



Netzprogrammierung Socketkommunikation im Internet

Prof. Dr.-Ing. Robert Tolksdorf
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
mailto: tolk@inf.fu-berlin.de
<http://www.robert-tolksdorf.de>

- Socketkommunikation im Internet
 - Internet Namen und Nummern
 - TCP
 - UDP
 - Multicast

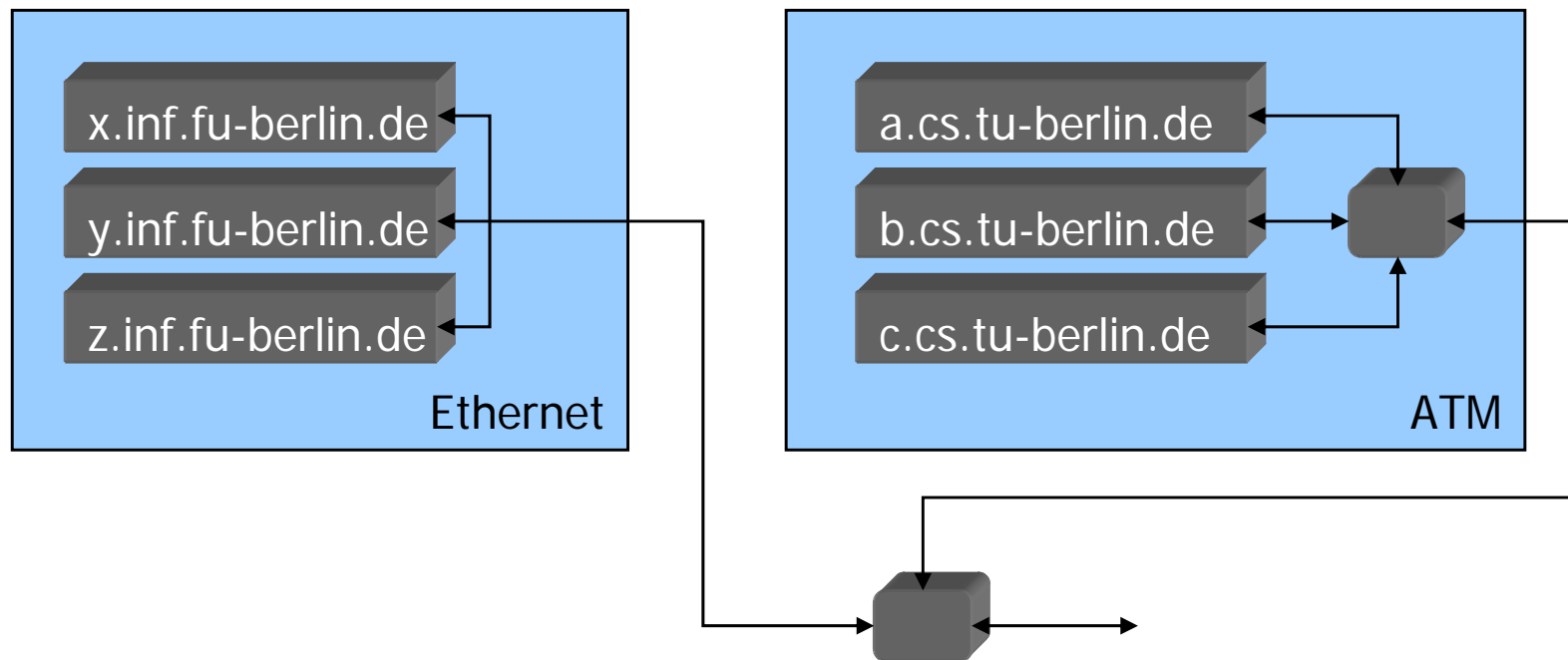


Mitteilungsaustausch Internet

Was ist das Internet

- Eine weltweiter *Verbund von Rechnern*, die über Netze Daten austauschen können.
 - Hardware-bezogene Sicht
 - Zusammenschalten von lokalen Netzen zum Internet
 - Dabei notwendige Verarbeitung von Datenpaketen
- Eine *Protokollfamilie*
 - Netzbezogene Sicht
 - Protokollspezifikationen
- Ein *offenes System*, in dem Dienste genutzt und angeboten werden können.
 - Nutzungs- und anwendungsbezogen
 - Beschreibt die Anwendungsmöglichkeiten des Internet

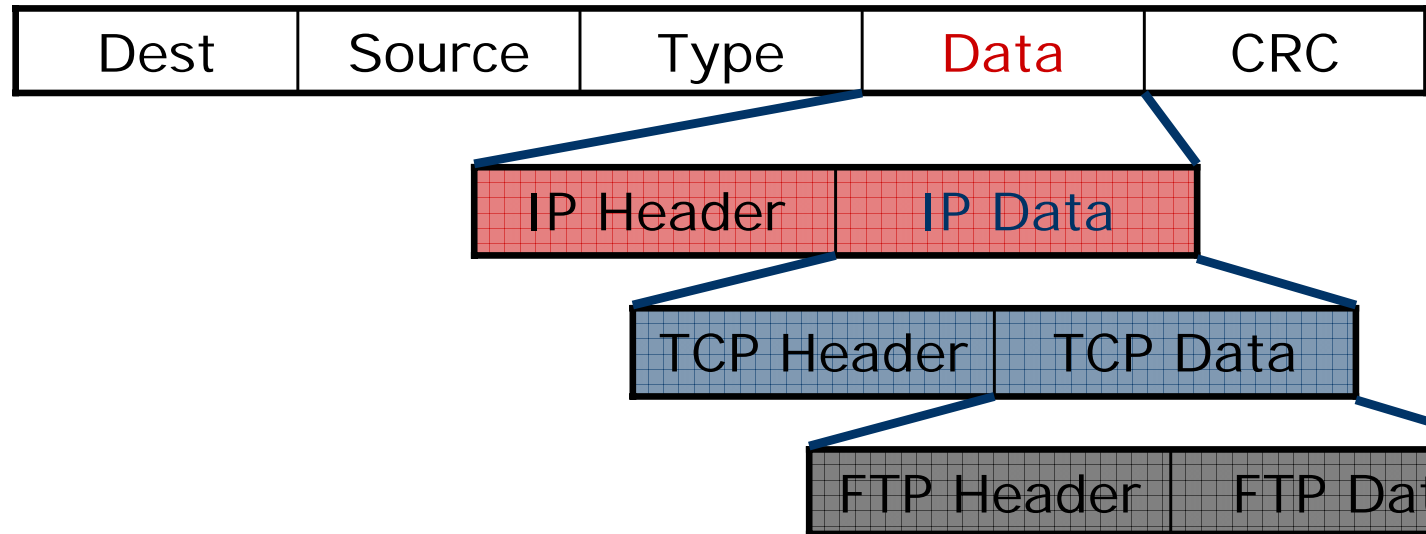
- Das Internet Protokoll IP ermöglicht Internetworking durch Etablierung eines Datenformats und Transportprotokollen, die auf unterschiedlichen Datenverbindungen aufgesetzt werden können



