TKN Telecommunication Networks Group

Praktikum
Rechnernetze und Verteilte Systeme

Block 4

— DHT & P2P —

Termin: 24.-26.11.2014 & 1.-3.6.2014

1 Theoretische Vorbereitungsaufgaben

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

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

a) Welche Knoten bieten den Service an?

b) Sind die Knoten, die den Service anbieten, 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, der eine bestimmete Datei anbietet, zu finden?

e) Was sind die Nachteile eines solchen zentralisierten Systems wie Napster?

Aufgabe 2:
Sie kennen Sich aus der Vorlesung mit verteilten Hashtabellen (engl. distributed hash tables, DHTs) aus. Beantworten Sie bitte die folgenden Fragen:

a) Welches minimale Interface bieten DHTs mindestens an (3 Funktionen) und was tun diese Funktionen?

b) Wieso ist es einfach, verschiedene Anwendungen mit der selben DHT Software zu betreiben? Funktioniert das auch gleichzeitig?

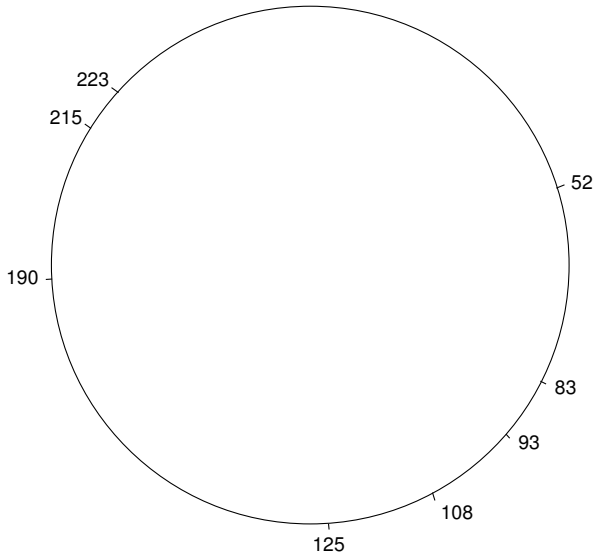
c) Welche Probleme gibt es mit der Dynamizität und der Größe solcher DHTs? Wie sind jeweils die Lösungen, die in der Vorlesung vorgestellt werden?

d) Erinnern Sie sich an die Struktur von DNS. Welche Struktur haben DHTs im vergleich zu DNS?

e) Wie funktioniert der “Chord Lookup”?

f) Wie funktioniert die “Chord Joining Operation”?

Aufgabe 3:
Eine Distributed Hash Table benutzt *Chord* als Implementierung. Die Keys haben eine Länge von 8 Bits. Es sind 8 Knoten vorhanden. Die IDs der Knoten sind in der Grafik verzeichnet.



a) Was ist die allgemeine Formel zum Ausrechnen des i-ten Wertes in der Finger Table des Knotens mit der ID n bei Chord?

b) Welches ist der Eintrag mit $i = 3$ in der Finger Table des Knoten mit der ID $n = 52$?

c) Wie vereinfacht eine Finger Table den Chord Lookup?

2 Praktische Aufgaben

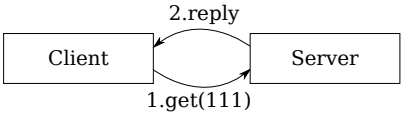
Die praktischen Aufgaben sind in Kleingruppen von i. d. R. 3 Personen zu lösen. Die Ergebnisse des ersten Termins führen Sie im zweiten Termin dem Tutor vor. Reichen Sie bitte des weiteren den Quelltext bzw. Lösungen bis Sonntag vor dem dem zweiten Termin 23:55 Uhr per ISIS ein.

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

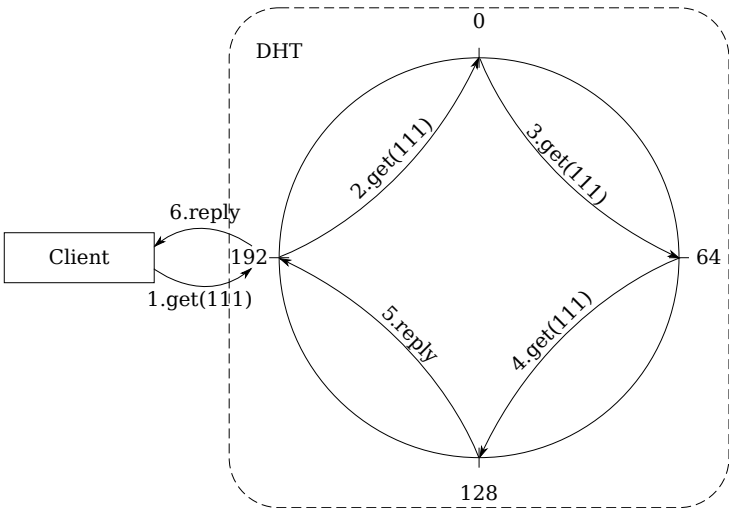
Es besteht in beiden Terminen grundsätzlich Anwesenheitspflicht.

