

Kommunikation und Datenhaltung

3. Protokollmechanismen



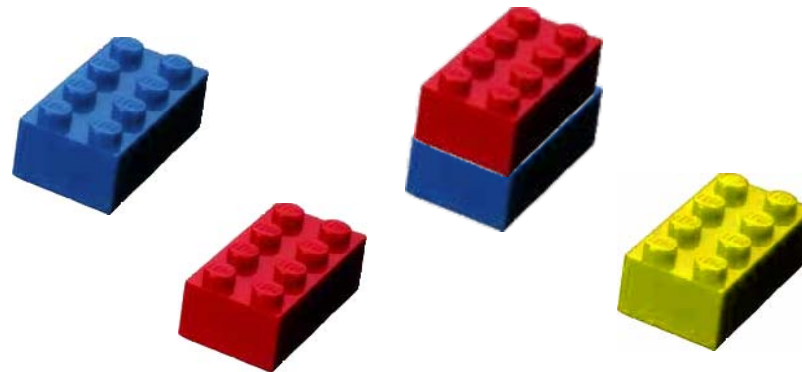
Prof. Dr. Martina Zitterbart
Dipl.-Inform. Martin Röhrich
[zit | roehricht]@tm.uka.de



1. Einführung
2. Physikalische Grundlagen
3. **Protokollmechanismen**
4. Geschichtete Architekturen
5. Sicherungsschicht: HDLC
6. Beschreibungsmethoden
7. Sicherungsschicht:
Lokale Netze
8. Netzkopplung und Vermittlung
9. Die Transportschicht
10. Anwendungssysteme
11. Middleware

- 3.1 Fehlertypen, Fehlerursachen
- 3.2 Mechanismen zur
Fehlererkennung und
-behebung
- 3.3 Fehlerkontrolle bei Bitfehlern
- 3.4 Fehlerkontrolle bei Paketfehlern
- 3.5 Flusskontrolle
- 3.6 Verbindungen

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



- Verfälschung von Bits bei der Übertragung - **Bitfehler**
 - Beispiel
 - ▶ Null-Bit werde durch 0 Volt repräsentiert; Eins-Bit durch 5 Volt
 - ▶ Entscheidungsschwelle sei 2,5 Volt
 - ▶ Übertragung ist nicht optimal: Rauschen, Signaldämpfung
 - ▶ Ergebnis: Empfänger empfängt Signalwert von 3 Volt, obwohl ursprünglich 0 Volt gesendet wurde
 - ▶ Es handelt sich dann um ein sogenanntes „umgekipptes“ Bit, d.h. um einen Bitfehler
 - Fehlerursachen
 - ▶ Rauschen
 - ▶ Verlust der Bit-Synchronisation

- Verfälschung von Dateneinheiten – oft als **Paketfehler** bezeichnet
 - Begriff **Dateneinheit** in der Vorlesung häufig als Oberbegriff verwendet für
 - ▶ Rahmen, Nachrichten, Pakete, Segmente ...
 - Fehlerarten
 - ▶ Verlust/Duplizierung einer Dateneinheit
 - ▶ Empfang einer Phantom-Dateneinheit
 - ▶ Abweichung der Empfangsreihenfolge von Dateneinheiten
 - Fehlerursachen
 - ▶ Überlastung von Zwischensystemen
 - ▶ Unterschiedliche Wege durch das Netz
 - ▶ Verfrühte Datenwiederholung
 - ▶ ...

- Paketfehler

	Verlust
	Phantom-Dateneinheit
	Duplizierung
	Reihenfolge-vertauschung
<ul style="list-style-type: none"> • Bitfehler 	Verfälschung

- Weiterhin wird unterschieden zwischen
 - Einzelbitfehler
 - ▶ Z.B. Rauschspitzen, die die Detektionsschwelle bei digitaler Signalerfassung überschreiten
 - ▶ Ein einzelnes Bit ist fehlerhaft
 - Bündelfehler
 - ▶ Länger anhaltende Störung durch Überspannung, Starkstromschaltprozesse etc.
 - ▶ Mehrere direkt aufeinanderfolgende Bits sind fehlerhaft
 - Synchronisationsfehler
 - ▶ Empfänger kann den Anfang eines Bits nicht korrekt detektieren
 - ▶ Alle Bits werden falsch erkannt

- Maß für die Fehlerhäufigkeit

$$\text{Bitfehlerrate} = \frac{\text{Summe gestörte Bits}}{\text{Summe übertragene Bits}}$$

- Typische Wahrscheinlichkeiten für Bitfehler
 - Analoges Fernsprechnet: $2 \cdot 10^{-4}$
 - Funkstrecke: $10^{-3} - 10^{-4}$
 - Ethernet (10Base2): $10^{-9} - 10^{-10}$
 - Glasfaser: $10^{-10} - 10^{-12}$



[Holz91]

- Fehlerauswirkungen sind u.a. abhängig von der Datenrate
 - Rechenbeispiel ... eine Störung von 20 ms führt
 - ▶ Bei Telex (50 bit/s, Bitdauer: 20 ms)
 - ▶ zu einem Fehler von 1 Bit
 - Einzelbitfehler
 - ▶ Bei ISDN (64 kbit/s, Bitdauer: 15,625 μ s)
 - ▶ zu einem Fehler von 1280 Bit
 - Bündelfehler
 - ▶ Bei ADSL2+ (16 Mbit/s, Bitdauer: 62,5 ns)
 - ▶ zu einem Fehler von ca. 320 kbit
 - ▶ ... und bei Gigabit-Ethernet (1 Gbit/s)?

- **Unzuverlässige** Kommunikation
 - Sender und Empfänger sind „glücklich“, wenn möglichst viel Daten korrekt beim Empfänger ankommen
 - Bei Fehlern werden keine weiteren Maßnahmen unternommen
- **Zuverlässige** Kommunikation
 - Sender und Empfänger erwarten, dass alle gesendeten Daten korrekt beim Empfänger ankommen
 - ▶ Alle Daten korrekt
 - ▶ In der richtigen Reihenfolge
 - ▶ Ohne Duplikate
 - ▶ Ohne Phantom-Dateneinheiten
 - Bei Fehlern sind entsprechende Maßnahmen erforderlich

- **Bezüglich Bitfehlern**
 - Fehlererkennende Codes
 - Fehlerkorrigierende Codes

- **Bezüglich kompletter Dateneinheiten**
 - Sequenznummern
 - Zeitüberwachung
 - Quittungen
 - Sendewiederholungen

- **Redundanz**
 - Fehlererkennung von Datenfehlern beim Empfänger durch Hinzufügung von Redundanz beim Sender

- Gegeben
 - eine Menge A und ein (endliches) Alphabet B
- Code
 - injektive Abbildung $f: A \rightarrow B^*$
- Codewort
 - Für $a \in A$ heißt $f(a)$ ein Codewort von f
- Beispiele
 - Morse Code
 - ASCII Code
 - ...