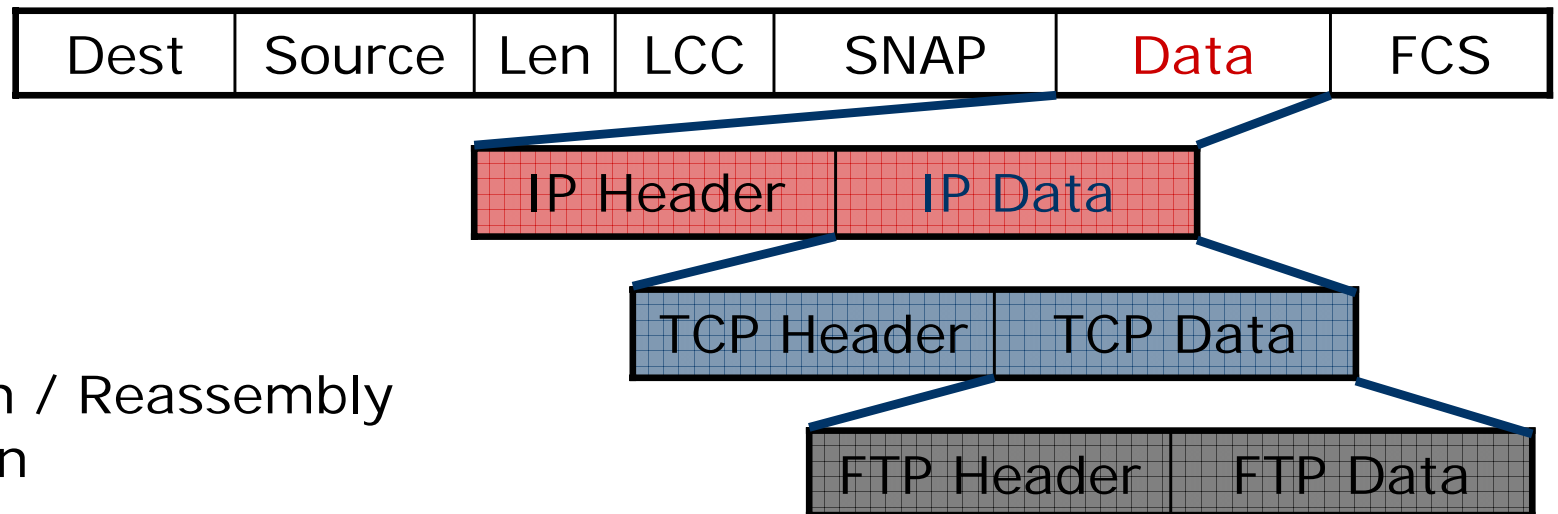
Enveloping / Encapsulating

- File Transfer Protokoll FTP: Übertragung ganzer Dateien zwischen Rechnern

- Ethernet:



- IEEE802.3:



- Fragmentation / Reassembly von IP Paketen

IP Adressen

- Aktuell 32 Bit: z.B. 130.149.27.12
- Abbildung je nach Medium auf die MAC (Media Access Control), die physikalische Netzadresse
 - ARP (Address Resolution Protocol)
 - RARP (Reverse Address Resolution Protocol)
- Netzwerkmaske definiert, was im lokalen Netz ist, und was nach außen geht
- Netzmaske 255.255.0.0 ->
 - 130.149.0.0 bis 130.149.255.255 sind lokales Netz
 - alles andere muss über einen Router laufen
- Routing: Weiterleiten von Paketen in andere Netzwerke

IP Namen

- Internetadresse (IP Adresse) bezeichnet einen Rechner eindeutig
 - als Nummer
160.45.114.204
 - als Name
fock.pcpool.mi.fu-berlin.de
 - Dienste bilden Namen und Nummern aufeinander ab
 - DNS (Domain Name Service)

- InetAddress Objekte werden durch statische Methoden generiert:
 - `static InetAddress getByName(String host)`
Aus Internet-Namen ermitteln (per DNS)
`w3c = InetAddress.getByName("www.w3c.org")`
 - `static InetAddress getByAddress(byte[] addr)`
Aus IP-Nummer ermitteln
`InetAddress myserver =
InetAddress.getByAddress(new byte[] {10,1,2,12});`
 - `static InetAddress getLocalHost()`
Objekt, das den lokalen Rechner bezeichnet
(fast immer: `getLocalHost()=getByName("localhost")`)

- Adresse in unterschiedlichen Formaten:
 - `byte[] getAddress()`
Als Nummer (`byte[] {10,1,2,12}`)
 - `String getHostAddress()`
Nummer als Zeichenkette in Punktnotation ("`10.2.1.21`")
 - `String getHostName()`
Rechnername
- Adressen vergleichen
 - `boolean equals(Object obj)`

Beispiel

- Von Namen nach Punktnotation:

```
import java.net.*;
class IPNumber {
    public static void main(String[] args)
        throws java.lang.Exception {
        InetAddress addr = InetAddress.getByName(args[0]);
        System.out.println(addr.getHostAddress());
    }
}
```

- Beispiel:

