Praxis 3: Dynamische DHT

Fachgruppe Telekommunikationsnetze (TKN)

23. Januar 2024

Formalitäten

Diese Aufgabenstellung ist Teil der Portfolioprüfung. Beachten Sie für Ihre Abgaben unbedingt die entsprechenden Modalitäten (siehe Anhang A).

In dieser Abgabe erweitern Sie Ihre Implementierung aus der letzten Aufgabenstellung. Die statische Distributed Hash Table (DHT) soll nun so modifiziert werden, dass neue Nodes dynamisch beitreten können, also eine dynamische DHT entsteht. Dazu sollen Sie den in der Vorlesung beschriebenen Ablauf implementieren.

Dabei werden neue Nodes via `Join`-Nachricht an der passenden Stelle in der DHT hinzugefügt. Durch regelmäßige `Stabilize` Nachrichten erkennen Nodes in der Nachbarschaft die veränderte Struktur des Netzwerks und verwenden diese.

1. Tests

Die im Folgenden beschriebenen Tests dienen sowohl als Leitfaden zur Implementierung, als auch zur Bewertung. Beachten Sie hierzu insbesondere auch die Hinweise in Anhang B.

Jeder einzelne Test kann als Teilaufgabe verstanden werden. In Summe führen die Tests zu einer vollständigen Implementierung der Aufgabe. Teilweise werden spätere Tests den vorigen widersprechen, da sich die Aufgabenstellung im Verlauf weiter konkretisiert. Die Tests sind daher so geschrieben, dass diese auch die weitere Entwicklung Ihrer Lösung als korrekt akzeptieren.

Für das Debugging Ihres Programms empfehlen wir zusätzlich zu einem Debugger (z.B. `gdb`) auch `wireshark` zu verwenden. Die Projektvorgabe enthält einen *Wireshark Dissector* in der Datei `rn.lua`. Wireshark verwendet diesen automatisch, wenn er im `Personal Lua Plugins`-Verszeichnis (`Help` → `About Wireshark` → `Folders`) abgelegt wird.

1.1. Join gesendet

Bisher haben Nodes der DHT die Informationen über ihre Nachbarschaft DHT beim Start erhalten. Über Kommandozeilenparameter wurde ihre Identität (IP, Port, und ID) übergeben, und per Umgebungsvariablen die Identitäten der Vorgänger & Nachfolger.

In dieser Aufgabenstellung implementieren wir einen Mechanismus mit dem Nodes den zweiten Teil dieser Informationen dynamisch etablieren können. Dazu wird weiterhin ihre eigene Identität beim Start übergeben, zusätzlich aber IP und Port einer existierenden Node in der DHT, dem `Anchor`:

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

Dabei ist die Angabe des Anchor optional: Fehlt diese, startet die Node eine neue DHT mit sich als einziger Node. Ist dieser aber angegeben, soll die Implementierung eine `Join` Nachricht an diesen schicken, welche eine Beschreibung der beitretenden Node enthält:

- Message Type: 4 (Join)
- Hash ID: 0
- Node ID, IP, und Port: Beschreibung der beitretenden Node

> **!** Senden Sie eine `Join` Nachricht an den beim Start übergebenen Anker-Knoten.

1.2. Join Verarbeitung

`Join` Nachrichten werden ähnlich wie Lookups behandelt. Sie werden innerhalb der DHT weitergeleitet, bis sie die korrekte Node erreichen.

> **!** Leiten Sie `Join` Nachrichten weiter, wenn die empfangende Node nicht der direkte Nachfolger der beitretenden ist.

Dies ist der neue Nachfolger der beitretenden Node. Als Reaktion sendet dieser der beitretenden Node ein `Notify`, mit einer Beschreibung von sich selbst, sodass diese ihren Nachfolger kennt:

- Message Type: 3 (Notify)
- Hash ID: 0
- Node ID, IP, und Port: Beschreibung des neuen Nachfolgers

> **!** Ersetzen Sie den Vorgänger durch die beitretende Node und benachrichtigen diese mit einem `Notify`, wenn Sie ein `Join` empfangen und die verantwortliche Node sind.