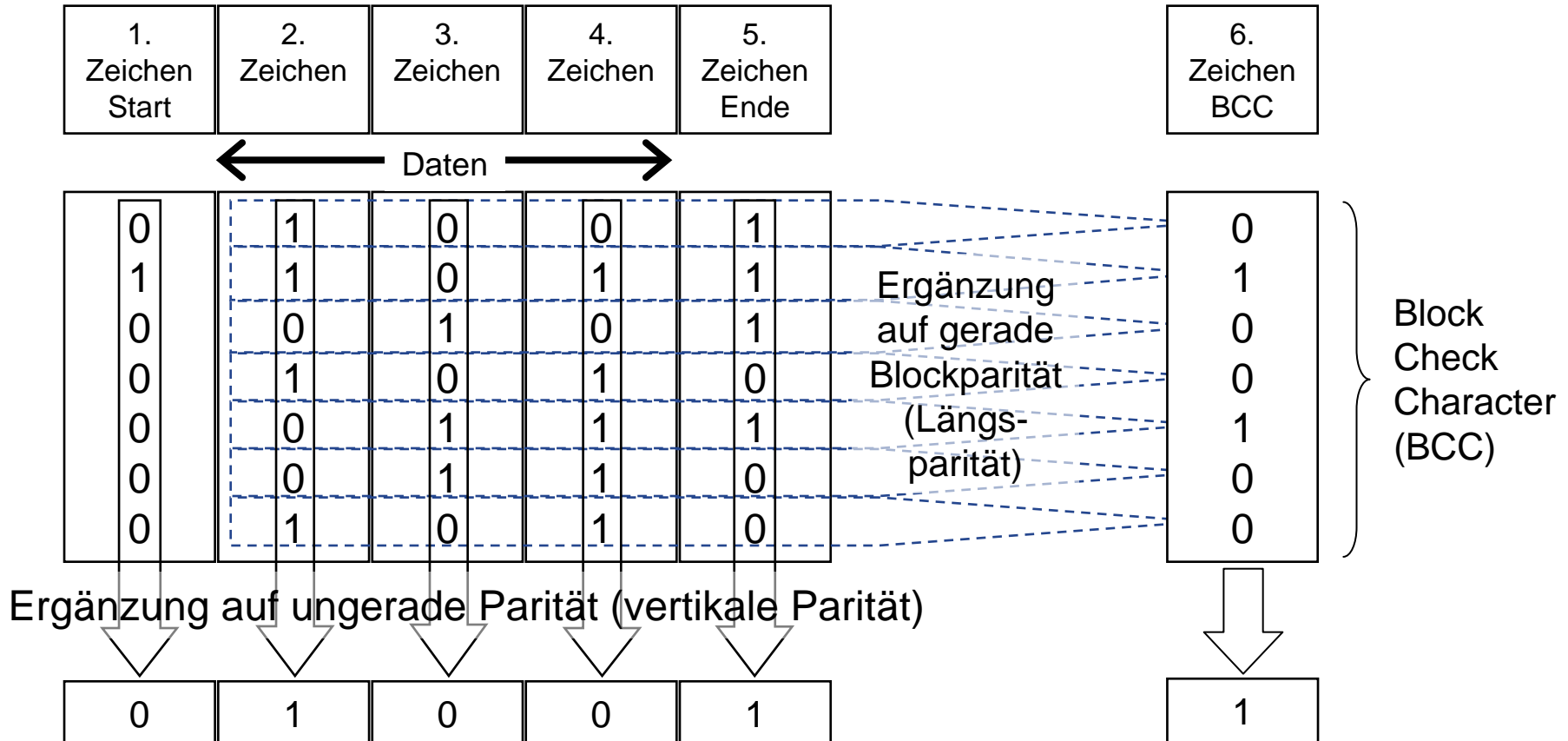
A	.-	N	-. .	0	-----
B	-... .	O	--- .	1	.----
C	-.-. .	P	.-.. .	2	..---
D	-.. .	Q	--. .	3	...--
E	. .	R	.-. .	4-
F	..-. .	S	5
G	---. .	T	- .	6	-....
H	U	..- .	7	--... .
I	.. .	V	...- .	8	---.. .
J	.-... .	W	.-. .	9	----. .
K	-.- .	X	-..- .	Fullstop	.-.-.- .
L	.-.. .	Y	-.-- .	Comma	--..-- .
M	-- .	Z	--.. .	Query	..--.. .

- Problem
 - Wie können **Bitfehler** beim Empfänger oder in netzinternen Zwischensystemen erkannt werden?

- Grundlegende Ansätze
 - Ausnutzung der „Distanz“ zwischen gültigen Codewörtern, d.h. nicht alle Codewörter, die mit den vorhandenen Bits erzeugt werden können, sind gültig
 - ▶ z.B. Hamming-Abstand
 - Hinzufügen von Redundanz bei der Übertragung
 - ▶ z.B. Paritätsbits, CRC, ...

- Zu einer „Einheit“ wird jeweils ein redundantes Bit hinzugefügt.
 - Gerade Parität
 - ▶ Es wird auf gerade Anzahl von 1-Bits ergänzt
 - Ungerade Parität
 - ▶ Es wird auf ungerade Anzahl von 1-Bits ergänzt
- Folgende Varianten werden unterschieden
 - Vertikale Parität
 - ▶ An jede einzelne Einheit (Zeichen) – bestehend aus n Bits – wird ein Paritätsbit angefügt (d.h. ein Paritätsbit pro Reihe)
 - ▶ Erkennung von Bitfehlern ungerader Anzahl (1-Bitfehler, 3-Bitfehler etc.)
 - Längsparität
 - ▶ An eine Folge von Einheiten wird ein dediziertes Prüfzeichen angefügt. Dieses enthält jeweils ein Paritätsbit pro Spalte (d.h. pro n -tem Bit aller Zeichen)
 - ▶ Auch als Block Check Character (BCC) bezeichnet.