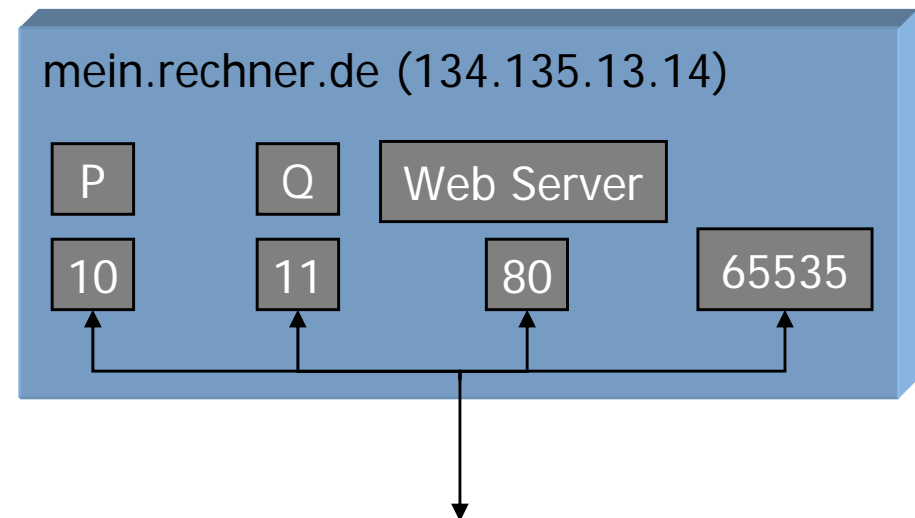
```
java IPNumber fock.inf.fu-berlin.de
160.45.114.204
```

Transportprotokolle

- Form der transportierten Daten: Datagramme
- Ursprung und Ziel der Daten: IP Adressen
- Regeln des Transports: Transportprotokolle
- Zwei Klassen
 - Verbindungorientiert
 - Verbindungsaufbau – Transfer – Verbindungsabbau
 - Beispiel: Fax
 - Overhead für Verbindungsaufbau/-management
 - Verbindungslos
 - Transfer
 - Beispiel: Brief
 - Overhead für Ordnung/Vollständigkeit der Übertragung

Transport Protokolle

- Zwei Protokolle zum Datentransport
 - *UDP*: Ein Paket (Datagramm) von Rechner A nach Rechner B schaffen – Verbindungslos
 - *TCP*: Pakete werden *geordnet* und *zuverlässig* über eine Verbindung transportiert – Verbindungsorientiert
- Ports als Kommunikationsadresse
 - Ein *Port* ist ein logischer Netzanschluss, benannt von 0 bis 65535
- *Socket* ist Endpunkt einer Verbindung