1.3. Stabilize

Nun ist die DHT in einem inkonsistenten Zustand. Um dies zu korrigieren und die Integrität der DHT wiederherzustellen sieht Chord `Stabilize` Nachrichten vor. Diese sendet jede Node periodisch an ihren Nachfolger Das Nachrichtenformat bietet in diesem Fall mehr Platz als wir benötigen, wir vermeiden allerdings, mehrere Formate zu verwenden. Die zusätzlichen Felder belegen wir redundant:

- Message Type: 2 (Stabilize)
- Hash ID: ID der sendenden Node
- Node ID, IP, und Port: Beschreibung der sendenden Node

> **!** Senden Sie sekündlich ein `Stabilize` an den Nachfolger.

1.4. Notify

Nodes beantworten empfangene `Stabilize` Nachrichten mit einem `Notify`. In diesem wird der Vorgänger der antwortenden Node mitgeteilt:

- Message Type: 3 (Notify)
- Hash ID: 0
- Node ID, IP, und Port: Beschreibung des Vorgängers

Im Allgemeinen empfangen Nodes also `Notify` Nachrichten, mit die eine Beschreibung von sich selbst enthalten.

> **!** Beantworten Sie `Stabilize` Nachrichten mit einem `Notify`.

An dieser Stelle werden die Tests für Praxis 2 nicht mehr funktionieren, auch wenn ihre Implementierung weiterhin die Umgebungsvariblen beachtet, da die `Stabilize` Nachrichten von den Tests nicht erwartet werden. Als Workaround setzen diese Tests die `NO_STABILIZE` Umgebungsvariable. Dies erlaubt, das Senden von `Stabilize` Nachrichten in diesem Fall zu deaktivieren. Dies ist allerdings optional, Ihre Abgabe wird nur anhand der Tests für Praxis 3 bewertet.

1.5. Update des Nachfolgers

Wenn allerdings eine Node beigetreten ist, weicht die im `Notify` enthaltene Beschreibung allerdings ab. In diesem Fall hat sich der Nachfolger geändert und muss angepasst werden. Entsprechend werden folgende `Stabilize` Nachrichten an diese Node gesendet.

> **!** Korrigieren Sie ihren Nachfolger, wenn Sie ein `Notify` empfangen und die Beschreibung des Vorgängers abweicht.

1.6. Nach Join: Notify enthält neuen Vorgänger

Ein weiterer Test stellt sicher, dass eine Node in ihren Antworten auf `Stabilize` Nachrichten den korrekten Vorgänger beschreibt, auch, nachdem sich dieser aufgrund eines Joins geändert hat.

1.7. Voller Test

Analog zu den vorigen Praxisaufgaben testen wir zuletzt das Gesamtsystem. Hierfür erstellen wir wieder eine DHT mit fünf Nodes, und führen die gleichen Requests durch. Diesmal werden die Nodes allerdings der Reihe nach gestartet und treten der DHT bei.

A. Abgabeformalitäten

Die Aufgaben sollen von Ihnen in Gruppenarbeit mit jeweils *bis zu drei Kursteilnehmern gelöst* werden. *Jedes Gruppenmitglied muss seine Abgabe* einzeln hochladen. Ohne eine eigene Abgabe auf ISIS können Sie keine Punkte erhalten! Fügen Sie Ihrer Abgabe eine Datei `group.txt` hinzu, die Name und Matrikelnummer aller Gruppenmitglieder enthält. Dadurch bleibt die reguläre Gruppenarbeit bei unserem Plagiarismuscheck unbeachtet.

Ihre Abgaben laden Sie ausschließlich auf ISIS bis zur entsprechenden *Abgabefrist* hoch. Sollten Sie technische Probleme bei der Abgabe haben, informieren Sie uns darüber unverzüglich. Lassen Sie uns dabei auch zur Sicherheit ein Archiv Ihrer Abgabe per Mail zukommen.