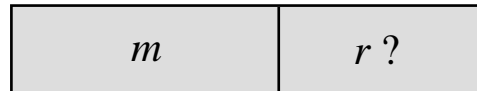
- Paritätssicherung bei Zeichen-basierter Übertragung
 - Modulo-2-Arithmetik



- Hamming-Abstand d einzelner Codewörter
 - Anzahl der Bitpositionen, in denen sich zwei Codewörter c_1 und c_2 unterscheiden
 - Beispiel
 - ▶ $d(10001001, 10110001) = 3$
 - ▶ Anzahl der Eins-Bits von $c_1 \text{ XOR } c_2$
- Hamming-Abstand D des vollständigen Codes C
$$D(C) := \min\{d(c_1, c_2) \mid c_1, c_2 \in C, c_1 \neq c_2\}$$
- Der Hamming-Abstand bestimmt die Fähigkeit eines Codes, Fehler zu erkennen und zu beheben
 - Erkenne k -Bitfehler: Hamming-Abstand von $k + 1$ notwendig
 - Behebe k -Bitfehler: Hamming-Abstand von $2k + 1$ notwendig

- Codewort besteht aus m Bits

- Welche Anzahl Prüfbits r werden benötigt, um 1-Bitfehler zu beheben?



$$n = m + r$$

- 2^m legale Codewörter können mit m Bits erzeugt werden
- Allgemein gilt: Pro Codewort c_i der Länge $n=m+r$ existieren n illegale Codewörter gleicher Länge mit Hamming-Abstand 1
 - ▶ Jeweils mit einem komplementierten Bit
 - ▶ Nur Codewort c_i selbst hat Hamming-Abstand 0
- Es soll 2^m legale Codewörter im Code geben, von denen jedes $n+1$ Codewörter belegt (n illegale und ein legales, c_i selbst)
- Hierzu müssen $(n + 1)2^m$ Codewörter mit $n=m+r$ Bits darstellbar sein
 - ▶ $(n + 1)2^m \leq 2^n \rightarrow (m + r + 1) \leq 2^r$ untere Grenze für r

- Fehlererkennender Code

- Code mit einem einzigen Paritätsbit (gerade oder ungerade)
- Hamming-Abstand = 2
- Erkennen eines 1-Bitfehlers möglich (oder aller Fehler mit einer ungeraden Anzahl Bits)

- Fehlerbehebender Code

- Code: 00000 00000, 00000 11111,
11111 00000, 11111 11111
- Hamming-Abstand = 5
- Korrektur von 2-Bitfehlern möglich
- Beispiel: 00000 00111 → 00000 11111

3.3.3 Cyclic Redundancy Check (CRC)

- Anstelle eines einzigen Paritätsbits wird hier eine Sicherungssequenz angefügt.
 - auch als FCS (Frame Check Sequence) bezeichnet
- Basiert auf Division in Modulo-2-Arithmetik; keine Überträge
 - Entspricht bitweiser XOR-Operation
 - ▶ $1+1 = 0+0 = 0, 1+0 = 0+1 = 1$
 - Beispiele für bitweise XOR-Operation

$$\begin{array}{r} 10011011 \\ +11001010 \\ \hline 01010001 \end{array}$$

$$\begin{array}{r} 00110011 \\ +11001101 \\ \hline 11111110 \end{array}$$

$$\begin{array}{r} 11110000 \\ -10100110 \\ \hline 01010110 \end{array}$$

$$\begin{array}{r} 01010101 \\ -10101111 \\ \hline 11111010 \end{array}$$

- Bitstrings als Repräsentation von Polynomen, z.B. Dateneinheit 10011010 entspricht Polynom $M(x) = x^7 + x^4 + x^3 + x^1$
- Dateneinheit wird als unstrukturierte Bitfolge aufgefasst, d.h. auch die Anzahl der zu prüfenden Bits ist beliebig
 - Es werden auch keine ganzzahligen Vielfache von 8 Bit gefordert

- Gleiches Generatorpolynom $G(x)$ für Sender und Empfänger
 - Höchstes und niederwertigstes Bit von $G(x)$ müssen 1 sein

- Prüfsumme (Checksum) wird berechnet
 - Dateneinheit mit m Bits entspricht $M(x)$
 - Prüfsumme entspricht Rest R der Division $(x^r M(x)) / G(x)$
 - ▶ r : Grad des Generatorpolynoms; $m > r$
 - ▶ $x^r M(x)$ fügt r Nullstellen an das Ende der Dateneinheit

- Prüfsumme wird zur um Nullen erweiterten Dateneinheit hinzugefügt
 - Entspricht der Addition des Restes: $x^r M(x) + R$

- Empfänger überprüft Dateneinheit
 - Division der empfangenen Dateneinheit durch $G(x)$
 - ▶ Ist der Rest der Division Null, dann wurde kein Fehler erkannt
 - ▶ Ist der Rest ungleich Null, dann ist die empfangene Dateneinheit fehlerhaft

- Generatorpolynom $G(x) = 1101$; Nachricht $M(x) = 1001\ 1010$

$$\begin{array}{r}
 1001\ 1010\ 000 \quad / \quad 1101 \quad = \quad 1111\ 1001 \\
 \hline
 1101 \\
 \hline
 100\ 1 \\
 110\ 1 \\
 \hline
 10\ 00 \\
 11\ 01 \\
 \hline
 1\ 011 \\
 1\ 101 \\
 \hline
 1100 \\
 1101 \\
 \hline
 1\ 000 \\
 1\ 101 \\
 \hline
 101
 \end{array}$$

- Welche Bitfolge wird übertragen?

