Netzwerktechnologien
3 VO

Univ.-Prof. Dr. Helmut Hlavacs
helmut.hlavacs@univie.ac.at

Dr. Ivan Gojmerac
gojmerac@ftw.at

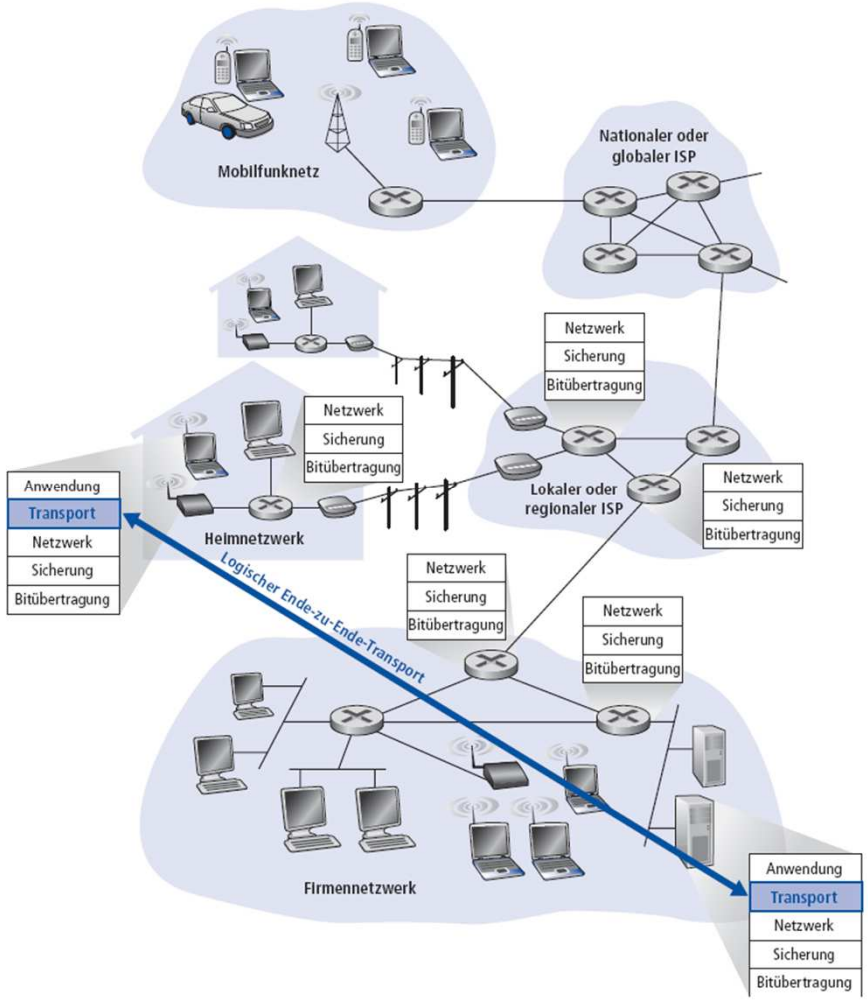
Bachelorstudium Medieninformatik
SS 2012

Kapitel 3 – Transportschicht

3.1 Dienste der Transportschicht
3.2 Multiplexing und Demultiplexing
3.3 Verbindungsloser Transport: UDP
3.4 Grundlagen der zuverlässigen Datenübertragung
3.5 Verbindungsorientierter Transport: TCP
3.6 Grundlagen der Überlastkontrolle
3.7 TCP-Überlastkontrolle

3.1 Dienste der Transportschicht

- Stellen **logische Kommunikation** zwischen Anwendungsprozessen auf verschiedenen Hosts zur Verfügung
- Transportprotokolle laufen auf Endsystemen
  - *Sender*: teilt Anwendungsnachrichten in **Segmente** auf und gibt diese an die Netzwerkschicht weiter
  - *Empfänger*: fügt Segmente wieder zu Anwendungsnachrichten zusammen, gibt diese an die Anwendungsschicht weiter
- Es existieren verschiedene Transportschichtprotokolle
  - Internet: TCP und UDP



