Technische Universität Braunschweig

Institut für Betriebssysteme und Rechnerverbund

Verteilte Systeme - 1. Übung

Dr. Jens Brandt

Sommersemester 2011

1. Rechnerverbünde

- **Kommunikationsverbund:**
  Beispiele: E-Mail (SMTP, POP/IMAP), Instant Messaging (XMPP, IRC, ICQ,...), Newsgroups (NNTP)
- **Informationsverbund:**
  Beispiele: WWW / Intranets (HTTP), Newsgroups, Videostreaming (Mediatheken; z.B. RTP, RTMP)
- **Datenverbund:**
  Beispiele: Datenbank-Replikation/-Partitionierung, Verteilte Versionsverwaltung (Git, Mercurial,. . .)
- **Lastverbund:**
  Beispiel: Load-Balancing

1. Rechnerverbünde (cont.)

- **Leistungsverbund:**
  Beispiele: Cloud-Computing (Amazon EC2, Google App Engine, Microsoft Azure), SETI/Folding@Home, map/reduce-Systeme (Apache Hadoop), Message Passing Interface (MPI)
- **Wartungsverbund:**
  Beispiele: SNMP, Nagios, Update-Systeme (Aptitude/YUM/emerge/. . . , Windows Server Update Services), Secure-Shell (SSH), Desktop-Sharing-Systeme (VNC, Windows Remote Desktop, KVM over IP)
- **Funktionsverbund:**
  Beispiele: Netzwerkdrucker (Internet Printing Protocol IPP, Server Message Block SMB), Service-orientierte Architekturen (SOA)

Eigenschaften Verteilter Systeme

- Transparenz
  - Das verteilte System erscheint als einheitliches System, die Verteiltheit bleibt verborgen
- Skalierbarkeit
  - Verschiedene Dimensionen: Größe, geographische Verteilung, administrative Skalierbarkeit
- Offenheit
  - Offene Schnittstellen und Standards
- Nebenläufigkeit
  - Gleichzeitige Nutzung durch unterschiedliche Nutzer
- Fehlertoleranz
  - Verfügbarkeit auch bei Fehlern in einzelnen Komponenten
- Sicherheit
  - Vertraulichkeit, Integrität, Authentizität und Verfügbarkeit

