Internettechnologien I

Sockets

Inhaltsverzeichnis

Socket-Programmierung

Socket: Schnittstelle auf einem Host zur Datenübertragung an andere Prozesse

- Sockets werden von Anwendungen erzeugt, verwendet und geschlossen
- Zwei Transportdienste werden über die Socket-API angesprochen:
  - Unzuverlässige Paketübertragung
  - Zuverlässige Übertragung von Datenströmen

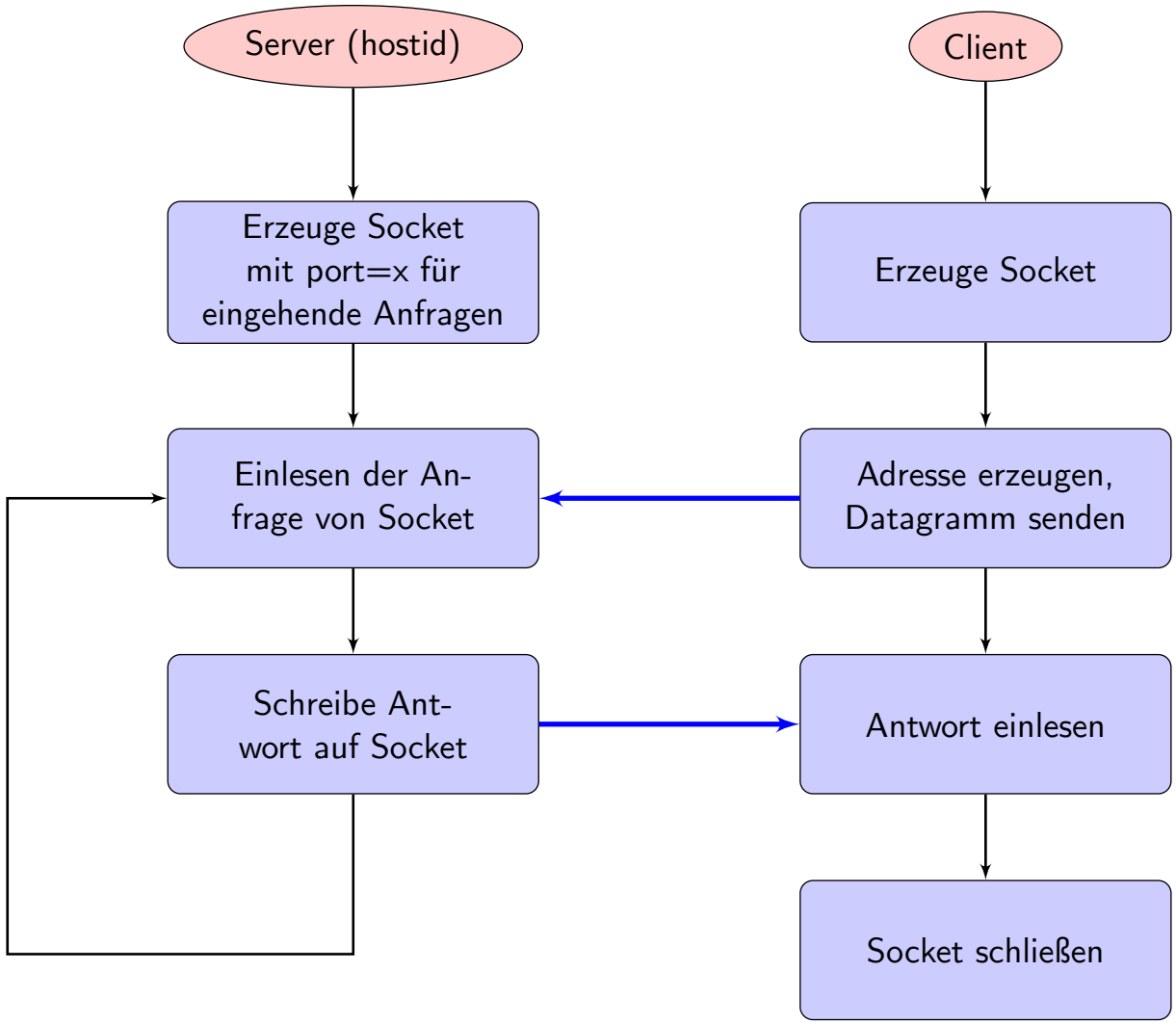
UDP-Sockets

Anwendungsperspektive

UDP stellt einen unzuverlässigen Transport einer Gruppe von Bytes ("Paket") zwischen Client und Server zur Verfügung

