TECHNISCHE UNIVERSITÄT BERLIN
Fakultät IV – Elektrotechnik und Informatik
Fachgebiet Intelligente Netze und Management verteilter Systeme
Prof. Anja Feldmann, Ph.D.
Gregor Maier, Jörg Wallerich, Andi Wundsam

5. Blatt Praktikum Protokolldesign WS 07/08

Aufgabe 1: (30 Punkte) **Ein einfaches Chat-System**

Der einfachste Fall eines Client-Server-basierten Chat-Systems besteht darin, dass alle Clients {a,b,c,...} mit einem einzelnen Server S verbunden sind (vergleiche *Abbildung 1*). Alle Klienten müssen sich zum Server S verbinden, damit sie am Chatsystem teilnehmen können. Eine Nachricht die einer der Clients (z.B. a) über seine Verbindung zum Server an S sendet, wird von S an alle *anderen* Clients {b,c,...} verteilt (flooding), so dass alle die Nachricht lesen können.

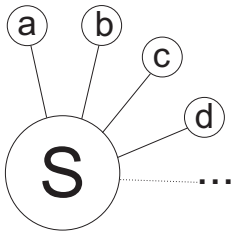

Abbildung 1: Ein rudimentäres Chat-System mit zentralem Server

Die Aufgabe besteht darin ein solch rudimentäres Chat-System zu implementieren.

Dazu soll ein Programm/Skript erstellt werden, welches im Stande ist sich entweder als Client oder auch als Server zu verhalten. Als Client soll es sich zu einem angegebenen Server verbinden, eigene Nachrichten von der Tastatur lesen und an den Server weiterleiten als auch eingehende Nachrichen auf dem Bildschirm anzeigen.

Als Server soll es auf einem TCP Port auf eingehende Verbindungsanfragen lauschen, eingehende Verbindungen akzeptieren und alle eingehenden Nachrichten flooden.

Abzugeben sind:

- Dein Programm (sowohl Quelltext als auch ggf. compiliert im Falle von C, C++ und Java)
- Ggf. eine Beschreibung, was funktionieren sollte, dies leider aber immer noch nicht tut...

Implementierung eines Peer-to-Peer Systems

Mit diesem Aufgabenblatt starten wir in die schrittweise Implementierung eines einfachen Peer-to-Peer Systems, welches auch als Grundgerüst für weitere Aufgaben dienen wird. Wir werden mit einem sehr einfachen Basissystem beginnen und das System im Laufe der nächsten Aufgabenblätter Schritt für Schritt erweitern.

Protokoll-Spezifikation

Das P2P-System benutzt ein einfaches text-basiertes Protokoll das typische Syntax-Elemente realer Protokolle im Internet aufweist. Das System besteht aus zwei Hauptbestandteilen. Der erste Teil besteht aus

einem Mechanismus zum gleichzeitigen Behandlen von mehreren TCP-Verbindungen (ähnlich wie in Aufgabe 1). Der zweite Teil besteht aus dem im Folgenden beschriebenen einfachen Protokoll zum Aufbauen von P2P-Verbindungen, Suchen, und Herunterladen von Dateien und zum Verwalten der Topologie des logischen P2P-Netzes (overlay).
Das Protokoll ist dabei folgendermaßen spezifiziert:

