Karlsruher Institut für Technologie

Vorlesung Einführung in Rechnernetze
3. Die Transportschicht

Prof. Dr. Martina Zitterbart

Sebastian Friebe (M.Sc.), Markus Jung (M.Sc.), Matthias Flittner (M.Sc.), Tim Gerhard (M.Sc.)
[zitterbart | friebe | m.jung | flittner | t.gerhard]@kit.edu

Institut für Telematik, Prof. Zitterbart

Kapitel der Vorlesung

1. Einführung
2. Anwendungsschicht
3. Transportschicht
4. Vermittlung
5. Link Layer
6. Netzarchitektur
7. Netzsicherheit

Überblick über Kapitel 3

3. Transport-schicht

- 3.1 Grundlagen
- 3.2 Multiplexen/Demultiplexen
- 3.3 UDP
- 3.4 Prinzipien zuverlässiger Datenübertragung
- 3.5 Flusskontrolle
- 3.6 TCP

Kapitel 3.1

GRUNDLAGEN

Internet-Protokollstack

- Grundlagen
  - Multiplexen / Demultiplexen
  - Zuverlässiger, unzuverlässiger Transport
  - Flusskontrolle
  - Staukontrolle
- Transportprotokolle im Internet
  - UDP: unzuverlässiger Transportdienst
  - TCP: zuverlässiger Transportdienst

Transportdienste und -protokolle

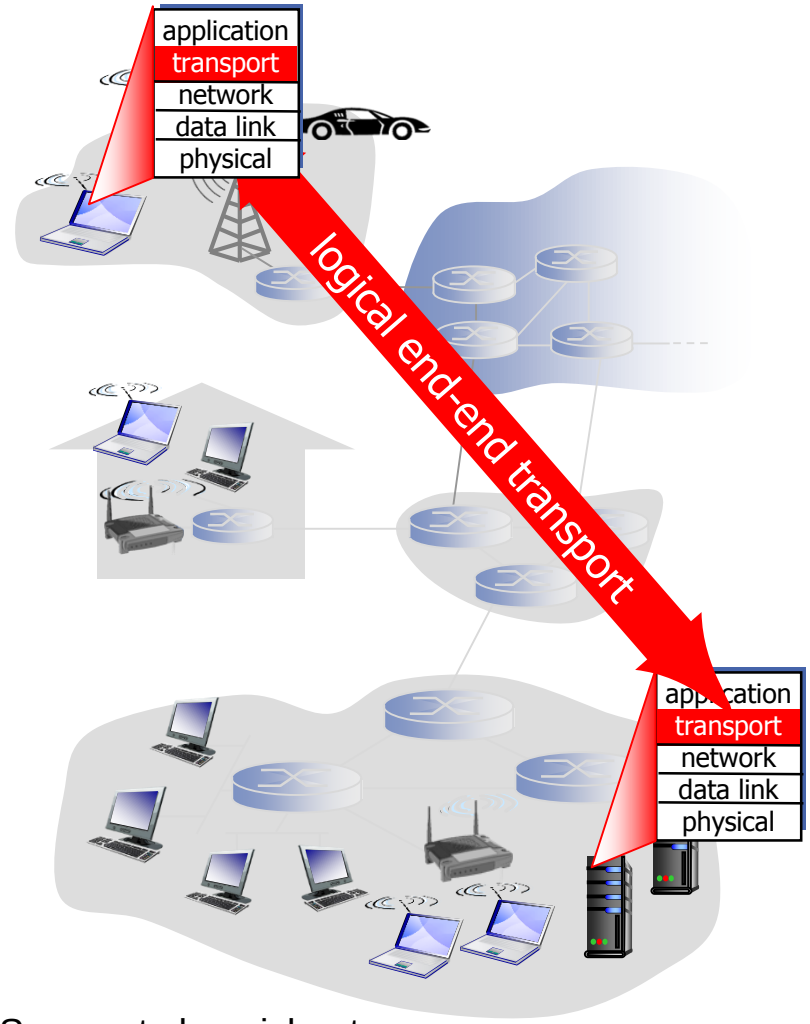
- Grundlegende Aufgabe der Transportschicht
  - Verbergen von Details der transportorientierten Kommunikation vor den höheren Schichten (de-facto der Anwendung), z.B.
    - Fehlercharakteristika
    - Genutzte Technologien

  … nicht alles lässt sich verbergen: Verzögerung bleibt „sichtbar“!

- Bereitstellung von Transportdiensten
  - Hierzu logische Kommunikationsbeziehung zwischen Nutzern, i.d.R. (Anwendungs-)prozessen
    - Nutzer-zu-Nutzer Kommunikation

Transportdienste und -protokolle

- Transportprotokoll läuft auf den Endsystemen
- Sendende Seite
  - segmentiert Anwendungsnachrichten in Segmente, die sie an die Vermittlungsschicht weiterleitet
- Empfangende Seite
  - reassembliert Segmente in Nachrichten, die sie an die Anwendungsschicht weiterleitet



* Pakete der Transportschicht werden in dieser Vorlesung als Segmente bezeichnet

Transportdienste und -protokolle

- Im Internet (im wesentlichen)
  - UDP (User Datagram Protocol)
    - Bietet verbindungslosen, unzuverlässigen Transportdienst
  - TCP (Transmission Control Protocol)
    - Bietet verbindungsorientierten, zuverlässigen Transportdienst

- Weitere Transportprotokolle zur
  - Unterstützung von Anforderungen multimedialer Anwendungen
  - Unterstützung von Gruppenkommunikation
  - Unterstützung von Signalisierung
  - Unterstützung von mehreren Pfaden
  - …

Unzuverlässig / Zuverlässig

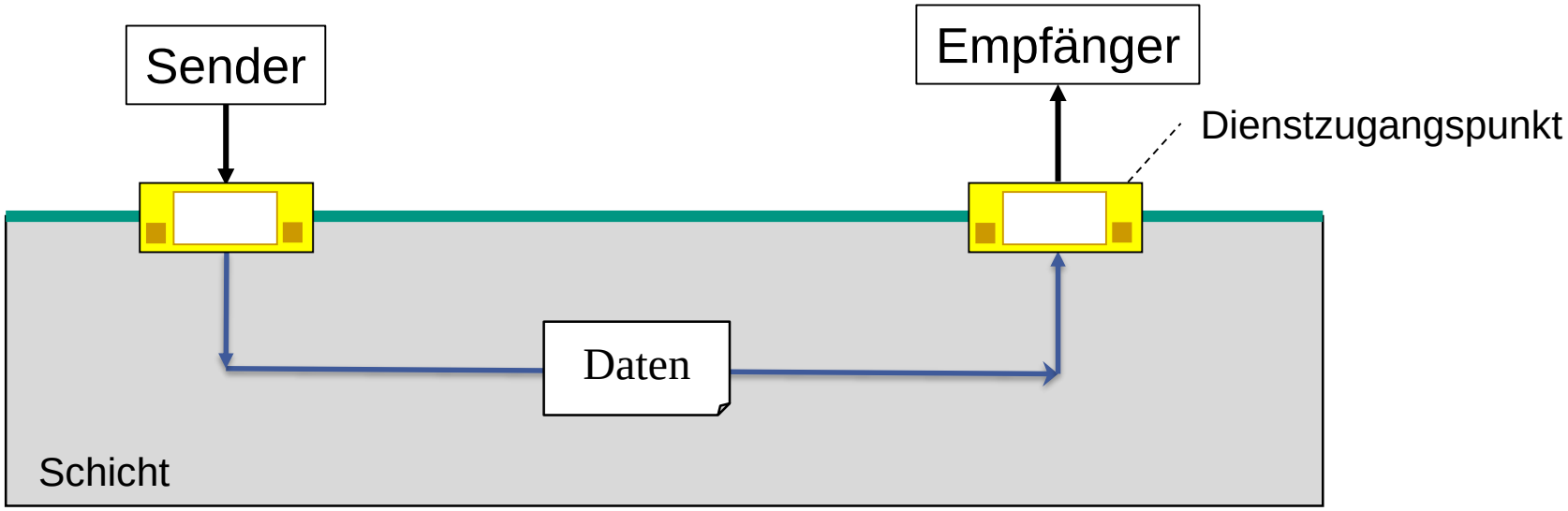
- Unzuverlässiger Kommunikationsdienst
  - Es wird keinerlei Aussage darüber getroffen, wie viel der gesendeten Daten korrekt beim Empfänger ankommen
    … meist geht man davon aus, dass Großteil der Daten korrekt ankommt
  - Bei Fehlern werden keine weiteren Maßnahmen unternommen

- Zuverlässiger Kommunikationsdienst
  - Für die empfangende Instanz wird folgendes garantiert
    - Daten sind korrekt und vollständig
    - Daten werden in der richtigen Reihenfolge ausgeliefert
    - Es werden keine Duplikate ausgeliefert
    - Es werden keine Phantom-Pakete ausgeliefert
  - Bei Fehlern sind entsprechende Maßnahmen erforderlich
    - Mechanismen zur Fehlererkennung und -behebung

Betrachtung hier zunächst unabhängig von der individuellen Schicht.

Schicht und Dienst

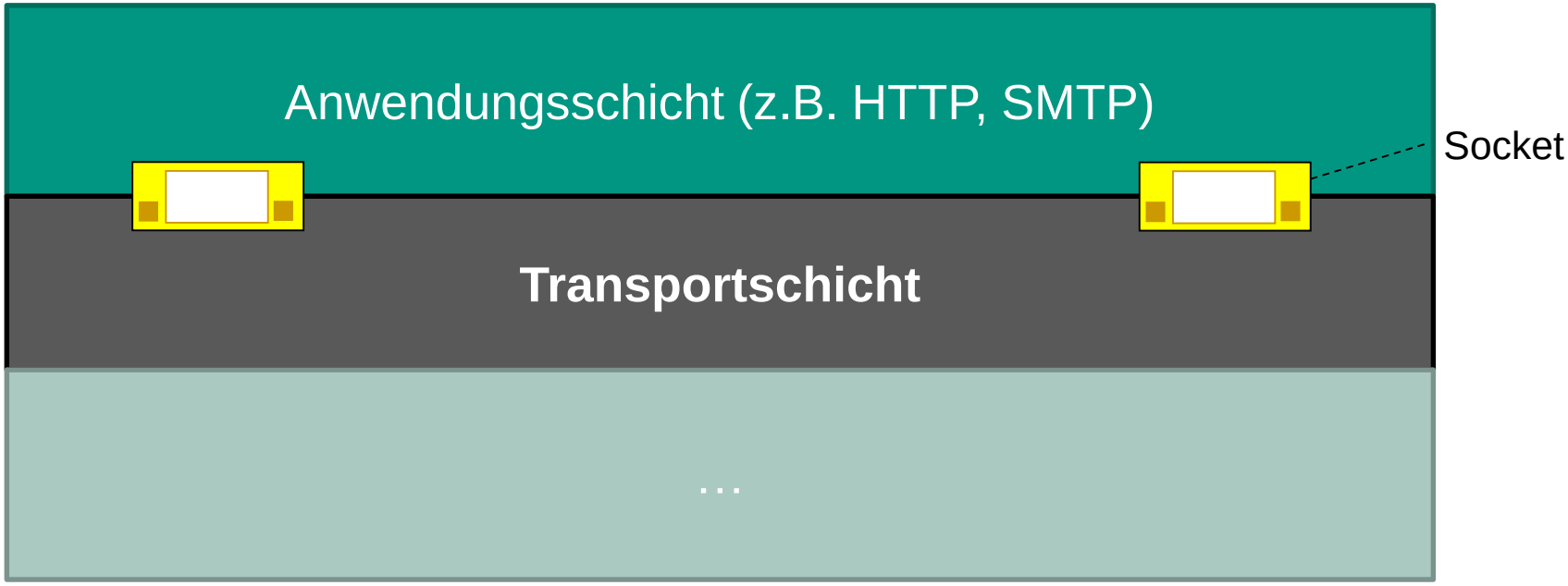
- Schichten stellen Abstraktion dar
  - Dienste werden an der Schnittstelle nach oben angeboten
  - Höhere Schicht nutzt Dienste der darunterliegenden Schicht
    - Dienste werden an Dienstzugangspunkt bereitgestellt



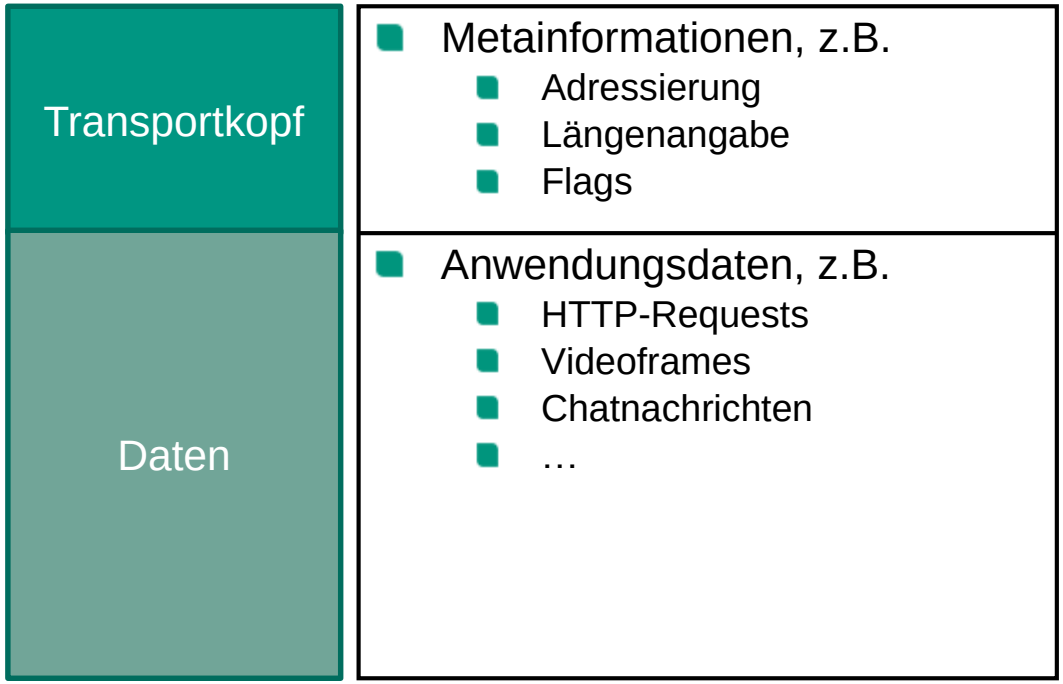
- Dienst
  - Bündelung zusammengehöriger Funktionen

Transportdienst

- Kommunikationsdienst, der von der Transportschicht angeboten wird
  - Transportdienst wird durch Funktionen der Transportprotokolle erbracht
  - Im Internet meist
    - Unzuverlässiger Transportdienst: UDP
    - Zuverlässiger Transportdienst: TCP



Segmente


Transportkopf
Daten
Metainformationen, z.B.
Adressierung
Längenangabe
Flags
Anwendungsdaten, z.B.
HTTP-Requests
Videoframes
Chatnachrichten
…

Kapitel 3.2

MULTIPLEXEN / DEMULTIPLEXEN

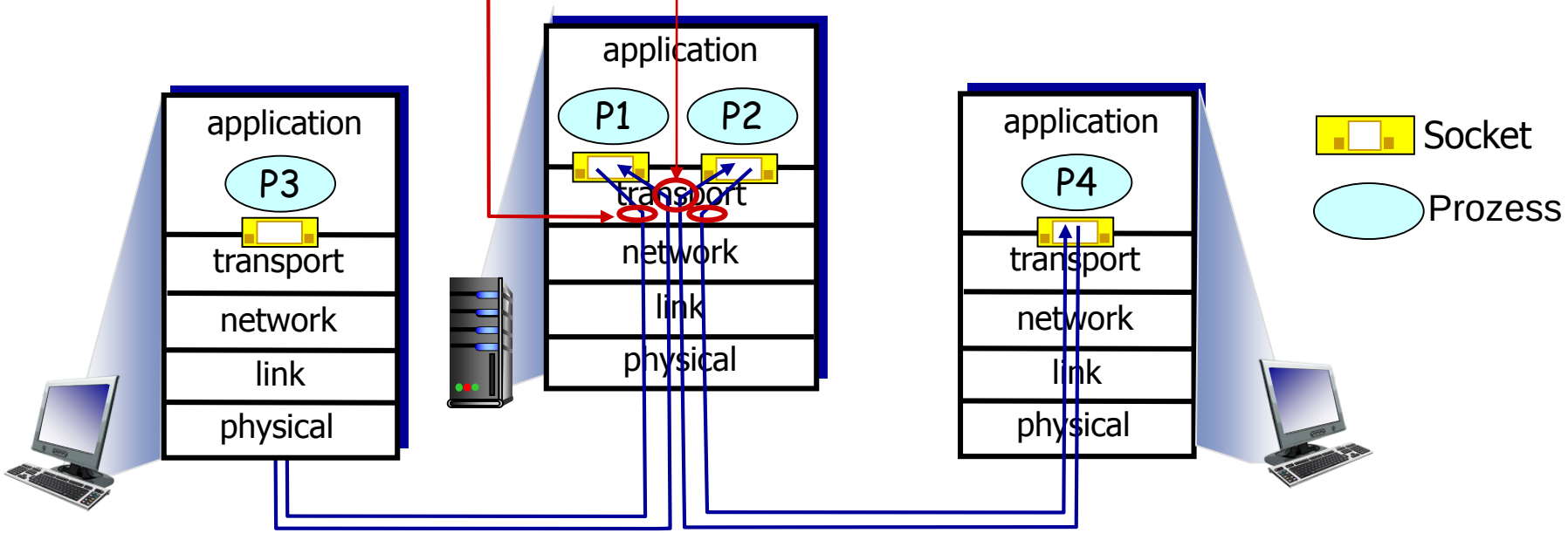
Multiplexen / Demultiplexen

Multiplexing beim Sender:

Daten von mehreren Sockets, Transportkopf hinzufügen

Demultiplexing beim Empfänger:

Information im Transportkopf nutzen, um empfangene Segmente an die korrekten Sockets auszuliefern



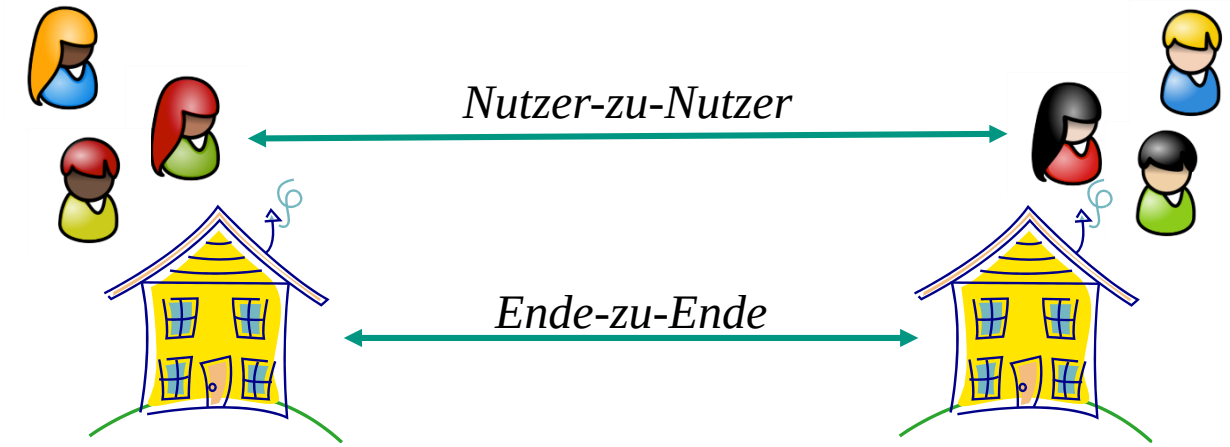
Nutzer-zu-Nutzer versus Ende-zu-Ende

- Transportschicht
  - Logische Kommunikation zwischen (Anwendungs-)Prozessen
  - Nutzer-zu-Nutzer
  - Benutzt hierzu die Dienste der Vermittlungsschicht

- Vermittlungsschicht
  - Logische Kommunikation zwischen Endsystemen
  - Ende-zu-Ende

Nutzer-zu-Nutzer versus Ende-zu-Ende

- Anschauliches Beispiel: Senden von Briefen mittels traditioneller Post
  - Brief: Anwendungsnachricht
  - Kommunikation zwischen Häusern (Endsystemen)
    - Ende-zu-Ende
  - Bob leert Briefkasten (Demultiplexen der Briefe auf Bewohner)
    - Bob ist im Haus (also im Endsystem)
  - Kommunikation zwischen Bewohnern (Prozessen)
    - Nutzer-zu-Nutzer
    - Vgl. Brieftransport durch Post



Pingo

http://pingo.upb.de/

Adressierung auf Transportschicht

- Ziel
  - Eindeutige Kennzeichnung der Nutzer (Prozesse) in einem Endsystem

- Ports
  - Adressen der Transportschicht
  - Unstrukturierte Nummer der Länge 16 Bit
    - Werte: 0 … 65535
  - Viele Portnummern kleiner 1024 für häufig benutzte Anwendungen reserviert (well-known ports), z.B.
    - 23 für Telnet
    - 25 für SMTP
    - 80 für HTTP

Port-Nummern-Konventionen (well-known ports)

- Anwendung muss richtigen Port wählen, um auf der Gegenseite mit der gewünschten Anwendung zu kommunizieren
  - Port 13: Tageszeit

```
> telnet osiris 13
Trying 129.13.3.121...
Connected to osiris.
Escape character is '^]'.
Mon Aug  4 16:57:19 1997
Connection closed by foreign host
```

  - Port 25: SMTP

```
> telnet sokrates 25
Trying 129.13.3.161...
Connected to sokrates .
Escape character is '^]'.
220 sokrates ESMTP Sendmail 8.8.5/8.8.5;
Mon, 4 Aug 1997 17:02:51 +0200
HELP
214-This is Sendmail version 8.8.5
214-Topics:
214-    HELO    EHLO    MAIL    RCPT    DATA
214-    RSET    NOOP    QUIT    HELP    VRFY
214-    EXPN    VERB    ETRN    DSN
214-For more info use "HELP <topic>".
...
214 End of HELP info
```

Adressierung auf Vermittlungsschicht

- Ziel
  - Eindeutige Kennzeichnung der Endsysteme (genauer Interfaces) in einem Netz

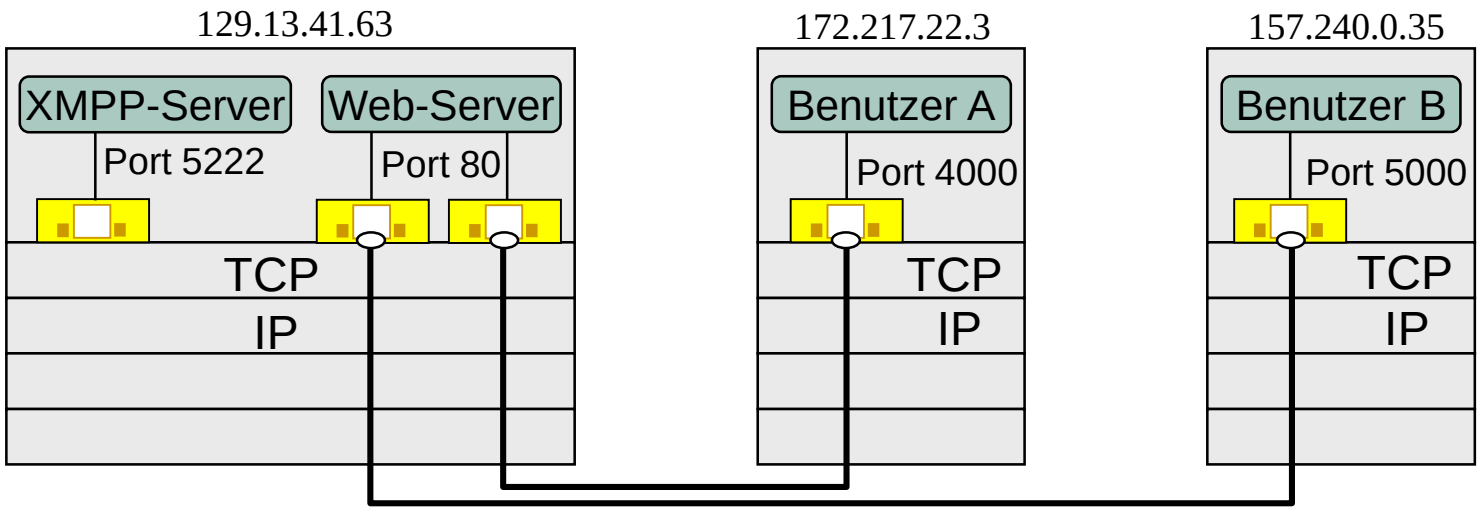
- IP-Adressen
  - Adressen der Vermittlungsschicht im Internet
  - IPv4: Länge von 32 Bit
    - Z.B. 207.142.131.235
  - IPv6: Länge von 128 Bit
    - Z.B. 2001:0db8:85a3:08d3:1319:8a2e:0370:7344

… Internet-weit eindeutige Adressierung eines Anwendungsprozesses

- IP-Adresse + Port

Eindeutige Adressierung: Beispiel

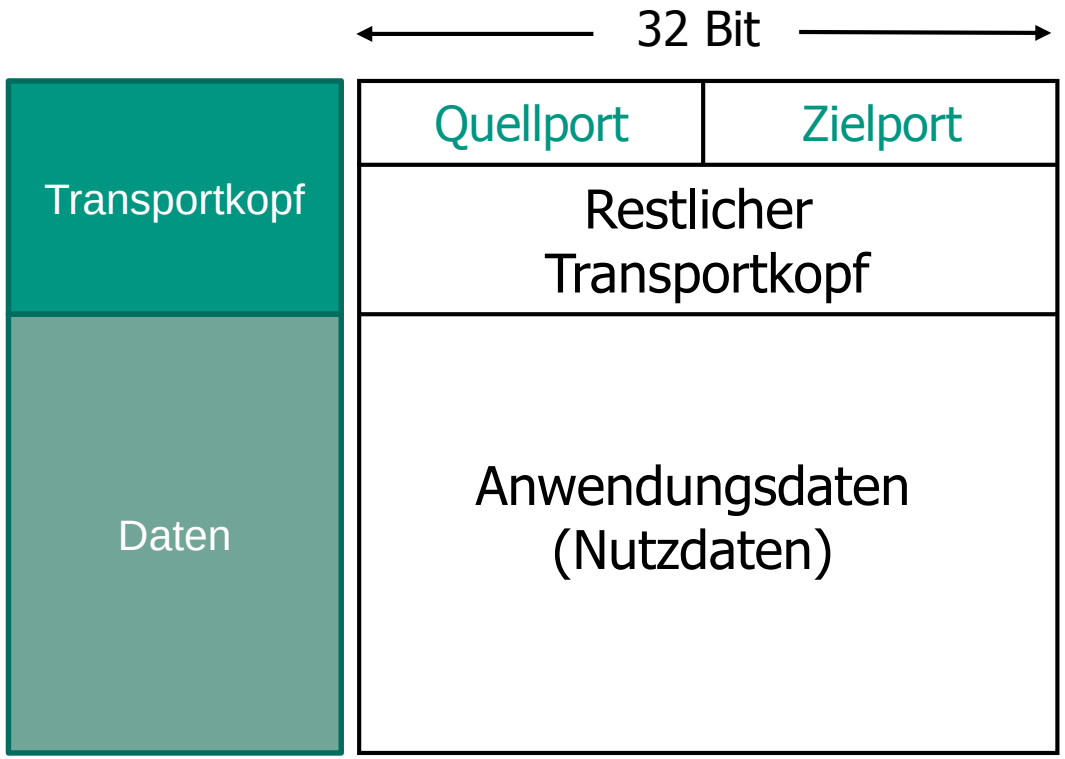
- Web-Server über HTTP
  - Erreichbar über 129.13.41.63:80
    - <IP-Adresse>:<Port>



- Die IP-Adresse bestimmt das Endsystem
- Der Port bestimmt den Server auf dem Endsystem

Ports im Transportkopf

- Quell- und Zielport werden im Transportkopf übertragen
  - Information wird genutzt, um Segment dem richtigen Anwendungsprozess (bzw. dem entsprechenden Socket) zuzuordnen

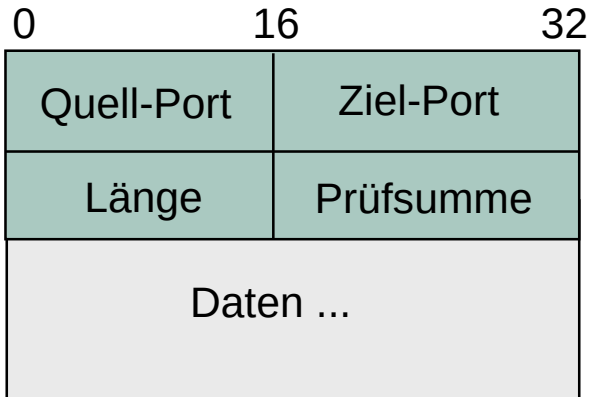


Kapitel 3.3

UDP

UDP (User Datagram Protocol): Überblick

- Sehr einfaches Transportprotokoll
  - RFC 768, August 1980
- Eigenschaften
  - Multiplexen / Demultiplexen von Segmenten für Prozesse
  - Überprüfung von Bitfehlern: Prüfsumme
  - Keine Verbindung
  - Keine Zuverlässigkeit
- UDP-Segment



[RFC768]

Eigenschaften von UDP

- Geringer Overhead durch Transportkopf
  - 8 Byte: Ports, Längenfeld und Prüfsumme
- Unreguliertes Senden
  - UDP kann Daten so schnell senden wie sie von der Anwendung geliefert und vom Netz abgenommen werden
- Best Effort
  - Keine Zusagen über Auslieferung der Daten beim Empfänger

Eigenschaften von UDP

- Keine Verbindungsaufbauphase
  - Daten können sofort gesendet werden, keine „Vorarbeiten“ erforderlich
  - Damit keine zusätzliche Verzögerung
- Kein Verbindungszustand
  - Keine verbindungsrelevanten Informationen im Endsystem
    - z.B. Flusskontrollfenster, Sequenznummern, Timer
  - Skaliert z.B. für Server besser
    - Können mit UDP typischerweise mehr Kommunikationsbeziehungen unterstützen als mit TCP

Einsatz von UDP

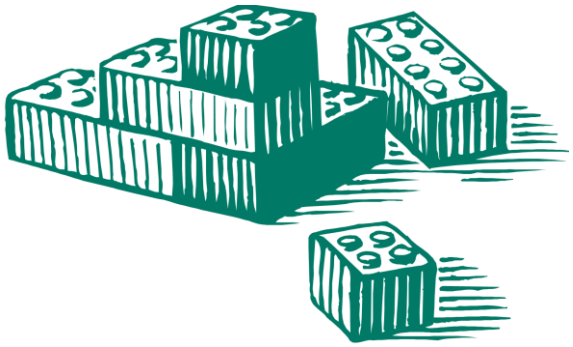
- Beispiele
  - Interaktives Multimedia, z.B. VoIP
    - Fehlertolerant
    - Sensitiv hinsichtlich der Rate der Auslieferung
  - DNS
    - Schnelle Reaktion wünschenswert
    - Meist nur ein einzelnes Paket als Antwort
  - …
- Zuverlässiger Datentransfer über UDP
  - Anwendungsschicht muss sich darum kümmern
  - Kennt spezifische Anforderungen der Anwendungen

Kapitel 3.4

PRINZIPIEN ZUVERLÄSSIGER DATENÜBERTRAGUNG

Protokollmechanismen

- Ziel
  - Geräte bzw. Anwendungen möchten Daten austauschen
- Protokolle erforderlich, die Formate und Regeln des Datenaustauschs festlegen
  - Protokollmechanismen stellen die Bausteine der Protokolle dar

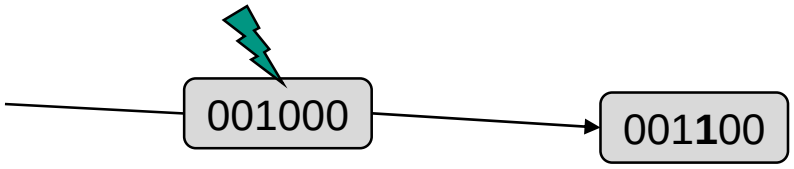


- Problem
  - Bei der Übertragung von Daten können Fehler auftreten

Anmerkung: Ausführungen in diesem Teilkapitel sind nicht nur für Transportschicht relevant!