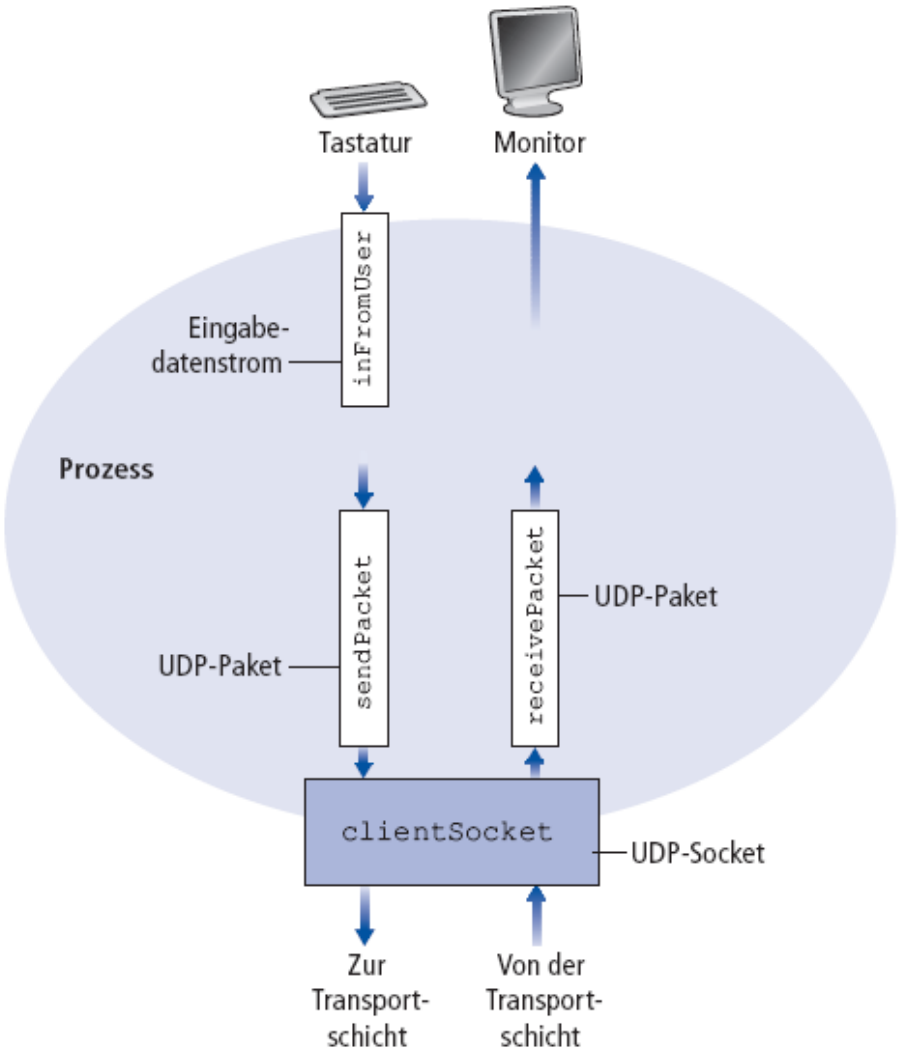
- Kein Verbindungsaufbau zwische Client und Server
- Server liest die IP-Adresse und Portnummer explizit aus dem empfangenen Paket aus
- UDP-Pakete können in falscher Reihenfolge empfangen werden oder verloren gehen

