LANDESBERUFSSCHULE 4 SALZBURG

Informatik 3

Grundlagen Netzwerkprogrammierung

Version 2.3 – 04.06.2020

Dieses Skript dient als zusätzliche Lernunterlage für Informatik

1 Inhalt

1 Lernziele und Kompetenzencheck

- Ich kann den TCP/IP Protokoll-Stack erläutern
- Ich kann Transportprotokolle nennen und deren Eigenschaften erklären
- Ich kann den Verbindungsaufbau mit TCP erklären und skizzieren
- Ich kann den Begriff „Socket“ erläutern und deren Verwendung in der Informatik formulieren
- Ich kann mit Java eine Verbindung zu einem Serverdienst aufbauen und Nachrichten versenden

2 Netzwerkprogrammierung allgemein

Kommunikation über Netzwerke kann in drei Kategorien eingeteilt werden:

- Nachrichten-basierte Kommunikation
- Objekt-basierte Kommunikation
- Web-basierte Kommunikation

Nachrichten-basierende Verfahren sind z.B. synchrone Nachrichtenübertragungen über TCP- und UDP-Sockets. Der Objekt-basierende Ansatz ermöglicht die Kommunikation zwischen Java-Objekten. Die Web-basierten Verfahren werden normal mittels http oder https Verbindungen realisiert.

Die Begriffe Client und Server sind allgegenwärtig, allerdings werden diese oft falsch definiert. Ein Applikationsentwickler versteht als Server eine Software, die im Netzwerk einen Dienst zur Verfügung stellt. Der Client ist dementsprechend eine Software, die diesen Dienst nutzt.

3 Grundlagen „Verteilte Systeme“

Ein „Verteiltes System“ besteht aus Komponenten, die räumlich getrennte (Netzwerk)Knoten besitzen, welche miteinander kooperieren und kommunizieren, um eine gemeinsame Aufgabe durchzuführen. Allgemein kann die Struktur physisch (Aufbau des Netzwerkes/Computer) und logisch (Aufbau einer Software) betrachtet werden. Die Schwierigkeit besteht darin diese beiden Sichten so abzubilden, damit maximaler Nutzen entsteht.

In Abbildung 1: Vergleich Netzwerkstruktur wird ein Vergleichsbeispiel dargestellt. Für den Benutzer muss der Entwurf und somit der physische Aufbau verborgen bleiben. Dieser soll unabhängig von der logischen Schicht verändert werden können.

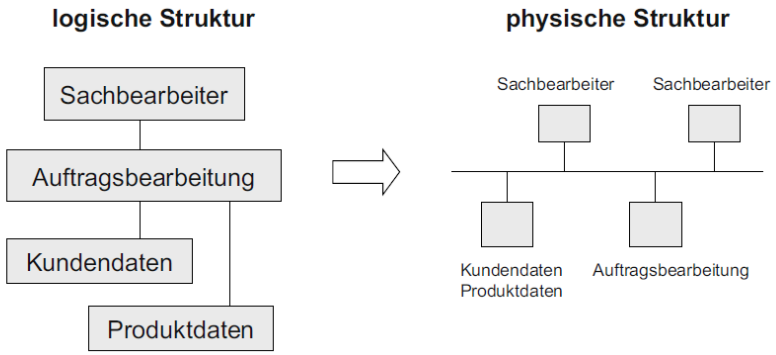


Abbildung 1: Vergleich Netzwerkstruktur

Der Vorteil von verteilten Systemen wird nachstehend angeführt:

- Wirtschaftlichkeit
  - teure Ressourcen können gemeinsam genutzt werden
- Lastverteilung
  - parallele Verarbeitung und kürzere Verarbeitungszeit
- Skalierbarkeit
  - einzelne Komponenten können besser angepasst werden
- Fehlertoleranz
  - beim Ausfall eines Systems, bleiben andere betriebsbereit

Als Nachteil kann die Komplexität des Systems sowie die Sicherheitsrisiken auf den verteilten Knotenpunkten gesehen werden.

3.1 Client/Server-Modell

Ein Client stellt eine Anfrage an den Server, der bestimmte Dienste anbietet. Der Server arbeitet die Anfrage ab und liefert dem Client das Ergebnis. Ein Kommunikationsdienst kann einen Client und Server verbinden. Somit wird als minimale Anforderung ein Client (Auftraggeber), ein Server (Auftragnehmer) und ein Kommunikationsdienst (Netzwerk, Bus) benötigt.