TCP

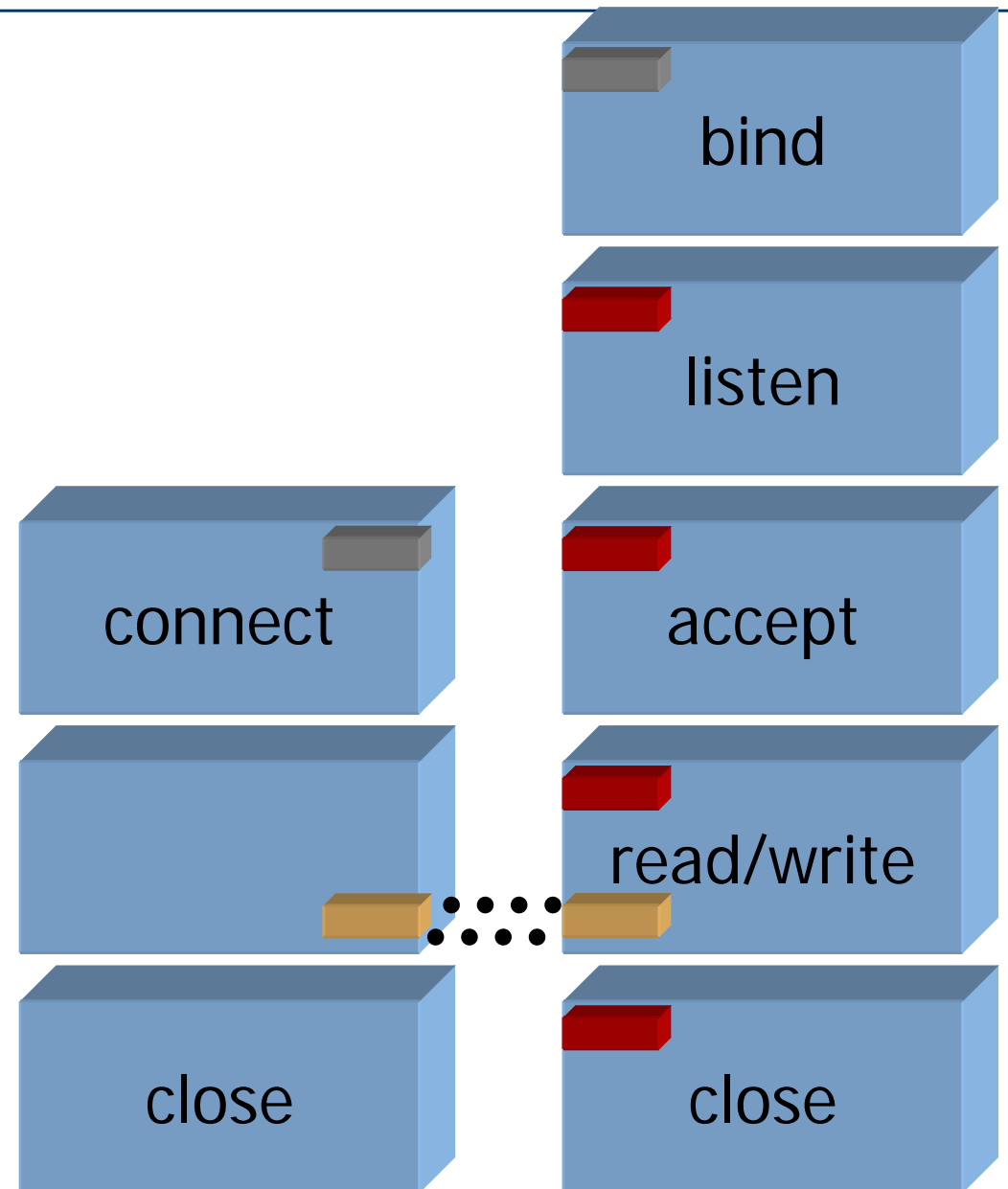
Transmission Control Protocol

TCP Sockets

- Sockets sind die Kommunikationsendpunkte einer Internet-Verbindung
- Die Server Seite, verbindungsorientiert:
 - Ein Prozess reserviert auf einem Rechner einen Port („bind“)
 - Ein Prozess „lauscht“ an dem Port auf Verbindungswünsche („listen“)
 - Einem Verbindungswunsch nimmt er an („accept“) und erzeugt einen Kommunikationssocket
 - Der Kommunikationssocket hat eine andere Nummer als der Verbindungswunschsocket!
- Die Client Seite:
 - Melden des Verbindungswunsches („connect“)
 - „Einstecken“ in den Kommunikationssocket
- Protokollkommunikation:
Zeichen über Kommunikationssocket schicken

TCP Sockets

1. Server reserviert Port
2. Server nimmt Verbindungswünsche an
3. Client schickt Verbindungswunsch
4. Client und Server sind verbunden
bidirektional!
5. Verbindung wird abgebaut



Beispiel

- Server wartet auf eine Zeichenkette und antwortet mit einer anderen Zeichenkette
- Client:

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

```
class PingClient {  
    public static void main (String args[]) throws java.io.IOException  
    {  
        new PingClient(10000);  
    }  
    ...[siehe nächste Folie]  
}
```

```
PingClient(int portNo) throws java.io.IOException {
    String message;
    // zu diesem Rechner verbinden
    Socket socket = new Socket("dein.rechner.de",portNo);
    // Ströme vorbereiten
    BufferedReader in = new BufferedReader(new
        InputStreamReader(socket.getInputStream()));
    PrintStream out = new PrintStream(socket.getOutputStream());
    // Ping sagen
    out.println("Ping");
    // Antwort lesen und ausgeben
    message = in.readLine();
    System.out.println("Got " +message);
    // alles schliessen und Schluss.
    out.close();
    in.close();
    socket.close();
}
```

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

class PongServer {
    public static void main (String args[]) throws java.io.IOException
    {
        new PongServer(10000);
    }

    ... [siehe nächste Folie]
}
```

```
PongServer(int portNo) throws java.io.IOException {  
String message;
```

```
ServerSocket listenSocket = new ServerSocket(portNo); // Socket  
reservieren  
while (true) {  
    // Auf Verbindungswunsch warten  
    Socket commSocket = listenSocket.accept();  
    // Ströme vorbereiten  
    BufferedReader in = new BufferedReader(new InputStreamReader(  
        commSocket.getInputStream()));  
    PrintStream out = new PrintStream(commSocket.getOutputStream());  
    // Eine Zeile lesen und ausgeben  
    message = in.readLine();  
    System.out.println("Got " + message);  
    out.println("Pong"); // "Pong" sagen  
    // Alles schliessen und Schluss  
    out.close();  
    in.close();  
    commSocket.close();  
}  
}
```

Übersetzen und Ausführen

- javac PingClient.java
- javac PongServer.java

- dein.rechner.de: >java PongServer
Got Ping
Got Ping
Got Ping

- >java PingClient
Got Pong
>java PingClient
Got Pong
>java PingClient
Got Pong

Socket API in Java (java.net.Socket)

- Konstruktoren

- `Socket()`

- Socket Objekt erzeugen

- `Socket(InetAddress address, int port)`

- `Socket(String host, int port)`

- Socket Objekt erzeugen und an Rechner/Port verbinden

- Information

- `InputStream` `getInputStream()`

- `OutputStream` `getOutputStream()`

- Lese-/Schreibstrom des Sockets

- `InetAddress` `getInetAddress()`

- Partneradresse (oder null)

- `int` `getPort()`

- Partnerport

Socket API in Java (java.net.Socket)

- Verbindungen
 - `void bind(SocketAddress bindpoint)`
Clientenseitigen Socket-Endpunkt festlegen
(SocketAddress: InetAddress x Port)
 - `void connect(SocketAddress endpoint)`
`void connect(SocketAddress endpoint, int timeout)`
Mit Partnerseitigen Socket-Endpunkt verbinden
 - `void shutdownInput()`
`void shutdownOutput()`
Sende-/Empfangsstrom schliessen
 - `void close()`
Socket schliessen – Verbindung beenden
- Zustand
 - `boolean isBound()`
Gebunden?
 - `boolean isConnected()`
Verbunden?
 - `boolean isInputShutdown()`
`boolean isOutputShutdown()`
Lese-/Schreibkanal geschlossen oder offen?
 - `boolean isClosed()`
Socket geschlossen?

- Optionen

- `void setKeepAlive(boolean on)`
`boolean getKeepAlive()`
Socket durch Probepakete offen halten
- `void setReuseAddress(boolean on)`
`boolean getReuseAddress()`
Socket nach Schließen sofort wieder verwenden
- `void setReceiveBufferSize(int size)`
`int getReceiveBufferSize()`
`void setSendBufferSize(int size)`
`int getSendBufferSize()`
Größe des Empfangs-/Sendepuffers festlegen/lesen
- `void setSoTimeout(int timeout)`
`int getSoTimeout()`
Nach timeout ms `java.net.SocketTimeoutException` werfen

- Konstruktoren
 - `ServerSocket()`
ServerSocket Objekt erzeugen
 - `ServerSocket(int port)`
ServerSocket Objekt erzeugen und an port binden
 - `ServerSocket(int port, int backlog)`
... mit Puffer für backlog Verbindungswünsche
 - `ServerSocket(int port, int backlog, InetAddress bindAddr)`
... auf lokale binAddr
- Information
 - `int getLocalPort()`
Lokale Portnummer
 - `InetAddress getInetAddress()`
Lokale Adresse

- Verbindung
 - `void bind(SocketAddress endpoint)`
`void bind(SocketAddress endpoint, int backlog)`
Binden an Port (SocketAddress: IP-Adresse x Port)
 - `Socket accept()`
Verbindungswunsch annehmen
 - `void close()`
Schliessen
- Zustand
 - `boolean isBound()`
Gebunden?
 - `boolean isClosed()`
Geschlossen?

- Optionen

- `void setReuseAddress(boolean on)`

`boolean getReuseAddress()`

Socket nach Schliessen sofort wiederverwenden

- `void setReceiveBufferSize(int size)`

`int getReceiveBufferSize()`

Größe des Empfangspuffers festlegen/lesen

- `void setSoTimeout(int timeout)`

`int getSoTimeout()`

Nach timeout ms `java.net.SocketTimeoutException` werfen

- Einer der Internet-Standarddienste ist daytime
 - Wartet auf Port 13 / TCP
 - übermittelt auf Anfrage die lokale Uhrzeit als Textzeile
- Ziel: Java-Klasse schreiben, die diesen Dienst abfragt
- ```
>java DayTime np.ag-nbi.de
11 NOV 2005 09:55:02 CET
```
- ```
>java DayTime www.w3.org  
Sun Mar 15 06:41:01 1998
```

DayTime.java

