Internetprogrammierung
Spezialvorlesung, SS2005

Peter Thiemann

Übersicht

- Einführung
- Grundlagen der Netzwerkprogrammierung in Java
- Netzwerkprotokolle der Anwendungsebene
  - DNS, HTTP, SMTP, FTP, RPC, RMI
- XML Grundlagen
  - Umfeld, Definition, DTDs, XHTML, CSS
- Programmierung interaktiver Webanwendungen
  - CGI, Servlets, JSP

- Weiterführende XML Konzepte und Anwendungen
  - Verarbeitung
    - Namespaces, Schemasprachen, XPath, XSLT, XQuery
  - XML Programmierung
    - XML Parser, DOM, SAX, STX, XDuce, Xact, Xtatic
  - Web Services
    - WSDL, SOAP, UDDI, XOP, ...

Organisatorisches

- Tutor L. Wissmann
- Vorbesprechung mit Terminabsprache Mi, 13.4., 17 Uhr
- Aufgaben in Teams zu bearbeiten
  - Programmieraufgaben
  - Experimente
- Abschlussprüfung mündlich in Teams
  auch Fragen zu den Ü-Aufgaben
- Aufzeichnung
- Webseite der Vorlesung

`http://proglang.informatik.uni-freiburg.de/teaching/inetprog/2005/`

1 Das Internet

- globales Kommunikationssystem
- Verbindungen zwischen angeschlossenen Endgeräten

  unicast Rechner — Rechner

  oder

  multicast ein Rechner — viele Rechner
- einheitlicher Adressraum
  (Internet-Adressen, Domainnamen)

1.1 Geschichte

1969: ARPANET (4 Hosts, 50kb/s)

1973: Netzwerkprotokoll TCP/IP (Vinton Cerf, Bob Kahn)
Kommunikation zwischen Netzwerken

1974: Name "Internet"

1976: Ethernet (Bob Metcalfe)

1979: USENET/News

1981: ARPANET, CSNET (213 Hosts)

1983: DNS (562 Hosts)

1988: (56000 Hosts)

1990: Hypertext-System von Tim Berners-Lee (CERN)

1992: Gründung der Internet Society

World-Wide-Web Artikel veröffentlicht

"Surfing the Internet" (Jean Armour Polly)

(1.136.000 Hosts)

1993: Mosaic Browser

1994: Netscape Browser

W3C am MIT gegründet `http://www.w3c.org/`

erste kommerzielle Anwendungen:

Pizzabestellung, First Virtual, Shopping Malls

(3.864.000 Hosts)

1995: 6.642.000 Hosts

1998: $>$ 30.000.000 Hosts

2000: 10.910.395 DNS Einträge,

2003: 19.087.254 DNS Einträge

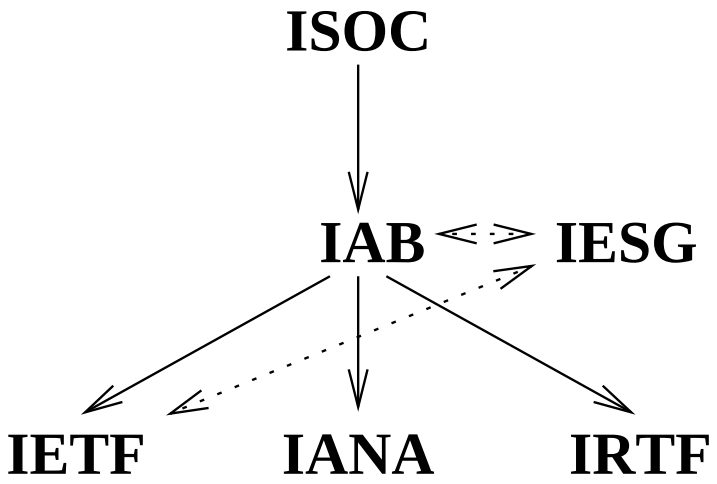
2005: 25.389.171 DNS Einträge aber dynamische (Mehrfach-) Nutzung!

Quelle:

`http://www.ripe.net/info/stats/hostcount/2005/01/`

1.2 Organisation

- dezentral und offen
- Festlegung von Protokollen und Standards durch Vorschlag, Implementierung und Bekanntmachung



IESG Internet Engineering Steering Group

IANA Internet Assigned Numbers Authority

IRTF Internet Research Task Force

1.2.1 The Internet Society

`http://www.isoc.org/`

The Internet Society is a non-profit, non-governmental, international, professional membership organization. It focuses on: standards, education, and policy issues.

1.2.2 Internet Architecture Board

`http://www.iab.org/iab/`

The Internet Architecture Board (IAB) is a technical advisory group of the Internet Society. Its responsibilities include:

1. **IESG Selection**: The IAB appoints a new IETF chair and all other IESG candidates, from a list provided by the IETF nominating committee.
2. **Architectural Oversight**: The IAB provides oversight of the architecture for the protocols and procedures used by the Internet.
3. **Standards Process Oversight and Appeal**: The IAB provides oversight of the process used to create Internet Standards. The IAB serves as an appeal board for complaints of improper execution of the standards process.

4. **RFC Series and IANA**: The IAB is responsible for editorial management and publication of the Request for Comments (RFC) document series, and for administration of the various Internet assigned numbers.