- Erkennen aller Einzelbitfehler
 - \mathbf{x}^k und \mathbf{x}^0 dürfen nicht gleich Null sein

- Nahezu alle Doppelbitfehler
 - $\mathbf{G}(\mathbf{x})$ muss mindestens drei Terme besitzen
 - sämtliche Doppelbitfehler
 - ▶ $(\mathbf{x}^k + 1)$ nicht durch $\mathbf{G}(\mathbf{x})$ teilbar ($\forall k \leq$ Länge der Dateneinheit)

- Jede ungerade Anzahl an Bitfehlern
 - $\mathbf{G}(\mathbf{x})$ muss den Faktor $\mathbf{x}+1$ enthalten

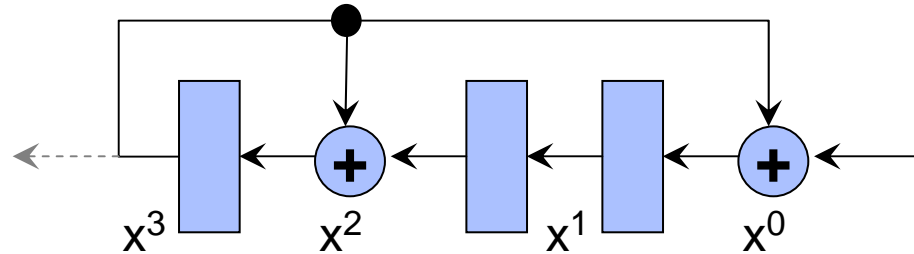
- Alle Bursts mit bis zu m Bitfehlern
 - für $m =$ Grad von $\mathbf{G}(\mathbf{x})$

- International genormt sind u.a. folgende Generatorpolynome
 - CRC-12 = $x^{12} + x^{11} + x^3 + x^2 + x + 1$
 - CRC-16 = $x^{16} + x^{15} + x^2 + 1$
 - CRC-CCITT = $x^{16} + x^{12} + x^5 + 1$
 - CRC-32 = $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
- CRC-16 und CRC-CCITT entdecken
 - alle Einzel- und Doppelfehler
 - alle Fehler ungerader Anzahl
 - alle Fehlerbursts mit der Länge ≤ 16
 - 99,997 % aller Fehlerbursts mit der Länge 17 (gilt nicht für CRC-16)
 - 99,998 % aller Fehlerbursts mit der Länge 18 und mehr (gilt nicht für CRC-16)

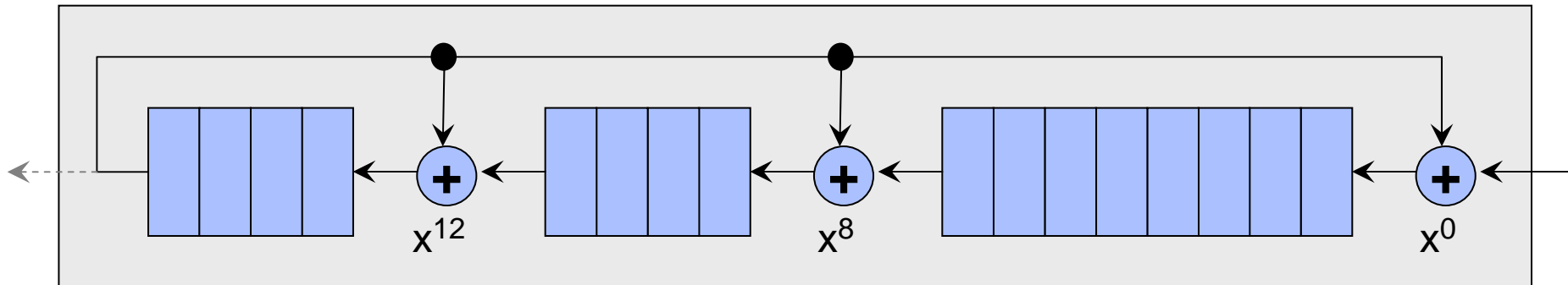
- Realisierung in Hardware
 - Benutzung von rückgekoppelten Schieberegistern
 - CRC kann während des „Durchschiebens“ durch das Schieberegister berechnet werden
- Prinzip
 - Daten werden bitweise empfangen und durchlaufen das Schieberegister
 - Rückkopplung durch ein XOR-Gatter erfolgt am Eingang und an den Stellen, an denen Bits im Generatorpolynom auf 1 gesetzt sind
 - ▶ Ohne das höchste und niederwertigste Bit
 - Nach Durchschieben der Dateneinheit und ihrer angehängten Nullen steht Prüfsumme im Register
 - ▶ **Zu beachten:** Soll Prüfsumme ebenfalls „aus dem Register geschoben werden“, noch etwas mehr Schaltungslogik nötig