Socket-Programmierung UDP



UDP-Demo: Client

Datenströme müssen explizit in Pakete gewandelt werden



UDP-Demo: Client

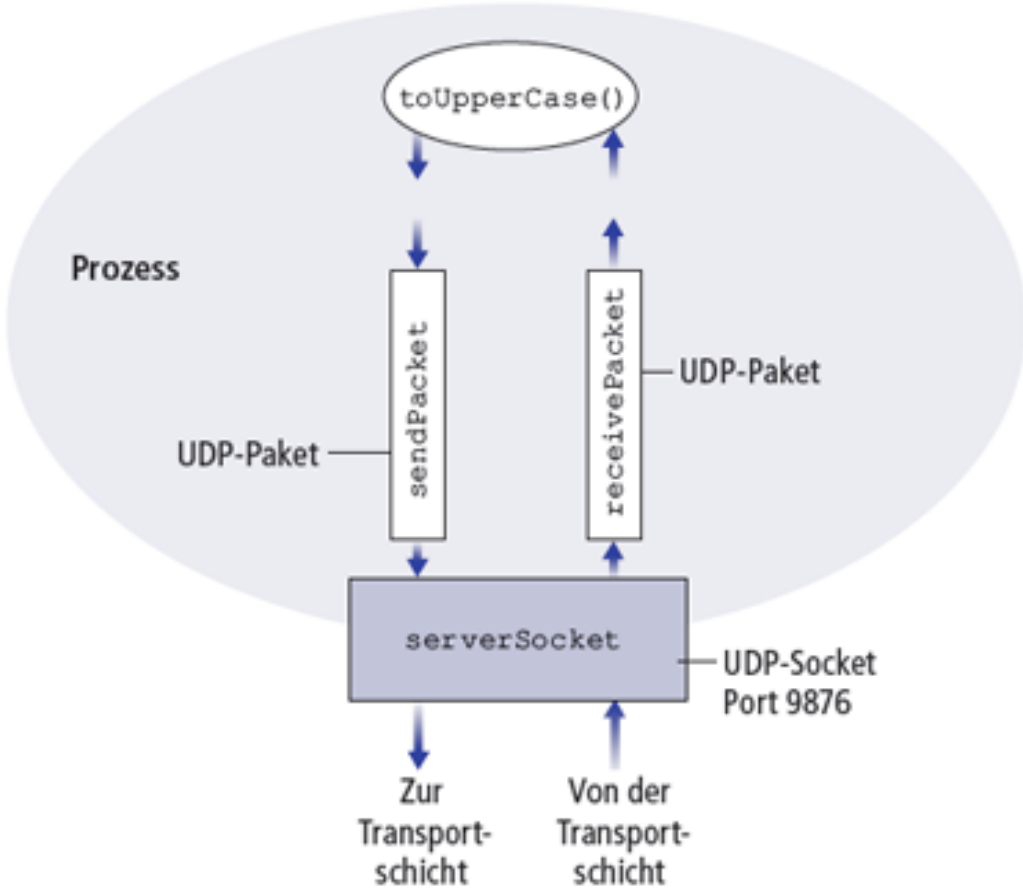
```
1  import java.io.*;
2  import java.net.*;
3
4  class UDPClient {
5      public static void main( String args[]) throws Exception
6      {
7        // Anlegen eines Eingabestroms (Buffered Reader: readLine)
8        BufferedReader inFromUser =
9          new BufferedReader( new InputStreamReader( System.in));
10
11       // Anlegen des Client-Sockets
12       DatagramSocket clientSocket = new DatagramSocket();
13
14       // Übersetzen des Hostnamens in IP-Adresse (static)
15       InetAddress IPAddress = InetAddress.getByName("hostname");
16
17       byte[] sendData = new byte[1024];
18       byte[] receiveData = new byte[1024];
19
20       String sentence = inFromUser.readLine();
21       sendData = sentence.getBytes();
```

