Kadmelia implementation

Mobile and distributed systems, D7024E

Group 12

Authors:

Adrian Fingal	adrfin-1@student.ltu.se
Oscar Johansson	osajoh-9@student.ltu.se
Hugo Hedlund	hughed-0@student.ltu.se

Github:

https://github.com/Hedlund01/d7024e

Department of Computer Science, Electrical and Space Engineering

September 20, 2025

Contents

1 Sprints

1.1 Organizing Sprints

To structure and organize the work efficiently, GitHub Projects is used to create a storyboard for both sprints. The storyboard is available under the Projects tab in the repository, providing an overview of the workflow throughout the course.

1.2 Sprint 1

1.2.1 Pinging, Joining and Finding Nodes

During sprint 1 we focused on completing work on the M* tickets we had planned out in sprint 0. We started of with getting the PING procedure to work. In the process we established how each node is going to perform non-blocking I/O operations. This was done through a main go routine that reads incoming communication and spawns of new go routines to handle the incoming messsages. We also made sure that these communications have to include a message ID, so that we can match responses to requests. This main go routine is also responsible for adding new nodes to the routing table through the already provided routing table implementation.
To test what we had built so far, we had to create a mock network. This was done mostly according to the tutorial provided with the lab instructions. Once we were satisfied with the PING procedure, we moved on to implementing the iterative node lookup procedure. In order for node joining to work. This required a lot of tinkering, especially to make sure that the procudure worked as described in the Kademlia paper.
It had to be capable of sending out multiple requests in parallel, and also handle the responses. All while updating a common shortlist of nodes to probe. In the end we created a separate package for the shortnode list and made sure that it would be thread safe. Thereby simplifying the implementation of the iterative node lookup procedure.
The iterative node lookup procedure itself uses a sort of producer-consumer pattern. Where we always have at most *alpha* go routines available to send out requests. Which are fed their nodes to probe from a channel on the main procedure thread. Which in turn come from the shortlist. As such the main procedure thread is the producer, whilst the go routines are the consumers.

1.2.2 Store and Find Value

After completing the iterative node lookup procedure, we moved on to implementing the STORE and FIND VALUE procedures. As well as their coresponding iterative procedures. A lot of this work could be reused from the iterative node lookup procedure. With only some data structure additions for handling the storing and finding of values on nodes.

1.2.3 Unit Testing

To ensure that we were on the right track during development we created unit tests for almost all of the implemented functionality. Like creating a mock network, as well as writing specific test for the procedures.

1.2.4 Concurrency and Thread Safety

As mentioned earlier, we focused on concurrency and thread safety from the begging. Which has made our diagnosing of issues a lot easier. We use mutexes for any mutually shared data structures and create separate go routines for handling concurrent tasks.

1.2.5 Command Line Interface

Finally we tried to implement simple command line interfaces for the nodes.

1.2.6 Review

To sum up sprint 1, we implemented the M* tickets we had planned out in sprint 0. We focused on concurrency and thread safety from the start. Implemented unit tests to ensure we had sufficient coverage. All of the necessary procedure s were implemented and tested. Finally we also implemented a simple command line interface for the nodes.

1.3 Sprint 2

1.3.1 Planned work

Under sprint 2 we plan to complete as many U* tickets as we can manage. Which of the tickes we will be able to complete we cannot say ahead of time. But we plan to discuss which ones we would like to complete the most. Such that we arrive at some order of priority for the tickets.