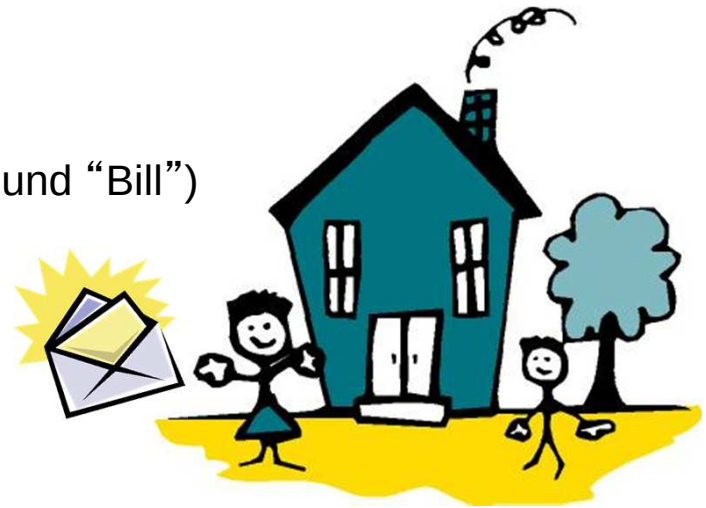
3.1 Unterschied: Transport- und Netzwerkschicht

- *Netzwerkschicht:* logische Kommunikation zwischen Hosts
- *Transportschicht:* logische Kommunikation zwischen Prozessen
  - verwendet und erweitert die Dienste der Netzwerkschicht

<u>Analogie: Briefzustellung an mehrere Bewohner des selben Hauses</u>

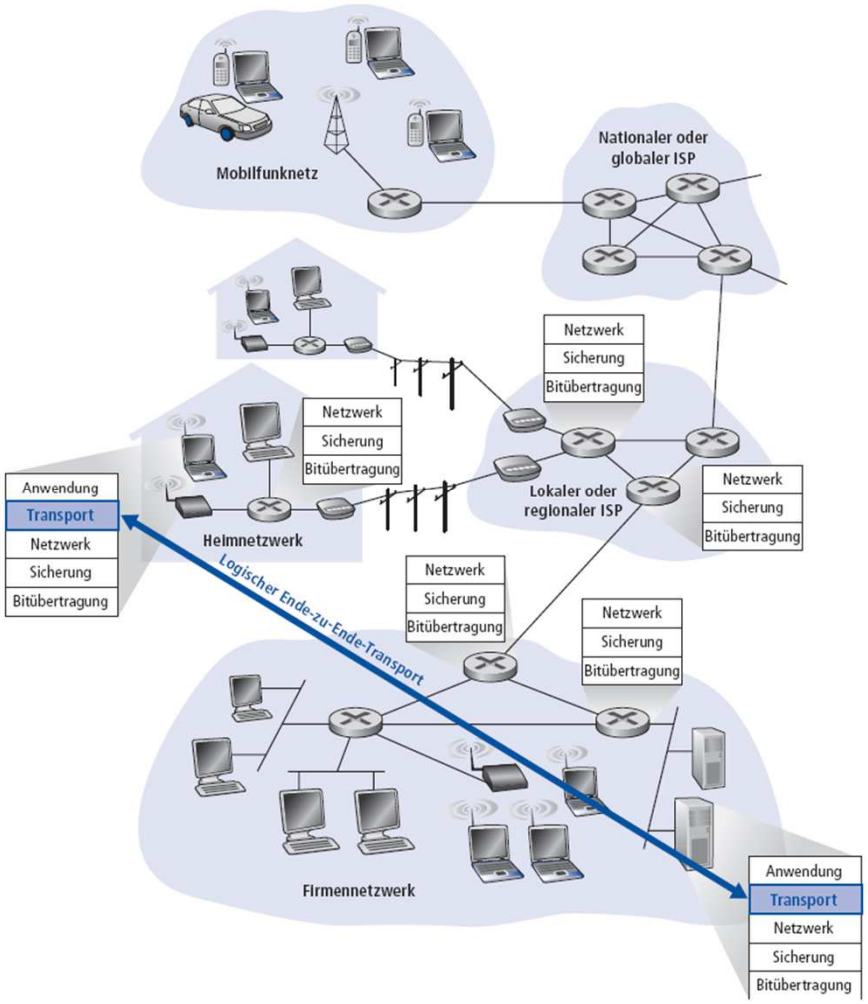
→ *12 Kinder senden Briefe an 12 andere Kinder.*

- Prozess = Kind
- Anwendungsnachricht = Brief in einem Umschlag
- Host = Haus
- Transportprotokolle = Namen der Kinder (z.B. “Anne” und “Bill”)
- Netzwerkprotokoll = Postdienst