UDP-Demo: Client cont.

```
1       // Paket anlegen, Daten, Länge. IP, Port
2       DatagramPacket sendPacket =
3        new DatagramPacket(sendData, sendData.length, IPAddress,9876);

5       // Paket an Server senden
6       clientSocket.send( sendPacket);

8       DatagramPacket receivePacket =
9         new DatagramPacket( receiveData, receiveData.length);

11      // Paket von Server empfangen: blockiert
12      clientSocket.receive( receivePacket);

14      String modifiedSentence =
15        new String( receivePacket.getData());

17      System.out.println("FROM SERVER:" + modifiedSentence);
18      clientSocket.close();
19    }
20  }
```

UDP-Demo: Server

Datenströme müssen explizit in Pakete gewandelt werden



UDP-Demo: Server

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

4  class UDPServer {
5    public static void main( String args[]) throws Exception
6    {
7      // Datagramm-Socket auf Port 9876 anlegen
8      DatagramSocket serverSocket = new DatagramSocket(9876);

10     byte[] receiveData = new byte[1024];
11     byte[] sendData = new byte[1024];

13     while(true)
14     {
15       // Speicher für Empfangspaket reservieren
16       DatagramPacket receivePacket =
17           new DatagramPacket( receiveData, receiveData.length);

19        // Paket empfangen
20        serverSocket.receive( receivePacket); // blockiert
```

UDP-Demo: Server cont.

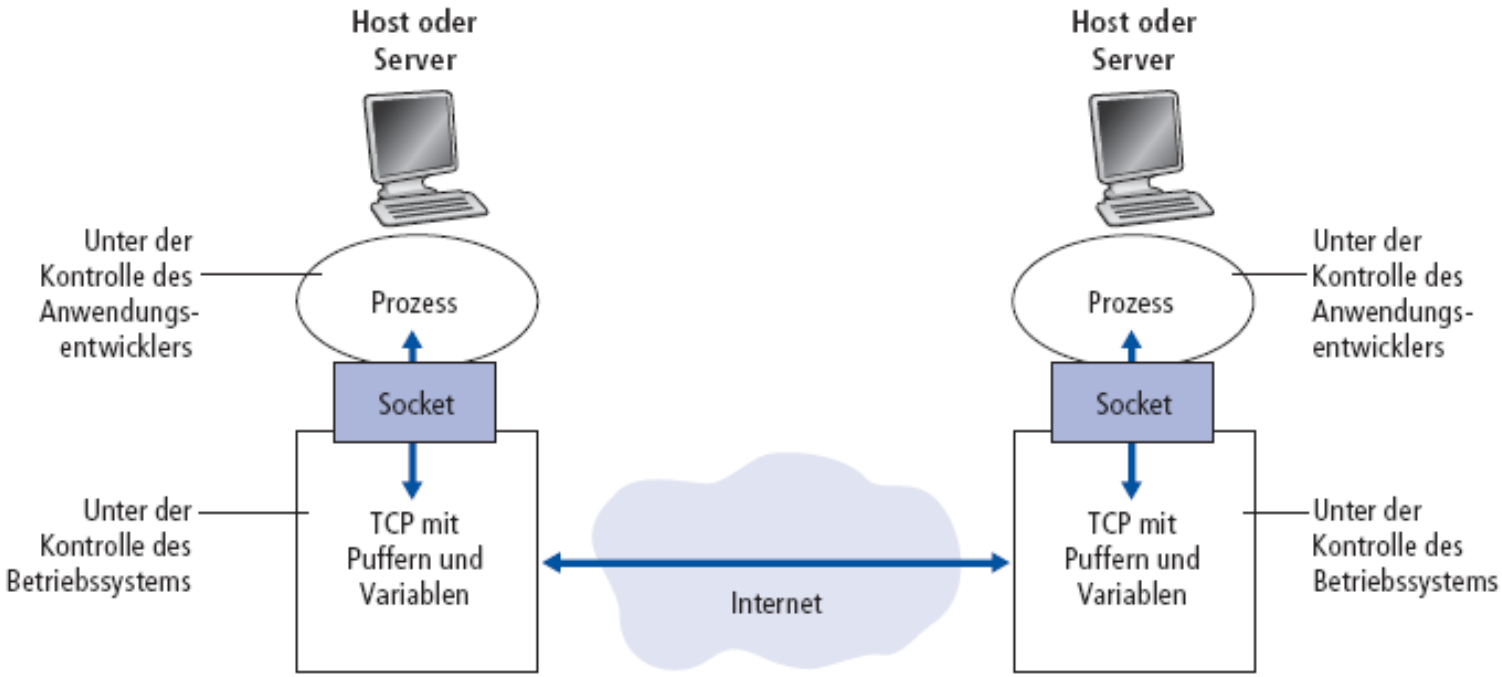
```
1         String sentence = new String( receivePacket.getData());