2.1 Im ersten Termin zu lösen

Aufgabe 4:
Sie haben in Block 3 einen Datagram-Socket Client und Server für eine Hash Table geschrieben. In folgender Grafik ist das Zusammenspiel von Server und Client am Beispiel des Auslesens des `<key, value>`-Paares für den Key 130 dargestellt:

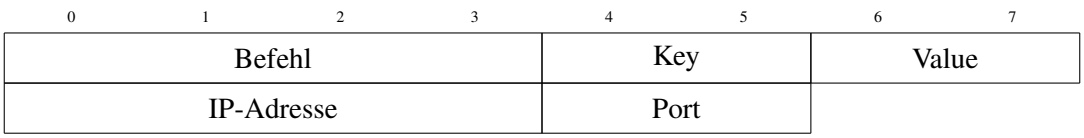


Die Funktionalität soll nun im wesentlichen gleich bleiben, jedoch die Speicherung der Werte über mehrere Prozesse bzw. Rechner verteilt werden. Ihre (jetzt verteilte) Hashtabelle sollte nach wie vor das gleiche Interface nach außen besitzen, also immer noch die Befehle `set()` um einen Wert zu speichern oder zu aktualisieren, `get()` um einen Wert auszulesen und `delete()` um einen Wert zu löschen über Stubs anbieten. Die Speicherung der Werte soll allerdings eine DHT auf Basis von einem vereinfachten Chord übernehmen. Eine DHT ist dabei ein P2P-Netz, also alle Knoten der DHT fungieren als gleichwertige Peers, die sowohl Anfragen entgegennehmen als auch weiterleiten können. Da das Interface sich nicht geändert hat, sollen Sie Ihren Client aus der letzten Aufgabe unverändert verwenden. Die Interaktion zwischen Client und einem der Peers der DHT soll beispielhaft für eine gleichmäßige Topologie aus 4 Peers wie folgt aussehen:



Wie im letzten Block erkennen die Peers Aufrufe bzw. Rückgabewerte für eine bestimmte Methode an einem String-Präfix. Befehle sind nach wie vor jeweils die Strings “SET”, “GET” und “DEL”. Das Paketformat zwischen Client und Peers bleibt dabei erhalten.

Der Befehl wird von einem beliebigen Peer entgegengenommen und von dort ggf. an den Knoten weitergeleitet, der dafür zuständig ist, die jeweiligen Daten zu speichern bzw. auszulesen. Es sollen noch keine Finger Tables implementiert werden. Der Knoten führt den Befehl aus, und antwortet dann wie gehabt mit “OK!” oder “VAL” für Erfolg oder mit “ERR” bzw. mit “NOF” für Misserfolg[1]. Diese Antworten erfolgt direkt an den aufrufenden Peer, welcher die Antwort wiederrum an den Client weiterleitet. Die Strings sollen dabei jeweils Null-terminiert sein, womit sich eine Größe von 4 Bytes ergibt. Der Einfachheit halber sollen Key und Value, falls die Werte nicht gebraucht werden, ebenfalls mit Null aufgefüllt werden. Zusätzlich muss jede Anfrage noch die IP-Adresse (32bit) und den Port (16bit) des aufrufenden Peers enthalten. Die Antworten sollen jeweils die IP-Adresse und den Port des zuständigen Knotens enthalten. Nutzen Sie diesmal die Funktionen `htonl`, `htons`, `ntohl` und `ntohs` (`man byteorder`) um die Werte in die richtige Byte-Reihenfolge zu bringen. Das Paketformat zwischen den Peers soll wie folgt aussehen:



[1] Bitte beachten: Dies ist etwas anders als in der Vorlesung! In der Vorlesung ist der Ergebnis des Chord-Lookups lediglich, welcher Peer für welchen key zuständig ist. Eine Abfrage müsste gesondert stattfinden. Zur Vereinfachung passieren diese Schritte hier zusammen. Die theoretischen Aufgaben bzw. die Klausur nehmen jeweils bezug auf den Chord-Lookup im Sinne der Vorlesungsunterlagen.