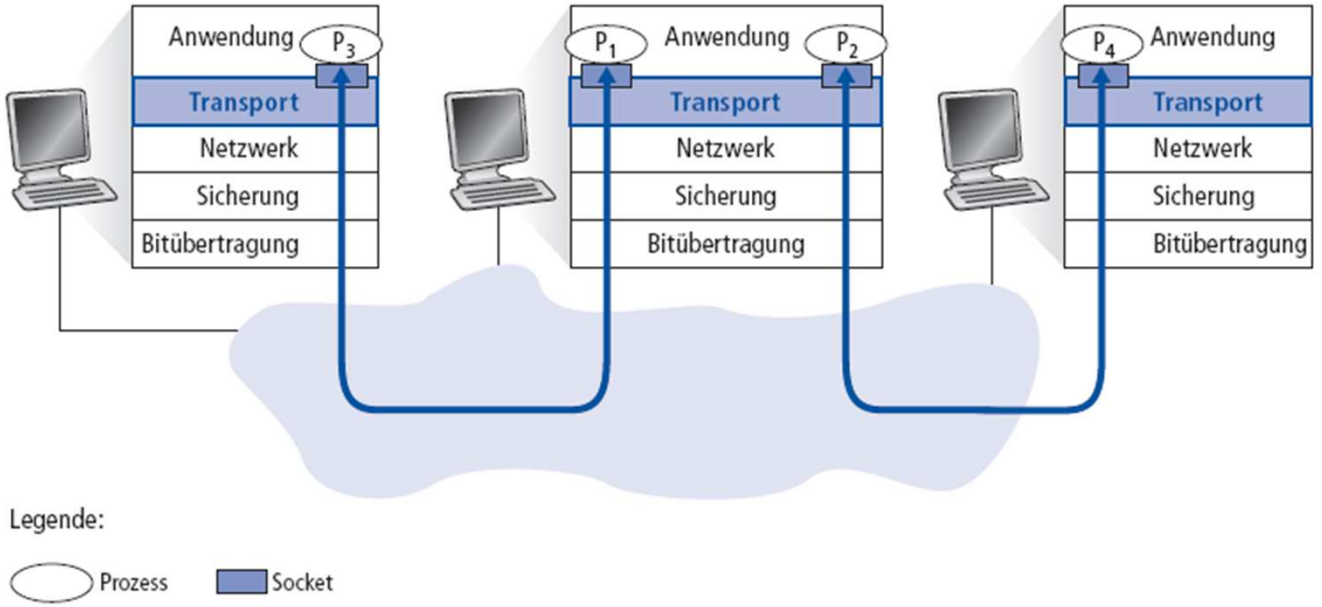


3.1 Transportprotokolle im Internet

- Zuverlässige, reihenfolgeerhaltende Auslieferung (TCP)
  - Überlastkontrolle
  - Flusskontrolle
  - Verbindungsmanagement
- Unzuverlässige Datenübertragung ohne Reihenfolgeerhaltung: UDP
  - Minimale Erweiterung der “Best-Effort”-Funktionalität von IP
- Dienste, die <u>nicht</u> zur Verfügung stehen:
  - × Garantien bezüglich Bandbreite oder Verzögerung



3.2 Multiplexing und Demultiplexing



- **Multiplexing** beim Sender:

Daten von mehreren Sockets einsammeln, Daten mit einem Header versehen (der später für das Demultiplexing verwendet wird).

- **Demultiplexing** beim Empfänger:

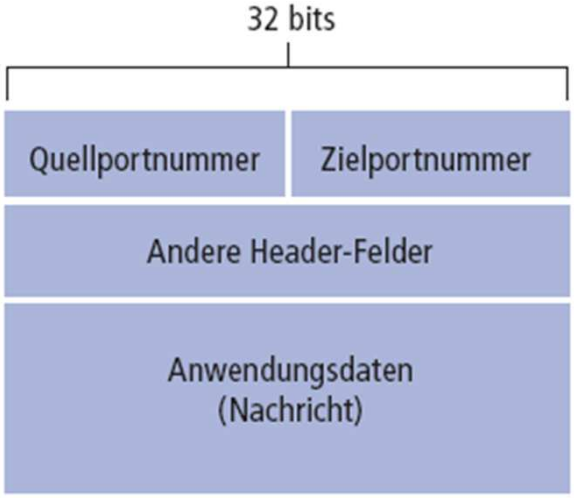
Empfangene Segmente am richtigen Socket abliefern.