Bit- und Paketfehler

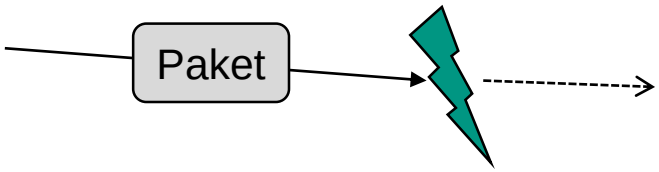
- Bitfehler
  - Verfälschung von Bits während dem Datentransport
    - Z.B. 0 gesendet und 1 empfangen
    - Fehlerursachen: Dämpfung des Übertragungssignals, Übersprechen, Verlust der Bit-Synchronisation …



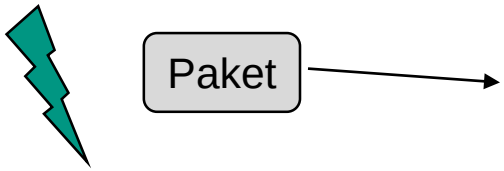
- Einzelbitfehler
  - Ein *einzelnes* Bit ist fehlerhaft
- Bündelfehler
  - *Mehrere direkt aufeinanderfolgende* Bits sind fehlerhaft

Bit- und Paketfehler

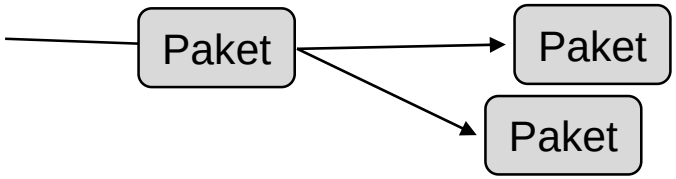
- Paketfehler
  - Fehlerarten
    - Verlust eines Pakets



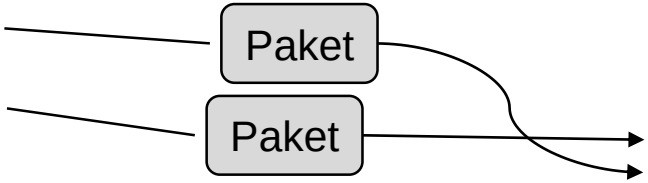
    - Empfang eines Phantom-Pakets



    - Duplizierung eines Pakets

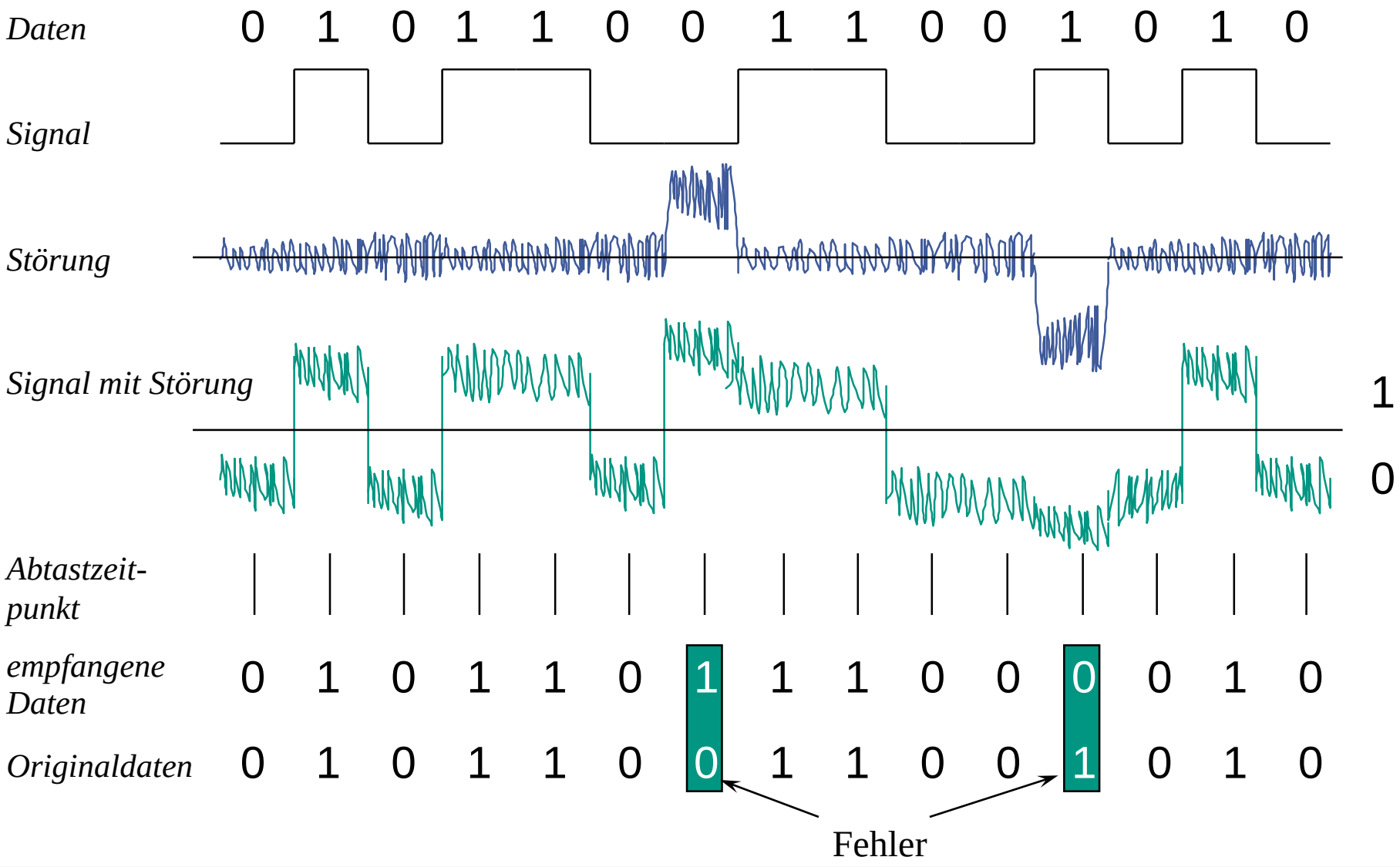


    - Reihenfolgevertauschung



  - Fehlerursachen
    - Überlastung von Zwischensystemen
    - Unterschiedliche Wege durch das Netz
    - Verfrühte Datenwiederholung
    - …

Beispiel: Auswirkung von Störungen



Fehlerhäufigkeit

- Bitfehlerrate Maß für die Fehlerhäufigkeit

$$Bitfehlerrate = \frac{Summe\ gestörter\ Bits}{Summe\ übertragener\ Bits}$$

- Beispielhafte Werte für Bitfehlerrate
  - Funkstrecke: $10^{-3} - 10^{-4}$
  - Ethernet (10Base2): $10^{-9} - 10^{-10}$
  - Glasfaser: $10^{-10} - 10^{-12}$

Fehlerauswirkungen

- Auswirkung einer Störung u.a. abhängig von der Datenrate
  - Beispiel: Eine Störung von 20 ms führt…
    - Bei Telex (50 bit/s, Bitdauer: 20 ms)
      - zu einem Fehler von 1 Bit

      → *Einzelbitfehler*
    - Bei ADSL2+ (16 Mbit/s, Bitdauer: 62,5 ns)
      - zu einem Fehler von ca. 320 kbit

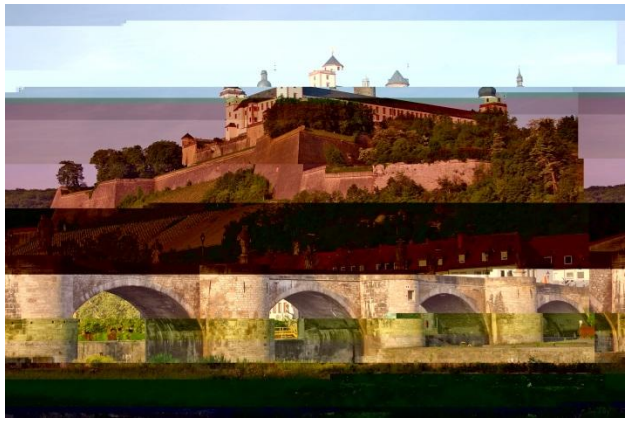
      → *Bündelfehler*
    - Bei Gigabit-Ethernet (1 Gbit/s, Bitdauer: 1 ns)
      - zu einem Fehler von 20 Mbit

      → *Bündelfehler*

Fehlerauswirkungen

- Falsche Daten und Reaktion: Fertigung, Roboter, Auto …

- Beispiel Bild: 3.5 Mbit mit 7 zufälligen Bitfehlern (BER 2e-6)



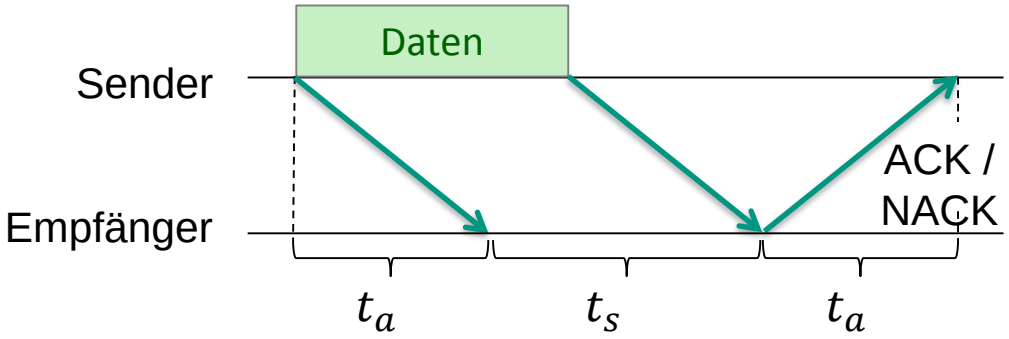
- Text mit 1% Bitfehlerwahrscheinlichkeit
  - „Der Umfang eings Kreis mit dum Radaus r ist U=0*pi*r“

Natürliche Sprache ist redundant.
Trotz Fehler ist Text erkennbar.

Keine Redundanz!

Idee: Redundanz

- Fehlererkennung *(Error Detecting Code, EDC)*
  - Redundanz zu Daten hinzufügen
  - Verwende Codewörter, die sich hinreichend unterscheiden
  - Beispiel „t“ und „d“ manchmal schwer unterscheidbar. Verwende Nato-Alphabet *„Tango“ und „Delta“*
- Fehlerkorrektur *(Forward Error Correction, FEC)*
  - Falls Fehler erkannt, versuche mittels Redundanz zu korrigieren
- Wiederholungsaufforderung *(Automatic repeat request, ARQ)*
  - Empfänger teilt Ergebnis der Fehlererkennung dem Sender mit



- ACK (Acknowledgement): Korrekte Übertragung
- NACK (Negative Acknowledgement): Übertragung nicht korrekt

Beispiele eingesetzter Protokollmechanismen

- Bei Bitfehlern
  - Fehlererkennende Codes (EDC)
  - Fehlerkorrigierende Codes (FEC)
- Bei Paketfehlern
  - Zur Erkennung erforderlich
    - Sequenznummern
    - Zeitüberwachung
    - Quittungen
  - Reaktion auf Paketfehler (ARQ)
    - Sendewiederholungen
  - Prävention
    - Vorwärtsfehlererkennung
      - Redundanz bereits beim Senden hinzufügen

Kapitel 3.4.1 Fehlerkontrolle bei Bitfehlern

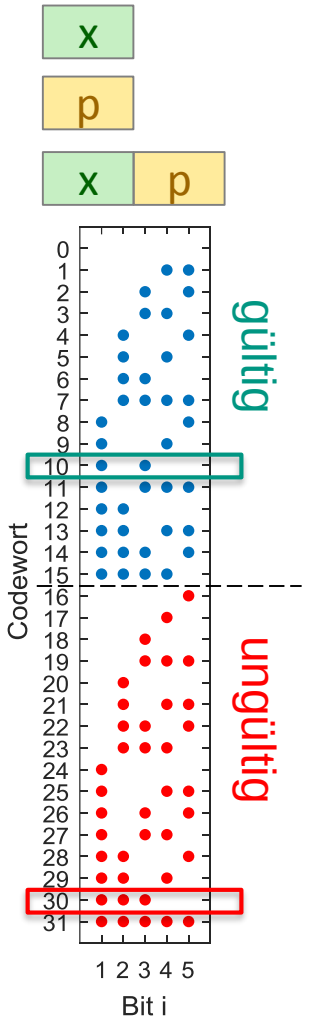
Fehlerkontrolle bei Bitfehlern

- Problem
  - Wie können Bitfehler beim Empfänger oder in netzinternen Zwischensystemen erkannt werden?
- Grundlegender Ansatz
  - Hinzufügen von Redundanz bei der Übertragung
    - z.B. Paritätsbits, Prüfsumme …

Einfache Paritätsprüfung

- Fehlererkennung von einzelnen Bitfehlern: Paritätsbit
  - Nutzdaten: $\mathrm{x} = (x_1 \; x_2 \; \ldots \; x_k)$
  - Paritätsbit: $\mathrm{p} = f(x) = x_1 + x_2 + \ldots + x_k \; mod \; 2$
  - Codewort: $\mathrm{z} = (xp)$

- Beispiel für $k = 4$ Nutzbits
  - Nutzdaten $\mathrm{x} = (1\ 0\ 1\ 0)$ werden codiert als $\mathrm{z} = (1\ 0\ 1\ 0\ 0)$
  - Fehler bei Übertragung: $\mathrm{z}^* = (1\ 1\ 1\ 0\ 0)$
  - Empfänger erkennt Fehler, da Paritätsbit verschieden
    $\mathrm{p}' = 1 \neq \mathrm{p}^* = 0$
    → Codewort ist ungültig



Hamming-Abstand

- Hamming-Abstand $d_{i,j}$
  - Anzahl der Bitpositionen, in denen sich zwei Codewörter c_i und c_j unterscheiden
  - Beispiel: Hamming-Abstand 3
    - Codewort c_i: 1 0 0 0 1 0 0 1
    - Codewort c_j: 1 0 1 1 0 0 0 1
    - $d_{i,j}(10001001, 10110001) = 3$ (Anzahl der Eins-Bits von `c_i XOR c_j`)
- Hamming-Abstand eines Codes d_{min}
  - Minimum aller Abstände zwischen Codewörtern innerhalb des Codes

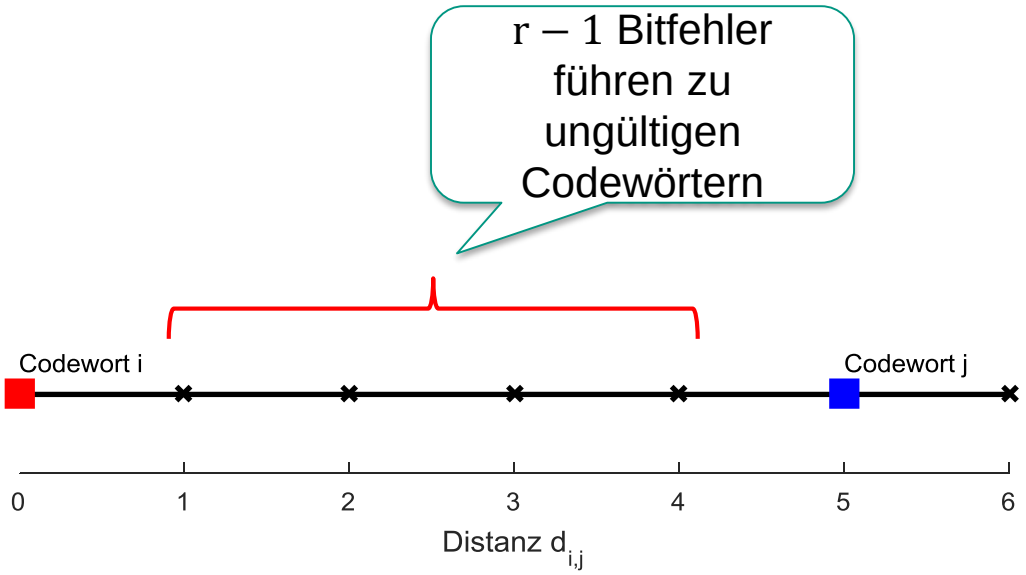
$$d_{min} := \min\{d(c_i, c_j) \mid c_i, c_j \in C, c_i \neq c_j\}$$

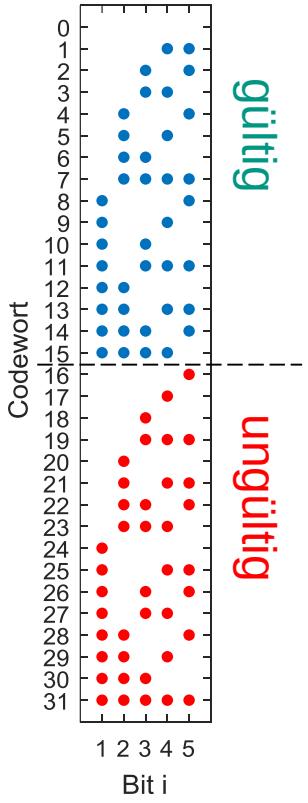
Beispiel: Paritätsbit

- Paritätsbit führt zu Hamming-Abstand $d_{min} = 2$
  - Nutzdaten unterscheiden sich in 1 Bit
    - → unterschiedliches Paritätsbit
    - → Hamming-Abstand $d_{i,j} = 2$
  - Nutzdaten unterscheiden sich in mehr als 1 Bit
    - → Hamming-Abstand $d_{i,j} \geq 2$

Fehlererkennung und -korrektur

- Gegeben: Minimaler Hamming-Abstand $\mathrm{r} = d_{min}$
- Fehlererkennung von bis zu $\mathrm{r} - 1$ Fehlern





Beispiele

- Fehlererkennender Code
  - Code mit einem Paritätsbit
  - Erkennen eines 1-Bitfehlers möglich

- Fehlerbehebender Code
  - Beispielcode B: `00000 00000, 00000 11111, 11111 00000, 11111 11111`
  - Hamming-Abstand des Codes B: $d_{min} = 5$
  - Korrektur von $\mathrm{r} = \left\lfloor \frac{d_{min}-1}{2} \right\rfloor = 2$ Bitfehlern möglich
  - Beispiel: `00000 00111` → `00000 11111`

Hamming-Kugel

- Hamming-Kugel von Codewort v mit dem Radius r

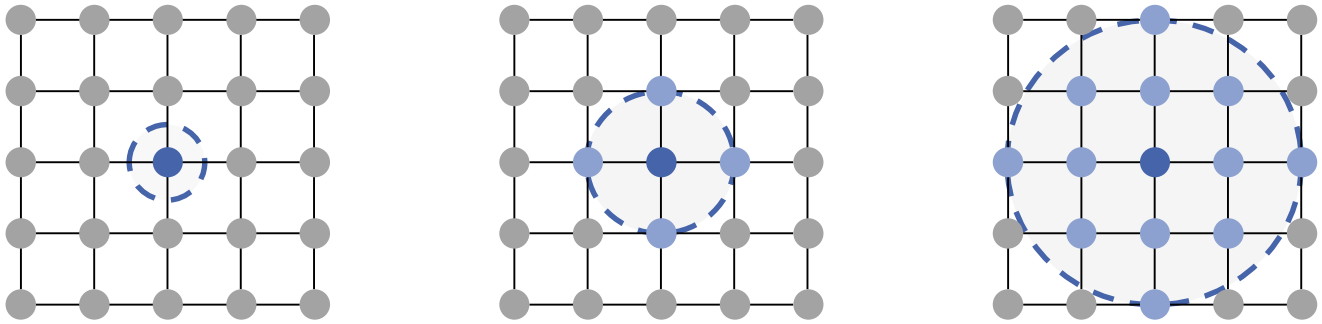
$$K_r(v) := \{w \in C \mid \Delta(v,w) \leq r\}$$

  - C: Code fester Länge, $v \in C$, $r \in N$

→ Codewörter, die sich von v an höchstens r Positionen unterscheiden

- Bei Codelänge von n kann man sich $K_r(v)$ als n–dimensonale Kugel mit Radius r vorstellen

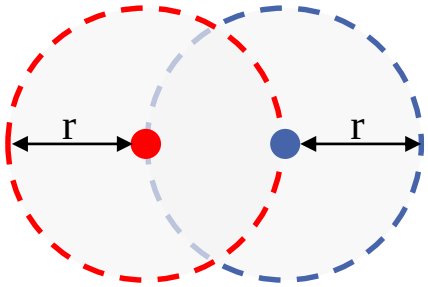
$K_0(v)$ $K_1(v)$ $K_2(v)$

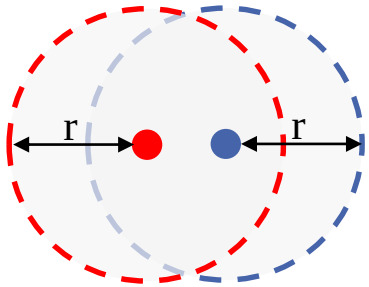


  - Punkt: Bitsequenz
  - Kante: verbundene Bitsequenzen unterscheiden sich an genau einer Position

r-fehlererkennende Codes

- Hamming-Kugeln zweier Codewörter so nah zusammen, dass Codewort *auf dem Rand* oder *innerhalb der anderen Kugel* liegt

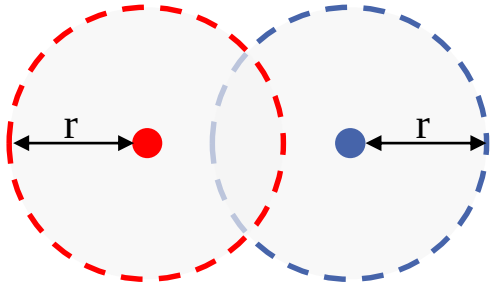




- Codewort kann durch ändern von r oder weniger Bits in anderes, gültiges Codewort überführt werden

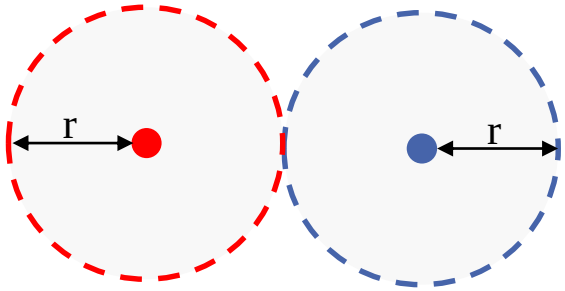
→ Code ist nicht r-fehlererkennend

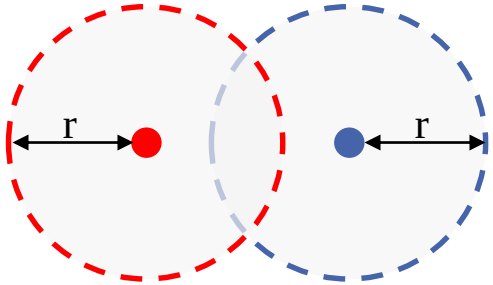
- Code ist r-fehlererkennend, wenn $d_{min} > r$



r-fehlererkorrigierende Codes

- Sind zwei Hamming-Kugeln so nah zusammen, dass sie sich
  berühren oder schneiden

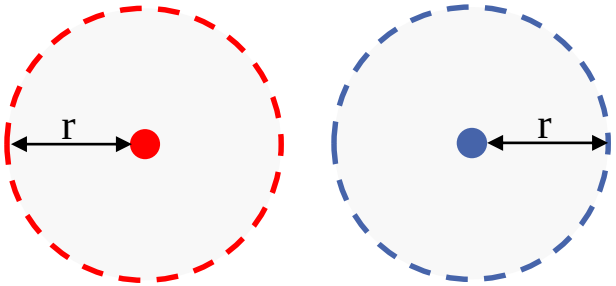




  - Codewörter in der Schnittmenge können durch ändern von r oder weniger Bits aus beiden Codewörtern entstanden sein

  → Code ist nicht r-fehlerkorrigierend

  - Code ist r-fehlererkorrigierend, wenn $d_{min} > 2r$



Fehlererkennung vs. Korrektur

- Hamming-Abstand bestimmt die Fähigkeit eines Codes, Fehler zu erkennen und zu beheben
  - Erkenne r-Bitfehler: Hamming-Abstand d_{min} von r+1 notwendig
  - Behebe r-Bitfehler: Hamming-Abstand d_{min} von $2r + 1$ notwendig

Hamming-Schranke

- Wieviel Prüfbits für r-fehlerkorrigierenden Code erforderlich?

m	r?

$$n = m + r$$

  - i Bitfehler auf n Bits verteilen: $\binom{n}{i}$ Möglichkeiten
  - Damit $2^m \times \sum_{i=0}^{r} \binom{n}{i}$ Wörter, die mit n Bits unterschieden werden müssen
  - Also $2^n \geq 2^m \times \sum_{i=0}^{r} \binom{n}{i}$

Beispiel: (7,4) Hamming Code

- 4 Bit Nutzdaten mit 3 Paritätsbits
  - Nutzdaten: $x = (x_1 \; x_2 \; x_3 \; x_4)$
  - Paritätsbits: $p = (p_1 \; p_2 \; p_3)$

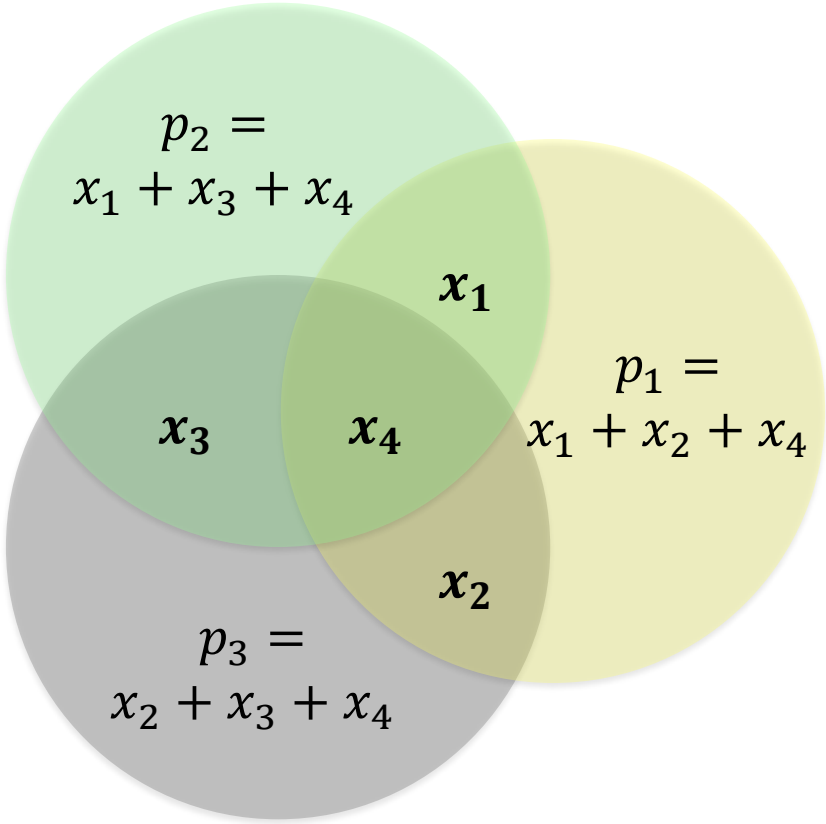
$$p_1 = x_1 + x_2 + x_4 \mod 2$$
$$p_2 = x_1 + x_3 + x_4 \mod 2$$
$$p_3 = x_2 + x_3 + x_4 \mod 2$$

- Zu übertragendes Codewort

$$z = (p_1 \; p_2 \; x_1 \; p_3 \; x_2 \; x_3 \; x_4)$$

Bits des Codeworts $z_0 \; z_1 \; z_2 \; z_3 \; z_4 \; z_5 \; z_6$

→ kann 1-Bit-Fehler korrigieren



Venn Diagramm

Anmerkung

- Berechnete Paritätsbits

$$
\begin{aligned}
p_1 &= x_1 + x_2 + x_4 \mod 2 \\
p_2 &= x_1 + x_3 + x_4 \mod 2 \\
p_3 &= x_2 + x_3 + x_4 \mod 2
\end{aligned}
$$

- Es gilt

$$
\begin{aligned}
0 &= p_1 + x_1 + x_2 + x_4 \mod 2 \\
0 &= p_2 + x_1 + x_3 + x_4 \mod 2 \\
0 &= p_3 + x_2 + x_3 + x_4 \mod 2
\end{aligned}
$$

(7,4) Hamming Code

- Paritätsbits

$$p_1 = x_1 + x_2 + x_4 \mod 2$$
$$p_2 = x_1 + x_3 + x_4 \mod 2$$
$$p_3 = x_2 + x_3 + x_4 \mod 2$$