5. **External Liaison**: The IAB acts as representative of the interests of the Internet Society in liaison relationships with other organizations concerned with standards and other technical and organizational issues relevant to the world-wide Internet.

6. **Advice to ISOC**: The IAB acts as a source of advice and guidance to the Board of Trustees and Officers of the Internet Society concerning technical, architectural, procedural, and (where appropriate) policy matters pertaining to the Internet and its enabling technologies.

1.2.3 The Internet Engineering Task Force

`http://www.ietf.org/`

The Internet Engineering Task Force is a loosely self-organized group of people who make technical and other contributions to the engineering and evolution of the Internet and its technologies. **It is the principal body engaged in the development of new Internet standard specifications**.

The IETF meeting is not a conference, although there are technical presentations. The IETF is not a traditional standards organization, although many specifications are produced that become standards. The IETF is made up of volunteers who meet three times a year to fulfill the IETF mission.

IETF's mission includes:

- Identifying, and proposing solutions to, pressing operational and technical problems in the Internet;
- Specifying the development or usage of protocols and the near-term architecture to solve such technical problems for the Internet;
- Making recommendations to the Internet Engineering Steering Group (IESG) regarding the standardization of protocols and protocol usage in the Internet;
- Facilitating technology transfer from the Internet Research Task Force (IRTF) to the wider Internet community; and
- Providing a forum for the exchange of information within the Internet community between vendors, users, researchers, agency contractors and network managers.

RFC

`http://www.ietf.org/rfc.html`

IETF gibt RFCs (Request for Comments) heraus. Diese beschreiben (meist) Protokolle (nicht Datenformate wie z.B. das W3C). Sie durchlaufen einen Standardisierungsprozess. Manche werden zu Standards.

Anforderungen an ein RFC

- Format
- Reviewprozess: Internet-Draft $\Rightarrow$ RFC $\Rightarrow$ Standard (RFC Editor)
- RFCs werden nie revidiert, nur überschrieben.
- De-facto Dokumentation des Internets

1.3 Protokolle

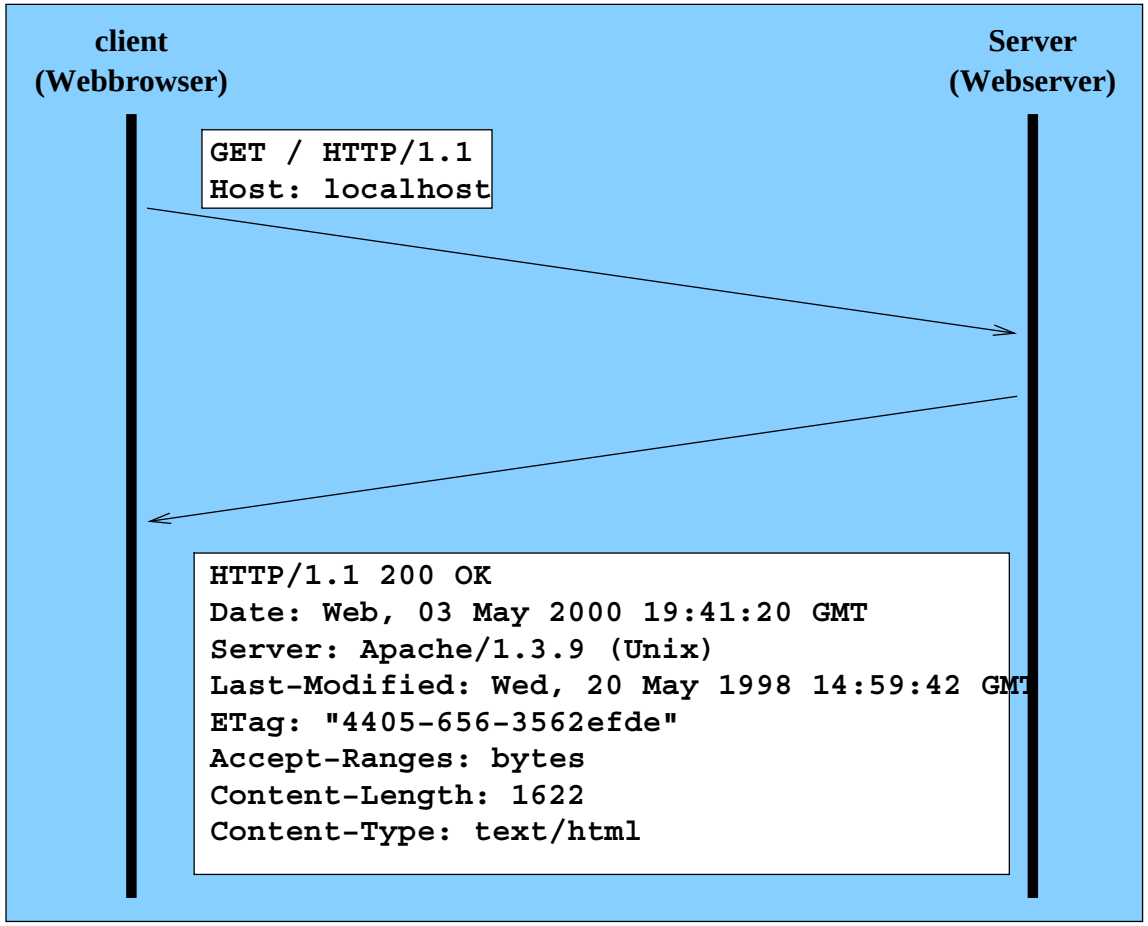
Protokoll Spezifikation der Struktur einer Kommunikation

