3 Netzwerkprogrammierung in Java

- In package `java.net`

3.1 Internet-Adressen (IP-Adressen)

- Internet-Adresse = vier Oktette (je 8 Bit)
- jedes direkt mit dem Internet verbundene Endgerät besitzt eindeutige Internet-Adresse
- maximal $2^{32} = 4.294.967.296$ Endgeräte
  (überhöht, da Adressraum strukturiert und teilweise reserviert)

0nnnnnnn. H. H. H	class A Netzwerk
10nnnnnn. N. H. H	class B Netzwerk
110nnnnn. N. N. H	class C Netzwerk
1110nnnn. . .	class D Netzwerk (Multicast)
1111nnnn. . .	class E Netzwerk (Experimentell)

Beispiele

132.230. 1. 5	WWW-Server der Uni Freiburg
132.230.150.17	WWW-Server der Informatik
127. 0. 0. 1	localhost (eigener Rechner, für Experimente)

Zukünftige IP-Adressen: IPv6 [RFC 2060]

- Befürchtung: IPv4 Adressraum bald erschöpft
- daher: 128bit IP-Adressen [RFC 2373]
- viele Konzepte eingebaut bzw vorgesehen
  - selbständige Adresskonfiguration (mobiler Zugang)
  - *quality of service* Garantieen möglich
  - Authentisierung, Datenintegrität, Vertraulichkeit
- Schreibweise: 4er Gruppen von Hexziffern

```
1080:0:0:0:8:800:200C:417A  a unicast address
1080::8:800:200C:417A       ... compressed
```

3.2 Die Klasse `java.net.InetAddress`

- Objekte repräsentieren IP-Adressen
- Subklassen für IPv4 und IPv6
- kein öffentlicher Konstruktor, stattdessen

```
public static InetAddress[]
  getAllByName(String host)
    throws UnknownHostException
```

sämtliche IP-Adressen von `host`

```
public String
  getHostAddress()
```

liefert die IP-Adresse als Text

```
public static InetAddress
  getLocalHost()
    throws UnknownHostException
```

IP-Adresse des lokalen Rechners

Beispiel: DomainName2IPNumbers

```
import java.net.*;

public class DomainName2IPNumbers {
  public static void main(String[] args) {
    try {
      InetAddress[] a = InetAddress.getAllByName(args[0]);
      for (int i = 0; i<a.length; i++)
        System.out.println(a[i].getHostAddress());
    } catch (UnknownHostException e) {
      System.out.println("Unknown host!");
    }
  }
}

/* > java DomainName2IPNumbers www.google.com
216.239.59.147
216.239.59.104
216.239.59.99
*/
```

Beispiel: Eigene Adresse

```
import java.net.*;

public class MyAddress {
  public static void main(String[] args) {
    try {
      InetAddress a = InetAddress.getLocalHost();
      System.out.println("domain name: "+a.getHostName());
      System.out.println("IP address:  "+a.getHostAddress());
    } catch (UnknownHostException e) {
      System.out.println("Help! I don't know who I am!");
    }
  }
}

/* > java MyAddress
domain name: abacus.informatik.uni-freiburg.de
IP address:  132.230.166.150
*/
```

3.3 Sockets

Ein Socket (Steckdose) ist eine Datenstruktur zur Administration von (Netzwerk-) Verbindungen. An jedem Ende einer Verbindung ist ein Socket erforderlich. Es gibt sie in mehreren Dimensionen:

