

Netzprogrammierung 2. Mitteilungsorientierte Middleware

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



[1] © Robert Tolksdorf, Berlin

Überblick

[2] © Robert Tolksdorf, Berlin

Überblick

- Client-Server
- Mitteilungsaustausch im Internet
 - TCP
 - UDP
 - Nebenläufige Serverprogramme
 - Multicast

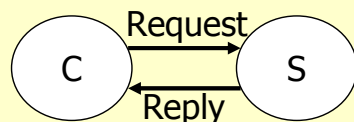
[3] © Robert Tolksdorf, Berlin

Client Server

[4] © Robert Tolksdorf, Berlin

Client-Server

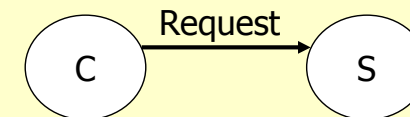
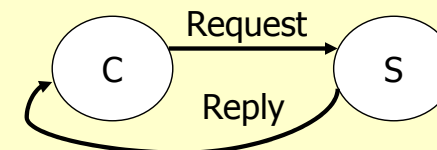
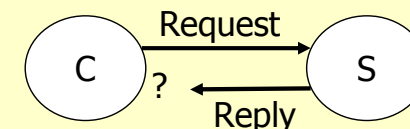
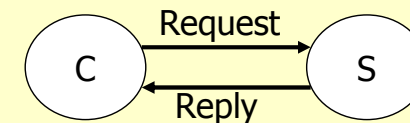
- Rechner interagieren über Rechnergrenzen durch
 - Nachrichtenaustausch (z.B. Internet Mitteilungen)
 - Fernaufruf (RPC, RMI, CORBA)
 - Simulierten gemeinsamen Speicher (Tupelraum)
 - ...
- Dominierendes Interaktionsmodell: Client/Server
- *Client*: Prozess, der Dienst von einem anderen Prozess anfordert (Anforderung, Request)
- *Server*: Prozess, der auf Anforderung eines Clients einen Dienst erbringt und Ergebnis vermeldet (Antwort, Reply)
- Feste (starre) Rollenverteilung
- Fester Interaktionsablauf, z.B. keine Zwischenergebnisse vorgesehen



[5] © Robert Tolksdorf, Berlin

Interaktionssemantik

- Blockierend/synchron: Client wartet auf Antwort
- Zurückgestellt synchron: Client schaut selber nach
- Asynchron: Ereignis oder Callback bei Ergebnis
- Einwege: Keine Antwort



[6] © Robert Tolksdorf, Berlin

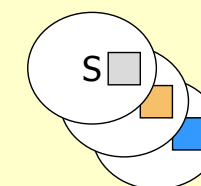
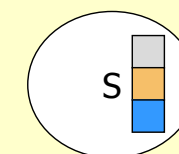
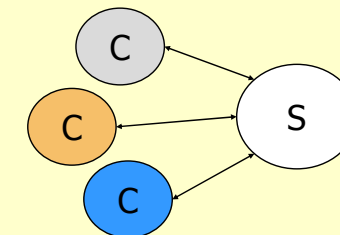
Serverzustand

- Angebotene Dienste lassen sich unterscheiden hinsichtlich ihrer Auswirkungen auf den Zustand des Servers:
 - Zustandsinvariante Dienste:
 - Bewirken keine Änderungen des Serverzustands
 - Beispiel: Webseitenabruf
 - Serverzustand kann durch andere Dienste geändert werden
 - Zustandsändernde Dienste:
 - Anfrage bewirkt neuen Serverzustand
 - Beispiel: Löschen auf einem FTP-Server
 - Reihenfolge der Dienstanforderungen relevant

[7] © Robert Tolksdorf, Berlin

Server

- Mehrere Clients stellen unabhängig voneinander Anfragen
- Server kann arbeiten
 - Sequentiell/Iterativ
 - Schlechtere Antwortzeiten
 - Einfacher
 - Parallel
 - Antwortzeiten günstiger
 - Synchronisationsaufwand