- $G(x) = 1101$

$$G(x) = x^3 + x^2 + 1$$



- $G(x) = x^{16} + x^{12} + x^8 + 1$

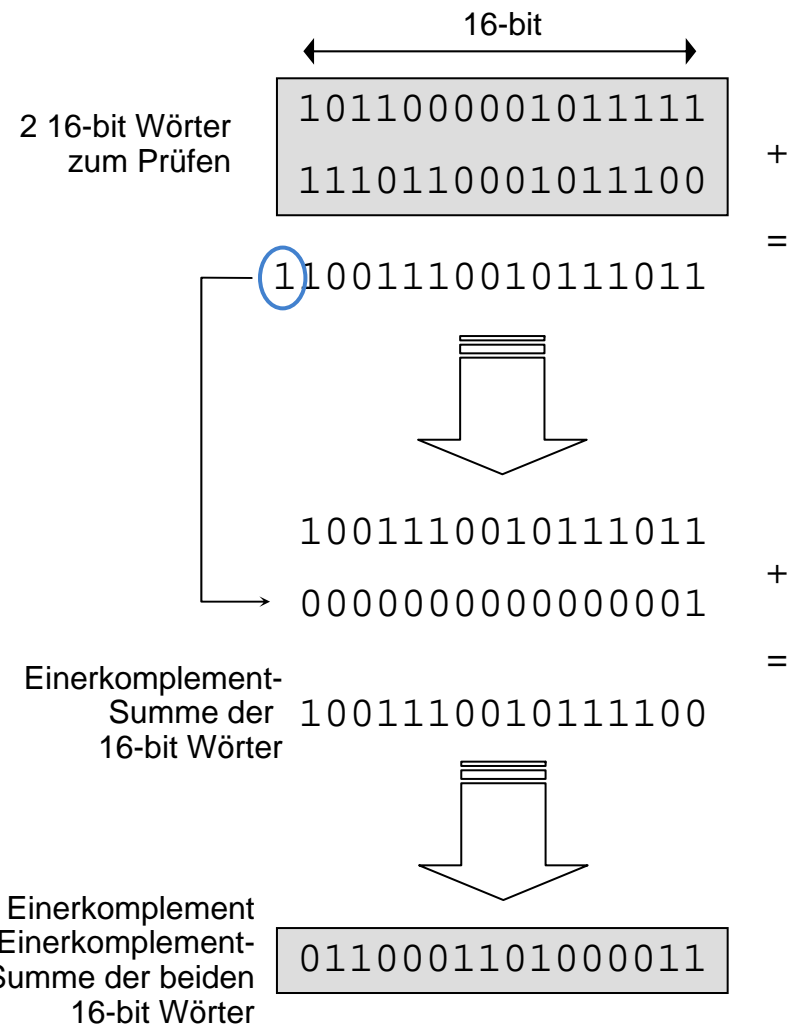


XOR-Gatter

1 Bit-Schieberegister

- In den Internet-Protokollen wird eine andere Variante für Prüfsummen eingesetzt
 - Speziell für eine Realisierung in Software ausgelegt
 - In den Eigenschaften zur Fehlererkennung nicht so gut wie CRC
 - ▶ Wörter in falscher Reihenfolge können nicht erkannt werden
 - Verwendet bei: Internet Protocol (IP), Universal Datagram Protocol (UDP), Transmission Control Protocol (TCP)
- Prinzip
 - Aufaddieren aller übertragenen Wörter, wobei eine Wortlänge von 16 Bit zugrunde liegt (die Wörter werden als **Integer** aufgefasst)
 - Resultat der Addition wird wie beim CRC mit der Dateneinheit übertragen
 - Empfänger führt die gleiche Addition der empfangenen Wörter durch
- Implementierung
 - Addition unter Verwendung des Einer-Komplements

- Einerkomplement zur Berechnung von BCC
- Internet Protocol Version 4
 - Protokollkopf besitzt Prüfsumme für Werte des Protokollkopfs
 - ▶ Muss bei jedem Router neu berechnet werden, da sich Werte wie Time to Live im Protokollkopf ändern
 - Prüfsummenberechnung
 - ▶ Addition aller 16-bit Wörter
 - ▶ Übertrag auf das Least Significant Bit addieren
 - ▶ Einerkomplement der Summe bilden



[Benv05]

- Vorwärtsfehlerkorrektur (Forward Error Correction – FEC)
 - Fehlerkorrigierende Codes
- Soll dazu dienen, verloren gegangene Dateneinheiten zu rekonstruieren

- Beispiel:

Zu senden sind die Dateneinheiten

0101 - D1

1111 - D2

0000 - D3

- Dazu wird über XOR eine weitere Dateneinheit berechnet

1010 - D4

- Diese vier Dateneinheiten werden jetzt an den Empfänger gesendet



- Der Empfänger muss nur drei der vier Dateneinheiten korrekt empfangen, um die fehlende Dateneinheit rekonstruieren zu können
 - Verknüpft korrekt empfangene Dateneinheiten mit XOR und rekonstruiert so die fehlende
 - ▶ D1 geht verloren

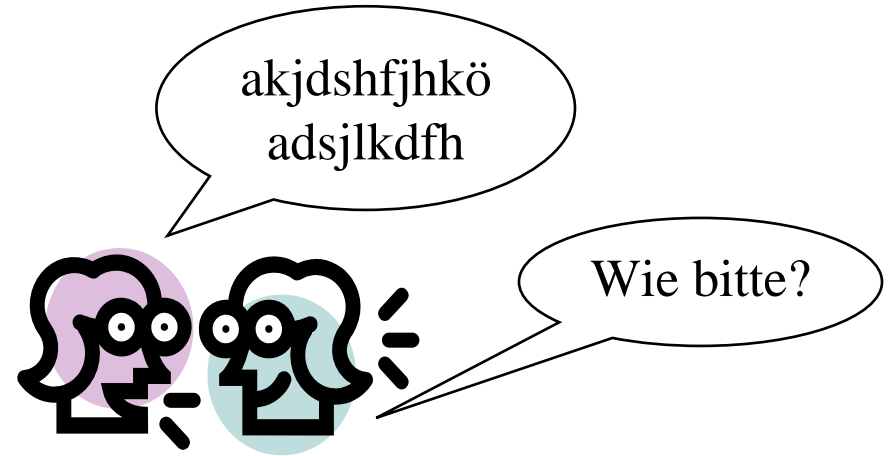
1111
0000
1010
0101 - D1
 - ▶ D2 geht verloren

0101
0000
1010
1111 - D2
 - ▶ D3 geht verloren

0101
1111
1010
0000 - D3
- Der Empfänger muss wissen, welche Dateneinheit verloren gegangen ist ...

- Mit Paritätsbits und Prüfsummen können Bitfehler erkannt werden, allerdings nur dann, wenn die Dateneinheit beim Empfänger ankommt oder in den netzinternen Zwischensystemen analysiert wird
- Zur **Erkennung** von Fehlern bzgl. kompletter Dateneinheiten (Paketfehler) sind zusätzliche Mechanismen erforderlich
 - **Sequenznummern** (*Sequence Number*)
 - **Zeitgeber** (*Timer*)
- Auch wenn alle Dateneinheiten empfangen wurden, können die genannten Mechanismen erforderlich sein
 - Nennen Sie Gründe hierfür!
- Zur **Behebung** der Fehler werden die folgenden Mechanismen verwendet
 - **Quittungen** (*Acknowledgements*)
 - **Sendewiederholungen** (*Retransmissions*)

... Verständigungsprobleme?



1, 2, 3, 5, ... etwas fehlt hier ...

- Problem
 - Woher weiß der Empfänger, ob
 - ▶ die Dateneinheiten in der richtigen Reihenfolge ankommen?
 - ▶ keine Duplikate enthalten sind?
 - ▶ keine Dateneinheiten fehlen?

- Mechanismus
 - Die Dateneinheiten (oder die Bytes) werden durchnummeriert
 - Eine entsprechende Kennung wird mit jeder Dateneinheit übertragen
 - Diese Kennung wird als **Sequenznummer** bezeichnet

- Fehlerszenarien
 - Sequenznummern helfen, wenn Dateneinheiten nicht ausgeliefert werden
 - ▶ Beispiele?

