

Übungsblatt 3

zur Vorlesung

Verteilte Systeme/Ubiquitous Computing

Wintersemester 2007/2008

Achtung: Bitte beachten Sie auch die Web-Site der Vorlesung:
<http://www.mobile.ifi.lmu.de/Vorlesungen/ss06/vs/>

Aufgabe 7: (H) Client/Server-Kommunikation über UDP

- a. Worin unterscheiden sich UDP und TCP? Wann wird welches Protokoll verwendet?
- b. Java stellt die Klassen DatagramPacket und DatagramSocket für die Implementierung einer Kommunikation über UDP zur Verfügung. Nutzen Sie diese, um folgende Aufgabe zu lösen: Die Uhr im Computer des A ist verstellt. Es bleibt ihm nichts anderes übrig, als bei einem Zeitserver die Zeit abzufragen und dann die Übertragungsdauer einzurechnen. Programmieren Sie einen hierfür geeigneten Client und den zugehörigen Zeitserver.

Antwort:

- a. Vergleich:

Transportschicht-Protokolle	User Datagram Protocol (UDP)	Transmission Control Protocol (TCP)
Verbindung	verbindungslos	verbindungsorientiert
Übermittlung	Messages, bestehend aus Datagramms	Streams, bestehend aus Segmenten
Zuverlässigkeit (Fehlerkorrektur, Bestätigungen, Regulierung des Datenflusses, und garantierte Reihenfolge)	Nein	Ja
typische Anwendungen	DNS, NFS, Voice over IP, Video Streaming, ...	SMTP, FTP, Telnet,...

Fazit: UDP kommt dann zum Einsatz, wenn es auf Geschwindigkeit (Header sind bei TCP deutlich größer!) ankommt und Sicherheit/Garantien entweder ohnehin gegeben oder nicht so wichtig sind im Vergleich zu anderen Gesichtspunkten.

- b. Erzeugen Sie mehrere Instanzen des Zeit-Servers auf unterschiedlichen Computern und starten Sie Ihren Klienten. Wie Sie sehen, ist die Synchronisation der Uhren in einem verteilten System ein reales Problem.

(i) Time-Client:

```
import java.net.*;
import java.io.*;

public class UDPTIMEClient
{
    public static void main(String[] args)
    {
        try
        {
            DatagramSocket sock = new DatagramSocket();
            InetAddress tServer;
            if (args.length == 0)
                tServer = InetAddress.getByName("localhost");
            else tServer = InetAddress.getByName(args[0]);
            int port = 1880;

            byte[] anfrage = "Zeit_bitte".getBytes();

            DatagramPacket request = new DatagramPacket(anfrage,
                                                         anfrage.length, tServer, port);

            sock.send(request);
            long sendTime = System.currentTimeMillis();

            byte[] buffer = new byte[1000];
            DatagramPacket reply = new DatagramPacket(buffer,
                                                         buffer.length);

            sock.receive(reply);
            long receiveTime = System.currentTimeMillis();

            sock.close();

            byte[] antwortData = reply.getData();

            String antwort = (new String(antwortData)).trim();

            long timeMillis = Long.parseLong(antwort);

            long now = timeMillis + ((receiveTime - sendTime)/2);

            System.out.println(now);

            java.util.Date lesbar = new java.util.Date();
            lesbar.setTime(now);
            System.out.println(lesbar);
        }
        catch (SocketException e)
        {
            System.err.println(e);
        }
    }
}
```

```

58         }
        catch (UnknownHostException e)
60         {
            System.err.println(e);
62         }
        catch (IOException e)
64         {
            System.err.println(e);
66         }
        }
68 }

```

(ii) Time-Server:

```

import java.net.*;
2 import java.io.*;

4 public class UDPTIMEserver
{
6     public static void main(String[] args)
    {
8         try
        {
10             DatagramSocket serverSock = new DatagramSocket(1880);

12             System.out.println("Server_läuft_und_läuft_und...");

14             byte[] buffer = new byte[1000];

16             while (true)
            {
18                 DatagramPacket request = new DatagramPacket(buffer,
                                                                buffer.length);

20                 serverSock.receive(request);
22                 long ZEIT = System.currentTimeMillis();
                byte[] zeit = (" " + ZEIT).getBytes();

24                 DatagramPacket reply = new DatagramPacket(zeit,
                                                                zeit.length, request.getAddress(), request.getPort());
                serverSock.send(reply);

28             }
        }
        catch (SocketException e)
        {
30             System.err.println(e);
        }
        catch (IOException e)
        {
34             System.err.println(e);
        }
38    }
}

```

Aufgabe 8: (H) Grundlagen des Naming

- Was ist der Unterschied zwischen Namen, Adressen und Identifiern?