```

import java.io.*;
import java.net.*;
public class DayTime {
    public static void main(String args[]) throws java.io.IOException
    {
        String message;
        Socket socket = new Socket(args[0],13);
        // Lesestrom vorbereiten
        BufferedReader in = new BufferedReader(new
            InputStreamReader(Socket.getInputStream()));
        message = in.readLine();
        System.out.println(message);
        in.close();
        socket.close();
    }
}

```



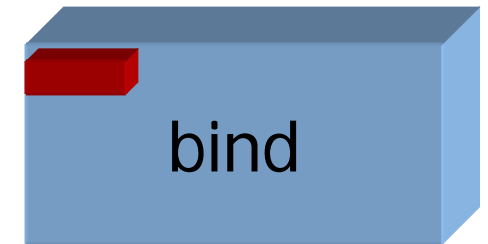
UDP

User Datagram Protocol

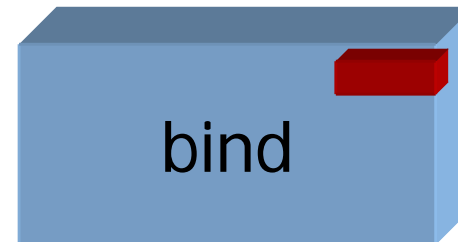
TCP vs. UDP

- *TCP*: Pakete werden *geordnet* und *zuverlässig* über eine Verbindung transportiert
- *UDP*: Ein Paket (Datagram) von Rechner A nach Rechner B schaffen

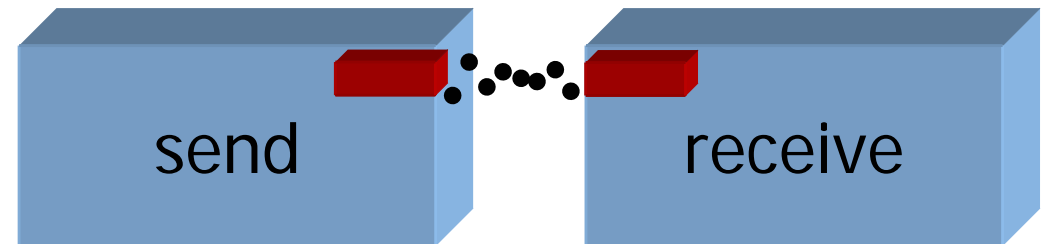
1. Server bindet Socket



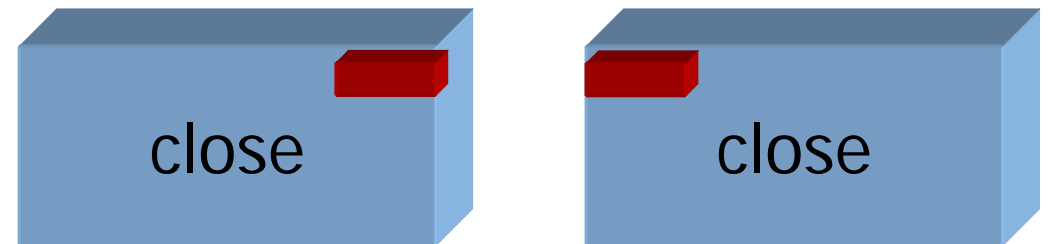
2. Client bindet Socket



3. Client und Server
senden und empfangen
bidirektional!



4. Sockets werden
aufgegeben



- Server wartet auf ein Paket mit Zeichenkette und antwortet mit einer anderen Zeichenkette

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

```
class UDPPongServer {
```

```
    public static void main (String args[]) throws java.io.IOException  
    {
```

```
        new UDPPongServer(10000);
```

```
    }
```

```
    ...[siehe nächste Folie]
```

```
}
```

```
UDPPongServer(int portNo) throws java.io.IOException {
byte[] inData = new byte[1024];    // Platz für Pakete vorbereiten
byte[] outData = new byte[1024];
String message;

