**Sprint 2 – Lab group 1**

* Ludovic Peter – ludpet-7
* Thomas Rochette – thoroc-7
* Dolores Raigada Romero – dolrai-7

Mobile and Distributed Computing Systems

**INDEX**

[1. Backlog 2](#_Toc493434144)

[2. Use cases 3](#_Toc493434145)

[3. Requirements 3](#_Toc493434146)

[4. Assumptions 3](#_Toc493434147)

[5. Frameworks and tools used 4](#_Toc493434148)

[6. System architecture description and an overview of the implementation 4](#_Toc493434149)

[7. Questions on the subject: 5](#_Toc493434150)

[8. Link to code 5](#_Toc493434151)

[9. References 5](#_Toc493434152)

1. Kademlia

Kademlia is a peer-to-peer distributed hash table. Its algorithm is compare to a binary tree where the leaves are nodes and each of them is identified by a specific ID. Kademlia protocol ensures that that every node knows of at least one node in each of its subtrees, if that subtrees contains a node. This way, any node can locate another one by knowing its ID.

2. Backlog

|  |  |  |  |
| --- | --- | --- | --- |
| **TASKS** | **Not started** | **In progress** | **Done** |
| **Network:** |  |  |  |
| Have udp asynchronous communication |  |  |  |
| (optional) Communicate through NAT |  |  |  |
| **Kademlia:** |  |  |  |
| Have Lookup for an ID |  |  |  |
| Be able to send ping and update routing table |  |  |  |
| Be able to store data on the network |  |  |  |
| Have lookup for data |  |  |  |
| Be able for a node to join the network |  |  |  |
| **File Storing:** |  |  |  |
| TCP communication to send/receive the file |  |  |  |
| Be able to store an upload a file on the network |  |  |  |
| Have republish system to keep alive data |  |  |  |
| Have pin and unpin system |  |  |  |
| (optional) Preserve data from a node crash (write data on file or database) |  |  |  |
| (optional) Made a simple graphical interface |  |  |  |
| **Quality objective:** |  |  |  |
| Implemented basic unit test |  |  |  |
| Do more elaborated test |  |  |  |
| Check thread safety |  |  |  |

3. Use cases

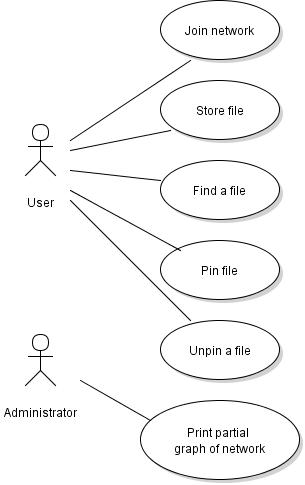


Figure 1: Use cases of Kademlia Assignment

4. Requirements

It is needed to have a Golang compiler on your computer.

5. Assumptions

It’s a not byzantine resilient system, so a node cannot cheat.

A node or the network could have failure, the system is resilient.

6. Frameworks and tools used

- Json instead of Protobuf  
- Visual Studio Code  
- Atom  
- Golang  
- Github

- Graphviz

- Violet UML

7. System architecture description and an overview of the implementation

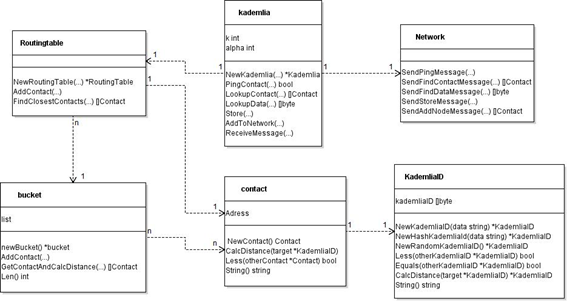


Figure 2: System architecture UML diagram

The Kademlia file contents all the function needed for use case, it’s communicates with other Kademlia class on another node via the network function. We use tcp communication and Json marshalling to communicate between node. The routing table file provides simple task to explore known nodes (for instance to find closest nodes you know). Bucket file contents a simple list of contacts you know and simple functions to explore it and add new nodes. Contact file contents information about a node, and methods to manipulate this.

The main function provides simple CLI interface, you can obtain a list of available command by writing help.

8. Questions on the subject:

- **Describe your threading model in the report**

The threading model used in this assignment is free threaded. By this way, methods can be parallelly run without having to wait one for another. Our threating model is based on creating a threat for each call to functions that need to run parallely, simply by doing “go funct()” where funct() is the name of the function required.

- **What do you think will happen if the network rapidly increases in size? This can be a quite severe issue, how do you handle it? Write a section in the lab report mentioning theoretical problems with Kademlia and possible solutions.**

If network rapidly increases in size could cause network congestion and UDP packets could fail to respond the incoming or outgoing requests to update nodes’ buckets.  This will induce to lock unresponsive contacts and avoid sending them any RPC message. A possible solution to this issue is to have a bigger buffer where to store all the request messages arriving.

Another problem occurs if a lot of nodes join the network on the same time, in this case a data could be stored on any of the k closest nodes (because all are new). It could be resolved by a system to resend store message regularly.

9. Link to code

<https://github.com/lpeter68/mobile-and-distributed.git>

10. References

- Kademlia paper  
- Course slides

- Golang by Gyga Code