[8] © Robert Tolksdorf, Berlin

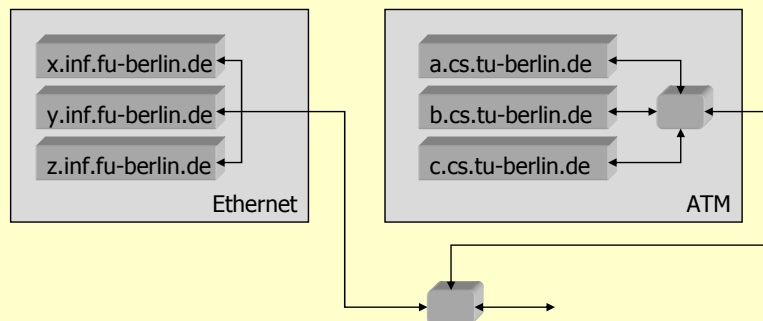
Mitteilungsaustausch Internet

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

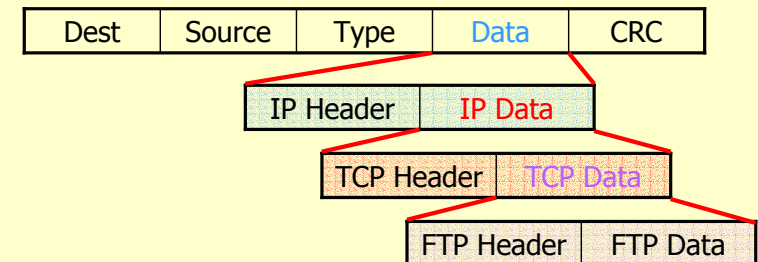
Internet als vernetzter Rechnerverbund

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

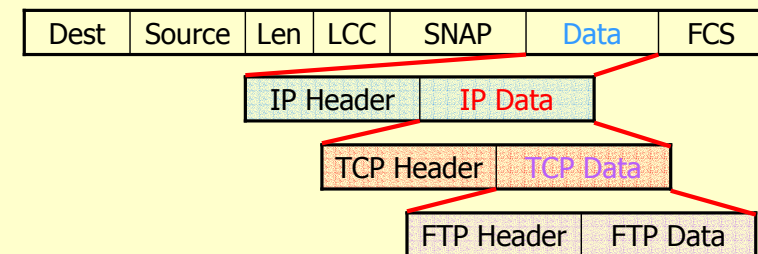


Enveloping / Encapsulating

- 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

[13] © Robert Tolksdorf, Berlin

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)

[14] © Robert Tolksdorf, Berlin

InetAddress API in Java (java.net.InetAddress)

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

[15] © Robert Tolksdorf, Berlin

InetAddress API in Java (java.net.InetAddress)

- 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)`

[16] © Robert Tolksdorf, Berlin

Beispiel

- Von Namen nach Punktnotation:

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

- Beispiel:

```
java IPNames www.inf.fu-berlin.de
160.45.117.8
```

[17] © Robert Tolksdorf, Berlin

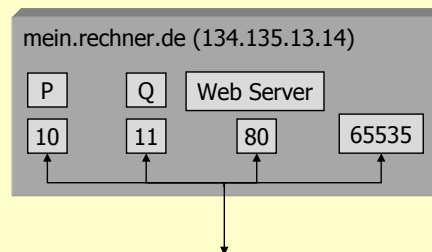
Transportprotokolle

- 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

[18] © Robert Tolksdorf, Berlin

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



[19] © Robert Tolksdorf, Berlin

TCP Transmission Control Protocol

[20] © Robert Tolksdorf, Berlin

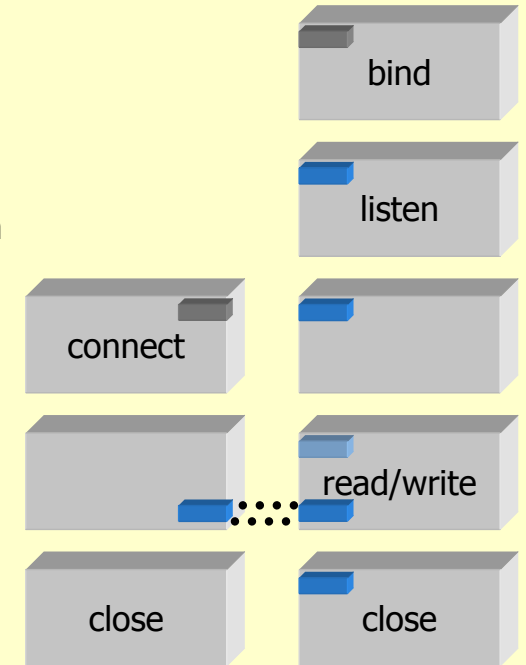
TCP Sockets

- 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

[21] © Robert Tolksdorf, Berlin

TCP Sockets

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



[22] © Robert Tolksdorf, Berlin

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]
}
```

[23] © Robert Tolksdorf, Berlin

Client

