TKN Telecommunication Networks Group

Praktikum
Rechnernetze und Verteilte Systeme

Block 4

— DHTs & P2P-Netzwerke —

Termin: 28.11-30.11.2016 & 05.-07.12.2016

1 Theoretische Vorbereitungsaufgaben

Die folgenden Aufgaben sollen Ihnen helfen, sich auf den Vorbereitungstest vorzubereiten. Klären Sie bitte möglicherweise aufkommende Fragen oder Unklarheiten unbedingt *vor* den ISIS-Testaten!

Aufgabe 1:
Beantworten Sie im Kontext von Peer-to-Peer (kurz: P2P) folgende Fragen:

a) Welche Knoten des P2P-Netzwerks bieten den Service an?

b) Sind diese Knoten zuverlässig, d.h. in der Regel immer erreichbar?

c) Was ist der Unterschied zwischen dem Napster- und dem Gnutella-Modell?

d) Wie viele Knoten müssen bei Napster konsultiert werden, um einen Knoten zu finden, der eine bestimmte Datei anbietet? Was sind die Nachteile eines eines solchen zentralisierten Systems?

Aufgabe 2:
In der Vorlesung haben Sie bereits verteilte Hashtabellen (engl. „distributed hash tables“, kurz: DHTs) kennengelernt. Beantworten Sie hierzu bitte die folgenden Fragen:

a) Aus welchen 3 Funktionen besteht ein minimales DHT-Interface mindestens? Wozu dienen diese?

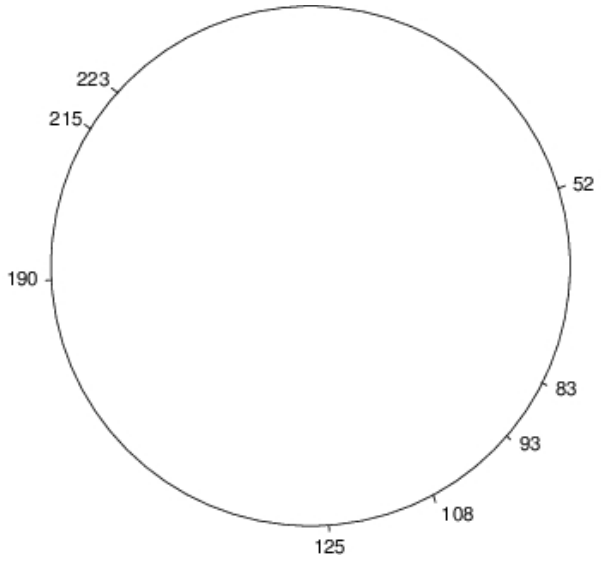
b) Wodurch wird es so einfach, verschiedene Anwendungen mit ein und der selben DHT-Software zu betreiben? Wäre das auch gleichzeitig möglich? Warum?

c) Wieso sind Dynamizität und Größe einer DHT potentiell problematisch? Welche Lösungen für diese zwei Aspekte wurden in der Vorlesung vorgestellt?

d) Welche strukturellen Unterschiede bestehen zwischen DHTs und der Organisation von DNS?

e) Wie funktioniert ein „Chord Lookup“? Wie funktioniert ein „Chord Join“?

Aufgabe 3:
Eine Distributed Hash Table wird mit dem Chord-Protokoll implementiert. Die verwendeten Keys haben eine Länge von jeweils 8 Bits. Die IDs der acht aktiven Knoten sind in folgender Grafik verzeichnet:



a) Was ist eine Finger Table und wie vereinfacht sie den Chord Lookup?

b) Wie lautet die in der Vorlesung vorgestellte allgemeine Formel zum Ermitteln des i-ten Eintrags aus der Finger Table des Knotens mit der ID n im Chord-Ring?

c) Was ergibt sich für die Werte $i = 3$ und $n = 52$? Was für $i = 7$ und $n = 223$? [1]

2 Praktische Aufgaben

Die praktischen Aufgaben sind in Kleingruppen von i. d. R. 3 Personen zu lösen. Die Ergebnisse des ersten Termins führen Sie im zweiten Termin dem Tutor vor. Reichen Sie bitte außerdem den Quelltext und andere Lösungen bis Sonntag vor dem dem zweiten Termin, um 23:55 Uhr per ISIS ein.