Einige Denkanstöße:

- Sie sollten sich überlegen, wie sie Anfragen vom Client und Anfragen von anderen Peers unterscheiden.
- Als Hashing-Funktion soll weiterhin nur ein Bytes des Keys behandelt werden. Überlegen Sie sich, wie groß dabei Variable *m* aus der Vorlesung ist, was sie bedeutet und wie viele IDs es auf dem Ring höchstens gibt.
- Ein Knoten ist jetzt nur noch für einen (meist kleinen) Teil der Hash-Tabelle zuständig. Überlegen Sie sich für welchen. Sie dürfen Ihre alte Hash Table Implementierung verwenden, um diesen Teil der Tabelle zu speichern.
- Schreiben Sie Ihren Hash Table Server so um, dass er als Peer in einem DHT P2P Netz teilnimmt. Dabei liegt der Fokus auf dem Auslesen der Werte. Chord Joining muss nicht implementiert werden - sie sollten sich im Vorfeld eine Ringtopologie aus 4 Prozessen/Knoten mit jeweiligen IDs ausdenken. Übergeben Sie Ihren Peers beim Starten die eigene IP-Adresse, den eigenen Port, die eigene Node-ID, den Vorgänger und Nachfolger (jeweils Node-ID, IP-Adresse und Port). Es darf angenommen werden, dass die Knoten bei jeder Anfrage blockieren, bis sie eine Antwort bekommen haben, also nicht für gleichzeitige Anfragen zur Verfügung stehen. Gehen Sie ferner von Verlust-freier Übertragung aus.
- Nutzen Sie Ihren Test-Client aus dem letzten Block, um Anfragen an einen Teilnehmer der DHT zu schicken und wieder 25 (z.B. zufällige) `<key, value>`-Paare hintereinander zu speichern, auszulesen, zu löschen und abschließend nochmals (vergeblich) auszulesen.[2]

2.2 Im zweiten Termin zu lösen

Aufgabe 5:
Ihre DHT und die Ihrer Kommilitone kommunizieren über das gleiche Protokoll, sollten also zueinander kompatibel sein. Versuchen Sie mit jeweils einer anderen Gruppe Ihre Implementierung gegenseitig zu überprüfen. Funktioniert Ihre Implementierung mit der anderen Implementierung? Falls nein, warum nicht?

Aufgabe 6:

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:

a) Implementieren Sie in Ihrem Client Zeitstempel und lassen Sie die durchschnittliche Zeit einer Antwort auf eine Anfrage (“SET”/“GET”/“DEL”) mit ihren zufälligen Keys ausgeben.

b) Lassen Sie Ihre Peers nach einer geringen Wartezeit, in der sie die restlichen Peers Ihrer DHT starten können, eine Finger-Table aufbauen. Sie können dafür ausnutzen, dass die zuständigen Peers für einen Wert auf eine Anfrage mit Ihrer eigenen IP-Adresse bzw. ihrem eigenen Port antworten werden.

c) Benutzen Sie Ihre Finger Table, um Anfragen effizienter an Ihr Ziel zu bringen.

d) Wiederholen Sie Ihren Test aus dem ersten Aufgabenteil, nachdem sie alle Peers aufgefordert haben ihre Finger Table aufzubauen.

e) Laden Sie Quelltext und Ergebnisse der Messungen bei ISIS hoch.

[2] 25x set(), 25x get(), 25x del(), 25x get()

3 Vertiefungsaufgaben

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

Aufgabe 7:
Beantworten Sie die folgenden Fragen rund um P2P:

a) Nennen sie zwei wesentliche Herausforderungen, die ein Peer-to-Peer-System zu lösen hat.

b) Vergleichen Sie die beiden Technologien Client/Server und P2P 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 ist im Kontext von BitTorrent ein Seeder, Leecher und ein Tracker?

Aufgabe 8:
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 9:
Eine Distributed Hash Table benutzt *Chord* als Implementierung identisch mit Aufgabe 3. Die Keys haben eine Länge von 8 Bits. Es sind 8 Knoten vorhanden.

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 von Aufgabe 3 ein, wenn **keine** Finger-Tables benutzt werden. Welche Knoten-ID wird dem anfragenden Knoten gemeldet?

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