command line argument parser for Rust that easily permits to specify short and long names, type, default value, and further constraints for each of the available options.

The list of the possible arguments is reported in the following:

- `-a, --adapter`: specifies the name of the network adapter to be inspected; if omitted the default adapter is chosen.
- `--app`: filters packets based on the provided application layer protocol.
- `-d, --device-list`: prints list of the available network interfaces. Immediately terminates the program.
- `-h, --highest-port`: specifies the maximum port value to be considered; if omitted there is no port higher bound.
- `-i, --interval`: sets the interval of time between report updates (value in seconds).
- `-l, --lowest-port`: specifies the minimum port value to be considered; if omitted there is no port lower bound.
- `-m, --minimum-packets`: sets the minimum value of transited packets for a connection to be printed in the report.
- `-n, --net`: filters packets based on the provided IP address version (IPv4 or IPv6).
- `-o, --output-file`: specifies the name of the output file containing the textual report; if omitted the file name is `sniffnet_report.txt`
- `-t, --trans`: filters packets based on the provided transport layer protocol (TCP or UDP).

Several **constraints** had to be met by the user-provided options for the application to correctly start, otherwise a proper informative error message was generated.
Sniffnet, supported by *clap*, had to make sure that each argument respected its type, that the supplied adapter existed in the PC, and that the specified filters were valid.

3.1.2. User interactions during the execution

Following the project requirements, the program had to offer the possibility to be **temporarily paused** and allow users to **later resume** the sniffing process.

The application originally consisted of three **different threads** and, to pause an ongoing analysis, it was necessary to properly coordinate all the execution flows.
To make it possible, an enum `Status` was defined to track the current condition of the application (running, paused, or stopped).
Defining a status was not sufficient since to correctly share it between the various threads, auxiliary primitives were needed: for this reason, the application state was associated with a `Mutex` and a `Condvar`.
The `Mutex` is a construct to guarantee that a certain resource is only accessed by one thread at a time, while the `Condvar` is a variable that makes it possible for a thread to block without consuming CPU cycles until a given condition occurs.
Changes of status were defined by **user inputs** and were captured by Sniffnet putting the terminal in raw mode and creating a synchronous reader.

In this way, it was possible for the main thread to **update the application status** when specific inputs were supplied and to inform all the execution flows that they had to pause, resume, or stop.

```
fn set_status_by_key(status_pair: Arc<(Mutex<Status>, Condvar)>) {
    let _raw = RawScreen::into_raw_mode();
    let mut reader = input().read_sync();
    let cvar = &status_pair.1;
    loop {
        if let Some(event) = reader.next() { // Blocking call
            let mut status = status_pair.0.lock().unwrap();
            match event {
                InputEvent::Keyboard(KeyEvent::Char('p')) => {
                    if *status == Status::Running {
                        *status = Status::Pause;
                    }
                }
                InputEvent::Keyboard(KeyEvent::Char('r')) => {
                    if *status == Status::Pause {
                        *status = Status::Running;
                        cvar.notify_all();
                    }
                }
                InputEvent::Keyboard(KeyEvent::Char('s')) => {
                    *status = Status::Stop;
                    cvar.notify_all();
                    return;
                }
                _ => { /* Other events */ }
            }
        }
    }
}
```
Code snippet 3.1 *- Function to wait for user inputs and update the app status accordingly.*

Before entering the function to wait for user inputs, the main thread oversaw generating the two other execution flows: the first was dedicated to **parsing network packets**, while the second had the task of constantly **updating the textual report**.

3.1.3. The network traffic analysis

The thread in charge of parsing packets from the network can be considered **the heart of the application** since it's responsible to collect and organize the material on which all data structures and modules of the program rely.
Its implementation evolved over time, being integrated with new features and capabilities, but the original skeleton from the earlier versions remained stable.

As previously anticipated, due to modern encryption techniques, I decided to focus the analysis more on the **packets' header** than the packets' payload, and this is exactly the module with the task of intercepting every packet, reading its header, and collecting information in ad-hoc structures.

Two Rust crates are particularly useful for reading and interpreting network data: **Pcap** [16] and **Etherparse**. [17]
The former is a cross-platform packet capture library that supports the creation and configuration of capture contexts, while the latter is intended to provide parsing functionalities to easily permit accessing specific portions of a packet.

More specifically, Pcap lets accessing PC network devices and putting them in **promiscuous** mode to examine all the incoming and outgoing flows of data.
Additional options can be specified before starting a capture session; in Sniffnet's case, a `snaplen` equal to 256 is specified (defining the maximum length of each captured packet) because I was only interested in the headers, and `immediate_mode` is turned on, allowing to deliver packets as soon as they arrive, without batching them.

After having set up the capture, it's sufficient to invoke the `next_packet()` method, which blocks the caller until a packet is returned from the capture handle.
This function is called inside a **loop**, to keep receiving packets as they arrive; Pcap places the captured packets into a **buffer** of finite length and it's therefore important to make the process efficient, to avoid discarding data in presence of high traffic loads.

As soon as a packet is captured, it is **decoded** into the different headers by the `from_ethernet_slice(...)` method offered by Etherparse.
The returned value consists in a struct whose fields correspond to the headers of the packet (link, IP, and transport), each with its relevant attributes.

The following **headers' parameters** are checked by Sniffnet: source and destination MAC addresses (from the data link header), payload length, source and destination IP addresses (from the network header), source and destination ports (from the transport header).
Furthermore, Sniffnet relies on headers to determine the **IP version** of the packet and the **transport protocol** used.

Based on the transport ports, Sniffnet also tries to guess the **upper layer service** involved: application layer protocols are inferred from the port number following the convention maintained by **IANA**, the Internet Assigned Numbers Authority. [18]
Despite the IANA is responsible for defining the official assignments of port numbers for specific use cases, it must be kept into account that this is just a convention and that many unofficial uses of **well-known ports** occur in practice.
I decided to include, in the list of supported application layer protocols, the most common services to let users have an idea about the possible kind of exchanged traffic.

The collected information is used to decide if the packet respects all the **filters** specified: if affirmative, the statistics related to the packet are **saved** for further examination by the user.

More specifically, each packet's data are aggregated with data from the corresponding connection.
A **unidirectional network connection** is defined by the transport protocol used and the pair of sender and receiver, each identified by a specific IP address and transport port.
To adequately store information related to a network connection, proper data structures are needed: to this purpose, I defined the `AddressPortPair` struct, representing a unique unidirectional connection, and the `InfoAddressPortPair` struct, which encapsulates statistics related to a specific connection.

```
pub struct AddressPortPair {
    /// Source IP address.
    pub address1: String,
    /// Source port number.
    pub port1: u16,
    /// Destination IP address.
    pub address2: String,
    /// Destination port number.
    pub port2: u16,
    /// Transport layer protocol (TCP or UDP).
    pub trans_protocol: TransProtocol,
}
```

Code snippet 3.2 *- Definition of the AddressPortPair struct.*

```
pub struct InfoAddressPortPair {
    /// Amount of bytes transmitted between the connection.
    pub transmitted_bytes: u128,
    /// Amount of packets transmitted between the connection.
    pub transmitted_packets: u128,
    /// First occurrence of information exchange.
    pub initial_timestamp: String,
    /// Last occurrence of information exchange.
    pub final_timestamp: String,
    /// Application layer protocol carried by the connection.
    pub app_protocol: AppProtocol,
}
```
Code snippet 3.3 *- First version of the struct encapsulating statistics related to a connection.*

The two structures were used respectively as key and as value of a `HashMap` associated with a `Mutex` to be shared between threads without incurring in race conditions.

An additional map (`HashMap<AppProtocol, u128>`) was also defined to save the number of captured packets divided by application layer protocol.

Other helper functions were designed to carry out supplementary tasks, such as determining the local address of a connection, recognizing multicast addresses, and converting IPv6 addresses from a decimal format to the normal hexadecimal notation, to make them conformant to the best practices of representation.

3.1.4. The program output

Up to version 0.3.2, the output of the program consisted of a **textual report** listing all the **observed connections with the related statistics**: the source and destination addresses and ports, the number of transmitted bytes and packets, the carried protocols, and the initial and final timestamps of information exchange.
The `Display` trait was implemented for both the structs previously reported, to allow for a human-readable representation of such data.

By default, the output file, named `sniffnet_report.txt`, was placed in the same directory from where Sniffnet was launched, but the location and file name could be customised via CLI options.
The report was updated every five seconds to include the latest network data, but the user was able to specify an additional argument to arbitrarily change the update frequency.

An additional **thread** was designed to exclusively operate on the report file, creating and periodically updating it.

The thread responsible for this task operated inside a **loop** made of the following steps:

1. The execution flow is put into **sleep** for an amount of time corresponding to the report update frequency; during this phase, the thread is blocked without consuming CPU time.
2. As soon as the thread wakes up, it requests for the acquisition of the `Mutex` encapsulating the shared map related to the gathered network traffic; until this thread will **hold the lock**, the flow in charge of parsing network data cannot proceed and new packets are stored inside the Pcap buffer.
3. In the earliest releases of the program, the connections contained in the map were sorted following a decreasing order of exchanged packets.
4. Each connection present in the map is **printed** in a formatted way on the output file.
5. The **lock is released** to permit the other thread resuming its activity.
6. The application **status** is checked, and different actions are taken accordingly:
    - If the program is **running**, the thread re-enters the loop from step 1.
    - If the program has been **paused**, the thread blocks on the corresponding condition variable until the main execution flow will notify it to wake up.
    - If the program has been **stopped**, the thread returns and will be joined in the main thread that waits for it (this is essential to avoid killing the program while the report is being written).

What is particularly critical in this context is **minimizing the time elapsed** between steps 2 and 5: if the lock on the map of network connections is held for too long, the **Pcap buffer would saturate,** and new incoming or outgoing packets would not be recorded by the program (they would be *dropped*, to say it in the networking slang).

The first version of the software used a very basic approach to update the report, since **the file was rewritten from scratch** at every round of the loop.
This strategy was adopted to avoid appending data to the previous output, which would have made the overall size of the file blow.
However, this approach was **not efficient and scalable** at all: as the map grows during a run of the program, to include new connections, printing all its content obviously takes longer and longer.
With this concern in mind, I decided to measure the time needed to update the report and it was sadly confirmed that my worries were tangible: the program took about 600 milliseconds to update the output corresponding to a map of 3500 entries. [19]
Even if for a human it’s a negligible amount of time, I was aware that from the point of view of a CPU it’s like ages: millions of operations could be performed, and thousands of packets could be exchanged in such a time frame.

The first optimization of this process was introduced using a `BufWriter`.
The initial version of the report was updated inside a `for` cycle iterating on each of the map entries, and the write was directly performed on the `File` object: every single write operation triggered a system call, and this is highly inefficient considering the large number of possible connections.
Conversely, a `BufWriter` can **wrap a writer and buffer its output**: it keeps an in-memory buffer of data and writes it to the underlying writer in large, infrequent batches, making it an extremely valuable solution in cases where small and repeated write calls are performed on the same file.

```
// define the buffer encapsulating the file to be written
let mut buf = BufWriter::new(File::create(output_file).unwrap());
...
for (key, val) in map.iter() {
    // invoke a write operation on the buffer
}
...
// ensure that all buffered contents reach their destination
output.flush().unwrap();
```
***Code snippet 3.4** - The use of a `BufWriter` to improve report update performance.*

Buffering data instead of directly writing them to the file made the process about 6 times faster, but it was not enough to guarantee that packets weren't dropped in presence of elevated network throughputs.

To find a definitive solution to this concern, a **totally different approach** had to be considered: it didn't make sense to completely rewrite the report even if only a few entries had changed.
I needed a mechanism to track **only the changed entries** in each time interval and to update the file by changing just the corresponding lines.

Tracking the changed entries is not particularly difficult: it was enough to define a new collection of `AddressPortPair` in which elements are inserted when data are exchanged from the respective connection.
Such a collection of elements, implemented as a `HashSet`, is then emptied once the file is updated.

What has been more challenging was to correctly **identify the portion of the file to update** (i.e., the line corresponding to a given entry).
Two problems existed:

1. Entries were sorted by decreasing number of packets, thus making their position on the report file not stable over time.
2. `File` operations support the ability to be performed from a given offset (`Seek`) which can be expressed as a number of bytes, but not as quantity of lines.

Problem 1 was solved by using an `IndexMap` for the collection of the network connections: this structure consists of a hash table that **preserves insertion order**, allowing entries to be in a fixed position known a priori.
In this way, it's sufficient to save the indexes of the changed connections, and the index of an entry inside the map corresponds to the line of the output file where the same entry is printed.

The second problem was solved by structuring the file with **lines of fixed length**, so that the seek position (expressed in number of bytes) of a specific entry is given by `LINE_LENGTH * entry_index`.

Adopting such a strategy allows to **only rewrite the changed connections**, making the update time hundreds of times lower than that of the initial version.

```rust
// acquire the lock
let mut info_traffic = info_traffic_mutex.lock().unwrap();

// iterate only on the changed entries
for index in &info_traffic.addresses_last_interval {
    let (key, val) = info_traffic.map.get_index(*index).unwrap();
    // compute the seek position
    let seek_pos = LINE_LENGTH * (*index) as u64;
    // update the corresponding line of the file
    buf.seek(SeekFrom::Start(seek_pos)).unwrap();
    writeln!(buf, "{}{}", key, val).unwrap();
}
// empty the set of changed entries
info_traffic.addresses_last_interval = HashSet::new();
// release the lock
drop(info_traffic);
// flush the buffer content
buf.flush().unwrap();
```

Code snippet 3.5 *- Strategy used to rewrite, on the report file, only the changed entries.*

From version 0.4 up to the introduction of the graphical interface, the program also produced an additional output made of **SVG charts** aimed at visually presenting the traffic rate registered during the analysis.
The charts reported on the vertical axis the amount of incoming and outgoing data (per second and cumulative), together with the respective timestamps on the horizontal axis.

Consequently, the program output consisted no more of a single file, but of a folder containing both the textual report and the SVG pictures corresponding to the line charts.

A Rust drawing library named **Plotters** was used to generate the illustrations:
"Plotters is a drawing library designed for rendering figures, plots, and charts, in pure Rust. Plotters supports various types of back-ends, including bitmap, vector graph, piston window, GTK/Cairo and WebAssembly." [20]

Plotters provides a high-level, easy-to-use API for **data visualization** and it's designed to be highly flexible and extensible.
Furthermore, it comes with a handy developer's guide [21] and a lot of ready-made examples featuring practical use cases.

The first step to build a chart is to define a **drawing area** by specifying its destination path and dimensions.
Given a drawing area, a `ChartBuilder` is used to create a **chart context** characterized by several parameters, such as a caption, a mesh, a label formatter, a legend, and a series of points to plot.
Finally, the pending changes are published to the backend for the final rendering.

Plotters allows to choose from several of the most common chart types to represent the specified series of points, including area charts (used in the case of Sniffnet), histograms, scatter plots, or even three-dimensional surfaces.

I consider the introduction of charts a **milestone for the project** since this feature put the foundations for the idea of a complete graphical user interface.
The choice of Plotters as a rendering library has also been determinant in the subsequent decision of the GUI library of the application, as explained in the next section.

3.2. Version 1.0: the graphical user interface

After the introduction of charts, I thought that one final step would have been to include them, together with other information, in a **dashboard-like view** to see the updates in **real-time**, without the need of closing and reopening the SVG pictures and the textual report.

It was natural at this point to consider the use of a GUI library, to permit an integrated and user-friendly experience of the program.

3.2.1. The choice of the GUI library

Before implementing the ideas I had in mind, it was first necessary to decide on the **GUI library** to make use of.
I was sure about one thing: I wanted to develop the interface in **Rust** due to its performance and safety guarantees and because I liked the idea of exercising my front-end programming skills in a new language.

As anticipated in the first chapter, Rust GUI libraries are not so mature yet compared to other languages, but different alternatives seemed to me complete enough to be considered: Slint [22], Dioxus [23], Egui [24], and Iced [25] to name a few.
In particular, I was interested in **Egui** and **Iced**, two of the most popular options, constantly maintained and regularly updated.
Their reputation and large user base were an assurance to me that such libraries weren't projects that could have been abandoned from one day to another.

The most important difference between the two alternatives is the **kind of graphics APIs** used: Egui is immediate-mode based, while Iced uses retained-mode.
Immediate-mode APIs are **procedural**: each time a new frame is drawn, the application directly issues the respective drawing command.
Conversely, **retained-mode** APIs are **declarative**: the graphics library stores a model of the scene in memory, and it transforms the scene into a set of drawing commands. [26]

Egui's documentation carefully reports the main advantages and drawbacks of immediate mode: on the bright side, immediate mode is extremely easier to deal with from a programmer's perspective and it has lower memory requirements, while one of its main disadvantages is that it tends to have a higher CPU usage, since it does a complete layout each frame.
None of the API modes is definitively better than the other, so I had to consider more elements to choose between the two alternatives.

Egui is more focused on web development, while Iced has stronger attention towards **native applications**, and this made me more prone to consider the second option for Sniffnet.
What finally brought me to opt for Iced is the fact that it has a working integration with Plotters, the plotting library I was already using for Sniffnet's charts.

Iced is a cross-platform GUI library for Rust focused on simplicity and type safety, characterized by a reactive programming model, responsible layout, and built-in widgets. [25]

Figure 3.1 *- Logo of Iced, a GUI library for Rust.*

The library is inspired by **Elm**, a functional language to develop web applications. [27]
The language is based on a well-defined pattern, namely **The Elm Architecture**, which is based on top of four main concepts:

- **State**: an object or a collection of objects storing the application state.
- **Messages**: data structures to represent user interactions and other meaningful events that may occur.
- **View logic**: the mechanism to display the application state in the form of widgets that may produce messages on user interaction.
- **Update logic**: the set of procedures that make it possible to react to messages updating the application state accordingly.

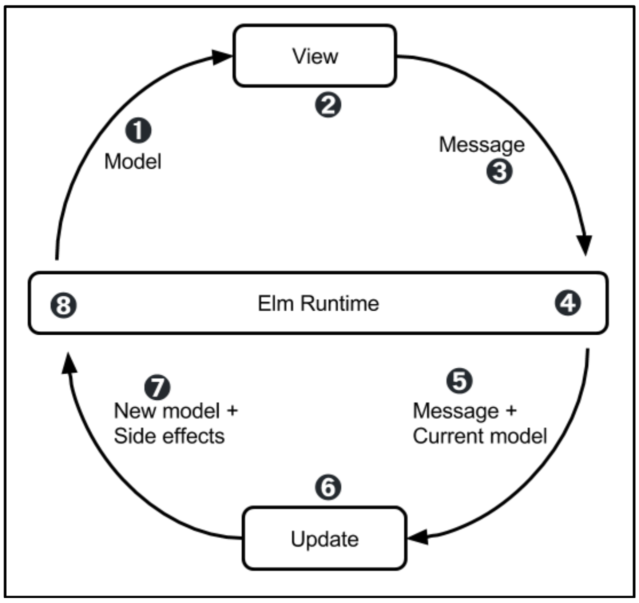


Figure 3.2 *- Visual representation of The Elm Architecture's principle.*

I immediately liked the pattern, and the idea of learning a completely different framework was exciting to me.

Iced was still in its 0.4 version when I started using it for Sniffnet in October 2022, but I judged the available features enough for my development use case.
However, having not reached its stable release yet, the library came with some **limitations**:

- The **documentation** is not complete, and a developer guide currently doesn't exist; this made the learning curve a lot steeper, at least in the first period of usage.
- Some **advanced widgets** are not integrated yet and require additional libraries or the developer defining its own custom components.
- The **text handling** strategy is basic and doesn't support features like text shaping or font fallback (essential for rotating and scaling text, and displaying glyphs from different languages without the need of specifying a font file).
- The **graphical renderer** is fixed in phase of development instead of being chosen at run-time and this is cause of incompatibilities with the underlying hardware in some circumstances.

Luckily, these and other minor limitations are being worked on and some have already been fixed, as will be explained in the following.

Another plus point in favor of Iced came short before I started using it: in early October 2022, System76 announced that the library will be used over GTK for the interface of **Pop!_OS** [28], a Linux distribution based on Ubuntu.

One of the System76 engineers involved in the development of **COSMIC**, the GNOME-based desktop environment which will be used by Pop!_OS, commented:

*"The UX team has been carefully designing widgets and applications over the last year.
We are now at the point where it is critical for the engineering team to decide upon a GUI toolkit for COSMIC.
After much deliberation and experimentation over the last year, the engineering team has decided to use Iced instead of GTK.
Various COSMIC applets have already been written in both GTK and Iced for comparison.
The latest development versions of Iced have an API that's very flexible, expressive, and intuitive compared to GTK.
It feels very natural in Rust, and anyone familiar with Elm will appreciate its design."*
[29]

3.2.2. GUI architecture

Evolving from the previous version of the program, since version 1.0 on, the **main** thread is no longer in charge of waiting for user inputs via the terminal interface but it's instead responsible for **setting up and operating the GUI** (after having spawned the threads responsible of secondary tasks).

The **state** of the application is encapsulated in a struct made of all the relevant fields needed to construct a graphical representation of the information of interest.
The data structure, named **`Sniffer`** in this scenario, includes the active traffic filters, the selected network adapter, different kinds of capture statistics, and other useful configurations.

The creation of an interface with Iced starts with the implementation of the `iced::application::Application` trait, which allows configuring an interactive **application based on the Elm architectural pattern**.
Such a trait is implemented for the `Sniffer` struct, allowing the GUI to respond to changes of the application state.

The `Application` trait requires the **implementation of different methods**:

- **`new`**, which setups the application with the provided initial state.

  ```
  fn new(flags: Sniffer) -> (Sniffer, Command<Message>) {
      (flags, iced::window::maximime(true))
  }
  ```

  Code snippet 3.6 *- Implementation of the `new` method of the `Application` trait*

  The method receives the initial state from the main thread, which passes the corresponding flags to the `Application::run` method (used to create the app instance).
  The `new` method can also optionally produce a command useful to perform asynchronous action in the background on start-up.

- **`title`**, a method specifying the application name to be reported on the respective window.

  ```
  fn title(&self) -> String {
      String::from("Sniffnet")
  }
  ```

  Code snippet 3.7 *- Implementation of the `title` method of the `Application` trait*

  Such a method can also be used to produce a dynamic title, in the case it's appropriate to have a different window name depending on the currently displayed page of the app.

- **`update`**, in charge of **handling messages** and changing the application state accordingly. It defines the update logic of the program.

```
fn update(&mut self, message: Message) -> Command<Message> {
    match message {
        Message::TickRun => {…}
        Message::AdapterSelection(name) => {…}
        Message::IpVersionSelection(version) => {…}
        Message::Reset => {…}
        …
    }
}
```

***Code snippet 3.8** - Structure of the `Application::update` method*

- **`view`**, the method responsible for returning the widgets to be displayed. Such widgets will produce messages when interacted with by the user.

```
fn view(&self) -> Element<Message> {
    let body = match *self.status_pair.0.lock().unwrap(){
        Status::Init => {
            initial_page(self)
        }
        Status::Running => {
            run_page(self)
        }
    };

    Container::new(
        body
    )
        .width(Length::Fill)
        .height(Length::Fill)
        .center_x()
        .center_y()
        .style(self.style)
        .into()
}
```

***Code snippet 3.9** - Original implementation of the `Application::view` method*

It's important to note that the `update` method receives `&mut Sniffer`, while `view` receives `&Sniffer`: the update logic oversees mutating the application state, while the view logic can only respond to such changes — the two mechanisms are decoupled, each one with its distinct task, but at the same time they cooperate in the definition of the interface.

Every time the `update` method is called, changing the app state, **the `view` method will be automatically invoked in its turn** to reflect the new conditions.

Beyond these mandatory members, other optional methods can be overridden.
That's the case of **`subscription`**, a method useful to be notified when specific events happen, which means the ability to generate a message when a determined event takes place.
The events captured by the subscription don't correspond to widgets interactions but rather to other (possibly asynchronous) events occurring at runtime.

Sniffnet overrides the `subscription` method to produce asynchronous messages to **periodically update** the application screens.

```
fn subscription(&self) -> Subscription<Message> {
    match *self.status_pair.0.lock().unwrap() {
        Status::Running => {
           iced::time::every(Duration::from_millis(PERIOD_RUNNING))
           .map(|_| Message::TickRun)
        }
        _ => {
           iced::time::every(Duration::from_millis(PERIOD_INIT))
           .map(|_| Message::TickInit)
        }
    }
}
```

***Code snippet 3.10** - Original override of the `Application::subscription` method*

This is essential to constantly update charts and other displayed information, typically once per second, even in absence of user interactions.

Other optional methods of the trait are `scale_factor`, which can be used to dynamically control the zoom of the UI at runtime, and `theme`, in charge of returning the current application `Theme` if more than one is available.