- Codewort
  - $z = (p_1\ p_2\ x_1\ p_3\ x_2\ x_3\ x_4)$

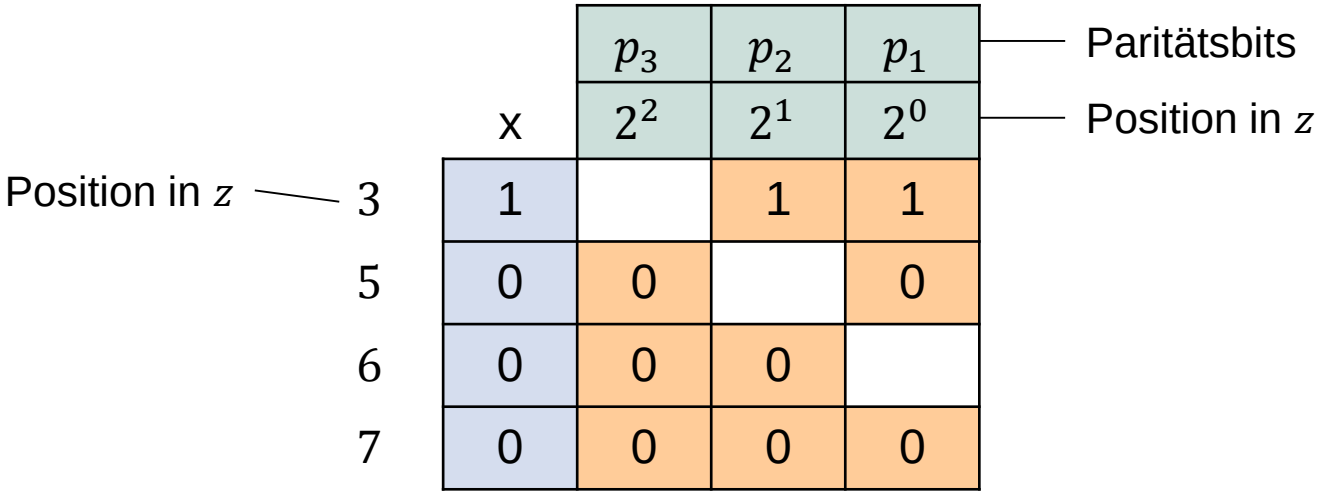
- Beispiel
  - Nutzdaten: $x = (1\ 0\ 0\ 0)$
  - Paritätsbits: $p_1 = 1 + 0 + 0 \mod 2 = 1$
    $p_2 = 1 + 0 + 0 \mod 2 = 1$
    $p_3 = 0 + 0 + 0 \mod 2 = 0$
  - Codewort: $z = (1\ 1\ 1\ 0\ 0\ 0\ 0)$

(7,4) Hamming Code

- Paritätsbits

$$p_1 = x_1 + x_2 + x_4 \mod 2$$
$$p_2 = x_1 + x_3 + x_4 \mod 2$$
$$p_3 = x_2 + x_3 + x_4 \mod 2$$

- Hamming-Tabelle



(7,4) Hamming Code

- Übertragen über fehlerhaften Kanal: $z^* = z + e$
  - e: Fehlervektor (1 bedeutet Bitfehler)
  - $z^* + e = z$

Empfänger kennt e nicht und z auch nicht

- Empfänger: Bitfehler in den Nutzdaten

	x	p_3 2^2	p_2 2^1	p_1 2^0
3	1		1	1
5	0	0		0
6	1	1	1	
7	0	0	0	0

0	1	1
1	0	1
1	1	0

Empfangene Paritätsbits

Berechnete Paritätsbits

→ *Bitfehler an Position 6*

XOR gleich Null → fehlerfreie Übertragung

(7,4) Hamming Code

- Übertragen über fehlerhaften Kanal: $z^* = z + e$
  - e: Fehlervektor (1 bedeutet Bitfehler)
  - $z^* + e = z$

Empfänger kennt e nicht und z auch nicht

- Empfänger: Bitfehler in den Paritätsbits

	x	p_3 2^2	p_2 2^1	p_1 2^0
3	1		1	1
5	0	0		0
6	0	0	0	
7	0	0	0	0

0	1	1	Berechnete Paritätsbits
0	0	1	Empfangene Paritätsbits
0	1	0	→ *Bitfehler an Position 2*

Kontrollmatrix

- Es sei H die Kontrollmatrix eines linearen Codes. H hat folgende Eigenschaft

$$w \in V \leftrightarrow w * H^T = (0 \ldots 0)$$

- … vgl. vorne

$$\begin{aligned} 0 &= p_3 + x_2 + x_3 + x_4 \mod 2 \\ 0 &= p_2 + x_1 + x_3 + x_4 \mod 2 \\ 0 &= p_1 + x_1 + x_2 + x_4 \mod 2 \end{aligned}$$

$$\begin{aligned} 0 &= z_4 + z_3 + z_5 + z_6 \mod 2 \\ 0 &= z_1 + z_2 + z_5 + z_6 \mod 2 \\ 0 &= z_0 + z_2 + z_4 + z_6 \mod 2 \end{aligned}$$

  - $z = (p_1\ p_2\ x_1\ p_3\ x_2\ x_3\ x_4)$

- Kontrollmatrix des (7,4) Hamming Codes

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Syndrom

- Syndrom
  - Es sei H die Kontrollmatrix eines linearen Codes
  - Der Vektor $w = H^T$ heißt Syndrom von w

- Berechnen des Syndroms $S = (s_1\ s_2\ s_3)$ beim Empfänger

$$
\begin{aligned}
s_1 &= x_1^* + x_3^* + x_4^* + p_1^* \mod 2 \\
s_2 &= x_1^* + x_2^* + x_4^* + p_2^* \mod 2 \\
s_3 &= x_2^* + x_3^* + x_4^* + p_3^* \mod 2
\end{aligned}
$$

Reduktion der Komplexität!
Im Beispiel: von 128 Vektoren auf 8 Syndrome

- $S = z^* \cdot H^T = (z + e) \cdot H^T = z \cdot H^T + e \cdot H^T = e \cdot H^T$

- Für Kontrollmatrix von vorne gilt
  - Syndrom ist mit Bitposition des Fehlers identisch

- Syndrom $S = (0\ 0\ 0)$ → Übertragung war fehlerfrei

Pingo

http://pingo.upb.de/

Pingo

http://pingo.upb.de/

Internet-Prüfsumme

- Prüfsumme bei Internet-Protokollen
  - Speziell für Realisierung in Software ausgelegt
  - Verwendet bei: UDP, TCP und IP

- Prinzip
  - Aufaddieren aller übertragenen Wörter (16 Bit Wortlänge)
    - Wörter werden als **`Integer`** aufgefasst
    - $Prüfsumme = \sum alle\ übertragenen\ Wörter$
  - Prüfsumme wird im Paket übertragen
  - Anmerkung
    - Wörter in falscher Reihenfolge können nicht erkannt werden

- Implementierung
  - Addition unter Verwendung des Einer-Komplements
  - RFC 1071: Hinweise und Techniken für effiziente Implementierungen

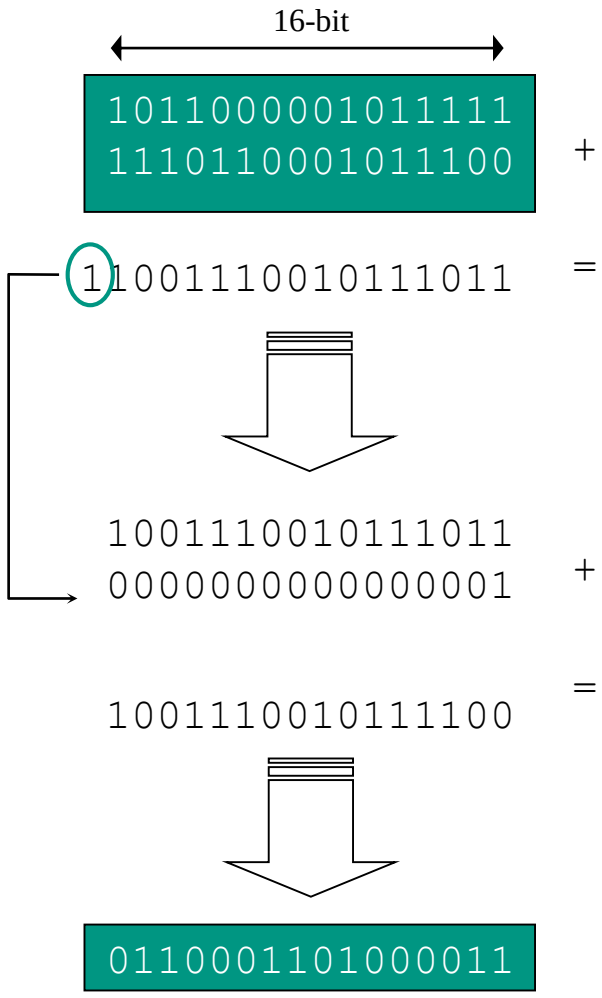
[RFC1071]

Rechenbeispiel Internet-Prüfsumme

(1) 16-Bit-Worter addieren

(2) Übertrag addieren

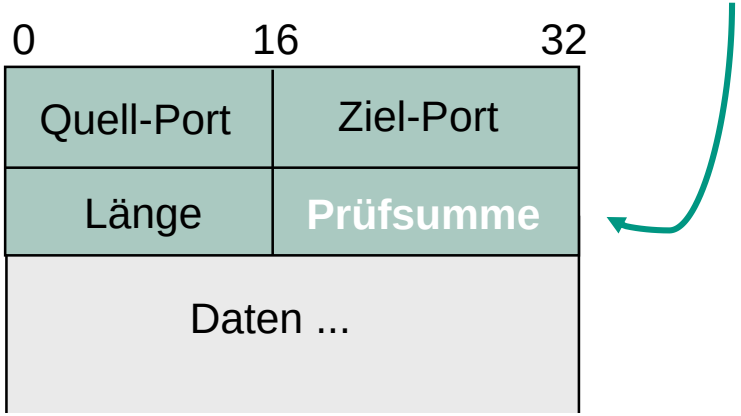
(3) Einerkomplement bilden



[Benv05]

UDP-Prüfsumme

- Sender
  - UDP-Segment (inkl. UDP-Kopf) wird als Folge von 16 bit Wörtern aufgefasst
  - Berechnung der Prüfsumme
  - Ergebnis in Feld Prüfsumme im UDP-Kopf eintragen



- Empfänger
  - Prüfsumme des empfangenen UDP-Segments berechnen
  - Berechnete Prüfsumme mit empfangener Prüfsumme vergleichen
    - Werte ungleich → Fehler erkannt
    - Werte gleich → kein Fehler erkannt … es könnte aber doch noch Fehler existieren

Kapitel 3.4.2 Fehlerkontrolle bei Paketfehlern

Fehlerkontrolle bei Paketfehlern

- Mit Paritätsbits und Prüfsummen können Bitfehler erkannt werden, allerdings nur dann, wenn das Paket beim Empfänger ankommt oder in den netzinternen Zwischensystemen analysiert wird

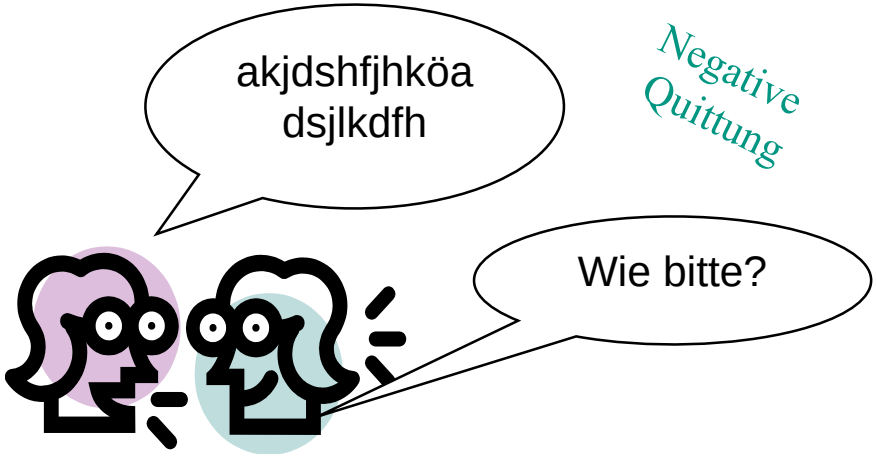
- Zur Erkennung von Fehlern bzgl. kompletter Pakete (Paketfehler) sind zusätzliche Mechanismen erforderlich
  - Sequenznummern (*Sequence Number*)
  - Zeitgeber (*Timer*)

- Zur Behebung der Fehler werden die folgenden Mechanismen verwendet
  - Quittungen (*Acknowledgements*)
  - Sendewiederholungen (*Retransmissions*)

… Verständigungsprobleme?

Sendewiederholung

Negative Quittung



Positive Quittung

Sequenz-nummern

1, 2, 3, 5, … etwas fehlt hier …

Sequenznummern

- Problem
  - Woher weiß der Empfänger, ob
    - Pakete in der richtigen Reihenfolge ankommen?
    - keine Duplikate enthalten sind?
    - keine Pakete fehlen?
- Mechanismus
  - Pakete (oder die Bytes) werden durchnummeriert
  - Entsprechende Kennung wird mit jedem Paket übertragen
  - Kennung wird als Sequenznummer bezeichnet
- Fehlerszenarien
  - Sequenznummern helfen, wenn Pakete nicht ausgeliefert werden
    - Beispiele?
- Größe
  - Bei einer Länge der Sequenznummer von n Bit umfasst der Sequenznummernraum 2^n Sequenznummern

Quittungen

- Problem
  - Wie erfährt der Sender, dass ein Paket überhaupt nicht bzw. nicht korrekt beim Empfänger angekommen ist?
- Mechanismus
  - Empfänger informiert Sender, ob er Paket empfangen hat oder nicht
  - Versendung spezieller Pakete, sogenannte Quittungen (ACK: Acknowledgement)
- Varianten
  - Positive Quittung
    - Empfänger teilt dem Sender mit, dass er die Daten erhalten hat (ACK)
  - Negative Quittung
    - Empfänger meldet dem Sender, dass er die Daten nicht erhalten hat (z.B. wenn er nachfolgendes Paket erhält)
    - NACK: Negative Acknowledgement

Quittungen

- Weitere Varianten
  - Selektive Quittungen
    - SACK: Selective Acknowledgement
    - Quittung bezieht sich auf ein einzelnes Paket
    - Beispiel
      - Negative selektive Quittung, falls der Verlust eines Pakets vom Empfänger vermutet wird (NACK)
  - Kumulative Quittungen
    - Quittung bezieht sich auf eine Menge von Paketen, die in der Regel durch eine obere Sequenznummer beschränkt ist
    - Beispiel
      - Positive kumulative Quittung, die besagt, dass alle Pakete bis zur angegebenen Sequenznummer korrekt empfangen wurden
- Quittungen werden oftmals in Kombination mit Zeitgebern verwendet

Pingo

http://pingo.upb.de/

Zeitgeber

- Problem
  - Woran merkt ein Sender, dass ein Paket nicht angekommen ist?
- Mechanismus
  - In Abhängigkeit einer zeitlichen Obergrenze wird vermutet, dass ein Paket beim Empfänger nicht angekommen ist
  - Sender kann dann Sendewiederholung starten
- Implementierung
  - Welcher Wert wird für den Zeitgeber gewählt?

Automatic Repeat Request: ARQ

- Grundlegende Variante zur Sendewiederholung
  - Sender erhält positive Quittungen über den Erhalt eines Pakets vom Empfänger
  - Sender kann Sendewiederholungen ausführen

- Varianten
  - Wann werden Quittungen versendet?
  - Wann werden Sendewiederholungen veranlasst?

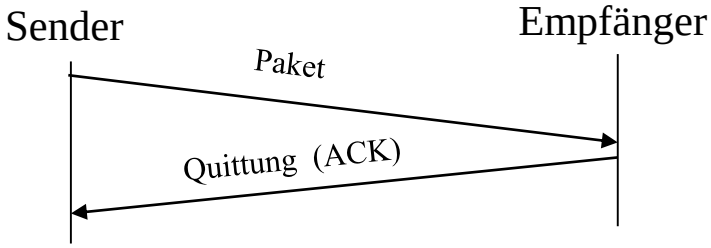
Stop-and-Wait

- Sehr einfaches ARQ-Verfahren
  - Sender wartet auf Quittung zu einem gesendeten Paket, bevor er neues Paket senden darf
  - Falls keine Quittung empfangen wird, erfolgt Wiederholung des Pakets
  - Wartezeit auf Quittung wird durch Zeitgeber geregelt

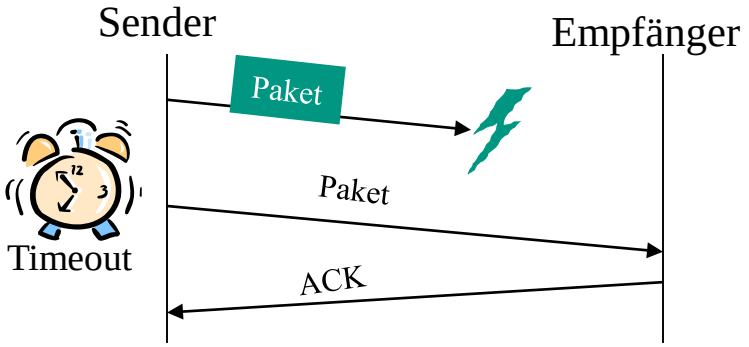
- Einsatzbeispiel
  - WLAN

Stop-and-Wait: Szenarien

Regulärer Ablauf

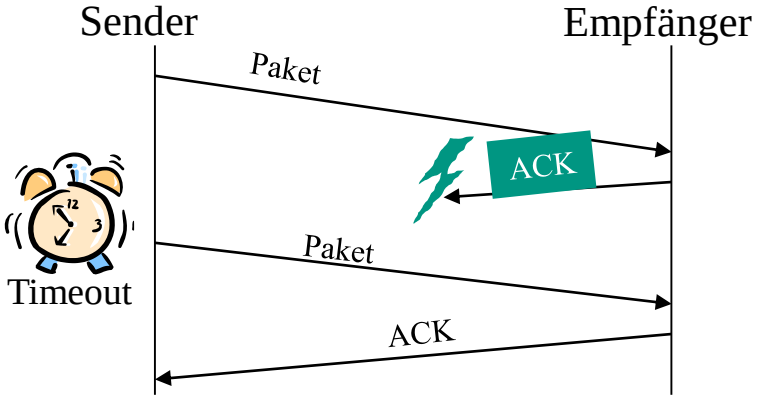


Verlust eines Pakets



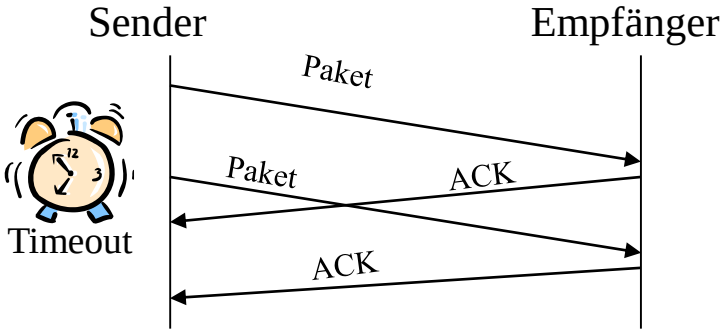
Verlust einer Quittung

- Unterschied zum Verlust eines Pakets?



Zu schneller Ablauf des Zeitgebers

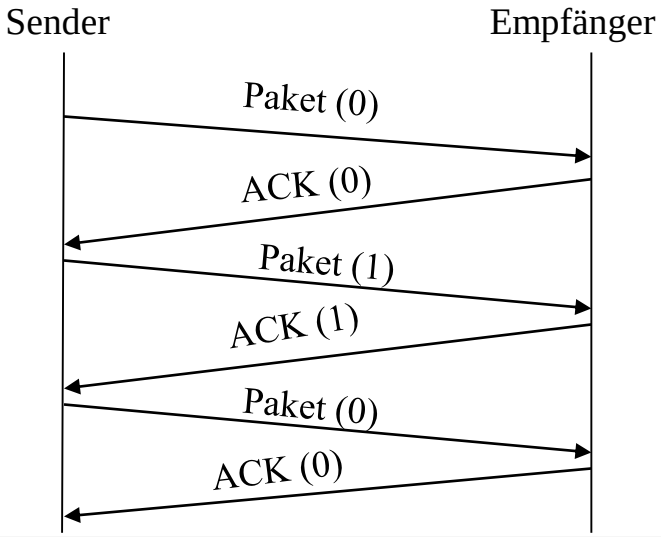
- Problem?



- Frage: Auf welchen Wert wird der Zeitgeber gesetzt?
- Frage: Unterscheiden sich Verluste bzw. Übertragungsfehler in der Behandlung?

Stop-and-Wait: Sequenznummern

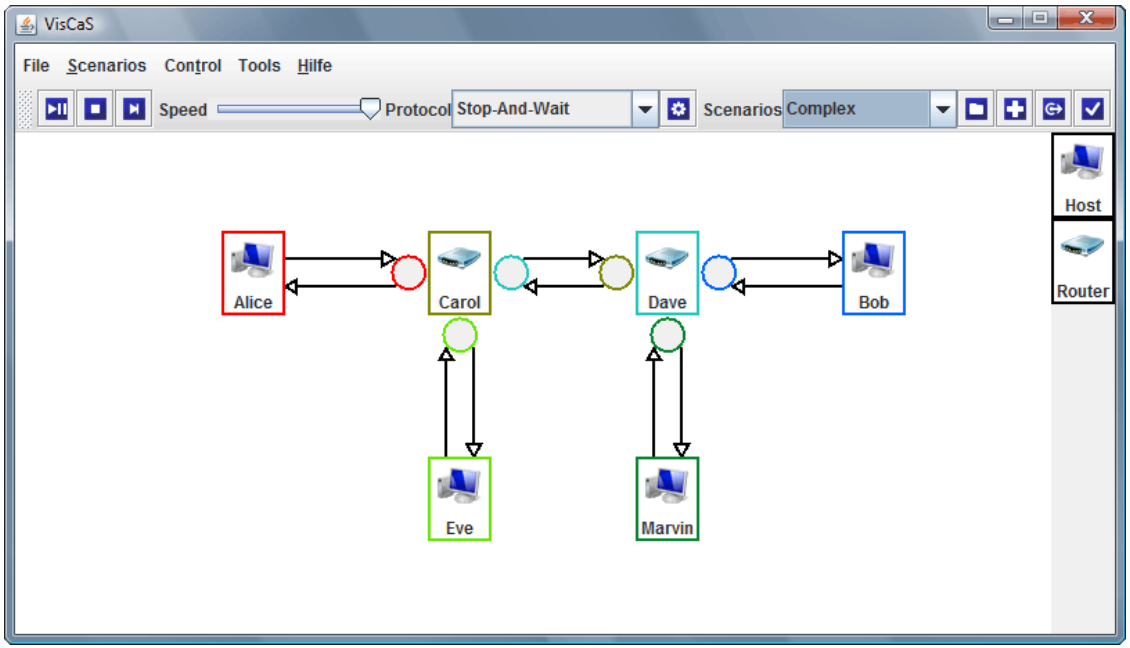
- Problem
  - In den vorangegangenen Szenarien besteht die Möglichkeit, dass der Empfänger ein Paket doppelt erhält
    - Er kann dies nicht erkennen
- Mechanismus
  - Sequenznummern
    - Die Pakete werden mit einer Kennung versehen, die es dem Empfänger ermöglicht, diese zu unterscheiden
    - Für Stop-and-Wait ist eine Sequenznummer von einem Bit ausreichend (0 und 1)
- Ablauf



Java-Anwendung: Stop-and-Wait

Testen Sie unsere Java-Anwendung und Lernumgebung im Internet:
http://www.tm.kit.edu/software/viscas/

- Einfaches Szenario mit zwei direkt verknüpften Rechnern
- Testen Sie die Leistungsfähigkeit der Fehlerkontrolle
  - Experimentieren Sie mit unterschiedlichen Verlust- bzw. Fehlerraten
  - Experimentieren Sie mit unterschiedlichen Datenraten



Leistungsbewertung

- Nachteil Stop-and-Wait
  - Sender darf nur ein Paket senden und muss dann auf Quittung warten

→ Wie sieht es mit der erzielbaren Leistungsfähigkeit aus?

- Kriterien zur Leistungsbewertung
  - Durchsatz (engl. *Throughput*)
    - Maß für die Menge an Daten, die pro Zeiteinheit übertragen werden kann
  - Auslastung (engl. *Utilization* – U)
    - Maß für die Auslastung einer Ressource (z.B. Übertragungsmedium)
    - Verhältnis von tatsächlicher Nutzung zu möglicher Nutzung
      - Tatsächliche Nutzung: erfolgreich übertragene Daten bzw. Pakete
      - Mögliche Nutzung: Wie viele Daten bzw. Pakete hätten in dieser Zeit übertragen werden können?

Leistungsparameter

- Durchsatz
  - Synonym: Datenrate
  - Gemessen in bit/s

  … kann an unterschiedlichen Stellen eines Kommunikationssystems angegeben / gemessen werden

Anmerkung: Begriff Bandbreite manchmal als Synonym verwendet. Bezeichnet aber auch die Breite eines Frequenzbands

Leistungsparameter

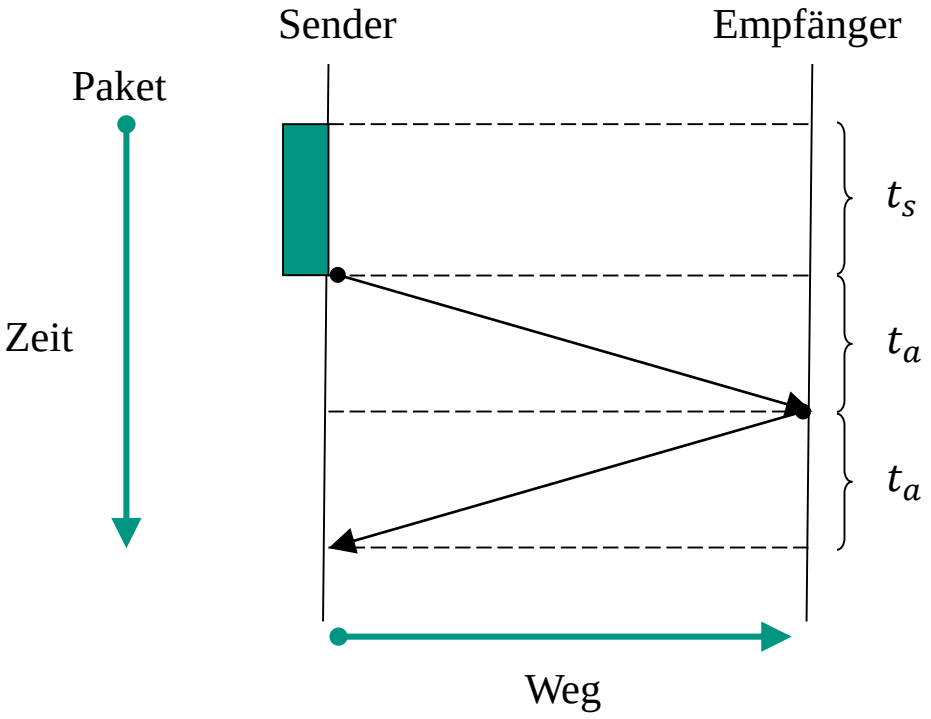
- Verzögerung / Latenz
  - Verarbeitungsverzögerung (engl. *Processing Delay*)
    - Zeit, die nötig ist um ein Paket zu verarbeiten
      - z.B. Berechnung von Prüfsummen
  - Warteschlangenverzögerung (engl. *Queuing Delay*)
    - Zeit, die gewartet werden muss bis ein Paket gesendet werden kann
    - abhängig von der Auslastung des Netzes
  - Sendezeit (engl. *Transmission Delay*)
    - Zeit, die nötig ist um n Bits zu senden
    - abhängig von der Datenrate
  - Ausbreitungsverzögerung (engl. *Propagation Delay*)
    - Zeit, die ein Bit von A nach B benötigt
    - abhängig von der Länge des Mediums und der Ausbreitungsgeschwindigkeit

Stop-and-Wait: Leistungsbewertung

- Annahme: Fehlerfreie Kommunikation
- Welche Parameter haben einen Einfluss?
  - Wie schnell ist ein komplettes Paket bzw. eine Quittung auf das Medium gesendet?
    - Sendezeit t_s
      - t_s = Länge Paket / Datenrate
  - Wie lange benötigt ein Bit vom Sender zum Empfänger (und umgekehrt)?
    - Ausbreitungsverzögerung t_a
      - t_a = Länge des Mediums / Ausbreitungsgeschwindigkeit
  - Wie viel Zeit wird für die Verarbeitung eines Pakets bzw. einer Quittung benötigt?
    - Verarbeitungszeit t_V
- Vereinfachungen
  - Verarbeitungszeit des Pakets wird vernachlässigt
  - Sende- und Verarbeitungszeit einer Quittung werden vernachlässigt
    - Quittung klein; Verarbeitung schnell

Stop-and-Wait: Gesamtübertragungsdauer

- Insgesamt benötigte Zeit t_{Ges}
  - $t_{Ges} = t_s(\text{Daten}) + t_a + t_V(\text{Daten}) + t_s(\text{Quittung}) + t_a + t_V(\text{Quittung})$
- Mit Vereinfachungen
  - $t_{Ges} = t_s(\text{Daten}) + 2t_a = t_s + 2t_a$



Stop-and-Wait: Auslastung des Mediums

- Auslastung U
  - Tatsächliche Nutzung
    - Sendezeit des Pakets: t_s
  - Mögliche Nutzung
    - Zeitintervall vom Beginn des Sendens des Pakets bis zum vollständigen Empfang der Quittung: t_{Ges}

$$U = \frac{t_s}{t_{Ges}} = \frac{t_s}{t_s + 2t_a} \qquad \text{bzw.} \qquad U = \frac{1}{1 + 2t_a/t_s}$$

- Mit $a = t_a/t_s$ ergibt sich

$$U = \frac{1}{1 + 2a}$$

Parameter a

- Definition
  - $a = t_a / t_s$ = Ausbreitungsverzögerung / Sendezeit
- Einflussgrößen
  - Länge des Mediums: m
  - Ausbreitungsgeschwindigkeit: v
  - Länge der Dateneinheit: X
  - Datenrate: r

$$\rightarrow \quad a = \frac{\frac{m}{v}}{\frac{X}{r}} = \frac{\frac{m}{v} r}{X}$$

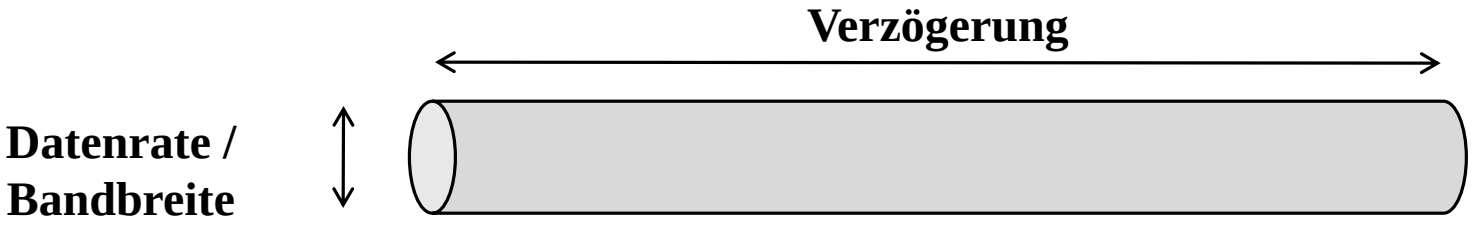
Bandbreiten-Verzögerungs-Produkt

- Interpretation
  - Bandbreiten-Verzögerungs-Produkt $(m/v * r)$: Länge des Mediums in Bit

  - Parameter a repräsentiert das Verhältnis der Länge des Mediums in Bit zur Länge der Dateneinheit