- Das P2P-Anwendungsprotokoll benutzt TCP als Transport-Protokoll.
- In der Version `0.1` besteht jede Anfrage und jede Antwort aus genau einer Text-Zeile
- Jede Zeile wird mit `\r\n` abgeschlossen. Dies erlaubt es mittels des Programms `telnet` manuell eine Protokollsitzung zu simulieren.
- Am Ende jeder Anfrage steht der Protokoll-Identifikations-String (z.B. `P2P/0.1`), wobei `0.1` die Protokoll-Version ist.
- Antworten beginnen immer mit dem Protokoll-Identifikations-String und einem 3-stelligen numerischen Response-Code.
- Antworten auf Anfragen mit Request-ID enthalten die selbe Request-ID wie die Anfrage. Pro Knoten werden Request-IDs höchstens einmal zum Generieren einer neuen Anfrage verwendet.
- Jeder Knoten darf maximal 4 Verbindungen zu anderen Knoten *aktiv aufbauen*, auch wenn er mehr als 4 Knoten kennt.
  Allerdings soll er beliebig viele Verbindungen *annehmen*!
- Ein Knoten, der eine direkte TCP-Verbindung zu einem zweiten Knoten mit einer darüber laufenden Protokollsitzung unterhält, wird als dessen *Nachbar* bezeichnet. Dabei spielt es, wie in P2P-Systemen üblich, keine Rolle, wer die Verbindung initiiert oder wer eine eingehende Verbindungsanfrage angenommen hat.
- Die maximale Suchtiefe ins Peer-to-Peer Netz ist 3. Jeder Knoten generiert Anfragen und Antworten mit dieser TTL. Beim Weiterleiten wird die TTL um eins erniedrigt und die Nachricht weggeworfen, wenn die TTL auf 0 sinkt.
- Protokoll-Nachrichten sind entweder Anfragen oder Antworten. Es gibt drei Anfrage/Antwort Typen:
  - Solche, die bei der Protokoll-Initialisierung eingesetzt werden (handshake). Diese Nachrichten werden niemals an dritte Knoten weitergesendet! Aus diesem Grund enthalten diese Nachrichten auch keine TTL-Zähler.
  - Solche, die an einen bestimmten Knoten gerichtet sind. Diese Nachrichten enthalten immer eine Ziel-Knoten-ID mit `FOR` und eine Absender-Knoten-ID mit `FROM`.
    Nachrichten dieses Typs werden vorläufig noch durch Flooding weitergeleitet. Später sollen sie mit Hilfe eines Routing-Verfahrens gezielt an einen Empfänger zugestellt werden.
  - Solche, die bis zu einer maximalen Tiefe alle Knoten des P2P-Netzes erreichen sollen, z.B. PING.
    Diese Nachrichten enthalten immer als Absender eine `FROM <Knoten-ID>` aber keine Empfänger-Spezifikation. Allerdings enthalten sie eine Request-ID. Diese Request-ID zusammen mit der Knoten-ID des Absenders muß eindeutig sein und kann dann beim Flooden zur Wiedererkennung bereits vorher empfangener Nachrichten benutzt werden.

Es folgt nun eine Spezifikation des Protokolls in einer BNF-ähnlichen Notation:

```
node-id ::=  <hostname>":"<local port>
_ ::=  <blank>
protoname ::=  "P2P"
version ::= "0.1"
proto ::= protoname"/"version
crlf ::= "\r\n"
return-code ::= [1245][0-9][0-9]
```

```
key ::= KEY _ <filename>
msgid ::= MESSAGE-ID _ <number>
size ::= SIZE _ <number>
ttl ::= TTL _ <number>

sender-addr ::= FROM _ node-id
target-addr ::= FOR _ node-id
addresspair ::= sender-addr _ target-addr

neighbour-list ::= node-id | node-id _ neighbour-list
setup-request  ::= HELLO _ NODE-ID _ node-id _ proto crlf |
               ::= HELLO _ NODE-ID _ node-id _ NEIGHBOURS _ neighbour_list _ proto crlf

search ::= SEARCH _ sender-addr _ key _ msgid _ ttl _ proto crlf
get ::= GET _ addresspair _ key _ msgid _ ttl _ proto crlf
put ::= PUT _ addresspair _ key _ size _ msgid _ ttl _ proto crlf
ping ::= PING _ sender-addr _ msgid _ ttl _ proto crlf

reply-head ::= proto
reply-tail ::= target-addr _ sender-addr _ msgid _ ttl

setup-reply ::= reply-head _ 200 _ node-id crlf |
                reply-head _ 200 _ node-id _ NEIGHBOURS _ neighbour-list crlf |
                reply-head _ 400 _ HANDSHAKE _ FAILURE crlf
                reply-head _ 402 _ ALREADY _ CONNECTED crlf

disconnect-request ::= DISCONNECT _ proto crlf
disconnect-reply   ::= reply-head _ 210 _ GOODBYE crlf

ping-reply ::= reply-head _ 230 _ PONG _ reply-tail crlf
search-reply ::= reply-head _ 220 _ FOUND _ reply-tail _ key crlf
get-reply ::= not-implemented
put-reply ::= not-implemented

not-implemented ::= reply-head _ 510 _ NOT _ IMPLEMENTED _ reply-tail _ key crlf
```

Beschreibungen für Protokoll-Nachrichten

Hier folgen nun für jeden Anfrage-Typ eine Beschreibung, wie Nachrichten zu versenden sind und wie auf den Empfang einer Nachricht zu reagieren ist.