3.2.1 Demultiplexing

Analogie: Briefzustellung an mehrere Bewohner des selben Hauses

Bill erhält einen Stapel Briefe vom Briefträger und gibt jeden davon dem Kind, dessen Name auf dem Brief steht.

- Host empfängt IP-Datagramme
  - Jedes Datagramm hat eine Absender-IP-Adresse und eine Empfänger-IP-Adresse
  - Jedes **Datagramm** beinhaltet ein **Transportschichtsegment**
  - Jedes **Segment** hat eine Absender- und eine Empfänger-Portnummer
- Hosts nutzen **IP-Adressen** und **Portnummern**, um Segmente an den richtigen Socket weiterzuleiten



TCP/UDP Segmentformat

3.2.1 Verbindungsloses Demultiplexing (UDP)

- Sockets mit Portnummer anlegen:

```
DatagramSocket mySocket1 = new DatagramSocket(12534);
DatagramSocket mySocket2 = new DatagramSocket(12535);
```

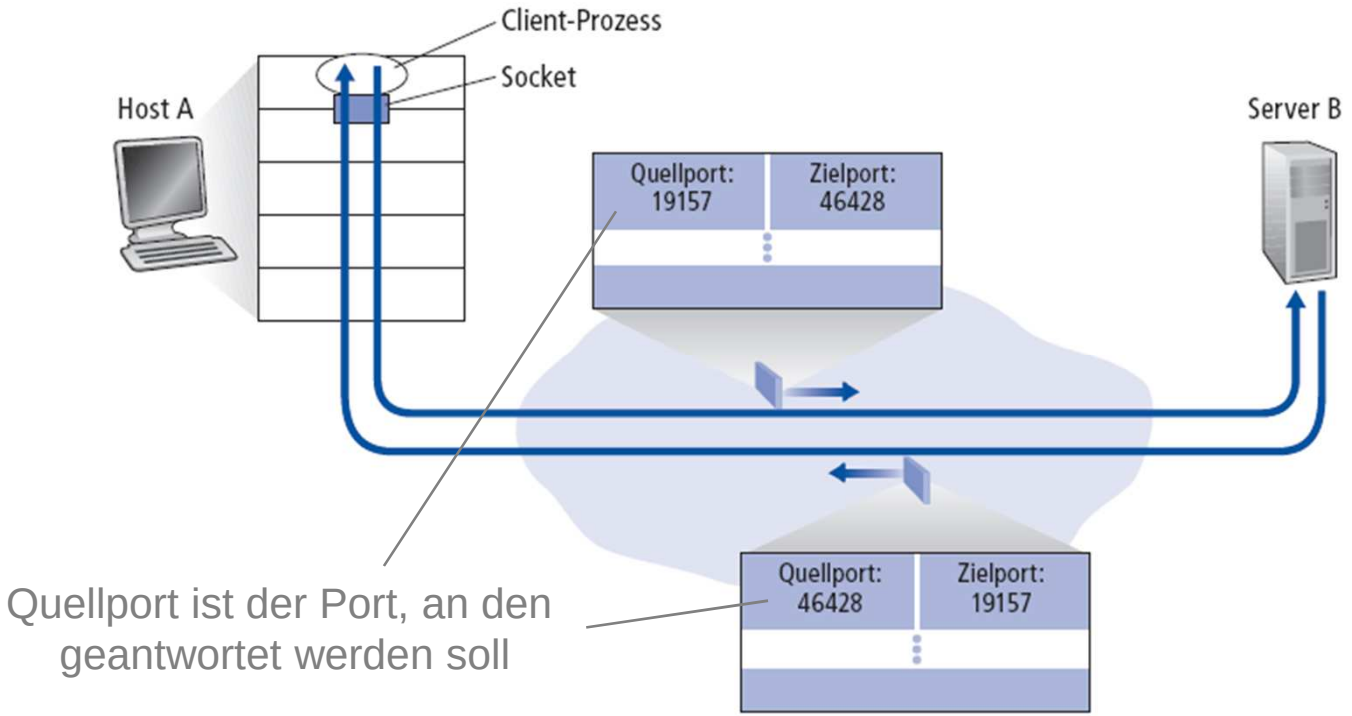
- **UDP**-Socket wird durch ein 2-Tupel identifiziert:

(Empfänger-IP-Adresse, Empfänger-Portnummer)

- Wenn ein Host ein UDP-Segment empfängt:
  1. Lese Empfänger-**Portnummer**
  2. Das UDP-Segment wird an den UDP-Socket mit dieser Portnummer weitergeleitet
- IP-Datagramme mit anderer Absender-IP-Adresse oder anderer Absender-Portnummer werden an **denselben Socket** ausgeliefert