- Definition
 - Teilmenge der natürlichen Zahlen, die den möglichen Sequenznummernraum umfasst
- Größe
 - Bei einer Länge der Sequenznummer von n Bit umfasst der Sequenznummernraum 2^n Sequenznummern
- Erforderliche Länge der Sequenznummer?
 - Ziel
 - ▶ Overhead durch Übertragung von Sequenznummern gering halten
 - Parameter
 - ▶ T_l : maximale „Lebenszeit“ einer Dateneinheit [Sekunden]
 - ▶ T_w : Maximaler Zeitraum in dem eine Sendewiederholung durchgeführt wird [Sekunden]
 - ▶ T_q : Maximale Zeit bevor Empfänger nach Erhalt der Daten eine Quittung sendet [Sekunden]
 - ▶ D : Maximale Datenrate des Senders [Dateneinheiten/Sekunde]
 - Untere Schranke
 - ▶ $2^n \geq (2 T_l + T_w + T_q) * D$

- Problem
 - Wie erfährt der Sender, dass eine Dateneinheit überhaupt nicht bzw. nicht korrekt beim Empfänger angekommen ist?
- Mechanismus
 - Der Empfänger teilt dem Sender mit, ob er eine Dateneinheit empfangen hat oder nicht
 - ▶ Hierzu werden spezielle Dateneinheiten, sogenannte Quittungen, versendet (ACK: [Acknowledgement](#))
- Varianten
 - [Positive Quittung](#)
 - ▶ Empfänger teilt dem Sender mit, dass er die entsprechenden Daten erhalten hat
 - [Negative Quittung](#)
 - ▶ Empfänger meldet dem Sender, dass er die entsprechenden Daten nicht erhalten hat (z.B. bei falscher Reihenfolge)
 - ▶ NACK: [Negative Acknowledgement](#)

- Weitere Varianten
 - Selektive Quittungen
 - ▶ SACK: *Selective Acknowledgement*
 - ▶ Die Quittung bezieht sich auf eine einzelne Dateneinheit
 - ▶ Beispiel
 - ▶ Negative selektive Quittung, falls der Verlust einer Dateneinheit vom Empfänger vermutet wird (NACK)
 - ▶ Lässt sich mit NACKs eine zuverlässige Kommunikation realisieren?
 - Kumulative Quittungen
 - ▶ Die Quittung bezieht sich auf eine Menge von Dateneinheiten, die in der Regel durch eine obere Sequenznummer beschränkt ist
 - ▶ Beispiel
 - ▶ Positive kumulative Quittung, die besagt, dass alle Dateneinheiten bis zur angegebenen Sequenznummer korrekt empfangen wurden
- Quittungen werden oftmals in Kombination mit Zeitgebern verwendet

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

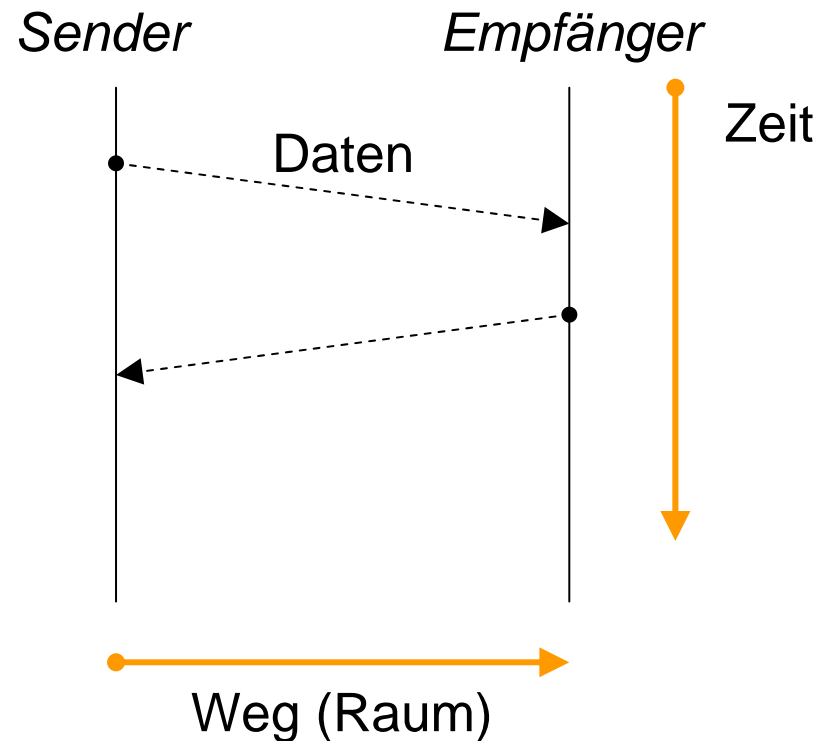
- Grundlegende Variante zur **Sendewiederholung**
 - Sender erhält positive Quittungen über den Erhalt einer Dateneinheit vom Empfänger
 - Sender kann Sendewiederholungen ausführen, falls Quittung ausbleibt

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

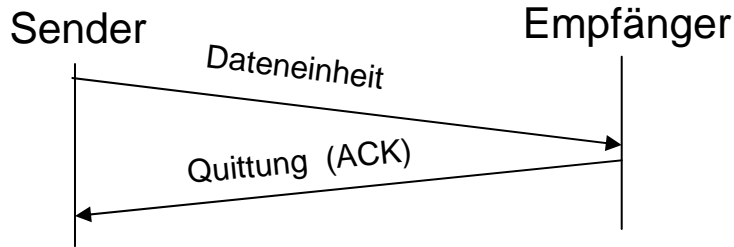
- Situation
 - Eine Reihe verschiedener ARQ-Mechanismen ist verfügbar, wobei zwischen grundlegenden Unterschieden und Implementierungseinheiten differenziert werden muss
 - Im folgenden werden grundlegende Varianten diskutiert

- Grundlegendes, einfaches ARQ-Verfahren
 - Sender wartet auf die Quittung zu einer gesendeten Dateneinheit, bevor er eine neue Dateneinheit senden darf
 - Falls keine Quittung empfangen wird, erfolgt die Wiederholung der Dateneinheit nach dem Ablauf des Zeitgebers
 - Es können Duplikate von Dateneinheiten entstehen
 - ▶ Wie?
- Einsatz beim Medienzugriff in drahtlosen lokalen Netzen

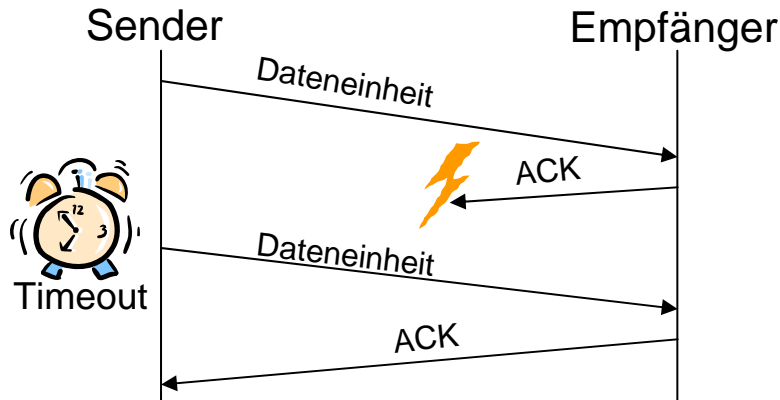
- Beschreibungsmethode zur Darstellung von Abläufen in Kommunikationssystemen
 - Vorwiegend für einfachere Sachverhalte geeignet