```
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();
}
```

[24] © Robert Tolksdorf, Berlin

Server

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

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

    ... [siehe nächste Folie]
}
```

[25] © Robert Tolksdorf, Berlin

Server

```
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);
        // "Pong" sagen
        out.println("Pong");
        // Alles schliessen und Schluss
        out.close();
        in.close();
        commSocket.close();
    }
}
```

[26] © Robert Tolksdorf, Berlin

Übersetzen und Ausführen

- javac PingClient.java
- javac PongServer.java

- dein.rechner.de:M:\files\Sockets>java PongServer
Got Ping
Got Ping
Got Ping

- M:\files\Sockets>java PingClient
Got Pong
M:\files\Sockets>java PingClient
Got Pong
M:\files\Sockets>java PingClient
Got Pong

[27] © Robert Tolksdorf, Berlin

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

[28] © Robert Tolksdorf, Berlin

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?

[29] © Robert Tolksdorf, Berlin

Socket API in Java (java.net.Socket)

- Optionen
 - `void setKeepAlive(boolean on)`
`boolean getKeepAlive()`
Socket durch Probepackete offen halten
 - `void setReuseAddress(boolean on)`
`boolean getReuseAddress()`
Socket nach Schliessen sofort wiederverwenden
 - `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

[30] © Robert Tolksdorf, Berlin

ServerSocket API in Java (java.net.ServerSocket)

- 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

[31] © Robert Tolksdorf, Berlin

ServerSocket API in Java (java.net.ServerSocket)

- 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?

[32] © Robert Tolksdorf, Berlin

ServerSocket API in Java (java.net.ServerSocket)

- 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

[33] © Robert Tolksdorf, Berlin

Weiteres Client-Beispiel: daytime Dienst nutzen

- 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 www.inf.fu-berlin.de`
Thu Sep 11 13:55:34 2003
- `>java DayTime www.w3.org`
Sun Mar 15 06:41:01 1998

[34] © Robert Tolksdorf, Berlin

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();
    }
}
```

[35] © Robert Tolksdorf, Berlin

UDP
User Datagram Protocol

[36] © Robert Tolksdorf, Berlin

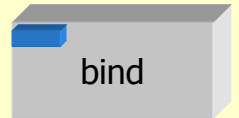
TCP vs. UDP

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

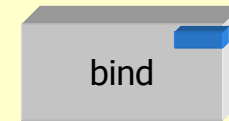
[37] © Robert Tolksdorf, Berlin

UDP Sockets

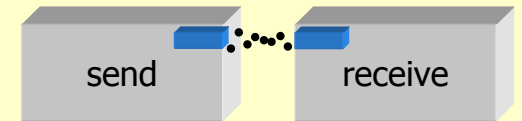
1. Server bindet Socket



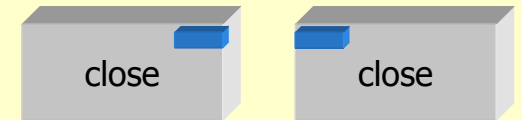
2. Client bindet Socket



3. Client und Server senden und empfangen



4. Sockets werden aufgegeben



[38] © Robert Tolksdorf, Berlin

Beispiel

- 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]
```

```
}
```

[39] © Robert Tolksdorf, Berlin

Server

```
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
```

```
    }
```

```
}
```

[40] © Robert Tolksdorf, Berlin

Client

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

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

[41] © Robert Tolksdorf, Berlin

Client

```
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();
}
```

[42] © Robert Tolksdorf, Berlin

Übersetzen und Ausführen

- javac UDPPingClient.java
- javac UDPPongServer.java

- M:\files\Sockets>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
- M:\files\Sockets>java UDPPingClient
Got Pong
M:\files\Sockets>java UDPPingClient
Got Pong
M:\files\Sockets>java UDPPingClient
Got Pong

[43] © Robert Tolksdorf, Berlin

DatagramSocket API in Java (java.net.DatagramSocket)

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

[44] © Robert Tolksdorf, Berlin

DatagramSocket API in Java (java.net.DatagramSocket)

▪ Verbindung

- `void send(DatagramPacket p)`
`void receive(DatagramPacket p)`
Datagramm senden/empfangen
- `void close()`
Socket schliessen

▪ Zustand

- `boolean isClosed()`
Geschlossen?