- b. Wodurch unterscheidet sich ein absoluter Pfadname von einem relativen?
- c. Was ist ein Alias? Wie wird ein solcher implementiert?
- d. Was ist mounting? Was ist der Unterschied zwischen einem mount point und einem mounting point?

Antwort:

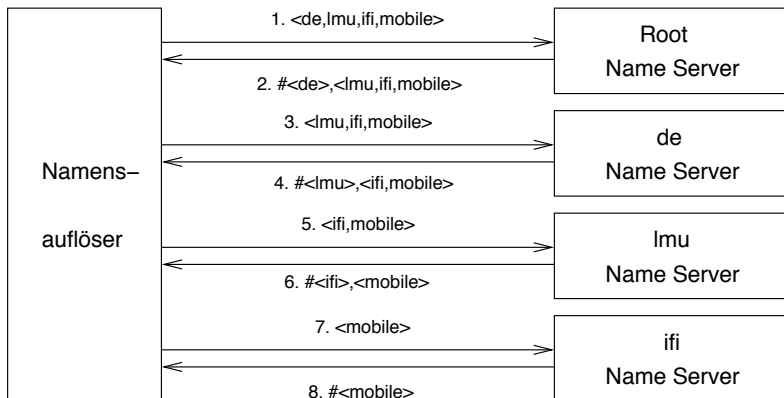
- a. Ein *Name* ist eine Abfolge von Zeichen, die für den Menschen lesbar ist und die eine Entität referenziert. Um auf eine Entität zugreifen zu können, benötigt man einen Zugriffspunkt. Dieser wird auch *Adresse* genannt und ist in maschinenlesbarer Form. Die Adresse wird nicht als Name der Entität verwendet, weil eine Entität mehr als einen Zugriffspunkt besitzen kann (somit also unklar wäre, welche Adresse als Name zu verwenden wäre) und weil bei einem Ortswechsel der Entität sich die Adressen verändern würden, der Name der Entität jedoch konstant bleiben soll. Ein *Identifier* ist ein Name mit den folgenden Eigenschaften:
 - (i) Ein Identifier bezeichnet höchstens eine Entität. Er ist also eindeutig.
 - (ii) Jede Entität wird von höchstens einem Identifier bezeichnet. Es gibt also keine zwei Identifier, die auf dieselbe Entität verweisen.
 - (iii) Ein Identifier zeigt immer auf dieselbe Entität. Ein Identifier wird somit niemals wiederverwendet.
- b. Wenn der erste Knoten eines Pfades die Wurzel des Namensbaums referenziert, dann handelt es sich um einen *absoluten* Pfad. Andernfalls wird der Pfad als *relativ* klassifiziert. Absolute Pfadangaben sind unabhängig von der aktuellen Position. Relativen Pfadangaben können je nach aktueller Position unterschiedlich interpretiert werden.
- c. Ein Alias ist ein zusätzlicher Name für eine Entität, die bereits einen Namen trägt. Ein Alias kann entweder über einen *Hard Link* oder *Symbolic Link* implementiert werden. Ein *Hard Link* liegt vor, wenn mehrere absolute Pfadnamen auf den selben Knoten in einem Namensgraphen verweisen. Bei einem *Symbolic Link* wird in dem Knoten, der den Namen des Aliases trägt, nicht die Adresse der referenzierten Entität gespeichert, sondern ein absoluter oder relativer Pfadname auf den eigentlichen Knoten im Namensgraphen.
- d. Mounting bezeichnet das transparente Verschmelzen zweier Namensräume. Die Verschmelzung erfolgt dadurch, dass ein Knoten (*mount point*) in dem dem System bereits bekannten Namensraum einen Identifier speichert, der auf den Wurzelknoten (*mounting point*) des einzubindenden Namensgraphen zeigt.