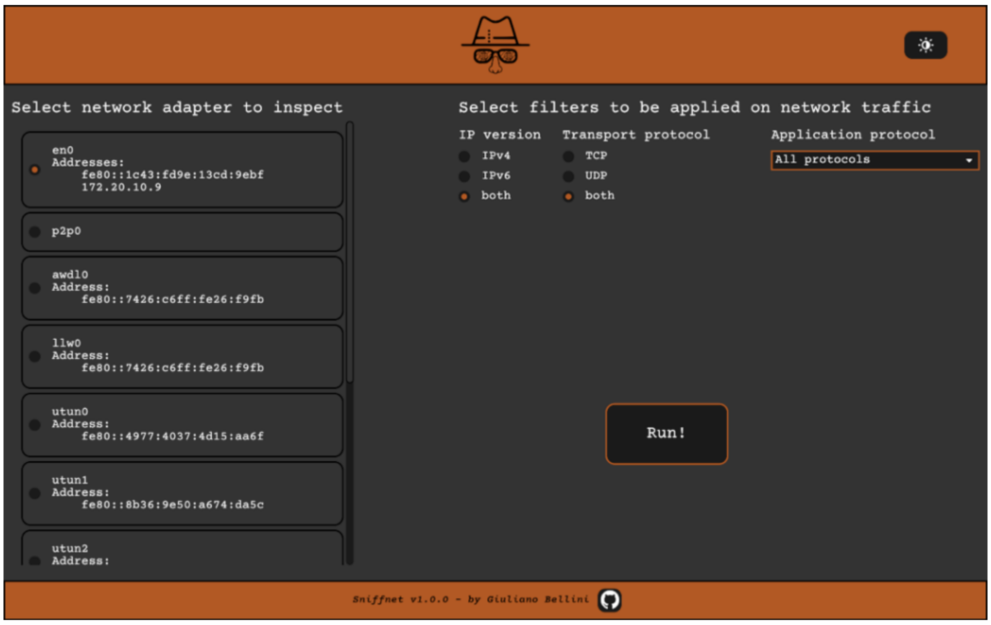
After the `Application` trait is implemented, the **`run`** method can be invoked to spawn the application window, specifying attributes to set the window's initial size, position, icon, as well as the initial state, the default text size, and other useful options.

3.2.3. The start page

When starting Sniffnet, it opens displaying a page that features, on the left, a scrollable column containing all the available **network adapters** and, on the right, checkboxes and a picklist reporting the applicable **filters**.

The header and footer, common among all the pages, respectively contain a button to switch the app style and a button linking to the GitHub repository of the project.

To correctly start the analysis is sufficient for the user to set the desired network adapter and, after having optionally defined some filters, click on the "*Run!*" button in the lower portion of the page.



Select network adapter to inspect

```
en0
Addresses:
    fe80::1c43:fd9e:13cd:9ebf
    172.20.10.9

p2p0

awdl0
Address:
    fe80::7426:c6ff:fe26:f9fb

llw0
Address:
    fe80::7426:c6ff:fe26:f9fb

utun0
Address:
    fe80::4977:4037:4d15:aa6f

utun1
Address:
    fe80::8b36:9e50:a674:da5c

utun2
Address:
```

Select filters to be applied on network traffic

IP version: IPv4, IPv6, [✓ both]

Transport protocol: TCP, UDP, [✓ both]

Application protocol: All protocols

Figure 3.3 *- Sniffnet start page (v1.0.0)*

The list of **network adapters** is retrieved through the `Device::list()` method offered by Pcap, which includes the adapter name, its description, and the set of active IPv4 and IPv6 addresses.
On macOS the adapter's description is often unavailable, while on Windows the adapter name is verbose and displaying its description is preferable: for this reason, a logic has been set up to conditionally include or exclude some parameters depending on the operating system and on the parameter availability within the specific machine.

The user can easily set **filters** on the observed network traffic, choosing the IP version, the transport protocol, or the application protocol desired for analysis.
By default, none of the filters is active because most of the users are generally interested in monitoring all the exchanged traffic.

The original version of the application included a total of 13 different kinds of **messages**, represented through the `Message` enum.
Several new message categories were introduced with the subsequent development of new features and is therefore not possible to describe the update logic behind each of them in detail.

To **exemplify** the mechanism that allows to update the application state, in the following are reported the relevant code fragments used to **change the transport protocol filter**, accessible to users from the start page.

```
// Definition of the widget.

let mut transport_filters = Column::new();
for option in [TransProtocol::TCP,
               TransProtocol::UDP,
               TransProtocol::Other] {
    transport_filters = transport_filters.push(
        Radio::new(
            option.get_radio_label(), // label
            option, // value
            Some(active), // currently selected
            Message::TransportProtocolSelection, // message to produce
        )
    );
}
```

```
// Update logic of the corresponding message.

pub fn update(&mut self, message: Message) -> Command<Message> {
    match message {
        …
        Message::TransportProtocolSelection(protocol) => {
               self.filters.transport = protocol;
        }
        …
    }
}
```

***Code snippet 3.11** - Widget definition and logic to update the transport protocol filter.*

The defined **widget**, in this case consisting of a collection of radio buttons, specifies the **message** to be produced when the user interacts with it; the corresponding message is then received by the **update logic** which takes care of updating the transport protocol filter, and immediately after the **view is re-rendered**.

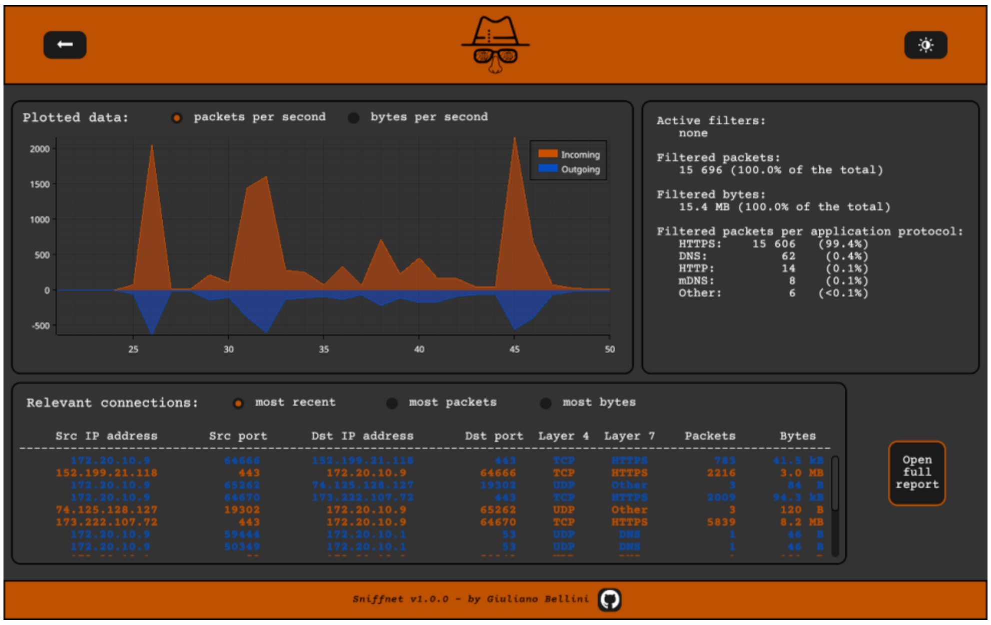
Despite this being a simple example, it clarifies how the Elm pattern works in practice. Excluded the messages generated via the subscription, all the remaining ones are produced in a way like that shown — what changes from one message kind to another are mainly the update logic and the fact that each message can be associated with one or more different **typed parameters**, thanks to Rust enum's expressiveness.

3.2.4. The overview page

Whenever the user wants to **start** the analysis, pressing the "*Run!*" button, a `Message::Start` is produced.
The update logic handles such a message by creating a **new thread** responsible for **parsing packets** exchanged from the network adapter chosen by the user.

If an **error** occurs opening a capture on the specified adapter, a proper message will be displayed; the causes of error can be various: the user doesn't have the required privileges to inspect the selected adapter, the adapter doesn't have active addresses and therefore it cannot exchange any traffic, or a different problem is raised by the underlying Pcap library.
In other circumstances, configurations are correct, but the chosen adapter simply doesn't receive/send any packet from/to the network.
All the possible different scenarios are managed by the application, **informing the user** about possible countermeasures to take.

In case everything goes smoothly, the analysis is started, and the user can see **live statistics** about network data in transit.



```
Active filters:
   none

Filtered packets:
   15 696 (100.0% of the total)

Filtered bytes:
   15.4 MB (100.0% of the total)

Filtered packets per application protocol:
   HTTPS:    15 606   (99.4%)
   DNS:          62    (0.4%)
   HTTP:         14    (0.1%)
   mDNS:          8    (0.1%)
   Other:         6   (<0.1%)
```

Relevant connections: ● most recent ● most packets ● most bytes

Src IP address	Src port	Dst IP address	Dst port	Layer 4	Layer 7	Packets	Bytes
172.20.10.9	64666	152.199.21.118	443	TCP	HTTPS	783	41.5 kB
152.199.21.118	443	172.20.10.9	64666	TCP	HTTPS	2216	3.0 MB
172.20.10.9	65262	74.125.128.127	19302	UDP	Other	3	84 B
172.20.10.9	64670	173.222.107.72	443	TCP	HTTPS	2009	94.3 kB
74.125.128.127	19302	172.20.10.9	65262	UDP	Other	3	120 B
173.222.107.72	443	172.20.10.9	64670	TCP	HTTPS	5839	8.2 MB
172.20.10.9	59444	172.20.10.1	53	UDP	DNS	1	46 B
172.20.10.9	50349	172.20.10.1	53	UDP	DNS	1	46 B

Figure 3.4 *- Sniffnet overview page (v1.0.0)*

As in the previous version of the app, the thread parsing packets insert them into **shared data structures** associated with a `Mutex`.
The same structures are also accessed, in mutual exclusion, by the thread in charge of periodically updating the textual report and by the main thread in the view logic of the GUI, to display the corresponding information.

The **charts** are realized with Plotters and are integrated with Iced using a backend named **`plotters-iced`** [30], which has been created by a Chinese developer.
To correctly render a chart with `plotters-iced`, it's necessary to implement the `Chart` trait for a struct representing the chart object.
The trait requires implementing a method named `build_chart` responsible for defining parameters required by Plotters (i.e., the series of points, their label, a legend, and so on).
Finally, to render the chart, a method `view` must be defined for the chart object, featuring the code necessary to embed it into an Iced widget with defined dimensions, alignment, and padding.

Sniffnet offers the possibility for the user to choose which **unit of measure** to adopt for the chart, between bytes per second and packets per second; in both cases, the real-time chart reports the **last 30 seconds of data**, stored inside a `VecDeque` on which every second it's performed a `pop_front()` followed by a `push_back(…)` operation to make the representation shift as time passes.

The lower section of the page displays a report about the **most relevant connections**, that can be ordered by timestamp, number of packets or bytes exchanged.
Only the top 15 connections are shown in the view because Iced doesn't support yet the possibility of computing only the visible portion of a scrollable, and therefore building a widget with different thousands of entries could've been inefficient.

However, the **complete report** of network connections remains available in a textual file that can also be opened via the graphical interface.
This is possible using a **resource opener command**, which is a command used by an operating system to open a file, a directory, or an URL.
Due to the **cross-platform** nature of the project, I had to make sure of carefully selecting the resource opener command depending on the target OS: luckily Rust offers an easy way of conditionally including or excluding from compilation specific fragments of code, and using the `target_os` attribute it's possible to determine the operating system of the machine running the program.

```
Message::OpenReport => {
    #[cfg(target_os = "windows")]
    let command = "explorer";
    #[cfg(target_os = "macos")]
    let command = "open";
    #[cfg(target_os = "linux")]
    let command = "xdg-open";

    std::process::Command::new(command)
        .arg(r"./sniffnet_report/report.txt")
        .spawn()
        .unwrap();
}
```
***Code snippet 3.12** - Command used to open the report file on the different OSs.*

Differently from the previous version of the program, since the introduction of the GUI Sniffnet supports the possibility of **stopping the ongoing analysis and later starting another analysis targeting a different adapter**, without the need of quitting the program.
This option would've been uncomfortable in a command line interface, but it's straightforward and intuitive in a graphical interface.

A user can now easily interrupt the current sniffing process simply by coming back to the initial screen of the application: in correspondence with this event, a `Message::Reset` is produced.
When said message is received by the update logic of the program, the data structures containing information about the preceding analysis are **re-initialized**.

Furthermore, an additional variable representing the **analysis identifier** is incremented; it was in fact necessary to introduce a monotonically increasing ID to execute, in the secondary threads, proper actions when a new sniffing process is created:

- The **thread parsing packets**, unique for each analysis, checks if the analysis identifier is changed with respect to that assigned to it; in this case, it immediately returns.
  The thread needs to be unique for each analysis because this execution flow blocks on the `next_packet()` method offered by Pcap: if an adapter not exchanging packets is selected, the call to this method will block indefinitely making it impossible to re-use the same thread for a different adapter.
  From this observation also derives that the analysis ID is needed to be checked after a packet is received, to be sure that the respective sniffing process wasn't killed and then a different one was created in the meantime.
- The **thread writing report**, which remains the same for the whole duration of the application instance, checks the ID to decide if updating the file with data from the last time interval or rewriting the file from scratch to accommodate information related to a distinct sniffing process.

3.2.5. GUI styling

Similarly to HTML, whose elements are styled via CSS, **Iced widgets can be customised**.

This is done by implementing the **`StyleSheet`** trait of a certain widget; every kind of widget has its own theming rules which are specified in the definition of the respective trait (`button::StyleSheet` for buttons, `slider::StyleSheet` for sliders, and so on).

Each `StyleSheet` takes care of defining peculiar methods based on the widget's nature (e.g., `active()`, `hovered()`, and `disabled()` for buttons).
Iced automatically recognizes the current state of a widget and displays the element according to the rules defined in the corresponding method of the `StyleSheet`.

During Sniffnet development, I didn't only want to specify a style for each of the application components, but I also wished to let users decide from **different themes**: for this reason, two different color palettes were used in version 1.0 (representing respectively day-mode and night-mode), and additional themes were made available in the following release.

To improve the code modularity, a **`Palette`** struct was defined to encapsulate all the needed information about a specific application theme:

```rust
pub struct Palette {
    /// Main color of the GUI (background, hovered buttons)
    pub primary: Color,
    /// Secondary color of the GUI (incoming connections, header, footer)
    pub secondary: Color,
    /// Color of outgoing connections
    pub outgoing: Color,
    /// Color of buttons
    pub buttons: Color,
    /// Color of header and footer text
    pub text_headers: Color,
    /// Color of body and buttons text
    pub text_body: Color,
}
```

***Code snippet 3.13** - Definition of the Palette data structure.*

To style a given widget, take for example a `Container`, it's necessary to specify both the **widget type** (i.e., if the `Container` is intended to be a header, a box, or the application main scene) and the current **application theme** (i.e., day-mode, night-mode, and similar).
Consequently, the styling traits were implemented for a struct made of two fields: the widget type referred to a specific element and the global application theme chosen by the user.

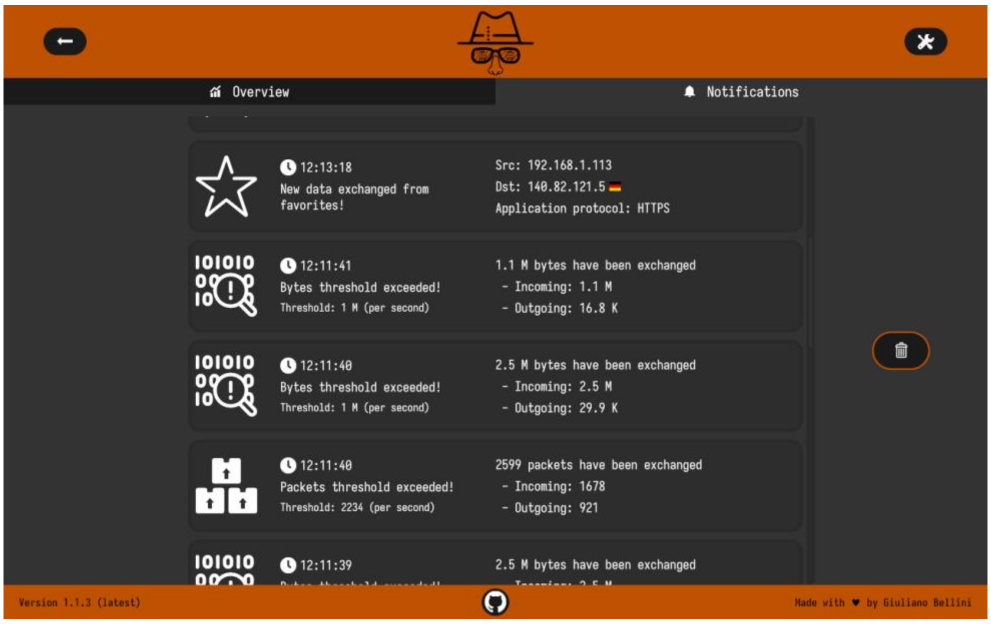
3.3. Version 1.1: notifications, IP geolocation, and further configurations

3.3.1. Custom notifications

Several new features and improvements were introduced with version 1.1.
One of the main changes consisted of the introduction of **custom notifications**, namely the possibility for users to set alerts when defined network events occur.

Sniffnet can warn the user by sending a notification whenever a **data threshold** is exceeded, or new data is exchanged from one of the **favourite connections**.

At every `Message::TickRun`, produced by a periodic subscription once per second, the conditions for a new notification are checked: if necessary a short sound effect is reproduced and the respective event is logged for additional examination by the user.



```
12:13:18
New data exchanged from
favorites!
                              Src: 192.168.1.113
                              Dst: 140.82.121.5
                              Application protocol: HTTPS

12:11:41
Bytes threshold exceeded!     1.1 M bytes have been exchanged
Threshold: 1 M (per second)    - Incoming: 1.1 M
                               - Outgoing: 16.8 K

12:11:40
Bytes threshold exceeded!     2.5 M bytes have been exchanged
Threshold: 1 M (per second)    - Incoming: 2.5 M
                               - Outgoing: 29.9 K

12:11:40
Packets threshold exceeded!   2599 packets have been exchanged
Threshold: 2234 (per second)   - Incoming: 1678
                               - Outgoing: 921

12:11:39                      2.5 M bytes have been exchanged
```

Figure 3.5 *- Application page dedicated to logging the received notifications (v1.1.3)*

Different **information** is collected based on the notification kind: in case a bytes or packets threshold has been exceeded, the amount of incoming and outgoing data is reported, while in case a favourite connection was involved, details about the connection are shown.
For each instance, the respective timestamp of occurrence is also displayed.

The log can optionally be **emptied** by the user at any moment, to clean the screen from previous notifications.

The different types of notifications are internally represented as an **enum**, each of whose variants is associated with a struct encapsulating information relative to the corresponding notification instance.

```
/// Enum representing the possible notification events.
pub enum LoggedNotification {
    /// Packets threshold exceeded
    PacketsThresholdExceeded(PacketsThresholdExceeded),
    /// Byte threshold exceeded
    BytesThresholdExceeded(BytesThresholdExceeded),
    /// Favorite connection exchanged data
    FavoriteTransmitted(FavoriteTransmitted),
}
```

***Code snippet 3.14** - The `LoggedNotification` enum, used to represent the different notifications*

When the conditions for an alert arise, a new instance of a notification is created, and it's pushed to the front of a list which is displayed in the log page of the UI.

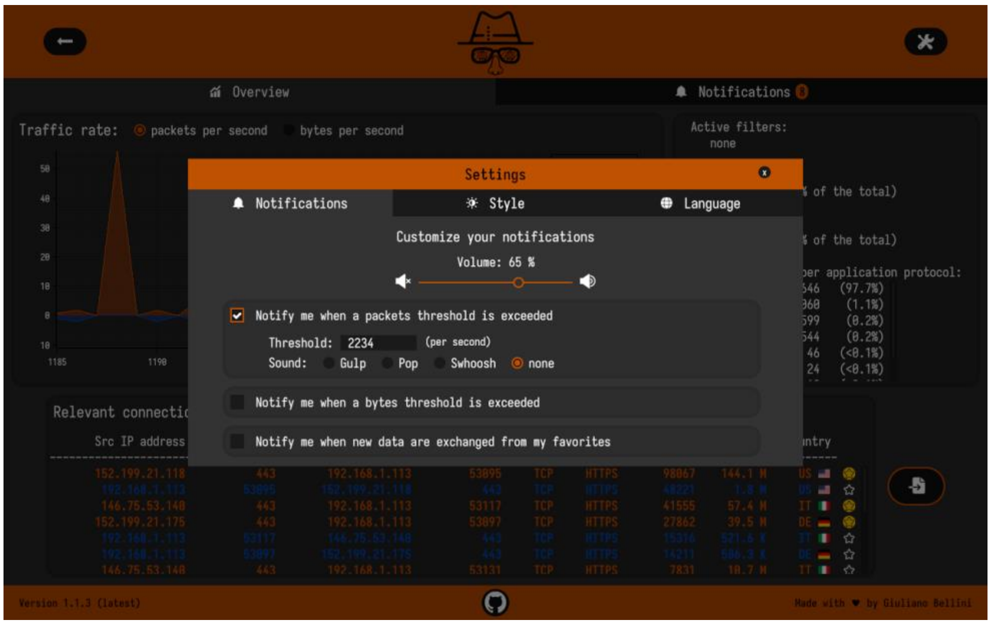
Besides logging the event, a **sound alert** is played to make the notification perceptible by the user even if the Sniffnet window is not on top of the screen.
This is possible thanks to **Rodio** [31], a playback library for Rust that allows to reproduce audio by defining a source consisting in the sound to play and getting an output stream handle to a physical device.

```
// Get a output stream handle to the default physical sound device
let (_stream, stream_handle) = OutputStream::try_default().unwrap();
let sink = Sink::try_new(&stream_handle).unwrap();
// Load sound from memory
let data = std::io::Cursor::new(mp3_sound);
// Decode sound file into a source
let source = Decoder::new(data).unwrap();
// Set the desired volume
sink.set_volume(f32::from(volume) / 200.0);
// Play the sound directly on the device
sink.append(source);
// Block current thread until the sink has finished playing queued sounds.
sink.sleep_until_end();
```

***Code snippet 3.15** - Reproduction of a sound with Rodio*

The thread invoking the reproduction of the sound effect sleeps until the audio is terminated; for this reason, it's necessary to execute the code above in a **new thread** (spawned for this specific purpose), otherwise the main thread would result busy, causing the loss of possible user interactions with the interface.

Individual notifications can be enabled, disabled, and customized from the new **settings** page of the application, which permits to specify the event for which it's desired to be notified and additional parameters such as the specific data threshold of activation, the sound effect to play, and its volume.



Customize your notifications

Volume: 65 %

[✓] Notify me when a packets threshold is exceeded

Threshold: 2234 (per second)

Sound: Gulp Pop Swhoosh [✓ none]

[ ] Notify me when a bytes threshold is exceeded

[ ] Notify me when new data are exchanged from my favorites

Figure 3.6 *- Settings page to customize notifications (v1.1.3)*

3.3.2. Interface translations

Sniffnet **settings** are not limited to the choice of user-defined notifications, but also allow to set an application colour scheme (based on the modalities described in paragraph 3.2.5 — GUI styling) and permit to select a language different from English.

In version 1.1, support for the **translation** of the interface text was added.
A new function, receiving the selected language as a parameter and returning the corresponding translation, was created for each of the UI sentences.

```rust
pub fn start_translation(language: Language) -> &'static str {
    match language {
        Language::EN | Language::DE | Language::RO | Language::KO =>
"Start!",
        Language::IT => "Avvia!",
        Language::FR => "Commencer!",
        Language::ES => "¡Empieza!",
        Language::PL => "Rozpocznij!",
        Language::UK => "Почати!",
        Language::ZH => "开始!",
        Language::TR => "Başla!",
        Language::RU => "Начать!",
        Language::PT => "Começar!",
        Language::EL => "Ξεκίνα!",
        Language::FA => "شروع!",
        Language::SV => "Starta!",
        Language::FI => "Aloita!",
    }
}
```

Code snippet 3.16 *- Example of function used to translate the UI sentences*

To have a higher degree of confidence about the translation quality, the process is not performed using external services but requires the translated phrases to be manually included in the code base.
Each of the supported languages was in fact introduced by a native speaker through a pull request and, thanks to community support, Sniffnet rapidly went from supporting only English and Italian to being translated into **17 different languages**.

Introducing new languages, a problem came out: **Iced doesn't support font fallback** and consequently a font file must be included to make specific Unicode characters available.
This is especially crucial when adding languages that don't use ASCII characters, such as Chinese, Korean, and Russian.

However, **font files** covering all the existing characters are **very large in size**: just consider that Simplified Chinese is composed of thousands of glyphs.
The file I ended up using contains all the languages of interest, but its size alone was three times the size of the whole binary.

For this reason, after searching for some solution, an idea came to my mind: the characters actually in use are a very small fraction of the full set, and to solve the problem it'd be sufficient to only include them.
Luckily, there exists a mechanism called **font subsetting** which allows to select a specific set of characters from a font file.
First of all, it's necessary to retrieve all the used characters and I performed this task using the `grep` command line utility against the `src` folder of the project.
The collected characters are stored in a file which is then given as input to `pyftsubset` [32], a font subsetting program that creates a new, reduced font file from the complete one.
In this way, the size of the font file decreased **from 13 MB to 100 KB**, allowing to include the font in the app binary without bloating its dimension.