Im zweiten Termin werden vertiefende praktische Aufgaben gestellt, während der Tutor die Lösungen des ersten Termins abnimmt. Reichen Sie bitte auch hier wieder den Quelltext und andere Lösungen dieser Aufgaben bis Sonntag vor dem nächsten Termin, um 23:55 Uhr per ISIS ein.

Es besteht in beiden Terminen grundsätzlich Anwesenheitspflicht.

[1] Kontrollergebnisse: 83 und 108

2.1 Im ersten Termin zu lösen

Aufgabe 4:
In der Vorlesung wurde Ihnen das Chord-Protokoll vorgestellt. Damit lässt sich eine Hashtabelle - bspw. die, die sie auf dem letzten Aufgabenblatt implementiert haben - dezentralisiert, d.h. über mehrere Server (sog. Knoten oder Peers) verteilt, speichern und als Distributed Hash Table (kurz: DHT) betreiben.

Ziel dieser Aufgabe ist es, einen Server zu entwickeln, der als solch ein Knoten in einem Chord-Ring fungieren kann: Entweder, indem er als Basis für einen neuen Ring fungiert. Oder, indem er mittels des sogenannten Chord Joining-Verfahrens einem bereits bestehenden Chord-Netzwerk beitritt.

Achtung: In dieser Aufgabe müssen noch keine Fingertables implementiert werden!

a) Schreiben Sie hierzu zunächst einen gewöhnlichen UDP-Server mit folgendem Interface:
   - Werden nur IP-Adresse und Port übergeben, soll der Knoten später als Basis für einen neuen Ring fungieren. Optional soll eine beliebige (16-Bit-)ID übergeben werden können, unter der der Knoten später arbeitet; als Standard-ID ist andernfalls stets 0 zu verwenden.
   - Soll sich der Knoten in einen bestehenden Ring einklinken, so benötigt er hierfür zusätzlich IP-Adresse und Port eines beliebigen anderen, bereits im Ring aktiven Knotens sowie eine freie (16-Bit-)ID. Übergeben Sie diese Informationen als zusätzliche Argumente.

b) Implementieren Sie anschließend das Chord-Joining-Verfahren, wie in der Vorlesung vorgestellt. Hierzu müssen `join`-, `notify`- und `stabilize`-Nachrichten ausgetauscht werden. Verwenden Sie für diese internen Nachrichten bitte das nachfolgend dargestellte Paketformat:

0	1	2	3	4	5	6	7
Intern	Reserved				S	N	J

IP-Adresse (MSB)
...
...
IP-Adresse (LSB)

Port (MSB)
Port (LSB)

ID (MSB)
ID (LSB)

Das Intern-Bit ist für alle drei Nachrichten-Typen auf 1 zu setzen. Wir werden es erst später benötigen. Die darauf folgenden Reserved-Bits sind für etwaige spätere Erweiterungen des Protokolls reserviert und sollten daher auf 0 gesetzt werden. Die Bit-Flags `S`(tabilize), `N`(otify) und `J`(oin) spezifieren die Art der Nachricht; in einer gültigen Nachricht kann also immer nur eines der drei gesetzt sein. Die nachfolgenden Felder enthalten IP-Adresse (4 Byte), Port und ID (je 2 Byte) eines Knotens. Je nach Art der Nachricht ist dies *nicht* äquivalent zu den Daten des Absenders.

Für die Rücksprache sollte ein Knoten mit jeder erhaltenen Nachricht folgendes ausgeben:

- Art der Nachricht
- ID, IP-Adresse und Port des Vorgänger-Knotens
- ID, IP-Adresse und Port des Nachfolge-Knotens

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. Achtung: Welchen bereits aktiven Knoten ein neu hinzukommender Knoten beim Chord-Joining kontaktiert, darf *nicht* durch Sie vorausgesetzt werden und sollte das Ergebnis *niemals* beeinflussen!