OSI-ISO Standard definiert 7 Protokollschichten mit festgelegten Aufgaben:

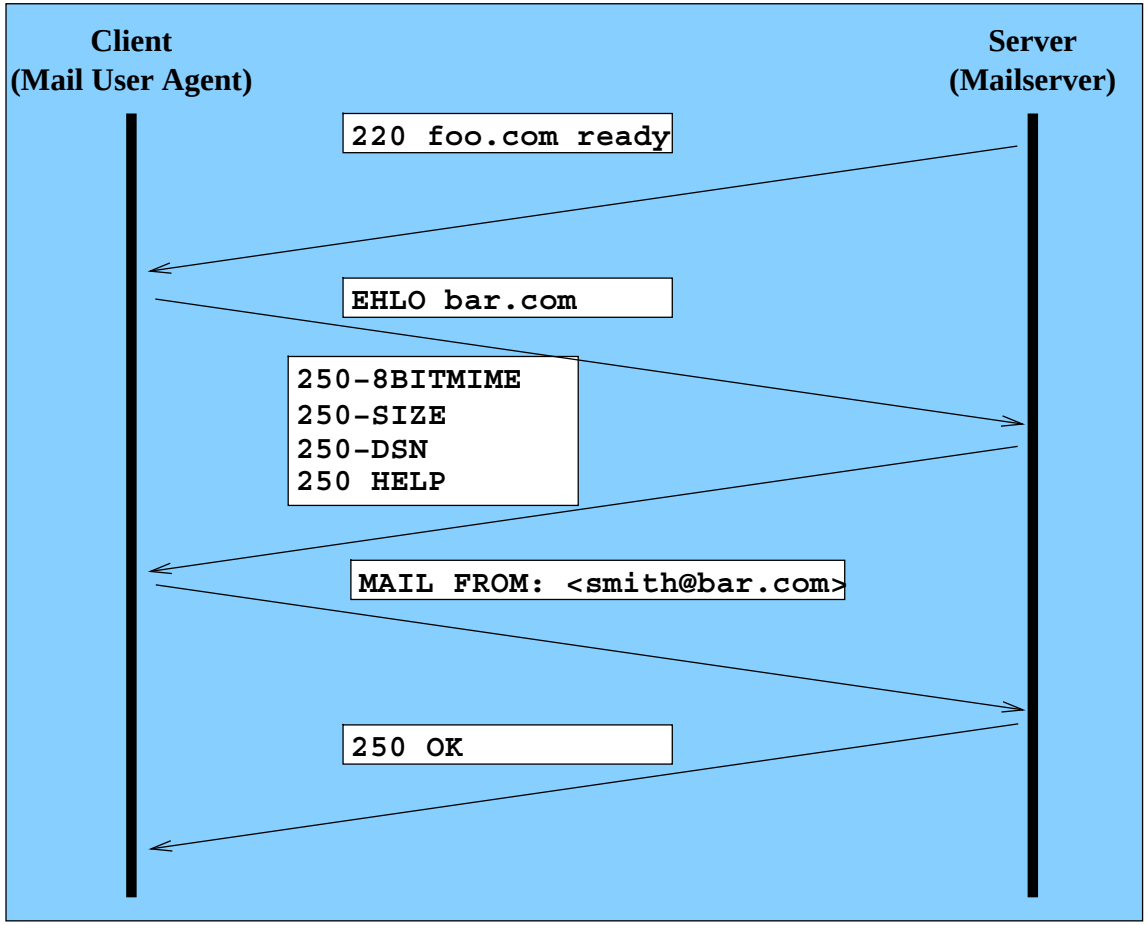
1. Bitübertragung
2. Sicherung
3. Vermittlung
4. Transport
5. Kommunikation
6. Darstellung
7. Anwendung

Praktisch verwendet: Internet Protokoll RFC 791

1.3.1 Beispiel: HTTP (Hypertext Transfer Protocol)



1.3.2 Beispiel: SMTP (Simple Mail Transfer Protocol)



1.3.3 IP Adresse

- Logische Rechneradresse
- Physikalische Rechneradresse wird durchs Netzwerk ermittelt (routing)
- Klassische IP Adresse (IPV4):
  - 32 Bit geschrieben in vier Dezimalzahlen x1.x2.x3.x4 (je 0-255)
  - Bsp: `132.230.168.1`
  - Einteilung in Klassen A, B, C, D heute irrelevant
- IPV6 Adresse:
  - 128 Bit geschrieben in acht Hexzahlen y1:y2:y3:y4:y5:y6:y7:y8 mit max vier Stellen
  - Bsp: `fe80::250:4ff:fe09:7028`
  - Mehr Struktur, Routinginformation, etc eingebaut

1.3.4 Low-level Internet Protokolle

- Datenpaket = Folge von Oktetten
- Adressierung mit Hilfe von IP

UDP – User Datagram Protocol

- Versenden eines Datenpakets (unidirektional)
- Keinerlei Garantieen!