One more problem due to Iced text handling is the **missing support for right-to-left languages**, such as Arabic and Persian.
A Persian translation of Sniffnet is available, but the interface displays it left-to-right, making it impossible to read correctly.
However, the text-handling strategy of Iced is rapidly evolving and a new release of the library will soon fix most of the related issues.

3.3.3. Configuration management

The addition of application settings led to the need for a mechanism to **save the configuration parameters** set during a run of the program.

Let's suppose that a user wishes to be notified when more than 2 MB per second are exchanged, using a color palette different from the default one, and operating the app using the French language.
If no mechanism to save the app settings is used at all, our user would need to set his desired configurations every time the application is restarted.

To make configurations persistent across different executions of Sniffnet, it's needed to store them somewhere, typically in the form of a JSON or TOML file.
I decided to use **Confy** [33], a zero-boilerplate configuration manager for Rust, to handle in an easy and robust way the read and write of such information.
Confy takes the burden to figure out **where** to place configuration files based on the given operating system and the environment paths.

A configuration file is the **serialized** version of a **struct** encapsulating the corresponding parameters.
When Sniffnet is run after a fresh installation, the configuration file doesn't exist yet but that's not a problem because Confy uses the `Default` trait implemented by the struct to create a new configuration: this means the developer can assume that a configuration already exists, without taking care of any special logic to handle its creation.

Every time the configuration file needs to be changed, it's sufficient to invoke `confy::store`, which **saves the changes** applied to the configuration object passed as a parameter.

To avoid **collisions** between different configuration files of the same program or configurations related to different applications, every read/write operation performed with Confy requires to specify the name of the app and, optionally, the identifier of the file that must be unique in the scope of a given app.

Sniffnet relies on two configuration files: one to store the **settings** selected by the user and the other to save the last successfully sniffed **network device**, so that both the information will be automatically set when reopening the app.

Since version 1.1, the **path** chosen by Confy to store the configurations is also used to host the **output report** of the program.
This is useful to avoid the presence of multiple reports in the system, since previously the output location was not fixed but coincident with the running path of the program.

3.3.4. IP geolocation

Another feature added in version 1.1 is the **geolocation of remote IP addresses**.
IP-based geolocation is the mapping of an IP address to the geographical location of a device.

To perform such mapping, a proper **database** is needed; there are many providers offering such databases, whose accuracy is claimed to be about 95% for what concerns IP-to-country resolutions.
Initially, during development, I started using a CSV database, but I soon realized that there exists a better format tailored for this kind of usage: the **MaxMind DB** (MMDB) file format. [34]

The MMDB file format is a database format that maps IPv4 and IPv6 addresses to data records using a binary search tree.
The format is optimized to perform **lookups on data indexed by IP network ranges** quickly and efficiently: thanks to this high performance, Sniffnet is potentially able to execute hundreds of different lookups in a matter of a few milliseconds.
Not only the MMDB format allows more performant read operations, but it also permits to largely reduce the binary size, being about 4 times smaller than the corresponding CSV file.

Given a network connection, the address subject of a lookup is the **remote** one since it wouldn't make sense to try resolving the private address of the sniffed device (such a resolution would result in an unknown location).

To further improve the efficiency of the geolocation process, IP lookups only occur in presence of **new network connections**, to avoid repeating duplicated resolutions that would happen if lookups were performed at every exchanged packet.

```
let len = info_traffic.map.len();
let index = info_traffic.map.get_index_of(&key).unwrap_or(len);
let country = if index == len {
    // first occurrence of this connection => retrieve country code
    get_country_code(traffic_type, &key, country_db_reader)
} else {
    // this connection was already featured
    String::new()
};
```

Code snippet 3.17 *- Code fragment to retrieve the country code only in case of new network connections*

3.3.5. Keyboard shortcuts

With the aim of improving the application's efficiency of use and the overall user experience, some **keyboard shortcuts** were introduced.

A keyboard shortcut, or hotkey, is a combination or sequence of keys on a computer keyboard that **triggers a command**.
Hotkeys allow for a **quicker** execution of certain tasks and are especially appreciated by more experienced people who frequently make use of the software.

Iced offers a family of events, including `Touch`, `Window`, `Mouse`, and `Keyboard` related events.
`Keyboard` events can be related to the pressure or the release of some keystrokes, and in both cases, they consist of a key and a set of modifiers (e.g., `ctrl`, `alt`, or `shift`).

Such events can be captured by the application **`subscription`**, which will send a proper message that in its turn will be handled by the update logic to perform specific actions.

```
let hot_keys_subscription =
    iced_native::subscription::events_with(|event, _| match event {
        // ctrl+Q => quit the app
        Event::Keyboard(iced_native::keyboard::Event::KeyPressed {
            key_code: iced_native::keyboard::KeyCode::Q,
            modifiers: iced_native::keyboard::Modifiers::COMMAND,
        }) => Some(Message::Quit),
        // tab => switch to next page
        Event::Keyboard(iced_native::keyboard::Event::KeyPressed {
            key_code: iced_native::keyboard::KeyCode::Tab,
            modifiers: NO_MODIFIER,
        }) => Some(Message::SwitchPage(true)),
        // shift+tab => switch to previous page
        Event::Keyboard(iced_native::keyboard::Event::KeyPressed {
            key_code: iced_native::keyboard::KeyCode::Tab,
            modifiers: iced_native::keyboard::Modifiers::SHIFT,
        }) => Some(Message::SwitchPage(false)),
        // ctrl+, => open settings
        Event::Keyboard(iced_native::keyboard::Event::KeyPressed {
            key_code: iced_native::keyboard::KeyCode::Comma,
            modifiers: iced_native::keyboard::Modifiers::COMMAND,
        }) => Some(Message::OpenLastSettings),
        // ...
        _ => None,
    });
```

***Code snippet 3.18** - Subscription to capture `Keyboard` events and produce proper messages*

All the available hotkeys are **documented** in a dedicated section of the project's README and are based on the most popular conventions.
A specific issue [35] was also opened to allow the community to suggest new shortcuts or changes to the existing ones.

3.3.6. Further additions and improvements

Several **other minor features and improvements** were released between versions 1.1.0 and 1.1.4.

- Implemented the possibility of marking a group of connections as **favorites** and added a favorites view to the report in the main page of the app.
- Added a modal to ask the user for confirmation before interrupting the ongoing analysis.
- **Aesthetic improvements** to create a more modern and minimal UI, as described in a dedicated issue. [36]
- Made the most complex widgets lazy to improve the app's efficiency.
  Lazy widgets are interface elements that specify a set of dependencies and call the view logic lazily only when one or more dependencies have changed.
  If all the dependencies have remained unchanged between consecutive invocations of the `view` method, a **previously cached instance** of the widget will be rendered instead of recomputing it from scratch.
- Added feature to **warn users when a newer release of the software is available**.
  This is done by instantiating a dedicated thread to make a call to the **GitHub API** and checking if the latest published version of the app is different from the running one (available in the manifest file of the project).
  In case of errors the call is repeated after 30 seconds and if a newer version is found to be available, a warning button linking to the latest release page is displayed in the application footer.

3.4. Version 1.2: host-based traffic analysis

The original version of the GUI only showed **IP addresses** to identify senders and receivers, but this kind of information consists of a series of numbers that are **not very meaningful at a first sight** for a human being.

With version 1.2, I wanted to provide a way to discover more details about the network host behind an IP address.

3.4.1. Host-related information

A **network host** is a device connected to a computer network, which may work as a server providing resources to other nodes of the network.
Each host is associated with one or more IP addresses, a **name**, and is usually managed by a defined **administrative entity**.

Given an IP address, it's possible to determine additional information about the corresponding host.
For instance, hostname and address are tightly linked one with the other and this bond is at the basis of the **DNS resolution**, i.e. the process of mapping human-readable domains to machine-readable IP addresses.

IP addresses are collected from the packets' headers and are available in Sniffnet since its very first release; **additional host-related attributes** have been obtained simply by making wise use of the already available information:

- **Host names** are retrieved performing **reverse DNS lookups** (i.e., the inverse process of DNS resolution, useful to obtain a human-readable name from an Internet address).
- **Autonomous Systems names and numbers** are obtained through lookups against an MMDB file, in the same way described for IP geolocation in section 3.3.4.

These apparently minor changes required a **considerable redesign of the application workflow** as well as the introduction of **proper data structures** to accommodate and adequately organize the newly collected details.

Reverse DNS lookups are subject to **varying latencies** depending on the presence of the entry in a local cache, the location of the DNS server, network congestions, and they can take **up to some seconds** in case of reachability issues: therefore, it's not possible to perform this kind of resolutions directly in the thread in charge of parsing packets and a new execution flow must be created for the purpose.

The possible **latency** of reverse DNS lookups led to two main **concerns**:

1. Due to timing constraints, it's not convenient to instantiate a single thread responsible for all the resolutions.
2. To save computational resources, it's important to minimize the number of lookups but it's not immediate to determine if the resolution of a specific address has already been invoked or not.

The first problem can be fixed by instantiating a **dedicated thread for each of the addresses** to resolve, but again this leaves open the second problem: *when is it really necessary to perform a lookup?*

If a reverse DNS lookup has already been completed for address A, it's easy to realize to not perform it again since the respective domain name is already present, but what if new data is exchanged from A while waiting for the result of a previously invoked resolution of the same address?
To avoid looking up the same IP address multiple times, it's not enough to have a collection of the **already resolved** hosts but it's also needed to maintain the set of addresses **waiting for a resolution**.
Addresses are inserted in such set right before invoking their resolution and when the result of the lookup is delivered, the address is removed from the set in which it was temporarily put and is finally inserted in the collection of network hosts.

In this way, the thread responsible for parsing packets can check the status of an address and it acts accordingly:

```
match (r_dns_waiting_resolution, r_dns_already_resolved) {
    (false, false) => {
        // rDNS not requested yet (first occurrence of this address)

        // Add the address to the map of addresses waiting for a resolution

        // Launch new thread to resolve host name
    }
    (true, false) => {
        // Waiting for a previously requested rDNS resolution

        // Update the statistics related to the waiting address
    }
    (_, true) => {
        // Reverse DNS already completed

        // Update the statistics related to the already identified host
    }
}
```

Code snippet 3.19 *- Pseudo code to handle the possible different states of the reverse DNS lookup of an IP address*

The created thread launches a reverse DNS resolution using the `lookup_addr` method provided by the `dns-lookup` crate [37], which blocks the execution until a valid result or an error is returned.
In case of problems, the IP address itself is used in place of the domain name.

Once the address is resolved, the thread also takes care of retrieving, from MMDB files, its geographic location and details about the **Autonomous System** (AS) managing it.

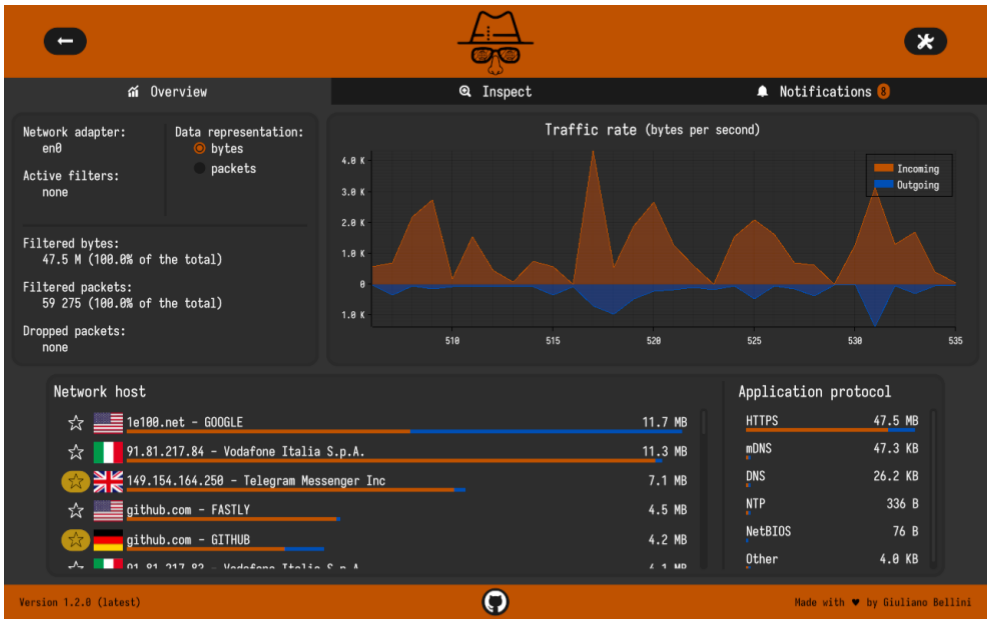
At this point, all the information to define the corresponding network host is available and a new `Host` is created.

```rust
pub struct Host {
    /// Hostname (domain). Consists of the last portion of the reverse DNS.
    pub domain: String,
    /// Autonomous System (name and number) which operates the host
    pub asn: Asn,
    /// Country
    pub country: Country,
}
```

Code snippet 3.20 *- The `Host` struct identifying a network host*

3.4.2. The new overview page

The newly integrated features are reflected on the graphical interface of the app, whose main page is now characterised by a section assigned to the presentation of host-related details.



Network adapter:
    en0
Active filters:
    none

Data representation:
    ● bytes
    ○ packets

Filtered bytes:
    47.5 M (100.0% of the total)
Filtered packets:
    59 275 (100.0% of the total)
Dropped packets:
    none

Network host

1e100.net - GOOGLE	11.7 MB
91.81.217.84 - Vodafone Italia S.p.A.	11.3 MB
149.154.164.250 - Telegram Messenger Inc	7.1 MB
github.com - FASTLY	4.5 MB
github.com - GITHUB	4.2 MB

Application protocol

HTTPS	47.5 MB
mDNS	47.3 KB
DNS	26.2 KB
NTP	336 B
NetBIOS	76 B
Other	4.0 KB

Version 1.2.0 (latest)

Made with ♥ by Giuliano Bellini

Figure 3.7 *- Sniffnet overview page (v1.2.0)*

The upper section of the page remains similar, except for the addition of the number of **dropped packets**, namely the amount of discarded packets because there was no room for them in the Pcap buffer when they arrived.

The lower portion of the page has completely been renewed to show the amount of data transmitted grouped by **network host** and by **application protocol**.
Information exchanged by a given entry is represented as a **horizontal bar** whose length is proportional to the number of packets or bytes transmitted and whose colours represent the directionality of the exchange (incoming or outgoing).

```
pub struct DataInfo {
    pub incoming_packets: u128,
    pub outgoing_packets: u128,
    pub incoming_bytes: u128,
    pub outgoing_bytes: u128,
}
```

Code snippet 3.21 *- The `DataInfo` struct, encapsulating details about the amount of exchanged data and used to store cumulative statistics related both to network hosts and application protocols*

Hosts are **sorted** by decreasing number of packets or bytes and can be marked as a **favourite** to be notified when a new data exchange will occur.

Each network host is associated with a flag representing the respective **country**, even if in some circumstances it's not possible to determine the location of a host because the corresponding entry is **not available** in the MMDB file.
Most of the times this happens when the host is in the same network as the sniffed network card (and has therefore a private IP address) or when the IP address is a multicast or broadcast one.

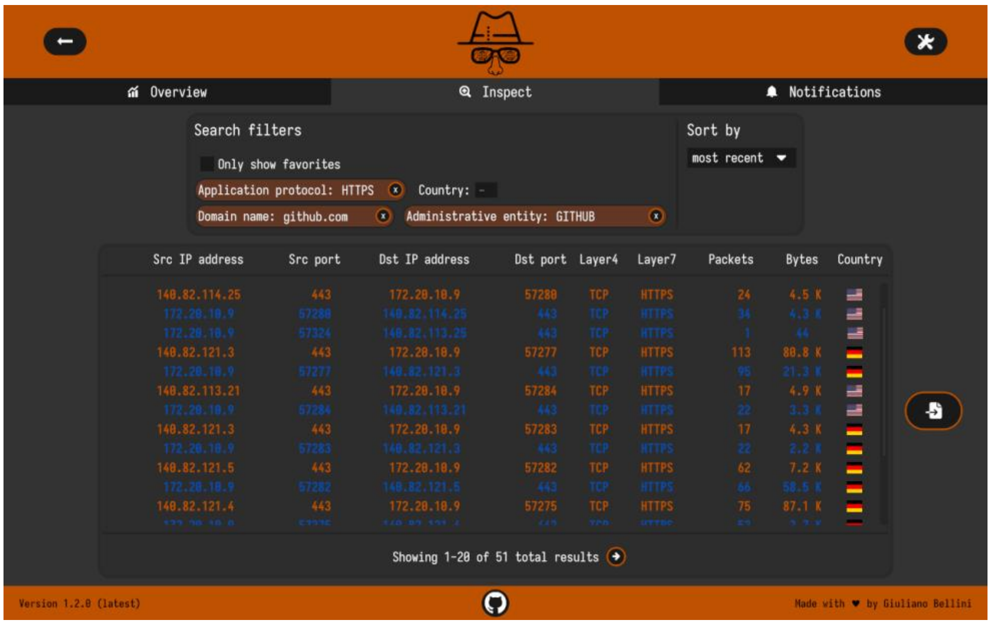
To limit the number of circumstances in which it's not possible to associate a host with a country flag, some countermeasures were taken:

- **Multicast addresses** recognition has been implemented for both IPv4 and IPv6 addresses.
- **Broadcast addresses** identification has been added for IPv4 addresses, including directed broadcast recognition.
- **Local addresses**, namely addresses in the same network of the analyzed adapter, are identified through operations involving the subnet mask of the sniffed device, provided by Pcap.

In such cases, instead of showing a country flag, a **proper icon** is displayed to make clarity about the nature of the traffic.

3.4.3. The inspect page

Individual network connections, identified by IP addresses and transport ports, remain available and are now reported in a dedicated page of the interface.



Search filters

Only show favorites

Application protocol: HTTPS

Country: -

Domain name: github.com

Administrative entity: GITHUB

Sort by

most recent

Src IP address	Src port	Dst IP address	Dst port	Layer4	Layer7	Packets	Bytes	Country
140.82.114.25	443	172.20.10.9	57280	TCP	HTTPS	24	4.5 K	🇺🇸
172.20.10.9	57280	140.82.114.25	443	TCP	HTTPS	34	4.3 K	🇺🇸
172.20.10.9	57324	140.82.113.25	443	TCP	HTTPS	1	44	🇺🇸
140.82.121.3	443	172.20.10.9	57277	TCP	HTTPS	113	80.8 K	🇩🇪
172.20.10.9	57277	140.82.121.3	443	TCP	HTTPS	95	21.3 K	🇩🇪
140.82.113.21	443	172.20.10.9	57284	TCP	HTTPS	17	4.9 K	🇺🇸
172.20.10.9	57284	140.82.113.21	443	TCP	HTTPS	22	3.3 K	🇺🇸
140.82.121.3	443	172.20.10.9	57283	TCP	HTTPS	17	4.3 K	🇩🇪
172.20.10.9	57283	140.82.121.3	443	TCP	HTTPS	22	2.2 K	🇩🇪
140.82.121.5	443	172.20.10.9	57282	TCP	HTTPS	62	7.2 K	🇩🇪
172.20.10.9	57282	140.82.121.5	443	TCP	HTTPS	66	50.5 K	🇩🇪
140.82.121.4	443	172.20.10.9	57275	TCP	HTTPS	75	87.1 K	🇩🇪

Showing 1-20 of 51 total results

Version 1.2.0 (latest)

Made with ♥ by Giuliano Bellini

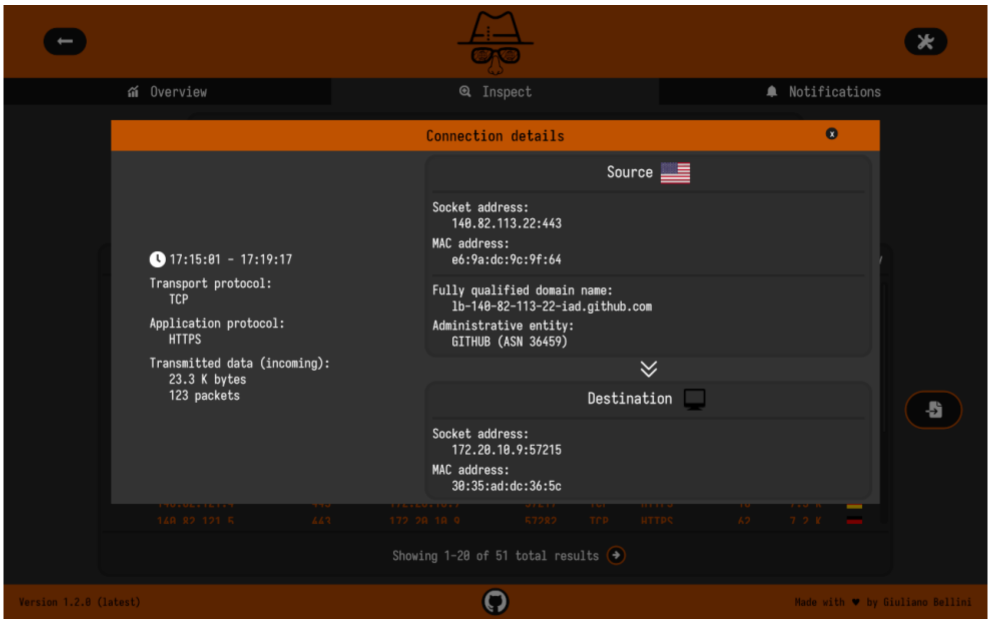
Figure 3.8 - *Sniffnet inspect page (v1.2.0)*

Having a whole page hosting the network connections allows to display them more comfortably, without being limited to showing a reduced number of entries as in the previous release.
More precisely, **all connections** are now shown in the UI through a **paging mechanism** developed ad hoc, which lets the user easily switch between pages either by pressing a button or via a keyboard shortcut.

Connections can be **filtered** to display only the entries of interest according to various criteria: carried application protocol, country of provenience, domain name, or Autonomous System of the correlated network host.

What's even more interesting is that this page can also be reached directly by **clicking on a network host or application protocol** listed in the main page of the app: in this case, **appropriate filters will be automatically set** to display only the connections related to the clicked object.

Additionally, each connection is clickable in its turn to open a pop-up reporting further details about the item.



```
Connection details

17:15:01 - 17:19:17
Transport protocol:
    TCP
Application protocol:
    HTTPS
Transmitted data (incoming):
    23.3 K bytes
    123 packets

Source
Socket address:
    140.82.113.22:443
MAC address:
    e6:9a:dc:9c:9f:64
Fully qualified domain name:
    lb-140-82-113-22-iad.github.com
Administrative entity:
    GITHUB (ASN 36459)

Destination
Socket address:
    172.20.10.9:57215
MAC address:
    30:35:ad:dc:36:5c

Showing 1-20 of 51 total results
```

Figure 3.9 *- In-app pop-up with details about the clicked network connection*

Beyond including information already available in the general screen, the pop-up dedicated to a specific connection displays the first and last **timestamp** of data exchange, **MAC addresses** of the sender and the receiver (i.e., unique identifiers assigned to a network interface), and the **fully qualified domain name** (FQDN) of the remote node.

4. Project management

A software project not only requires efforts directly related to writing code, but also needs **proper management** in terms of documentation, packaging for different platforms, and support to users who incurred in some issues, just to name a few.

If in the early stages of the project most of the time was dedicated to programming, as Sniffnet is growing and its features becoming stable, an increasing amount of effort is dedicated to **non-coding tasks**, most of which have to do with the **GitHub repository management**.

Given that these aspects are essential and require a considerable amount of dedication, I believe it's unavoidable to dedicate an entire chapter to better describe them.

4.1. Documentation

Code documentation is an essential phase in the lifecycle of software.
It consists of textual or visual representations aimed at describing what a codebase does and how it can be used, improving user productivity and the overall software usability.

In the case of **libraries**, code documentation mainly consists of the description of the library's APIs and usually targets other developers.
In the case of **applications** such as Sniffnet, documentation is usually done at a higher level to be understandable by all the end-users and can come in various forms, some of which are described in the following sections.

4.1.1. The README file

Including a README is in general the first step to take to properly document a project.
The **README file** is often the first item to be seen when visiting a GitHub repository and its goal is to expose **what** the project does, **why** it's useful, **how** can users get started with it, and other potentially useful information. [38]
The README file on GitHub is usually written in **Markdown**, a lightweight markup language for creating formatted text that also allows displaying pictures.

The presence of this file is crucial not only to document the already interested users but also to attract new potential users.
To this purpose, it's essential for a README to be **useful** as well as **aesthetically pleasing**.
I honestly think that part of Sniffnet's success derives from having created a good-looking README since the early stages of the project: of course, this file's primary goal is to be helpful for the users, but the reality is that most people will never even read it if it's just a high wall of text.
When reading a newspaper or articles on the Internet, people don't want to go through the whole content of the page, but rather focus on what are the most interesting sections: if I wanted my README to be noticed, I knew I had to make it in a way such that it could **capture the attention of the users**.

Since the first section of the README to be seen is the upper one, I made sure to put there the most relevant **pictures** together with the project's **value proposition**, while keeping the most helpful yet "boring" content below.

A good practice is to include in the upper part of the file also some little SVG **badges** reporting concise information about the project (e.g., license type, download count, and latest version tag).