Beachten Sie bei der Abgabe, dass die Abgabefrist fix ist und es *keine Ausnahmen für späte Abgaben oder Abgaben via E-Mail* gibt. Planen Sie also einen angemessenen Puffer zur Frist hin ein, um Eventualitäten, die Ihre Abgabe verzögern könnten, vorzubeugen. In Krankheitsfällen kann die Bearbeitungszeit angepasst werden, sofern diese ärztlich belegt sind. Senden Sie uns in diesem Fall so bald wie möglich das Attest zu.

Abgaben werden nur im `.tar.gz`-Format akzeptiert. Erstellen Sie ein entsprechendes Archiv, indem Sie das folgende Snippet an Ihre `CMakeLists` anhängen und im Build-Ordner `make package_source` ausführen:

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

Wir empfehlen *dringend*, nach der Abgabe Ihr Archiv einmal selbst herunterzuladen, zu entpacken, und die Tests auszuführen. Dadurch können Sie Fehler, wie leere Abgaben, fehlende Quelldateien, Tippfehler, Inkompatibilitäten, falsche Archivformate, und vieles mehr vermeiden.

B. Tests und Bewertung

Die einzelnen Tests finden Sie jeweils in der Vorgabe als `test/test_praxisX.py`. Diese können mit `pytest` ausgeführt werden:

```
pytest test # Alle tests ausführen
pytest test/test_praxisX.py # Nur die Tests für einen bestimmten Zettel
pytest test/test_praxis1.py -k test_listen # Limit auf einen bestimmten Test
```

Sollte `pytest` nicht auf Ihrem System installiert sein, können Sie dies vermutlich mit dem Paketmanager, beispielsweise `apt`, oder aber `pip`, installieren.

Ihre Abgaben werden anhand der Tests der Aufgabenstellung automatisiert bewertet. Beachten Sie, dass Ihre Implementierung sich nicht auf die verwendeten Werte (Node IDs, Ports, URIs, ...) verlassen sollte, diese können zur Bewertung abweichen. Darüber hinaus sollten Sie die Tests nicht verändern, um sicherzustellen, dass die Semantik nicht

unbeabsichtigterweise verändert wird. Eine Ausnahme hierfür sind natürlich Updates der Tests, die wir gegebenenfalls ankündigen, um eventuelle Fehler zu auszubessern.

Wir stellen die folgenden Erwartungen an Ihre Abgaben:

- Ihre Abgabe muss ein `CMake` Projekt sein.
- Ihre Abgabe muss eine `CMakeLists.txt` enthalten.
- Ihr Projekt muss ein Target `webserver` mit dem Dateinamen `webserver` (case-sensitive) erstellen.
- Ihre Abgabe muss interne CMake Variablen, insbesondere `CMAKE_BINARY_DIR` und `CMAKE_CURRENT_BINARY_DIR` unverändert lassen.
- Ihr Programm muss auf den EECS Poolrechnern[1] korrekt funktionieren.
- Ihre Abgabe muss mit CPack (siehe oben) erstellt werden.
- Ihr Programm muss die Tests vom jeweils aktuellen Zettel bestehen, nicht auch vorherige.
- Wenn sich Ihr Program nicht deterministisch verhält, wird auch die Bewertung nicht deterministisch sein.

Um diese Anforderungen zu überprüfen, sollten Sie:

- das in der Vorgabe enthaltene `test/check_submission.sh`-Script verwenden:

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

- Ihre Abgabe auf den Testystemen testen.

Fehler, die hierbei auftreten, werden dazu führen, dass auch die Bewertung fehlschlägt und Ihre Abgabe entsprechend benotet wird.

[1] Die Bewertung führen wir auf den Ubuntu 20.04 Systemen der EECS durch, welche auch für Sie sowohl vor Ort, als auch via SSH zugänglich sind.