Bandbreiten-Verzögerungs-Produkt

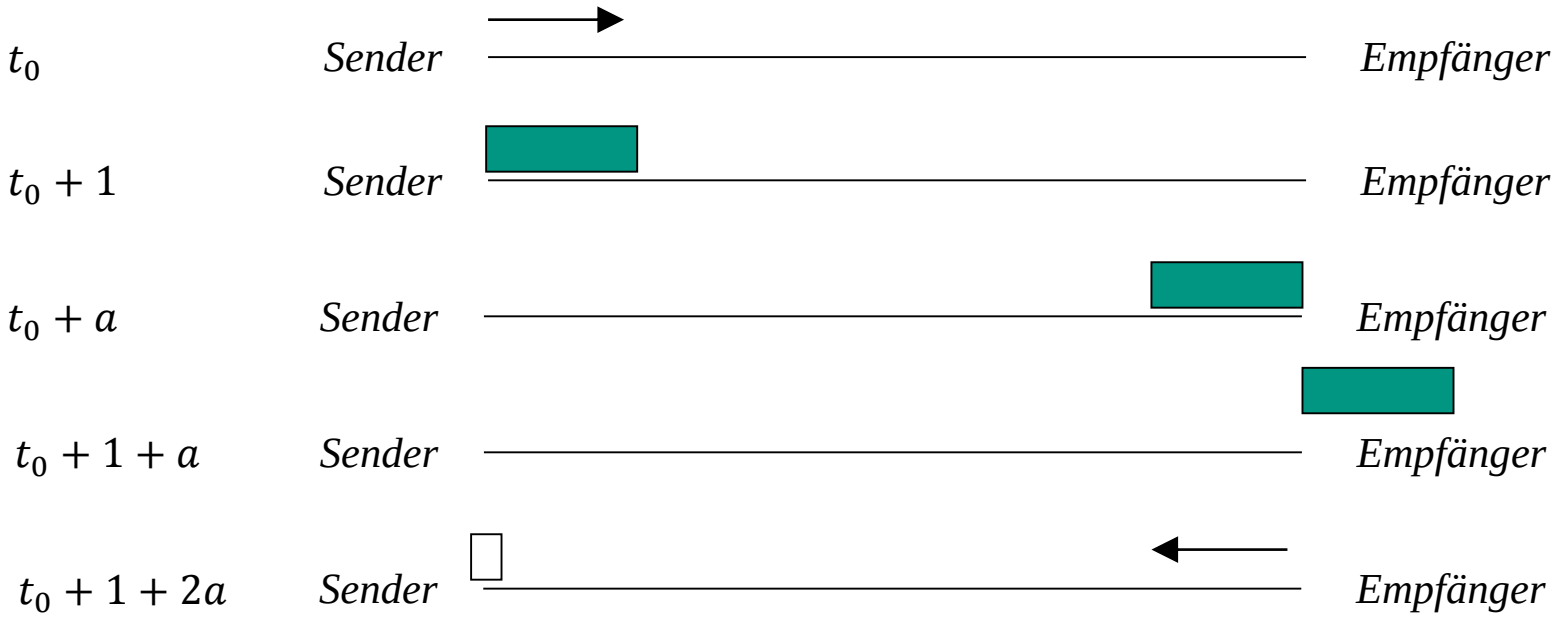
- *Bandbreite * Verzögerung*
- *Bandbreite * RTT / 2*



Anschluss	Datenrate (typisch)	Entfernung (typisch)	Round-Trip-Time	Bandbreiten-Verzögerungs-Produkt
Modem	56 kbit/s	10 km	87 µs	2,5 bits
WLAN	54 Mbit/s	50 m	0,33 µs	9 bits
Satellit	45 Mbit/s	35.786 km	230 ms	5,4 Mbit
Überland-Glasfaser	10 Gbit/s	4.000 km	40 ms	200 Mbit

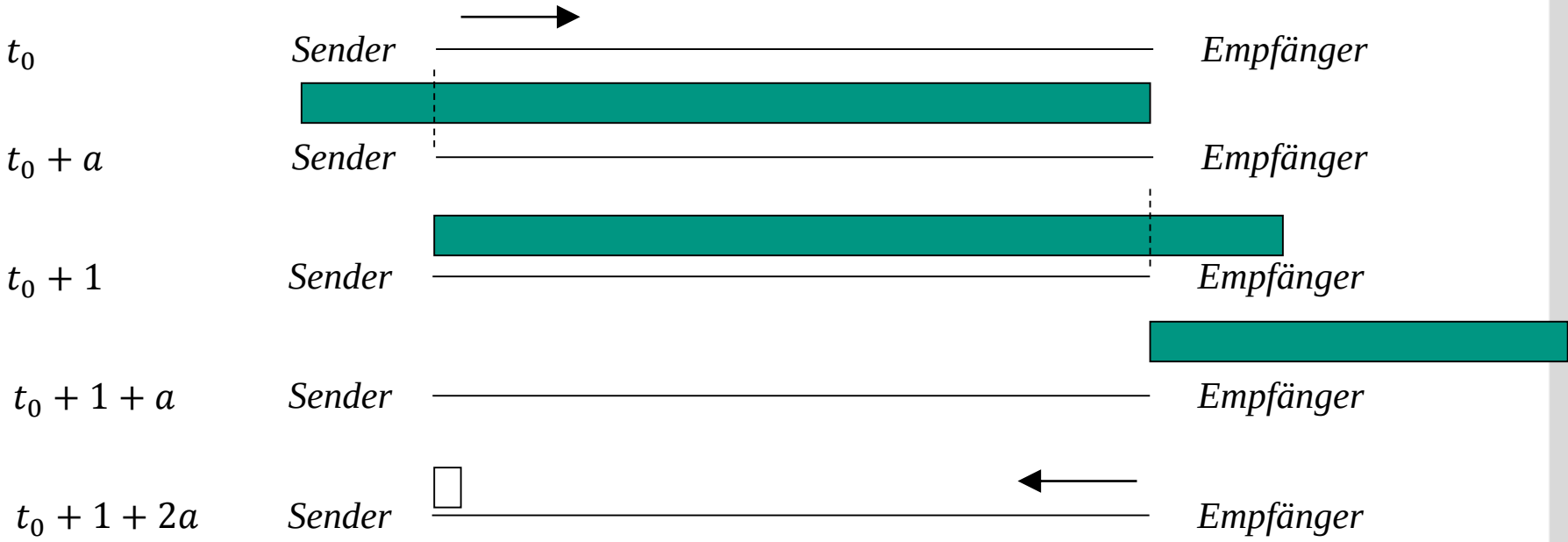
Beispiel zu Parameter a

- Normalisierung
  - Annahme: Sendezeit eines Pakets betrage 1
- Damit gilt
  - $a = t_a$
  - Für $a > 1$: Ausbreitungsverzögerung > Sendezeit
    d.h. Paket passt vollständig auf das Medium



Beispiel zu Parameter a

- Für $a < 1$: Ausbreitungsverzögerung < Sendezeit
  d.h. Paket passt *nicht* vollständig auf das Medium



Stop-and-Wait: Auslastung des Mediums

- Beispiel
  - Pakete der Länge $1000\ \text{Bit}$
  - Datenraten von $1\ \text{kbit/s}$ und $1\ \text{Mbit/s}$
  - Fehlerrate vernachlässigbar

- Drei unterschiedliche Medientypen
  - Verdrilltes Adernpaar (Ausbreitungsgeschwindigkeit $2 * 10^8\ \text{m/s}$), Länge des Mediums $1\ \text{km}$
    - Auslastung?
  - Standleitung (Ausbreitungsgeschwindigkeit $2 * 10^8\ \text{m/s}$), Länge des Mediums $200\ \text{km}$
    - Auslastung?
  - Satelliten-Verbindung (Ausbreitungsgeschwindigkeit $3 * 10^8\ \text{m/s}$), Länge des Mediums $50.000\ \text{km}$
    - Auslastung?

Stop-and-Wait: Leistungsbewertung

- Jetzt
  - Berücksichtigung von Übertragungsfehlern
- Fehlerfall: Sender erhält keine korrekte Quittung
- Annahme: $n-1$ konsekutive Sendewiederholungen
  - Benötigte Zeit hierfür
    - $t_{Ges} = t_s + (n-1)(\text{Timeout} + t_s) + 2t_a$
  - Annahme
    - Timeout entspricht zweifacher Ausbreitungsverzögerung t_a
    - Dann gilt: $t_{Ges} = n(t_s + 2t_a)$
  - Damit gilt für die Auslastung

$$U = \frac{t_s}{t_{Ges}} = \frac{t_s}{n(t_s + 2t_a)}$$

  - … in der Regel n nicht fest: Erwartungswert?

Stop-and-Wait: Leistungsbewertung

- Annahmen
  - p sei Wahrscheinlichkeit, dass ein Paket fehlerhaft übertragen wird
  - Quittungen seien nie verfälscht

- Wahrscheinlichkeit für genau k Übertragungsversuche?
  - $k-1$ fehlerhafte Übertragungsversuche gefolgt von einer erfolgreichen Übertragung
    - Wahrscheinlichkeit hierfür: $p^{k-1}(1-p)$

  - $n =$ Erwartungswert [Anzahl Übertragungen]

$$n = \sum_{i=1}^{\infty} (i \times P[\text{genau } i \text{ Übertragungen}]) = \sum_{i=1}^{\infty} i p^{i-1}(1-p) = \frac{1}{1-p}$$

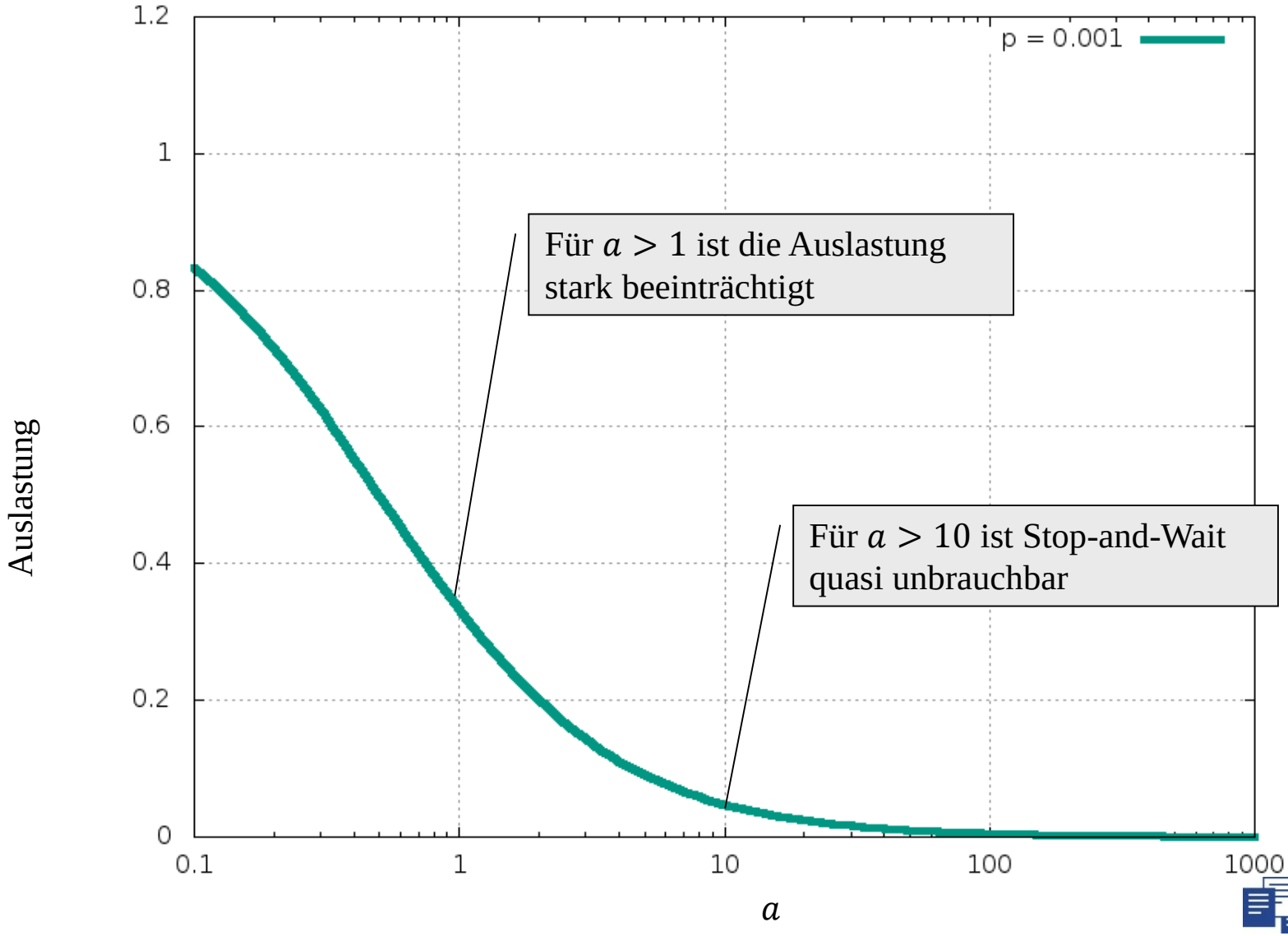
  - Damit gilt für die Auslastung

$$U = \frac{1}{n(1+2a)} = \frac{1-p}{1+2a}$$

für $-1 < x < 1$ gilt:

$$\sum_{i=1}^{\infty} i x^{i-1} = \frac{1}{(1-x)^2}$$

Bewertung von Stop-and-Wait



[Stal06]

Go-Back-N ARQ

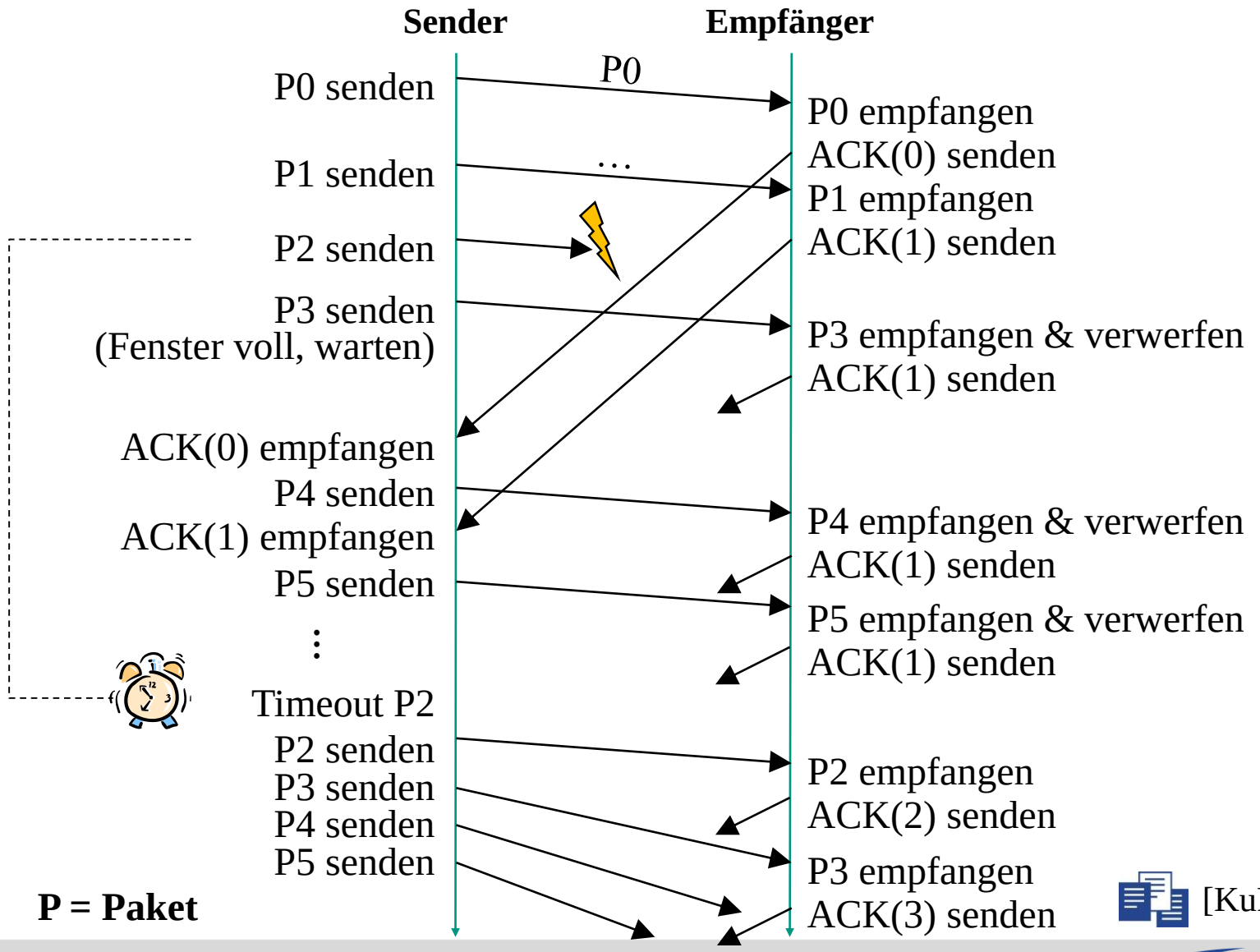
- Ziel
  - Erhöhung der Leistungsfähigkeit im Vergleich zu Stop-and-Wait
- Datenaustausch
  - Sender
    - Kann *mehrere* Pakete senden bis er Quittung erhalten muss
    - Maximale Anzahl der nicht quittierten Pakete ist begrenzt
    - Typischerweise durch ein Fenster (Window) auf Senderseite
  - Empfänger
    - Quittiert i.d.R. mit kumulativen Quittungen

Go-Back-N ARQ

- Verhalten im Fehlerfall
  - Empfänger
    - Empfang eines fehlerhaften Pakets oder eines Pakets außerhalb der Reihenfolge
      - Verwerfen aller nachfolgenden Pakete
  - Sender
    - Ablauf des entsprechenden Zeitgebers
      - Wiederholen aller noch nicht quittierten Pakete

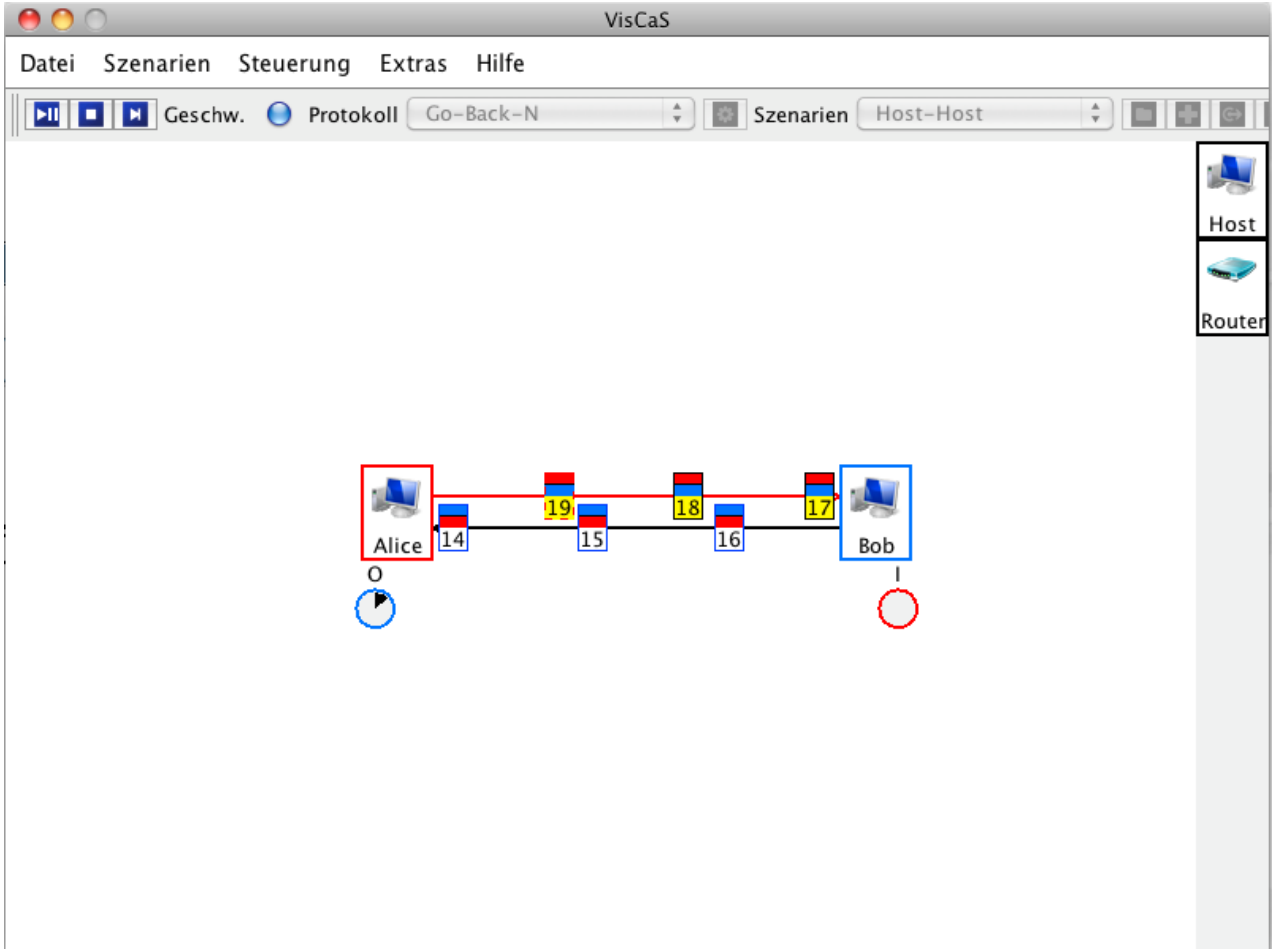
- Fragen
  - Wo ist Pufferung der Pakete erforderlich?
  - Wie viele müssen gepuffert werden?

Go-Back-N: Beispielablauf



[KuRo12]

Visualisierung: Go-back-N und Flusskontrolle



http://www.tm.kit.edu/software/viscas/

Go-Back-N: Leistungsbewertung

- Bemerkung
  - Fenstergröße W begrenzt maximale Anzahl gesendeter und noch nicht quittierter Pakete
- Annahme: Fehlerfreie Übertragung

- Zwei Fälle
  - Fenstergröße $W \geq 1 + 2a$
    - Sender kann ohne Pause senden
    - Übertragungsabschnitt ist 100% ausgelastet
  - Fenstergröße $W < 1 + 2a$
    - Sender kann nach dem Aufbrauchen des Fensters nicht weiter senden

- Erzielbare Auslastung

$$U = \begin{cases} 1 & W \geq 1 + 2a \\ \dfrac{W}{1 + 2a} & W < 1 + 2a \end{cases}$$

Go-Back-N: Leistungsbewertung

- *Jetzt*: Berücksichtigung von Übertragungsfehlern

- Herleitung ähnlich wie bei Stop-and-Wait, aber
  - Im Fehlerfall werden K Pakete wiederholt anstatt einem
  - $n =$ Erwartungswert [Anzahl übertragener Pakete für eine erfolgreiche Übertragung]
    - $f(i)$: Anzahl übertragener Pakete, falls das ursprünglich gesendete Paket i mal übertragen werden muss

$$f(i) = 1 + (i-1)K = (1-K) + Ki$$

    - Daraus ergibt sich für den Erwartungswert n

$$n = \sum_{i=1}^{\infty} f(i) p^{i-1}(1-p)$$

Go-Back-N: Leistungsbewertung

- Einsetzen von $f(i)$

$$n = \sum_{i=1}^{\infty} ((1-K) + Ki)p^{i-1}(1-p)$$

für $-1 < x < 1$ gilt:

$$\sum_{i=1}^{\infty} x^{i-1} = \frac{1}{1-x}$$

$$n = (1-K)\sum_{i=1}^{\infty} p^{i-1}(1-p) + K\sum_{i=1}^{\infty} ip^{i-1}(1-p)$$

für $-1 < x < 1$ gilt:

$$\sum_{i=1}^{\infty} ix^{i-1} = \frac{1}{(1-x)^2}$$

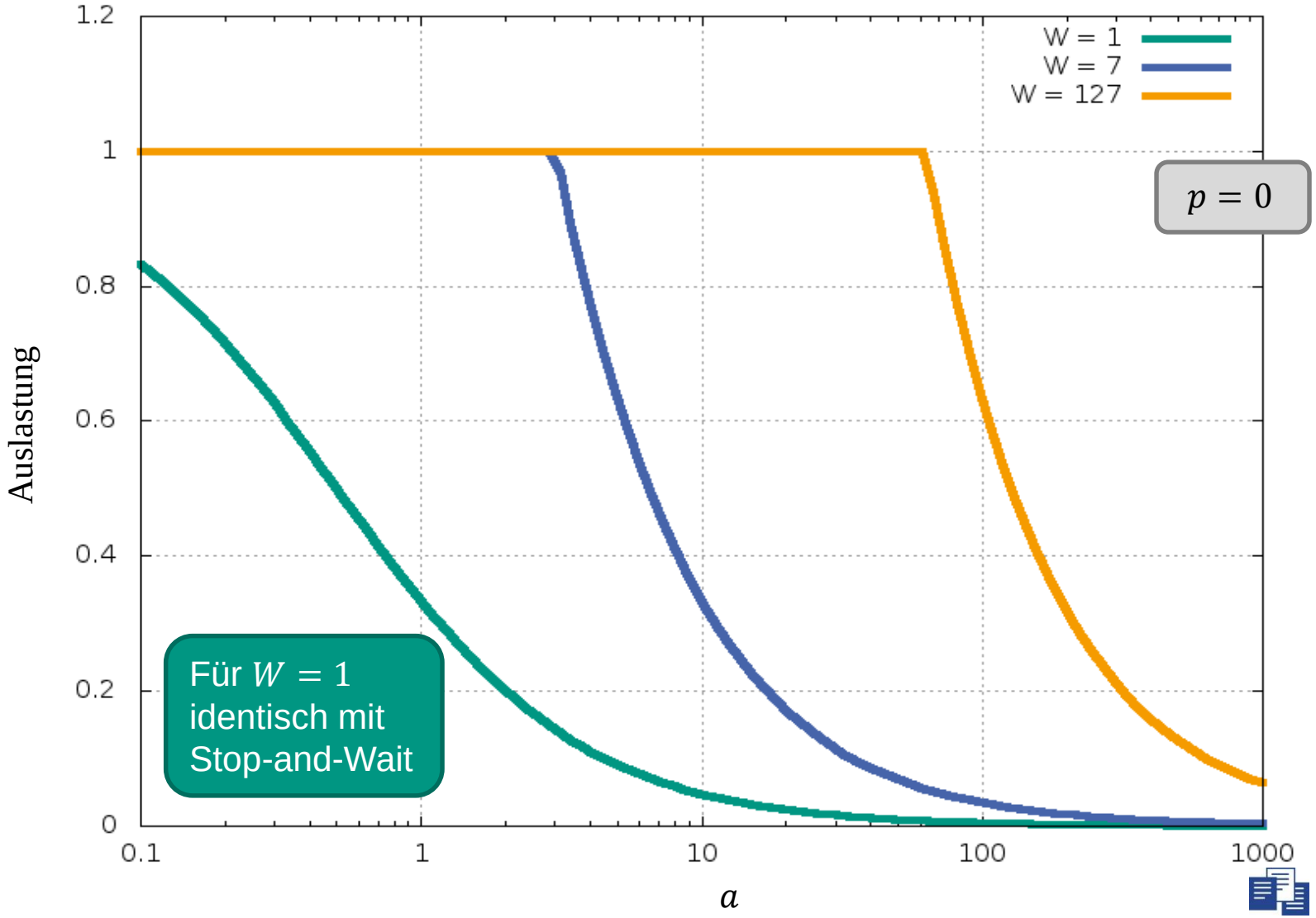
$$n = 1 - K + \frac{K}{1-p} = \frac{1-p+Kp}{1-p}$$

  - Für $W \geq (1+2a)$: K ungefähr $1+2a$
  - Für $W < (1+2a)$: $K = W$

- Damit

$$U = \begin{cases} \dfrac{1-p}{1+2ap} & W \geq 1+2a \\ \dfrac{W(1-p)}{(1+2a)(1-p+Wp)} & W < 1+2a \end{cases}$$

Bewertung von Go-Back-N



[Stal06]

Selective Repeat ARQ

- Ziel
  - Erhöhung der Auslastung im Vergleich zu Stop-and-Wait
  - Reduzierung des Datenaufkommens im Vergleich zu Go-Back-N

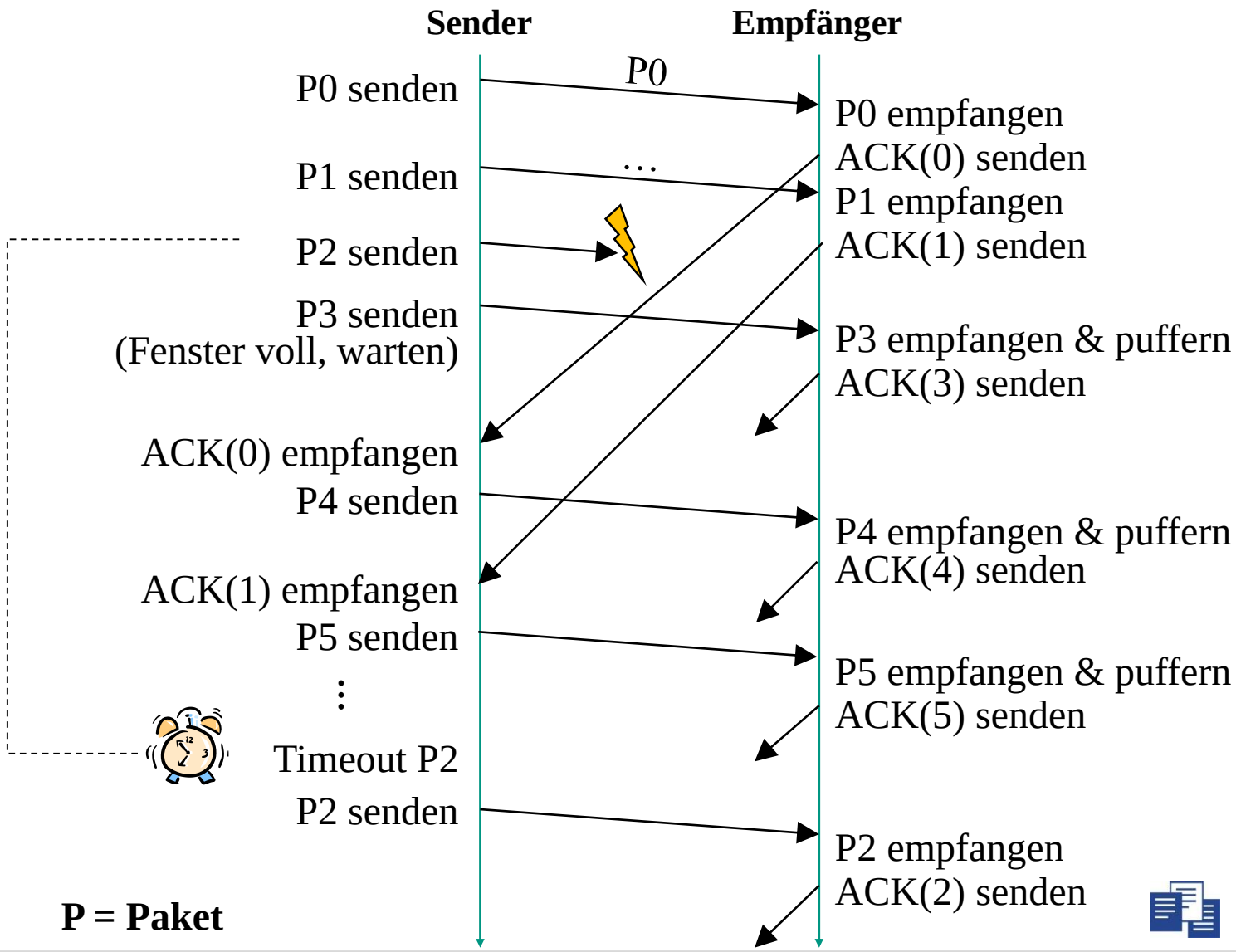
- Datenaustausch
  - Sender wie bei Go-Back-N
    - Sender kann mehrere Pakete senden, bis er eine Quittung erhalten muss
    - Maximale Anzahl der nicht quittierten Pakete ist begrenzt
  - Empfänger
    - Quittiert mit selektiven Quittungen

Selective Repeat ARQ

- Verhalten im Fehlerfall
  - Empfänger
    - Nachfolgende, korrekt empfangene Pakete werden vom Empfänger gepuffert und bestätigt
  - Sender
    - Nur nicht korrekt empfangene Pakete werden vom Sender wiederholt

- Fragen
  - Wo ist eine Pufferung der Pakete erforderlich? Wie viele müssen gepuffert werden?
  - Vor- und Nachteile von Go-Back-N und Selective Repeat im Vergleich?