3.2.1 Verbindungsloses Demultiplexing

```
DatagramSocket serverSocket = new DatagramSocket(46428);
```



3.2.2 Verbindungsorientiertes Demultiplexing (TCP)

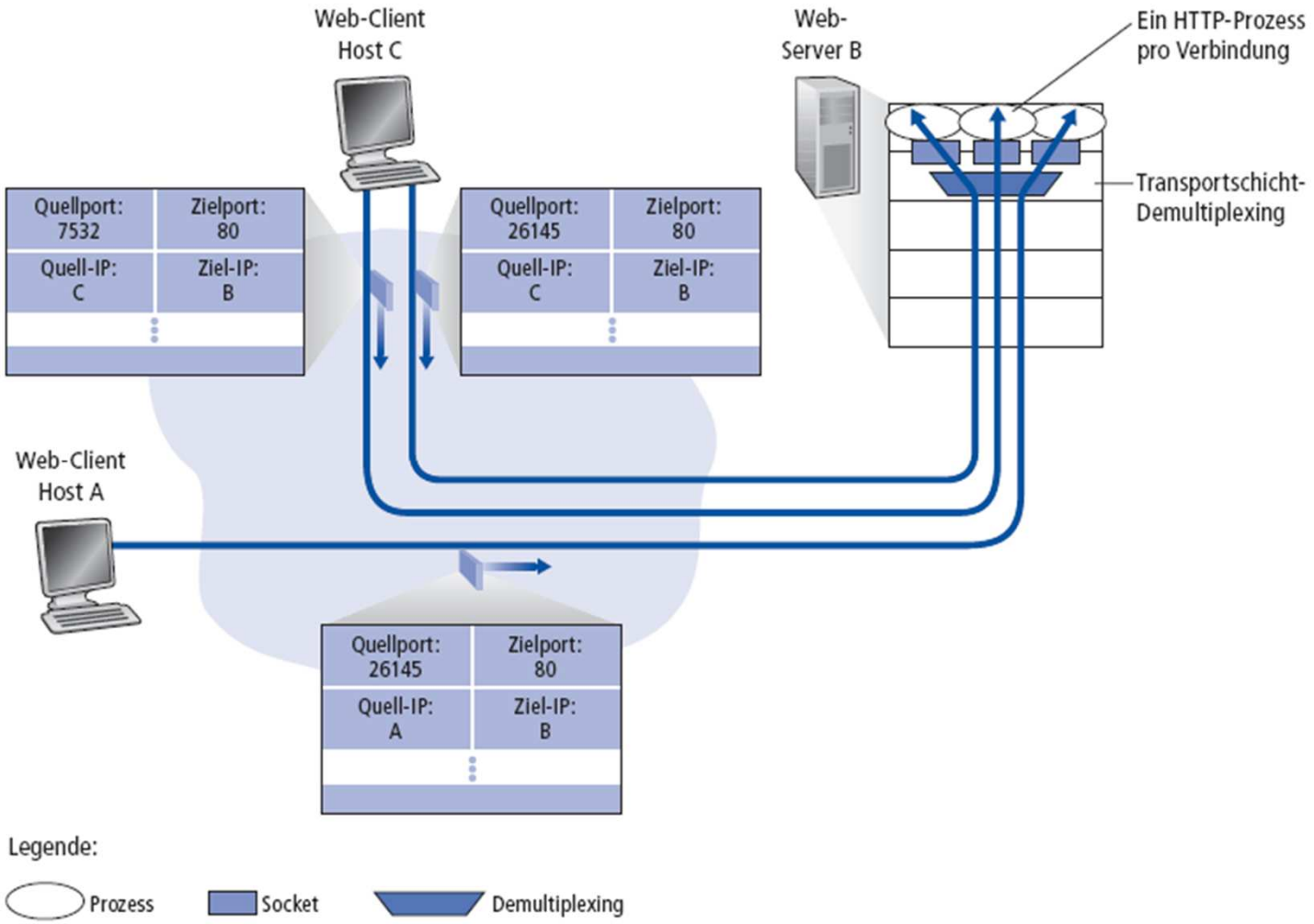
- **TCP**-Socket wird durch ein 4-Tupel identifiziert:
  - Absender-IP-Addresse
  - Absender-Portnummer
  - Empfänger-IP-Adresse
  - Empfänger-Portnummer