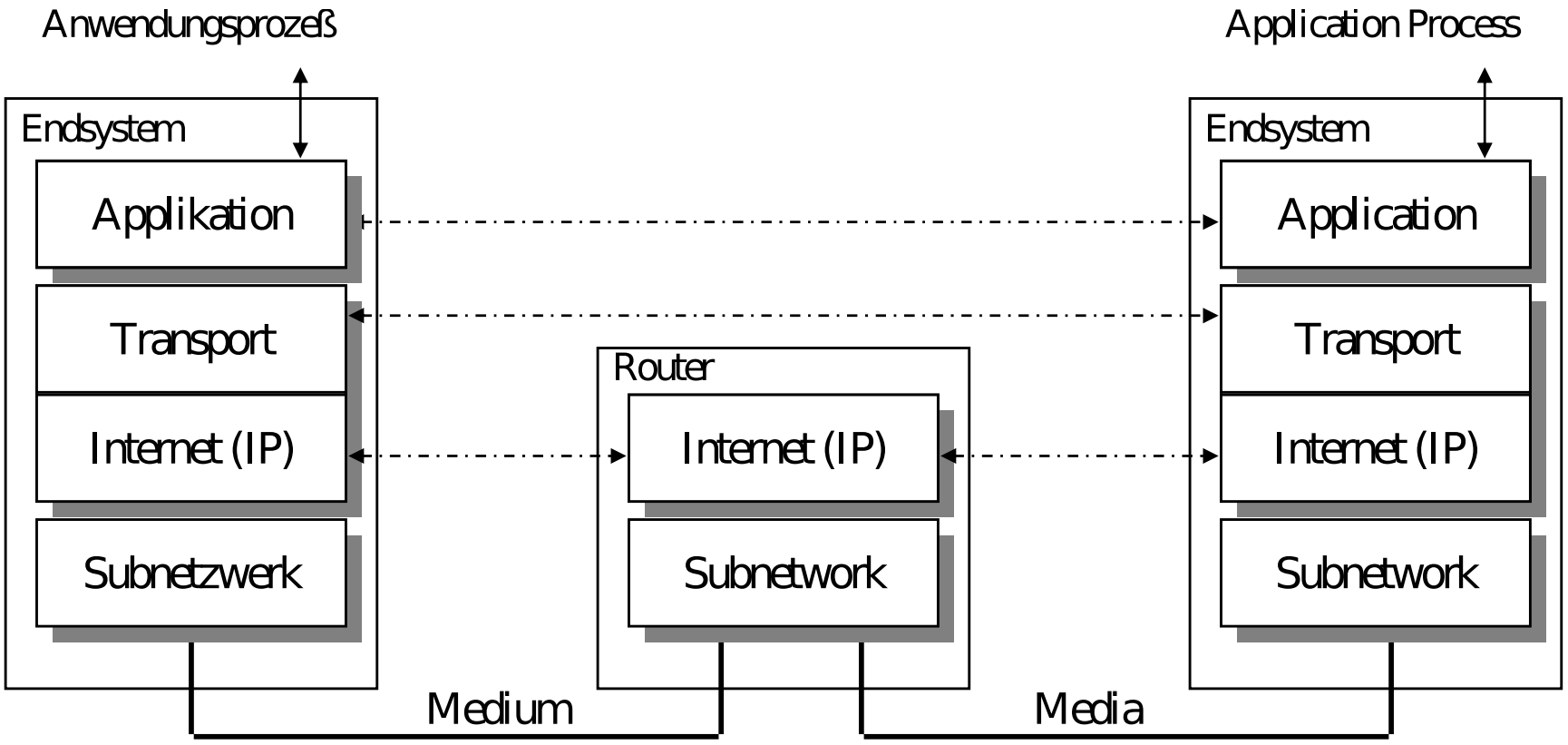
Interaktionsmodelle

- Client-Server
- Multi-Server
- Service-Oriented Architecture (SOA)
- Multi-Tiered
- Peer-to-Peer
- Grid Computing

Service-Oriented Architectures

- Konzept für modulare Systemarchitektur
- Lose gekoppelte Einheiten (Services)
- Standardisierte Schnittstellenbeschreibung (Interface Definition Language - IDL)
- Einheitliches Kommunikationsprotokoll
- Anwendungen durch Zusammensetzung verschiedener Services
- SOA-Prinzipien unabhängig von konkreter Realisierung (CORBA, J2EE, Web-Services)

Internet-Schichtenmodell



Wireshark

- Netzwerkprotokoll Analyse Werkzeug (Sniffer)
- Liest und visualisiert Datenstrom der Netzwerkschnittstelle
- Werkzeug zur Analyse des Netzwerkverkehrs
- Analyse und Aufbereitung von Headerinhalten

Wireshark

- Netzwerkprotokoll Analyse Werkzeug (Sniffer)
- Liest und visualisiert Datenstrom der Netzwerkschnittstelle
- Werkzeug zur Analyse des Netzwerkverkehrs
- Analyse und Aufbereitung von Headerinhalten

$\Rightarrow$ **Wireshark Demo**

Netzwerkprogrammierung

- Anwendungen nutzen Protokolle der Transportschicht
- Im Internet hauptsächlich: TCP und UDP
  - TCP: verbindungsorientiert, zuverlässig, Bytestrom, Flusskontrolle
  - UDP: verbindungslos, unzuverlässig, Datagramme
- Programmierschnittstelle: Socket-API
  - Socket = Kommunikationsendpunkt
  - Berkeley sockets / BSD sockets
  - Entwickelt für C
  - Verfügbar für die meisten weiteren Programmiersprachen

BSD Socket API

- `socket()`: Erzeugen eines Sockets
- `bind()`: Zuweisung einer Adresse
- `close()`: Schließen des Sockets

TCP

- `listen()`: Warten auf Verbindung eines Clients
- `accept()`: Annahme einer Clientverbindung
- `connect()`: Verbindungsaufbau durch Client
- `read/write()`: Senden und Empfangen von Daten

UDP

- `rcvfrom()`: Warten auf Daten von einer Adresse
- `sendto()`: Senden von Daten an eine Adresse