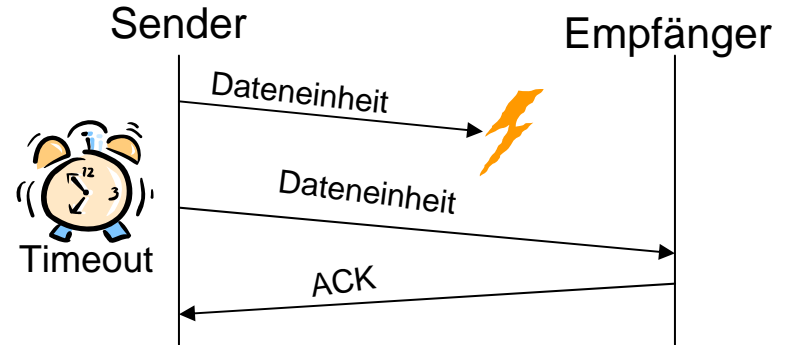
- Regulärer Ablauf



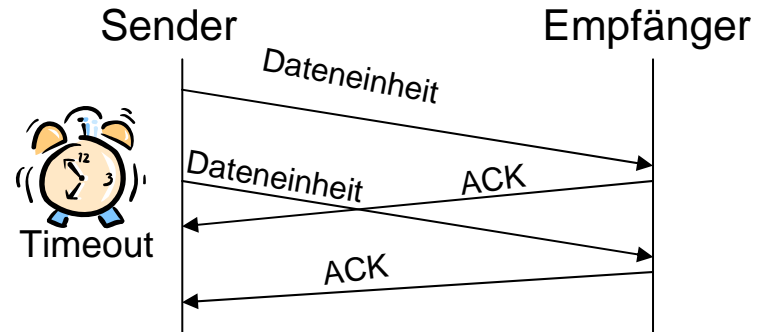
- Verlust einer Quittung
 - Unterschied zum Verlust einer Dateneinheit?



- Verlust einer Dateneinheit

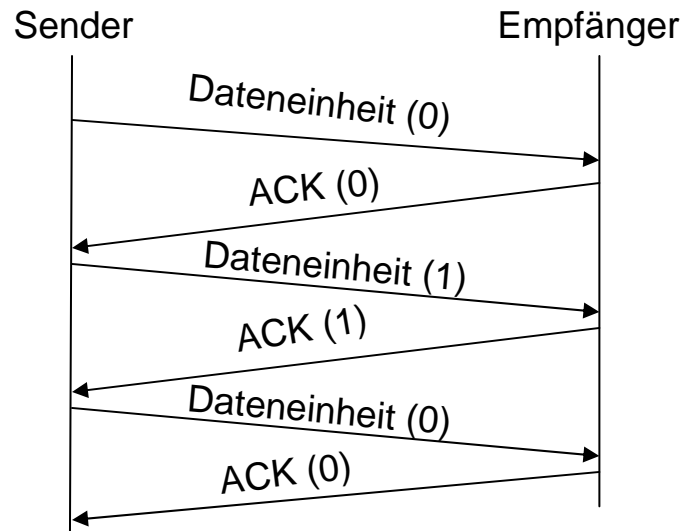


- Zu schneller Ablauf des Zeitgebers
 - Problem?



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

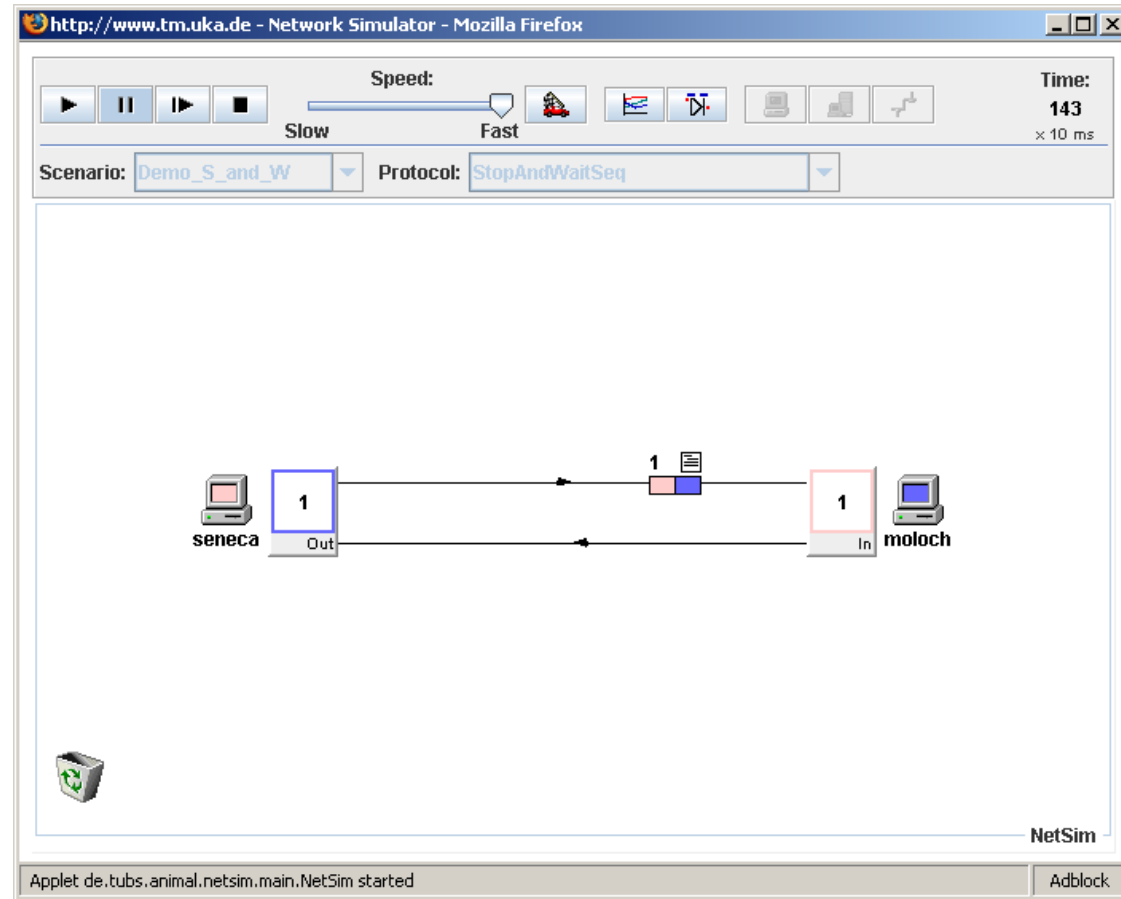
- Problem
 - In den vorangegangenen Szenarien besteht die Möglichkeit, dass der Empfänger eine Dateneinheit doppelt erhält
 - ▶ Er kann dies nicht erkennen
 - ▶ Konstruieren Sie einen solchen Ablauf!
- Mechanismus
 - Sequenznummern
 - ▶ Die Dateneinheiten werden mit einer Kennung versehen, die es dem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