TCP/IP – transmission control protocol / internet protocol

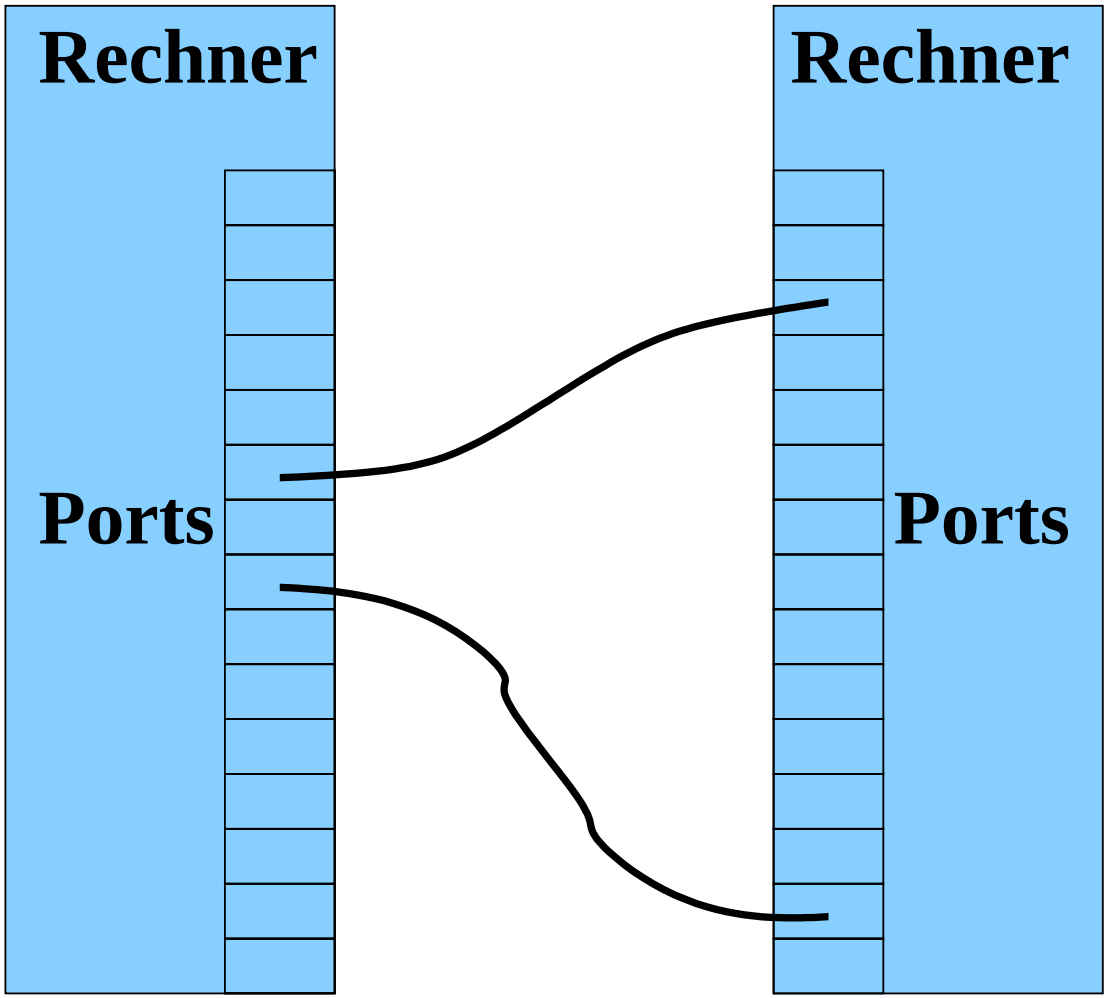
Virtuelle, strombasierte Verbindung (*stream connection*),
d.h. keine feste Länge für Datenpakete vorgegeben.

- Verbindungsaufbau
- **verlässliche** bidirektionale Verbindung
  - Bündelung zu Paketen
  - Paketbestätigung
  - Paketverlust
  - Reihenfolge
  - Lastregelung
- Verbindungsabbau

1.3.5 Dienste (Services)

- Protokolle auf höherer Abstraktionsebene
- meist aufbauend auf TCP/IP
- meist Client-Server Struktur
  aktuell interessant: Peer-to-peer Protokolle z.B. für Filesharing (edonkey, gnutella, freenet, etc)

Ein Rechner – mehrere Dienste



- Auswahl der Dienste durch *Portnummern* (16 Bit) oder Servicenamen (vgl. `/etc/services`; RFC 1700)
- Well Known Ports: 0 – 1023
  Server muss vom Administrator gestartet warten
- Registered Ports: 1024 – 49151
  Keine Restriktion bzgl. Server
- Dynamic/Private Ports: 49152 – 65535

1.3.6 Beispiele für Dienste

Telnet RFC 0854

Virtuelle Terminals für Remote Login (viele Erweiterungen, seit 1972)

Server: `telnetd`

Client: `telnet`; `telnet` ⟨*Host*⟩ ⟨*Portnummer*⟩

mit Verschlüsselung: `sshd` bzw. `ssh` (mit anderem Protokoll)

E-mail Versendung von E-mail: SMTP (Simple Mail Transfer Protocol) RFC 2821

Format einer E-mail: RFC 2822

Erweiterte Mail Inhalte: MIME (Multipurpose Internet Mail Extensions) RFC 2045 – RFC 2049 uva

Server: `sendmail` oder ein anderer MTA (Mail Transfer Agent)

Client: `mail` oder ein anderer MUA (Mail User Agent), z.B. `emacs`, `pine`, `netscape`, `mh`, ...