Hinweis: Da Sie zum Testen mehrere Server auf unterschiedlichen Ports betreiben und regelmäßig neu starten müssen, empfiehlt es sich, den folgenden Code-Ausschnitt im Server zu vewenden:

```
1 int yes = 1;
2 setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes));
```

c) Integrieren Sie abschließend ihre Hashtabelle aus dem letzten Aufgabenblatt mit jeweils passenden Größen in die einzelnen Knoten. Dabei können Sie vereinfachend davon ausgehen, dass nach dem Eingang des ersten `set`-Befehls dem Chord-Ring keine weiteren Knoten hinzugefügt werden. Gehen Sie ferner von verlustfreier Übertragung aus, auch wenn UDP dies nicht gewährleistet.

- Zum Testen ihrer Implementierung müssen Sie einen UDP-Client schreiben. Sie können sich hier, um Zeit zu sparen, gern an der Implementierung im Beej's Guide [2] orientieren.
- Verwenden sie das Intern-Bit, um interne Nachrichten (d.h. `join`, `notify` und `stabilize`) von externen Nachrichten (also: `ack`, `set`, `get` und `delete`) zu unterscheiden.
- Das Ergebnis eines Chord-Lookups ist nur, welcher Knoten für einen Schlüssel zuständig ist, d.h. es erfolgt zunächst ein Lookup des Schlüssels über irgendeinen Knoten im Ring und dann der eigentliche Befehl an die Hashtabelle direkt an den für den angegeben Schlüssel zuständigen Knoten, den der Chord-Lookup ergeben hat. **Theoretische Aufgaben bzw. die Klausur nehmen immer Bezug auf den Chord-Lookup in diesem ursprünglichen Sinn!**
- Für unsere Aufgabe wollen wir jedoch ein höheres Maß an Transparenz erreichen: Die DHT soll nach außen hin genau so wie Ihre Implementierung aus dem letzten Aufgabenblatt auftreten. Es soll also möglich sein, jedem Knoten im Ring direkt (d.h. ohne vorhergehenden Chord-Lookup) einen Befehl an die Hashtabelle zu senden. Es ist also nicht Aufgabe des Clients, den korrekten, für einen Schlüssel zuständigen Knoten ausfindig zu machen.
- Die Gestaltung eines geeigneten Paketformats für externe Nachrichten ist Ihnen überlassen. Sie können sich hierbei am Aufbau der Pakete aus dem letzten Aufgabenblatt orientieren. Beachten Sie aber, dass externe Nachrichten ggf. (trotz ihrer Benennung) auch innerhalb des Chord-Rings weitergeleitet werden müssen, bis der zuständige Knoten gefunden wurde. Und bedenken Sie diesbezüglich auch, welcher Knoten dabei am sinnvollsten abschließend auf eine Anfrage antworten sollte, um die geforderte Location-Transparency zu erhalten.

Abzugeben sind:

- Der Quellcode ihres UDP-Test-Clients
- Der Quellcode ihres UDP-DHT-Servers
- Ein Makefile o.ä. zum Kompilieren und Ausführen

[2] Siehe: `http://beej.us/guide/bgnet/output/html/multipage/clientserver.html#datagram`

2.2 Im zweiten Termin zu lösen

Aufgabe 5:
Ihre DHT aus Aufgabe 4 benutzt bislang noch keine Finger Tables, wie aus der Vorlesung bekannt. Diese etablieren Abkürzungen, um auf dem Ring weiter entfernte Knoten schneller kontaktieren zu können. Wir wollen diese nun in unsere Implementierung einbauen und die resultierende Zeitersparnis messen:

a) Integrieren Sie hierzu in Ihren Test-Client aus Aufgabe 4c) Zeitstempel (`clock_gettime`, vgl. Aufgabenblatt 2) und ermitteln Sie die durchschnittliche Dauer für jeweils eine `set-`, `get-` und `delete`-Anfrage an die DHT, gemittelt über jeweils 10 Aufrufe mit beliebigen Werten.

b) Passen Sie Ihren Server aus Aufgabe 4 so an, dass nach dem Eingang des ersten `set`-Befehls mit dem Aufbau einer Finger Table begonnen wird. Sie können hierbei ausnutzen, dass die für einen bestimmten Wert zuständigen Knoten auf eine Anfrage mit ihrer IP, ihrem Port und ihrer ID antworten. Vermerken Sie die entsprechenden Informationen für jeden Eintrag ihrer Finger Table.

c) Ist die Finger Table eines Knotens vollständig gefüllt, soll sie für alle weiteren eingehenden Anfragen zuerst konsultiert und - wann immer möglich - die beste Abkürzung gewählt werden.

d) Wiederholen Sie Ihre Messung aus Aufgabenteil a) mit vollständig befüllten Finger Tables.