DatagramSocket API in Java (java.net.DatagramSocket)

▪ 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

DatagramPacket API in Java (java.net.DatagramPacket)

▪ 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

DatagramPacket API in Java (java.net.DatagramPacket)

- `void setAddress(InetAddress iaddr)`
`InetAddress getAddress()`
Empfänger/Absender Adresse setzen/lesen
- `void setPort(int ipp)`
`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

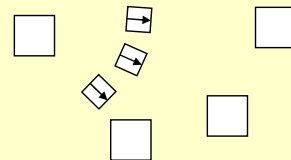
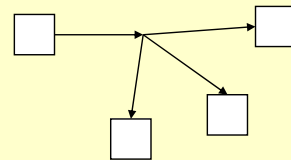
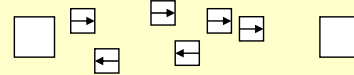
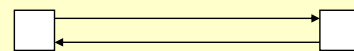
[49] © Robert Tolksdorf, Berlin

Multicast

[50] © Robert Tolksdorf, Berlin

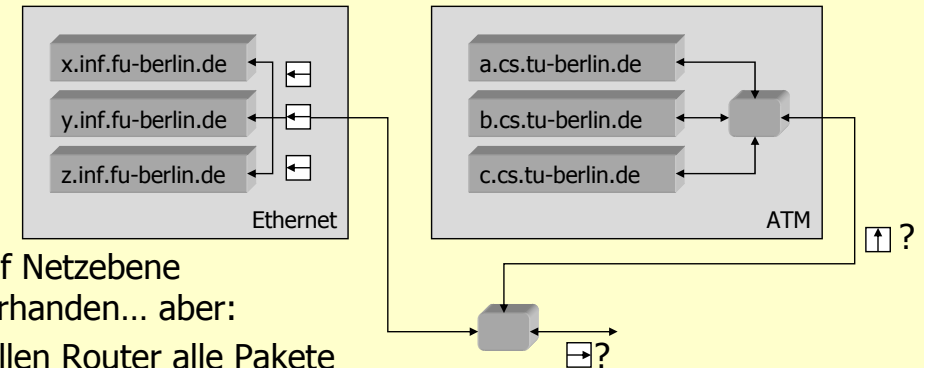
Transportmöglichkeiten

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



[51] © Robert Tolksdorf, Berlin

Wie weit soll transportiert werden?

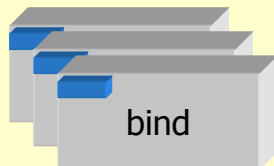


- Auf Netzebene vorhanden... aber:
- Sollen Router alle Pakete an alle weiterleiten?
- Wie wird Transport gesichert?
- Unklar: Viele IP-Protokollentwürfe
- Müsste in Routern implementiert werden
- Protocol Independent Multicast (PIM)
- Real-time Transport Protocol (RTP)
- Real-time Control Protocol (RTCP)
- 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)

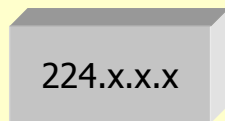
[52] © Robert Tolksdorf, Berlin

UDP Sockets

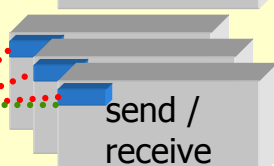
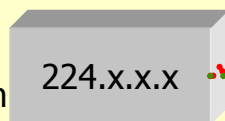
1. Teilnehmer binden Sockets



2. Teilnehmer treten Gruppe bei



3. Teilnehmer senden und empfangen



4. Teilnehmer verlassen Gruppe und geben Socket auf



[53] © Robert Tolksdorf, Berlin

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
```

[54] © Robert Tolksdorf, Berlin

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);  
    }  
}
```

[55] © Robert Tolksdorf, Berlin

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);
```

[56] © Robert Tolksdorf, Berlin

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);  
}  
}
```

[57] © Robert Tolksdorf, Berlin

Ausführen

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

[58] © Robert Tolksdorf, Berlin

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.