Selective Repeat: Beispielablauf



[KuRo12]

Varianten in den ARQ-Verfahren

- Negative Quittungen (NACKs)
  - Nicht korrekt empfangene Pakete werden mit negativer Quittung (NACK) bestätigt
  - Go-Back-N
    - Sender wiederholt ab dieser Sequenznummer alle gesendeten Pakete
  - Selective Reject
    - Sender wiederholt genau die Pakete mit dieser Sequenznummer
- Kumulative Quittungen (Go-Back-N)
  - Quittung erfolgt für mehrere Pakete auf einmal
  - Kumulative Sequenznummer gibt an, bis wohin die Daten korrekt empfangen wurden, d.h. es handelt sich um positive Quittungen

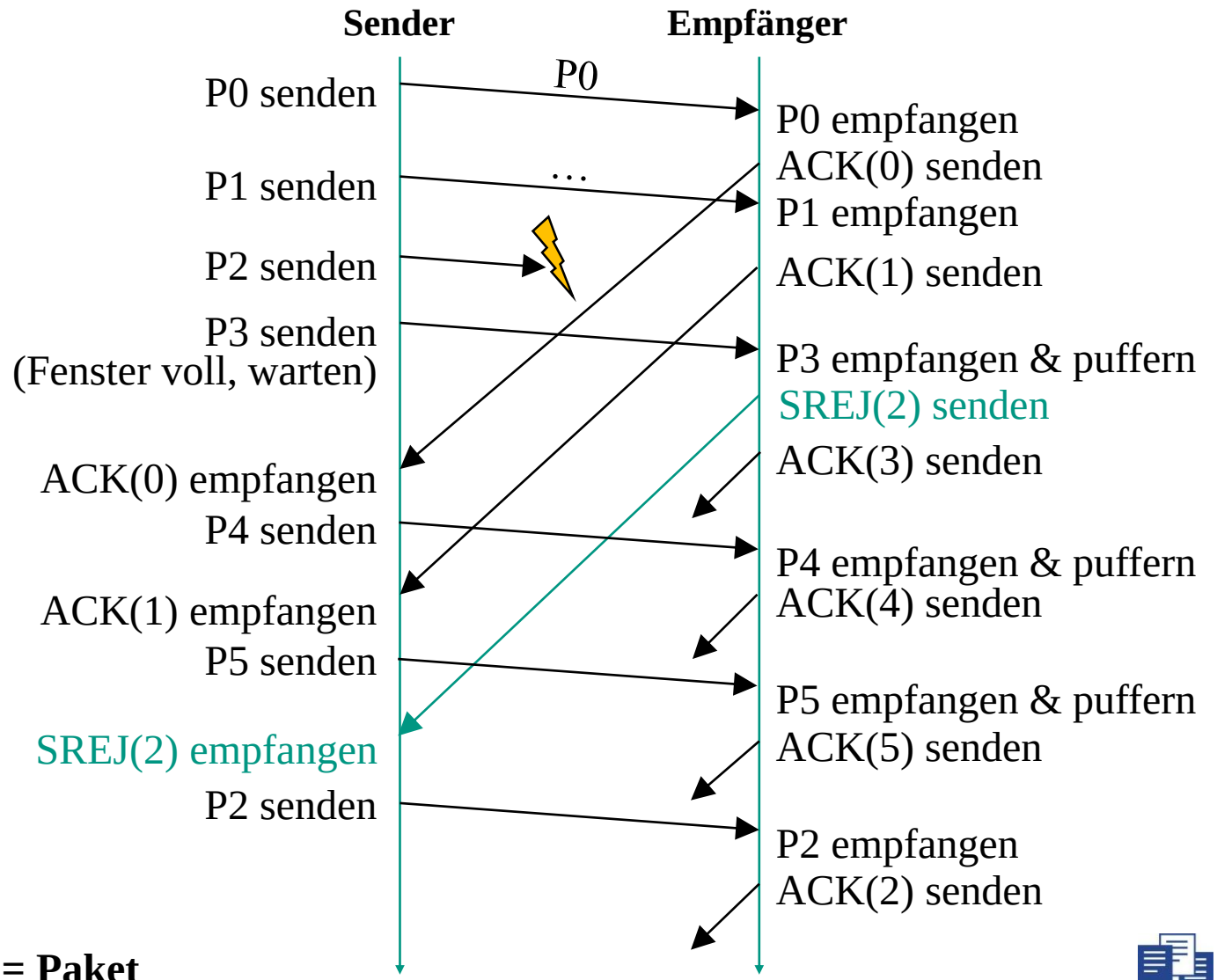
Selective Repeat vs. Selective Reject

- Bis auf das Quittierungsverhalten des Empfängers identisch

- Selective Repeat
  - Fehlerhaftes Paket wird nicht vom Empfänger bestätigt
  - Sender wiederholt dieses Paket nach Timeout

- Selective Reject
  - Empfänger teilt dem Sender mit einer negativen Quittung SREJ(Sequenz-Nr.) mit, dass Paket nicht korrekt empfangen wurde
  - Sender wiederholt sofort das Paket und wartet nicht auf Timeout

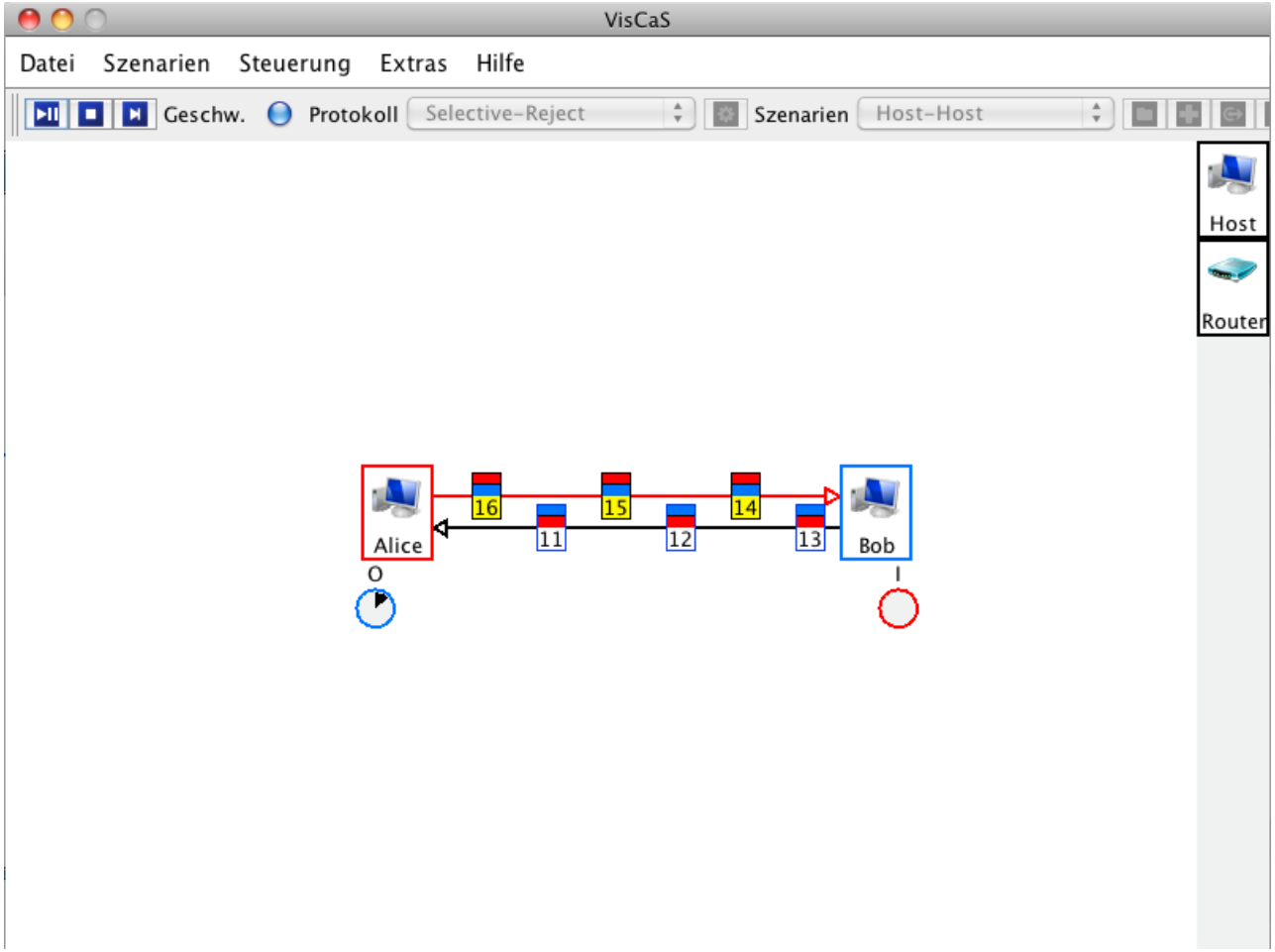
Selective Reject: Beispielablauf



P = Paket

[Stal10]

Visualisierung: Selective Reject und Flusskontrolle



http://www.tm.kit.edu/software/viscas/

Selective Reject: Leistungsbewertung

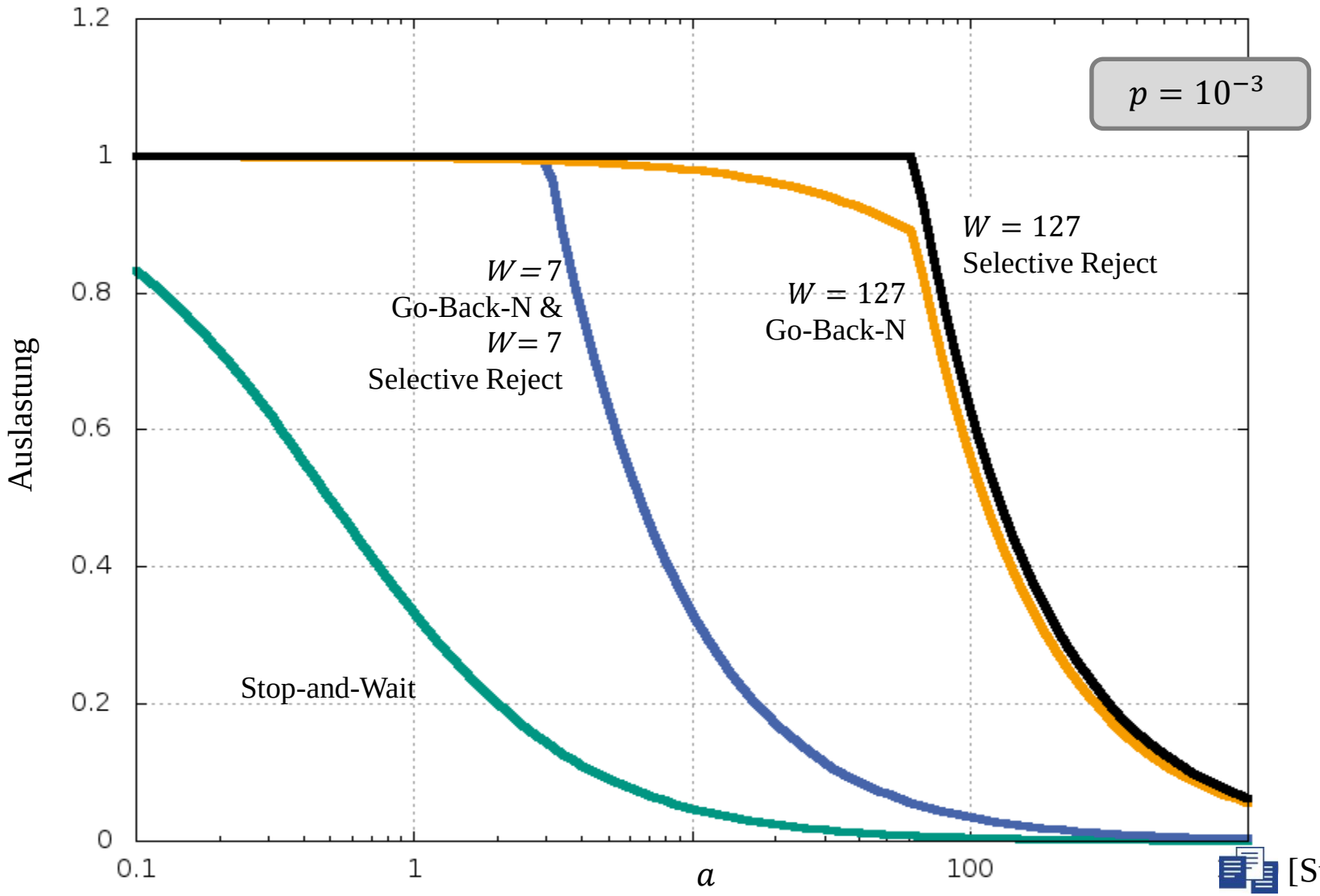
- Hier: mit Übertragungsfehlern
- Herleitung analog zu Stop-and-Wait
  - Es wird immer nur ein Paket wiederholt

$$n = \frac{1}{1-p}$$

  - Fallunterscheidung wegen Fenster

$$U = \begin{cases} 1-p & W \geq 1+2a \\ \dfrac{W(1-p)}{1+2a} & W < 1+2a \end{cases}$$

Vergleichende Bewertung

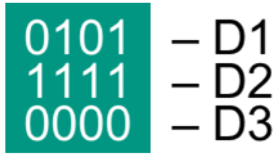


[Stal06]

4.5.5 Vorwärtsfehlerkorrektur

- Vorwärtsfehlerkorrektur (*Forward Error Correction* – FEC)
  - Fehlerkorrigierende Codes
  - Soll dazu dienen, verloren gegangenes Paket zu rekonstruieren

- Beispiel
  - Zu senden sind die Pakete



  - Dazu wird über XOR ein weiteres Paket berechnet

  - Diese vier Pakete werden an Empfänger gesendet

[Hals05]

Vorwärtsfehlerkorrektur – Ablauf

- Empfänger muss nur drei der vier Pakete korrekt empfangen, um fehlendes Paket rekonstruieren zu können
  - Er verknüpft die korrekt empfangenen Pakete mit XOR und rekonstruiert so die fehlende:
    - D1 geht verloren

      1111
      0000
      1010

      0101 – D1
    - D2 geht verloren

      0101
      0000
      1010

      1111 – D2
    - D3 geht verloren

      0101
      1111
      1010

      0000 – D3
  - Der Empfänger muss wissen, welches Paket verloren ging

Kapitel 3.5

FLUSSKONTROLLE

Flusskontrolle

- Problem
  - Empfänger kann von Sender überlastet werden
    - Daten können nicht empfangen werden, da Puffer nicht ausreichend
    - → Datenverluste
  - Sender muss Größe des Empfangspuffers berücksichtigen

- Anforderungen
  - Einfachheit
  - Möglichst geringe Nutzung von Netzressourcen
  - Fairness
  - Stabilität

Flusskontrolle

- Varianten
  - Closed Loop
    - Rückkopplung, um zu verhindern, dass Empfänger „überschwemmt“ wird
    - Sender adaptiert seinen Datenstrom entsprechend
  - Open Loop
    - Beschreibung des Verkehrs mit anschließender Ressourcenreservierung und Überwachung des eingehenden Verkehrs

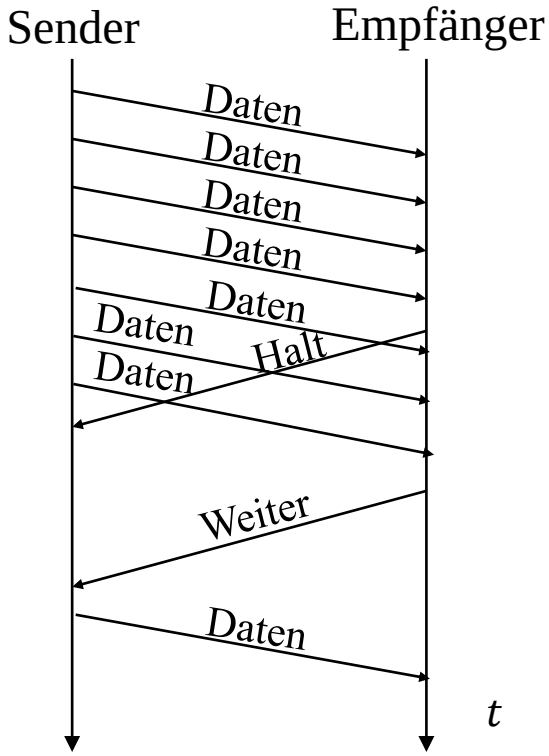
    … hier nicht weiter behandelt

Closed-Loop Flusskontrolle

- Varianten
  - Halt-und-Weiter
  - Stop-and-Wait
    - Kombiniert Fehler- und Flusskontrolle
  - Kreditbasierte Flusskontrolle
    - Statisches Fenster
    - Dynamisches Fenster

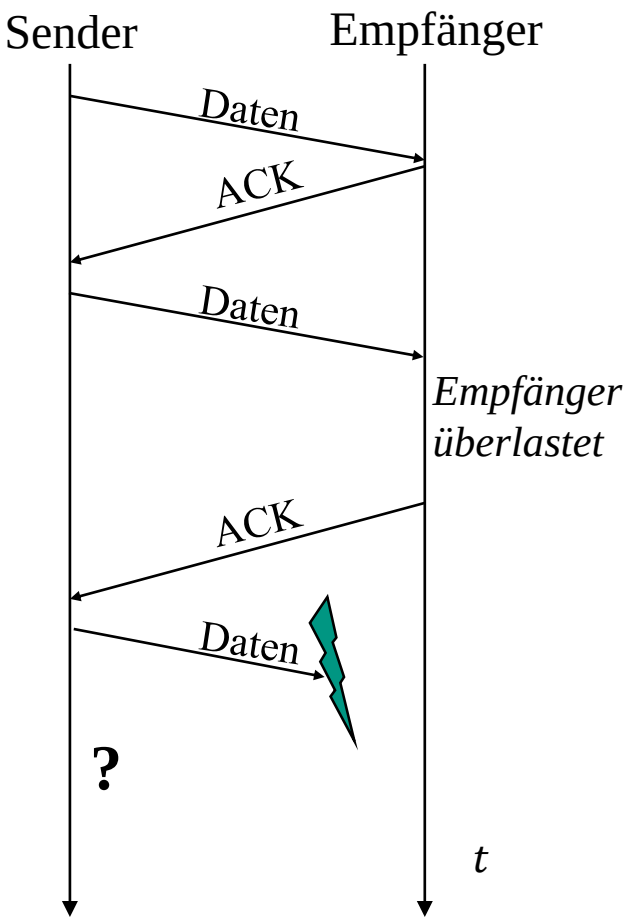
Halt-und-Weiter

- Sehr einfache Methode
  - Meldungen
    - Halt
    - Weiter
  - Kann Empfänger nicht mehr Schritt halten, schickt er eine Halt-Meldung
  - Ist Empfang wieder möglich, sendet Empfänger eine Weiter-Meldung
- Bewertung
  - Nur auf Vollduplex-Leitungen verwendbar
  - Bei hohen Verzögerungen nicht effektiv
  - Probleme bei Verlust der Halt-Meldung
- Beispiel
  - Fast-Ethernet, Gigabit-Ethernet



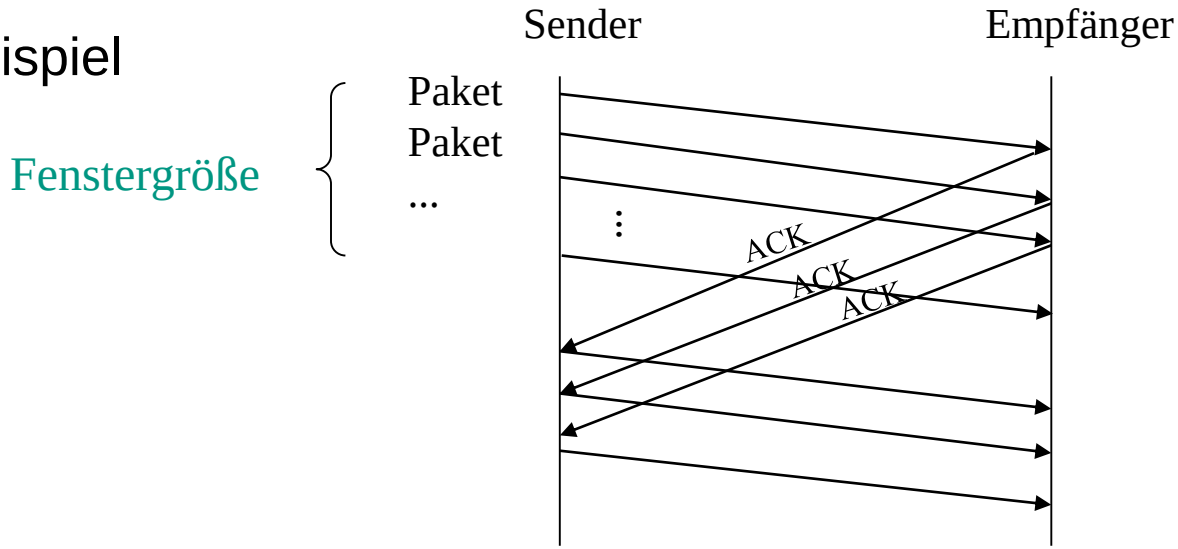
Stop-and-Wait

- Funktionsweise
  - Durch Zurückhalten der Quittung kann der Sender gebremst werden
  - Verfahren zur Fehlererkennung wird hier für Flusskontrolle mitbenutzt
- Problem
  - Der Sender kann nicht mehr unterscheiden,
    - ob Dateneinheit verloren ging, oder
    - ob Empfänger die Quittung wegen Überlast zurückgehalten hat



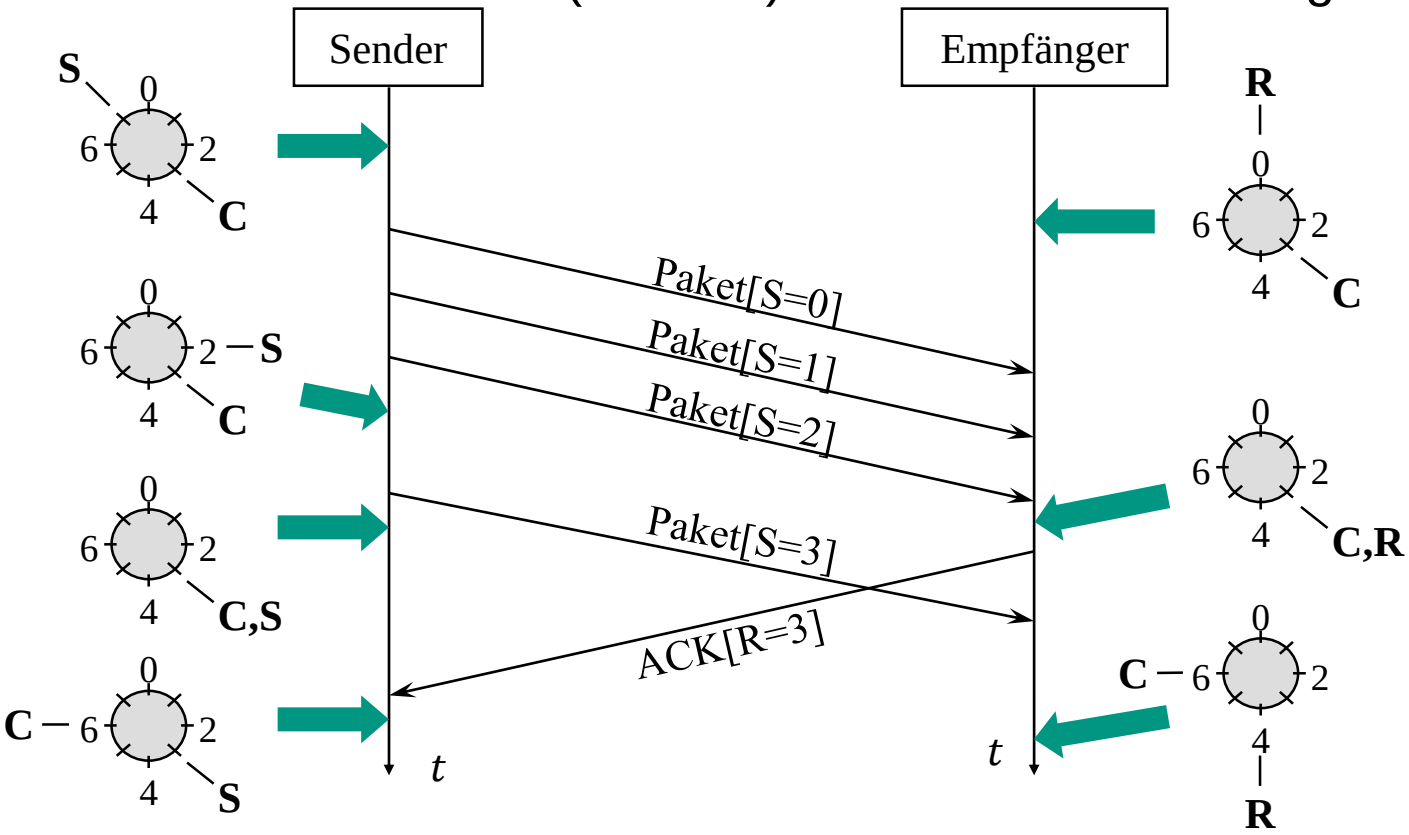
Kreditbasierte Flusskontrolle

- Prinzip
  - Sender kann bis zu einer maximalen Anzahl Pakete (bzw. Bytes) senden, ohne eine Quittung zu empfangen
  - Maximale Anzahl der Pakete repräsentiert die Pufferkapazität des Empfängers und wird als (Sende-)Kredit bezeichnet
  - Oftmals als fortlaufendes Fenster bezeichnet (engl.: Sliding Window)
    - Fenster wird mit jeder empfangenen reihenfolgetreuen positiven Quittung weitergeschaltet
    - Empfänger kann meist zusätzlich den Kredit explizit bestimmen (z.B. in TCP)
- Beispiel



Kreditbasierte Flusskontrolle: Sliding Window

Beispiel: Fenstermechanismus (Kredit 4) für eine Senderichtung

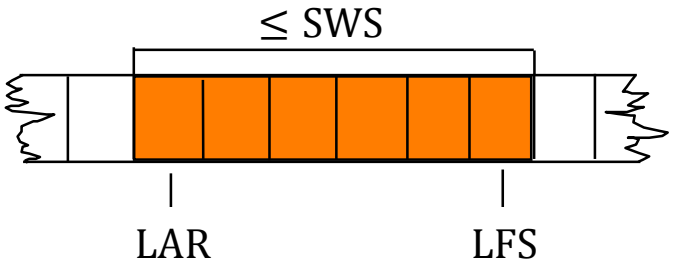


S: Sende-Sequenznummer (der zuletzt gesendeten Dateneinheit)
R: Nächste erwartete Sende-Sequenznummer = Quittierung bis Empfangs-Sequenznummer R – 1
C: Oberer Fensterrand (maximal erlaubte Sequenznummer)

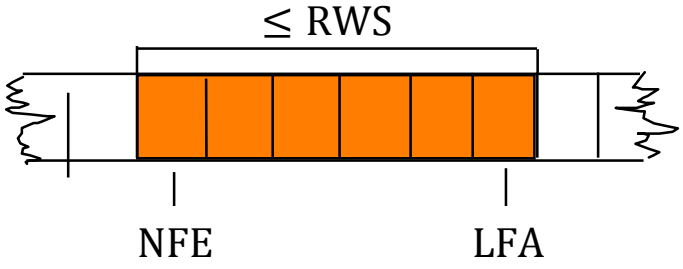
Nachteil: Kopplung von Fluss- und Fehlerkontrolle

Flusskontrolle mit Sliding Window

- Sender
  - SWS: Send Window Size (max. Anzahl ausstehender Pakete bzw. Bytes)
  - LAR: Last ACK Received (Sequenznummer des nächsten erwarteten Pakets bzw. Bytes)
  - LFS: Last Frame Sent (Sequenznummer des letzten gesendeten Pakets bzw. Bytes)
- Invariante
  - $\text{LFS} - \text{LAR} + 1 \leq \text{SWS}$



- Empfänger
  - RWS: Receiver Window Size (max. Anzahl nicht in Reihenfolge empfangener Pakete bzw. Bytes)
  - LFA: Last Frame Acceptable (Sequenznummer des letzten empfangbaren Pakets bzw. Bytes)
  - NFE: Next Frame Expected (Sequenznummer des nächsten in Reihenfolge erwarteten Pakets bzw. Bytes)
- Invariante
  - $\text{LFA} - \text{NFE} + 1 \leq \text{RWS}$

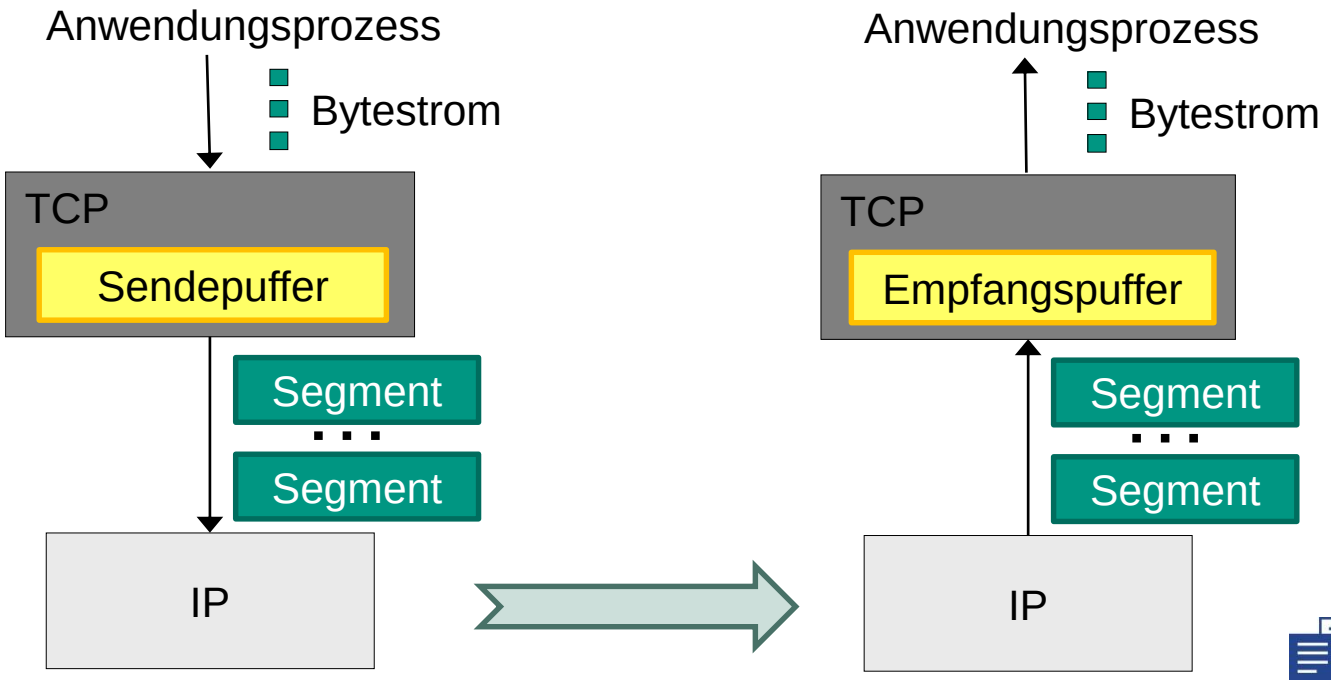


Kapitel 3.6

TCP

Transmission Control Protocol (TCP)

- TCP-Grundlagen
  - Stellt zuverlässigen Transportdienst zum Datentransfer zwischen Anwendungsprozessen zur Verfügung
    - RFC 793, September 1981
  - TCP erhält von der Anwendung einen Bytestrom und übergibt an IP TCP-Segmente



[RFC793]

Bytestrom → TCP-Segment?