File Transfer Protocol (FTP)

- Transport von Dateien
- RFC 0959 und viele Erweiterungen
- seit 1971 RFC 0114
- Persönlicher Modus: Account (Name/Passwort) auf Server erforderlich
- Anonymer Modus (anonymous FTP): jeder kann zugreifen

Server: `ftpd`

Client: `ftp`, ...

WWW Hypertext Transfer Protocol (HTTP 1.1)
RFC 2616

- Erweiterbares Protokoll zur Übertragung und Manipulation von getypten Dokumenten
- Adressierung der Dokumente durch URIs (Uniform Resource Identifiers)

Server: AOLserver, Apache, MS IIS, Jigsaw, ...

Clients: `mozilla`, `opera`, `msie`, `amaya`, `wget`, ...

URI (Uniform Resource Identifier) RFC 1630, RFC 3986:

This document defines the syntax used by the World-Wide Web initiative to **encode the names and addresses of objects on the Internet**. *The web is considered to include objects* **accessed using an extendable number of protocols**, *existing, invented for the web itself, or to be invented in the future.* **Access instructions** *for an individual object under a given protocol are* **encoded into forms of address string**. *Other protocols allow the use of object names of various forms. In order to abstract the idea of a generic object, the web needs the concepts of the universal set of objects, and of the universal set of names or addresses of objects.*

Spezielle URIs

- **Uniform Resource Locator (URL)** RFC 1738; RFC 1808, RFC 2368, RFC 3986
  - Symbolische Adresse für ein Dokument (Objekt)
  - Enthält spezifische **Zugriffsinformation**, d.h. Rechnernamen, Passwörter, etc
  - Format:
    $\langle Schema \rangle : \langle schemaspezifische\ Information \rangle$
  - Beispiele:

```
http://www.informatik.uni-freiburg.de/proglang
ftp://ftp.informatik.uni-freiburg.de/iif
mailto:president@whitehouse.gov
```

- **Uniform Resource Name (URN)** RFC 2141
  - **Eindeutiger Name** für ein Dokument (Objekt)
  - Impliziert globalen Namensraum und Persistenz
  - Namensraum verwaltet durch IANA
    `http://www.iana.org/assignments/urn-namespaces`
  - Format:
    `urn:`⟨*NI*⟩`:`⟨*NSS*⟩
    NI — Namespace Identifier (assigned by IANA)
    NSS — Namespace Specific String
  - Beispiel:

```
urn:ietf:rfc:2141
urn:ietf:std:50
urn:ISSN:1560-1560
urn:newsml:iptc.org:20001006:NewsMLv1.0:1
urn:newsml:reuters.com:20000206:
           IIMFFH05643_2000-02-06_17-54-01_L06156584:1U
```

2 Netzwerkprogrammierung in Java

- In package `java.net`

2.1 Internet-Adressen (IP-Adressen)

- Internet-Adresse = vier Oktette (je 8 Bit)
- jedes 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. 8	Newsserver der Uni Freiburg
132.230. 1. 5	WWW-Server der Uni Freiburg
132.230.150.17	WWW-Server der Informatik
129.143. 2. 9	WWW-Server des BelWue (Uni-Netz Baden-Würtemberg)

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
```

2.2 Java: Klasse `InetAddress`

- Objekte repräsentieren IP-Adressen
- kein öffentlicher Konstruktor, stattdessen

```
public static InetAddress getByName(String host)
                          throws UnknownHostException
```

eine IP-Adresse von `host`

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

sämtliche IP-Adressen von `host`

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

IP-Adresse des lokalen Rechners

2.3 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)

2.3.1 Klasse `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 Socket

2.3.2 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 ("-------------------------------------");
        }
    }
}
```

2.3.3 Klasse `ServerSocket`

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 Verbindung.

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

Schließt den `ServerSocket`

2.3.4 Beispielserver

```
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");
            }
        }
    }
}
```

2.4 Verbindungen über URLs

2.4.1 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

2.4.2 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`

2.4.3 Klasse `HttpURLConnection extends URLConnection`

abstrakte Klasse, daher keine Konstruktoren

Wichtige Methoden

- Setzen von HTTP-spezifischen Anfrageparametern
  `static void setFollowRedirects(boolean set)`
  `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;
            }
        }
}
```

2.4.4 UDP Sockets

Zwei Klassen:

- `DatagramPacket` repräsentiert ein Datenpaket (zum Versenden oder nach dem Empfang)
- `DatagramSocket` repräsentiert die eigentliche Verbindung

Klasse `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)
```

zum Versenden von `length` Bytes aus `buf` an `address` und `port`

Wichtige Methoden

```
byte[] getData()

void setData(byte[] buf)

int getLength()

void setLength(int length)

void setAddress(InetAddress iaddr)

void setPort(int iport)
```