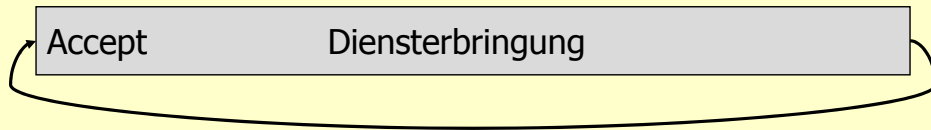
[59] © Robert Tolksdorf, Berlin

Nebenläufige Serverprogramme

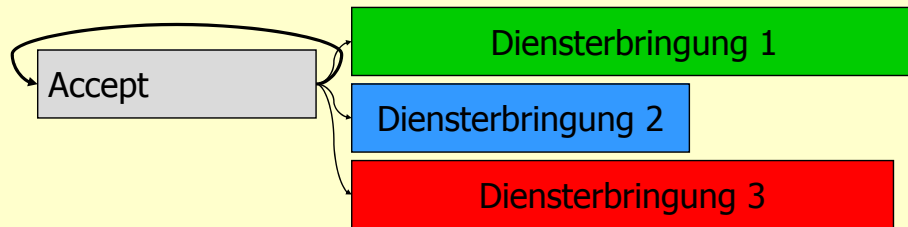
[60] © Robert Tolksdorf, Berlin

Multithreaded Server

- Das Serverprogramm ist während der Erbringung eines Dienstes nicht in der Lage, neue Verbindungen anzunehmen



- Abhilfe: Multithreading:



[61] © Robert Tolksdorf, Berlin

Threads als Java Objekte

- Ausführungszweig ist Objekt
- Muß java.lang.Thread erweitern (oder java.lang.Runnable implementieren)
- Muß eine Methode run() besitzen, besitzt Methode start()
- start() erzeugt neuen Ausführungszweig und ruft run() auf
- Thread terminiert, wenn run() beendet ist
- Objekt existiert weiter

[62] © Robert Tolksdorf, Berlin

Beispiel: Alle reden durcheinander

```
public class SayA extends Thread {
    public void run() {
        for(;;true;System.out.print("A"));
    }
}
```

```
public class SayB extends Thread {
    public void run() {
        for(;;true;System.out.print("B"));
    }
}
```

```
public class Talkshow {
    public static void main(String[] argv)
        SayA a=new SayA();
        SayB b=new SayB();
        a.start();
        b.start();
    }
}
```

```
BBBBBBBBBBBBBAAAAABBBBBBBBBBBB
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAB
BBBBBBBBBBBBBBBBBBBBBAAAAABBBBB
AAABBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBAAAAABAAABBBBBBBBB
```

[63] © Robert Tolksdorf, Berlin

Beispiel: Multithreaded Server

```
import java.net.*;
public class MultiServer {
    public static void main(String[] argv) {
        try{
            ServerSocket socket=
                new ServerSocket(3000);
            while (true) {
                Socket connection=socket.accept();
                Handler handler= new Handler(connection);
                handler.start();
            }
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }
}
```

```
import java.net.*;
public class Handler extends Thread {
    public Handler(Socket s) {}
    public void run() {...}
}
```

[64] © Robert Tolksdorf, Berlin

Zusammenfassung

Zusammenfassung

1. Client-Server als dominierendes Interaktionsmuster
2. Netzwerkkommunikation im Internet
 1. Sichten auf Internet
 2. Internet verbindet Netze
3. Internet Namen und Nummern
 1. Internet-Rechner haben Nummern und Namen
 2. DNS bildet zwischen ihnen ab
4. TCP Sockets
 1. Verbindungsorientiert
 2. Server bindet und lauscht
 3. Eigener Kommunikationssocket pro Verbindung
5. UDP Sockets
 1. Verbindungslos
 2. Nur Datagramme verschicken
 3. Nebenläufige Serverprogramme
 4. Server können mehrere Anfrage quasiparallel verarbeiten
 5. Sequentielles Accept, jeweiliger Start eines Prozesses für Anfrageverarbeitung
6. Multicast
 1. Versenden einer Mitteilung an viele Empfänger

Literatur

- 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

