Technische Universität Berlin

TKN

Praktikum

Rechnernetze

Projekt 3:

Erweiterte Distributed Hash Table

In diesem Projekt erweitern Sie ihre Implementierung einer Distributed Hash Table (DHT) basierend auf dem vorigen Projekt. Die erweiterte DHT wird nicht mehr komplett statisch sein, sondern den Beitritt neuer Nodes erlauben. Ihre finale Abgabe wird anhand von automatisierten Tests bewertet.

Hinweise

Die praktischen Aufgaben sind in Kleingruppen von bis zu vier Personen zu lösen. Reichen Sie deshalb Ihren Quelltext bzw. Lösungen dieser Aufgaben bis zum **14.02.2023 23:59 Uhr per ISIS** ein. Aufgaben werden automatisch sowohl auf Plagiate, als auch auf Korrektheit getestet, halten Sie sich deshalb genau an das vorgegebene Abgabeformat.

Auf der ISIS-Seite zur Veranstaltung finden Sie zusätzliche Literatur und Hilfen, insbesondere den *Beej's Guide to Network Programming Using Internet Sockets*[1], für die Bearbeitung der Aufgaben!

Dynamische DHT

Im vorigen Projekt haben Sie eine DHT auf Basis des Chord-Protokolls implementiert. Die Struktur der DHT war allerdings statisch: Beim Start der einzelnen Nodes wurden diesen Informationen über deren Nachbarschaft in der DHT übergeben. Die dadurch definierte Struktur der DHT hat sich im Verlauf der Ausführung nicht verändert. Außerdem verwendet ihre Implementierung bisher keine *Fingertable*, sodass jeder `Lookup` sukzessive über die direkten Nachbarn der Peers aufgelöst werden muss.

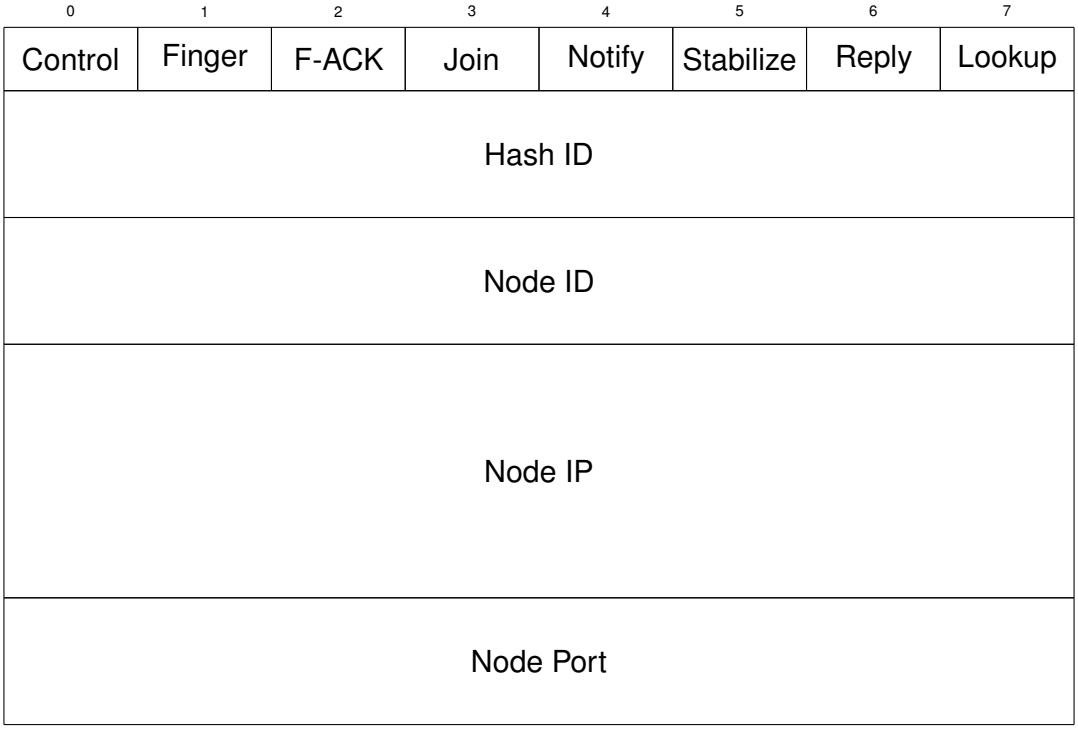
[1] https://beej.us/guide/bgnet

In diesem Projekt erweitern Sie die Implementierung um Funktionalität die für eine dynamische DHT benötigt wird, sowie um eine Fingertable die effizientere Lookups ermöglicht. Dadurch können Nodes der DHT beitreten, sie müssen dafür nur einen existierenden Teilnehmer der DHT kennen. Der Ablauf dieses `Join`-Vorgangs wird im Detail in der Vorlesung besprochen. Kurz zusammengefasst finden dabei folgende Schritte statt:

1. Die beitretende Node (`a`) sendet eine `Join`-Nachricht an die bekannte Node der DHT.
2. Die `Join`-Nachricht wird weitergeleitet bis Sie den zukünftigen Nachfolger der neuen Node, Node `b` erreicht, welcher die Änderung über ein `Notify` bestätigt.
3. Der bisherige Vorgänger von Node `b`, Node `c` stellt durch die Antwort von `b` auf periodische `Stabilize`-Nachrichten fest, dass Node `a` ihr neuer Nachfolger ist.
4. Node `a` wird über die nächste `Stabilize`-Nachricht von Node `c` darüber informiert, dass `c` ihr Vorgänger ist.

