Programming Languages and Paradigms (HS24)
Project Proposal in Rust
-
Concurrent high-performance network scanner

Dominik Arnold
dominikjohann.arnold@uzh.ch
11-747-680

November 7, 2024

Description

This project aims at developing a Commandline interface (CLI) utility similar to nmap [1] and other network analysis tools (although obviously much less powerful). The program can be run from the command line and will scan a given IPv4 address range and subnet mask for active hosts within the subnet. It will print progress and feedback to the user and will summarise the detected hosts.

Motivation

I'm currently taking a course about communications systems and network protocols which sparked an interest in those topics. One utility which is used often is the `nmap` command to scan all possible active hosts in a given network. I was surprised to see that a network scan can be rather slow even for small networks.

At the same time, I learned about Rust and its ability to produce fast and reliable CLI tools that can support concurrent and async processing relatively straightforward. Therefore I want to try to implement my own version of a network scanner and see if I can optimize it to be as fast as possible.

Rust features used in this project

- **Ownership and Borrowing**. In order to understand how to implement concurrency one must naturally have a solid understanding of this fundamental Rust principle
- **Error Handling**. The same is true for the Rust specific error handling with `Result` and `Option`.
- **Concurrency and async processing**. At the core of the idea lies the concurrent processing of the network IP range in order to speed up the runtime and utilize available multi-threading the OS provides. Apart
- **Memory Safety**. A big part why concurrent processing is so big with Rust is the possibility to have multiple processes access a shared memory in a safe way. This feature is also a key reason I chose this project.

Architecture

There will be three core modules in its most basic form: The **network module** will provide the functionality to interact with a network interface and send data to possible hosts to detect its status and gather information. The **concurrency module** will be the orchestrator which will spawn processes in parallel to speed up the scanning. For this end, it will receive an IP range and organize the splitting into sub ranges that can be scanned by the processes. It will also be responsible to spawn the appropriate number of processes. The **UI module** will provide the UI functionality of the CLI tool; it will receive user input when necessary and log the process to the user.

Scope

Must-haves

- Be able to scan a host given by an IPv4 address and subnetmask for his status and information.
- Concurrently spawn multiple processes that can scan on different parts of a given IP range.
- CLI interface that provides the user with basic information about the process and the final state.

Nice-to-haves

- Useful man page.
- CLI flag to choose verbosity level of output.
- CLI flag to limit the number of parallel processes spawned.
- An interactive mode that keeps the process running and informs the user as soon as a host enters or leaves the network.
- Potential other flags that enhance the usability of the tool.

Out-of-scope

- A graphical user interface.
- Any information that goes beyond basic information like status, MAC addresses and maybe latency when pinging.

Potential challenges

For this project I will depend on existing solutions to interact with the OS network interfaces to actually send and receive information through the network. A short research indicates that there are well supported solutions like `pnet` [2], but since I'm new to Rust and the low-level networking functionality as well, the usage of those libraries could pose a challenge.

Another challenge I cannot asses is how well the program will translate between different OS. In theory, this should work since it's a Rust binary, but there might be problems which I don't see yet.

References

[1] nmap. Nmap.org. https://nmap.org/, 2024. Accessed: October 21, 2024.

[2] pnet. pnet. https://docs.rs/pnet/latest/pnet/, 2024. Accessed: October 21, 2024.