DatagramSocket socket = new DatagramSocket(portNo); // Socket binden
while (true) {
    // Ein Paket empfangen
    DatagramPacket in = new DatagramPacket(inData,inData.length);
    socket.receive(in);
    // Infos ermitteln und ausgeben
    InetAddress senderIP = in.getAddress();
    int senderPort = in.getPort();
    message=new String(in.getData(),0,in.getLength());
    System.out.println("Got "+message+" from "+senderIP+", "+senderPort);
    // Antwort erzeugen
    outData = "Pong".getBytes();
    DatagramPacket out =
        new DatagramPacket(outData,outData.length, senderIP,senderPort);
    socket.send(out); // Antwort senden
}
}
```

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

class UDPPingClient {
    public static void main (String args[]) throws java.io.IOException
    {
        new UDPPingClient(10000);
    }
    ...[siehe nächste Folie]
}
```

```
UDPPingClient(int portNo) throws java.io.IOException {
    byte[] inData = new byte[1024];
    byte[] outData = new byte[1024];
    String message;

    // Socket erzeugen
    DatagramSocket socket = new DatagramSocket();
    // Paket bauen und adressieren
    InetAddress serverIP = InetAddress.getByName("localhost");
    outData = "Ping".getBytes();
    DatagramPacket out =
        new DatagramPacket(outData,outData.length, serverIP,portNo);
    // Paket senden
    socket.send(out);
    // Antwort empfangen und ausgeben.
    DatagramPacket in = new DatagramPacket(inData,inData.length);
    socket.receive(in);
    message=new String(in.getData(),0,in.getLength());
    System.out.println("Got "+message);
    // Socket schliessen
    socket.close();
}
```

Übersetzen und Ausführen

- javac UDPPingClient.java
- javac UDPPongServer.java

- >java UDPPongServer
Got Ping from /127.0.0.1,2278
Got Ping from /127.0.0.1,2279
Got Ping from /127.0.0.1,2280

- >java UDPPingClient
Got Pong
>java UDPPingClient
Got Pong
>java UDPPingClient
Got Pong

- Konstruktoren
 - `DatagramSocket()`
Socket Objekt erzeugen
 - `DatagramSocket(int port)`
Socket Objekt erzeugen und an Port binden
- Informationen
 - `InetAddress` `getLocalAddress()`
Lokale Adresse
 - `int` `getLocalPort()`
Lokale Portnummer

- Verbindung
 - `void send(DatagramPacket p)`
`void receive(DatagramPacket p)`
Datagramm senden/empfangen
 - `void close()`
Socket schließen
- Zustand
 - `boolean isClosed()`
Geschlossen?

- Optionen

- void setSendBufferSize(int size)
 int getSendBufferSize()
 void setReceiveBufferSize(int size)
 int getReceiveBufferSize()
 Größe des Empfangs-/Sendepuffers festlegen/lesen
- void setReuseAddress(boolean on)
 boolean getReuseAddress()
 Socket nach Schliessen sofort wiederverwenden
- void setSoTimeout(int timeout)
 int getSoTimeout()
 Nach timeout ms java.net.SocketTimeoutException werfen

- Konstruktoren
 - `DatagramPacket(byte[] buf, int length)`
Paket der Länge `length` konstruieren
 - `DatagramPacket(byte[] buf, int length, InetAddress address, int port)`
Paket der Länge `length` konstruieren und "adressieren"
 - `DatagramPacket(byte[] buf, int offset, int length)`
Paket der Länge `length` konstruieren, das ab `offset` gefüllt wird

- void setAddress(InetAddress iaddr)
InetAddress getAddress()
Empfänger/Absender Adresse setzen/lesen
- void setPort(int ipport)
int getPort()
Empfänger/Absender Port setzen/lesen
- void setData(byte[] buf)
void setData(byte[] buf, int offset, int length)
byte[] getData()
Inhaltsdaten schreiben/auslesen
- void setLength(int length)
int getLength()
Länge der Daten setzen/lesen
- int getOffset()
Offset des Datenbereichs lesen

Wichtige Ausnahmen in java.net

- Socket für Verbindungswünsche nicht aufbaubar
`BindException`
- Verbindungsaufbau gescheitert
`ConnectException`
- Rechnername konnte nicht aufgelöst werden
`UnknownHostException`
- Keine Netzverbindung zum Zielrechner
`NoRouteToHostException`
- Fehler im Transportprotokoll
`ProtocolException`
`SocketException`

RMI vs. Sockets

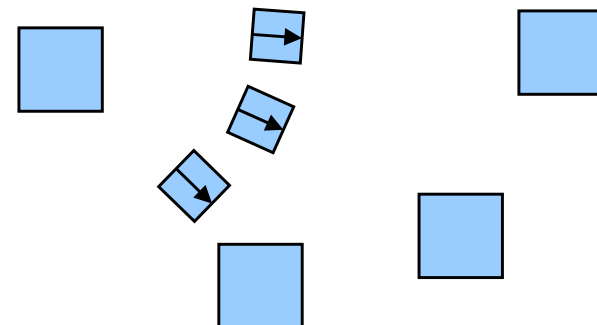
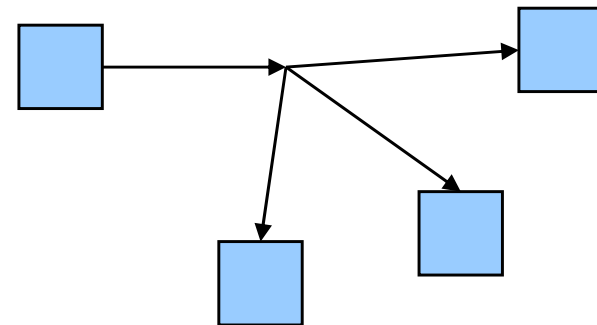
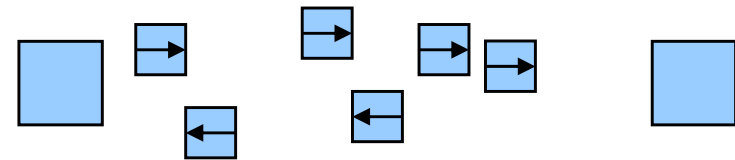
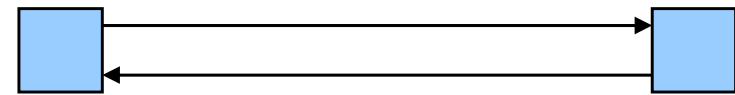
- RMI
 - Höheres Abstraktionsniveau
 - Interaktionsform fest vorgegeben
 - Übermittlung getypter Daten
 - Klassenübermittlung
 - ...
- Sockets
 - Kleinster gemeinsamer Nenner des Internets
 - Nicht vorgegebene Interaktionsform
 - Übermittlung ungetypter Byteströme
 - Effizienter
 - ...