Protokoll-Initialisierung Sobald durch einen Knoten A aktiv eine neue TCP-Verbindung aufgebaut worden ist, muß er sich bei seinem Gegenüber anmelden. Dies geschieht mit einer Anfrage, die wie folgt aussehen kann:

`HELLO NODE-ID viper:2000 NEIGHBOURS viper:3000 boa:2000 P2P/0.1`

Diese Nachricht besagt, dass Knoten A auf dem Rechner viper läuft und auf Port 2000 Verbindungen annimmt. Desweiteren nennt er die beiden Nachbarn `viper:3000` und `boa:2000`.

Sollte er noch keinen direkten Nachbarn haben, dann würde die Anfrage so aussehen:

`HELLO NODE-ID viper:2000 P2P/0.1`

Der Knoten, der die TCP-Verbindung angenommen hat, antwortet auf eine solche Anfrage mit:

`P2P/0.1 200 NODE-ID boa:3000 NEIGHBOURS boa:4000`

bzw.

`P2P/0.1 200 NODE-ID boa:3000`

Damit ist die Initialisierungsphase abgeschlossen und erst jetzt dürfen anderen Anfragen gestellt werden. Sollten vor Beendigung der Initialisierungsphase andere Anfragen oder Antworten empfangen werden, so ist mit

`P2P/0.1 400 HANDSHAKE FAILURE`

zu antworten und die TCP-Verbindung sofort abzubrechen.

Es ist außerdem nicht erlaubt zwei Knoten mehrfach zu verbinden, also eine Protokollsitzung zu starten, obwohl zwischen den beiden Knoten eine solche bereits besteht. In einem solchen Fall muss der Aufbau einer Protokollsitzung mit

`P2P/0.1 402 ALREADY CONNECTED`

abgebrochen werden.

Nachrichten die sich auf die Protokoll-Initialisierung beziehen, werden *niemals* and andere Knoten weitergeleitet.

Sitzungs-Abbau Sollte ein Knoten eine Sitzung von sich aus beenden wollen, so sendet er folgende Nachricht an seinen Nachbarn am anderen Ende der Verbindung:

`DISCONNECT P2P/0.1`

Der andere Knoten antwortet darauf mit

`P2P/0.1 210 GOODBYE`

und beide Knoten können jetzt die TCP-Verbindung schließen.

Nachrichten die sich auf einen Sitzungsabbau beziehen, werden *niemals* and andere Knoten weitergeleitet.

Such-Anfragen Eine Suche nach einer Datei `readme.txt` würde in etwa so aussehen:

`SEARCH FROM viper:2000 KEY readme.txt MESSAGE-ID 10 TTL 3 P2P/0.1`

Jeder Knoten, der eine solche Anfrage empfängt, muss die TTL um eins erniedrigen und falls sie dann immer noch größer 0 ist, mit der verringerten TTL flooden.

Außerdem kann er, sofern er eine Datei namens `readme.txt` kennt, *zusätzlich* folgendermaßen antworten:

`P2P/0.1 220 FOUND FOR viper:2000 FROM boa:3000 MESSAGE-ID 10 TTL 3 KEY readme.txt`

Dabei soll der `KEY` aus der `SEARCH`-Anfrage übernommen und eine neue `MESSAGE-ID` generiert werden.

Wenn eine solche Nachricht empfangen wird, ist sie unter Berücksichtigung der TTL an den Empfänger weiterzuleiten, es sei denn, man ist selbst `viper:2000`.