3         // IP und Port des Clients bestimmen
4         InetAddress IPAddress = receivePacket.getAddress();
5         int port = receivePacket.getPort();

7         String capitalizedSentence = sentence.toUpperCase();
8         sendData = capitalizedSentence.getBytes();

10        // Zu sendendes Paket anlegen
11        DatagramPacket sendPacket =
12         new DatagramPacket(sendData, sendData.length, IPAddress,port);

14        // Paket über Socket senden
15        serverSocket.send( sendPacket);
16      } // auf nächstes Paket warten
17    }
18  }
```

TCP-Sockets

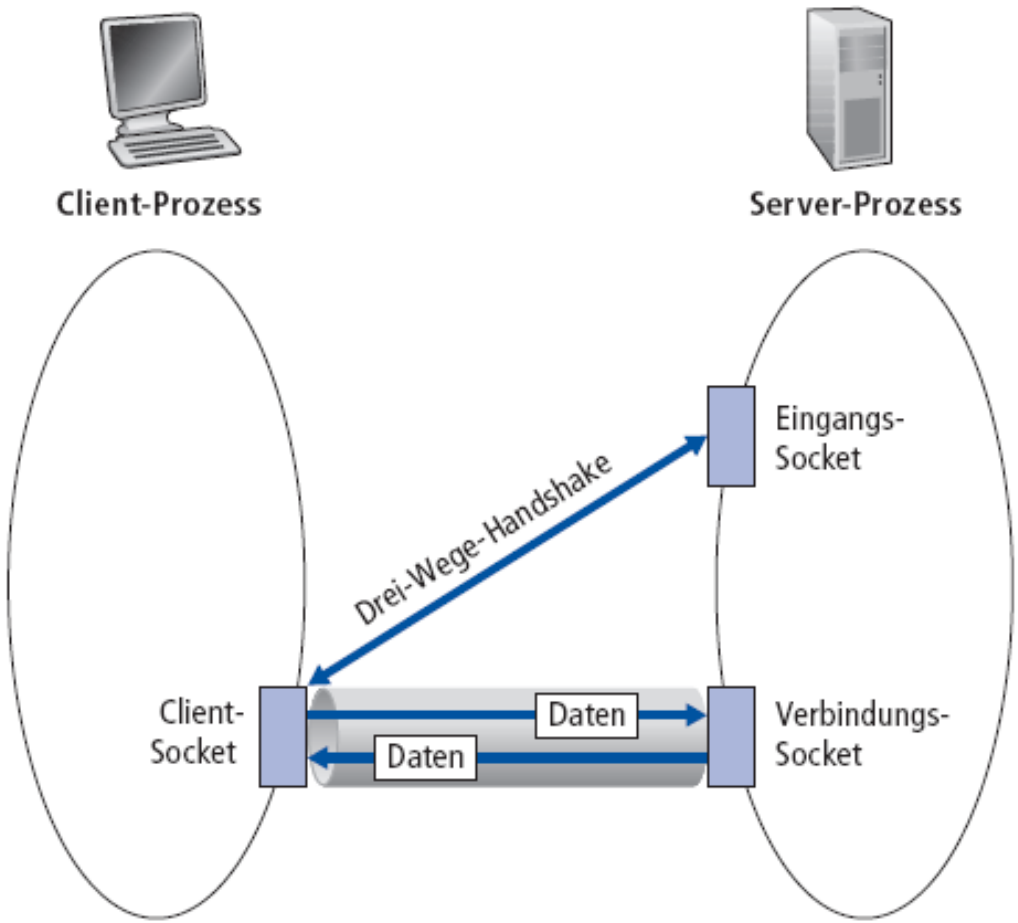
Anwendungsperspektive

TCP stellt einen zuverlässigen, reihenfolgeerhaltenden Transfer von Bytes zwischen Client und Server zur Verfügung

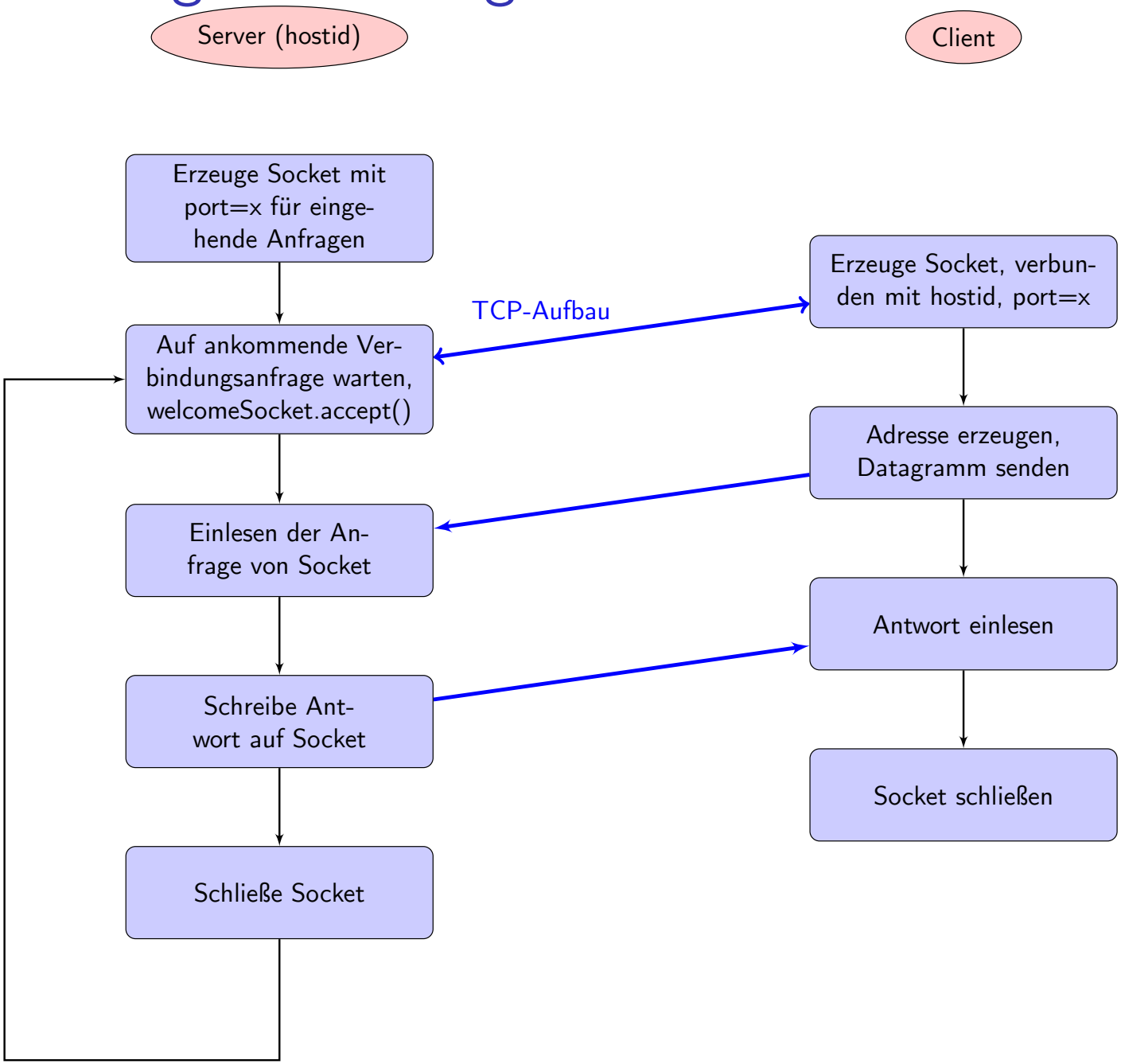
TCP-Sockets


Host oder Server
Host oder Server
Unter der Kontrolle des Anwendungs-entwicklers
Prozess
Socket
Unter der Kontrolle des Betriebssystems
TCP mit Puffern und Variablen
Internet
Prozess
Socket
TCP mit Puffern und Variablen
Unter der Kontrolle des Anwendungs-entwicklers
Unter der Kontrolle des Betriebssystems