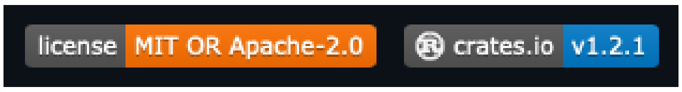


Figure 4.1 *- SVG badges on top of the README file, powered by shields.io. [39]*

The main body of Sniffnet's README, below the top-most part, consists of **different sections**, each one reporting different kinds of information as described in the following.

- **Installation**: this section describes how the application can be downloaded for use. It includes various install methods for the users to choose which one they prefer, depending on their machine's configuration, architecture, and operating system.
- **Required dependencies**: portion of the README reporting, for each operating system, the needed dependencies to install and other precautions to take to correctly build and run the application.
- **Features**: list of the main features and functionalities, to make it clear what the application does and how it can be helpful for the users.
- **Keyboard shortcuts**: section aimed at documenting the hotkeys available to make the user experience faster and more efficient.
- **Troubleshooting**: part reporting the most common problems that may occur and how they can be solved. It also reports, for each kind of problem, a link to the respective solved issues which already happened to other users.
- **Acknowledgments**: final section dedicated to shout-outs to Sniffnet's contributors.

Other minor sections, not mentioned in the list above, are also included in the README to document how more specific features of the applications work under the hood.

4.1.2. Release notes and the CHANGELOG file

Another important aspect is to document all the new features, changes, improvements, and fixes made to the software in the time.
After some changes are made to the codebase, it could be necessary to publish a **new version** of the application including such changes.

First of all, when a new version is released, there is the need to choose a proper and unique name for it.
Software versioning is the process of assigning version names or numbers to a specific state of a program, library, or application. [40]
Different numbering schemes exist to track software versions, and one of the most adopted is Semantic Versioning.
Semantic Versioning, also known as SemVer, is the scheme I choose for Sniffnet version numbers.
It encodes a version identifier into three different numbers separated by a '.' (dot) character: **Major.Minor.Patch** (e.g., 1.2.1).
In this versioning scheme, functionality and risk are the measure of an update's importance.
An update may introduce breaking changes (highest relevance) or may just patch minor bugs (lowest importance): based on the relevance of the introduced changes, the respective number of the version identifier is incremented.

Every release should list which changes it introduces and GitHub offers a way to automatically generate **release notes** based on the commit history.
However, it's suggested to write release notes by hand for them to be more engaging and interesting to read.
I personally write Sniffnet's release notes dividing them into distinct sections (new features, improvements, and fixes) and adding screenshots of the latest functionalities when applicable; this approach contributes to maintain the user base interested in following the evolution of the application.

Alongside the release notes there is the **CHANGELOG** file, usually placed in the root directory of the project.
The CHANGELOG (i.e., a log of the changes) lists every version of the software with its date of publication and all the apported modifications, and it can be considered as a history book about the software itself.

Beyond serving for documentation purposes, the CHANGELOG helps in debugging production bugs introduced in each software update, and immediately shows the cadence of such updates, without the need to search for every release page.

4.1.3. Other documentation resources

A project’s documentation can also appear outside the boundaries of GitHub, to reach people who are not directly involved in the FOSS ecosystem.
With this idea in mind, I created a **website** and registered a domain name (*sniffnet.net*) to reach a possibly wider and more heterogeneous audience.

One thing to always keep into account is to adjust the documentation according to the platform on which it’s hosted and the corresponding **target population**: on GitHub, it’s more likely to find people with a high level of technological expertise and as a consequence it’s possible to include computer engineering aspects, while a website is virtually accessible by any kind of users and therefor it should carry content which is more straightforward and easy to understand.
For this reason, I included in the website a less technical overview of the application, together with a greater variety of screenshots that could be appreciated by most of the public.

Another documentation resource is the **CONTRIBUTING** file on GitHub, whose purpose is to facilitate other developers contributing to the project.
This file contains guidelines to communicate how people can use their expertise to help a project maintainer, including submitting patches, developing new features, or simply opening a well-formed issue for a bug they found. [41]

GitHub also provides repositories with the possibility to set up **wikis**, a kind of extended documentation to share long-form content about the project: how it has been designed, which are its core principles, and additional information which are usually not reported in the project’s README. [42]
I’ve not published wikis for Sniffnet yet, but I’m definitely considering doing it, including more extensive and detailed descriptions of the application.

Finally, it’s important to note that GitHub **issues** and **pull requests** themselves are an extremely valuable sources of documentation that derive from the cooperation between the repository owner and the community.
To make issues and PRs even more helpful, GitHub gives access to mechanisms to properly manage them.
Some of such mechanisms are described more in detail in the next sections.

4.2. GitHub repository management

One of the consequences of Sniffnet's increasing adoption and popularity is the considerable amount of people seeking for help or willing to support the project themselves, and GitHub provides several ways of managing a community that is growing around software.
The following sections report my experience in managing the activity on Sniffnet GitHub repository.

4.2.1. Issues

GitHub issues are used to keep track of feedback, bugs, or tasks, related to a software project. [43]
Each issue has a title and is in the form of a discussion in which problems and ideas can be shared by anyone.
Issues allow to **organize and prioritize the work** to do and can be opened either by the project maintainer or by an end user to receive support or to suggest new features.
An issue is not only helpful for the person who directly opened it but can also be consulted by other users who will incur in the same problem at a different point in time.

The various issues flied to a repository **can be labeled** with one or more tags, each one with its color and short description, to easily categorize problems and feedback.
By default, GitHub provides several labels, such as *'bug'* to indicate something that's not working, *'duplicate'* to mark an issue that already exists, or *'enhancement'* to indicate a request for a new feature or improvement.
A project maintainer is free to create additional labels to satisfy his project-specific needs; in the case of Sniffnet, some custom labels were created for instance to recognize issues related to needed dependencies not being installed (*'missing dependency'*) or associated with the graphical renderer used by the GUI library (*'renderer'*).

Navigating issues of a repository can be done by filtering them with the corresponding tag, by specifying its creator, or the **issue state**.
At a given point in time, an issue is in one of the following states:

- **Open**, if the corresponding problem or feature request is not solved/completed yet.
- **Closed as complete**, in case the issue was solved.
- **Closed as not planned**, in the eventuality that the project maintainer is not planning to integrate the request into the project.

Each issue can also be **assigned** to a person in charge of solving it, and this is particularly useful in case of projects with more collaborators.

In the scope of the Sniffnet repository, most of the issues are about problems related to **missing or incorrect installation of the required dependencies**.
In such circumstances, it's just a matter of linking the users to the respective section of the documentation where it's explained in detail how to set them up.
To make it even easier for users to solve this kind of issues, I created a *Missing Dependencies* section below the *Troubleshooting* part of the README, where a link to the issues tagged with the respective label has been included. [44]
Since each **operating system** has its own peculiarities and required dependencies, I thought it was a good idea to add labels categorizing issues by the major OSs (*'Windows'*, *'macOS'*, *'Linux'*), so that users can immediately find problems related to their specific execution environment.

Another frequently raised problem is about incompatibilities with the *wgpu* **graphical renderer** adopted by Iced, the GUI library used for Sniffnet development.
Rendering issues usually cause widgets to display in a way they aren't supposed to, the whole screen blinking, and in the worst case they can make the application crash on start.
Iced set *wgpu* as the default graphical renderer because it's in general the most compatible one, but it also provides the *glow* renderer as an alternative.
After some time struggling with this category of issues, a secondary branch for Sniffnet (*glow-renderer*) was set up and it turned out that building the app using the alternative renderer was able to solve almost every issue related to graphical adapter incompatibilities.

Some GitHub issues were also opened directly by me to gather **feedback** and **help**.

An issue [45] that received particular attention was about the request for Sniffnet **translations**: I wanted the app to be available in more languages, but at the same time I didn't like the idea to translate it with automated tools that could potentially make the translation less reliable.
For this reason, an issue was opened asking native speakers to translate the graphical user interface; thanks to that request, today the application features 17 different languages, including German, French, Spanish, and Chinese.

Examples of other issues used to collect feedback are about tips **for aesthetic improvements** [36] and suggestions for **new keyboard shortcuts** [35] to include.

4.2.2. Pull requests

Pull requests (PRs) make it possible for everyone to apport modifications to a project codebase. [46]
Once a PR is opened, it's possible to **discuss** and **review** its modifications, as well as **add follow-up commits** before merging the changes in the base branch of the repository.

Pull requests can be labeled and filtered in the same way, with the same set of tags, as issues do.
PRs not only allow updates to the code itself but can also involve any other file existing in the repository, including documentation text and configuration assets.

Similarly to issues, also PRs have different possible states:

- **Open**, if the pull request is ready to be evaluated or is undergoing a discussion.
- **Draft**, when the apported changes are not yet ready for a deeper evaluation.
- **Closed**, in case the project maintainers don't want to integrate the modifications into their software for any reason.
- **Merged**, if after being evaluated by the repository owners, the PR changes are included in the project.

Pull requests are particularly helpful for project maintainers because they clearly show every modified file, permitting owners to add comments to each of the changed lines.
The process of a PR **review** can be more or less articulated, depending on the entity of the change.
The modified files can be singularly marked as viewed, so that a review can proceed in different steps, and it can be terminated in more than one working session.
After all the modifications have been considered, the repository maintainer can approve and merge the PR or request further changes before it can be reviewed again.
This process guarantees that the **code quality** of the submitted PRs is high enough to be incorporated without introducing bugs or vulnerabilities.

Since I'm the only long-term contributor behind Sniffnet evolution, most of Sniffnet PRs are generated by **automated tools**, described in the next section, to maintain the repository in a healthy state.
Another considerable fraction of the submitted pull requests came from the introduction of new **translations** for the applications. [47]
A particular mention goes to a PR by a guy who gave me substantial help in **automating the app packaging** strategy: the relative changes were merged after being long discussed with a review made of more than 50 messages. [48]

Issues and pull requests, together with the software source code, are the **core of a GitHub repository**.
Issues and PRs aren't two separate entities: a PR can specify a list of issues that could be solved by its merge, while an issue can refer to PRs to indicate that a particular bug or feature request is being considered.
A maintainer can also provide one or more templates to facilitate the creation of issues and PRs by the users, and each issue/PR is uniquely identified, inside a given repository, by a sequence number that can be used to immediately refer to it.

Finally, GitHub provides a way to group issues and PRs in the so-called **milestones**. [49]
Milestones are used to track the progress of a given set of tasks inside a repository, and each milestone usually refers to a future release of the software.
In this way, it's possible to easily keep an eye on the remaining work to do before publishing a new version of the application.

4.2.3. Automation with GitHub bots

Some of the tasks to manage a repository are **repetitive jobs** that can be **automated**.

One of such jobs is **keeping dependencies up to date**: each software project depends on some libraries, which in their turn depend on other ones, each coming with a specific version.
Maintained libraries will be updated sooner or later, introducing support for new features, or simply fixing bugs.
It's of key importance to make sure that a project's dependencies are not outdated, since software that is not renewed for a long time is more susceptible to **faults** and **security vulnerabilities**.

Performing a check on the version of every dependency by hand is not efficient at all and is very likely to be permanently postponed.
GitHub offers therefore the possibility to configure **Dependabot** [50], which as the name suggests is a bot to manage dependencies.
Dependabot will automatically generate a pull request when it detects an outdated dependency and can also be enabled to produce **security alerts** in case a vulnerability has been identified in one of the project dependencies.

In order to receive support from Dependabot, a file named *dependabot.yml* must be created inside the *.github* folder under the project root.

```yaml
# Maintain dependencies for GitHub Actions
- package-ecosystem: "github-actions"
  directory: "/"
  schedule:
    interval: "daily"

# Maintain dependencies for cargo
- package-ecosystem: cargo
  directory: "/"
  schedule:
    interval: "daily"
  target-branch: "main"
```

Code snippet 4.1 *- Content of the dependabot.yml file*

Dependabot can be configured to check for **dependencies updates** with a custom frequency, targeting a specified branch of the repository, and including dependencies from different ecosystems.
In the case of Sniffnet repository, Dependabot is set to open pull requests to update the version of Rust libraries present in the manifest file and to renew the version of tools used for GitHub Actions (discussed later).

There is a variety of other bots, available in the form of third-party applications, that can facilitate additional tasks; two of them, used for the maintenance of Sniffnet repository, are ImgBot [51] and AllContributors. [52]

- **ImgBot** is a tool that can be installed from the GitHub marketplace to automatically open PRs **optimizing all the images** present in a repository. This bot can detect if the size of one or more of the available images is unoptimized and will perform compression in case of need; by default, ImgBot compresses images using lossless compression, which allows to reduce the image size while causing no harm to the original quality.

- **AllContributors** is a project born to **recognize every single contributor** to a software project, not only those who directly apport modifications to the source code (as the GitHub interface does).
  Acknowledging every single contributor can be tedious; to simplify this process, the AllContributors team created a bot that can be tagged from any issue or pull request to add a user to the list of project contributors.
  Every contributor can be associated with one or more contribution types, and the AllContributors bot will take care of producing a dedicated section in a predefined file reporting every contributor's profile picture and the associated kind of effort made to support the project.

4.2.4. Automation with GitHub Actions

GitHub Actions is a continuous integration and continuous delivery (**CI/CD**) platform to automate workflow runs. [53]
An Actions **workflow** can be configured to activate every time predefined events happen in a repository, such as a new commit being pushed, or a PR being opened.
Each workflow is made up of one or more jobs that are run sequentially or in parallel by a **virtual machine** (runner) hosted on GitHub servers.

The workflows to be executed and the events that trigger them are defined by the repository owner in the *.github/workflows* folder, which may contain multiple workflows, each performing a different set of tasks.
For what concerns Sniffnet repository, two different workflows have been set up: ***rust.yml*** and ***package.yml***.

The first one oversees that the project compiles successfully and that the code follows the most opinionated guidelines.
More specifically, the *rust.yml* workflow performs a **build** of the project, followed by a check on the **code format** and **patterns** used, and finally runs all the available **tests**.

Particularly interesting are the Cargo subcommands used to control the code format and patterns:

- ***cargo fmt*** [54], which allows formatting Rust code according to stylistic guidelines and raises warnings in case some pieces of the code aren't properly formatted.
- ***cargo clippy*** [55], a powerful code linter for Rust to make sure that the project under evaluation follows the most idiomatic programming patterns.
  Clippy can identify common semantic and syntactic imprecisions from a collection of more than 600 code lints, and it suggests possible improvements.

The *rust.yml* workflow is triggered at every commit on any of the repository branches and every opened pull request.

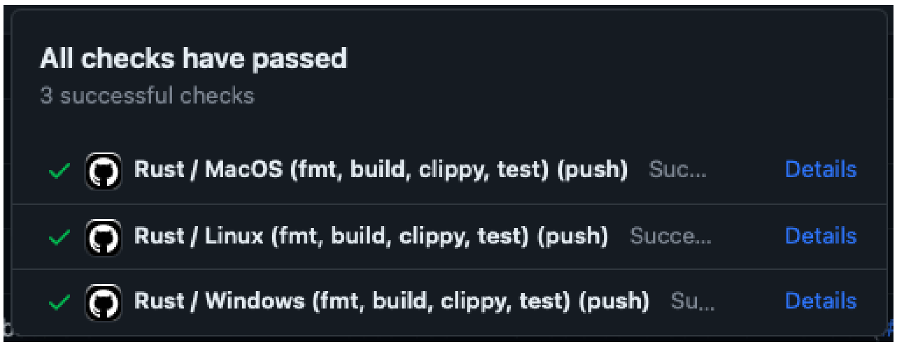

All checks have passed
3 successful checks
✓ Rust / MacOS (fmt, build, clippy, test) (push) Suc... Details
✓ Rust / Linux (fmt, build, clippy, test) (push) Succe... Details
✓ Rust / Windows (fmt, build, clippy, test) (push) Su... Details

Figure 4.2 *- A successful workflow run of the rust.yml GitHub Action*

Each workflow job is executed on three different runner VMs (*windows-latest*, *macos-latest*, and *ubuntu-latest*) to ensure that parts of the code that compile conditionally only on a specific operating system don't contain flaws.

In case one of the workflow steps raises a warning or an error, an alert is shown near the respective triggering event, and a notification is sent via e-mail to the repository maintainer.
Putting in place such a process contributes to making the codebase maintenance more **scalable**, **robust**, and **secure**.

The second workflow of Sniffnet repository, whose configuration is specified in the *package.yml* file, takes care of **creating application packages** for the different existing architectures and OSs.
More details about the app packaging strategy are described in the next dedicated section, since I consider it to be a topic that requires a more in-depth explanation.

4.3. Application packaging

Distributing an application, making it available for other users (*packaging*), is an essential part of a software life cycle.
Particularly important in this context is to provide more than just one single method of distribution, assuring that more people, possibly using different architectures and operating systems, can benefit from the use of the application.

4.3.1. Cargo crate

In the early stages of the project, Sniffnet was only available to download via **Cargo**, the Rust package manager. [56]
Cargo is responsible for compiling a package, making it distributable, and uploading it to *crates.io*, the Rust community's package registry. [57]
Managing and distributing packages, commonly known as ***crates*** in the Rust ecosystem, is extremely straightforward thanks to Cargo: after having obtained an API token for *crates.io*, it's just necessary to add some **metadata** in the *[package]* section of *Cargo.toml*, the manifest file of the project. [58]
Sniffnet's relevant metadata used for packaging the crate are reported below.

```
[package]
name        = "sniffnet"
version     = "1.2.1"
authors     = [ "Giuliano Bellini" ]
edition     = "2021"
description = "Application to comfortably monitor your network
traffic"
readme      = "README.md"
repository  = "https://github.com/GyulyVGC/sniffnet"
license     = "MIT OR Apache-2.0"
keywords    = [ "filter", "network", "packet", "sniffer",
"gui" ]
categories  = [ "visualization", "gui", "network-programming"
]
include     = [
    "src/**/*",
    "LICENSE-*",
    "README.md",
    "CHANGELOG.md",
    "resources/countries_flags/**/*",
    "resources/DB/*",
    "resources/palettes/*",
    "resources/fonts/subset/*",
    "resources/sounds/*"
]
```
Code snippet 4.2 *- Sniffnet's manifest [package] section*

The *name* field specifies the identifier of the Rust crate and must be unique, while the *version* field reports the package version tag.
Other fields include the author of the project, the Rust edition used to compile the package, a brief description of what the project does, references to the package README, repository, and license files, as well as the categories assigned to the crate and the list of files to include when packaging it.

After having included these parameters in the manifest of the project, it's enough to launch the ***cargo publish*** subcommand to upload it to the package registry.

This approach has the advantage of being fast and painless for the developer, but it also carries a huge **limitation**: the generated crate can only be compiled and installed by users who already have **Rust available on their machine**.
To overcome this restriction, I decided to package Sniffnet with additional and more widespread mechanisms.

In order to package software for a given **operating system**, it's necessary to build a version of the binary compatible with the specific OS: an application binary built on macOS is not compatible with Windows, for instance.
To solve this problem, I initially built manually a different version of the application for each of the main operating systems, using Virtual Machines to generate compatible binaries.
However, I later realized that packaging the app manually at every release was cumbersome and time-consuming: for this reason, I decided to set up an **automated GitHub workflow** to do it.
The workflow to package the application, differently from the workflow to check code correctness, doesn't activate at every commit but is set up to be triggered on demand, and this typically happens when a new version of the app is released.

4.3.2. Windows Installer

A **Windows Installer** is the main component used for the installation, maintenance, and removal of software on Windows operating systems.
Providing such an installation method, allows Windows users to easily download the application even if they have not installed Rust on their machine.

Luckily, there exists the ***cargo wix*** subcommand, whose goal is to help Rust developers create Windows Installers for their projects. [59]

To correctly work, the *cargo wix* subcommand needs the **WiX toolset** [60], which is installed on the GitHub action runner with the following instruction:

```
- name: Install dependencies
  shell: powershell
  run: |
    Write-Host "::group::WiX Toolset"
    Invoke-WebRequest `
      -Uri
"https://github.com/wixtoolset/wix3/releases/download/wix3112rtm/wix311-
binaries.zip" `
      -OutFile "$env:TEMP\wix-binaries.zip" -Verbose
    Expand-Archive -LiteralPath "$env:TEMP\wix-binaries.zip" -
DestinationPath "$env:TEMP\wix" -Verbose
    Set-Item -Path env:Path -Value "$env:Path;$env:TEMP\wix"
    Write-Host "::endgroup::"
```
Code snippet 4.3 *- Command to install the WiX toolset on the GitHub action runner*

By default, *cargo wix* generates a very basic version of the installer; to produce a more complex and customized installer, it's necessary to write an XML formatted document, characterized by the *.wxs* extension, containing instructions to set the application **icon**, create **shortcuts** to the executable, specify the **license file**, and to perform other useful custom actions.

One of the main problems that occurred during this process is that to correctly function on Windows, Sniffnet needs **Npcap** [61] as an external dependency, but it's not possible to download it through a custom action of the installer because **the free version of the library doesn't allow its installation in silent mode** (that is the ability to install Npcap from the command line).
The reasons behind this choice by the Npcap team were clearly explained in a Sniffnet GitHub issue by one of the library maintainers:
"Npcap is a commercial project developed by full-time paid programmers.
We fund this work by selling two types of license: the Npcap OEM redistribution license [62] *(for redistributing Npcap within other software or hardware products) and the internal-use license (for using Npcap within an organization beyond the limits of the free version).*
These license sales are critical to pay our expenses so Npcap remains viable and does not meet Winpcap's fate.
While we wish we could allow every person and organization to use and redistribute unlimited copies of Npcap, we would go out of business and it would end up abandoned like WinPcap was."

Considering that Npcap paid licenses are quite expensive, I ended up including in the final page of the installer a warning for the users to manually install this required dependency, together with a **checkbox to optionally open the Npcap download page**.

4.3.3. Apple Disk Image

Apple Disk Image is a disk image format commonly used to mount volumes in the macOS operating system.
A disk image is a compressed copy of the contents of a folder and the process of mounting is what allows an OS to make files and directories on a storage device available for users through the file system.

Apple disk images can be useful to mount different kinds of content and usually have the ***.dmg*** file extension.
A disk image can also be used to install applications and in such cases, it contains a ***.app*** file.
A *.app* file is recognized by macOS as a **package bundle**, which is a special directory containing an executable and other useful resource the binary may need.
More specifically, the bundle requires an ***Info.plist*** (information property list), which is a structured file containing configuration information about the app, and the **executable** itself, that includes the application entry point.
Beyond these two required artifacts, the package may contain other support files: this is the case of Sniffnet's bundle, whose content is reported in the following:

```
Sniffnet.app
└── Contents/
    ├── Info.plist
    ├── MacOS/
    │   ├── sniffnet
    │   └── wrapper.sh
    └── Resources/
        └── sniffnet.icns
```

The main executable binary consists of the *sniffnet* file, but in the case of this package the real entry point of the bundle is the file ***wrapper.sh***.
In fact, Sniffnet needs **administrator privileges** to monitor network adapters on macOS and therefore it's necessary to launch the app as a *superuser*, which wouldn't be possible if the *sniffnet* file was called directly.
For this reason, it's been necessary to create a thin wrapper able to spawn a system window prompting for a password: if the password inserted by the user corresponds to that of the admin account, the application will be started with elevated privileges.

```
#!/usr/bin/env zsh
osascript -e "do shell script \"/*/Sniffnet.app/Contents/MacOS/sniffnet
>/dev/null 2>&1 &\" with administrator privileges"
```

Code snippet 4.4 *- Content of wrapper.sh, used to launch Sniffnet with admin privileges on macOS.*

The *Info.plist* file is where it’s specified that the bundle entry point is *wrapper.sh* and that the **icon** of the package corresponds to the file located in the *Resources* folder.
The information property list also specifies the app **short version string**, directly taken from the version field of the project’s manifest, and the **long version string**, assigned in this case to the hash of the GitHub commit against which the workflow runner is triggered.

Once the application bundle is built, it must be included inside the disk image for distribution.
This last step has been achieved with the help of ***create-dmg*** [63], a shell script to build good-looking disk images for macOS which can receive several parameters:

```
create-dmg \
  --volname "Sniffnet Installer" \
  --background "resources/packaging/macos/graphics/dmg_bg.png" \
  --window-pos 200 120 \
  --window-size 900 450 \
  --icon-size 100 \
  --app-drop-link 620 240 \
  --icon "Sniffnet.app" 300 240 \
  --hide-extension "Sniffnet.app" \
  "artifacts/sniffnet-${target%%-*}.dmg" \
  "target/${target}/release/bundle/osx/"
```

Code snippet 4.5 - *Command to set up the disk image for macOS*

The resulting *.dmg* also contains a link to the *Applications* folder of the target machine, so that it’s possible for the users to easily **drag and drop** Sniffnet into the list of their apps.