- Problem
  - Wann wird aus Bytestrom ein TCP-Segment an IP weitergegeben?
  - RFC 793
    - *„TCP should „send that data in segments at its own convenience“*

- Alternativen
  - MSS: Maximum Segment Size
    - Länge der Anwendungsdaten, nicht Länge des TCP-Segments
    - Typische Größen (vermeiden das Fragmentieren durch IP)
      - 1460 Byte, 536 Byte, 512 Byte
  - Push (PSH-Flag im Kopf des TCP-Segments)
    - Sender verlangt sofortiges Versenden der Daten (Vorrangdaten)
    - Z.B. bei Telnet verwendet
  - Zeitgeber
    - Nach Zeitintervall der Inaktivität werden vorhandene Daten gesendet

Zu den Eigenschaften von TCP

- Verbindungsorientierter, zuverlässiger Dienst
  - Phasen: Verbindungsaufbau, Datentransfer, Verbindungsabbau
- Fehlerkontrolle
  - Sequenznummern, Prüfsumme, Quittierung, Sendewiederholungen im Fehlerfall
- Flusskontrolle
  - Ziel: Empfänger nicht überlasten
- Staukontrolle
  - Ziel: Netz nicht überlasten

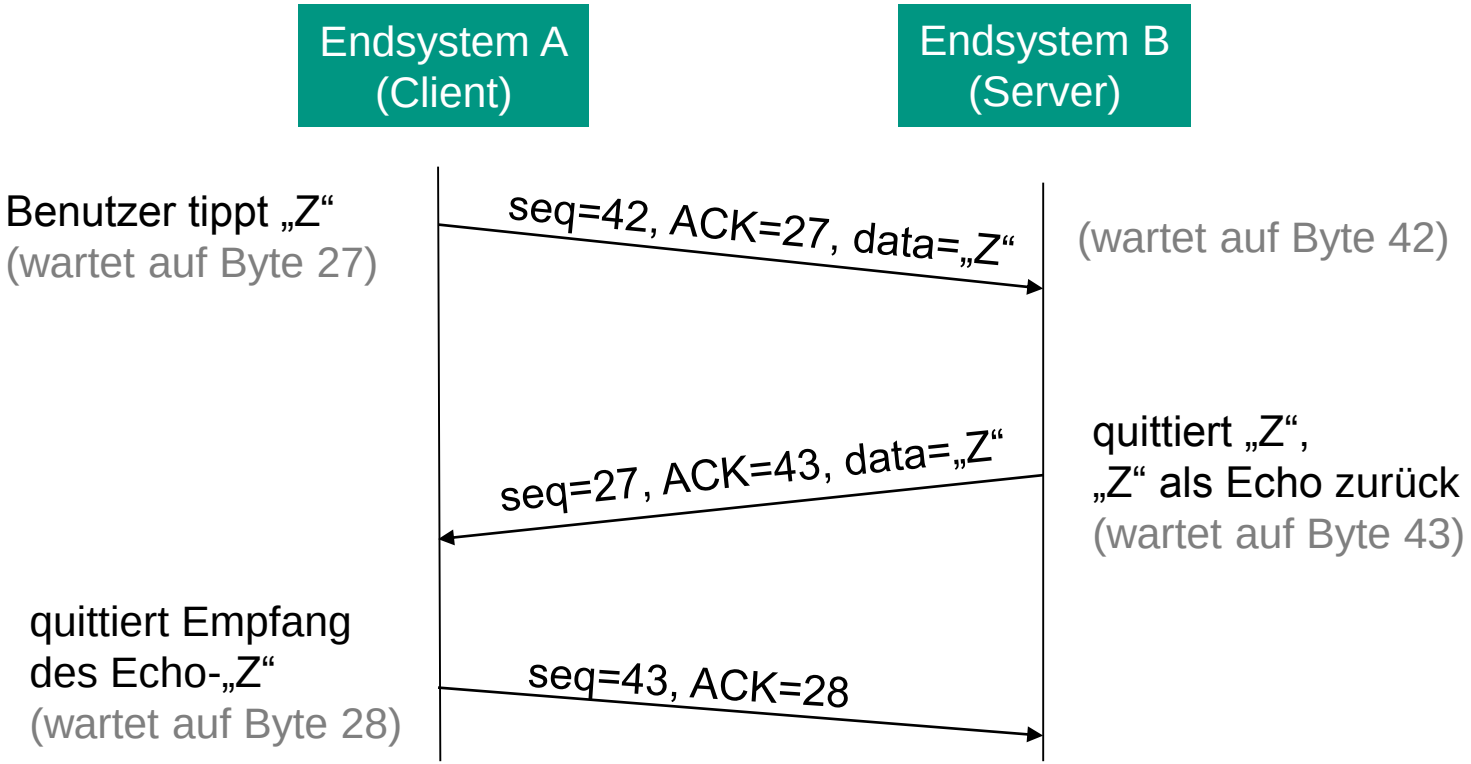
TCP-Sequenznummern und -Quittungen

- Sequenznummern
  - pro Byte, nicht pro Segment (erstes Byte in Segment)
  - Initiale Sequenznummer von Endsystemen *zufällig* gewählt

- Quittungen
  - positive kumulative Quittungen
  - Sequenznummer des nächsten Bytes, das vom Kommunikationspartner erwartet wird

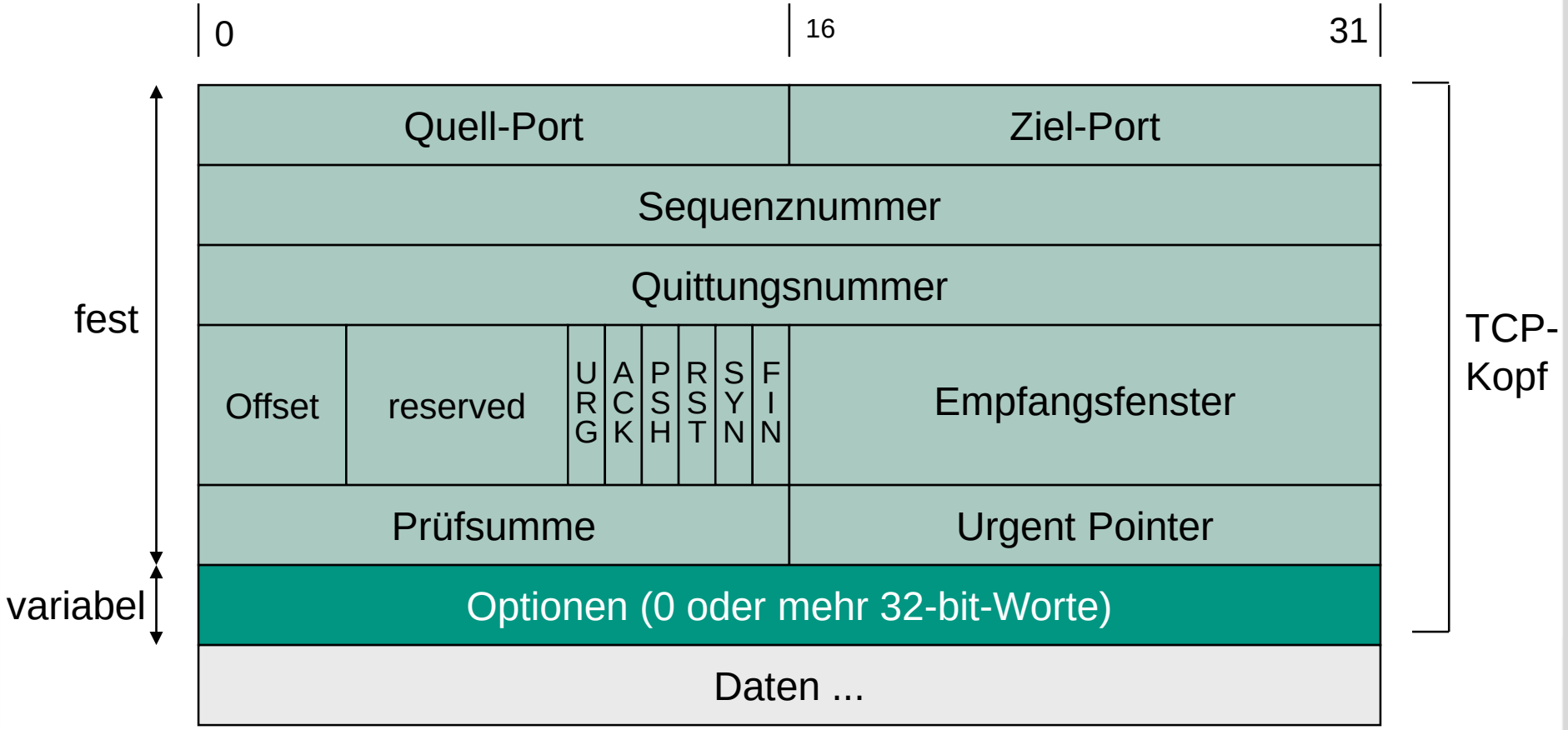
Beispiel

- Telnet-Szenario
  - Client tippt Zeichen ein, Server sendet Kopie zurück (Echo)



ACK in Segment mit Daten:
„piggybacked“ ACK

Format eines TCP-Segments



Felder eines TCP-Segments

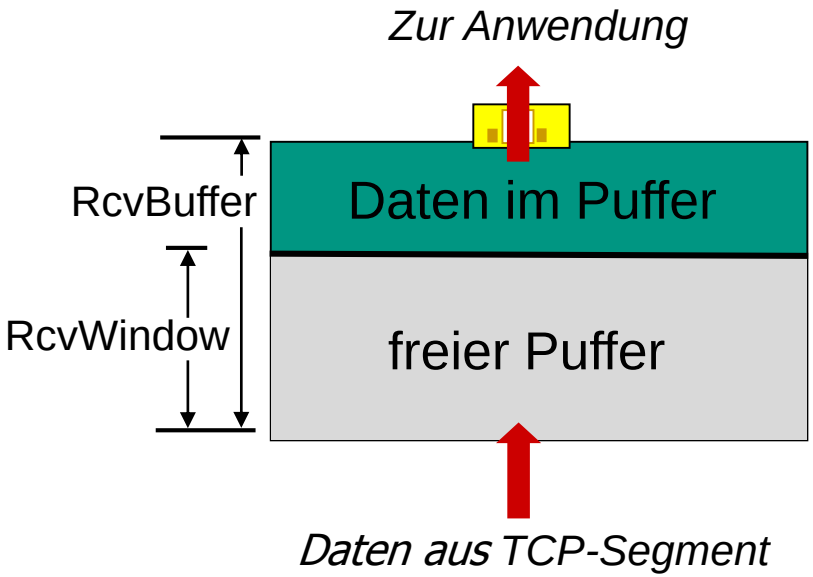
- Quell-Port und Ziel-Port
  - Identifizieren Endpunkte der Verbindung
- Sequenznummer
  - Gemessen in Byte (nicht pro Segment)
- Quittung
  - Die nächste vom Empfänger erwartete Sequenznummer
- Offset
  - Anzahl der 32-Bit-Wörter im TCP-Kopf
- URG
  - Wird auf 1 gesetzt, falls der Urgent Pointer verwendet wird
    - *… in der Regel nicht benutzt*
- SYN
  - Wird beim Verbindungsaufbau verwendet, um *Connection Request* (TConReq) bzw. *Connection Confirmation* (TConCnf) anzuzeigen
- ACK
  - Unterscheidet bei gesetztem SYN-Bit eine TConReq-PDU von einer TConCnf-PDU
  - Signalisiert die Gültigkeit des Quittungs-Feldes

Felder eines TCP-Segments

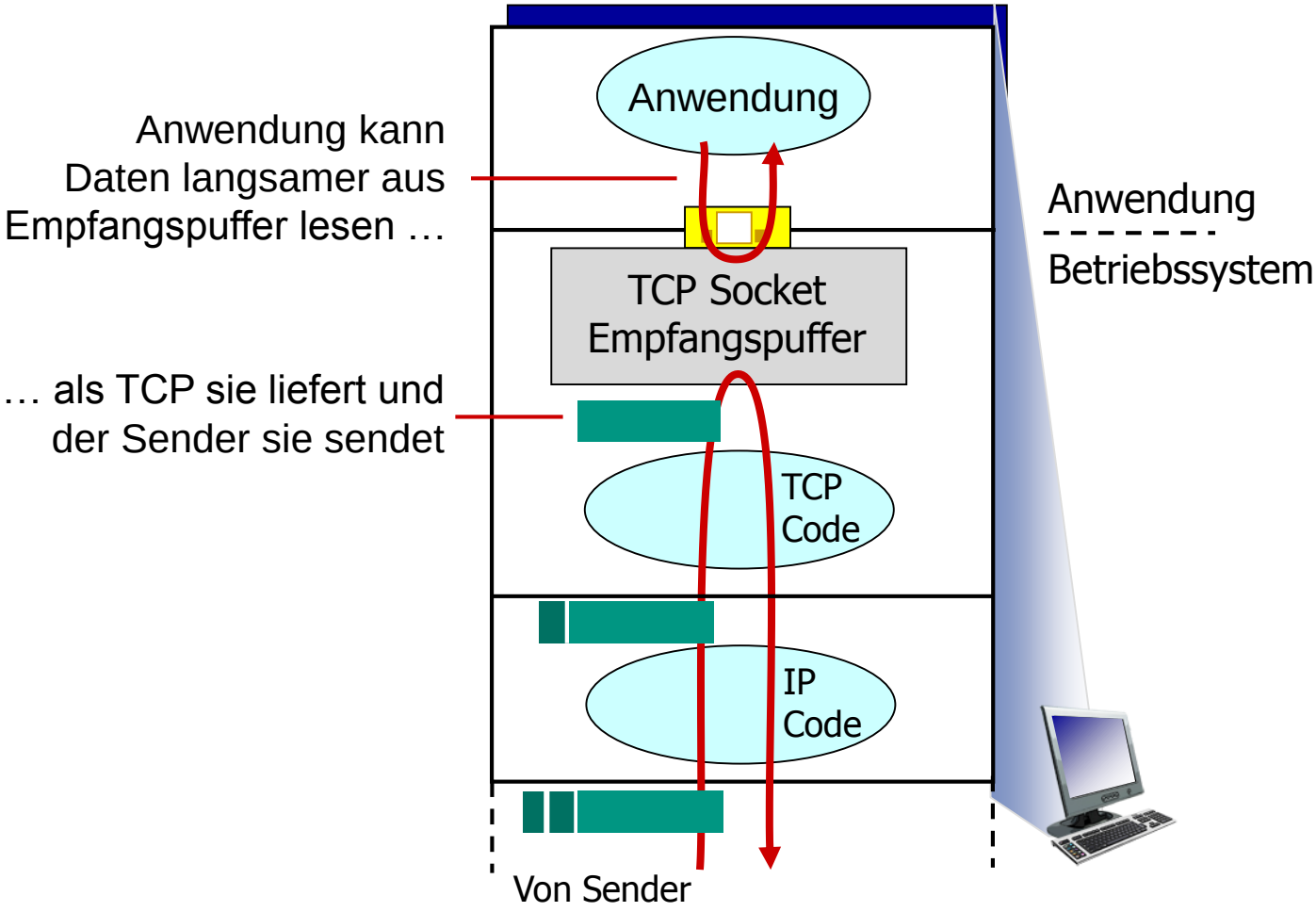
- FIN
  - Gibt an, dass der Sender keine Daten mehr senden möchte
- RST
  - Wird benutzt, um eine Verbindung zurückzusetzen
- PSH
  - Signalisiert, dass übergebene Daten sofort weitergeleitet werden sollen
  - Gilt sowohl für den Sender als auch für den Empfänger
  - Wird in der Regel nicht benutzt
- Empfangsfenster
  - Dient zur Flusskontrolle
- Prüfsumme
  - Enthält Prüfsumme über TCP-Kopf, Daten und Pseudoheader (wie bei UDP)
- Urgent-Zeiger
  - Relativer Zeiger auf wichtige Daten
- Das Optionen-Feld
  - kann Optionen variabler Länge aufnehmen $(n * 32\ \text{Bit})$

Flusskontrolle

- Ziel
  - Überlastung beim Empfänger vermeiden
- Vorgehensweise
  - Empfänger reserviert Pufferplatz pro Verbindung (explizite Kreditvergabe)
    - RcvBuffer: gesamter Pufferplatz für zu empfangende Daten
      - Default: 4096 Byte
    - RcvWindow: derzeit freier Pufferplatz (Empfangsfenster)
      - Feld Empfangsfenster im Kopf des TCP-Segments
      - Empfänger informiert Sender
  - Sender sendet nicht mehr unbestätigte Daten als in RcvWindow passen



Protokollstack im Empfänger



Flusskontrolle

- Szenario
  - Endsystem A schickt große Datei über TCP an Endsystem B
- Variablen beim *Empfänger*
  - LastByteRead
    - Letzte Sequenznummer, die von der Anwendung aus dem Empfangspuffer gelesen wurde
  - LastByteRcvd
    - Letzte Sequenznummer, die über das Netz empfangen und in den Empfangspuffer geschrieben wurde
  - Es muss immer gelten

    LastByteRcvd – LastByteRead <= RcvBuffer
  - Empfangsfenster

    RcvWindow = RcvBuffer – (LastByteRcvd – LastByteRead)

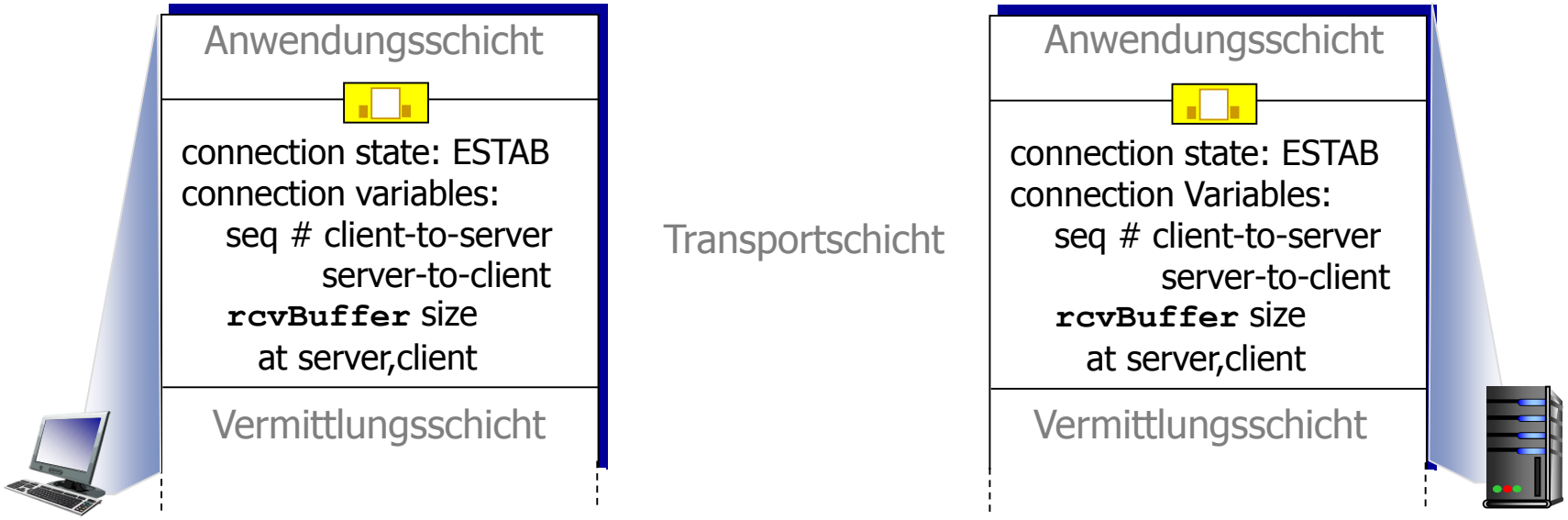
Flusskontrolle

- Variablen beim *Sender*
  - LastByteSent
    - letzte Sequenznummer, die gesendet wurde
  - LastByteAcked
    - letzte Sequenznummer, die quittiert wurde
  - Es muss immer gelten

LastByteSent – LastByteAcked <= RcvWindow

Verbindungsverwaltung

- Handshake zwischen Sender und Empfänger vor dem Datenaustausch
  - Zustimmung, eine Verbindung aufzubauen
  - Vereinbarung von Verbindungsparametern




```
Socket clientSocket =
  newSocket("hostname","port
   number");
```

```
Socket connectionSocket =
   welcomeSocket.accept();
```

Verbindungslos vs. Verbindungsorientiert

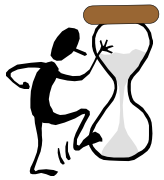
- Verbindungslose Kommunikation
  - Daten werden ohne vorherigen Handshake gesendet
  - Vorteil
    - schnelle Datenversendung möglich
  - Nachteil
    - keine Möglichkeit der Kontrolle, ob Kommunikationspartner überhaupt zuhört bzw. zuhören kann

Verbindungslos vs. Verbindungsorientiert

- Verbindungsorientierte Kommunikation
  - Dem Datenaustausch geht *Aufbau* einer Verbindung voraus
  - Am Ende erfolgt der *Abbau* einer Verbindung

  - Vorteil
    - Aushandlung von Kommunikationsparametern möglich
      - Z.B. benötigte Fenstergrößen, verwendete Fehlerkontrollmechanismen, Sequenznummern ...
  - Nachteile
    - Eigentlicher Datenaustausch verzögert
    - Overhead der Verbindungsetablierung und -verwaltung kann höher sein als der eigentliche Datenaustausch

    → häufig beim Aufruf von Webseiten der Fall

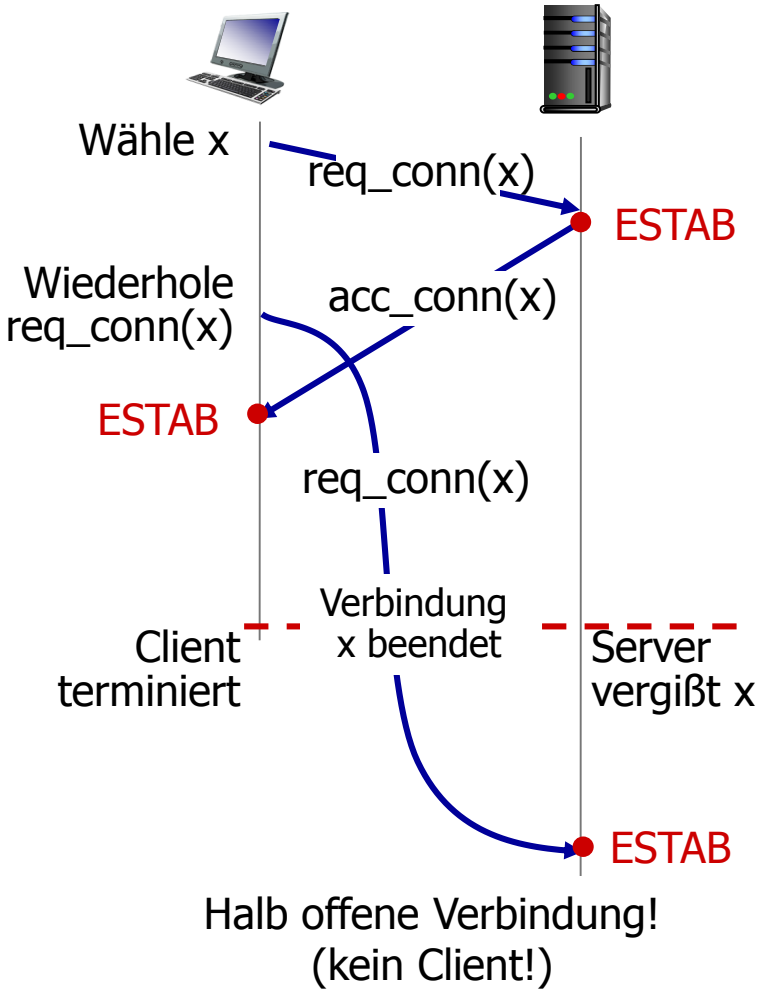


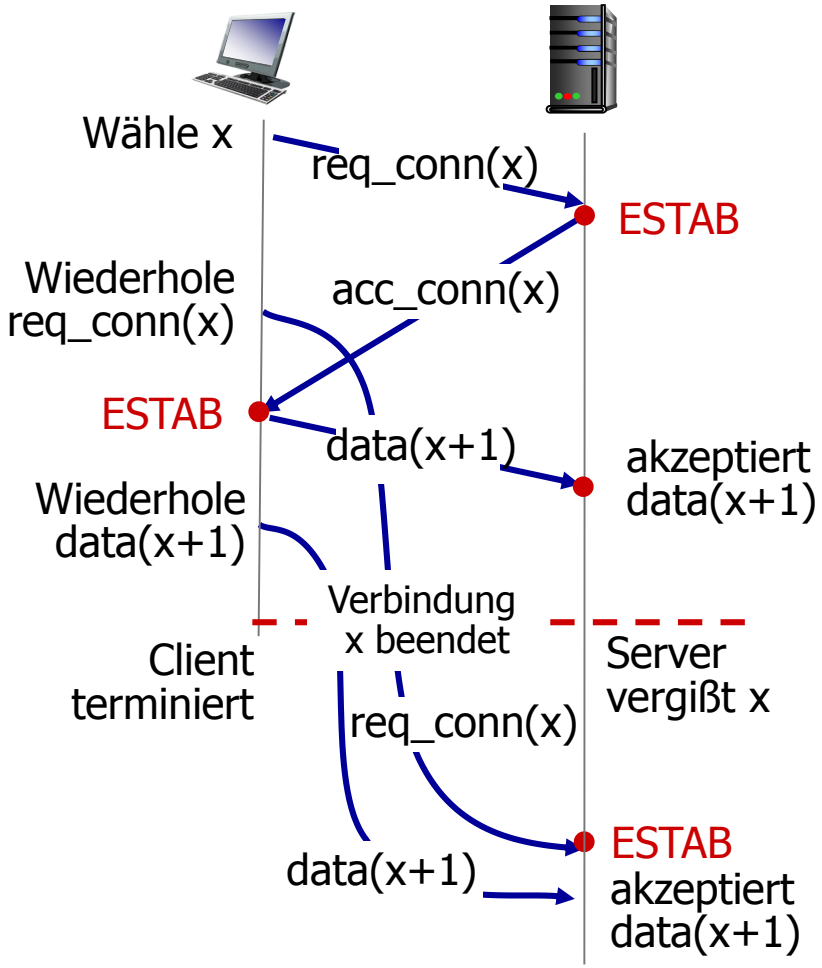
Zustimmung zum Verbindungsaufbau

- 2-Wege-Handshake

- Funktioniert 2-Wege-Handshake immer?
  - Unterschiedliche Verzögerungen
  - Wiederholte Nachrichten (z.B. req_conn(x))
  - Vertauschte Reihenfolge von Paketen
  - Kann die andere Seite nicht „sehen“