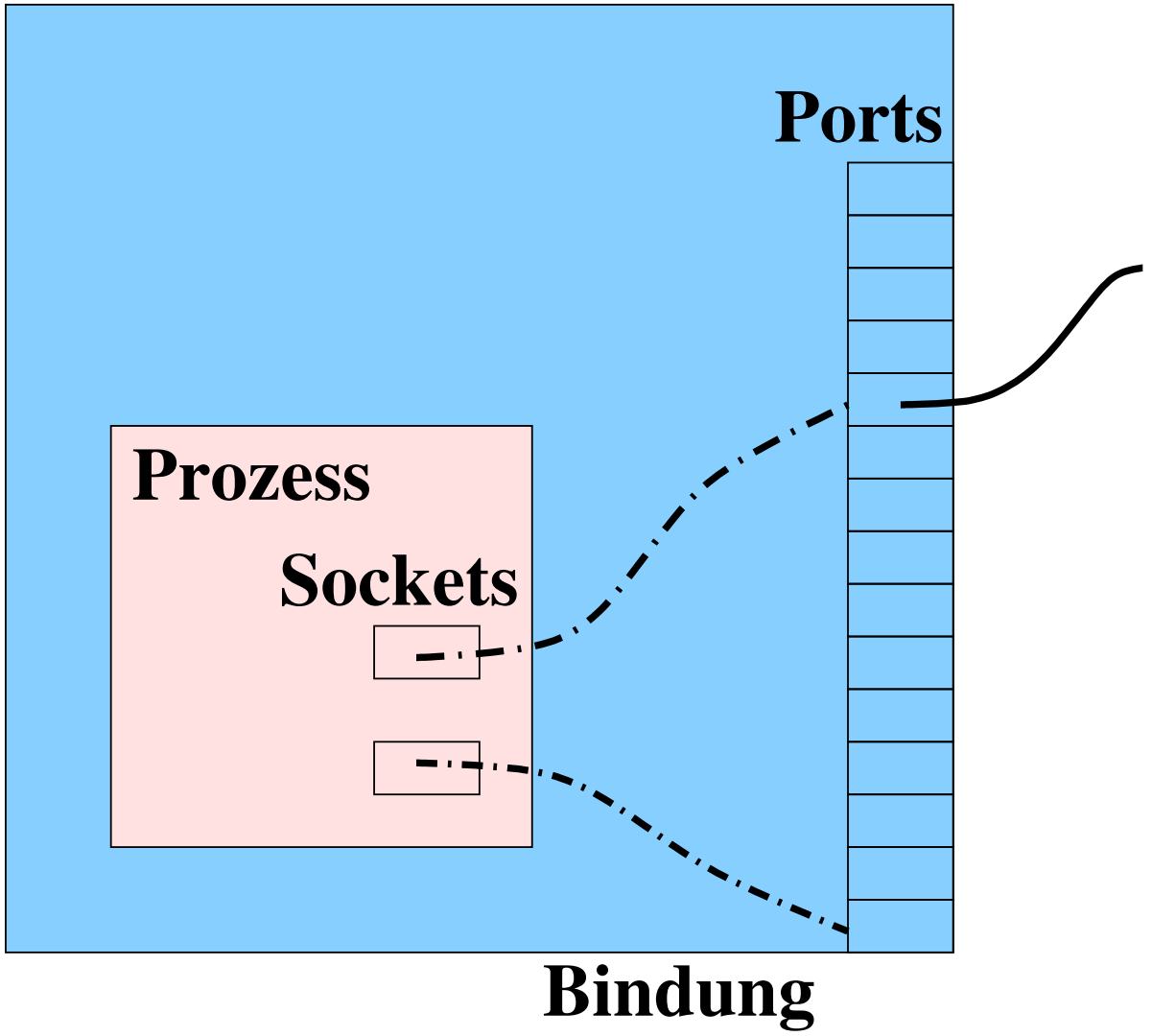
Aktivität

- Client Socket:
  Verbindung mit existierendem Dienst
- Server Socket:
  Stellt Dienst zur Verfügung

Verbindungsart

- UDP (Datagram, unidirektional)
- TCP (Stream, bidirektional)

Sockets und Ports



3.3.1 Klasse `java.net.Socket` für Clients

Socket Konstruktoren

```
Socket (InetAddress address, int port)
```

Verbindung zum Server auf `address` und `port`

```
Socket (String host, int port) {
  Socket (InetAddress.getByName (host), port);
}
```

Verbindung zum Server `host` und `port`

→ auch ein Client Socket ist auf dem lokalen Rechner an einen (meist beliebigen) Port gebunden

Socket Methoden

- `OutputStream getOutputStream() throws IOException`
  Ausgabe auf diesem Strom wird zum Server gesendet (Anfragen an den Server)
- `InputStream getInputStream() throws IOException`
  Eingaben von diesem Stream stammen vom Server (Antworten des Servers)
- `void close() throws IOException`
  Schließen des Sockets

3.3.2 Beispiel

```
class HTTPGet {
    public static void main (String[] args) throws Exception {
        if (args.length != 2) {
            System.out.println ("Usage: java HTTPGet host path");
        } else {
            String hostname = args[0];
            String path = args[1];
            Socket s = new Socket (hostname, 80);
            PrintWriter out = new PrintWriter (s.getOutputStream (), true);
            // send request
            out.print ("GET "+path+" HTTP/1.1\r\n");
            out.print ("Host: "+hostname+"\r\n");
            out.print ("\r\n");
            // read & echo response
            System.out.println ("-------------------------------------");
            in = new BufferedReader (new InputStreamReader (s.getInputStream ()));
            String line = in.readLine ();
            while (line != null) {
                System.out.println (line);
                line = in.readLine ();
            }
            // may hang for a while
            System.out.println ("-------------------------------------");
        }
    }
}
```

3.3.3 Klasse `java.net.ServerSocket`

Konstruktoren

```
ServerSocket (int port) throws IOException
```

Erzeugt einen Socket für Verbindungen über `port`. Dient nur zum Verbindungsaufbau.

Wichtige Methoden

```
Socket accept() throws IOException
```

Wartet am `port` des `ServerSocket` auf eine (externe) Verbindung. Liefert einen gewöhnlichen `Socket` für die Abwicklung der Kommunikation.

```
void close() throws IOException
```

Schließt den `ServerSocket`

3.3.4 Beispielserver

Das Interface `DialogHandler` trennt die Handhabung der Verbindung von der Abwicklung der Kommunikation.

```
import java.io.*;

public interface DialogHandler {
  // @return false to exit the server loop
  boolean talk (BufferedReader br, PrintWriter pw);
}
```

Beispiel — Implementierung

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

public class TCPServer {
    ServerSocket ss;

    public TCPServer (int port)
        throws IOException {
        ss = new ServerSocket (port);
    }

    public void run (DialogHandler dh)
        throws IOException {
        boolean acceptingConnections = true;
        while (acceptingConnections) {
            Socket s = ss.accept ();
            BufferedReader br = new BufferedReader
                (new InputStreamReader (s.getInputStream ()));
            PrintWriter pw = new PrintWriter (s.getOutputStream (), true);
            acceptingConnections = dh.talk (br, pw);
            s.close ();
        }
    }
}
```

DialogHandler für BackTalk

```
public class BackTalkDialog
    implements DialogHandler {

    public boolean talk (BufferedReader br, PrintWriter pw) {
        String line = null;
        BufferedReader terminal = new BufferedReader
            (new InputStreamReader (System.in));
        while (true) {
            try {
                if (br.ready ()) {
                    line = br.readLine ();
                    System.out.println (line);
                } else if (terminal.ready ()) {
                    line = terminal.readLine ();
                    if (line.equals ("STOP!")) {
                        break;
                    }
                    pw.println (line);
                }
            } catch (IOException ioe) {
                return false;
            }
        }
        return false;           // stop the server
    }
}
```

Beispiel — ein handbetriebener Server

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

public class BackTalk {

    public static void main (String[] arg) throws Exception {
        if (arg.length != 1) {
            System.out.println ("Usage: BackTalk port");
        } else {
            try {
                int port = new Integer (arg[0]).intValue ();
                TCPServer server = new TCPServer (port);
                server.run (new BackTalkDialog ());
            } catch (RuntimeException e) {
                System.out.println ("Argument not an integer");
            }
        }
    }
}
```

3.4 Verbindungen über URLs

URL (Uniform Resource Locator) RFC 1738, RFC 1808, RFC 2368

Symbolische Adresse für ein Dokument

Format: ⟨*Schema*⟩:⟨*schemaspezifische Information*⟩

Mögliche Schemata und schemaspezifische Informationen

`mailto:` Internet-Mailadresse

Beispiel: `mailto:president@whitehouse.gov`

`http:` //⟨*User*⟩:⟨*Password*⟩@⟨*Host*⟩:⟨*Port*⟩/⟨*URL-Path*⟩

Dabei sind optional

- ⟨*User*⟩:⟨*Password*⟩@
- :⟨*Port*⟩

Beispiel: `http://www.informatik.uni-freiburg.de/proglang`

`ftp:` //⟨*User*⟩:⟨*Password*⟩@⟨*Host*⟩:⟨*Port*⟩/⟨*Path*⟩

Auch hier sind ⟨*User*⟩, ⟨*Password*⟩ und ⟨*Port*⟩ Informationen optional.