TCP: Dreiwege-Handshake

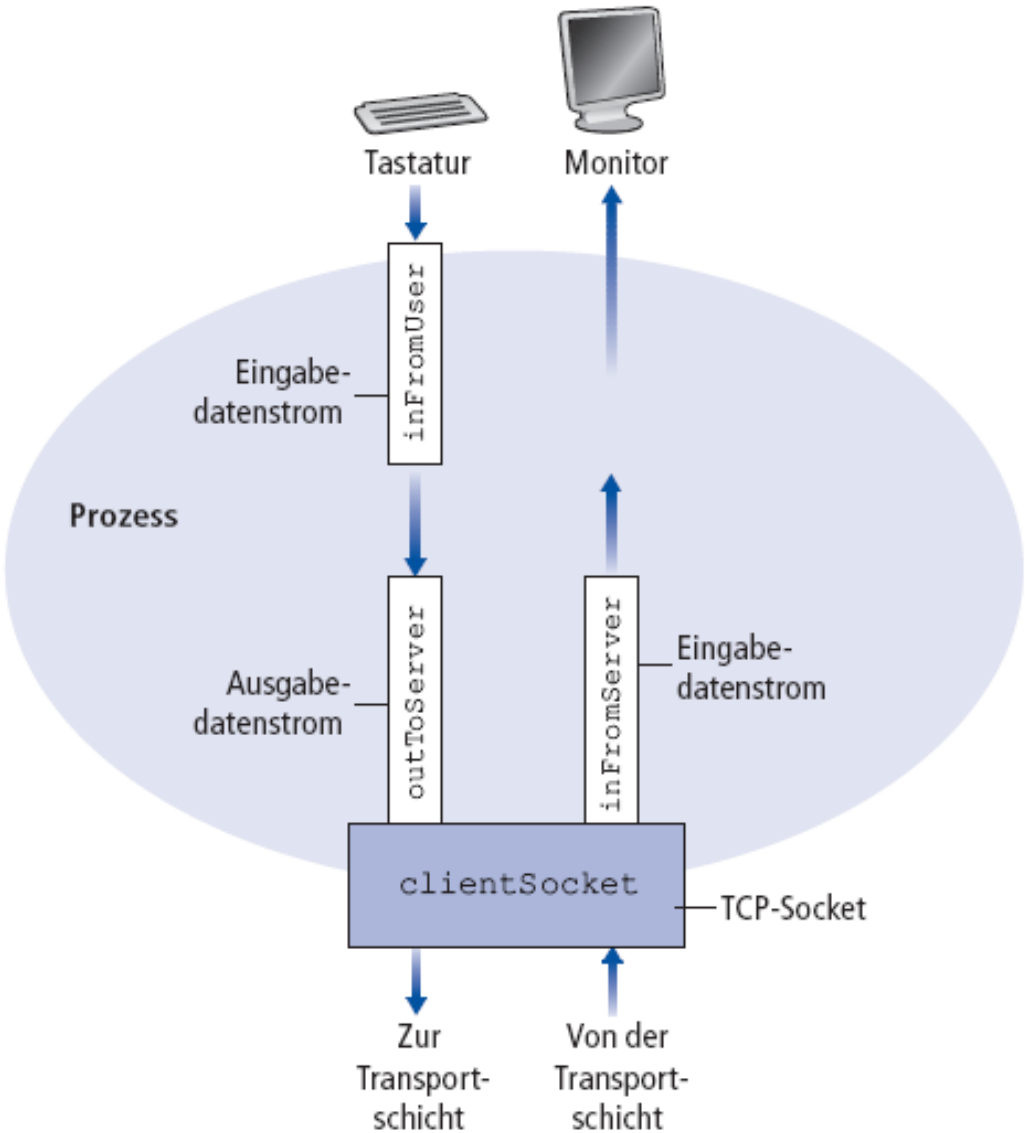


Socket-Programmierung TCP



TCP-Demo: Datenströme

Eingabestrom - Quelle:Tastatur, Ausgabestrom - Senke:Socket



Socketprogrammierung mit TCP

1. Client liest Zeilen von der Standardeingabe (inFromUser Strom) und sendet diese über einen Socket (outToServer Strom) zum Server
2. Server liest die Zeile vom Socket
3. Server konvertiert die Zeile in Großbuchstaben und sendet sie zum Client zurück
4. Client liest die konvertierte Zeile vom Socket (inFromServer Strom) und gibt sie aus

TCP-Demo: Client