Klasse `DatagramSocket`

Wichtige Konstruktoren

```
DatagramSocket()

DatagramSocket(int port)
```

Wichtige Methoden

```
void send(DatagramPacket p) throws IOException

void receive(DatagramPacket p) throws IOException

void close()
```

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 ();
    }
}
```

2.5 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

2.6 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.

2.6.1 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
```

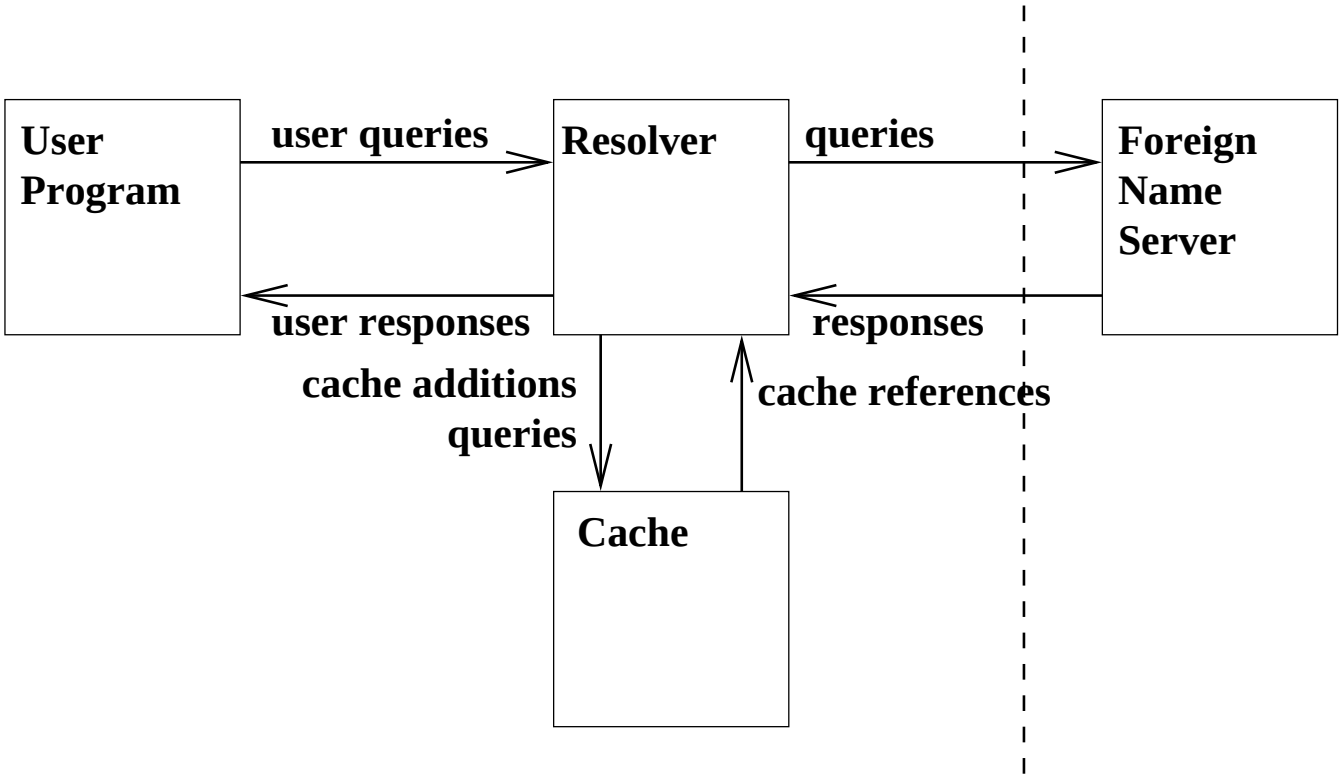
Für Deutschland:

```
> 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
```

