VoidPhone Project **Initial Approach Report**
P2PSEC (IN2194) Team 45 - Rhodium
Prof. Dr.-Ing Georg Carle Benedikt Seidl, Stefan Su
Sree Harsha Totakura, Dr. Heiko Niedermayer Due Date: 05/25/18

Team Composition

We are Benedikt Seidl and Stefan Su, both studying Informatics in the sixth semester of our Bachelor's degree. As a team name we chose "Rhodium" [1] and will be working on the Distributed Hash Table (DHT) project.

Programming Language

We decided to use Rust [1] to implement the DHT. Rust is a modern system programming language with features known from high level programming languages. It includes a strong type system and supports object orientation as well as functional programming. This allows to write safe and fast code on a high abstraction level. One does not have to pay for this with runtime costs like garbage collection or interpreters as Rust compiles to native LLVM code using zero-cost abstractions [2].

Operating System

As operating systems, we use macOS and Linux which have the advantage of being Unix-based and thus supporting most development tools and libraries that may be needed during development. However, since Rust supports all major operating systems, it should also be possible to run the software under Windows, at least as long as we do not require any special libraries.

Build System

Rust comes with its own build system called "Cargo" [3]. Cargo serves as a build tool but can do much more. It only compiles files that have changed since the last compilation. Furthermore, it serves as a test runner, can create HTML documentation based on inline comments and manages the dependencies of our project. Since Cargo is so easy to setup, we currently do not plan to provide further build files for Docker or Vagrant. If it turns out that our build process is more complicated than expected, we can still integrate a more elaborate build system.

Quality Measures

Rust supports writing automated unit and integration tests natively and with Cargo it is a breeze to run them. By applying the concept of test driven development (TDD) we make sure that our code matches the intended design. We also aspire to maintain a high test coverage. Furthermore, by its language design Rust avoids common bugs. It does not allow invalid memory access such as use after free or null pointers and its strong type system guarantees that all type errors are caught at compile time [4].

[1] Rhodium is the element with atomic number 45.

Available Libraries

The Rust standard library supports the usual network functionality like opening and listening for TCP connections and writing bytes. For more advanced functionality, one can use packages which are called "crates" in the context of Rust. There exists a extensive package registry called "Crates.io" [5] which contains a lot of useful libraries. However, as with all unmoderated package registries we have to be careful which packages to include in order to not download malicious code. If one specific feature is not available through a crate, one can also create bindings to existing C libraries.

Software License

We decided to publish our project under the GNU Affero General Public License (AGPL) [6] version 3. It is a free software license enforcing the copyleft principle. This means that peers who make changes to the code are required to publish it under the same license. Improvements to the software are therefore given back to the open source community and have to be made publicly available.

Previous Programming Experience within the Team

We already know each other since the beginning of our studies and worked together on the "Verleihtool" software project for the student council [7]. Therefore we do not expect any specific difficulties in our working dynamic. During our networking course, we have acquired basic knowledge in dealing with networking protocols and TCP standards. We plan to steadily learn the benefits of the Rust language features throughout the course of this project.

Workload Sharing

Based on classical software engineering principles, we plan on dividing the project into smaller tasks in order to split the code into independent components. This allows us to work independently after defining our common interfaces. Furthermore, we plan to work on individual branches but merge often into the master branch to integrate our changes continuously.

References

[1] *The Rust Programming Language*, https://www.rust-lang.org/en-US/index.html, visited May 2018.

[2] *Frequently Asked Questions*, https://www.rust-lang.org/en-US/faq.html, visited May 2018.

[3] *The Cargo Book*, https://doc.rust-lang.org/cargo/index.html, visited May 2018.

[4] *The Rust Programming Language*, 2nd ed., https://doc.rust-lang.org/stable/book/second-edition/, visited May 2018.

[5] *Rust Package Registry*, https://crates.io/, visited May 2018.

[6] *Affero General Public License*, https://www.gnu.org/licenses/agpl.html, visited May 2018.

[7] *Verleihtool*, https://github.com/verleihtool/verleihtool, visited May 2018.