Aufgabe 9: (H) Domain Name System

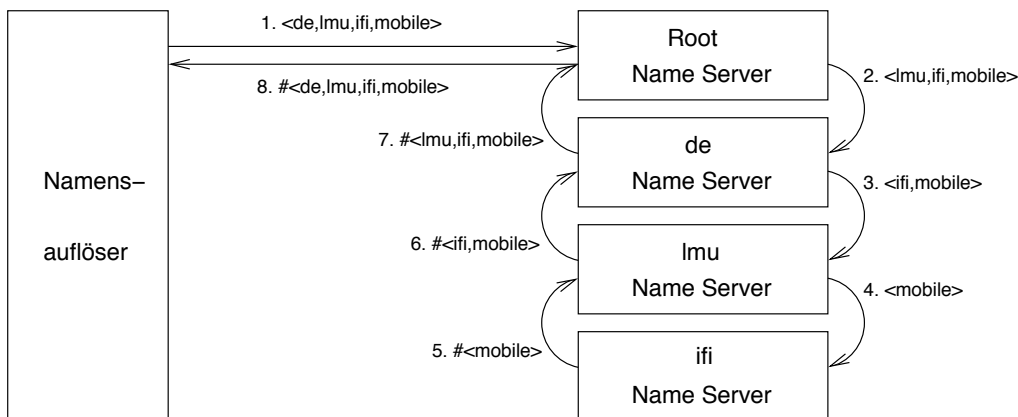
- a. Welche Vor- und Nachteile hat das rekursive Namensauflösungsverfahren gegenüber dem iterativen?
- b. Stellen Sie sich vor, Sie wollten mit dem Domain Name System Klientenanfragen zu jeweils nahe gelegenen Servern schicken. Was wäre zu tun?
- c. Stellen Sie sich vor, ein Name Server möchte die herein kommenden Anfragen auf n Server so verteilen, dass alle ungefähr die selbe Last zu bearbeiten haben. Was wäre zu tun? Worauf wäre zu achten?

Antwort:

Für das iterative Verfahren ergibt sich folgende Darstellung:



Die rekursive Namensauflösung läuft wie folgt ab:



a. Vorteile des rekursiven Verfahrens:

- (i) Bessere Caching-Ergebnisse: Im obigen Szenario ist es denkbar, dass bereits der Root-Name-Server die IP-Adresse des LMU-Servers kennt oder dass der DE-Name-Server die Namensauflösung bis zum Mobile-Server aufgrund gecachter Daten spontan durchführen kann.
- (ii) Eventuell geringere Kommunikationskosten: Falls der Namensauflöser sich auf einem anderen Kontinent befindet als die angefragten Name-Server, führt das rekursive Verfahren zu zwei interkontinentalen und sechs intrakontinentalen Anfragen bzw. Antworten, während die iterative Namensauflösung acht interkontinentale Datenaustausche veranlasst.

Nachteile des rekursiven Verfahrens:

- (i) Hohe Prozesslast für die Name-Server: Beim rekursiven Verfahren muss jeder Name-Server, der nicht am Ende der Auflösungskette steht, pro Auflösungsanfrage einen eigenen Prozess kreieren und verwalten, bis die Auflösung für ihn beendet ist. Gerade bei Name-Servern, die viele Anfragen erhalten, kann dies zu einer Überlasten führen. Dies führt dazu, dass Name-Server in der globalen Schicht in der Regel nur das iterative Verfahren unterstützen.
- b. Wenn ein Klient eine URL auflösen möchte, so schickt er diese an den Namensauflöser. Seinen Namensauflöser kennt er aufgrund manueller Konfiguration oder aufgrund eines Autokonfigurationspro-

tolks wie dem Dynamic Host Configuration Protocol (DHCP). Der Namensauflöser liegt in der Regel für Firmen im Firmennetz und für Endkunden beim Internet Service Provider.

Der Namensauflöser gibt seine IP-Adresse an, wenn er sich an einen anderen DNS-Server wendet. Einer der hierarchisch hohen DNS-Server muss nun fähig sein, aufgrund der IP-Adresse des Namensauflösers zu bestimmen, aus welcher Internetregion die zugehörige Anfrage stammt. Hierfür muss der DNS-Server Tabellen besitzen, die jeder IP-Adresse eine Internetregion zuordnen und für jede Internetregion den nächstgelegenen Server angeben.