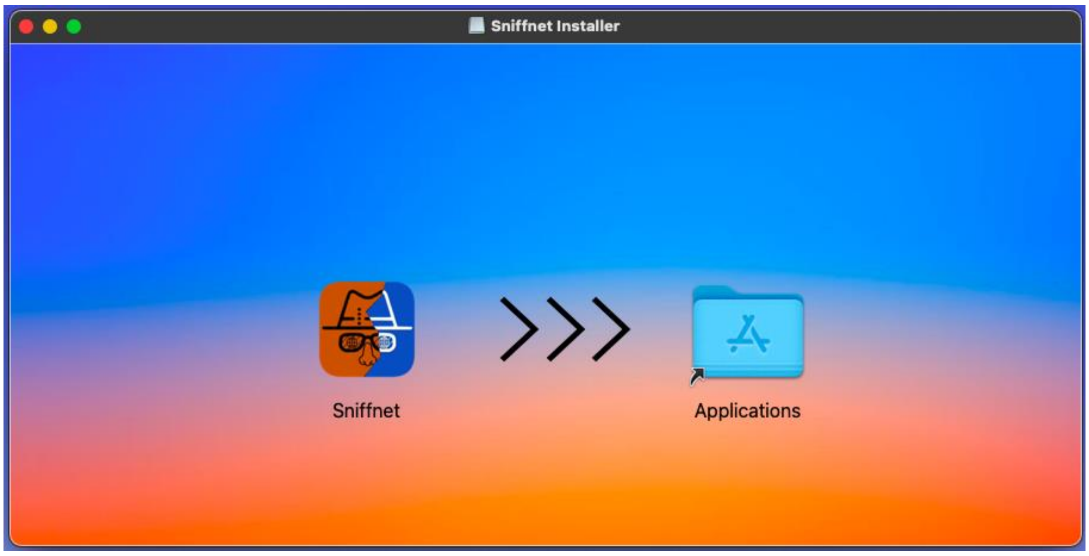


Figure 4.3 - *Sniffnet disk image for installation on macOS*

4.3.4. Linux packages

For what concerns **Linux**, the matter of packaging the application is slightly more complex, due to the variety that characterizes its environment.
Linux is an open-source operating system kernel on the top of which several different **distributions** are built; a Linux distribution is typically made of a Linux kernel, GNU tools and libraries, a window system, a window manager, and a desktop environment. [64]

The different distributions, commonly referred to as *distros*, can be divided into two main classes:

- **Red Hat-based**: including Linux Fedora, which use the ***.rpm*** file format.
- **Debian-based**: including Ubuntu, Mint, BackBox, and Kali, which use the ***.deb*** package format.

With the purpose to cover the main distributions of these two classes, I included the creation of the *.deb* and *.rpm* packages in the automated GitHub workflow.
Two Rust crates, respectively ***cargo-deb*** [65] and ***cargo-generate-rpm*** [66], have been used in the workflow to build the packages.
Both utilities permit specifying various parameters in the project manifest to enrich the generated package, including the application category and description, its dependencies, the license file, the icons to use, and other assets.

The first versions of these packages didn't allow to launch the app directly but instead opened a terminal asking the user to insert a password.
This was required for reasons like macOS: on Linux, the app needs administrator privileges as well to inspect a network adapter.
However, the most recent packages are built in a way that makes it possible to launch Sniffnet without needing the *sudo* command.
To permit this behavior, a short **post-installation script** has been introduced; the task of the script consists in modifying the executable capabilities to allow it to perform various network-related operations.

```
#!/bin/sh
set -e
setcap cap_net_raw,cap_net_admin=eip /usr/bin/sniffnet
```
Code snippet 4.6 *- Post-installation script used to set network inspection capabilities to the Linux executable.*

Many other classes of distributions do exist, comprising **Pacman-based** that include Arch Linux.
I didn't personally package Sniffnet for the Pacman package manager, but it has been kindly done by a folk who takes care of maintaining packages for Arch Linux.

5. Sniffnet’s adoption by the community

I’ve already mentioned how Sniffnet was originally born as an academic project.
After the project delivery and discussion, I thought it was a good idea to **share the code with the community to get feedback**.
Since the first public announcement, Sniffnet unexpectedly **raised a lot of interest** and this motivated me to keep developing it in my spare time, until it was elected for the GitHub Accelerator Program.

This chapter contains a discussion about the stunning growth of Sniffnet, which led the project to reach more than **10 thousand stars** on GitHub and **50 thousand cumulative downloads**. (*)

(*) Data are updated to the month of July 2023

5.1. The project publication

I wanted as many people as possible to see the project, in order to get **multiple points of view** about my work.

Gaining some early adopters would have made it possible to notice the presence of **eventual bugs,** and to get opinions about **new features** to include.

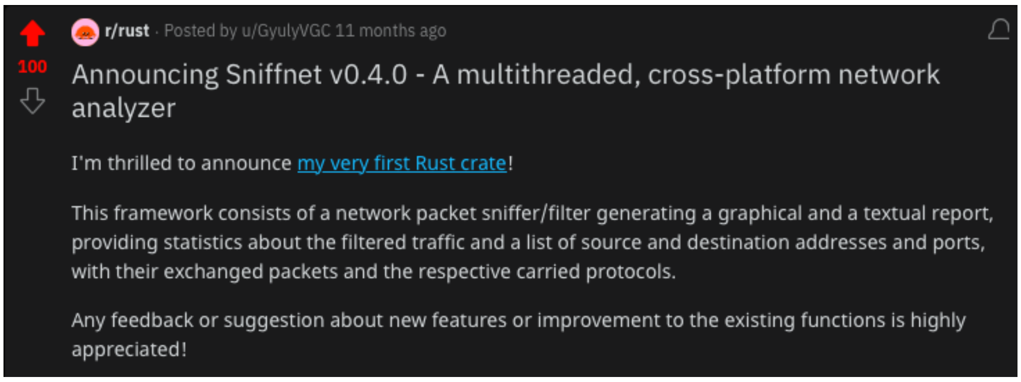
5.1.1. The first announcement

With this idea in mind, I decided to share Sniffnet with the **Rust community of Reddit**, one of the most frequented resources among Rust programmers.
Reddit [67] is a network of communities where people can follow their interests and it's based on **content rating**, that is the practice of letting users vote positively or negatively each post, which will become more or less popular as a consequence.

Two reasons brought me to choose Reddit to spread the project:

- Sharing Sniffnet with a specific community mostly made of Rust programmers was the best way for me to **learn from more experienced people**.
- Reddit's **rating system** would've automatically told me if people were interested in what I was doing.

The **first public announcement ever** [68] (made on the 11TH of September 2022) got a surprisingly high amount of appreciation, receiving a hundred upvotes on Reddit and several thousand views, after being featured on the community front page.
The announcement was about the 0.4.0 release of Sniffnet, which at that point was in the form of a CLI.


r/rust · Posted by u/GyulyVGC 11 months ago

100

Announcing Sniffnet v0.4.0 - A multithreaded, cross-platform network analyzer

I'm thrilled to announce my very first Rust crate!

This framework consists of a network packet sniffer/filter generating a graphical and a textual report, providing statistics about the filtered traffic and a list of source and destination addresses and ports, with their exchanged packets and the respective carried protocols.

Any feedback or suggestion about new features or improvement to the existing functions is highly appreciated!

Figure 5.1 *- The very first public announcement of Sniffnet [68]*

The program was liked despite its features were **still very limited and simple**, and this made me realize it had a high potential that could've been better exploited with the addition of further functionalities.

5.1.2. The announcement of the graphical user interface

Motivated by the success of the first shared version of the program, I started developing a **GUI** for the application, which took about a month of development before being finally published on the 21st of November, tagged as version 1.0.0.

I was so thrilled to share the first stable version of the app that I posted the announcement not only on the Rust subreddit, but also in **other communities** focused around open-source, programming, and software in general.

The **appreciation** for the new release was even bigger than the support received by the previous version.
The day after the announcement, a popular German IT website (***heise.de***) published an article about Sniffnet [69], putting it under the spotlight of even more readers.

Netzwerkmonitoring mit Sniffnet: Open Source und komplett in Rust geschrieben

Mit Sniffnet erscheint eine neue Überwachungs-Software für den Netzwerkverkehr. Sie ist betont simpel gehalten, soll dank Rust aber besonders stabil sein.

Lesezeit: 2 Min. In Pocket speichern 13

***Figure 5.2** - Cover of the article about Sniffnet published on heise.de, a popular German journal about technology [69]*

In that occasion, Sniffnet's repository was visited by an incredible quantity of developers, to the point of being featured twice on the overall **GitHub trending page**, a daily updated space to host 25 of the most exciting open-source projects on the platform.
The GitHub trending page [70] is consulted every day by thousands of programmers, and being featured is a unique opportunity for a project to be known by the world.

Shortly after, Sniffnet has also been the subject of a **tweet by GitHub** itself [71] and of **several additional articles**, that are listed in *Appendix B* of this thesis.

5.2. Evaluation of the project's adoption

The **popularity** or **adoption** of software is intended as the number of people that use, have contributed to, or are interested in a specific application or program.

There exist multiple ways to **evaluate the level of adoption** of an open-source project, some of which are discussed in the following paragraphs.

5.2.1. Why is the measure of popularity relevant?

Different **motivations** justify the importance of doing such an evaluation before deciding to make use of an application:

- Software that is widely used has more chances of being **constantly maintained** over time.
  Constant maintenance guarantees a faster bug-fixing process, as well as a more frequent introduction of new features and prompt support to the users.
- A program used by nobody has no assurance of being free of **malware** or other **security vulnerabilities** that could lead to data corruption or the disclosure of sensible information.
  Conversely, a popular application is unlikely to contain code intentionally written to harm the users, even if it's not possible to exclude the presence of unintentional vulnerabilities that could also be due to one of the software dependencies.
- Last but not least, one must consider that **given a specific need or field of use there are many applications available**, each with its peculiarities but all performing similar tasks.
  If a software is popular (say for example Wireshark) in its belonging context (say for example network monitoring), this means that a considerable number of people has chosen it **over its competitors**.
  If an application has been selected instead of another one to carry a task, in turn this is a signal that the app has unique features or does something better than the competitors.

Despite the level of adoption being a good indicator of the health of a codebase, these motivations don't want to point out that a popular project is *always* preferable with respect to a less famous one, or that the goodness of an application is *strictly proportional* to its popularity.

Another factor that must be considered to have a more comprehensive view is for instance the **project's maturity**, since it's obvious how younger applications have less chances of being as popular as a more dated one.

5.2.2. GitHub Stars

Starring a GitHub repository is a way to easily find it again later in a personal collection thought to host the **favorite** projects of a user.
Leaving a star to a project also shows **appreciation** and **support** to the repository maintainers for the work they are doing.

When starring a repository, a user becomes a **stargazer** of the project and can decide to save it in a **dedicated list** related to a specific topic, to organize different liked projects by their nature or functionalities.

The only requirement to star a repository is to have an active GitHub account and this means that *stargazers aren't necessary users of the application/library*.
A GitHub star can in fact assume **different meanings**: the star could have been given by an active user of the project, by someone who plans to use it in the future, or maybe by a person who finds the idea cool but will end up never trying it directly.
Therefore, a star can be interpreted as a signal analogous to "likes" in the context of social media, which is nothing less and nothing more than a form of positive feedback by a user of the platform.

Despite being "just" an indicator that *someone cared enough to click on a button*, GitHub stars are by far the most influential **metric to immediately create a good first impression** about a repository.
Rapidly looking at the number of stars is the first thing that most of the visitors of a repository do, including myself.
A research study [72] dated back to 2018 surveyed more than 700 developers and confirmed the relevance of this metric: *"We report that three out of four developers consider the number of stars before using or contributing to a GitHub project"*.

The quantity of GitHub stars is influential to the point of determining not only the possible **adoption** of software, but also the **willingness of contributing** to it and its overall **public image**.

For these reasons, while repositories with lots of stars are appraised positively, projects with few stars are extremely less likely to be considered.
Every new project starts of course with a star count of zero and this makes it a challenge for it to be noticed at first.
The vast majority of GitHub repositories don't break the wall which stops them from gaining a sufficient amount of initial traction: in this sense, it can be asserted that stars create a **vicious circle** for which popular projects become even more renowned, while unpopular repositories are meant to remain so.

It was a huge surprise and pleasure for me to see that, following the first publication of the project, Sniffnet was appreciated by several early adopters and gained the first hundred stars.
This allowed the application to obtain **initial credibility** before the most important releases happened shortly after, which definitively made the project explode in popularity.

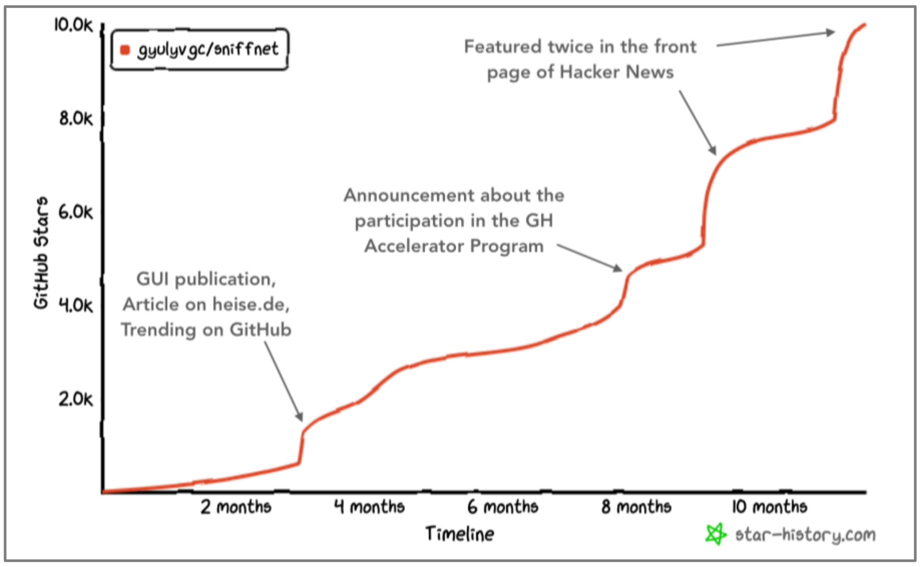


Figure 5.3 *- The number of GitHub stars of Sniffnet from August 2022 to July 2023*

The chart above, realized with the help of Star History [73], reports the **growth** of Sniffnet GitHub stars over time (*updated to the month of July 2023*).
It can be observed how there are certain **spikes** in the chart, corresponding to **citations** of the project in articles from blogs and websites (the list of such mentions is available in *Appendix B*).
Particularly notable was the rise in popularity after the project was featured, in two different occasions, on the front page of Hacker News [74], a social news ICT website visited by millions of people every day.

One additional chart, displayed on the right, **compares** the star growth of the repository with that of **other popular network analyzers**, underlining even more the exceptional escalation of Sniffnet on GitHub.

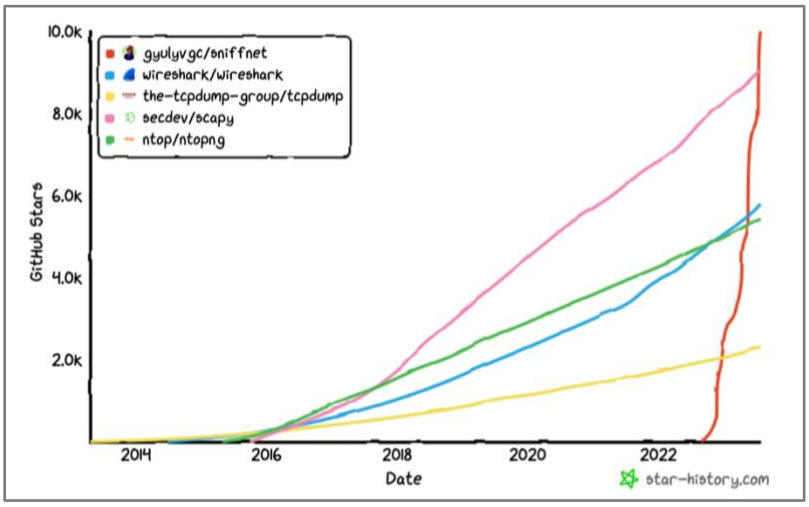


Figure 5.4 *- GitHub stars over time of Sniffnet compared to other popular network monitoring tools*

5.2.3. Downloads

While, as previously discussed, GitHub stars can have different nuances, the **number of downloads** of a library or an app is a much more **direct indication** about the actual quantity of users of the software.

In section 4.3 — Application packaging — it's been talked over the **multiple channels of distributions** of Sniffnet and some of these channels allow for an easy **tracking** of the corresponding download number.

That's the case of ***crates.io*** for example, whose interface shows the daily downloads over the past 90 days as well as the total number of installations of a given crate.
Sniffnet's source binary, published on *crates.io*, received **7 thousand downloads** in the 11-months period after its first release.
For a user to download a binary in the form of a crate, it's necessary to have a working installation of Rust on the target machine: this tends to limit the reachable audience and is the reason why more popular installation methods were considered.

One of the said methods is **GitHub releases**, which permits the upload of one or more artifacts hand in hand with the publication of the respective code version.
Every time a new release is made public, I take care of including multiple **packages**, each related to a different platform or architecture: packages for Windows, macOS, Red-Hat based Linux, and Debian based Linux are currently available, both for 32-bit and 64-bit architectures.
Each uploaded artifact can be downloaded by directly clicking on the highlighted link in the release page and is accessible from a URL whose structure is known a priori, so that it can be easily shared and linked to from different places.

GitHub provides a long and comprehensive list of **APIs** to access data of various kinds, including detailed **information about all the releases** of a repository, which can be consulted altogether or singularly by providing a specific release tag.
In particular, the endpoint to get details about all the releases of a repository is *https://api.github.com/repos/OWNER/REPO/releases*, where *OWNER* and *REPO* must be replaced respectively with the name of the project owner and of the repository.
The returned value is in the form of a complex JSON object composed of a numerous set of fields specifying the releases' URLs, the dates of creation, information about the authors, the name and body of each release, and many more parameters.

Among the different attributes, it's featured the download count for each of the releases' artifacts, along with their names.

To clean up the output obtained from the endpoint, I wrote a simple **bash script** that makes it possible to keep only the information needed to determine the **number of downloads divided by release and by artifact**.

```
$ curl -s https://api.github.com/repos/gyulyvgc/sniffnet/releases |
egrep '"name"|"download_count"'
```

Code snippet 5.1 - Bash script to retrieve the number of downloads related to artifacts uploaded on GitHub releases

The script simply filters the original JSON object keeping only the lines that start with specific strings, and the result looks like the following:

```
    "name": "v1.2.1",
            "name": "Sniffnet_LinuxDEB_amd64.deb",
            "download_count": 1639,
            "name": "Sniffnet_LinuxDEB_arm64.deb",
            "download_count": 167,
             ...
             ...
            "name": "Sniffnet_macOS_AppleSilicon.dmg",
            "download_count": 1456,
            "name": "Sniffnet_macOS_Intel.dmg",
            "download_count": 868,
            "name": "Sniffnet_Windows_32-bit.msi",
            "download_count": 275,
            "name": "Sniffnet_Windows_64-bit.msi",
            "download_count": 4913,
    "name": "v1.2.0",
            "name": ...
```

After having played with additional scripts, I also managed to obtain:

- The total number of downloads considering all the artifacts and all the releases.

```
$ curl -s
    https://api.github.com/repos/gyulyvgc/sniffnet/releases |
    egrep 'download_count'  | cut '-d:' -f 2 | sed 's/,/+/' |
    xargs echo | xargs -I N echo N 0  | bc
```

Code snippet 5.2 - Script to retrieve the cumulative download number from GitHub releases

- The total number of downloads related to artifacts with a specific file extension.

```
$ curl -s
    https://api.github.com/repos/gyulyvgc/sniffnet/releases |
    egrep '"name": ".*.dmg"|"download_count"' |
    egrep -A1 'name' | egrep 'download_count' |
    cut '-d:' -f 2 | sed 's/,/+/' | xargs echo |
    xargs -I N echo N 0 | bc
```

Code snippet 5.3 - Script to count the downloads of artifacts with a given extension (.dmg in the example), useful to verify the level of adoption among users of the different Oss

The returned values, updated to the end of July 2023, confirm that GitHub releases are the most common channel for Sniffnet users to install the app, with more than **32 thousand downloads**.
The partition of downloads based on the different OSs is reported in the pie chart below.

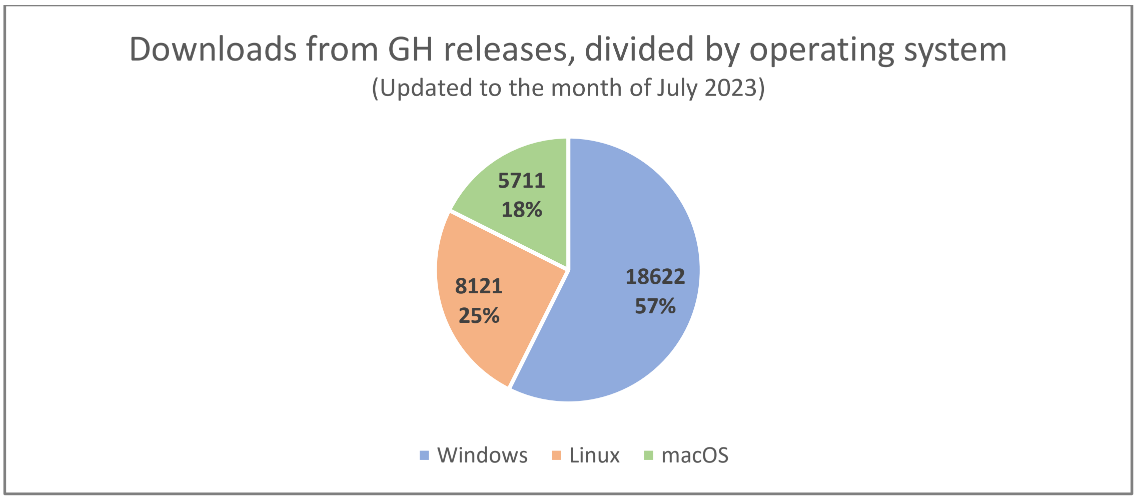


Figure 5.5 *- Pie chart reporting the download count of Sniffnet for the different OSs (from GitHub releases)*

The lower prevalence of macOS can be explained by considering that Sniffnet is also available for installation from **Homebrew** [75], a popular package manager for macOS from which the application was installed more than **9 thousand** times.

Considering the data available from *crates.io*, GitHub releases, and Homebrew, it turns out that Sniffnet was installed almost **50 thousand times in a period of 11 months**, with **an average of 150 downloads per day**.

The application can also be installed from **other distribution channels** which don't seem to have a feature to track the exact number of downloads, including the official package managers for FreeBSD, NetBSD, and Arch Linux.

Overall speaking, these numbers give an **esteem** about the amount of users of the software, but don't say anything about the frequency of use of the app.
Furthermore, it must be considered that the real number of distinct users is for sure lower, since the previously cited count is cumulative and refers to all the different versions of the application.

It's also interesting to consider that **the growth in terms of GitHub stars is not a well-defined function of the size of the actual audience**: to make an example, Sniffnet has many more stars than Wireshark, but Wireshark has millions of users, a number that probably won't be ever reached by Sniffnet.

5.3. Sniffnet's official website

At the end of May 2023, I started thinking about setting up an **official website** for the project, to reach an even broader public.

The **motivations** for this choice are multiple:

- The documentation on GitHub is limited to being displayed as a **Markdown document**, without allowing more articulated layouts.
- The repository README on GitHub is displayed after the **list of files and directories** of the project, and as the size of this list grows it's more and more difficult for a visitor to immediately notice the README.
- A website is generally more **user-friendly** than a GitHub repository and has more chances of being seen by non-programmers.
- Having an official website contributes to building a **stronger and more trusted brand image** for the application.

I finally decided to publish the website after the recommendations received during the GitHub Accelerator Program [76] — discussed in the next chapter — in which experts reminded us how having a website is essential to highlight the project's qualities and distinctive functionalities.

GitHub provides free access to **GitHub Pages** [77], a static site hosting service that builds HTML, CSS, and JavaScript files from a repository and publishes the resulting website.
Pages dedicates, to each GitHub account, a **special repository** for this purpose, whose name must be in the form *username*.github.io, where *username* is the GitHub account handle.
By default, the generated website is available at *https://username.github.io*, and additional **steps** are required to use a **custom domain name** for the site:

1. A proper name must be first **registered** at a web hosting company; in the case of my project, I bought the domain from **Aruba** [78], an Italian company managing domain registration businesses.
2. A ***CNAME* file** must be added to the root of the repository, in which it's reported the name of the chosen domain.
3. A ***CNAME* DNS record** pointing from the custom domain to the standard one must be set by navigating to the website's DNS provider.

Having a website without a dedicated domain name is like making a paint without putting a signature on it: an adequate name makes the site more **credible**, **recognizable**, and helps improve **SEO**, which is the set of practices guaranteeing a high exposure of the website in the scope of search engines.
After having considered some alternatives and having discarded some options because already taken, **sniffnet.net** was chosen as the official domain of the project.

