Practice 3: Dynamic DHT

Telecommunications Networks Specialist Group (TKN)

January 23, 2024

formalities

This task is part of the portfolio examination. Please note for your
Be sure to provide the appropriate modalities (see Appendix A).

In this submission you expand your implementation from the last task.
The static Distributed Hash Table (DHT) should now be modified so that new nodes can join dynamically, i.e. a dynamic DHT is created. To do this, you should implement the process described in the lecture.

New nodes are added to the appropriate location in the DHT via a join message. Through regular Stabilize messages, nodes in the neighborhood recognize the changed structure of the network and use it.

1. Tests

The tests described below serve as both a guide for implementation and evaluation. Please also pay particular attention to the information in Appendix B.

Each individual test can be understood as a sub-task. Overall, the tests lead to a complete implementation of the task. In some cases, later tests will contradict the previous ones, as the task becomes more specific as it progresses. The tests are therefore written in such a way that they also accept the further development of your solution as correct.

To debug your program, we recommend using wireshark in addition to a debugger (e.g. gdb) . The project preset includes a Wireshark dissector in the rn.lua file. Wireshark automatically uses this when placed in the Personal Lua Plugins directory (Help ÿ About Wireshark ÿ Folders).

1.1. Join sent

So far, DHT nodes have received information about their neighborhood from DHT

Start received. Their identities (IP, port, and ID) were passed via command line parameters, and the identities of the predecessors and successors were passed via environment variables.

In this task we implement a mechanism with which nodes can dynamically establish the second part of this information. To do this, your own identity is still passed on at startup, but additionally the IP and port of an existing node in the DHT, the anchor:

```
1 #./build/webserver <Node IP> <Node Port> <Node ID> [<Anchor IP> <Anchor Port>] 2 ./build/webserver
0.0.0.0 1000 42 1.2.3.4 1395
```

Specifying the anchor is optional: If this is missing, the node starts a new DHT with itself as the only node. However, if this is specified, the implementation should send a join message to it, which contains a description of the joining node:

- Message Type: 4 (Join)
- Hash ID: 0
- Node ID, IP, and Port: Description of the joining node

! Send a join message to the anchor node passed at startup.

1.2. Join processing

Join messages are treated similarly to lookups. They are routed within the DHT until they reach the correct node.

! Forward join messages if the receiving node is not the direct successor of the joining one.

This is the new successor of the joining node. In response, it sends the joining node a notification with a description of itself so that it knows its successor:

- Message Type: 3 (Notify)
- Hash ID: 0
- Node ID, IP, and Port: Description of the new successor

! Replace the predecessor with the joining node and notify it with a notify when you receive a join and are the responsible node.

1.3. Stabilize

Now the DHT is in an inconsistent state. To correct this and restore the integrity of the DHT, Chord provides Stabilize messages. Each node sends this periodically to its successor. In this case, the message format offers more space than we need, but we avoid using multiple formats.
We use the additional fields redundantly:

- Message Type: 2 (Stabilize)
- Hash ID: ID of the sending node
- Node ID, IP, and Port: Description of the sending node

! Send a Stabilize to the successor every second.

1.4. Notify

Nodes respond to received Stabilize messages with a Notify. This tells you the predecessor of the responding node:

- Message Type: 3 (Notify)
- Hash ID: 0
- Node ID, IP, and Port: Description of the predecessor

In general, nodes receive Notify messages containing a description contained by itself.

! Reply to Stabilize messages with a Notify.

At this point, the tests for Practice 2 will no longer work, even if their implementation still respects the environment variables, because the Stabilize messages are not expected by the tests. As a workaround, these tests set the NO_STABILIZE environment variable. This allows you to disable sending Stabilize messages in this case. However, this is optional; your submission will only be assessed based on the tests for Practice 3.

1.5. Update of the successor

However, if a node has joined, the description contained in the notification differs. In this case, the successor has changed and needs to be adjusted.
Accordingly, the following Stabilize messages are sent to this node.

! Correct your successor if you receive a notification and the description of the predecessor is different.

1.6. After Join: Notify contains new predecessor

Another test ensures that a node describes the correct predecessor in its responses to Stabilize messages, even after it has changed due to a join.

1.7. Full test

Analogous to the previous practical tasks, we finally test the entire system. To do this, we create another DHT with five nodes and carry out the same requests.
This time, however, the nodes are started one after the other and join the DHT.

A. Submission formalities

The tasks should be solved by you in group work with up to three course participants each. Each group member must upload their submission individually. You cannot receive any points without submitting your own contribution to ISIS! Add a group.txt file to your submission that contains the name and student number of all group members.
This means that regular group work is not taken into account in our plagiarism check.

You upload your submissions exclusively to ISIS by the relevant submission deadline. If you have technical problems with the delivery, please inform us immediately. To be on the safe side, please send us an archive of your submission by email.

When submitting, please note that the deadline is fixed and there are no exceptions for late submissions or submissions via email. So plan an appropriate buffer before the deadline to avoid any eventualities that could delay your submission.
In cases of illness, the processing time can be adjusted as long as this is documented by a doctor. In this case, send us the certificate as soon as possible.

Submissions are only accepted in .tar.gz format. Create an appropriate archive by appending the following snippet to your CMakeLists and running make package_source in the build folder :

```
1 # Packaging 2
set(CPACK_SOURCE_GENERATOR "TGZ")
3 set(CPACK_SOURCE_IGNORE_FILES ${CMAKE_BINARY_DIR} /\\..*$ .git .venv) 4
set(CPACK_VERBATIM_VARIABLES YES) 5
include(CPack)
```

We strongly recommend that you download your archive yourself after submitting it, unzip it and run the tests. This allows you to avoid errors such as empty submissions, missing source files, typos, incompatibilities, incorrect archive formats, and much more.

B. Testing and Evaluation

You can find the individual tests in the default as test/test_praxisX.py. These can be run with pytest :

```
1 pytest test # Run all tests 2 pytest test/
test_praxisX.py # Only the tests for a specific note 3 pytest test/test_praxis1.py -k test_listen #
Limit to a specific test
```

If pytest is not installed on your system, you can probably install it using the package manager, for example apt or pip .

Your submissions will be automatically evaluated based on the tests of the task.
Note that your implementation should not rely on the values used (Node IDs, Ports, URIs, . . .), these may differ for evaluation. Additionally, you should not modify the tests to ensure that the semantics are not

is accidentally changed. An exception to this is of course updates to the tests, which we may announce in order to correct any errors.

We have the following expectations for your submissions:

- Your submission must be a CMake project.
- Your submission must contain a CMakeLists.txt.
- Your project must create a target webserver with the file name webserver (case-sensitive).
- Your submission must contain internal CMake variables, especially CMAKE_BINARY_DIR and Leave CMAKE_CURRENT_BINARY_DIR unchanged.
- Your program must function correctly on the EECS pool computers1.
- Your submission must be created using CPack (see above).
- Your program must pass the tests from the current sheet, not also previous.
- If your program behaves non-deterministically, the evaluation will also be non-deterministic.

To verify these requirements you should:

- use the test/check_submission.sh script included in the default:

```
1 ./test/check_submission.sh praxisX
```

- Test your submission on the test systems.

Any errors that occur will result in the evaluation failing and your submission will be graded accordingly.

1We carry out the evaluation on the EECS Ubuntu 20.04 systems, which also applies to you can be accessed locally or via SSH .