Beispiel: `ftp://ftp.informatik.uni-freiburg.de/iif`

`news:` Newsgruppe

Nicht alle Zeichen sind im schemaspezifischen Anteil einer URL erlaubt (vgl. RFC), sie werden *URL-kodiert* durch %⟨*hexdigit*⟩⟨*hexdigit*⟩, die Hexadezimaldarstellung der Nummer des Zeichens. (Siehe Klasse `java.net.URLEncoder`.)

3.4.1 Klasse `URL`

Wichtige Konstruktoren

```
URL(String spec) throws MalformedURLException
```

parst den String spec und —falls erfolgreich— erstellt ein `URL` Objekt.

Wichtige Methoden

```
URLConnection openConnection() throws IOException
```

liefert ein Objekt, über das

1. die Parameter der Verbindung gesetzt werden
2. die Verbindung hergestellt wird
3. die Verbindung abgewickelt wird

3.4.2 Klasse `URLConnection`

abstrakte Klasse, daher keine Konstruktoren

Wichtige Methoden

- Methoden zum Setzen von Anfrageparametern (Request-Header für HTTP):
  `setUseCaches`, `setIfModifiedSince`, `setRequestProperty`, ...
- `void connect()`
  Herstellen der Verbindung
- Methoden zum Abfragen von Antwortparametern (Response-Header für HTTP):
  `getContentEncoding`, `getContentLength`, `getHeaderField`, ...
- `InputStream getInputStream()`
  zum Lesen von der Verbindung
- `Object getContent ()`
  zum Parsen von der Verbindung in ein passendes Objekt
  kann selbst bestimmt werden: `setContentHandlerFactory`

3.4.3 Klasse `HttpURLConnection extends URLConnection`

abstrakte Klasse, daher keine Konstruktoren

Wichtige Methoden

- Setzen von HTTP-spezifischen Anfrageparametern
  - `static void setFollowRedirects(boolean set)` Standardwert: `true`
  - `void setRequestMethod(String method)` (`method` ist `GET`, `HEAD`, `POST`, ...)
- Abfagen von HTTP-spezifischen Antwortparametern
  - `int getResponseCode()`
  - `String getResponseMessage()`
  - `InputStream getErrorStream ()`

Beispiel: Inhalt eines Dokuments als `byte[]`

```
public class RawURLContent {

    private URLConnection uc;

    public RawURLContent (URL u)
        throws IOException {
        uc = u.openConnection ();
    }

    public byte[] getContent ()
        throws IOException {
            int len = uc.getContentLength ();
            if (len <= 0) {
                System.err.println ("Length cannot be determined");
                return new byte[0];
            } else {
                byte[] rawContent = new byte [len];
                uc.getInputStream ().read (rawContent);
                return rawContent;
            }
        }
}
```

Beispiel: I'm Feeling Lucky

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

public class ImFeelingLucky2 {
  public static void main(String[] args) {
    try {
      String req = "http://www.google.com/search?"+
        "q="+URLEncoder.encode(args[0], "UTF8")+"&"+
        "btnI="+URLEncoder.encode("I'm Feeling Lucky", "UTF8");

      HttpURLConnection con = (HttpURLConnection) (new URL(req)).openConnection();
      con.setRequestProperty("User-Agent", "IXWT");
      con.setInstanceFollowRedirects(false);

      String loc = con.getHeaderField("Location");
      if (loc!=null)
        System.out.println("Direct your browser to "+loc);
      else
        System.out.println("I am sorry - my crystal ball is blank.");

    } catch (IOException e) {
      System.err.println(e);
    }
  }
}
```

3.5 SSL Verbindungen

- JSSE (Java Secure Socket Extension)
- `import javax.net.ssl.*`
- Wesentliche Änderung im **Client** Programm: (Factory Pattern)

  Ersetze