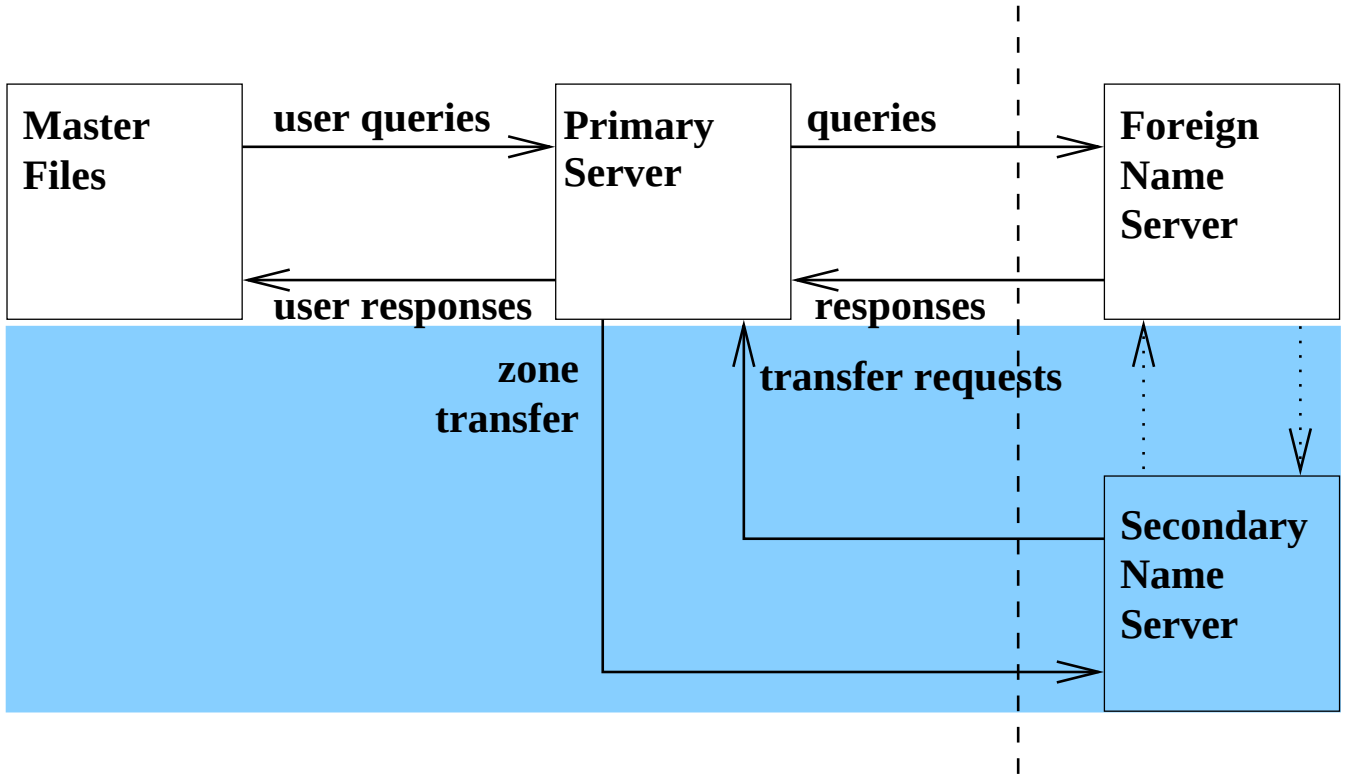
Reverse Query (IP-Adresse → Domainname):

```
> 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
```