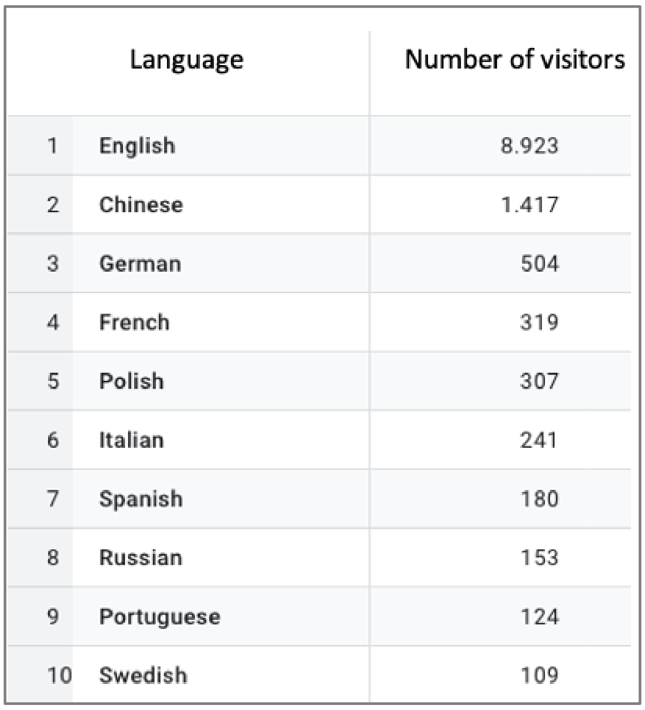
Despite having set up the website late, about 9 months after the first publication of the app, a non-negligible amount of traffic was registered, with **13 thousand unique visitors** in the first 60 days of activity.

Since the website publication, a notable **rise in the download count** has been observed and that's probably because making the installation links clearly visible helps reach more users, as opposed to simply having the links in the release pages of the repository.

Furthermore, enabling a **web analytics service** it's possible to get statistics and insights about the traffic to the domain.
This is useful for instance to discover how users learned about the application, which are the most visited pages inside the domain, what's the average time per visit, and other details potentially interesting from a marketing perspective.
I chose **Google Analytics** [79] for this purpose because it's simple, fast to configure, and free of charge.

Among the most useful information offered by Google Analytics, there are the traffic sources, the country of the visitors, and attributes about the device used to connect to the domain.

The metric about the visitors' **mother tongue**, for example, has been helpful to verify that Sniffnet was translated into the most common languages.
The table on the right features the top 10 languages by number of visitors, each of whom is today available in the application.



	Language	Number of visitors
1	English	8.923
2	Chinese	1.417
3	German	504
4	French	319
5	Polish	307
6	Italian	241
7	Spanish	180
8	Russian	153
9	Portuguese	124
10	Swedish	109

Figure 5.6 *- Table featuring the top 10 most used languages by the visitors of the domain (data gathered by Google Analytics)*

6. Participation in the GitHub Accelerator Program

The **GitHub Accelerator** [76] is an initiative to help software developers sustainably work on their projects full-time.
More specifically, it consists of a 10-week program in which 20 open-source teams from all around the world receive an **initial sponsorship** to work on their personal projects, paired with **guidance and workshops** from open-source leaders.

As Sniffnet was selected to participate in the first cohort of this program, from middle April to late June 2023, I had the incredible opportunity of getting precious insights and mentorships about working in the open-source ecosystem.

This chapter includes the most interesting takeaways about the Accelerator program itself.

6.1. Call for applications and selection process

The Accelerator program was first announced in November 2022 during **GitHub Universe**, the annual event dedicated to developers in which new features and initiatives are made public. [80]

Any maintainer or contributor of a GitHub repository was called to participate in the selection process for the Accelerator before the application deadline set on December 31st, with the possibility to apply also as a team composed of a maximum of 3 people. The only requirements to be met were not to be employed by GitHub itself and to be located in one of the 68 countries supported by GitHub Sponsors.

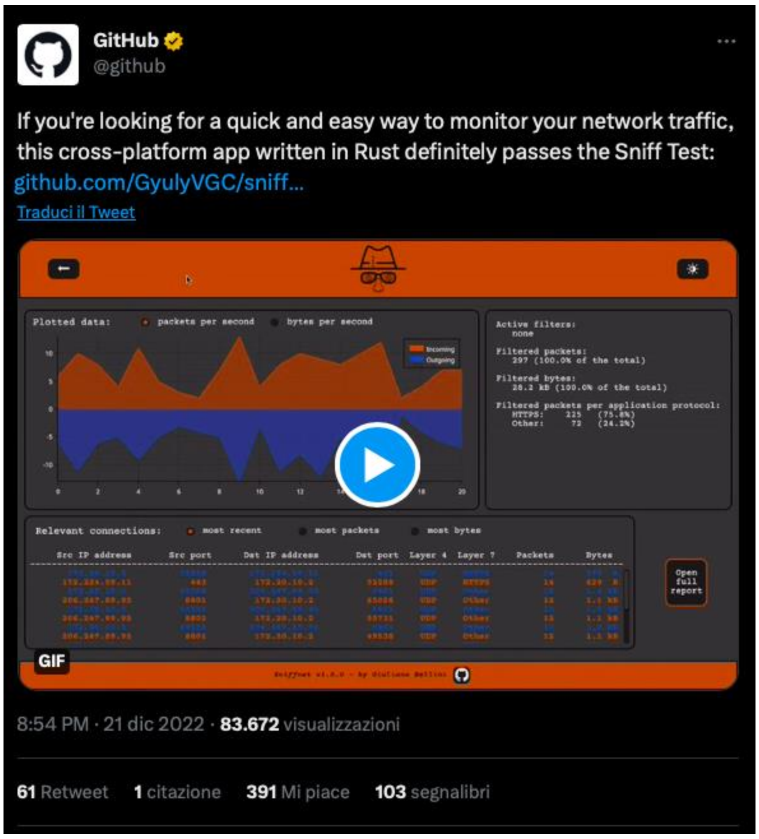
The application form included general questions about the repository under evaluation and required all the applicants to submit a one-minute video presenting themselves and their projects.

More than a thousand total applications were submitted and thoroughly evaluated by a **selection committee** composed of some of the most influential open-source representatives:

- Daniel Stenberg, founder and lead developer of cURL, a popular command line tool for transferring data with URLs
- Duan O'Brien, director of open-source at Indeed.com
- Ezra Olubi, cofounder and CTO at Paystack, a tech company with the aim of solving payments problems for ambitious businesses in Africa
- Mike Perham, author and maintainer of Sidekiq, a background job framework for Ruby
- Dawn Foster, director of the open-source community strategy at VMware
- Erica Brescia, an investor at Redpoint Ventures and board member at the Linux Foundation
- Kailash Nadh, CTO at Zerodha, India's largest stockbroker
- Viral Shah, cofounder and CEO at JuliaHub, the platform to accelerate the development and deployment of programs written in the Julia programming language

I got to know about the Accelerator through a blog post by GitHub summarizing the most relevant news from GitHub Universe, and I decided to **apply with Sniffnet** on the 29th of December, just two days before the final deadline.
To be completely honest, I discovered about the Accelerator program some weeks before, but I was unsure to apply because I believed that Sniffnet had few chances of being selected, since it was still in its very early stages of development.

What made me finally apply is the fact that GitHub published on its Twitter account a post [71] about Sniffnet a few days before Christmas:


GitHub
@github

If you're looking for a quick and easy way to monitor your network traffic, this cross-platform app written in Rust definitely passes the Sniff Test: github.com/GyulyVGC/sniff...

Traduci il Tweet

```
Active filters:
   none

Filtered packets:
   297 (100.0% of the total)

Filtered bytes:
   28.2 kB (100.0% of the total)

Filtered packets per application protocol:
   HTTPS:     225   (75.8%)
   Other:      72   (24.2%)
```

8:54 PM · 21 dic 2022 · **83.672** visualizzazioni

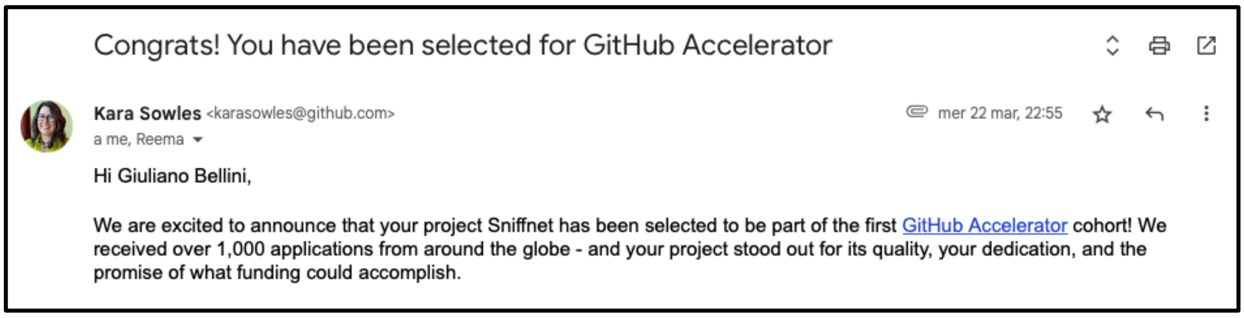
61 Retweet **1** citazione **391** Mi piace **103** segnalibri

Figure 6.1 - *Tweet about Sniffnet by GitHub itself (2.5 million followers on Twitter). [71]*

6.2. Announcement of the selected applicants

The Accelerator website reported that a public announcement about the results of the application process would have taken place on February 15th, 2023.
Due to some internal **delays**, they ended up shifting the date more than once, and I started forgetting about my application.
I didn't even have so high expectations about it, since I know that there are a bunch of valuable open-source projects that probably deserved to be elected far more than Sniffnet.

On March 22nd, right before going to bed for the night, I received an unexpected **email from GitHub** and the message object was pretty clear: *"Congrats! You have been selected for the GitHub Accelerator".*



Congrats! You have been selected for GitHub Accelerator

Kara Sowles <karasowles@github.com>
a me, Reema
mer 22 mar, 22:55

Hi Giuliano Bellini,

We are excited to announce that your project Sniffnet has been selected to be part of the first GitHub Accelerator cohort! We received over 1,000 applications from around the globe - and your project stood out for its quality, your dedication, and the promise of what funding could accomplish.

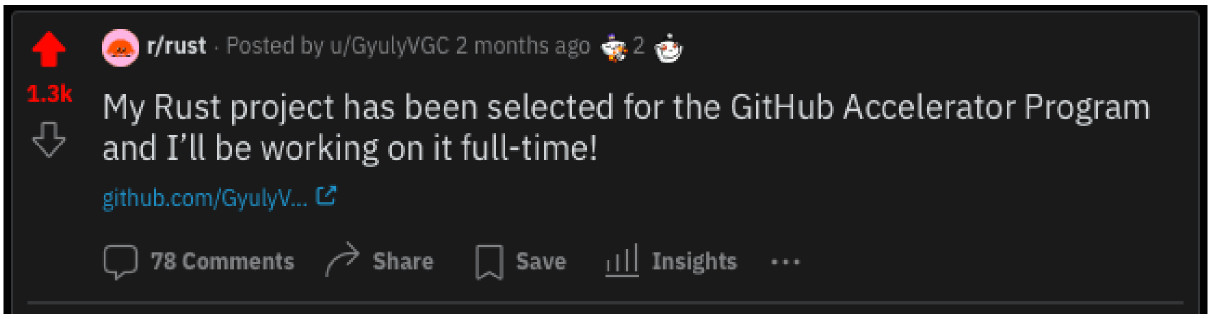
Figure 6.2 *- Email from GitHub announcing Sniffnet's election for the first GitHub Accelerator cohort.*

At first, I thought it was a joke.
After realizing it was real, I was both amazed and incredulous at the same time.

They recommended us to hold on sharing the news until the **formal announcement** on their blog, which took place on April 12th:

"Today, we're thrilled to announce GitHub Accelerator's first cohort! The 2023 cohort has 20 projects, with 32 participants from all over the world, including Argentina, Australia, Colombia, Denmark, France, Germany, India, Italy, Luxembourg, Pakistan, South Africa, Spain, Sweden, Turkey, the UK, and the US." [81]

I didn't delay in sharing my excitement with the Rust community on Reddit, the place where I first publicly posted about Sniffnet some months before.



r/rust · Posted by u/GyulyVGC 2 months ago 2

1.3k

My Rust project has been selected for the GitHub Accelerator Program and I'll be working on it full-time!

github.com/GyulyV...

Figure 6.3 *- My public announcement about Sniffnet's selection for the Accelerator program, shared with the Rust community of Reddit. [82]*

The post [82] received a huge amount of support and appreciation and was linked to a **discussion** [83] **on Sniffnet's GitHub repository** in which I expressed my enthusiasm for having such a unique opportunity:

"Sniffnet has grown a lot during the past months, and it's been a pleasure for me to spend most of my spare time on its development.
I'm not gonna lie: passing hours and hours on this project has become my favorite hobby lately.
I can't deny that sometimes it's been hard to balance Sniffnet's development with the daily University routine, made of never-ending lectures, group projects, deliveries, and home study.
I often ended up coding late (I mean very *late) at night or skipping meals/lectures, since 24 hours a day never seemed enough.*
Seeing Sniffnet getting traction and evolving, improving day after day, motivated me to keep pushing despite the obstacles along the way.
Long story short: having more time to dedicate to open-source was one of my primary wishes (and needs).
Today, the 12th of April 2023, it's such a joy for me to announce that Sniffnet has been selected for the GitHub Accelerator Program.
[…]
This means that during the next few months I'll be able to work on Sniffnet full-time, with the aspiration to bring it to the next level.
I've lots of ideas for new features and improvements and I'm looking forward to implementing them all.
Turning such a huge passion into a full-time job is the best thing I could've ever asked for.
I truly wish that this journey will continue even after the 10 weeks of the program, with the hope of working on open-source for my entire career.
I believe that more and more programmers deserve to have opportunities like this one, and I hope the Accelerator will pave the way to a brighter perspective for the open-source community.
I can't wait to see what the future has in store, and I feel blessed to have such an amazing occasion.
I'll do my best to get the most out of this experience."

6.3. The GitHub Accelerator Program

During the 10 weeks of the program, we had the chance of getting mentored by some of the most relevant open-source exponents; this section aims at summarizing the main outcomes of their talks and what it means for them to work on what they love doing.

The first lectures of the program were given by **Abby Cabunoc Mayes**, one of the organizers of the Accelerator and founder of Mozilla Open Leaders, an initiative to mentor open-source teams on how to lead their projects.

Figure 6.4 *- Abby Cabunoc Mayes*

6.3.1. Open practices (introduction to the program by Abby Cabunoc Mayes)

Open practices can be defined as the methods through which an organization programmatically collaborates with external groups to share knowledge, work, and influence, with the aim of obtaining a specific business goal.
Open-source software largely depends on this kind of practices and can obtain concrete benefits from their use. [84]

In the following, it's reported a list of the most common open practices and their related advantages.

- **Gifting**: consists of no-strings-attached giving of products, which in simpler words means to give away a service for free without carrying special conditions or restrictions for the users.
  This practice usually comes along with more permissive licenses and has the advantage of incentivizing adoption and driving a standard.
  Gifting is usually adopted by software companies where development and distribution costs are low; such companies will in the end be able to make a profit from the consequent installed base.
- **Soliciting ideas**: to develop a product tailored to the people's needs, this open practice uses its own community to generate ideas and solutions.
  In this case, end users are directly involved in the development process playing an active role.
- **Learning through use**: by carefully examining usage patterns, companies can provide added value improving their products.
  In the era of Big Data and constant connectivity, goods built on such a practice are more common and valuable than ever.
- **Creating together**: consists in sharing with the community the tasks needed to achieve a set of pre-established goals.

Inviting others to contribute permits to have access to more potential talent and can lower operating costs in terms of time and effort put in by single individuals.

- **Networking common interests**: different teams can coordinate to ensure that their activities achieve more towards a shared mission, while working each one on their own project.
  This practice enables separate groups to help each other, creating a more solid and scalable ecosystem and allowing their products improvement by learning from partners.

This list of open practices is not exhaustive and open-source projects usually don't rely just on a single practice but are rather based on a mix of them.
The common ground of all these practices resides in the fact that they're able to create some **added value**: it can be in the form of an overall better product, increased market share, or lowered operating cost.

6.3.2. Licensing (introduction to the program by Abby Cabunoc Mayes)

For what concerns open-source code, the possibility to put in place the aforementioned practices is natural and immediate thanks to the definition of open-source software itself:
"Open-source software is software that can be freely used, modified, and shared (in both modified and unmodified form) by anyone."

However, when making a creative work, including code, that work is under **exclusive copyright by default**.
Consequently, if not stated otherwise, nobody would be able to use, edit and share the work without being at risk of lawsuits.
What makes it possible for others to use and redistribute open-source code are **licenses**, namely documents listing what it's permitted to do with the code they refer to. [85]
Open-source licenses are today standardized and easy to use: it's in fact sufficient to copy-paste an existing license text in the root folder of the project to be licensed.
There exist various kinds of open licenses, each one granting different permissions.

I decided to release Sniffnet under both **MIT** and **Apache 2.0** licenses, since they are recommended by the Rust API guidelines to have the highest level of compatibility within the Rust ecosystem.
The MIT license is a very short, easy to understand, and permissive license that allows anyone to do anything as long as they keep a copy of the license.
The Apache 2.0 license is also permissive, despite having a few more restrictions regarding trademark use, liability, and warranty.

6.3.3. Getting sponsors and fundraising (with Caleb Porzio)

We were given an insightful speech by **Caleb Porzio** on how to find sponsors and build trust with our users.

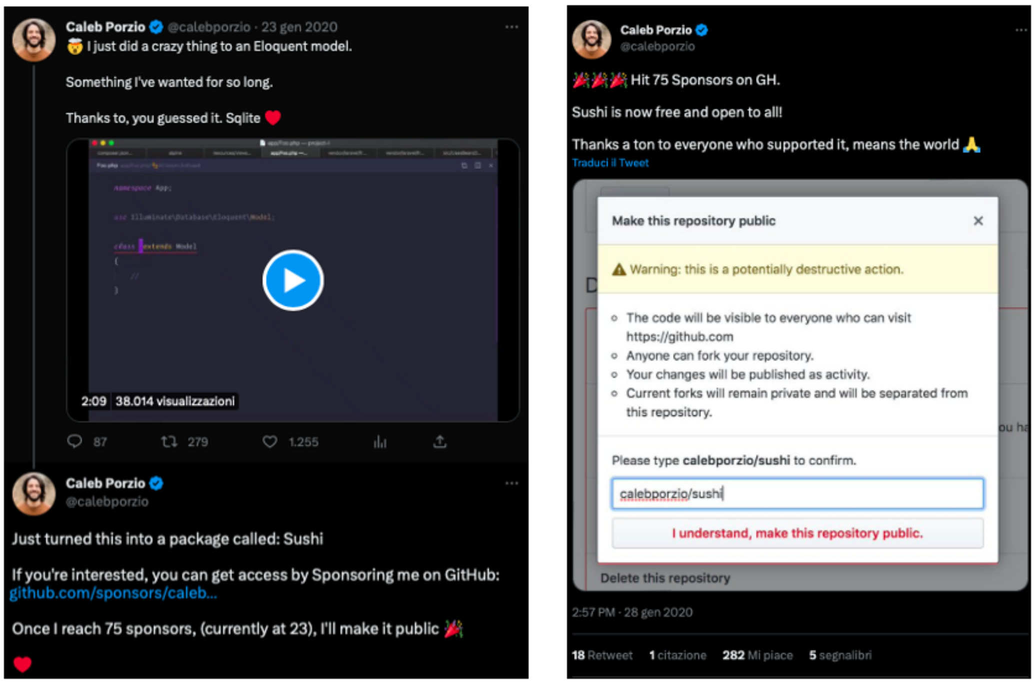
Caleb is a developer currently working on **open-source full-time**.
He has worked on many different projects, including **Livewire**, a full-stack framework for Laravel that makes building dynamic interfaces simple, and **AlpineJS**, a minimal framework for composing JavaScript behavior in your markup.

Figure 6.5 *- Caleb Porzio, creator of Livewire.*

Before dedicating to open-source, he was working as a developer at Tighten.
In 2018, he decided to **take a break** from his job to work on his personal projects.
Since open-source software doesn't pay the bills, in this initial period he also gave code mentorships to different clients to earn some money, and this choice costed him a 70% salary reduction in 2019 with respect to his previous full-time job.

GitHub Sponsors was initially a place where devoted and generous users, who wanted to support Caleb's work, could donate.
Regardless of how virtuous these people are, they are few compared to the number of overall users of the product.
Due to the nature of open-source, individuals already receive the full software for free, so this approach is severely constrained because it doesn't add any value to the users' experience.

He was initially earning about 500$ a month from his GitHub sponsors, but it wasn't enough to make a living out of it.
To raise his income, Caleb decided to launch a **sponsorware**: it consists of a piece of software exclusively distributed to personal sponsors until a predefined number of sponsors is hit, and after the threshold is reached the software is made open.



Caleb Porzio @calebporzio · 23 gen 2020
🤯 I just did a crazy thing to an Eloquent model.

Something I've wanted for so long.

Thanks to, you guessed it. Sqlite ❤️

2:09 38.014 visualizzazioni

87 279 1.255

Caleb Porzio
@calebporzio

Just turned this into a package called: Sushi

If you're interested, you can get access by Sponsoring me on GitHub:
github.com/sponsors/caleb...

Once I reach 75 sponsors, (currently at 23), I'll make it public 🎉

❤️

Caleb Porzio
@calebporzio

🎉🎉🎉 Hit 75 Sponsors on GH.

Sushi is now free and open to all!

Thanks a ton to everyone who supported it, means the world 🙏
Traduci il Tweet

Make this repository public ×

⚠ Warning: this is a potentially destructive action.

- The code will be visible to everyone who can visit https://github.com
- Anyone can fork your repository.
- Your changes will be published as activity.
- Current forks will remain private and will be separated from this repository.

Please type **calebporzio/sushi** to confirm.

calebporzio/sushi

I understand, make this repository public.

Delete this repository

2:57 PM · 28 gen 2020

18 Retweet 1 citazione 282 Mi piace 5 segnalibri

Figure 6.6 *- Tweets by Caleb Porzio announcing the sponsorware and its open publication.*

This strategy worked extremely well for Caleb, allowing him to increase his yearly revenue of 11k$ in a few days.
However, sponsorware requires a continuous stream of innovative ideas and the constant development of new projects, which is something not scalable and affordable in the long run.

To build a more durable stream of funding, Caleb opted for **sponsored screencasts**, consisting of video tutorials about how to use his frameworks, made available to his sponsors only.
This ended up being the path that definitively changed the game for Caleb: his yearly revenue went up from 40k$ to 100k$ in about three months, overcoming his previous wage as a full-time developer at Tighten. [86]
Finally, to differentiate sources and build more robust incomes, he started selling Livewire sticker packs and published an eBook course about how to personalize Visual Studio Code to make it more aesthetically pleasing.

Hearing Caleb's journey has been really inspiring and is a good example of how full-time open-sourcing is viable under certain circumstances.
Despite such an opportunity isn't for everyone, Caleb showed us that passion, versatility, and hard work can make a huge difference.
One thing to especially keep in mind from his experience is that **being able to adapt** is crucial to have success in open-source: a key aspect in Caleb's path is that he's been capable of covering several different roles, from programmer, to screencaster, video editor, writer, public speaker, and financial planner.

6.3.4. Sustainable Open Source (with Evan You)

Evan You is an independent open-source software developer based in Singapore who shared with us his experience with supporting his open-source development work full-time.
Evan is the creator of **Vue.js**, a progressive framework for building web interfaces, and **Vite**, a front-end build tool for JavaScript.

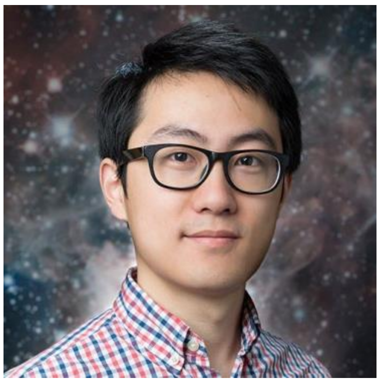


Figure 6.7 *- Evan You, creator of Vue.js.*

Vue is the project Evan is currently most focused on and it started as a side-project when he was working at Google in 2013.
He kept dedicating himself to Vue as a side-project even after leaving Google for Meteor, and it finally became his main occupation in 2016, when he decided to work on it full-time sustained by his sponsors through Patreon.

As the project grew, Vue got several corporate partnerships that allowed Evan to **form a loosely structured team** around the project: the involved people have no hard responsibilities, varying levels of involvement, and the contributions don't come just in the form of code, but also in the form of communication and moderation on the channels used by Vue community.
During the team-formation phase, it's important to give capable people autonomy and ownership: "Trust *them and give them the freedom to do what they are good at; amazing things will happen as a consequence*".

Another key point that Evan highlighted for us to build a sustainable project is to **grow a community** of active users around the customer segment which is more interested in the product, organizing offline meetups and conferences to form stronger bonds with the members.

Sustainable open-source software has **no universal formula** since different languages in different ecosystems involve people at different levels of the stack.
Evan suggested us that to better understand which model could fit each of us, we must first think what our real motivations behind open-source development are.
To this purpose, he described what he thinks are the **two main kinds of people** working on open-source:

- **Lifestyle entrepreneurs**, a category including people whose priority is working on something they really love doing and that give great value to the freedom of determining their own pace and balance. In such a case, one should also keep in consideration that there could be significant pay cuts with respect to an ordinary job.

- **Business minded**, category describing people that want to grow a business and believe that open-source is a strategic choice that will give their product an edge to succeed (for example by benefiting from the wider reach and lower adoption barriers, or by using the community to crowdsource feedback to help iterate on the project faster).