```
Socket s = new Socket (hostname, portnumber)
```

  durch

```
SSLSocketFactory sf = (SSLSocketFactory)SSLSocketFactory.getDefault();
SSLSocket s = (SSLSocket)sf.createSocket(hostname, portnumber);
```

- Dafür muss SSL konfiguriert sein (siehe unten).

SSL Server Sockets

- Wesentliche Änderung im **Server** Programm:

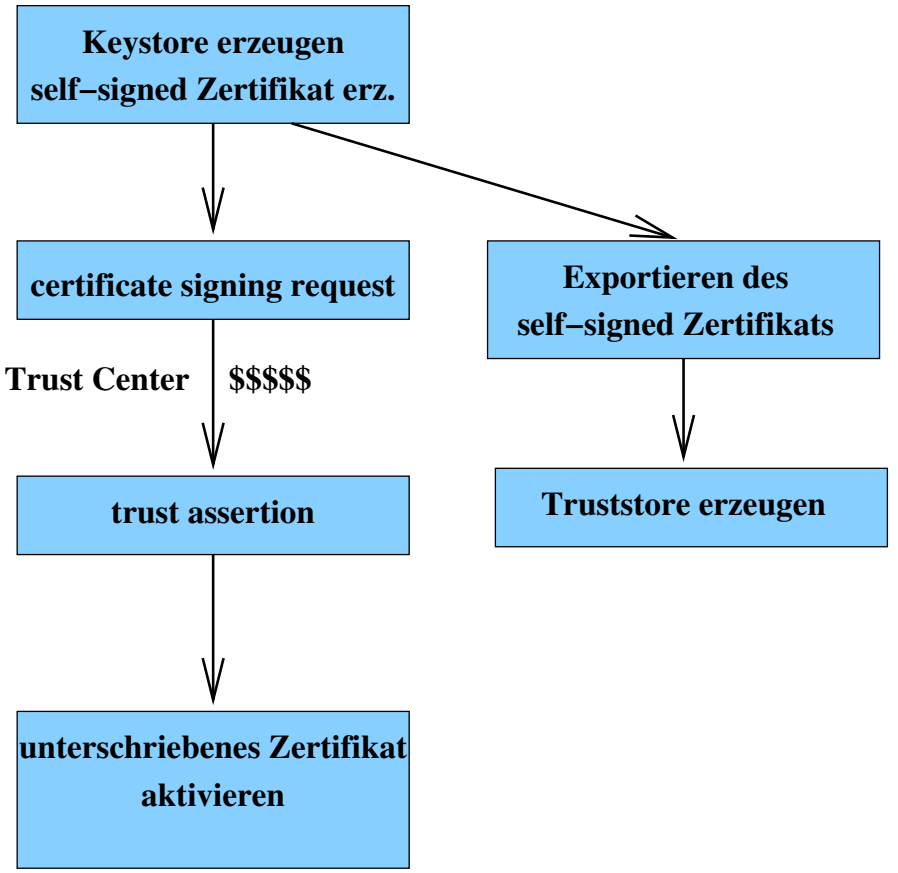
  Ersetze

```
ServerSocket ss = new ServerSocket (portnumber)
// ...
Socket con = ss.accept ()
```

  durch

```
SSLSocketFactory sf = (SSLSocketFactory)SSLSocketFactory.getDefault();
SSLServerSocket ss = (SSLServerSocket)sf.createServerSocket(portnumber);
// ...
SSLSocket con = (SSLSocket)ss.accept();
```

Konfiguration von SSL: Zertifikate



Erzeugen des Key Store

- Schlüsselpaar, Self-Signed Zertifikat

```
> keytool -genkey -alias widgetorg -keyalg RSA -validity 7 -keystore serverkey.jks
Enter keystore password:  ToPsEcReT
What is your first and last name?
  [Unknown]:  www.widget.inc
What is the name of your organizational unit?
  [Unknown]:  Web Division
What is the name of your organization?
  [Unknown]:  Widget Inc.
What is the name of your City or Locality?
  [Unknown]:  Punxsutawney
What is the name of your State or Province?
  [Unknown]:  Pennsylvania
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=www.widget.inc, OU=Web Division, O=Widget Inc., L=Punxsutawney, ST=Pennsylvania, C=US correct?
  [no]:  yes

Enter key password for <widgetorg>
(RETURN if same as keystore password):
>
```

Certificate Signing Request

```
> keytool -certreq -alias widgetorg -keystore serverkey.jks -keyalg RSA -file widgetorg.csr
Enter keystore password:  ToPsEcReT
```

- An certificate authority
- Zurück: unterschriebenes Zertifikat

Trust Assertion

```
> keytool -import -alias root -keystore serverkey.jks -trustcacerts -file ca.cer
Enter keystore password: ToPsEcReT
Certificate was added to keystore
```

- Dieser Schritt kann übersprungen werden, falls certificate authority bereits bekannt

Zertifikat aktivieren

```
> keytool -import -alias widgetorg -keystore serverkey.jks -file widgetorg.cer
Enter keystore password: ToPsEcReT
Certificate was added to keystore
```

Zertifikat verwenden

```
> java -Djavax.net.ssl.keyStore=serverkey.jks
       -Djavax.net.ssl.keyStorePassword=ToPsEcReT
       MySecureServer 8443
```

Alternative: self-signed Zertifikat

- Exportiere das self-signed Zertifikat nach Konstruktion des key stores:

```
> keytool -export -keystore serverkey.jks -alias widgetorg -file widgetorgself.cer
Enter keystore password:  ToPsEcReT
Certificate stored in file <widgetorgself.cer>
```

- Erzeuge dann einen *trust store*, der alle vertrauenswürdigen Schlüssel enthält.

```
> keytool -import -alias widgetorg -file widgetorgself.cer -keystore truststore.jks
Enter keystore password:  HuShHuSh
Owner: CN=www.widget.inc, OU=Web Division, O=Widget Inc., L=Punxsutawney, ST=Pennsylvania, C=US
Issuer: CN=www.widget.inc, OU=Web Division, O=Widget Inc., L=Punxsutawney, ST=Pennsylvania, C=US
Serial number: 42665235
Valid from: Wed Apr 20 14:59:33 CEST 2005 until: Wed Apr 27 14:59:33 CEST 2005
Certificate fingerprints:
 MD5:  4D:A7:09:CB:1A:8E:5F:91:5E:7A:2F:F1:CD:16:B6:4F
 SHA1: C6:90:05:6D:1D:B8:B1:5D:C9:83:BF:9F:79:2C:FD:28:54:58:B9:D6
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

- Verwendung in einem Client mit

```
> java -Djavax.net.ssl.trustStore=truststore.jks
       -Djavax.net.ssl.trustStorePassword=HuShHuSh
       MySecureClient 8443
```

3.6 UDP Sockets

- Wichtig: UDP Ports $\neq$ TCP Ports
- Java API: Zwei Klassen
  - `DatagramPacket` repräsentiert ein Datenpaket (zum Versenden oder nach dem Empfang)
  - `DatagramSocket` repräsentiert die eigentliche Verbindung

Klasse `java.net.DatagramPacket`

nur Aufbau von Datenstruktur, keine Verbindung!

Wichtige Konstruktoren

- ```
  DatagramPacket(byte[] buf, int length)
  ```
  zum Empfang von `length` Bytes in `buf`
- ```
  DatagramPacket(byte[] buf, int length, InetAddress address, int port)
  ```
  vorbereitet zum Versenden von `length` Bytes aus `buf` an `address` und `port`
  Beachte: die Adresse des Ziels befindet sich im Paket!