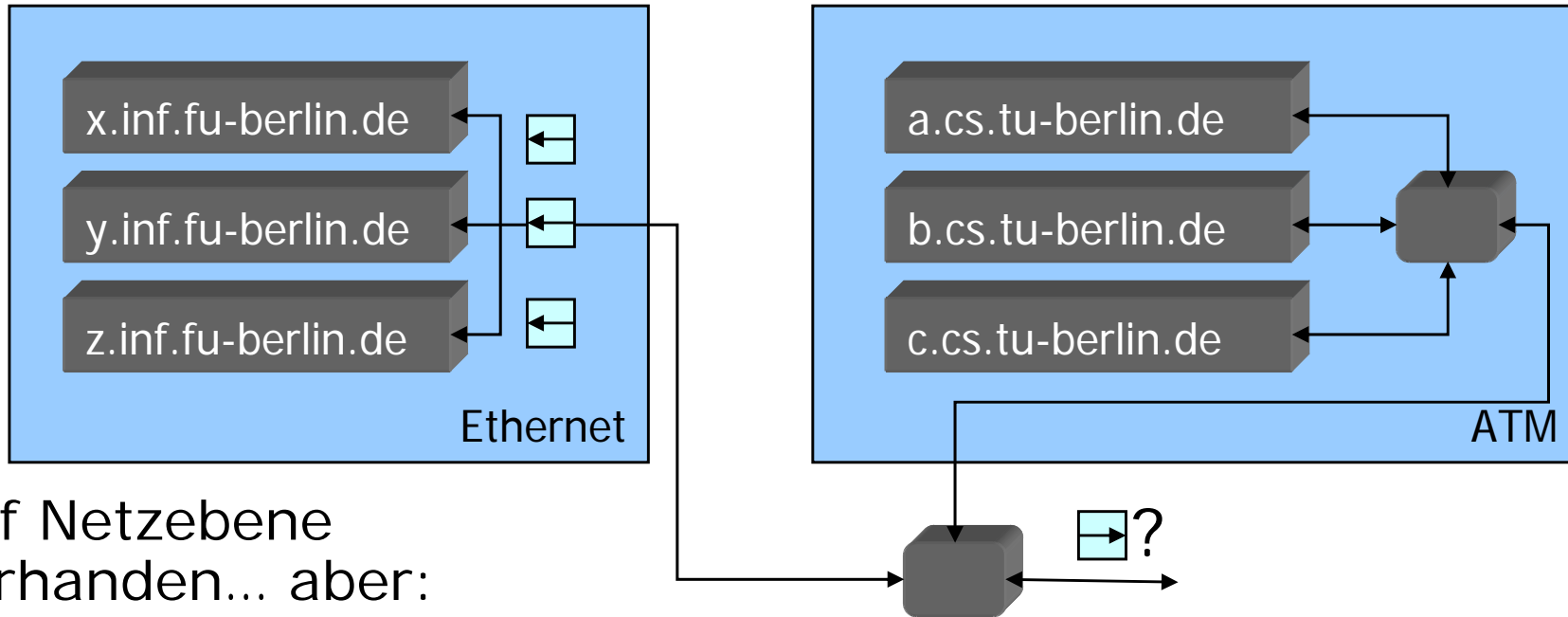
Multicast

Transportmöglichkeiten

- Verbindungsorientiert 1:1
TCP
- Verbindungslos 1:1
UDP
- Verbindungsorientiert 1:n
Multicast
- Verbindungslos 1:n



Wie weit soll transportiert werden?



- Auf Netzebene vorhanden... aber:
- Sollen Router alle Pakete an alle weiterleiten?
- Wie wird Transport gesichert?
- Unklar: Viele IP-Protokollentwürfe
- Müsste in Routern implementiert werden
- Protocol Independent Multicast (PIM)
- Real-time Transport Protocol (RTP)
- Real-time Control Protocol (RTCP)
- Real-Time Streaming Protocol (RTSP)
- Resource Reservation Protocol (RSVP)
- Reliable Multicast Transport Protocol (RTMP)
- Routing Information Protocol (RIP)
- Open Shortest Path First Protocol (OSPF)
- Cisco's Group Management Protocol (CGMP)