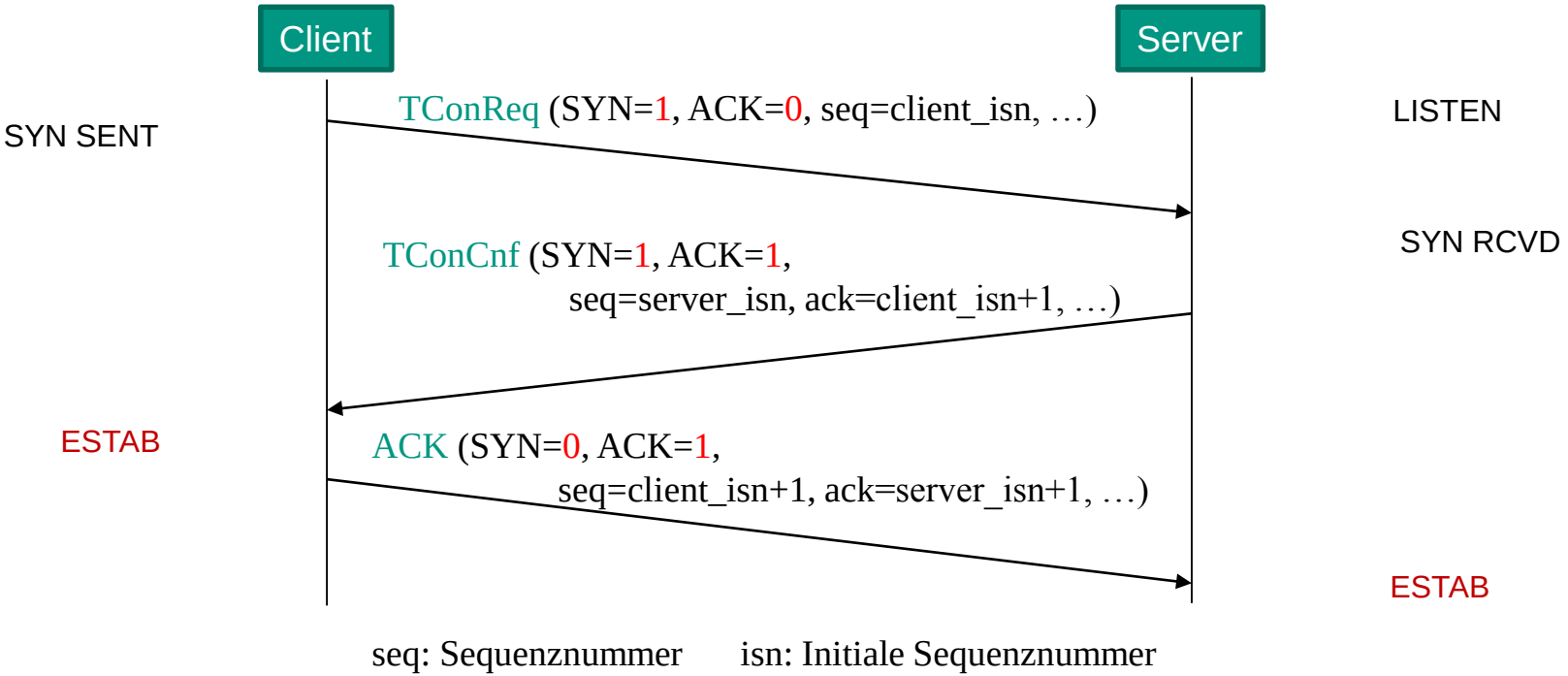
2-Wege-Handshake: Fehlerszenarien





TCP-Verbindungsverwaltung

- Verbindungsaufbau
  - Client initiiert Verbindung und Server wartet auf Verbindungswünsche



seq: Sequenznummer isn: Initiale Sequenznummer

  - 3-Wege-Handshake
  - Connection request/confirm führen keine Nutzdaten mit sich
  - Festlegung der initialen Sequenznummern (..._isn). Weshalb erforderlich?
  - Bekanntgabe der Größe des Flusskontrollfensters, Allokation Puffer …
  - Ab wann dürfen Client und Server Daten senden?

tcpdump

- Paketsniffer
  - Liest an einem Netzwerk-Interface, z.B. an einer Netzwerkkarte, alle empfangenen Pakete aus
  - Beispiel

```
$ tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
11:37:41.729131 IP HostA.42980 > Server.www: S 62713440:62713440(0) win 5840
<mss 1460,sackOK,timestamp 8835394 0,nop,wscale 2>
11:37:41.732888 IP Server.www > HostA.42980: S 1853613617:1853613617(0) ack 62713441 win 5792
<mss 1460,sackOK,timestamp 1199883883 8835394,nop,wscale 0>
11:37:41.732907 IP HostA.42980 > Server.www: . ack 1 win 1460 <nop,nop,timestamp 8835395 1199883883>
11:37:41.729297 IP HostA.42980 > Server.www: P 1:112(111) ack 1 win 1460
<nop,nop,timestamp 8835395 1199883883>
11:37:41.729724 IP Server.www > HostA.42980: . ack 112 win 5792 <nop,nop,timestamp 1199883883 8835395>
```

  - Offenbar TCP-Verbindungsaufbau zu Port 80 eines Empfängers
    - 3-Wege Handshake (SYN, SYN/ACK, ACK)
  - Aller Wahrscheinlichkeit nach ein HTTP-Request eines Clients an einen WWW-Server
    - Tcpdump zählt die Sequenznummer nach erfolgreichem Verbindungsaufbau von 1 beginnend (relativ)

tcpdump

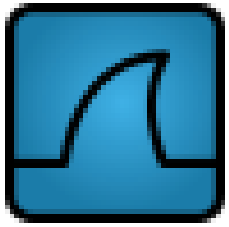
- Fortsetzung

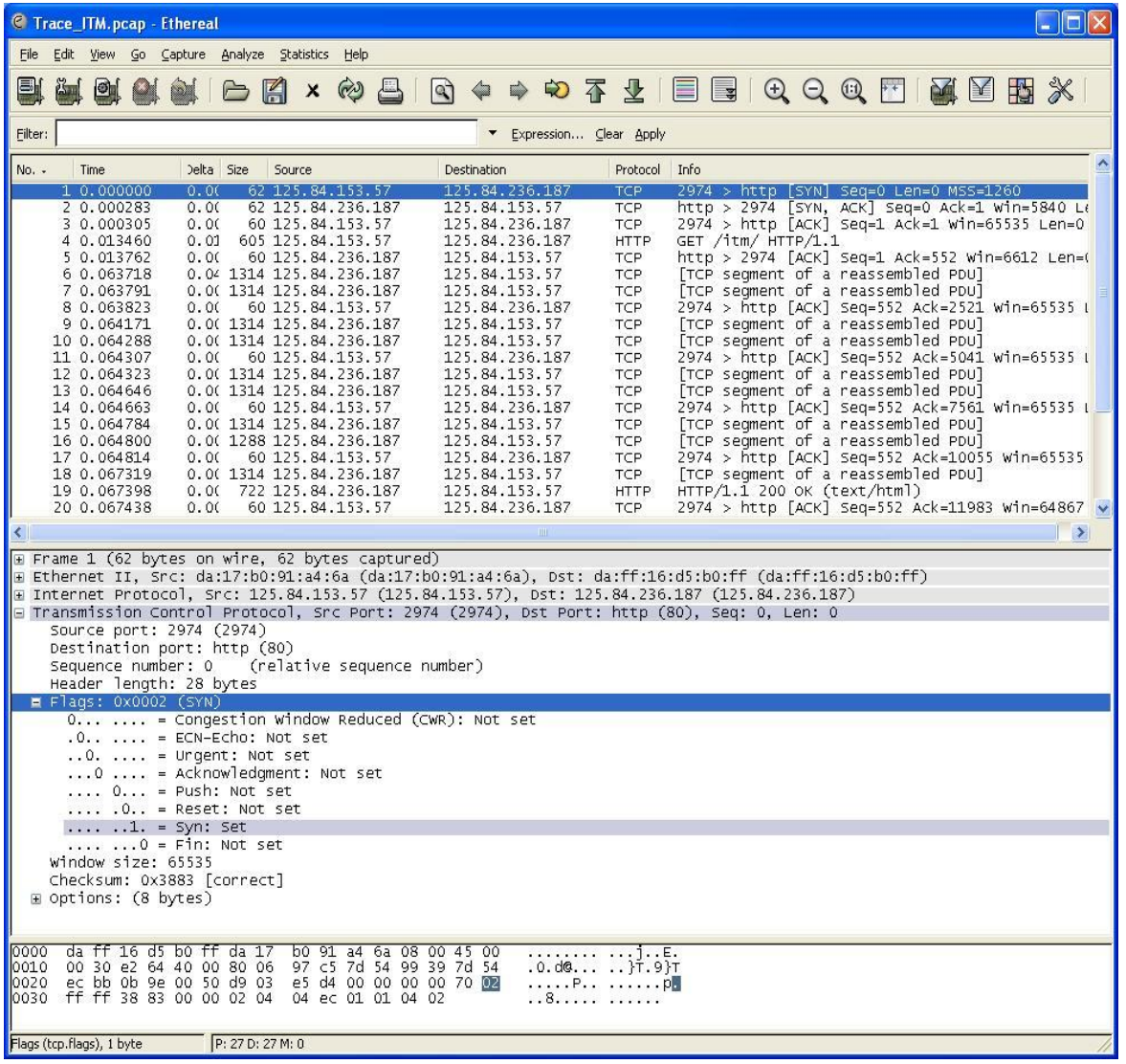
```
11:37:41.730116 IP HostA.42980 > Server.www: . ack 365 win 1728 <nop,nop,timestamp 8835396 1199883884>
11:37:41.730103 IP Server.www > HostA.42980: P 365:1131(766) ack 112 win 5792
<nop,nop,timestamp 1199883884 8835395>
11:37:41.730128 IP HostA.42980 > Server.www: . ack 1131 win 2111 <nop,nop,timestamp 8835396 1199883884>
11:37:41.730577 IP HostA.42980 > Server.www: F 112:112(0) ack 1131 win 2111
<nop,nop,timestamp 8835396 1199883884>
11:37:41.731096 IP Server.www > HostA.42980: F 1131:1131(0) ack 113 win 5792
<nop,nop,timestamp 1199883885 8835396>
11:37:41.731157 IP HostA.42980 > Server.www: . ack 1132 win 2111 <nop,nop,timestamp 8835397 1199883885>
```

- TCP-Verbindungsabbau
  - FIN, FIN/ACK, ACK

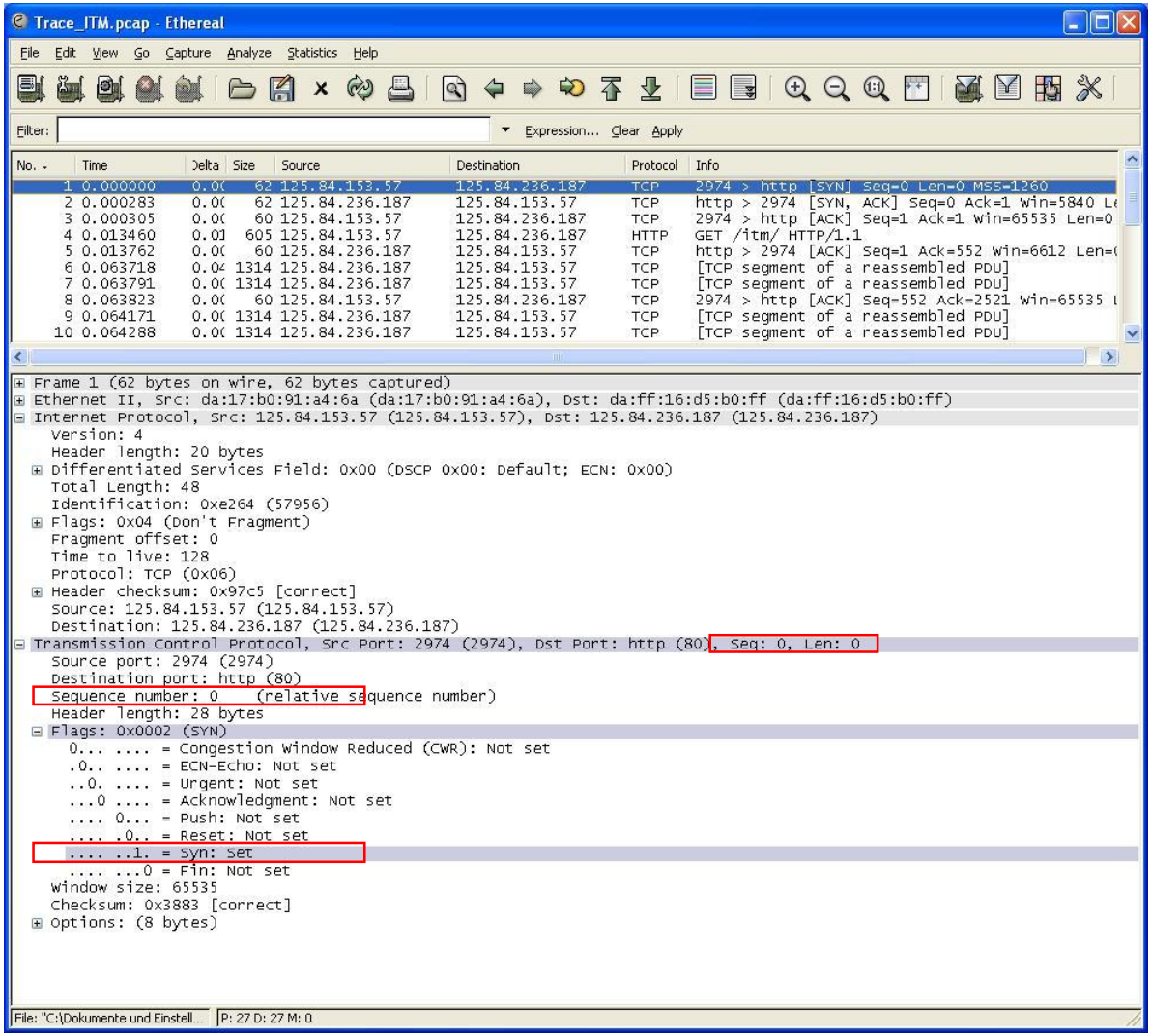
Wireshark

- Grafischer Paketsniffer
  - Dekodierung zahlreicher Protokolle
  - Diagramm-erstellung möglich

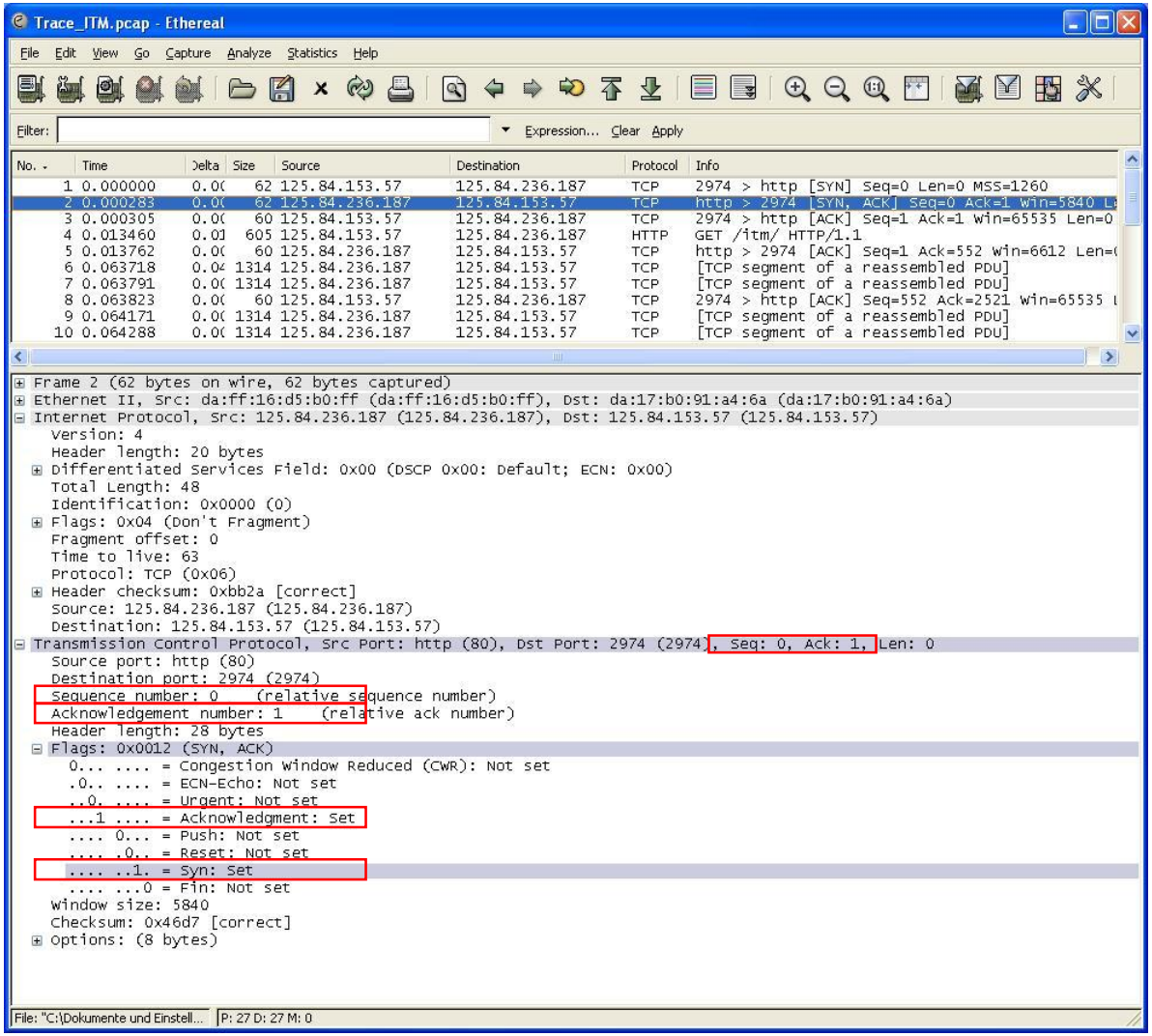




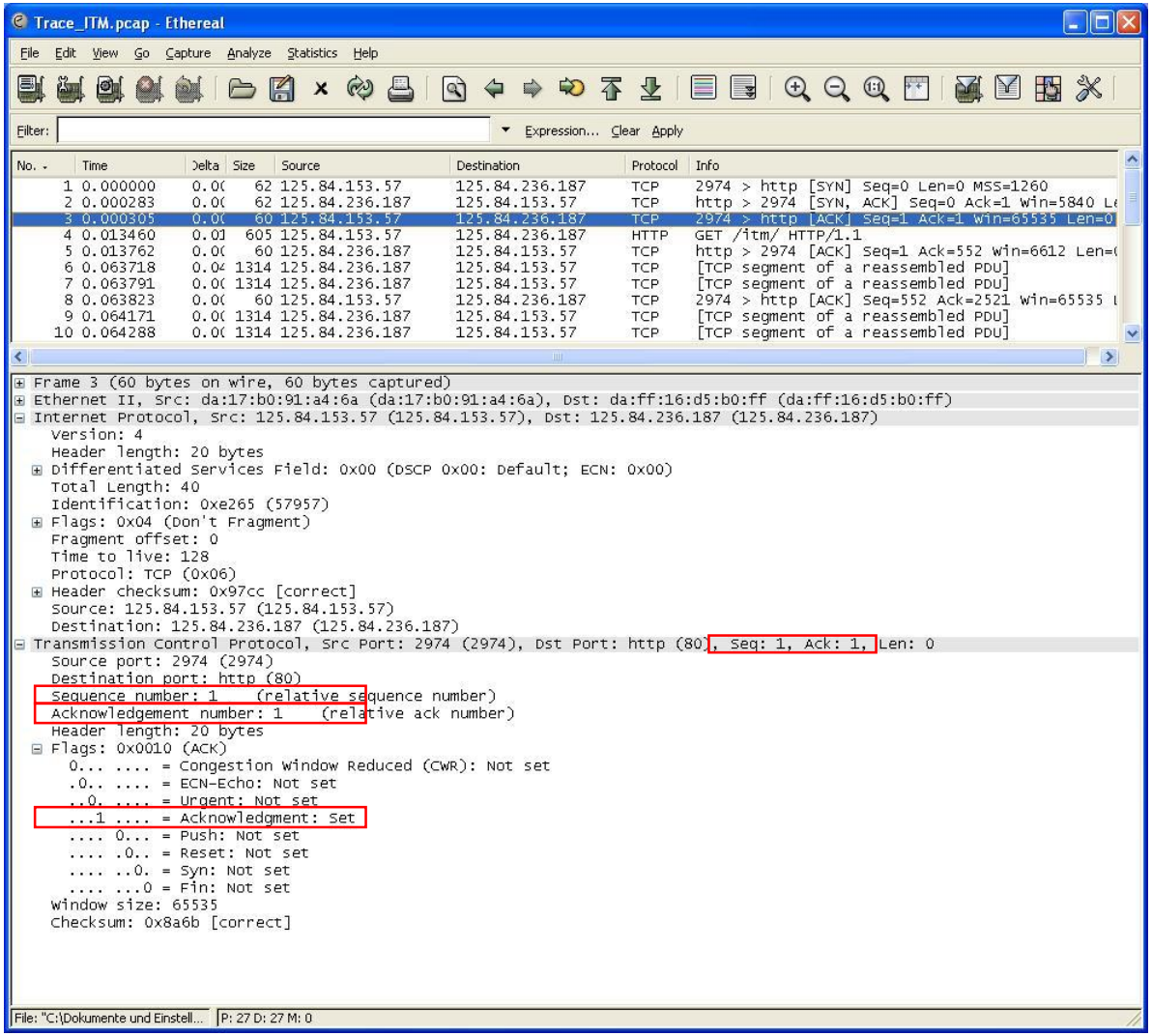
Beispiel: TCP-Verbindungsaufbau



Beispiel: TCP-Verbindungsaufbau

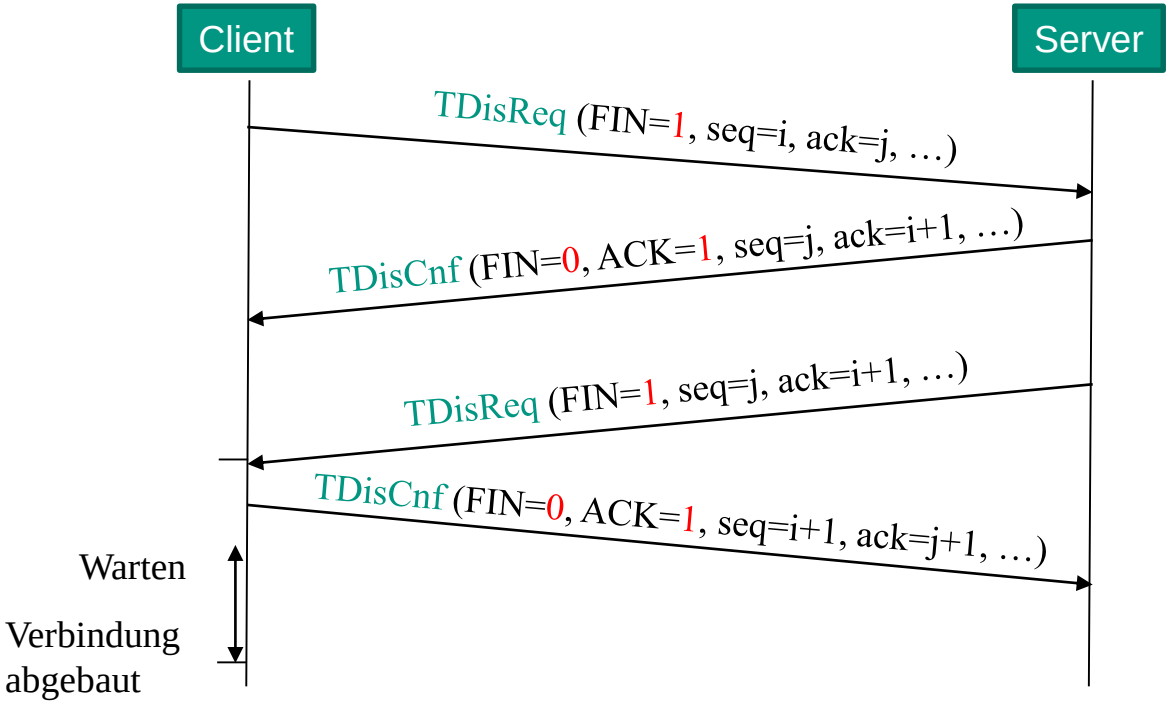


Beispiel: TCP-Verbindungsaufbau (ACK)



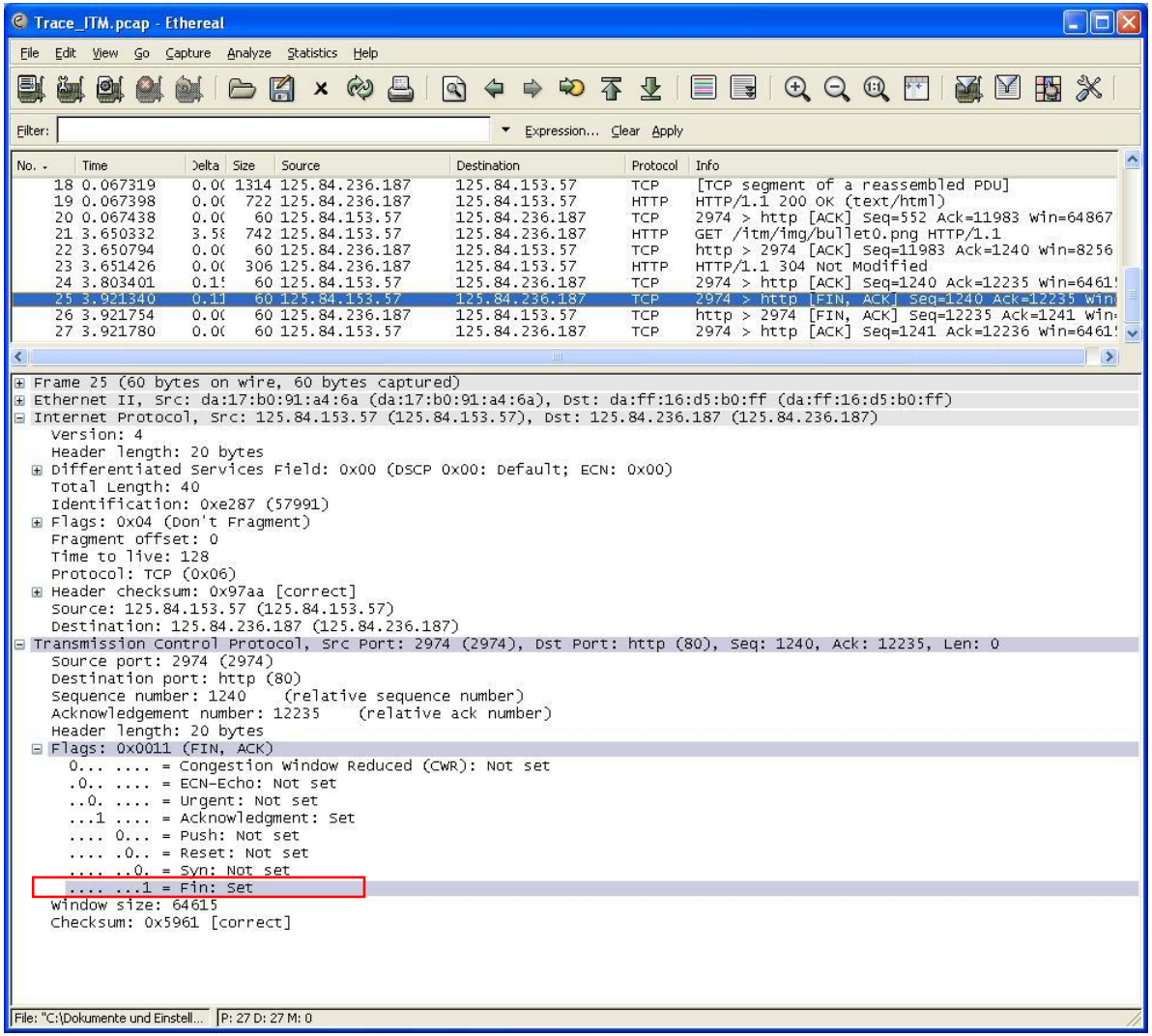
TCP-Verbindungsverwaltung

- Verbindungsabbau
  - Kann sowohl vom Client als auch vom Server initiiert werden

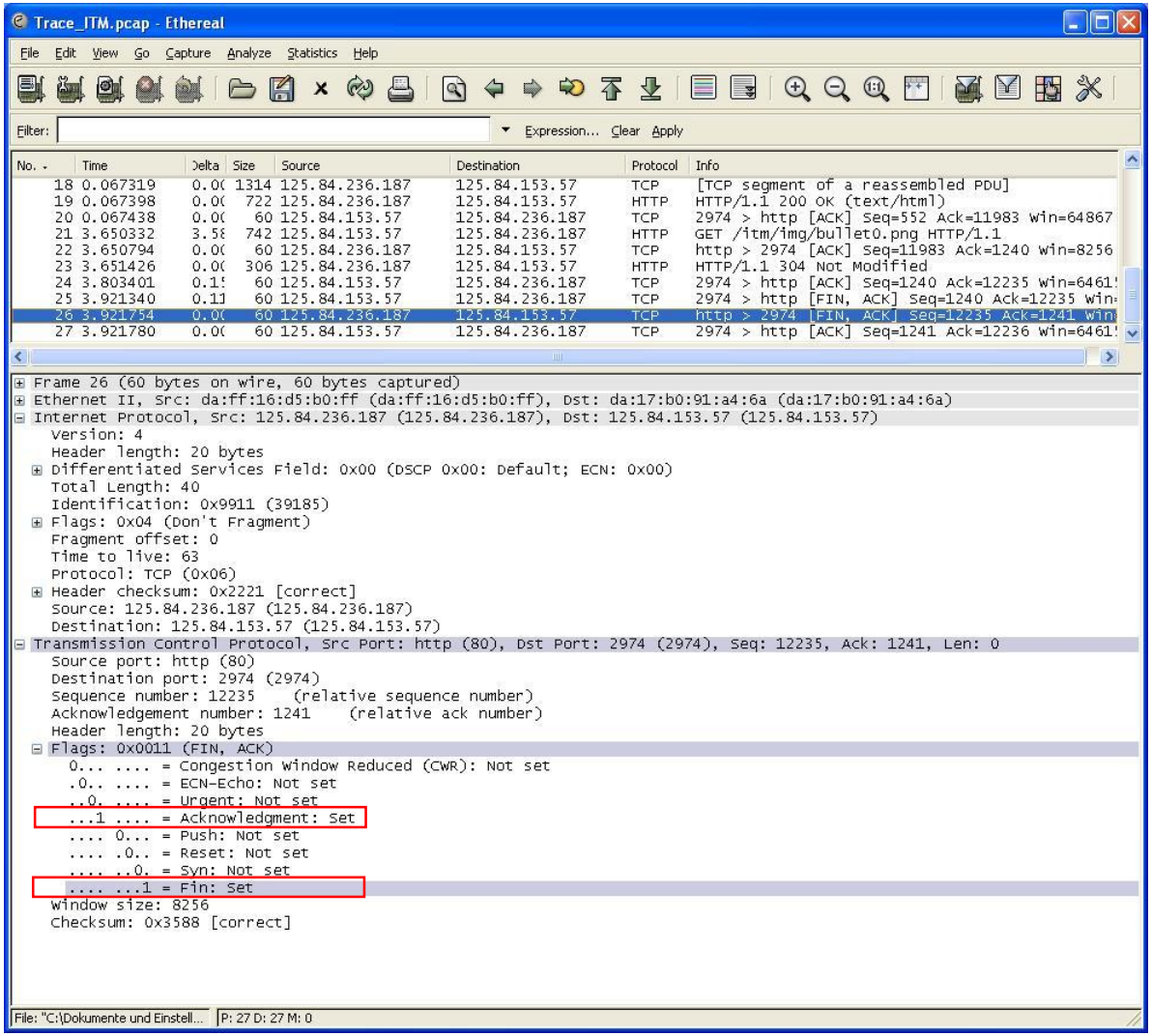


  - Nach letztem ACK wird noch gewartet, bevor lokaler Kontext gelöscht wird
    - Zur Vermeidung der Zustandshaltung: senden von RST-Segment
    - Problem damit?
  - Weitere Varianten im Ablauf möglich