Wichtige Methoden `java.nat.DatagramPacket`

Empfangen

- `byte[] getData()`
- `int getLength()`
- `InetAddress getAddress()`
- `int getPort()`

Senden

- `void setData(byte[] buf)`
- `void setLength(int length)`
- `void setAddress(InetAddress iaddr)`
- `void setPort(int iport)`

Klasse `java.net.DatagramSocket`

Wichtige Konstruktoren

- `DatagramSocket()`
- `DatagramSocket(int port)`

Wichtige Methoden

- `void send(DatagramPacket p) throws IOException`
- `void receive(DatagramPacket p) throws IOException`
- `void close()`

Ablauf

Senden

```
s = new DatagramSocket ();
p = new DatagramPacket (b,l);
p.setAddress (...);
p.setPort (...);
p.setData (...);
s.send (p);
```

Empfangen

```
s = new DatagramSocket (myport);
p = new DatagramPacket (b,l);
s.receive (p);
result = p.getData ();
sender = p.getAddress ();
seport = p.getPort ();
```

Beispiel — ein Client für `daytime` RFC 867

```
public class Daytime {
    static final int BUFSIZE = 128;
    static final int DAYTIME = 13;      // portnumber of daytime service
    //...
    public static String getTime (String hostname)
        throws Exception {
        byte[] buffer = new byte[BUFSIZE];
        InetAddress server = InetAddress.getByName (hostname);
        DatagramPacket answer = new DatagramPacket (buffer, BUFSIZE);
        DatagramSocket s = new DatagramSocket ();
        answer.setAddress (server);
        answer.setPort (DAYTIME);
        s.send (answer);        // contents do not matter
        s.receive (answer);
        s.close ();
        int len = answer.getLength ();
        buffer = answer.getData ();
        while (buffer[len-1] == 10 || buffer[len-1] == 13) {
            len--;
        }
        return new String (buffer, 0, len);
    }
}
```

Beispiel — ein Server für daytime RFC 867

```
public class DaytimeServer {

    static final int BUFSIZE = 128;
    static final int DAYTIME = 13;      // portnumber for daytime service
    // ...
    public static void serveTime (int port)
        throws Exception {
        byte[] buffer = new byte[BUFSIZE];
        DatagramPacket p = new DatagramPacket (buffer, BUFSIZE);
        DatagramSocket s = new DatagramSocket (port);
        // while (true) {
        s.receive (p);          // contents do not matter
        Date d = new GregorianCalendar ().getTime ();
        System.out.println ("Sending: " + d);
        String answer = d.toString ();
        p.setData ((answer + "\r\n").getBytes ());
        p.setLength (answer.length () + 2);
        s.send (p);
        // }
        s.close ();
    }
}
```

3.7 UDP vs. TCP

Application	Application-layer protocol	Underlying Transport Protocol
electronic mail	SMTP	TCP
remote terminal access	Telnet	TCP
Web	HTTP	TCP
file transfer	FTP	TCP
remote file server	NFS	typically UDP
streaming multimedia	proprietary	typically UDP
Internet telephony	proprietary	typically UDP
Network Management	SNMP	typically UDP
Routing Protocol	RIP	typically UDP
Name Translation	DNS	typically UDP

3.8 DNS, ein Paket-Protokoll

Hintergrund: RFC 1034. Technische Beschreibung: RFC 1035

DNS: Abbildung von *Domainnamen* auf *Resource Records* (RR)

Ein Domainname ist

- Folge von Strings (Labels), getrennt durch und beendet mit "."
- Maximale Länge eines Labels: 63
- Maximale Länge eines Domainnamen: 255 (inkl. der Punkte)

Menge der Domainnamen ist Hierarchie mit Wurzel "."

```
                                                .
                                              de.
                                    uni-freiburg.de.
                          informatik.uni-freiburg.de.
```

Typen von Resource Records (Ausschnitt):

`A`	host address
`NS`	authoritative name server
`CNAME`	canonical name for an alias
`SOA`	zone of authority
`PTR`	domain name pointer
`MX`	mail exchanger

Grundidee

DNS ist verteilte Datenbank, in der jeder Server zuständig (authoritativ) für eine bestimmte Domain ist.

- Abfrage der Datenbank: UDP Nachricht an *beliebigen* Server.
- Abgleich zwischen den Servern: TCP Verbindungen.

3.8.1 Beispielsitzung

`nslookup` ist ein textuelles Werkzeug für DNS-Anfragen, kontaktiert Port `domain` (53) mit UDP

```
shell> /usr/sbin/nslookup -
Default Server:  atlas.informatik.uni-freiburg.de
Address:  132.230.150.3
```

Alle folgenden Fragen beziehen sich auf Address RRs:

```
> set q=a
```

```
> www.informatik.uni-freiburg.de.
Server:  atlas.informatik.uni-freiburg.de
Address:  132.230.150.3

Name:    falcon.informatik.uni-freiburg.de
Address:  132.230.167.230
Aliases:  www.informatik.uni-freiburg.de
```


Frage nach Nameserver RRs:

```
unix> nslookup -
> set q=ns
> informatik.uni-freiburg.de.
Server:  atlas.informatik.uni-freiburg.de
Address:  132.230.150.3

informatik.uni-freiburg.de      nameserver = dns1.fun.uni-freiburg.de
informatik.uni-freiburg.de      nameserver = tolkien.imtek.uni-freiburg.de
informatik.uni-freiburg.de      nameserver = atlas.informatik.uni-freiburg.de
informatik.uni-freiburg.de      nameserver = dns0.fun.uni-freiburg.de
dns1.fun.uni-freiburg.de        internet address = 132.230.200.201
tolkien.imtek.uni-freiburg.de   internet address = 132.230.168.1
atlas.informatik.uni-freiburg.de        internet address = 132.230.150.3
dns0.fun.uni-freiburg.de        internet address = 132.230.200.200
```