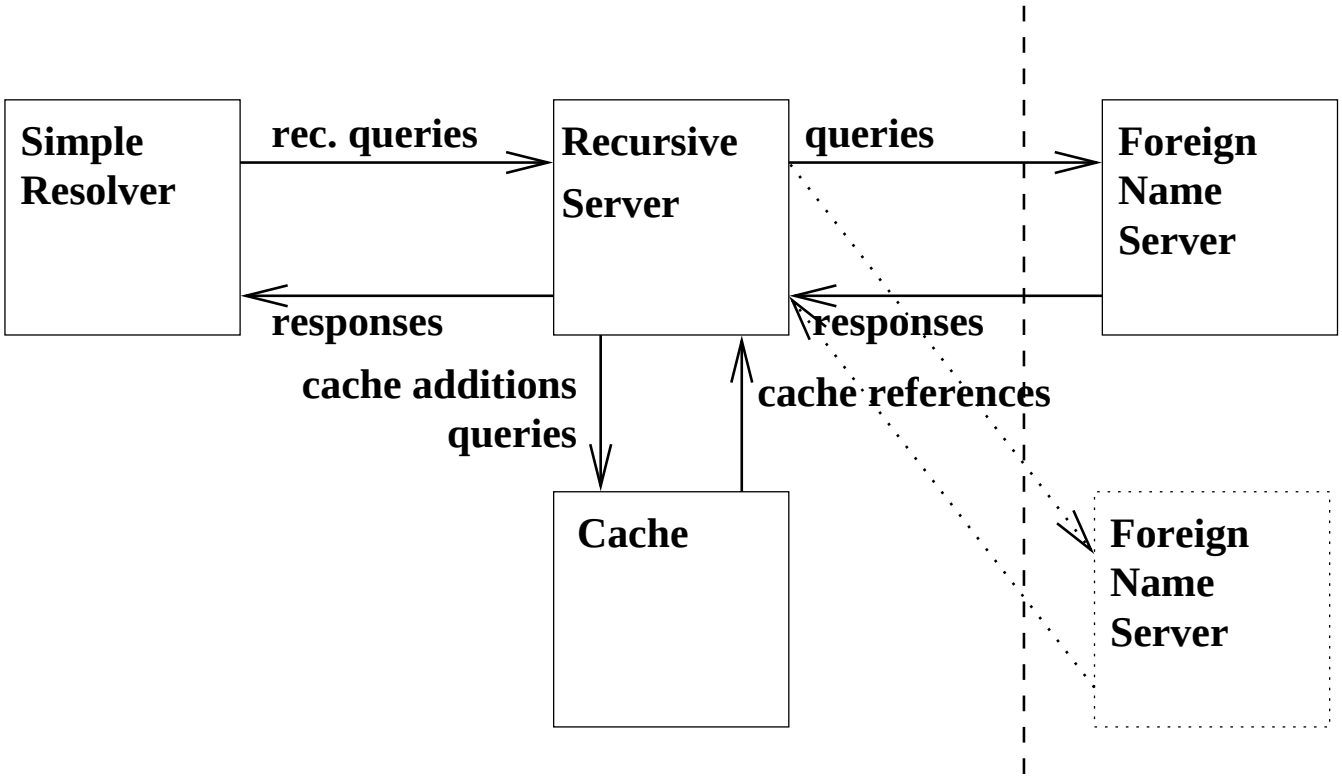
2.6.2 Benutzerperspektive



2.6.3 Primary und Secondary Server



2.6.4 Recursive Queries



2.6.5 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`

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

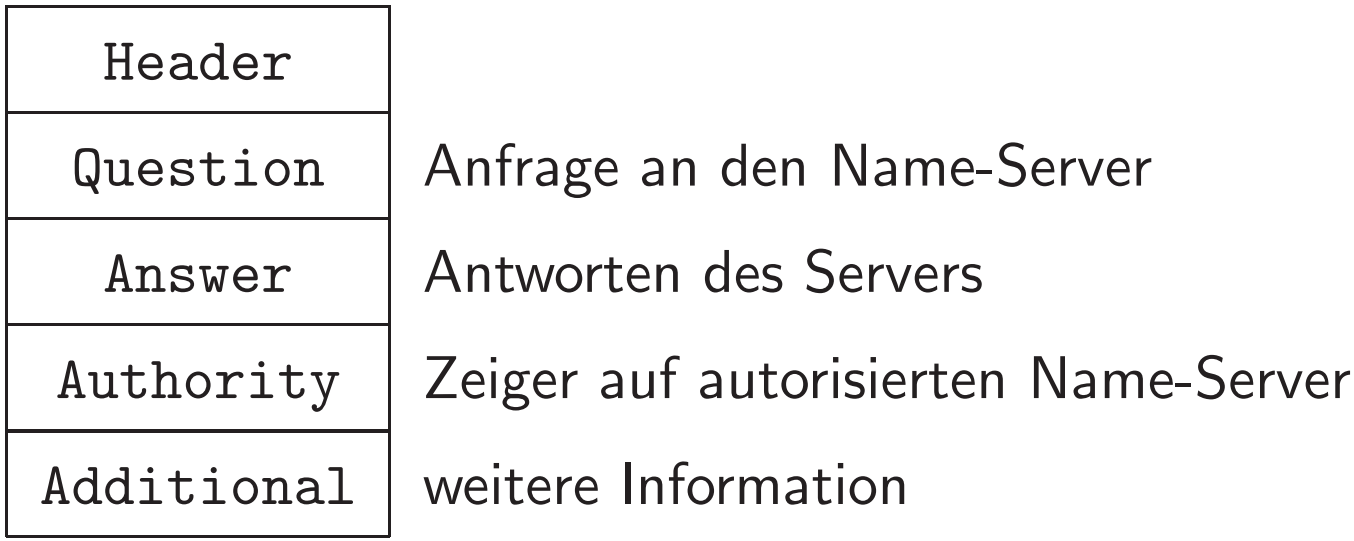
2.6.6 Internes Format eines Resource Record

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

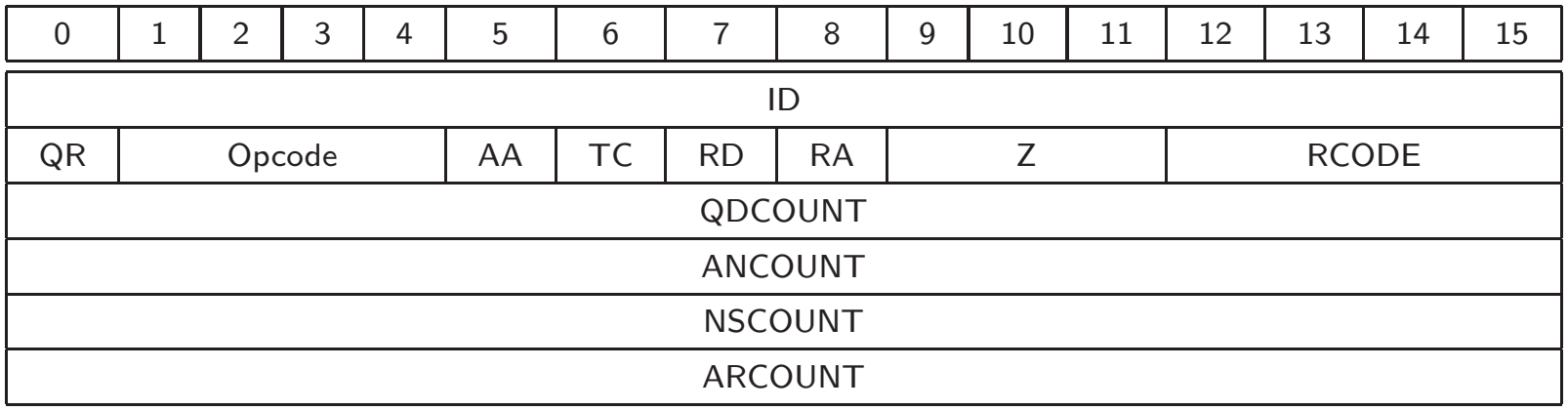
Kodes für CLASS (Ausschnitt)		
IN	1	Internet

2.6.7 Format einer Nachricht



- `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`