VoidPhone Project
P2PSEC (IN2194)
Prof. Dr.-Ing Georg Carle
Due Date: 2020-05-31

Initial Report
Team 48 - Bushaltestelle
Dominik Stolz, Oliver Jacobsen

Team Composition

We are Dominik Stolz and Oliver Jacobsen, both studying Informatics in the sixth semester of our Bachelor's degree. As a team name we chose "Bushaltestelle" [1] and will be working on the Onion module.

Programming Language

Our choice for the programming language to implement the Onion module in is Rust, which describes itself as A language empowering everyone to build reliable and efficient software." [1]. This is achieved by providing the level of control one would expect from a systems programming language while avoiding common memory and thread safety issues. During compile time the Rust compiler performs extensive static analysis. Rust also features zero-cost abstractions enabling fearless concurrency.

Operating System

We are going to target GNU/Linux running on the x86-64 architecture. In particular we will test our module on the Arch Linux distribution, as this is the system we will be developing on and is generally the most used OS among university students and personnel. Apart from that we will try to ensure compatibility with other targets by preferring cross-platform libraries.

Build System

Our project will use *Cargo* which is bundled with Rust and serves as a build system and package manager. It can also run tests and generate documentation. We also consider publishing our project in the Rust package registry *crates.io*, so it can be easily installed using Cargo.

Quality Measures

In order to ensure high quality of our software, we are going to take advantage of the concept of test driven development. Since unit tests are often prone to the same bugs as the software (especially when they are written by the same developer), we opted to apply the concept of property based testing in out project. Instead of writing explicit test cases, only properties of the software that should hold at any time are tested, whilst the explicit inputs to test these properties on are generated by a library. This makes writing of test cases easier and ensures that the implementation is complied with its definition. Furthermore, we are going to use unit and integration tests whenever appropriate to cover the entirety of our implementation. To ensure this goal we will use tools which determine test coverage. We would also like to use the continuous integration feature provided by GitLab. However, this would require the

[1] Bushaltestelle is German for bus stop

configuration of a runner. We would very much appreciate it if the admins of the GitLab group could provide a shared runner, so that all teams can use it for their projects.

Available Libraries

In the Rust ecosystem libraies are called *crates* and can be found on the central package registry *crates.io*. This registry currently counts over 40.000 crates [2] with many new crates published every day. We found the following crates which might be helpful in our implementation:

- async-std [3] — A runtime for async IO.
- protobuf [4] — Serialization for the P2P protocol.
- rustls [4] — A TLS implementation.
- byteorder [5] — A helper for parsing the API protocol.
- quickcheck [6] — A library for property based testing.

Software License

We chose the permissive MIT License (or Expat License) [7] which only requires preservation of copyright and license notices, because we do not intent to restrict access to our code or the way which in it is used. Thus our code is accessible to the broadest group of potential users. Any more restrictive form of license might restrain people from using our work.

Previous Programming Experience

We have practical experience from the computer networks and distributed systems lecture about sockets and network related programs. Although we already have some experience with Rust, we particularly chose this language to improve our familiarity with it.

Planned Workload Distribution

For a small team, creating a strict and static work schedule is unnecessarily complex and is often hindering progress. Instead, we are planning to split the workload dynamically into smaller tasks and assign work depending on upcoming challenges to achieve fair workload balancing. We especially want to avoid one-sided development, like one member exclusively working on tests.

References

[1] *Rust Programming Language*, https://www.rust-lang.org/, visited May 2020.

[2] *Rust Package Registry*, https://crates.io/, visited May 2020.

[3] *async-std*, https://crates.io/crates/async-std, visited May 2020.

[4] *protobuf*, https://crates.io/crates/protobuf, visited May 2020.

[5] *byteorder*, https://crates.io/crates/byteorder, visited May 2020.

[6] *quickcheck*, https://crates.io/crates/quickcheck, visited May 2020.

[7] *Expat License*, https://directory.fsf.org/wiki/License:Expat, visited May 2020.