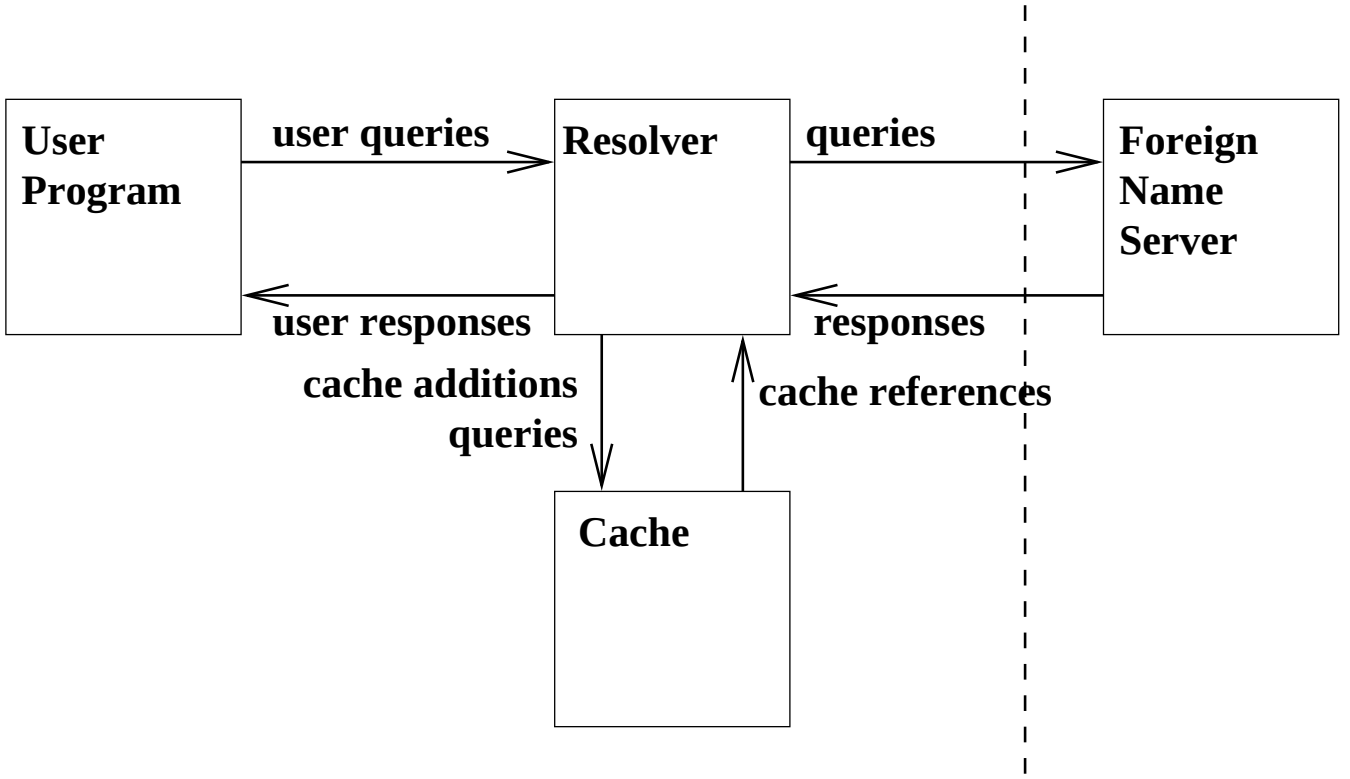
Für Deutschland:

```
> de.
Server:  atlas.informatik.uni-freiburg.de
Address:  132.230.150.3

Non-authoritative answer:
de      nameserver = s.de.net.
de      nameserver = z.nic.de.
de      nameserver = a.nic.de.
de      nameserver = c.de.net.
de      nameserver = f.nic.de.
de      nameserver = l.de.net.

Authoritative answers can be found from:
s.de.net        internet address = 193.159.170.149
z.nic.de        has AAAA address 2001:628:453:4905::53
z.nic.de        internet address = 194.246.96.1
a.nic.de        internet address = 193.0.7.3
c.de.net        internet address = 208.48.81.43
f.nic.de        internet address = 81.91.161.4
f.nic.de        has AAAA address 2001:608:6::5
l.de.net        internet address = 217.51.137.213
```

