VoidPhone Project: Initial Report

Group 14

May 21, 2024

Muhammed Orhun Gale
Kerem Kurnaz

Group Name:	14
Number of Team Members:	2
Module:	DHT
Implementation:	Kademlia

DHT Implementation

We have decided to use the Kademlia protocol to implement the Distributed Hash Table (DHT) module for the VoidPhone project. Several factors, including the availability of lecture materials that offer a strong foundation for Kademlia's general working logic, influenced this decision. Compared to other DHTs like Chord and Pastry, Kademlia's algorithm is easier to implement because it is more straightforward and has efficient node lookup procedures. It ensures effective and fast network searches by calculating distance using an XOR-based metric. Known for its scalability, Kademlia is a reliable option for a scalable P2P network because of its ability to manage a large number of nodes effectively. High reliability is ensured by its built-in redundancy and fault tolerance, which is essential for preserving an anonymous and unobservable VoIP application. Additionally, Kademlia is a sensible and strategic choice for our project's DHT module due to its demonstrated performance in a variety of P2P applications and its adaptable routing table structure, which effectively handles dynamic peer operations.

Programming Language and Operating System

We chose macOS as our operating system and Go as our programming language for a number of reasons. Go, known for its efficiency and simplicity, provides great concurrency support through goroutines, making it an ideal choice for implementing the concurrent operations needed by a DHT. Its performance provides effective handling of network and computational tasks, comparable to that of lower-level languages like C++. Go is also a good fit for our project because of its robust networking support and an extensive standard library. We chose macOS because it offers great development tools and a stable Unix-based environment that works with Go.

Build System

We have decided to use Make as our build system for this project. Make is a flexible build automation tool that gives us the ability to specify a series of actions to be carried out in a Makefile, offering a simple method for building and compiling our Go project. Make allows us to efficiently manage dependencies, automate complicated build processes, and guarantee consistency of our build process across various development environments. Furthermore, Make provides support and integration with our GitLab repository, easing the integration of our CI/CD pipeline. We intend to write tests on a regular basis, so this integration will be especially helpful in ensuring continuous testing and deployment, which will improve the overall quality and reliability of our software.

Quality Assurance Measures

A number of crucial measures will be put in place to guarantee the integrity of our software. We will start by creating thorough test cases, which will include both integration and unit tests. Unit tests, written with Go's integrated testing package, will confirm that distinct modules and functions are correct on their own. The mock "VoidPhone" testing module, which offers client and server scripts for testing API conformance, will be used for integration tests. We will be able to replicate fundamental features in this mockup environment, like the gossip module's information distribution, to make sure that various modules work together properly and maintain the integrity of the entire

system. Static analysis tools like golint and go vet, which assist in identifying possible problems and code inconsistencies early in development, will further strengthen quality control. Periodic code reviews will be carried out to verify coding standards and enable knowledge exchange among team members. In order to automate our build and test procedures, we will also set up a continuous integration (CI) pipeline using GitLab CI/CD. This will guarantee that each commit sets off a sequence of automated tests. Additional precautions include optimizing the application through performance testing with tools like PPROF, measuring test coverage with gotest -cover, and conducting regular security audits with tools like Gosec to find and fix vulnerabilities.

Additionally, we will continuously write comprehensive documentation for the codebase, including detailed comments, API documentation, and usage instructions, ensuring that the documentation is thorough and shows the entire development process.

Libraries and Tools

We have chosen a number of libraries to help with the DHT module's Go implementation. These libraries will guarantee reliable functionality and speed up development. Go-libp2p will be used for peer-to-peer networking, and net/http will be used to handle HTTP requests and responses. Go-ini will manage configuration files, and go-crypto will handle cryptographic operations. We will use Go-memstats to gather runtime memory statistics and pprof to profile CPU and memory usage in order to optimize performance. Essential synchronization techniques for concurrency management will be provided by the sync package, and errgroup will aid in the effective management of groups of goroutines. Go-routines will help with concurrent operation management and debugging. Furthermore, Testify will be utilized for both writing and executing tests, guaranteeing strong assertion techniques and mocking functionality. These libraries will help with a number of project-related tasks, making it easier for us to construct a dependable, safe, and efficient DHT module and to streamline network and API communication.

License

The software for our project will be licensed under the MIT License. With few limitations, anyone can use, modify, and distribute our software under the terms of the permissive open-source MIT License.

Team Experience

Our team consists of two members, Orhun and Kerem. Orhun has a background in software engineering with a focus on distributed systems and networking, particularly in C++. He has worked on a MANET system and has extensive experience in parallel and system programming from his professional work. Kerem is a game developer with expertise in C and Unity, giving him a well-rounded grasp of object-oriented programming. From their undergraduate degrees, both team members have experience with C++ and network programming.

Workload Distribution

We've divided the work according to our individual strengths, and we are going to work together on crucial tasks like network communication, testing, and documentation.

Orhun will take care of node management, making sure that the logic and data structures for controlling DHT nodes, such as routing and node lookup, are reliable. Additionally, he will be focusing on concurrency management, making use of his background in system and parallel programming to make sure that DHT operations can be carried out concurrently without encountering any problems. Orhun will also take care of node-to-node communication, including sending and receiving DHT messages (PUT, GET, SUCCESS, FAILURE), and it will use profiling and optimization to enhance the DHT implementation's performance. Kerem will focus on implementing the key-value storage system, making sure that key-value pairs are efficiently stored, retrieved, and refreshed. Using his knowledge of object-oriented programming, he will implement the encryption and decryption of data for secure communication within the DHT module.

Additionally, Kerem and Orhun will work together on network communication tasks to implement reliable and effective DHT message handling. Both team members will contribute to writing unit tests for the code segments they implemented.