Stellen Sie sicher, dass Sie diesen Ablauf verstanden haben, bevor Sie mit der Implementierung beginnen! Visualisieren Sie sich dazu den Prozess oder vollziehen Sie das Beispiel aus der Vorlesung nach.

Für diese Aufgabe wird das Protokoll um weitere Flags im Header der Kontrollnachricht erweitert um die neuen Operationen zu unterstützen:



Vorgabe

Da dieses Projekt auf der vorherigen Abgabe basiert haben Sie die Möglichkeit zwischen zwei Vorgaben zu wählen:

1. Ihre Abgabe für die statische Implementierung der DHT.
2. Das von uns bereit gestellte Skelett, welches Sie via ISIS herunterladen können (`praxis3.tar.gz`).

Detailierte Instruktionen wie das Projekt kompiliert und ausgeführt wird finden Sie auf dem Aufgabenzettel für das vorherige Projekt.

Verwenden Sie in beiden Fällen die aktualisierte Testbench, welche im Projektskelett enthalten sind. **Wichtig:** Wenn Sie diesen Schritt nicht durchführen, testen Sie Ihre Implementierung mit den falschen Tests! Sie können dies überprüfen indem Sie die Tests initial ausführen: Diese **müssen** zunächst fehlschlagen!:

```
$ sudo -H pip install --force-reinstall
↪  rnvs_tb-2022_projekt3_1-py3-none-any.whl
$ sudo rnvs-tb-dht -s .
```

Dynamische DHT

Passen Sie, basierend auf diesem erweiterten Protokoll, Ihre Implementierung an. In der von uns bereitgestellten Vorgabe sind die Stellen im Code als `TODO` markiert, an denen die Implementierung erweitert werden muss. Diese sind der Aufruf des Peers, die Behandlung von Kontrollnachrichten, sowie das periodische Senden von `Stabilize`-Nachrichten. Beachten Sie, dass dies nicht die einzigen Stellen sind an denen der Code angepasst werden muss, sondern lediglich Startpunkte an denen das neue Verhalten zuerst auftritt.

Aufruf des Peers

In der statischen Implementierung der DHT wurden Peers beim Aufruf Informationen über deren Nachfolger und Vorgänger übergeben. Dies ist nun nichtmehr der Fall. Passen Sie

ihren Peer so an, dass er die folgenden Parameter erwartet:

1. Die IP-Adresse und den Port den dieser verwenden soll
2. Seine ID. Diese ist optional, wenn keine ID übergeben wird soll `0` verwendet werden. Sie können davon ausgehen, dass die IDs eindeutig vergeben werden und müssen daher nicht gesondert prüfen ob die gewählte ID noch frei ist.
3. IP-Adresse und Port einer Node die Teil einer existierenden DHT ist. Dieser Parameter ist ebenfalls optional: Wenn er nicht übergeben wird ist der Peer der erste Teilnehmer einer neuen DHT. Wenn er übergeben wird soll der Peer über die angegebene Node einer existierenden DHT beitreten.

Die Executable soll also wie folgt aufgerufen werden können:

```
$ ./peer IP Port [ID] [Peer-IP Peer-Port]
```

Kontrollnachrichten

Zur Implementierung des Joins müssen Sie Ihre Implementierung so anpassen, dass sie `Join`-, `Stabilize`-, und `Notify`-Nachrichten korrekt interpretiert und gegebenenfalls versendet. Das Intervall in dem `Stabilize`-Nachrichten generiert werden, soll $2\,\mathrm{s}$ betragen.

Überlegen Sie sich hier auch, welche IP-Adresse, Port und ID jeweils bei den neuen Befehlen geschickt werden müssen. Am Ende dieser Aufgabe sollte es möglich sein, eine beliebige Anzahl an Knoten zu starten, die sich immer wieder zu einem gültigen Chord-Ring zusammensetzen. Dabei sollte jeder Knoten seinen korrekten Nachfolger und Vorgänger im Ring kennen und kontaktieren können. Sie können vereinfachend davon ausgehen, dass kein Knoten je den Ring verlässt. **Achtung:** Welchen bereits aktiven Knoten ein neu hinzukommender Knoten beim Chord-Joining kontaktiert, darf nicht durch Sie vorausgesetzt werden und sollte das Ergebnis nicht beeinflussen!

Fingertable

Ihre DHT benutzt noch keine Finger-Tables, um schneller entfernte Knoten auf dem Ring zu kontaktieren. In dieser Aufgabe sollen diese implementiert und evaluiert werden.

Passen Sie Ihre Peers so an, dass diese auf Kommando ihre Finger-Table aufbauen. Dazu sollen zwei zusätzliche Felder im Header benutzt werden, mit denen ein Peer aufgefordert wird, seine Fingertable zu erstellen. Die Felder sind `Finger` für den Befehl und `F-ACK` für die Bestätigung. Nutzen Sie für den Aufbau der Fingertable die Lookup-Operation.

Benutzen Sie Ihre Finger Table, um Lookup-Anfragen effizienter an Ihr Ziel zu bringen.

Abgabe

Die vorgegebene `CMakeLists.txt` ist so konfiguriert, dass Sie damit ein Archiv des Projekts erstellen können:

```
make -Cbuild package_source
```

Das generierte Archiv sollte nun im `build`-Ordner verfügbar sein: `RN-Praxis3-0.1.1-Source.tar.gz`
Laden Sie Ihre so generierte Abgabe bis zum **14.02.2023, 23:59 CET** auf ISIS hoch.
Abgaben in andern Formaten werden nicht akzeptiert bzw. mit 0 Punkten bewertet!

Viel Erfolg :)