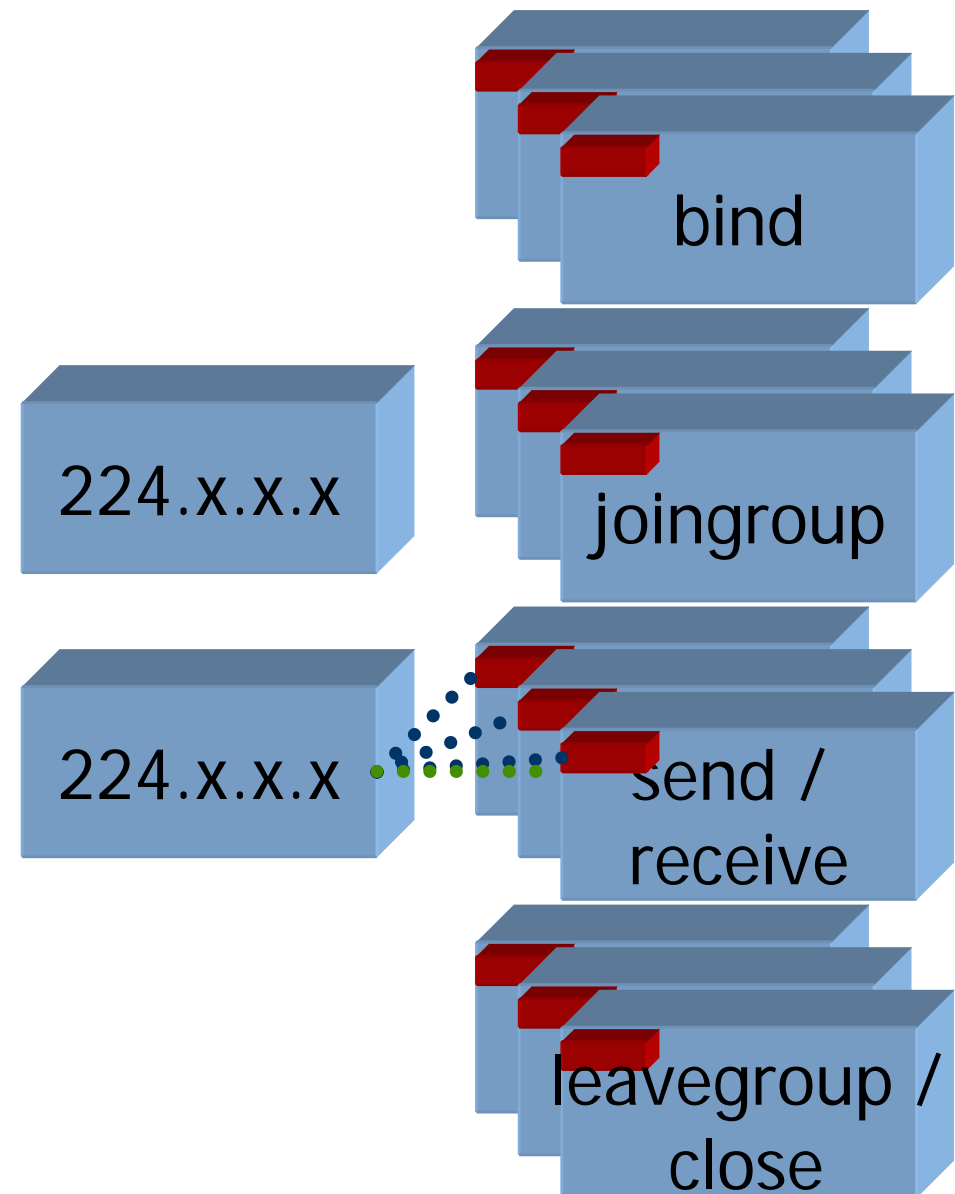
UDP Gruppen Sockets

1. Teilnehmer binden
Sockets

2. Teilnehmer treten
Gruppe bei

3. Teilnehmer
senden und empfangen

4. Teilnehmer verlassen
Gruppe und geben
Socket auf



Beispiel: 1:n Ping-Pong/1

```
import java.net.*;

class MCPing {
    public static void main(String[] argv) throws Exception {
        byte[] inData = new byte[1024]; // Gruppe beitreten
        InetAddress group = InetAddress.getByName("229.1.2.3");
        MulticastSocket mcSocket = new MulticastSocket(4000);
        mcSocket.joinGroup(group);

        DatagramPacket ping =
            new DatagramPacket("Ping".getBytes(), "Ping".length(),
                               group, 4000);
        mcSocket.send(ping); // ping an alle Teilnehmer
    }
}
```

Beispiel: 1:n Ping-Pong/1

```

mcSocket.setSoTimeout(1000); // Timeout zum Einsammeln
while (true) {
    try {
        DatagramPacket in =
            new DatagramPacket(inData,inData.length);
        mcSocket.receive(in);    // Antworten holen
        System.out.println("Got " +
            new String(in.getData(),0,in.getLength()));
    } catch (SocketTimeoutException ste) {
        break;
    }
}
mcSocket.leaveGroup(group);
}

```

Beispiel: 1:n Ping-Pong/1

```
import java.net.*;

class MCPong {
    public static void main(String[] argv) throws Exception {
        byte[] inData = new byte[1024];
        byte[] outData = new byte[1000];
        // Gruppe beitreten
        InetAddress group = InetAddress.getByName("229.1.2.3");
        MulticastSocket mcSocket = new MulticastSocket(4000);
        mcSocket.joinGroup(group);
    }
}
```

Beispiel: 1:n Ping-Pong/1

```
// Antwort empfangen und ausgeben.
DatagramPacket in =
    new DatagramPacket(inData,inData.length);
mcSocket.receive(in);
String message=new String(in.getData(),0,in.getLength());
System.out.println("Got "+message);
// Mit Kommandozeilen-Mitteilung antworten
DatagramPacket out =
    new DatagramPacket(argv[0].getBytes(),
                        argv[0].length(), group,4000);
mcSocket.send(out);
mcSocket.leaveGroup(group);
}
}
```

Ausführen

- ```
>java MCPing
Got Ping
Got two
Got one
```
- ```
>java MCPong two  
Got Ping  
>
```
- ```
>java MCPong one
Got Ping
>
```

# MulticastSocket API

## (java.net.MulticastSocket)

- Unterklasse von java.net.DatagramSocket
- Konstruktoren
  - `MulticastSocket(int port)`  
Socket auf port erzeugen
- Gruppenmanagement
  - `void joinGroup(InetAddress mcastaddr)`  
Gruppe unter der Adresse mcastaddr beitreten  
Für Multicast-Gruppen sind die Adressen  
224.0.0.0 bis 239.255.255.255 reserviert  
(224.0.0.0 nicht nutzen)
  - `void leaveGroup(InetAddress mcastaddr)`  
Gruppe verlassen
- Geerbt
  - send, receive
  - Socket Optionen, etc.



## Zusammenfassung

# Zusammenfassung

---

1. Netzwerkkommunikation im Internet
  1. Sichten auf Internet
  2. Internet verbindet Netze
2. Internet Namen und Nummern
  1. Internet-Rechner haben Nummern und Namen
  2. DNS bildet zwischen ihnen ab
3. TCP Sockets
  1. Verbindungsorientiert
  2. Server bindet und lauscht
  3. Eigener Kommunikationssocket pro Verbindung
4. UDP Sockets
  1. Verbindungslos
  2. Nur Datagramme verschicken
  3. Nebenläufige Serverprogramme
5. Multicast
  1. Versenden einer Mitteilung an viele Empfänger



- [www.ietf.org](http://www.ietf.org), RFC 768 (UDP) und RFC
- Washburn, Kevin; Evans, Jim  
TCP/IP, Aufbau und Betrieb  
eines TCP/IP-Netzes  
Preis: 49.95 Euro (Listenpreis)  
2000, Nachdr. 2000. X, 614 S.  
Addison-Wesley, München  
3-8273-1145-4