Download- und Upload-Anfragen Diese Anfragen könnten so

`GET FROM viper:2000 FOR boa:3000 KEY readme.txt MESSAGE-ID 11 TTL 3 P2P/0.1`

bzw. so

`PUT FROM viper:2000 FOR boa:3000 KEY readme.txt SIZE 428 MESSAGE-ID 12 P2P/0.1`

aussehen.

Wenn eine solche Nachricht empfangen wird, ist sie unter Berücksichtigung der TTL an den Empfänger weiterzuleiten, es sei denn, man ist selbst der Empfänger. In diesem Fall ist *vorläufig* zu antworten mit

`P2P/0.1 510 NOT IMPLEMENTED FOR viper:2000 FROM boa:3000 MESSAGE-ID 11 TTL 3 KEY readme.txt`

Ping-Anfragen Ping-Anfragen fordern eine Rückmeldung von allen Knoten innerhalb einer maximalen Tiefe an. Diese Rückmeldungen nennt man *Pong*. Ein Beispiel könnte so aussehen:

`PING FROM viper:2000 MESSAGE-ID 24 TTL 3 P2P/0.1`

Wenn eine solche Nachricht empfangen wird, ist sie unter Berücksichtigung der TTL zu flooden. Außerdem ist *zusätzlich* zu antworten mit:

`P2P/0.1 230 PONG FOR viper:2000 FROM boa:3000 MESSAGE-ID 24 TTL 3`

Wenn eine solche Pong-Nachricht empfangen wird, ist sie unter Berücksichtigung der TTL weiterzuleiten, es sei denn, man ist selbst `viper:2000`.

Aufgaben

Aufgabe 2: (70 Punkte) In dieser Aufgabe beginnen wir mit der Logik zum Verwalten von TCP-Verbindungen und einer vereinfachten Form des Aufbaus sowie dem kontrollierten Abbau von Protokollsitzungen. Alle anderen Teile werden in den folgenden Aufgaben nachgerüstet.

Für diese Aufgabe soll ein P2P-Knoten so weit implementiert werden dass der in der Lage ist TCP-Verbindungen zu anderen Knoten aufzubauen und auch selbst TCP-Verbindungen anzunehmen. Siehe dazu auch Aufgabe 1.

Desweiteren sollen die Teile des Protokolls implementiert werden, die für die Initialisierung und den Abbau einer P2P-Protokollsitzung zuständig sind, also der `HELLO`- und der `DISCONNECT`-Handshake (siehe Spezifikation und Beispiele oben). Beim `HELLO`-Handshake ist auf die Verwaltung der Nachbarn zu achten (neue Nachbarn aufnehmen, verschwundene Nachbarn löschen, Prüfung auf Mehrfachverbindung).

Für diese Aufgabe kann auf das Verwalten und Melden einer Peer-Liste (`NEIGHBOURS` in den Nachrichten) verzichtet werden. Diese Funktionalität wird später nachgerüstet werden.

Damit die Knoten-Software einigermaßen bequem zu bedienen ist, soll sie kurze Kommandos von der Tastatur lesen können. Solche Kommandos ***könnten*** sein:

connect <rechner> <port> Dieses Kommando könnte die Software anweisen eine Verbindung zum angegebenen Rechner und Port aufzubauen. Die Software führt dann selbstständig die Protokoll-Initialisierung durch.

disconnect <node-id> Weist die Software an, die Protokollsitzung zum genannten Knoten ordnungsgemäß zu beenden und die TCP-Verbindung zu schliessen.

list connections Weist die Software an, alle bestehenden Verbindungen zu Nachbarknoten zusammen mit der im Handshake gemeldeten Knoten-ID aufzulisten. Dies ist sehr hilfreich um die Funktionsfähigkeit der Software zu prüfen oder um Verbindungen referenzieren zu können.

Abzugeben sind:

- Dein Programm (sowohl Quelltext als auch ggf. compiliert im Falle von C, C++ und Java)
- Ggf. eine Beschreibung, was funktionieren sollte, dies leider aber immer noch nicht tut...

Details zur Abgabe der Aufgaben: siehe FAQ
(unterhalb `http://www.net.t-labs.tu-berlin.de/teaching/ws0708/PD_labcourse/`).
Abgabedatum: Dienstag, 27. November, 11:59h s.t.