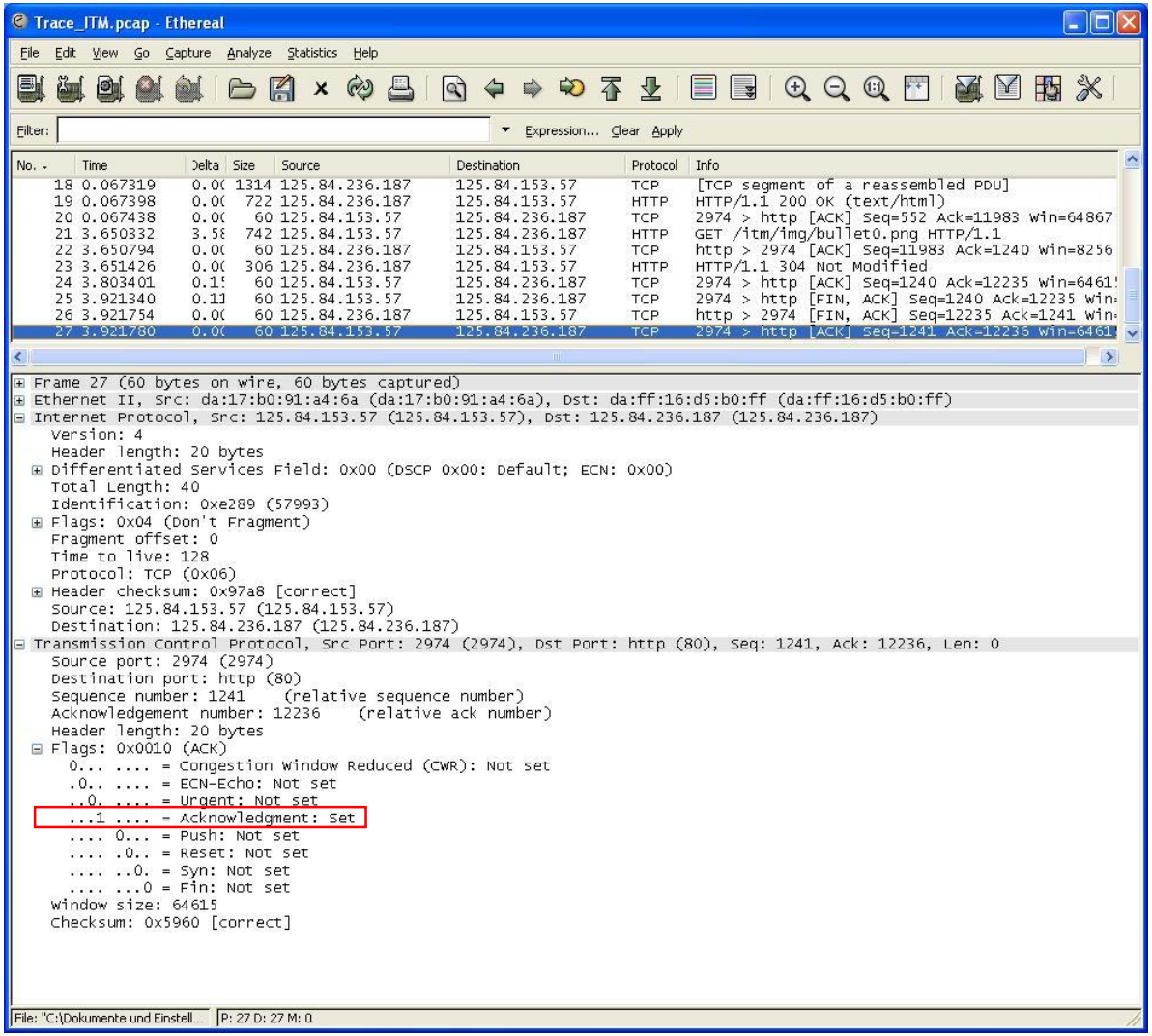
Beispiel: TCP-Verbindungsabbau (Client FIN)



Beispiel: TCP-Verbindungsabbau (Server ACK + FIN)



Beispiel: TCP-Verbindungsabbau (Client ACK)



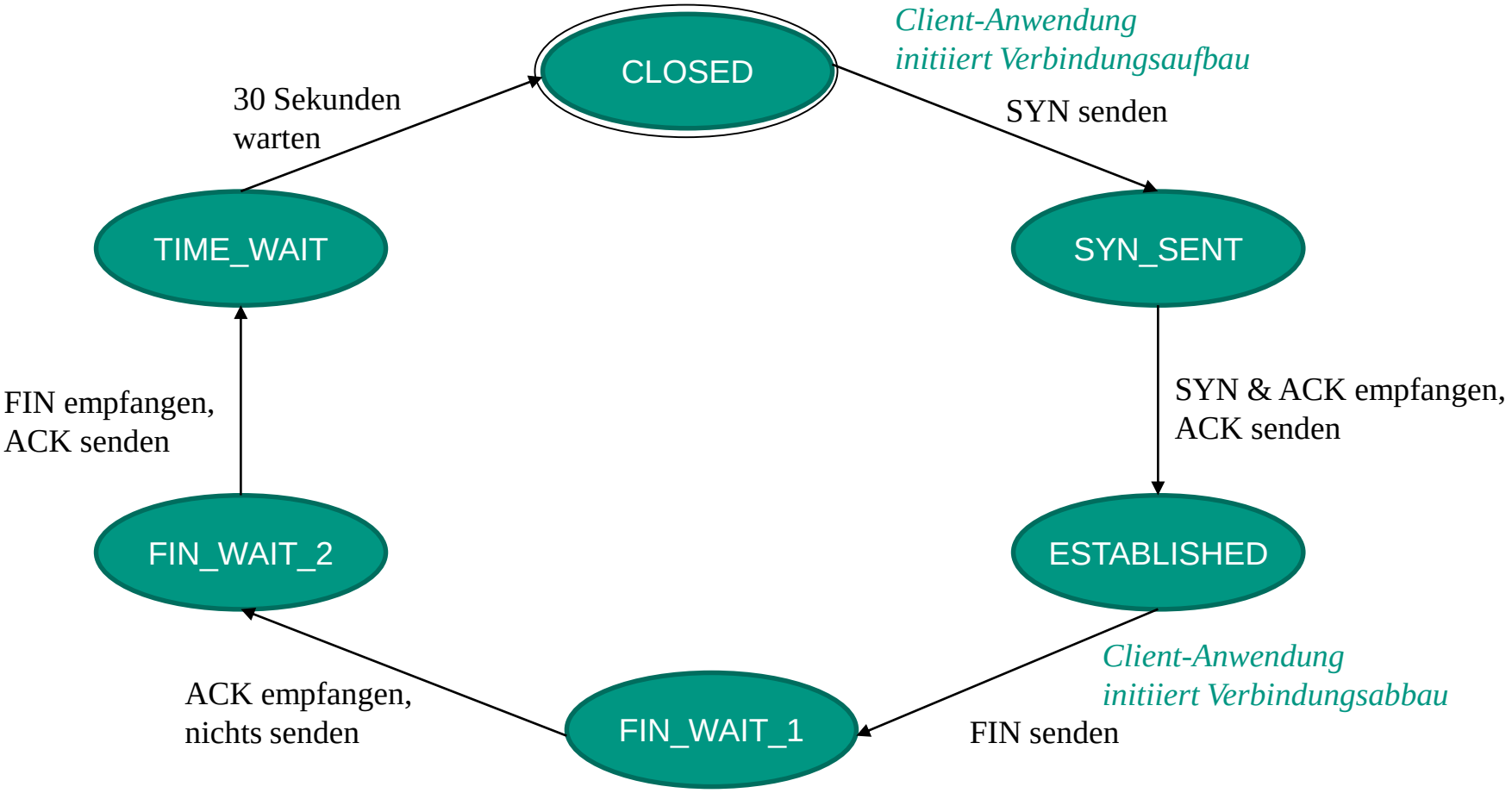
Verbindungsabbau

- Ordnungsgemäßer Abbau (vgl. vorangegangene Folie)
  - 4-Wege-Handshake
  - „Richtungen“ werden unabhängig voneinander geschlossen
    - FIN-, ACK- und FIN/ACK-Segmente
  - Lediglich eine Richtung geschlossen (simplex Datentransfer möglich)
    - Unidirektionaler Datenfluss
      - Neue Nutzdaten in „offener Richtung“ möglich
    - In „Gegenrichtung“
      - Kontrolldaten (z.B. ACK), keine neuen Nutzdaten

- Abbruch einer Verbindung
  - Reset (RST-Segment)
  - Verbindung wird unmittelbar geschlossen
    - Z.B. keine weiteren Sendewiederholungen
    - Kontext wird unmittelbar gelöscht

Zustandsautomat TCP-Client

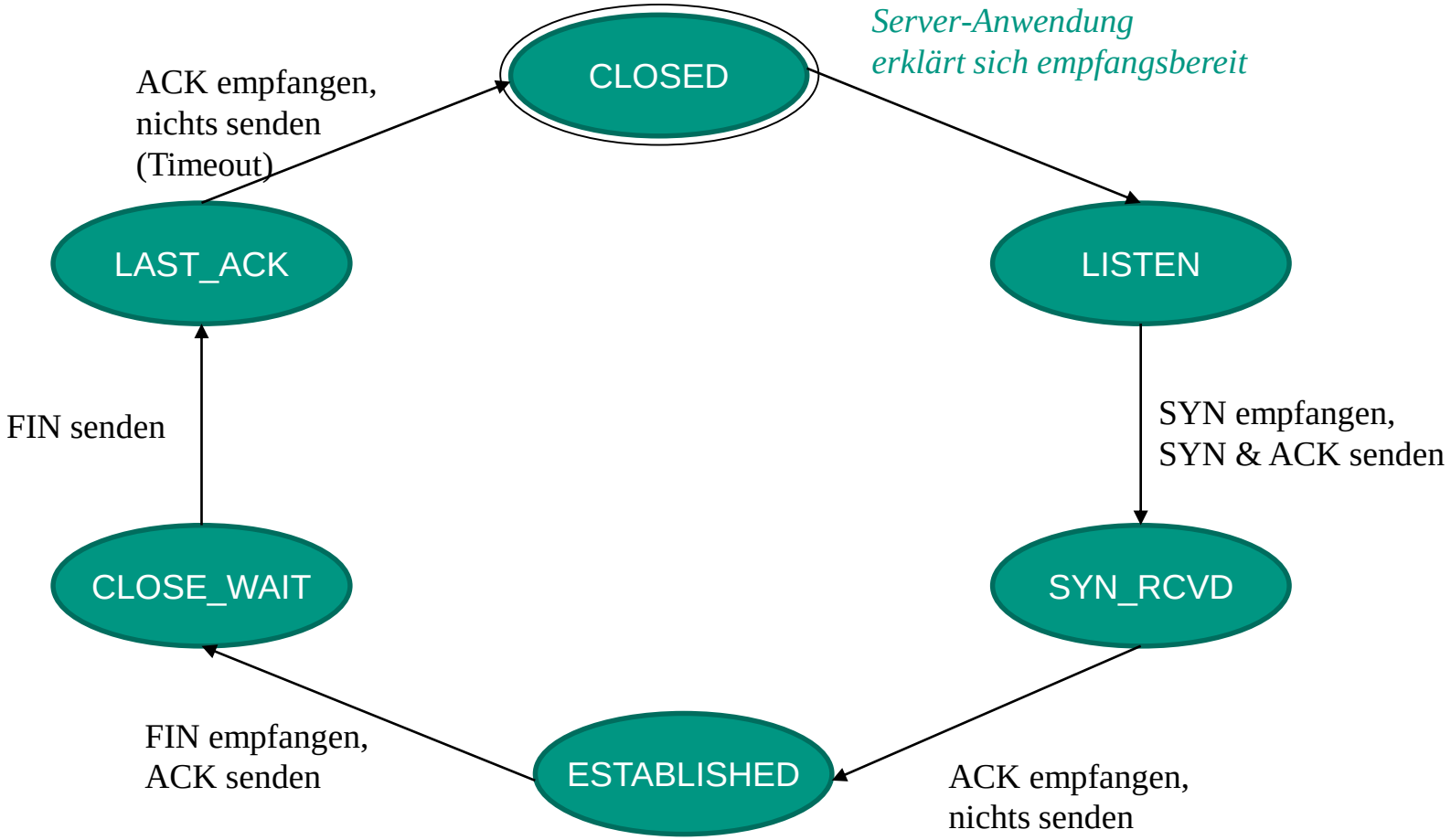
- Typische Sequenz von Zuständen



Dies ist eine vereinfachte Darstellung und nicht(!) der komplette TCP-Zustandsautomat

Zustandsautomat TCP-Server

- Typische Sequenz von Zuständen



Dies ist eine vereinfachte Darstellung und nicht(!) der komplette TCP-Zustandsautomat

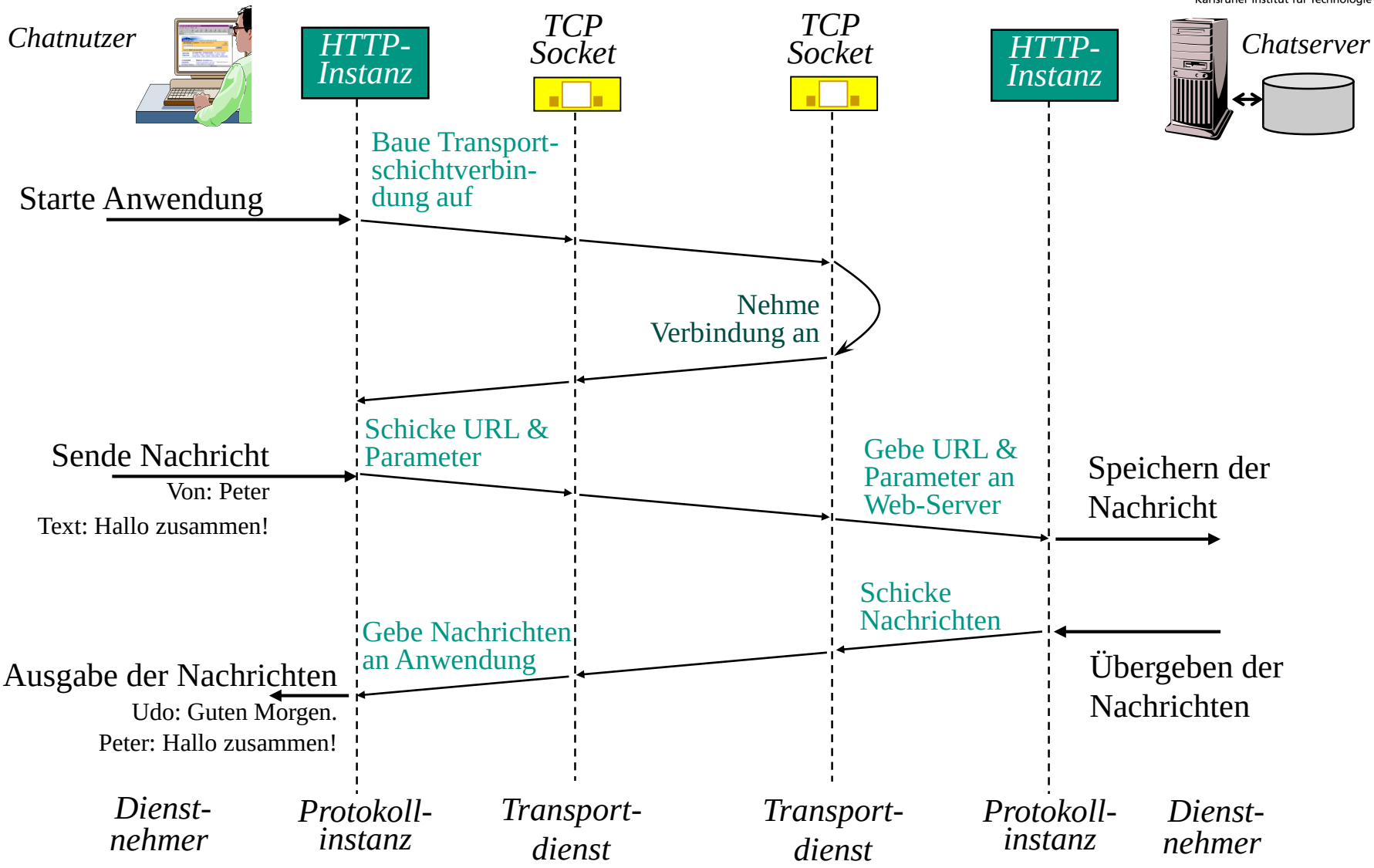
Pingo

http://pingo.upb.de/

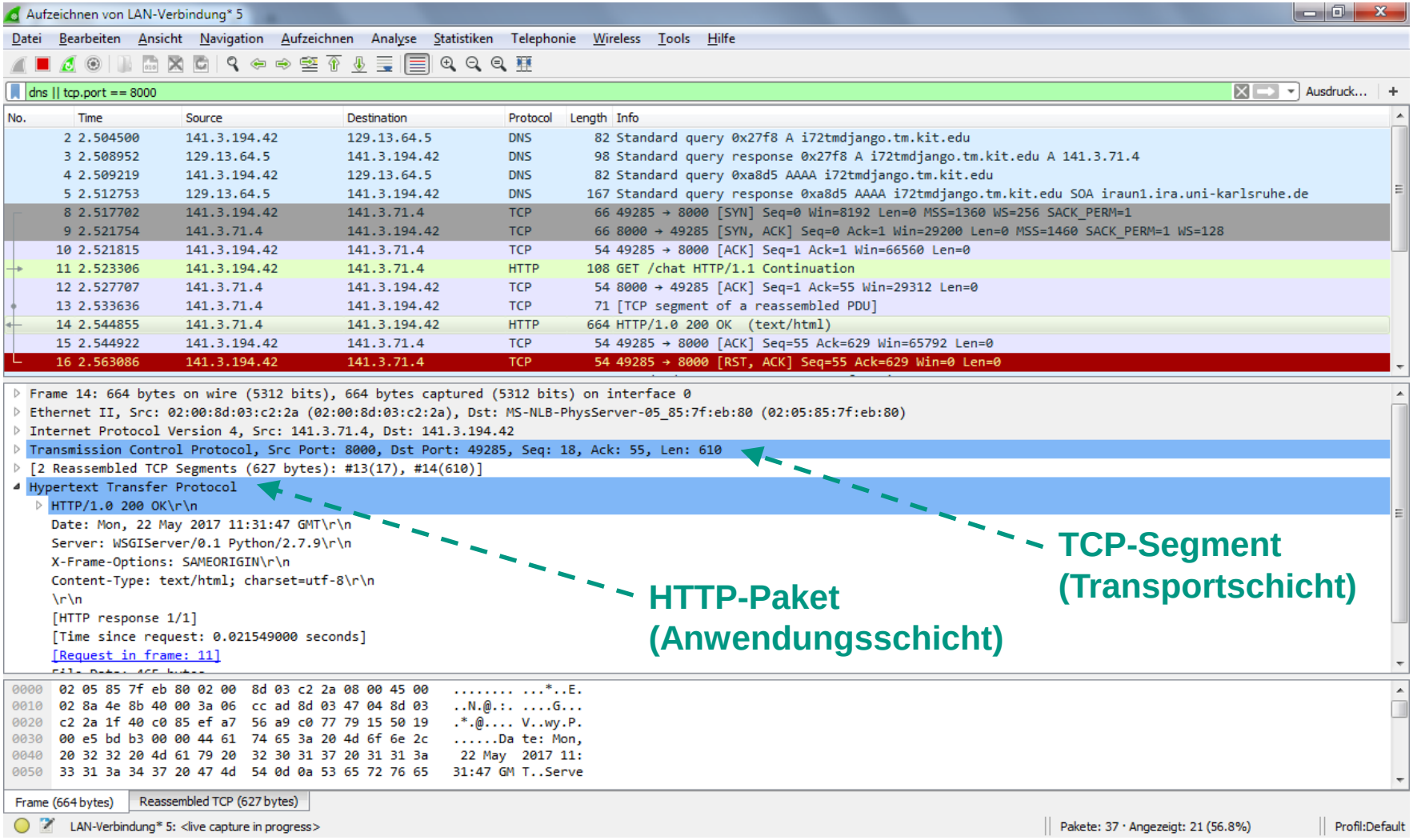
Pingo

http://pingo.upb.de/

Zusammenspiel: HTTP und TCP



Beispiel: TCP und HTTP



Stau im Internet!

… Netze werden schneller und Nutzung steigt

- Beobachtung
  - Drastischer Einbruch des Durchsatzes von TCP im Oktober 1986
    - Dann Serie so genannter „Staukollapse“
  - Stau im Netz … Puffer im Router füllen sich
    - Pufferüberlauf → Verwerfen von Dateneinheiten
      - … keine entsprechende Quittung vom Sender empfangen
      - … Sendewiederholungen durch den TCP-Sender
        → Verstärkung der Stausituation!

Staukontrolle

- Ziel
  - Vermeidung von Überlastsituationen im Netz
- Verfahren
  - Einführung eines Staukontrollfensters (CWnd, Congestion Window)
    - Beim Senden muss auch Empfangsfenster beachtet werden
      - … Minimum beider Fenstergrößen bestimmt maximal zu sendende Datenmenge

$$\text{LastByteSent} - \text{LastByteAcked} \leq \mathbf{min}\{\text{CWnd}, \text{RcvWindow}\}$$

  - … und eines Schwellenwertes (SSThresh)
  - Wie Stau erkennen?
    - Nutzung von Zeitgebern als Hilfsmittel für Stausignal
    - Ausbleibende Quittung … Vermutung einer Stausituation
  - Reaktion nach Erkennung?
    - Reduktion von CWnd
    - Langsames Erhöhen von CWnd … „herantasten“ an Netzkapazität

Ablauf TCP-Staukontrolle

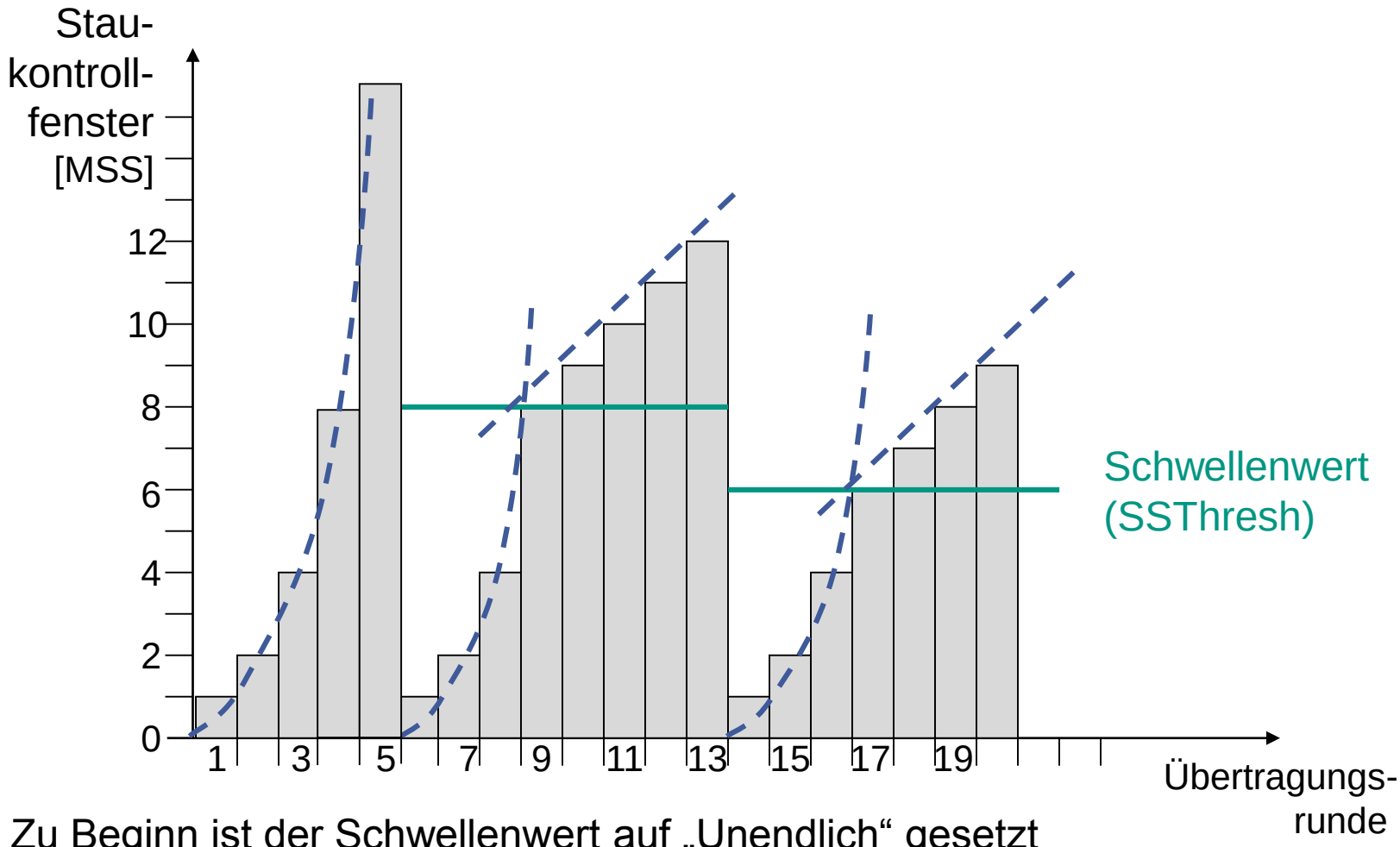
- Start: $\mathrm{CWnd} = 1\ MSS$ [Maximum Segment Size]
- $\mathrm{CWnd} \leq \mathrm{SSThresh}$ & Quittungen rechtzeitig empfangen: Slow-Start
  - Exponentielles Erhöhen des Staukontrollfensters
    - Bei jeder empfangenen Quittung: $\mathrm{CWnd} \mathrel{+}= 1$

- $\mathrm{CWnd} > \mathrm{SSThresh}$ & Quittungen rechtzeitig empfangen: Congestion Avoidance
  - Lineares Erhöhen des Staukontrollfensters
    - Erhöhen des Staukontrollfensters um 1 pro vollständig gesendetem und bestätigtem Sendefenster
    - Bei jeder empfangenen Quittung: $\mathrm{CWnd} \mathrel{+}= 1/\mathrm{CWnd}$

- Quittung nicht rechtzeitig empfangen (Timer abgelaufen)
  - Stau vermutet
  - $SSThresh = \max(FlightSize/2,\ 2 * MSS)$
    - FlightSize: Daten, die gesendet, aber noch nicht quittiert wurden
  - CWnd zurücksetzen (neue Slow-Start-Phase): $\mathrm{CWnd} = 1\ MSS$

Entwicklung des Staukontrollfensters

- Beispiel



- Zu Beginn ist der Schwellenwert auf „Unendlich“ gesetzt

Pingo

http://pingo.upb.de/

Ausblick: Protokoll-Engineering

- Detailliertes Protokoll-Engineering am Beispiel TCP in der Vorlesung Telematik
  - Dynamik und TCP
    - „Conservation of Packets“
    - Aktives Warteschlangenmanagement
  - Evaluierung von TCP
    - Periodisches Modell
    - Detailliertes Paketverlustmodell
  - TCP und Fairness
  - TCP in unterschiedlichsten „Einsatzgebieten“
    - Webanwendungen, Datenzentren …

ZUSAMMENFASSUNG

Zusammenfassung

Im Mittelpunkt der Vorlesung: **Das Internet**

- In diesem Kapitel
  - Aufgaben und Protokolle (UDP, TCP) der Transportschicht
    - Multiplexen, Demultiplexen
    - Unzuverlässige und zuverlässige Transportdienste
  - Prinzipien der zuverlässigen Datenübertragung
    - Bitfehler: Prüfsummen
    - Paketfehler: ARQ-Verfahren (Stop-and-Wait, Go-Back-N …)
  - Flusskontrolle
  - Staukontrolle

1) Wann versendet der Empfänger jeweils Quittungen?
2) Knoten A sendet mit Go-Back-N und Sliding-Window Daten an Knoten B. Die Fenstergröße betrage 4 Pakete. Zeichnen Sie die Fenster auf beiden Seiten
   1) Bevor A anfängt Daten zu senden
   2) Nachdem A die Pakete 0, 1 und 2 gesendet und B 0 und 1 quittiert hat
   3) Nachdem A die Pakete 3, 4 und 5 gesendet hat und B 4 quittiert hat
3) Welche Fehlerarten kennen Sie?
4) Eine Störung von $10\ \mathrm{ms}$ führt bei einer Datenrate von $100\ \mathrm{Mbit/s}$ zu wie vielen gestörten Bits?
5) Erklären Sie den Begriff des Hamming-Abstands. Warum werden unterschiedliche Hamming-Abstände zur Erkennung und Behebung von *r*-Bitfehlern benötigt?
6) Bei einem (15,11) Hamming Code: Wie viele Nutzdaten- und Paritätsbits sind enthalten?

7) Welche Aufgaben hat die Transportschicht?
8) Zwischen was genau transportiert die Transportschicht Daten?
9) Wie werden TCP- bzw. UDP-Anwendungen adressiert?
10) Durch welche Eigenschaften zeichnet sich das *User Datagram Protocol* aus und welche Vor- bzw. Nachteile besitzt es?
11) Erläutern Sie die Eigenschaften des *Transmission Control Protocols*
12) Geben Sie einige Beispiele für Anwendungen, bei denen Sie den Einsatz von UDP bzw. TCP bevorzugen würden und begründen Sie Ihre Wahl
13) Welches Verfahren kommt beim TCP-Verbindungsaufbau zum Einsatz?
14) Wozu wird für den Dienst, den TCP zur Verfügung stellt, überhaupt eine Verbindung benötigt?
15) Worin liegt der Unterschied zwischen Fluss- und Staukontrolle? Welche Mechanismen von TCP spielen in diesem Zusammenhang eine Rolle?

Literatur

[Benv05] Ch. Benvenuti; Understanding Linux Network Internals; O'Reilly; 2005

[Char09] J. Charzinski, Traffic, Structure and Locality Characteristic of the Web's Most Popular Services' Home Pages, Kommunikation in Verteilten Systemen (KiVS) 2009, Kassel, Germany

[FlFa99] S. Floyd, K. Fall, Promoting the Use of End-to-End Congestion Control in the Internet, IEEE/ACM Transactions on Networking, August 1999

[Hals05] F. Halsall, Computer Networking and the Internet, Addison-Wesley, 2005, 5th Edition

[KuRo12] James Kurose, Keith Ross, Computer Networking, 6/e, Pearson; 2012

[RFC768] J. Postel, User Datagram Protocol, RFC 768, August 1980

[RFC793] J. Postel (Ed.), Transmission Control Protocol, RFC 793, September 1981

[RFC1071] R. Braden, Computing the Internet Checksum, RFC 1071, September 1988

[Stal06] W. Stallings; High-Speed Networks: TCP/IP and ATM Design Principles, Prentice Hall, 2006

[Stal10] W. Stallings; Data and Computer Communications, Prentice Hall, 2010

[Stev94] W. R. Stevens, TCP/IP Illustrated, Volume 1, Addison-Wesley Professional, 1994