→ *Empfänger nutzt alle vier Werte, um den richtigen TCP-Socket zu identifizieren*

- Server kann viele TCP-Sockets gleichzeitig offen haben:
  - Jeder Socket wird durch sein eigenes 4-Tupel identifiziert

- Webserver haben verschiedene Sockets für jeden einzelnen Client
  - Bei nichtpersistentem HTTP wird jede Anfrage über einen eigenen Socket beantwortet (dieser wird nach jeder Anfrage wieder geschlossen)

3.2.2 Verbindungsorientiertes Demultiplexing (TCP)



3.3 Verbindungsloser Transport: UDP

- Minimales Internet-Transportprotokoll
- "Best-Effort"-Dienst, UDP-Segmente können:
  - verloren gehen
  - in der falschen Reihenfolge an die Anwendung ausgeliefert werden
- *Verbindungslos:*
  - kein Handshake zum Verbindungsaufbau
  - jedes UDP-Segment wird unabhängig von allen anderen behandelt

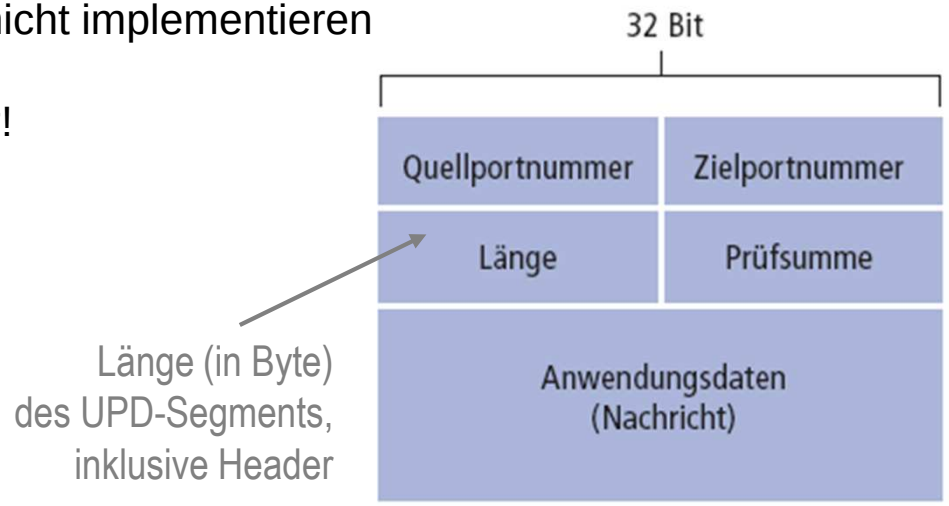
<u>Warum gibt es UDP?</u>

- Kein Verbindungsaufbau (der zu Verzögerungen führen kann)
- Einfach: kein Verbindungszustand im Sender oder Empfänger
- Kleiner Header
- Keine Überlastkontrolle: UDP kann so schnell wie von der Anwendung gewünscht senden

3.3 Verbindungsloser Transport: UDP

- Häufig für Anwendungen im Bereich **Multimedia-Streaming** eingesetzt
  - Verlusttolerant
  - Mindestrate
- Andere Einsatzgebiete
  - DNS
  - SNMP
- Zuverlässiger Datentransfer über UDP:
  Zuverlässigkeit auf der Anwendungsschicht implementieren

→ Anwendungsspezifische Fehlerkorrektur!



UDP Segmentformat

3.3.1 Fehlerkorrekturmechanismus: UDP Prüfsumme

Ziel: Fehler im übertragenen Segment erkennen (z.B. verfälschte Bits)

Sender:

- Betrachte Segment als Folge von 16-Bit-Integer-Werten
- Prüfsumme: Addition (1er-Komplement-Summe) dieser Werte
- Sender legt das *invertierte Resultat* im UDP-Prüfsummenfeld ab

Empfänger:

- Berechne die Prüfsumme des empfangenen Segments **inkl. des Prüfsummenfeldes**
- Sind im Resultat alle Bits 1?
  - → NEIN – Fehler erkannt
  - → JA – Kein Fehler erkannt

3.3.1 Fehlerkorrekturmechanismus: UDP Prüfsumme

Zahlen werden addiert und ein Übertrag aus der höchsten Stelle wird zum Resultat an der niedrigsten Stelle addiert.

Beispiel:
Addiere zwei 16-Bit-Integer-Werte.

		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	+	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Übertrag		1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
Summe (mit Übertrag)		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
Prüfsumme		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1