Client und Server müssen nicht räumlich getrennt sein, vielmehr können diese beiden Dienste am selben Computer arbeiten.

3.1.1 Server Merkmale

Ein Server bietet einen bestimmten Dienst an und kann mehrere Clients normalerweise gleichzeitig bedienen. Ein Server wartet immer passiv (listen) auf einem bestimmten Port auf die Verbindungsaufnahe eines Clients. Da Server eigenständige Programme sind, können mehrere Instanzen unabhängig voneinander laufen. Der Parallelbetrieb ist ein wichtiges Merkmal, damit ein Client nicht zu lange warten muss.

Bei der Kommunikation unterscheidet man zwischen synchroner- und asynchroner-Kommunikation (Abbildung 2: Arten der Kommunikation).

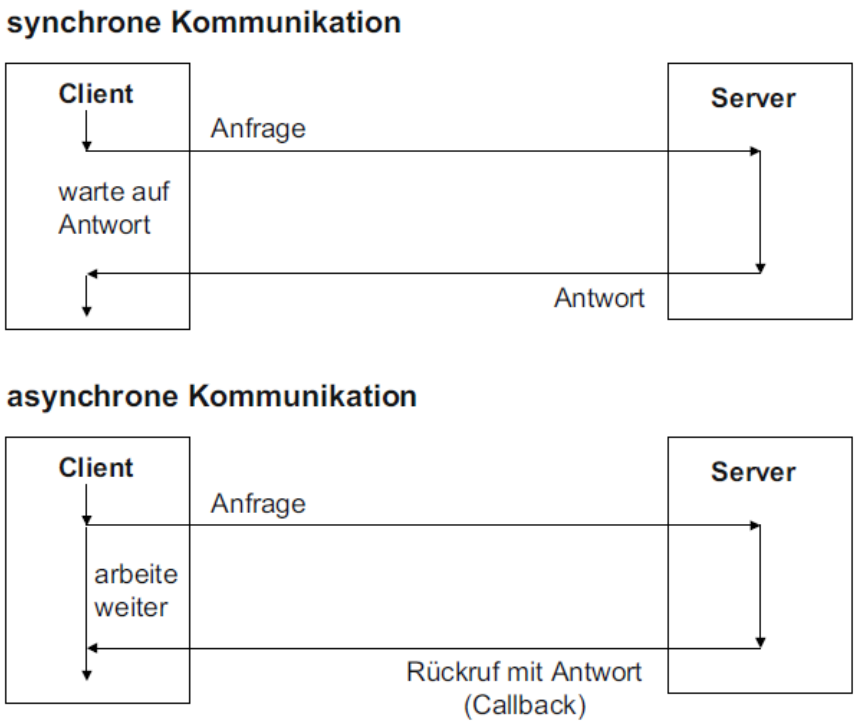


Abbildung 2: Arten der Kommunikation

Bei der synchronen Kommunikation wartet der Client nach Absenden der Anfrage an den Server so lange, bis er eine Antwort erhält.

Im asynchronen Fall sendet der Client die Anfrage an den Server und arbeitet sofort weiter. Beim Eintreffen der Antwort werden dann bestimmte Ereignisbehandlungsroutinen beim Client aufgerufen.

3.2 Netzwerkschichten

Die Kommunikation zwischen Endgeräten in einem Netzwerk läuft über das OSI oder das TCP-IP Schichtenmodell ab.

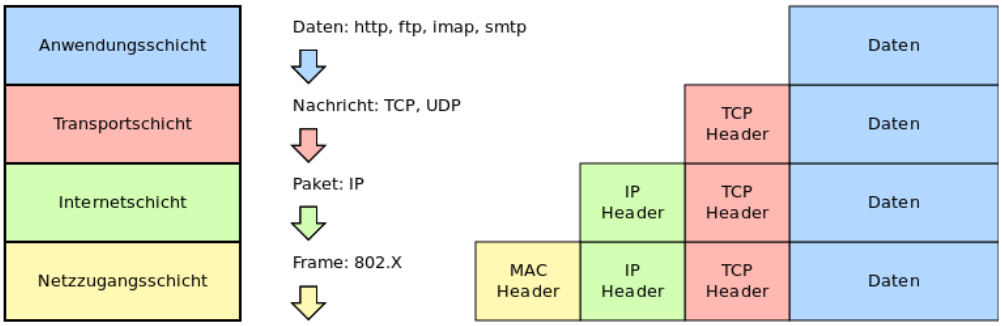


Abbildung 3:TCP-IP Schichtenmodell

Für die Netzwerkprogrammierung ist vor allem die Transportschicht mit den beiden Transportprotokollen UDP und TCP relevant.