Für die Erstellung solcher Karten ist ein aufwendiger Monitoring Prozess notwendig. Akamai geht hierfür wie folgt vor: Zunächst werden so genannte Core Points ermittelt. Core Points repräsentieren eine Menge von Namensauflösern. Core Points werden bestimmt, indem die Server einen inkrementellen Trace Route auf die Namensauflöser ausführen. Dort wo sich die Spuren (Traces) zum ersten mal kreuzen für eine Menge von Namensauflösern, wird ein Core Point gesetzt.

Alle Server ermitteln durch Pings regelmäßig die Round-Trip-Time zwischen ihnen und den Core Points. Mit Hilfe dieser Daten werden Karten erstellt, die den Namensauflösern, die sich hinter einem Core Point verbergen, den am schnellsten zu erreichenden Server zuordnen.

Diese Karten werden den hierarchisch hohen DNS-Servern zur Verfügung gestellt.

- c. Eine Möglichkeit wäre, dass der Name Server alle n Server durchnummeriert und bei der Verteilung ein Round-Robin Verfahren anwendet. Das heißt, Server 1 erhält die erste, die $1 + n$ -te, die $1 + 2n$ -te Anfrage usw.. Server 2 bearbeitet die zweite, die $2 + n$ -te, die $2 + 2n$ -te Anfrage usw.. Nachteil dieses Verfahrens ist, dass alle Server alle Dokumente speichern müssen.

Alternativ kann die URL d des Dokuments mittels einer Hash-Funktion verarbeitet werden. Eine denkbare Hashfunktion wäre $f(d) = ((ad + b) \bmod(n))$. Wenn $f(d) = 1$, so wird die Anfrage Server 1 zugeordnet. Wenn $f(d) = 2$, so muss Server 2 die Anfrage bearbeiten etc.. Auf diese Weise ist sicher gestellt, dass immer derselbe Server die Anfrage nach einem bestimmten Dokument erhält. Innerhalb der n Server muss also jedes Dokument nur einmal gespeichert werden. Leider hat dieses Verfahren ein Problem: Wenn sich die Anzahl der Server ändert, müssen fast alle Dokumente umgelagert werden, da dann z.B. die Hashfunktion $f(d) = ((ad + b) \bmod(n + 1))$ gilt.

Deswegen haben die Akamai-Gründer "Consistent Hashing" entwickelt. Hierbei werden sowohl die Server als auch die URLs in einen mathematischen Raum (mathematical mapping space) projiziert. Der mathematische Raum ist ein Kreis mit dem Radius 1. URLs werden dem Server zugeordnet, der entlang dem Uhrzeigersinn auf dem Kreis der nächste ist. Wenn nun einer Server zum System hinzugefügt wird, sind nur wenige Dokumente neu zu verteilen. Sagen wir, der neue Server 1 liegt auf dem Kreis zwischen Server 2 und 3. Geht man von Server 1 entlang dem Uhrzeigersinn gelangt man zu Server 2, gegen den Uhrzeigersinn liegt Server 3. Vor dem Einfügen von Server 1 speicherte Server 2 alle Dokumente zu den URLs zwischen Server 2 und 3. Nach dem Einfügen von Server 1 übernimmt Server 1 alle Dokumente von Server 2, die URLs besitzen, die nun auf dem Kreis zwischen Server 1 und Server 3 liegen.

Aufgabe 10: (T) DNS-Erweiterung DNSSEC

- a. Welche Sicherheitsprobleme existieren im klassischen DNS-Konzept?
- b. Im März 2005 verabschiedete die IETF die DNS-Erweiterung DNSSEC als Standard. Siehe auch:
- RFC 4033 - DNS Security Introduction and Requirements
 - RFC 4034 - Resource Records for the DNS Security Extensions
 - RFC 4035 - Protocol Modifications for the DNS Security Extensions

Welche der in Aufgabe a) ermittelten Sicherheitsprobleme werden von DNSSEC gelöst?

- c. Welche Sicherheitsprobleme werden von DNSSEC zum aktuellen Zeitpunkt noch nicht gelöst?

Antwort:

Siehe Folien.