For the first class of people Evan sees **sponsorships** as a possible funding model, but he warned us that it's feasible in the long term only if the project has a horizontal reach (i.e., it's widely used in a broad category of scenarios) or if it's something the users closely interact with.
Sponsorship sources can either be individuals, even if they typically have very low conversion rates, or enterprises whose business success largely depends on the open-source project or that are simply looking for advertisement and exposure.

For the business-minded category, he suggests instead to consider **selling a product**, for example in the form of a freemium (i.e., a free software with paid features) or using open-source as a gateway to a paid product.
Conversely to the previous funding model, selling a product is more likely to be successful if the project solves a vertical / niche problem.

6.3.5. Finding contributors to hire (with Brian Douglas)

Since, as mentioned before, two of the main open advantages are soliciting ideas and creating together, it's fundamental for an open-source project to **find long-term contributors** outside of the core team of the project itself.

Having more contributors working on your project is obviously a valuable asset for a number of reasons, but getting hands dirty in an open-source project can have lots of benefits for the contributors as well.

Diving in the open-source scenario can in fact help new contributors to **learn in a practical way** from more advanced developers.
More experienced programmers can also largely benefit from contributing, as it not only allows them to grow as a developer but can also improve **communication effectiveness** in exposing technical issues, possible solutions, or simply giving feedback.
In addition to learning from others and building self-confidence, contributors love helping because they feel they are doing something important for the community: after all, the open-source community heavily depends on its volunteers and contributing is a way to make sure that this cycle continues indefinitely.

GitHub itself helps project maintainers to signal their interest in embarking new contributors by providing dedicated issue labels such as "*good first issue*" or "*help wanted*", but often this is not enough to create a high level of interaction.

Brian Douglas, past leader of Developer Advocacy at GitHub, gave us a talk about how to find possible contributors to hire.
Brian is the founder and CEO of **Open Sauced** [87], a place to help people get involved in the open-source community beyond what GitHub provides.

Figure 6.8 *- Brian Douglas, creator of Open Sauced*

Open Sauced is a platform for people to **discover active projects**, getting recommendations based on their interests and followed topics or developers.
It lets contributors create customizable widgets reporting sharable insights about their work, with the goal of linking them with **companies** in the open-source industry, turning meaningful connections into opportunities.

Open Sauced also aims at helping organizations find the most suitable contributor for their needs based on developer-first metrics, claiming that the typical green-squares contribution graph shown by GitHub is only a surface-level indicator of how capable a developer is.

Brian suggested us that to find intermediate and high-level programmers for contribution, it's important to **showcase the most difficult technical challenges** of our projects, to attract folks who would like to solve harder problems.
He stated that open-source projects that are successful and have lots of contributors do marketing well: it's therefore a good habit to focus on **documentation** beyond code, including proper changelog, readme, and release notes.

Brian then shared with us how he experimented with different **models of funding** contributors for their work, saying he initially started a "bank" of issues, each with his monetary reward for whoever solved it.
However, he later realized that this approach wasn't very much scalable and what's more valuable are people who contribute consistently over the long term and are paid on retainer.

6.3.6. Working with enterprises (with Dawn Foster and Duan O'Brien)

Dawn Foster is the Director of open-source Community Strategy within VMware's Open Source Programs Office, while **Duan O'Brien** is the Director of open-source for Indeed, where he built out the FOSS Contributor fund framework to invest in Indeed's open-source infrastructure.

Figure 6.9 *- Dawn Foster*

Figure 6.10 *- Duan O'Brien*

Both Dawn and Duan served on the GitHub Accelerator advisory committee and have decades of wisdom around **corporate relationships with open-source projects**.

Dawn Foster started the talk introducing **Open Source Program Offices** (OSPOs).
The OSPO consists, within a company, of the department that makes the decisions about how the company behaves with respect to open-source: the team behind this department takes care of funding choices, which projects to contribute to, and makes sure that the overall relationship between the business and the ecosystem is healthy and sustainable. [88]
When a company uses open source software projects, they must be aware of licenses, compliance requirements, confirm that the project has no vulnerabilities, and, in some circumstances, find qualified community members for future employment processes.
Conversely, in case the company releases and maintains open source projects, OSPOs are accountable for assuring community growth and involvement, verifying there are no intellectual property conflicts, ensuring the company retains its footprint and leadership, and possibly attracting fresh talent to the organization.

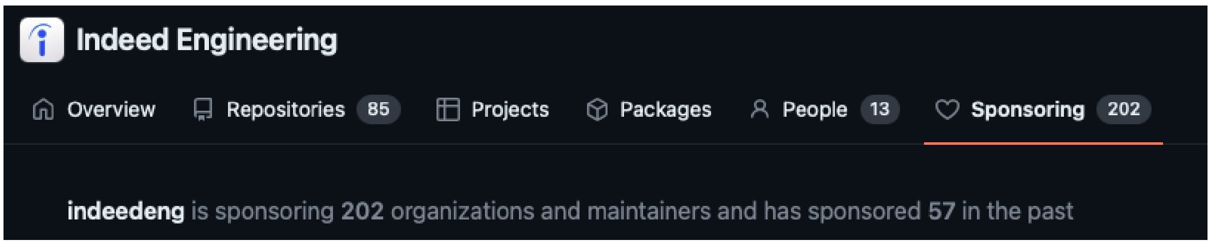
Nowadays, OSPOs are quite common among big tech companies, and we have been warned that in case we'd like to work with a corporate it's important to understand how their open-source department is structured and who works in it.

Dawn also shared with us the delicate topic of **barriers to corporate use**, namely the circumstances that might prevent companies from working with open-source.
As a director at VMware, Dawn is often in the position of assessing the viability of open-source projects, asking herself whether incorporating an emerging technology into a bigger and well-established product is a smart move or not.

With these concerns in mind, she listed what are in general the biggest barriers between a corporate and the open-source ecosystem:

- **Security**: determining if a project could have exploitable vulnerabilities is a key aspect from a company standpoint; for this purpose, it's useful for a repository to include a security.md file stating a way of privately reporting eventual vulnerabilities to the project maintainer.
- **Adoption**: software with more companies depending on it is more likely to be considered positively, since it's a signal that the project will be maintained and developed in the long term.
- **Governance**: projects governed by a solo maintainer are at risk of death since they rely uniquely on one person.
  What happens if the owner is not able to maintain the project anymore?
  For instance, Denis Pushkarev, creator of the popular core-js library, went to prison and wasn't allowed to fly updates to the project different millions of people depend on. [89]
  Having a larger pool of contributors, a code of conduct, and establishing an organization automatically makes the project more robust and long-lived, assuring it can keep up with all the issues and PRs.

After her speech, Dawn left the floor to Duan O'Brien, who most recently built the OSPO at **Indeed**, which is now one of the biggest public sponsors on GitHub (currently supporting, on a monthly basis, the work of more than 200 individual OSS developers).


Indeed Engineering

indeedeng is sponsoring 202 organizations and maintainers and has sponsored 57 in the past

Figure 6.11 *- Indeed GitHub organization, monthly sponsor of tens of open-source developers.*

Duan introduced himself with a very meaningful quote: *"Your fans love you. The people using your software love you. The problem is that sponsors are not your fans".*
End users of the software have so many projects they could support that it's unlikely for an open-source maintainer to make a living just out of donations by individuals.

Corporate partnerships are a more sustainable way of getting funding for FOSS, and usually happen because of one of the following **kinds of engagement**:

- **Individual**: someone at the company directly advocates for the sponsorship. This typically happens when the maintainer asks his community for support and a fan working at the company escalates the request to his manager, who uses an existing funding program to make the sponsorship happen.

- **Support contract**: someone at the company engages directly with the OSS maintainer for extended support or services.
  In this case, the company needs a professional service or a priority bug fix, and a representative reaches out to the project to negotiate for paid support.
- **Programmatic**: the company uses an automated tool or process to identify and select its sponsorships.
  This is the most modern way to find projects to sponsor and is based on various parameters, one of the most relevant being dependency analysis.

Different lessons for OSS maintainers came out from Duan’s talk: open-source developers should **ask often** for what they need (more than they feel comfortable to), have a **support contract ready** to be prepared for certain kinds of agreement, they should **assume automation** and use machine-readable funding info.

One final yet important thought from Duan was about simply **expressing gratitude**: Indeed sponsored tens of projects over the years, but the number of people saying *“thanks”* was really low - personally reaching out to your own sponsors to thank them is also a great opportunity to stick out from everyone else.

6.3.7. Project governance (with Shauna Gordon-McKeon)

Shauna Gordon-McKeon is an open-source member and maintainer since 2013.
She spent the past 6 years facilitating **projects governance** transitions, providing consultations, hosting workshops, and sharing resources and best practices at *governingopen.com.* [90]

Figure 6.12 *- Shauna Gordon-McKeon*

Trying to provide an exhaustive description of what governance really looks like from the perspective of FOSS, Shauna listed some of the most common **misconceptions** around it:

- Misconception #1: *Governance is just about who is in charge.*
  Governance not only establish who does what but includes all kinds of decisions around **what to work on**, **what to change**, **who are the participants and how to involve them**.
  Governance can be in the form of roadmaps, codes of conduct, release management, and more.
- Misconception #2: *Governance is another word for bureaucracy.*
  Governance is more a way to **prevent bureaucracy stuckness**, making the whole process adaptable and designing conflict resolution mechanisms.
- Misconception #3: *My project doesn't have governance.*
  Even if a project doesn't have formally defined its governance, there is always someone in charge of making decisions and planning; however, **formalizing governance** is highly suggested to let everyone know how the community works and to avoid a possible "tyranny of structurelessness". [91]
- Misconception #4: *There is a right way to do governance.*
  Governance is highly **context-dependent**: sometimes a democratic system is preferable, but in other circumstances a benevolent dictator may work better. What's important is to keep into account the kind of user base, the size of the project, its origin story, and other factors before defining rules, which are not fixed but can change over time.

Shauna summarised the role of governance with an impactful metaphor: "*Governance processes are **like a test suite** to catch governance bugs*".
In the same way as tests, governance evolves with a project over time and requires an effort in the present to save time and resources in later stages.

7. Conclusions

Spending the **last year** almost full-time on the development and management of Sniffnet **taught me a lot**.

I learned a new programming language, started appreciating a new framework to build UIs, understood how coding is just the tip of the iceberg in the context of a software project, and met a wonderful community of people always ready to get involved without demanding anything in return.

But most importantly… I did all of this with **passion** and found out that I enjoy programming and solving problems even more than I thought.

7.1. Next steps

Despite all the features introduced over the past few months, Sniffnet is still very far from being a complete network analyser.

The main idea for the upcoming development is to keep the application comfortably usable by everyone without sacrificing **additional, more advanced functionalities**.

The list of purposes I have in mind for the future is long and includes:

- **Identification of the PIDs generating a specific data exchange.**
  This is by far the most requested feature and one of the most difficult to implement, because it's heavily platform dependent.
  The inclusion of this functionality would allow to identify the processes responsible for each of the network connections.
- **Read and write PCAP files.**
  PCAP is a widespread file format containing packet-level evidence; supporting this format would mean permitting interoperability with other network analyzers, as well as the possibility of offline inspections.
- **Inspection of certain packets' payload.**
  Particularly useful in the case of non-encrypted services, to read the content of single packets.
- **Alerts based on the IP addresses' reputations.**
  Several services provide a way of determining if a specific IP address has been flagged as abusive or spammer based on a set of blacklists.
  In this case, it'd be useful to warn the users of the app, allowing them to possibly block the corresponding connections.
- **Details about unassigned IP addresses (bogon tag).**
  Bogons are IP addresses that are not in any range assigned by an official entity for public Internet use.
  As of today, in case a connection with a bogus address is established, Sniffnet tags it as coming from an unknown location, even if it's possible to determine more about it (for example if the address is reserved for future use or belongs to the private space).
- **Support for ICMP.**
  ICMP is a supporting protocol particularly used by routers to communicate errors or diagnostic information to other network nodes.
- **New filtering capabilities.**
  To permit a more fine-grained traffic sieve, new filter mechanisms must be added, comprehending regular expressions involving addresses and ports.
- **More complete details about each notification event.**
  The current notification page of the app cannot be interacted with by the user and is limited to displaying general information about network events; more useful details should be collected and shown.

- **Performance benchmarking.**
  Needed to assess how much data per second can be handled by the app without dropping packets.
- **Improvements to the overall structure of the app packages.**
  Including the fix of some minor problems and the signature of the packages with an SSL certificate.
- **Additional configurations and settings.**
  As the tool supports more functionalities, additional configurations are needed:
  - Option to define the path to a local JSON file with custom theme colors.
  - Definition of the MMDB files path, to allow using the commercial distributions of such databases.
  - Settings to personalise font size and scale factor of the UI.

Most of such functionalities have been long **requested directly by the users** of the tool and this will motivate me even more to find the time and energy to implement them.

Additionally, new capabilities related to the GUI will be added every time **Iced** gets updated.
The maintainer of the library published a clear **roadmap** [92] defining several features that will benefit Sniffnet and other interfaces realized with Iced.

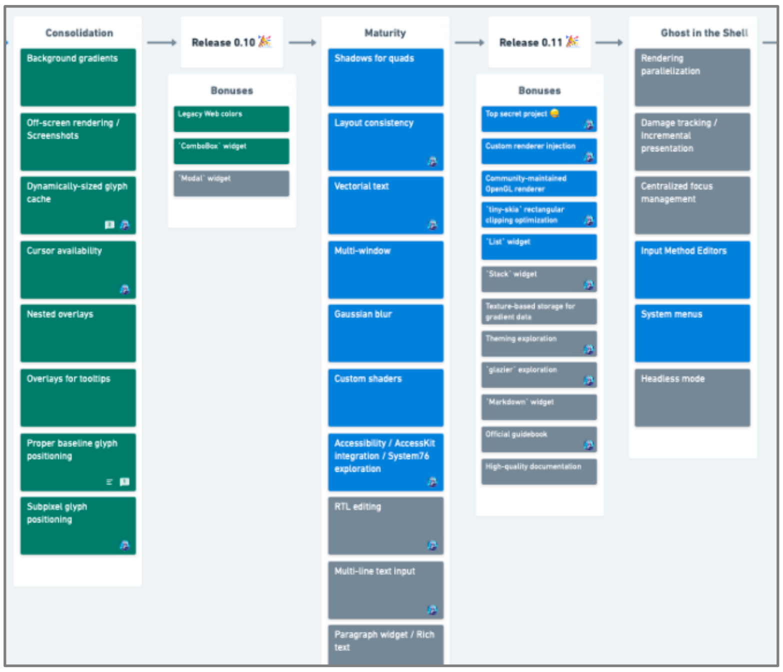


Figure 7.1 *- Iced roadmap of the upcoming releases [92]*

It's also very likely that **more new ideas will come along the way** because, even if I thought to be done 9 months ago, as time passes the to-do list is getting longer and longer.

Moreover, while writing the thesis I've realized how many things I have to say about the processes behind Sniffnet maintenance and I'm thinking about setting up the ***wiki*** section on the GitHub repository of the project, to share more in-depth analysis about every aspect related to the application.

7.2. Wrap up

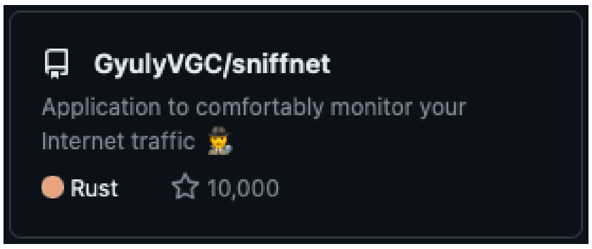
I'd like to conclude the dissertation with the content of a **discussion** I opened on the GitHub repository in the occasion of the **first anniversary of Sniffnet**.

---

*I'm excited to share with y'all that Sniffnet is **one year old** today!*
The last 365 days of my life have been almost totally dedicated to this project and I've learned a lot along the way.

On August 1st 2022, I'd have never imagined this project would have become what's today.
*Sniffnet was originally started in the scope of an **academic course** and went **much farther** than that: I fell in love with the project and countless additional features and improvements came to my mind.*

It's an unspeakable good feeling to have made software people use and appreciate, giving something back to this awesome community.
*Sometimes it still feels unreal that Sniffnet is now one of the most popular network analysers on GitHub, having passed **10k stars** just a few days ago and being in the top 100 most starred Rust repositories ever made.*



GyulyVGC/sniffnet
Application to comfortably monitor your Internet traffic 🕵️
● Rust ☆ 10,000

Figure 7.2 *- One of Sniffnet's most recent achievements: 10 thousand stars on GitHub*

***What have I learned** during this journey?*

*I cannot say to have drastically improved my coding skills, but what I can assert with certainty is to have understood how **developing software isn't just programming**: it requires providing an adequate documentation, user support, packaging, distribution, and opens the possibility for **exchange of ideas** with other folks, that is one of the most interesting and stimulating parts.*

*I had the one-in-a-life opportunity to be part of the first **GitHub Accelerator** cohort, which not only gave me important insights about open-source but also allowed me to work full-time on Sniffnet during the past months.*

*I’d also like to mention the little yet **heart-warming individual donations** I’ve received on PayPal, Patreon, and GitHub Sponsors: generous people do really exist, often they’re hidden just out there.*

However, after the sponsorship by GitHub, the financial support has not been enough to permit me to keep working on this project full-time.
It’s unfortunate not because of the stream of money itself, but because I truly love developing Sniffnet, and I’d like to be able to spend more and more time on it.
*Despite this, I can guarantee that **Sniffnet will be constantly updated and maintained** in the future, even if not at the pace of the past months.*

*I was suggested by some friends to make Sniffnet a **freeware**, introducing features reserved for paid users, and I was really tempted to do it with the functionalities introduced in v1.2.*
On the other hand, the open-source ecosystem gave me a lot and it didn’t feel right to turn away.
*Eventually, **I decided to listen none other than my heart**: Sniffnet is and will remain forever and ever **completely open-source** software.*

***Passion** is the engine that pushes us forward, that fills us with motivation, satisfaction, enthusiasm, and hope.*
Listen to your heart, do what you enjoy, and enjoy what you do: this is an outstanding method to spend life to the fullest.

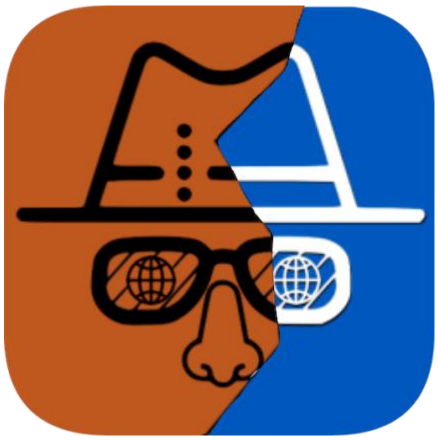


Figure 7.3 *- Sniffnet's official logo: it depicts an investigator focused on examining Internet traffic. His four-dotted hat is a reference to the notation used to represent IPv4 addresses, and the prominent nose allows him to better sniff network packets.*

Figure 7.4 *- GitHub merchandise sent as a welcome kit in the occasion of the kick-off of the GH Accelerator Program (April 2023).*

Figure 7.5 *- Sniffnet stickers, thought and printed with love by Martina.*

Appendix A: The project CHANGELOG

This appendix reports Sniffnet's Changelog, which is the file listing **all the software's releases and changes** over time.

```
Changelog

All Sniffnet releases with the relative changes are documented in this
file.


[1.2.2] - 2023-08-08

- Added option to set different shades of color gradients for each of the
available themes
- Added new application themes: Dracula, Gruvbox, Nord, and Solarized
(#330)
- Other aesthetic improvements (see #119 for more info):
    - redesigned page tabs
    - highlighted headings with different colors
    - simplified scrollables style
    - improvements to Deep Sea and Mon Amour color palettes
- Added Finnish translation FI (#310)
- Added support for `--help` and `--version` command line arguments (#272)
- Migrated to Iced 0.10, that is now able to select the graphical renderer
at runtime: a fallback one (`tiny-skia`) will be used in case the default
one (`wgpu`) crashes (#324)
- Added app `id` in order to correctly show the icon and app name on Linux
Wayland (fixes #292)
- Restructured issue templates to let users open issues in a more efficient
and effective way (#285)
- Updated French translation to v1.2 (#279)
- Color palettes in settings page are now built as `Rule` widgets, without
involving the use of external SVGs anymore
- Fixed `alt`+`tab` shortcut issue (#298 — fixes #262)
- Fixed problem that didn't allow opening links and the report file on
operating systems different from Windows, macOS, and Linux
- Use scrollable to make active filters visible when the selected adapter
name is long (overview page)
- Ensure no colored pixel is shown if the respective packets or bytes
number is zero
- Minor fix to Chinese translation (#271)
- Where is Sniffnet heading next? See the new roadmap of the project.


[1.2.1] - 2023-06-08

- Considerably refined the app packaging strategy (see #246 for more
details), fixing various related issues (#199, #220, #223, #224, #225,
#242)
- Added button to clear all the current search filters quickly in inspect
page
- Added Swedish translation SE (#213)
- Updated most of the existing translations to v1.2:
  - German - #191
  - Spanish - #203
  - Persian - #193
  - Korean - #205
  - Polish - #244
  - Romanian - #241
  - Russian - #187
  - Turkish - #192
  - Ukrainian - #216
  - Chinese - #214
- Renamed "Administrative entity" to "Autonomous System name" to avoid
confusion
- Improved filter columns relative width to avoid the "Application
protocol" label being cut when displayed in Swedish
- Footer URLs have been updated to include links to Sniffnet's official
website and GitHub Sponsor page
- Updated docs including installation instruction for Aarch Linux (#185)
- Minor improvements to packets and bytes number format
- Minor improvements to:
  - code readability (#248)
  - docs (#235)
- Solved a minor problem that caused flags to be slightly misaligned in
inspect page table


[1.2.0] - 2023-05-18

- Introduced host-based analysis: instead of just showing IP addresses, now
host names and network providers are available for a quicker and more
meaningful traffic interpretation
  * Added rDNS (reverse DNS) lookups to find out network host names
  * Added ASN (Autonomous System name and number) lookups to find out the
entity managing a given IP address (fixes #62)
- Individual connections identified by IP addresses remain available and
can now be filtered and further inspected through a simple click
- Support for identification of addresses in the local network
- Support for data link layer MAC addresses
- Full support for broadcast traffic recognition (added directed broadcast
identification)
- Added dropped packets number (fixes #135)
- Changed favorites management: instead of referring to single IP
addresses, favorites are now related to network hosts
- Added Greek translation GR (#160)
- Added Persian translation IR (#158)
- Do not open terminal window when starting the application on Windows
(fixes #85)
- Do not open terminal window when starting the application on macOS
- Changed macOS application icon to be consistent with standard icons
dimension (fixes #177)
- Made available RPM package for Linux and automated packaging process for
Windows, macOS, and Linux (#180 - fixes #20)
- Keep the active addresses of the selected network adapter up to date
during analysis
- Changed shortcut to interrupt analysis from `backspace` to
`ctrl+backspace`
- Images have been replaced with SVGs
- Added unit tests for `chart` and started unit tests for `gui` modules
(#132)
- Fixed problem that let users switch page pressing the tab key even if no
packets were received


[1.1.4] - 2023-04-18
```

```
- Added new translations of the GUI:
  * Portuguese PT (#134)
  * Russian RU (#151)
  * Korean KR (#128)
  * Turkish TR (#139)
  * ...the total number of supported languages is now 13 🎉
- Changed adapter buttons format and improved volume slider layout (see
#119 for more details or to give me further suggestions)
- Scrollbars are now highlighted when hovering on the respective scrollable
area
- Set up `iced_glow` feature on branch `glow-renderer` to overcome
unsupported graphics (#155)
- Modified `dependabot` configuration to update GitHub Actions as needed
(#141)
- Fixed problem causing a crash on macOS when starting Sniffnet's Homebrew
package or building from source in release mode (#109 - #137)


[1.1.3] - 2023-04-04

- Added Romanian translation RO (#113)
- Added feature to warn you when a newer version of Sniffnet is available
on GitHub 🆕 (#118)
- Added badge on tab bar to show unread notifications count 🔈
- Introduction of `lazy` widgets to improve the application efficiency
(#122)
- Aesthetic improvements to create a more modern and minimal UI (#119)
- Changed keyboard shortcut to open settings from `ctrl+S` to `ctrl+,`, as
suggested in #97
- Fixed problem that was causing a switch to the initial page when back
button was pressed with settings opened on running page and with no packets
received
- Fixed problem that was causing application logo to be partially hidden
when resizing the window to a lower dimension
- Show `-` option in app protocol picklist only when a filter is active
- Refactored and cleaned code modules (#123)
- Fixed header alignment


[1.1.2] - 2023-03-18

- Added new translations of the GUI, bringing the total number of supported
languages to 8!
  * German DE (#87)
  * Simplified Chinese CN (#89 - #93)
  * Ukrainian UA (#94)
- Added keyboard shortcuts to make the whole experience more enjoyable and
efficient: check out issue #97 to see all the available hotkeys or to
suggest new ones!
- Changed GUI font to `sarasa-gothic-mono` to support the introduction of
Simplified Chinese language
- Minor improvements to Overview page proportions and paddings


[1.1.1] - 2023-02-25

- Added new translations of the GUI!
  * French FR (#64 - #67)
  * Spanish EA (#70)
  * Polish PL (#78)
- The last successfully sniffed network adapter is now remembered on
application closure, so that users don't have to manually select it again
when restarting Sniffnet (implementing a feature requested in #77)
- Implemented possibility to quit the application pressing crtl+Q keys, as
requested in #68
- The last opened settings page is now remembered within a given session
- Fixed bug that caused settings configuration not to be permanently saved
across different sessions when closing settings from the 'x' button in the
top right corner (fixes #77)
- Textual report is now saved in a fixed directory, instead of using the
directory where the execution was started. The output is now saved in the
same folder containing configuration files storing Sniffnet settings. The
directory is automatically chosen by confy depending on your architecture,
and can be seen hovering on the "Open full report" button. (fixes #51)
- When multiple favorite connections are featured per time interval, now
it's possible to receive more than one favorite notification referred to
the same timestamp
- Fixed problem that was causing the Application Protocol picklist
placeholder not being translated


[1.1.0] - 2023-02-07

- Added Custom Notifications to inform the user when defined network events
occur:
  * data intensity exceeded a defined packets per second rate
  * data intensity exceeded a defined bytes per second rate
  * new data are exchanged from one of the favorite connections
- Added Settings pages to configure the state of the application
(persistently stored in a configuration file):
  * customise notifications
  * choose between 4 different application styles
  * set the application language (this release introduces the Italian
language IT, and more languages will be supported soon)
- Added Geolocation of the remote IP addresses (consult the README for more
information)
- Implemented the possibility of marking a group of connections as
favorites and added favorites view to the report
- Added modal to ask the user for confirmation before leaving the current
analysis
- Added Tooltips to help the user better understand the function of some
buttons
- Partially implemented support for broadcast IP addresses (still missing
IPv4 directed broadcast)
- The application window is now maximized after start
- All the GUI text fonts have been replaced with 'Inconsolata'
- Fixed issue #48 adding a horizontal scrollable to the report view


[1.0.1] - 2022-11-30

- Substituted command `open` with command `xdg-open` on Linux systems to
solve the problem described in issues #13 and #23
- Introduced a constraint on minimum window height to avoid problem
described in issue #12
- Added some tests on `AppProtocol` and improved GitHub workflows


[1.0.0] - 2022-11-21
```

```
- The application is no longer just a command line interface: Sniffnet has
now a whole graphical user interface!
  * Charts and traffic statistics are now constantly updated and shown
interactively in the GUI
  * Users don't have to worry about command line options anymore: it is now
possible to comfortably specify adapters and filters through the GUI
  * Sniffnet is now more accessible, available in real-time, easy to use
and aesthetically pleasing thanks to its new interface
- In order to reach out as many people as possible, I created installers
for Windows, macOS and Linux, to make it easier to install Sniffnet for
those that still doesn't have Rust on their machines


[0.5.0] - 2022-10-02

- Optimized textual report updates: only changed entries are rewritten
(file `report.txt`)
- Textual report elements are now ordered by timestamp instead of number of
packets
- Report header with statistics is now written on a separate textual file
(file `statistics.txt`)
- Removed command line option `--verbose` because considered redundant
- Removed command line option `--minimum-packets` because not meaningful
anymore


[0.4.1] - 2022-09-27

- Changed the default textual report representation
- Added command line option `-v` to set the textual report representation
to the former one (verbose mode)
- Sniffnet now also considers the transport layer protocol to define
textual report elements (now defined by the network 5-tuple)


[0.4.0] - 2022-09-11

- Added feature to produce a graphical report with the number of packets
per second and the number of bits per seconds, incoming and outgoing
- Added multicast addresses recognition
- Reports are not updated if the application is paused


[0.3.2] - 2022-09-07

- Changed output report structure: each element now corresponds to a couple
of network [address:port]
- When application is resumed after pause, the buffer containing packets is
reinitialized


[0.3.1] - 2022-08-31

- Added devices' description when application is launched with the `-d`
option
- Introduced feature to measure write timings and added a BufWriter to
improve write performance
- Fixed standard output colors for Windows systems
```

```
[0.3.0] - 2022-08-29

- Added global statistics: number of [address:port] pairs and sniffed
packets
- Added statistics on the number of packets for each application layer
protocol
- Fixed application layer protocols filtering


[0.2.1] - 2022-08-26

- Removed img folder and uploaded pictures on cloud


[0.2.0] - 2022-08-24

- Added command line option `--app` to filter application layer protocols
- Added feature to recognize local vs remote addresses
- Added function to parse IPv6 addresses
- Fixed secondary threads panics
- Changed the way application layer protocols are retrieved
- Improved textual report format


[0.1.2] - 2022-08-18

- Added video tutorial about the application


[0.1.1] - 2022-08-17

- Fixed README errors


[0.1.0] - 2022-08-17

- Sniffnet first release
```