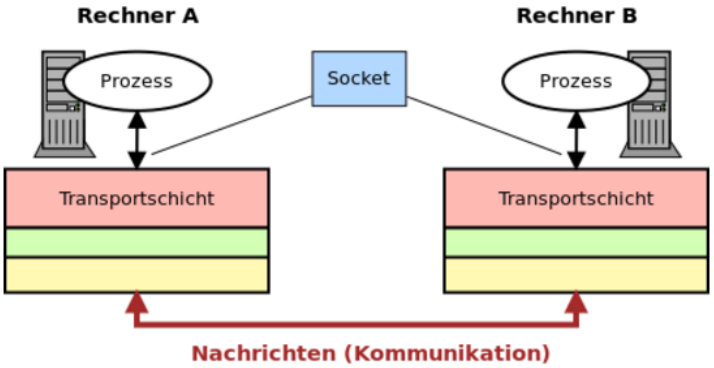


Abbildung 4: Kommunikation von 2 Endgeräten

3.3 Netzwerkprotokolle

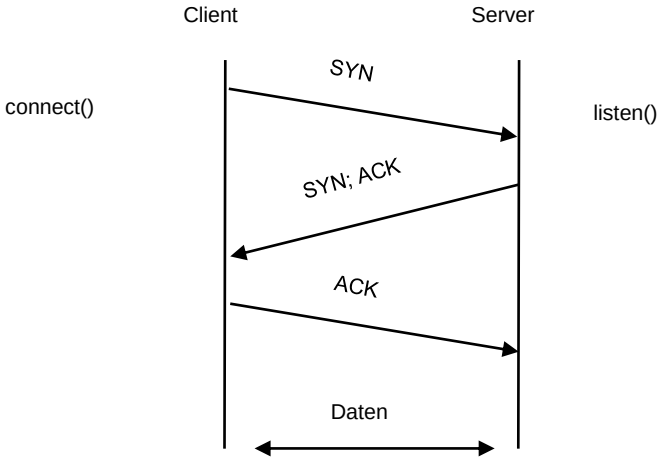
Transmission Control Protocol (TCP) ist ein verbindungsorientierter Dienst und stellt eine sichere und fehlerfreie Verbindung zwischen den Kommunikationsendpunkten dar. Die Endpunkte werden mit Sockets beschrieben. Daten können bidirektional übertragen werden und bauen auf dem IP auf. Am Ende der Kommunikation wird die Verbindung wieder abgebaut. TCP gilt als zuverlässig (Socket, ServerSocket, URL).

User Data Protocol (UDP) ist ein verbindungsloser Dienst. Es wird in dem Sinne keine Verbindung wie bei TCP aufgebaut und es gibt keine Bestätigungen über den Empfang der Daten. Die Anwendung muss selbst dafür sorgen, dass die

Datagramme beim Empfänger ankommen. UDP gilt als unzuverlässig, aber schnell. (DatagramPacket, DatagramSocket, MulticastSocket)

3.4 Ablauf einer Verbindung

Three-Way-Handshake: Der Client initiiert eine Verbindung zum Server. Der Server signalisiert, dass er zur Verbindung bereit ist. Der Client bestätigt die Antwort des Servers. Die Verbindung ist aufgebaut und der Client kann Daten an den Server senden.



Der Client sendet eine Frage (Request) an den Server, dieser sendet eine Antwort (Response) zurück. Grundlage ist die Verfügbarkeit von Sockets, die eine Kommunikation, ähnlich dem Datenaustausch mit einem Dateisystem erlauben. Man schreibt auf den Socket oder liest vom Socket, als ob es sich um eine einfache Text-Datei handeln würde. Die tatsächliche Umsetzung übernehmen Bibliotheken des JRE Sockets. Sie bieten eine stream basierte Schnittstelle zweier Kommunikationsendpunkte.