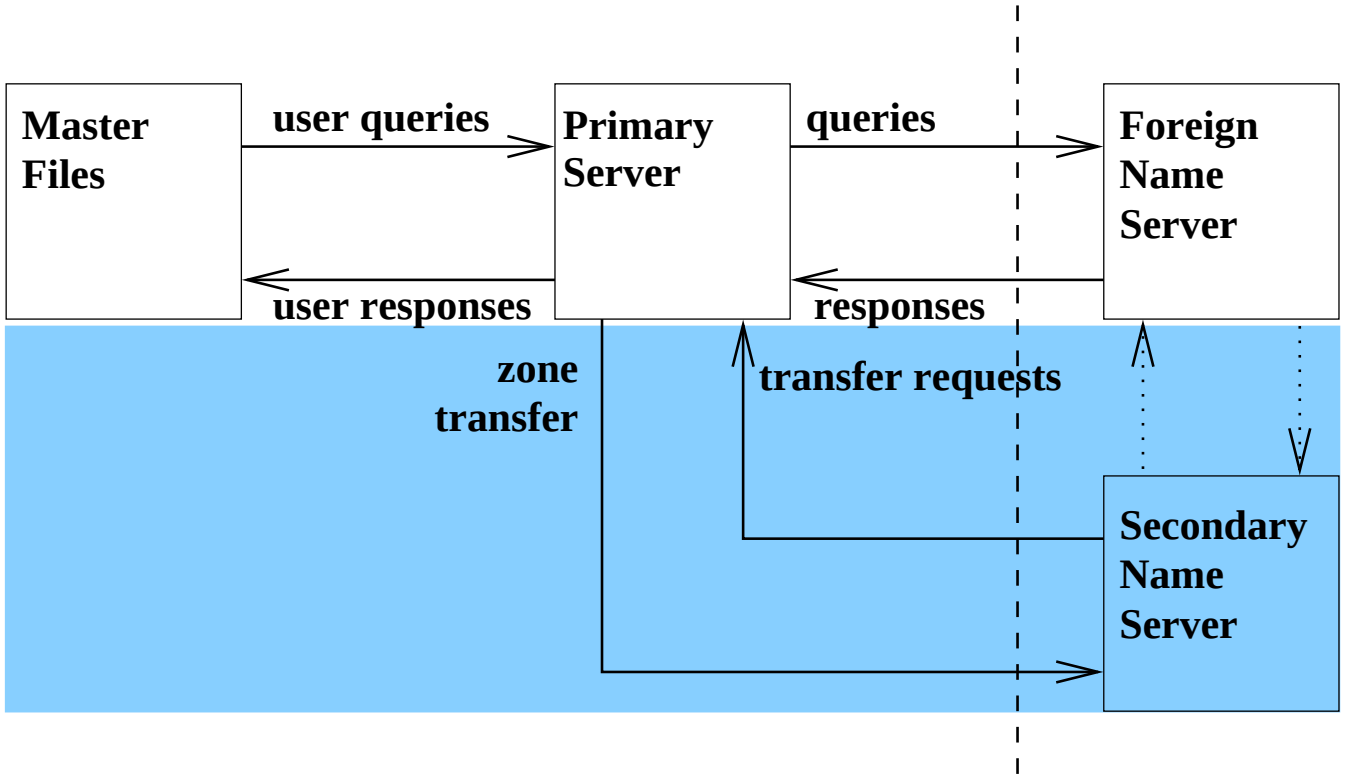
Reverse Query (IP-Adresse → Domainname):

```
> set q=ptr
> 134.2.12.1
Server:  atlas.informatik.uni-freiburg.de
Address:  132.230.150.3

1.12.2.134.in-addr.arpa name = willi.Informatik.Uni-Tuebingen.De
12.2.134.in-addr.arpa   nameserver = dns1.belwue.De
12.2.134.in-addr.arpa   nameserver = dns1.Uni-Tuebingen.De
12.2.134.in-addr.arpa   nameserver = dns3.belwue.De
12.2.134.in-addr.arpa   nameserver = mx01.Uni-Tuebingen.De
12.2.134.in-addr.arpa   nameserver = macon.Informatik.Uni-Tuebingen.De
12.2.134.in-addr.arpa   nameserver = snoopy.Informatik.Uni-Tuebingen.De
dns1.belwue.De  internet address = 129.143.2.1
dns1.Uni-Tuebingen.De   internet address = 134.2.200.1
dns3.belwue.De  internet address = 131.246.119.18
mx01.Uni-Tuebingen.De   internet address = 134.2.3.11
macon.Informatik.Uni-Tuebingen.De       internet address = 134.2.12.17
snoopy.Informatik.Uni-Tuebingen.De      internet address = 134.2.14.4
```

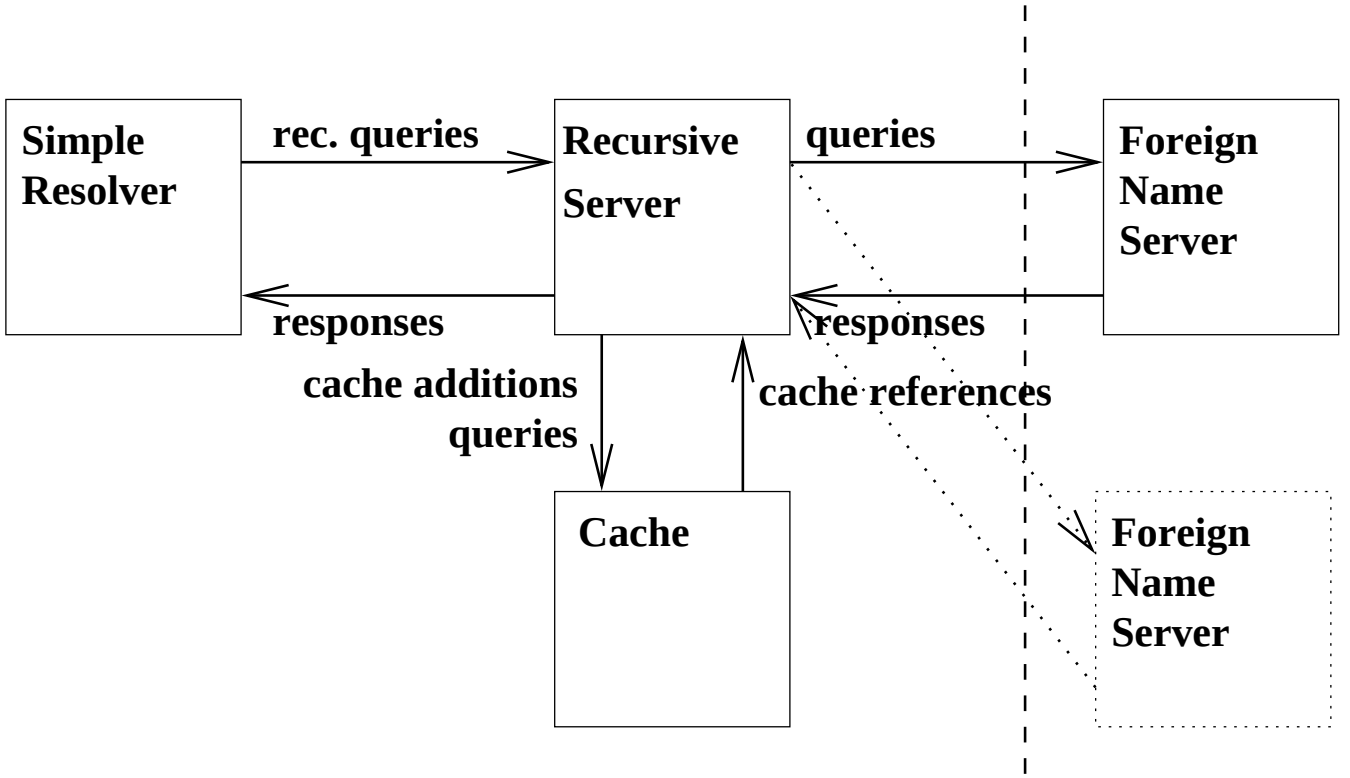
3.8.2 Benutzerperspektive



3.8.3 Primary und Secondary Server



3.8.4 Recursive Queries



3.8.5 Format eines Domainnamens

Folge von Strings (Labels), getrennt durch und beendet mit "."