Java Socket API

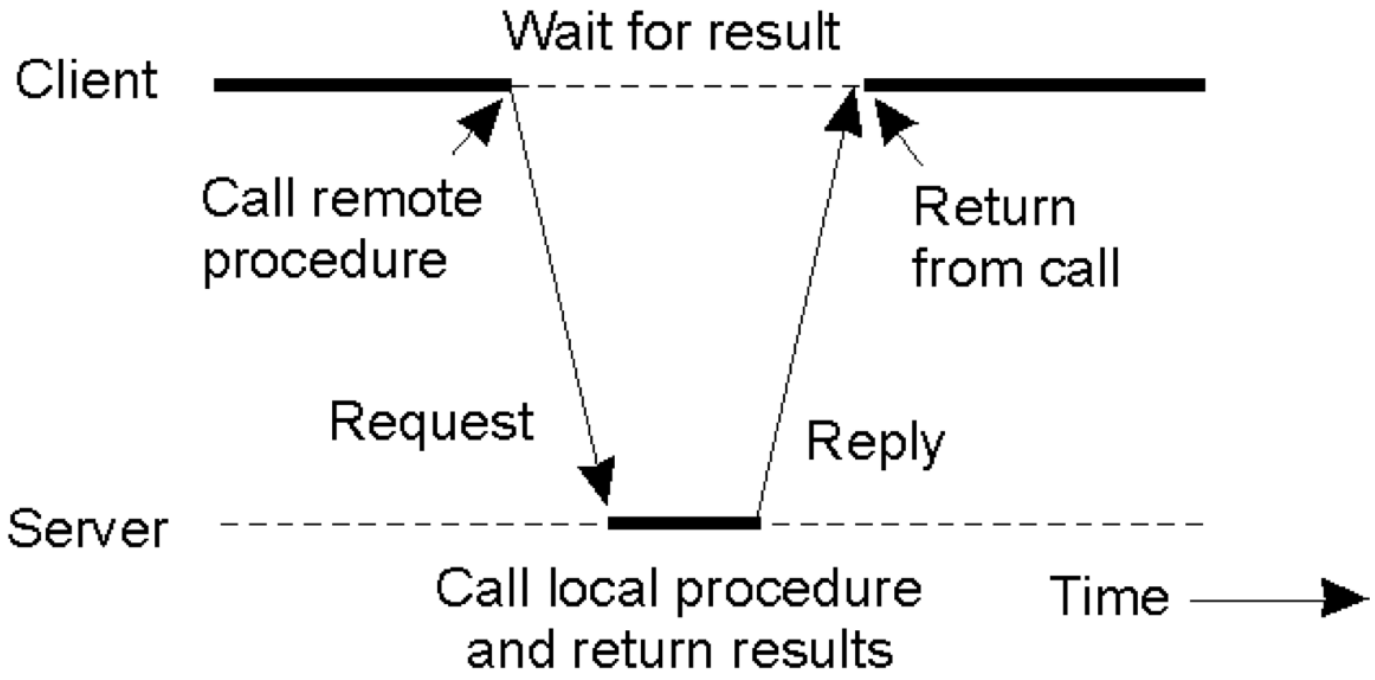
- `java.net.ServerSocket`: wartet auf eingehende Verbindungen
- `java.net.Socket`: Client Socket, zum Aufbau einer Verbindung
- Lesen und Schreiben mit Hilfe von Streams:
  - `Socket.getOutputStream()`
  - `Socket.getInputStream()`

Java Socket API

- `java.net.ServerSocket`: wartet auf eingehende Verbindungen
- `java.net.Socket`: Client Socket, zum Aufbau einer Verbindung
- Lesen und Schreiben mit Hilfe von Streams:
  - `Socket.getOutputStream()`
  - `Socket.getInputStream()`

⇒ **Demo**

Remote Procedure Call (RPC)



(aus Tanenbaum, van Steen: Distributed Systems)

Java RMI

- RMI = Remote Method Invocation
- Methodenaufruf auf entfernte Java-Objekte
- Finden + Aufruf von Objekten
- Signalisierung von Methodenaufrufen
- Serialisierung von Objekten
- Middleware

Java RMI

- RMI = Remote Method Invocation
- Methodenaufruf auf entfernte Java-Objekte
- Finden + Aufruf von Objekten
- Signalisierung von Methodenaufrufen
- Serialisierung von Objekten
- Middleware

⇒ **Demo**

Fragen?

brandt@ibr.cs.tu-bs.de

Nächste Übung: 10.05.2011 09:45 - 11:15 Uhr