```
1  import java.io.*;
2  import java.net.*;
3
4  class TCPClient {
5      public static void main( String argv[]) throws Exception
6      {
7          String sentence;
8          String modifiedSentence;
9
10         // Eingabestrom anlegen (Buffered Reader hat readLine())
11         BufferedReader inFromUser =
12           new BufferedReader( new InputStreamReader( System.in));
13
14         // Client-Socket anlegen und mit Server verbinden
15         Socket clientSocket = new Socket( "hostname", 6789);
16
17         // Ausgabestrom anlegen + mit Socket verbinden
18         DataOutputStream outToServer =
19           new DataOutputStream( clientSocket.getOutputStream());
```

TCP-Demo: Client cont.

```
1       // Eingabestrom anlegen, mit Socket verbinden
2       BufferedReader inFromServer = new BufferedReader( new
3             InputStreamReader( clientSocket.getInputStream()));

5       // Zeile einlesen -> blockiert
6       sentence = inFromUser.readLine();

8       // Zeile an Server senden
9       outToServer.writeBytes(sentence + '\n');

11      // Zeile von Server lesen -> blockiert
12      modifiedSentence = inFromServer.readLine();

14      System.out.println("FROM SERVER: " + modifiedSentence);

16      clientSocket.close();
17    }
18  }
```

TCP-Demo: Server

```
1  import java.io.*;
2  import java.net.*;
3
4  class TCPServer {
5
6    public static void main(String argv[]) throws Exception
7      {
8        String clientSentence;
9        String capitalizedSentence;
10
11       // Socket für Anfragen auf Port 6789
12       ServerSocket welcomeSocket = new ServerSocket(6789);
13
14       while(true) {
15         // Neues Verbindungssocket -> blockiert!
16         Socket connectionSocket = welcomeSocket.accept();
17
18         // Eingabestrom anlegen + mit Socket verbinden
19         BufferedReader inFromClient = new BufferedReader( new
20             InputStreamReader( connectionSocket.getInputStream()));
```

TCP-Demo: Server cont.

```
1       // Ausgabestrom anlegen, mit Socket verbinden
2       DataOutputStream outToClient =
3         new DataOutputStream( connectionSocket.getOutputStream());

5       // Zeile vom Socket einlesen
6       clientSentence = inFromClient.readLine();

8       capitalizedSentence = clientSentence.toUpperCase() + '\n';

10      // Zeile an Client schicken
11      outToClient.writeBytes( capitalizedSentence);
12    } // auf nächsten Client warten
13  }
14 }
```

Zusammenfassung

TCP- und UDP-Sockets sind die Schnittstelle zur Anwendung

TCP-Klassen

ServerSocket Socket für die Verbindungsaufnahme

Socket Socket zur Datenübertragung

UDP-Klassen

DatagramSocket UDP-Socket

DatagramPacket Datenstruktur für ein Datagramm

Literatur

- Kurose,Ross „Computernetzwerke", Person