Sockets werden durch zwei Parameter festgelegt:

- die IP-Adresse (oder der Name) des Computers 193.12.34.5
- eine gemeinsame Portnummer 80

3.5 Serialisierung

Mit Serialisierung können Objekte zum vorübergehenden speichern in Dateien abgelegt werden (Persistenz). Der Zustand des Objekts wird in Bytes umgewandelt (Serialisierung) und bei Bedarf wieder rekonstruiert (Deserialisierung). Voraussetzung ist, dass das Interface `java.io.Serializable` implementiert ist. Dieses Interface besitzt keine Methoden.

Das serialisierte Objekt muss kompatibel zu dem Objekt sein, das deserialisiert wird. Alle Attribute müssen vorhanden sein. Im nächsten Beispiel stellt eine einfache Klasse Kunde die Grundlage zur Serialisierung dar.

3.5.1 Beispiel Kunde:

```
public class Kunde implements java.io.Serializable {

   private static final long serialVersionUID = 1L;
   private String name;
   private String adresse;

   public Kunde(String name, String adresse) {
      this.name = name;
      this.adresse = adresse;
   }
   public void setName(String name) {
      this.name = name;
   }
   public String getName() {
      return name;
   }
   public void setAdresse(String adresse) {
      this.adresse = adresse;
   }
   public String getAdresse() { return adresse;
   }
}
```

Vor der Serialisierung wird eine ArrayList mit verschiedenen Informationen angelegt. Im Anschluss werden die Ströme konfiguriert und das Objekt in eine Datei geschrieben.

```
Kunde k1 = new Kunde("Meier, Hugo", "Hauptstr. 12, 5020 Salzburg");
Kunde k2 = new Kunde("Schmitz, Otto", "Dorfstr. 5, 1010 Wien");
ArrayList<Kunde> kunden = new ArrayList<Kunde>();
kunden.add(k1);
kunden.add(k2);
kunden.add(k2);

ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream("kunden.ser"));
out.writeObject(kunden);
out.flush();
```

Bei der Deserialisierung wird einfach die Datei geladen und der Inhalt in die ArrayList wieder eingefügt.

```
try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(
      "kunden.ser"))) {
   @SuppressWarnings("unchecked")
   ArrayList<Kunde> list = (ArrayList<Kunde>) in.readObject();
   for (Kunde k : list) {
      System.out.println(k.getName() + "; " + k.getAdresse());
   }
```

Somit ist es möglich ein Objekt über eine Netzwerkverbindung mittels Serialisierung und Deserialisierung auszutauschen. Eine Weitere Möglichkeit bietet ein JSON-Dokument. Auch damit ist es möglich einfache Objekte in eine Datei zu schreiben und über das Netzwerk zu versenden.

3.5.2 JSON

Ein JSON-Dokument stellt eine Menge von Name- Wert-Paaren (Objekte) in einer geordneten Liste dar.

Ein JSON-Objekt beginnt und endet mit einer geschwungenen Kammer ({ }).

Jedem Namen in doppelten Anführungszeichen (“ name“) folgt ein Doppelpunkt (“Name“ : “Wert“). Die einzelnen Daten sind mit einem Komma (,) getrennt. Das Nachstehende Beipsiel beschreibt eine Person mit Id, Vor- und Nachnamen, Alter und den Hobbys.

3.5.2.1 Beispiel JSON:

```
{
"id": 4711,
"firstname": "Max",
"lastname": "Muster“
"age": 28,
"hobbies": [
"Sport",
"Lesen",
"Musik"
]
}
```

3.5.3 JsonBuilder

Java stellt eine API für JSON-Binding zur Verfügung. Diese sind im Beispielprojekt im Ordner libs/json verfügbar. Der JSONBuilder unterstützt das Lesen (serialisieren – *fromJson()*)sowie das Schreiben (deserialisieren – *toJson()*) von JSON Objekten.

```
/** Erzeugen des JsonBuilders */
   Jsonb jsonb = JsonbBuilder.create();
   /** schreiben in eine Datei */
   MyJSON p = jsonb.fromJson(new FileInputStream("person.json"), MyJSON.class);
/** lesen von einer Datei*/
jsonb.toJson(p, new FileOutputStream("person.json"));
```