Maximale Länge eines Labels: 63

Maximale Länge eines Domainnamen: 255 (inkl. der Punkte)

Interne Darstellung: Ein Oktet Länge des Labels, gefolgt von den Zeichen des Labels, wiederholt bis Nulloktet (Label der Länge Null)

Beispiel: `informatik.uni-freiburg.de`

$$\lceil 10 \rceil \texttt{informatik} \lceil 12 \rceil \texttt{uni} - \texttt{freiburg} \lceil 2 \rceil \texttt{de} \lceil 0 \rceil$$

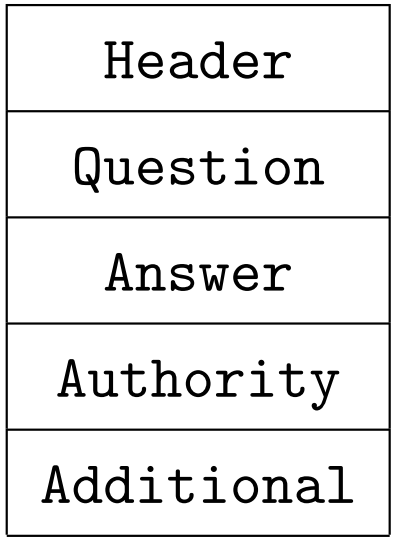
3.8.6 Internes Format eines Resource Record

Feldname	Größe/Oktetts	Beschreibung
NAME	2n	Domainname für den das Record gilt
TYPE	2	Kode für TYPE
CLASS	2	Kode für CLASS
TTL	4	Time to Live, Gültigkeitsdauer/Sek.
RDLENGTH	2	Anzahl der Oktetts im RDATA Feld
RDATA	2d	Inhalt je nach TYPE und CLASS

Kodes für TYPE (Ausschnitt)		
A	1	Host Address
NS	2	authoritative name server
CNAME	5	canonical name for an alias
SOA	6	zone of authority
PTR	12	domain name pointer
MX	15	mail exchanger

Kodes für CLASS (Ausschnitt)		
IN	1	Internet

3.8.7 Format einer Nachricht



Anfrage an den Name-Server

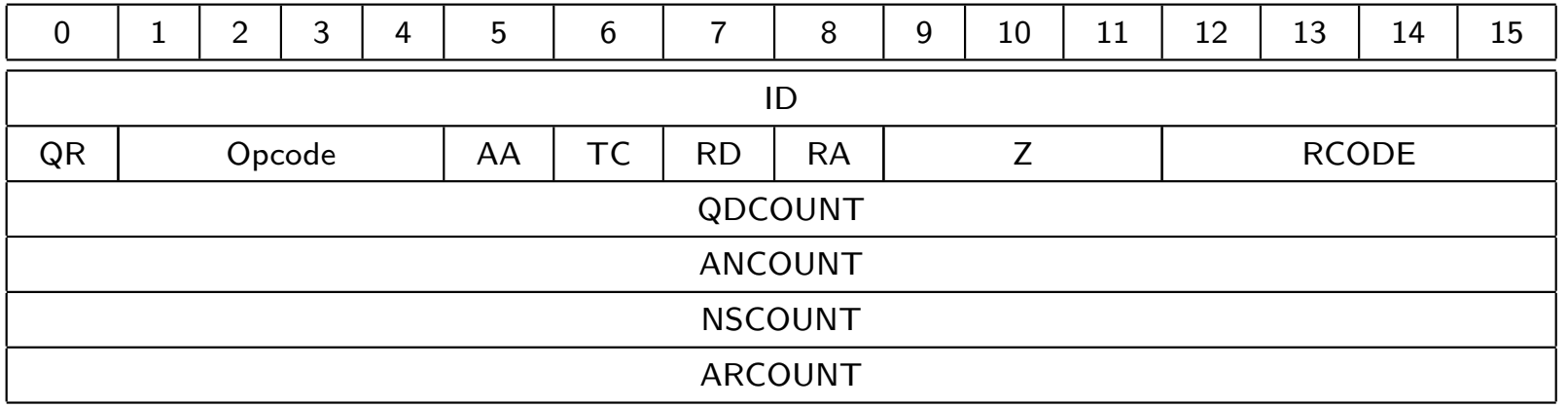
Antworten des Servers

Zeiger auf autorisierten Name-Server

weitere Information

- `Header` immer vorhanden
- `Answer`, `Authority` und `Additional` enthalten je eine Liste von *Resource Records* (RR)

Header 12 Oktette mit folgendem Inhalt



wobei

ID identifier erzeugt vom Client

QR 0= Frage, 1= Antwort

Opcode Art der Anfrage

- **0=** Standard-Anfrage (QUERY)
- **1=** Inverse Anfrage (IQUERY)
- **2=** Status-Anfrage (STATUS)
- **3–15** reserviert

AA 1= Authoritative Answer

TC 1= Truncated (abgeschnitten)

RD 1= Recursion Desired (Wunsch vom Client)

RA 1= Recursion Available (Anzeige vom Server)

Z immer 0

RCODE Response Code

- **0** kein Fehler
- **1** Formatfehler
- **2** Serverfehler
- **3** Gesuchter Name existiert nicht (nur falls AA)
- **4** nicht implementiert
- **5** Anfrage abgelehnt
- **6–15** reserviert

QDCOUNT Anzahl der Einträge in `Question`

ANCOUNT Anzahl der Resource Records in `Answer`

NSCOUNT Anzahl der Name-Server Resource Records in `Authority`

ARCOUNT Anzahl der Resources Records in `Additional`