Abzugeben sind:

- Der Quellcode ihres angepassten UDP-Test-Clients
- Der Quellcode ihres angepassten UDP-DHT-Servers
- Ein Makefile o.ä. zum Kompilieren und Ausführen
- Eine Textdatei mit den Ergebnissen ihre Zeitmessung
- Sowie einer **kurzen Erklärung** ihrer Messergebnisse

3 Vertiefungsaufgaben

Diese Aufgaben sind zu Ihrer eigenen Vertiefung in Hinblick auf die Klausurvorbereitung gedacht:

Aufgabe 6:
Beantworten Sie die folgenden Fragen rund um Peer-to-Peer (kurz: P2P):

a) Nennen Sie zwei wesentliche Herausforderungen, die ein P2P-System zu lösen hat.

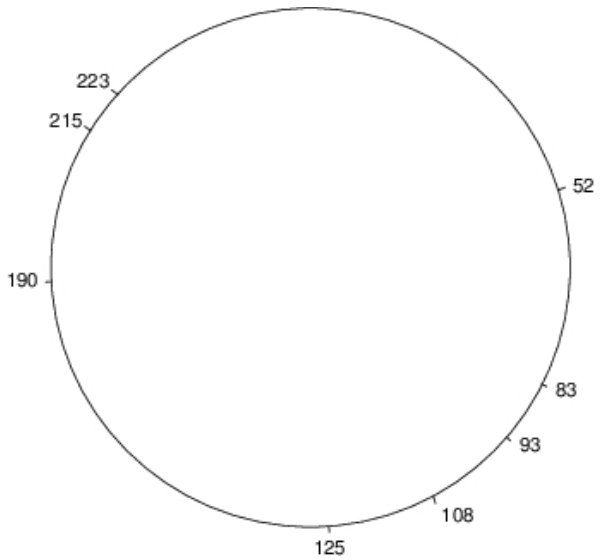
b) Vergleichen Sie das Client-Server-Modell und die P2P-Idee und stellen Sie gegenüber, welche Vor- bzw. Nachteil diese jeweils haben.

c) Wie findet ein Knoten im Gnutella-Netzwerk eine Datei? Was ist dabei das Problem?

d) Was bezeichnen (im Kontext von BitTorrent) jeweils die Begriffe Seeder, Leecher und Tracker?

Aufgabe 7:
Betrachten Sie eine verteilte Hashtabelle (engl. distributed hash table, DHT) mit einem Mesh-Overlay-Netzwerk (das heißt, alle Peers kennen alle anderen Peers im System). Was sind die Vor- und Nachteile eines solchen Systems? Was sind die Vor- und Nachteile einer DHT mit Ringstruktur (ohne Finger Table)?

Aufgabe 8:
Eine Distributed Hash Table wird mit dem Chord-Protokoll implementiert. Die verwendeten Keys haben eine Länge von jeweils 8 Bits. Die IDs der acht aktiven Knoten sind in folgender Grafik verzeichnet:



a) In welchen Knoten sind die Werte mit den Keys 99 bzw. 240 jeweils gespeichert?

b) Knoten 108 möchte erfahren, welcher Knoten für den Key 77 zuständig ist. Zeichnen sie die notwendigen Nachrichten für die Abfrage in die Grafik ein, wenn **keine** Finger Tables benutzt werden. Welche Knoten-ID wird dem anfragenden Knoten gemeldet?

c) Wie viele Nachrichten werden **ohne** Benutzung von Finger Tables maximal benötigt, um im dargestellten System von einem beliebigen Knoten aus einen beliebigen Key abzufragen?