4 IP-Adressen mit Java verwalten

Java bietet Klassen und Methoden zur Verwaltung von IP-Adressen und Sockets an. Die Klasse java.net.InetAddress kann IPv4 sowie IPv6 Adressen repräsentieren. Diese bietet 5 (statische) Methoden zum Erzeugen der Objekte an:

static InetAddress getLocalHost() //ermittelt die IP des Rechners

static InetAddress getByName(String host) //löst den Hostnamen auf

static InetAddress[] getAllByName(String host) // Rückgabewert ist ein Array mit allen Hostnamen

String getHostAddress() //Rückgabewert ist die IP-Adresse des Hosts

String getHostName() // Rückgabewert istder Hostname zur IP-Adresse

Das nachstehende Beispiel zeigt den Hostnamen und die IP-Adresse des eigenen Rechners einer Netzwerkkarte an.

```
public void showLocalAddress() throws UnknownHostException {
   InetAddress address = InetAddress.getLocalHost();
   System.out.printf("%s/%s%n",
         address.getHostName(),
         address.getHostAddress());
}
```

Zum Auflisten der Netzwerkschnittstellen kann die Klasse NetworkInterface verwendet werden.

Wenn man die IP-Adresse eines Domainnamen eruieren möchte kann das nachstehende Beispiel verwendet werden.

```
public void myLookup(String name) throws UnknownHostException {
   InetAddress address = InetAddress.getByName(name);
   System.out.printf("suche %s -> %s/%s%n",
         name,
         address.getHostName(),
         address.getHostAddress());
}
```

4.1.1 Übung

Die Übung wird am Moodle zur Verfügung gestellt.

5 Sockets

Java-Programme verwenden zur Kommunikation mittels UDP und TCP Sockets. Diese stellen jeweils einen Endpunkt einer Kommunikationsbeziehung dar. Die Klasse java.net.InetSocketAddress repräsentiert eine IP-Socket-Adresse.

InetSocketAddress(InetAddress addr, int port)

InetSocketAddress(String hostname, int port)

Es existieren drei Methoden die den Port, die IP-Adresse und den Hostnamen des verwendeten Sockets zurückgeben. Das nachstehende Beispiel zeigt die Möglichkeiten der Klasse InetSocketAddress.

```
public void createSocketAdress(){
   InetSocketAddress socketAddress = new InetSocketAddress(443);
   output(socketAddress);
   try {
      InetAddress address = InetAddress.getByName("lbs4.salzburg.at");
      socketAddress = new InetSocketAddress(address, 443);
      output(socketAddress);
   } catch (UnknownHostException e) {
      System.out.println(e);
   }
   socketAddress = new InetSocketAddress("www.salzburg.at", 443);
   output(socketAddress);
   socketAddress = new InetSocketAddress("localhost", 80);
   output(socketAddress);

}
```

5.1 Sockets mit UDP

UDP ist ein einfaches Protokoll und kann keine zuverlässige Übertragung gewährleisten. Der Vorteil gegenüber TCP liegt in geringeren Headerdateien. UDP besitzt keine Kontrollmechanismen zur Datenübertragung. Bekannte Dienste sind DNS, SNMP oder DHCP.

Die Klasse java.net.DatagramSocket repräsentiert ein UDP-Socket zum Senden und Empfangen von Datagrammen. Die logische Übertragung findet auf der Transportschicht statt. Die wichtigsten DatagramSocket-Methoden sind:

- void close()

- getLocalAddress() und setLocalAddress(InetAddress address)
- getLocalPort() und setLocalPort()
- setAddress(InetAddress address) und setPort() // Kommunikationspartner
- setSocketAddress(InetAddress address) und getSocketAddress()
- setData(byte[] buf) // für den Datenbuffer

Zum Testen verwenden Sie das fertige Beispiel am Moodle. Kompilieren Sie den Client und Server und starten Sie die beiden Programme mittels der *.bat Dateien. Im nachstehenden Beispiel ist die Bearbeitungszeit des Servers sehr kurz, daher reicht eine iterative Implementierung (Paket empfangen, verarbeiten, senden). Die Nachrichten werden mittels JSON (JavaScript Objekt Notation) Format gesendet.

5.2 Übung UDP-Socket

Die Übungsanweisung wird am Moodle zur Verfügung gestellt.