Appendix B: Articles and mentions

In the following, it's reported a collection of the **main articles and mentions** featuring Sniffnet, published by organizations, blogs, and online newspapers (ordered by date).

Published on / by	Date	Reference
Heise, a popular German ICT newspaper	22 November 2022	[69]
TrishTech, website about computer technology with a focus on Windows	22 November 2022	[93]
MajorGeeks, a platform hosting software tutorials and download links	23 November 2022	[94]
Softpedia, a website reporting reviews about software for different platforms	23 November 2022	[95]
GitHub Twitter account	21 December 2022	[71]
TechViewLeo, a website about programming news	29 December 2022	[96]
Korben, a French blog about technology	6 February 2023	[97]
Trend Oceans, a Linux web portal	25 March 2023	[98]
Console, a portal featuring interviews with open source maintainers	23 April 2023	[99]
Linux Magazine, a monthly newspaper featuring in-depth analysis about technology	May 2023	[100]
Hacker News, an ICT social news website	19 May 2023	[101]
ilSoftware, an Italian website about software of all kinds	19 May 2023	[102]
Caschys Blog, a German blog about technology	19 May 2023	[103]
GeekNews, a Korean technology news website	20 May 2023	[104]
CodeZine, a Japanese website focused on software releases	23 May 2023	[105]
It's FOSS, a portal about open source with a focus on Linux	29 May 2023	[106]
KeepItTechie, a YouTube channel diving into the world of Linux and open-source	13 June 2023	[107]
Linux China, a Chinese blog focused on Linux news	14 June 2023	[108]
OSTechNix, a website focused on tech and Unix	4 July 2023	[109]

Hacker News, an ICT social news website	14 July 2023	[110]
Zhang Xuan on his Twitter account, a famous Chinese freelance developer	16 July 2023	[111]
Behind The Mutex, a weekly newsletter with a focus on open-source	18 July 2023	[112]
Ruan Yifeng, a Chinese blogger	21 July 2023	[113]
MacBed, a website about macOS software	21 July 2023	[114]
365TIPŮ, a Czech ICT portal which publishes daily tips and weekly articles	23 July 2023	[115]
unknowNews, a Polish blog issuing IT news articles weekly	28 July 2023	[116]

The table is updated to the month of July 2023

Bibliography

[1] R. Mohanakrishnan, “What Is a Computer Network? Definition, Objectives, Components, Types, and Best Practices,” spiceworks, 2023. [Online]. Available: https://www.spiceworks.com/tech/networking/articles/what-is-a-computer-network/.

[2] M. E. Shacklett, “What is TCP/IP?,” TechTarget, 2021. [Online]. Available: https://www.techtarget.com/searchnetworking/definition/TCP-IP.

[3] “What is a packet? | Network packet definition,” CLOUDFLARE, [Online]. Available: https://www.cloudflare.com/en-gb/learning/network-layer/what-is-a-packet/.

[4] “What is packet sniffing?,” Paessler, [Online]. Available: https://www.paessler.com/it-explained/packet-sniffing.

[5] “What is Packet Sniffing and How Does It Work,” Sangfor Technologies, 2022. [Online]. Available: https://www.sangfor.com/glossary/cybersecurity/what-is-packet-sniffing-and-how-does-it-work.

[6] “Wireshark: The world's most popular network protocol analyzer,” Wireshark, [Online]. Available: https://www.wireshark.org.

[7] “Tcpdump: a powerful command-line packet analyzer,” Tcpdump, [Online]. Available: https://www.tcpdump.org.

[8] C. Thompson, “How Rust went from a side project to the world’s most-loved programming language,” MIT Technology Review, 2023. [Online]. Available: https://www.technologyreview.com/2023/02/14/1067869/rust-worlds-fastest-growing-programming-language/.

[9] “Stack Overflow 2022 Developer Survey,” Stack Overflow, 2022. [Online]. Available: https://survey.stackoverflow.co/2022/.

[10] T. Heartman, “Understanding the Rust borrow checker,” LogRocket, 2020. [Online]. Available: https://blog.logrocket.com/introducing-the-rust-borrow-checker/.

[11] “Are we GUI Yet? The state of building user interfaces in Rust.,” areweguiyet, [Online]. Available: https://areweguiyet.com.

[12] "tauri-apps/tauri," tauri-apps, [Online]. Available: https://github.com/tauri-apps/tauri.

[13] "Rapporto sulla trasparenza: Crittografia HTTPS sul Web," Google, [Online]. Available: https://transparencyreport.google.com/https/overview.

[14] "Domande frequenti sul protocollo HTTPS," Google, [Online]. Available: https://support.google.com/transparencyreport/answer/7381231#zippy=%2Cwhat-is-encryption.

[15] "clap," clap-rs/clap, [Online]. Available: https://github.com/clap-rs/clap.

[16] "Crate pcap," Docs.rs, [Online]. Available: https://docs.rs/pcap/latest/pcap/index.html.

[17] "Crate etherparse," Docs.rs, [Online]. Available: https://docs.rs/etherparse/latest/etherparse/index.html.

[18] "IANA," IANA, [Online]. Available: https://www.iana.org.

[19] G. Bellini, "Faster way to write textual report file," GyulyVGC/sniffnet, 2022. [Online]. Available: https://github.com/GyulyVGC/sniffnet/issues/3.

[20] "plotters," plotters-rs/plotters, [Online]. Available: https://github.com/plotters-rs/plotters.

[21] "Plotters Developer Guide," Plotters, [Online]. Available: https://plotters-rs.github.io/book/intro/introduction.html.

[22] "slint," slint-ui/slint, [Online]. Available: https://github.com/slint-ui/slint.

[23] "dioxus," dioxuslab/dioxus, [Online]. Available: https://github.com/dioxuslabs/dioxus.

[24] "egui," emilk/egui, [Online]. Available: https://github.com/emilk/egui.

[25] "iced," iced-rs/iced, [Online]. Available: https://github.com/iced-rs/iced.

[26] "Retained Mode Versus Immediate Mode," Microsoft, 2019. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/learnwin32/retained-mode-versus-immediate-mode.

[27] "Elm: A delightful language for reliable web applications.," Elm-lang, [Online]. Available: https://elm-lang.org.

[28] "Welcome to Pop!_OS," System76, [Online]. Available: https://pop.system76.com.

[29] "Is Iced replacing GTK apps for the new COSMIC desktop?," Reddit, 2022. [Online]. Available: https://www.reddit.com/r/pop_os/comments/xs87ed/comment/iqjc35b/?utm_source=reddit&utm_medium=web2x&context=3.

[30] "plotters-iced," Joylei/plotters-iced, [Online]. Available: https://github.com/Joylei/plotters-iced.

[31] "rodio," RustAudio/rodio, [Online]. Available: https://github.com/RustAudio/rodio.

[32] "Font subset," FontTools, [Online]. Available: https://fonttools.readthedocs.io/en/latest/subset/index.html.

[33] "confy," rust-cli/confy, [Online]. Available: https://github.com/rust-cli/confy.

[34] "MaxMind DB File Format Specification," MaxMind DB, [Online]. Available: https://maxmind.github.io/MaxMind-DB/.

[35] G. Bellini, "Suggest me keyboard shortcuts to add," GyulyVGC/sniffnet, 2023. [Online]. Available: https://github.com/GyulyVGC/sniffnet/issues/97.

[36] G. Bellini, "Suggest me aesthetic improvements," GyulyVGC/sniffnet, 2023. [Online]. Available: https://github.com/GyulyVGC/sniffnet/issues/119.

[37] "dns-lookup," keeperofdakeys/dns-lookup, [Online]. Available: https://github.com/keeperofdakeys/dns-lookup/.

[38] "GitHub Docs: About READMEs," GitHub, [Online]. Available: https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-readmes.

[39] "Concise, consistent, and legible badges," Shields.io, [Online]. Available: https://shields.io.

[40] "Software Versioning," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Software_versioning.

[41] "Setting guidelines for repository contributors," GitHub, [Online]. Available: https://docs.github.com/en/communities/setting-up-your-project-for-healthy-contributions/setting-guidelines-for-repository-contributors.

[42] "About wikis," GitHub, [Online]. Available: https://docs.github.com/en/communities/documenting-your-project-with-wikis/about-wikis.

[43] “About issues,” GitHub, [Online]. Available: https://docs.github.com/en/issues/tracking-your-work-with-issues/about-issues.

[44] G. Bellini, “Issues - missing dependency,” GyulyVGC/sniffnet, 2023. [Online]. Available: https://github.com/GyulyVGC/sniffnet/issues?q=is%3Aissue+label%3A%22missing+dependency%22.

[45] G. Bellini, “Help me translating Sniffnet in your native language,” GyulyVGC/sniffnet, 2023. [Online]. Available: https://github.com/GyulyVGC/sniffnet/issues/60.

[46] “About pull requests,” GitHub, [Online]. Available: https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests.

[47] G. Bellini, “Pull requests - translation,” GyulyVGC/sniffnet, 2023. [Online]. Available: https://github.com/GyulyVGC/sniffnet/pulls?q=is%3Apr+label%3Atranslation.

[48] 4r7if3x, “Fixed build and packaging issues,” GyulyVGC/sniffnet, 2023. [Online]. Available: https://github.com/GyulyVGC/sniffnet/pull/246.

[49] “About milestones,” GitHub, [Online]. Available: https://docs.github.com/en/issues/using-labels-and-milestones-to-track-work/about-milestones.

[50] “Dependabot,” GitHub, [Online]. Available: https://github.com/dependabot.

[51] “ImgBot,” ImgBot, [Online]. Available: https://imgbot.net.

[52] “Recognize All Contributors, including those that don't push code,” All Contributors, [Online]. Available: https://allcontributors.org.

[53] “Understanding GitHub Actions,” GitHub, [Online]. Available: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions.

[54] “rustfmt,” rust-lang/rustfmt, [Online]. Available: https://github.com/rust-lang/rustfmt.

[55] “rust-clippy,” rust-lang/rust-clippy, [Online]. Available: https://github.com/rust-lang/rust-clippy.

[56] “The Cargo Book,” Rust, [Online]. Available: https://doc.rust-lang.org/stable/cargo/.

[57] “The Rust community’s crate registry,” Rust, [Online]. Available: https://crates.io.

[58] “The Manifest Format,” Rust, [Online]. Available: https://doc.rust-lang.org/cargo/reference/manifest.html.

[59] “Crate cargo_wix,” volks73/cargo-wix, [Online]. Available: https://volks73.github.io/cargo-wix/cargo_wix/index.html.

[60] “WIX TOOLSET,” WiX Toolset, [Online]. Available: https://wixtoolset.org.

[61] “Npcap: Packet capture library for Windows,” Npcap, [Online]. Available: https://npcap.com.

[62] “Npcap OEM Edition—Redistribution License,” Npcap, [Online]. Available: https://npcap.com/oem/redist.html.

[63] “create-dmg,” create-dmg/create-dmg, [Online]. Available: https://github.com/create-dmg/create-dmg.

[64] “Linux distribution,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Linux_distribution.

[65] “cargo-deb,” kornelski/cargo-deb, [Online]. Available: https://github.com/kornelski/cargo-deb.

[66] “cargo-generate-rpm,” cat-in-136/cargo-generate-rpm, [Online]. Available: https://github.com/cat-in-136/cargo-generate-rpm.

[67] “Reddit,” [Online]. Available: https://www.reddit.com.

[68] G. Bellini, “Announcing Sniffnet v0.4.0 - A multithreaded, cross-platform network analyzer,” Reddit, 2022. [Online]. Available: https://www.reddit.com/r/rust/comments/xbn5o6/announcing_sniffnet_v040_a_multithreaded/.

[69] M. Förster, “Netzwerkmonitoring mit Sniffnet: Open Source und komplett in Rust geschrieben,” Heise, 2022. [Online]. Available: https://www.heise.de/news/Netzwerkmonitoring-mit-Sniffnet-Open-Source-und-komplett-in-Rust-geschrieben-7349019.html.

[70] “GitHub Trending: See what the GitHub community is most excited about today.,” GitHub, [Online]. Available: https://github.com/trending.

[71] GitHub, Twitter, 2022. [Online]. Available: https://twitter.com/github/status/1605652851245649931.

[72] M. T. V. Hudson Borges, “What’s in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform,” Department of Computer Science, UFMG, Brazil, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0164121218301961.

[73] “Star History: The No.1 GitHub star history graph on the web,” [Online]. Available: https://star-history.com/blog.

[74] “Hacker News front page,” [Online]. Available: https://news.ycombinator.com/news.

[75] “Homebrew: The Missing Package Manager for macOS (or Linux),” Homebrew, [Online]. Available: https://brew.sh.

[76] “GitHub Accelerator: Jumpstarting new careers in open source,” GitHub, 2022. [Online]. Available: https://accelerator.github.com.

[77] “GitHub Docs: About GitHub Pages,” GitHub, [Online]. Available: https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages.

[78] “aruba.it,” Aruba, [Online]. Available: https://www.aruba.it/.

[79] “Google Analytics,” Google, [Online]. Available: https://marketingplatform.google.com/intl/it/about/analytics/.

[80] T. Dohmke, “Everything new from GitHub Universe 2022,” GitHub, 2022. [Online]. Available: https://github.blog/2022-11-09-everything-new-from-github-universe-2022/.

[81] K. Sowles, “GitHub Accelerator: our first cohort and what’s next,” GitHub, 2023. [Online]. Available: https://github.blog/2023-04-12-github-accelerator-our-first-cohort-and-whats-next/.

[82] G. Bellini, “My Rust project has been selected for the GitHub Accelerator Program and I’ll be working on it full-time!,” Reddit, 2023. [Online]. Available: https://www.reddit.com/r/rust/comments/12jqvwy/my_rust_project_has_been_selected_for_the_github/?utm_source=share&utm_medium=web2x&context=3.

[83] G. Bellini, “Sniffnet has been selected for the GitHub Accelerator Program,” GyulyVGC/sniffnet, 2023. [Online]. Available: https://github.com/GyulyVGC/sniffnet/discussions/133.

[84] A. Klepel, “A Framework of Open Practices,” Medium, 2017. [Online]. Available: https://medium.com/mozilla-open-innovation/a-framework-of-open-practices-9a17fe1645a3.

[85] “The Legal Side of Open Source,” Open Source Guides, [Online]. Available: https://opensource.guide/legal/.

[86] C. Porzio, “I Just Hit $100k/yr On GitHub Sponsors! (How I Did It),” 2020. [Online]. Available: https://calebporzio.com/i-just-hit-dollar-100000yr-on-github-sponsors-heres-how-i-did-it.

[87] “OpenSauced: Your next open source journey starts here,” OpenSauced, [Online]. Available: https://opensauced.pizza.

[88] J. M. L. d. l. Fuente, “A guide to setting up your Open Source Program Office (OSPO) for success,” opensource.com, 2020. [Online]. Available: https://opensource.com/article/20/5/open-source-program-office.

[89] T. Claburn, “What happens when the maintainer of a JS library downloaded 26m times a week goes to prison for killing someone with a motorbike? Core-js just found out,” The Register, 2020. [Online]. Available: https://www.theregister.com/2020/03/26/corejs_maintainer_jailed_code_release/.

[90] “Governing Open: Come learn about governance with us!,” Governing Open, [Online]. Available: https://governingopen.com.

[91] J. Freeman, “The Tyranny of Structurelessness,” 1970. [Online]. Available: https://www.jofreeman.com/joreen/tyranny.htm.

[92] H. Ramón, “Roadmap — Iced,” whimsical, 2023. [Online]. Available: https://whimsical.com/roadmap-iced-7vhq6R35Lp3TmYH4WeYwLM.

[93] “Sniffnet : Real-Time Network Traffic Monitoring,” TrishTech, 2022. [Online]. Available: https://www.trishtech.com/2022/11/sniffnet-real-time-network-traffic-monitoring/.

[94] “Major Geeks - Sniffnet,” Major Geeks, 2022. [Online]. Available: https://www.majorgeeks.com/files/details/sniffnet.html.

[95] “Sniffnet,” Softpedia, 2022. [Online]. Available: https://www.softpedia.com/get/Network-Tools/Network-Monitoring/Sniffnet.shtml.

[96] A. Kamau, “Monitor Network Traffic on Linux / Windows using sniffnet,” TechViewLeo, 2022. [Online]. Available: https://techviewleo.com/monitor-network-traffic-using-sniffnet/.

[97] “Sniffnet – Surveillez votre trafic réseau en temps réel,” Korben, 2023. [Online]. Available: https://korben.info/sniffnet-surveiller-trafic-reseau.html.

[98] G. Mishra, “Sniffnet: Application to Comfortably Monitor your Network Traffic,” TREND OCEANS, 2023. [Online]. Available: https://trendoceans.com/sniffnet-to-monitor-your-network-traffic/.

[99] A. Mogul, “Console #154 - An Interview with Giuliano of Sniffnet - Rust app to easily monitor network traffic,” Console by CodeSee.io, 2023. [Online]. Available: https://console.substack.com/p/console-154.

[100] E. Bärwaldt, “Analyze network traffic with Sniffnet,” Linux Magazine, 2023. [Online]. Available: https://www.linux-magazine.com/Issues/2023/270/Sniffnet.

[101] thunderbong, “Sniffnet: Open-source, cross platform application to monitor network traffic,” Hacker News, 2023. [Online]. Available: https://news.ycombinator.com/item?id=35991811.

[102] M. Nasi, “Sniffer open source per analizzare e monitorare le connessioni di rete: come funziona Sniffnet,” ilSoftware, 2023. [Online]. Available: https://www.ilsoftware.it/focus/sniffer-open-source-per-analizzare-e-monitorare-le-connessioni-di-rete-come-funziona-sniffnet_26058/.

[103] V. Caschy, “Sniffnet: Software zur Überwachung des Netzwerkverkehrs,” Caschys Blog, 2023. [Online]. Available: https://stadt-bremerhaven.de/sniffnet-software-zur-ueberwachung-des-netzwerkverkehrs/.

[104] xguru, “ Sniffnet - 오픈소스 네트워크 트래픽 모니터링 도구,” GeekNews, 2023. [Online]. Available: https://news.hada.io/topic?id=9224.

[105] “オープンソースのネットワーク監視ツール「Sniffnet 1.2.0」がリリース,” CodeZine, 2023. [Online]. Available: https://codezine.jp/article/detail/17808.

[106] S. RUDRA, “Sniffnet: An Interesting Open-Source Network Monitoring Tool Anyone Can Use,” It's FOSS News, 2023. [Online]. Available: https://news.itsfoss.com/sniffnet/.

[107] J. L., “Master Your Network Monitoring with Sniffnet | Open-Source Tool for Linux,” KeepItTechie, 2023. [Online]. Available: https://www.youtube.com/watch?v=4b5i8XLhZq0&t=12s.

[108] geekpi, “Sniffnet：任何人都可以使用的有趣的开源网络监控工具,” Linux China, 2023. [Online]. Available: https://linux.cn/article-15904-1.html.

[109] S. Palani, “How To Effortlessly Monitor Your Internet Traffic Using Sniffnet Network Monitoring Tool In Linux And Unix,” OS Tech NIX, 2023. [Online]. Available: https://ostechnix.com/sniffnet-network-monitoring-tool/.

[110] kristianpaul, “Sniffnet – Comfortably monitor your internet traffic (like Wireshark),” Hacker News, 2023. [Online]. Available: https://news.ycombinator.com/item?id=36728672.

[111] Z. Xuan, Twitter, 2023. [Online]. Available: https://twitter.com/vikingmute/status/1680389881607094272?s=20.

[112] O. Danshyn, “#10: Rust - gping, Sniffnet, Iced and Mountpoint-S3,” Behind The Mutex, 2023. [Online]. Available: https://themutex.substack.com/p/10-rust-gping-sniffnet-iced-and-mountpoint?r=3fc3a&utm_campaign=post&utm_medium=web.

[113] R. Yifeng, “科技爱好者周刊（第 263 期）：开源软件如何赚钱？,” 2023. [Online]. Available: https://www.ruanyifeng.com/blog/2023/07/weekly-issue-263.html.

[114] “Recently Popular Mac Apps (Issue #11),” MacBed, 2023. [Online]. Available: https://www.macbed.com/recently-popular-mac-apps-issue-11/.

[115] “Python pro vylepšení fotek. Linkedin a autorské právo ~ 23. července,” 365TIPŮ, 2023. [Online]. Available: https://365tipu.substack.com/p/python-pro-vylepseni-fotek-linkedin.

[116] J. Mrugalski, “unknowNews: 28 lipca 2023,” unknowNews, 2023. [Online]. Available: https://news.mrugalski.pl/post/724036431766601728/28-lipca-2023.