Testen Sie hierzu unsere Java-Applets und unsere Lernumgebung im Internet:

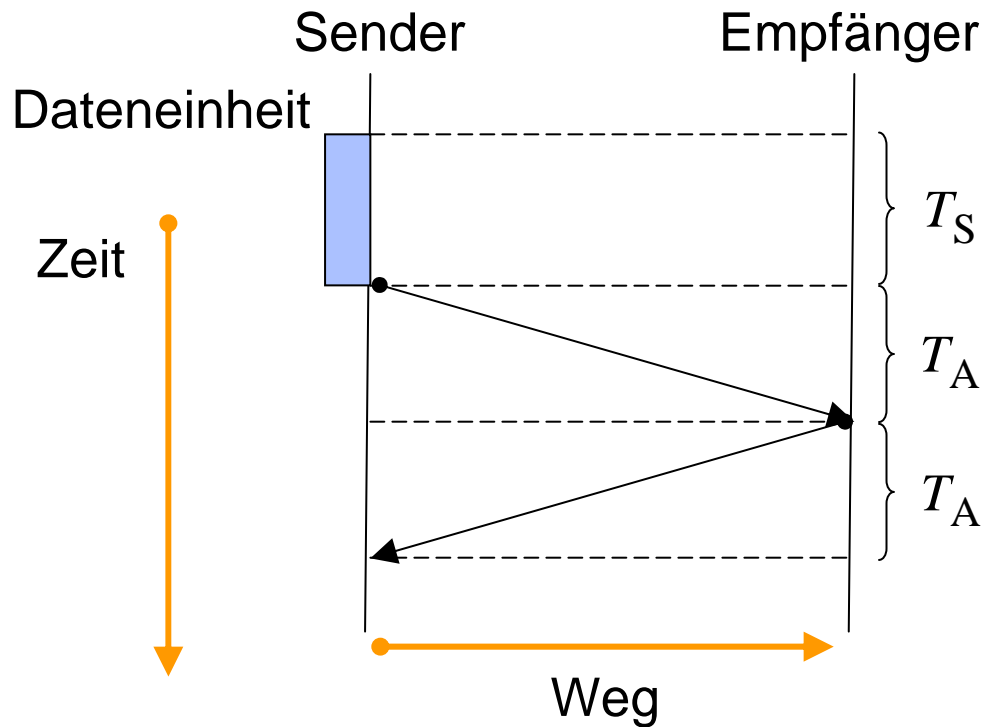
<http://www.tm.uka.de/lehre/aktuell/vl/telematik/sim/simulation.html>

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



- Sehr einfaches Verfahren
- Wie sieht es mit der Leistungsfähigkeit aus?
- **Durchsatz** (engl. *Throughput*)
 - Maß für die Menge an Daten, die pro Zeiteinheit über ein Netz übertragen werden kann
- **Auslastung** (engl. *Utilization – U*)
 - Maß für die Nutzung einer Ressource (z.B. Übertragungsmedium)
 - Verhältnis von tatsächlicher Nutzung zu möglicher Nutzung
 - ▶ Tatsächliche Nutzung: erfolgreiche Übertragung einer Dateneinheit
 - ▶ Mögliche Nutzung: Wieviel Daten hätten in dieser Zeit übertragen werden können?

- Annahme
 - Fehlerfreie Kommunikation
- Welche Parameter haben einen Einfluss?
 - Wie schnell ist eine komplette Dateneinheit bzw. eine Quittung auf das Medium gesendet?
 - ▶ **Sendezeit** – T_S
 - ▶ $T_S = \text{Länge Dateneinheit} / \text{Datenrate}$
 - Wie lange benötigt ein Bit vom Sender zum Empfänger (und umgekehrt)?
 - ▶ **Ausbreitungsverzögerung** – T_A
 - ▶ $T_A = \text{Länge des Mediums} / \text{Ausbreitungsgeschwindigkeit}$
 - Wieviel Zeit wird für die Verarbeitung einer Dateneinheit bzw. einer Quittung benötigt?
 - ▶ **Verarbeitungszeit** – T_V
- Vereinfachungen
 - Verarbeitungszeit der Dateneinheit wird vernachlässigt
 - Sende- und Verarbeitungszeit einer Quittung werden vernachlässigt
 - ▶ Quittung kurz; Verarbeitung schnell



- 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
 - $T_{Ges} = T_S(\text{Daten}) + 2 T_A = T_S + 2 T_A$

- Auslastung U

- Tatsächliche Nutzung

- ▶ Sendezeit der Dateneinheit: T_S

- Mögliche Nutzung

- ▶ Zeitintervall vom Beginn des Sendens der Dateneinheit bis zum vollständigen Empfang der Quittung: T_{Ges}

$$U = \frac{T_S}{T_{Ges}} = \frac{T_S}{T_S + 2T_A}$$

$$U = \frac{1}{1 + 2T_A/T_S}$$

- ▶ Mit $a = T_A / T_S$ ergibt sich

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

- Häufig bei Leistungsbetrachtungen verwendet

- Definition

- $a = T_A / T_S$
- $a = \text{Ausbreitungsverzögerung} / \text{Sendezeit}$

- Weitere Parameter

- Länge des Mediums: m
- Ausbreitungsgeschwindigkeit: v
- Länge der Dateneinheit: l
- Datenrate: d

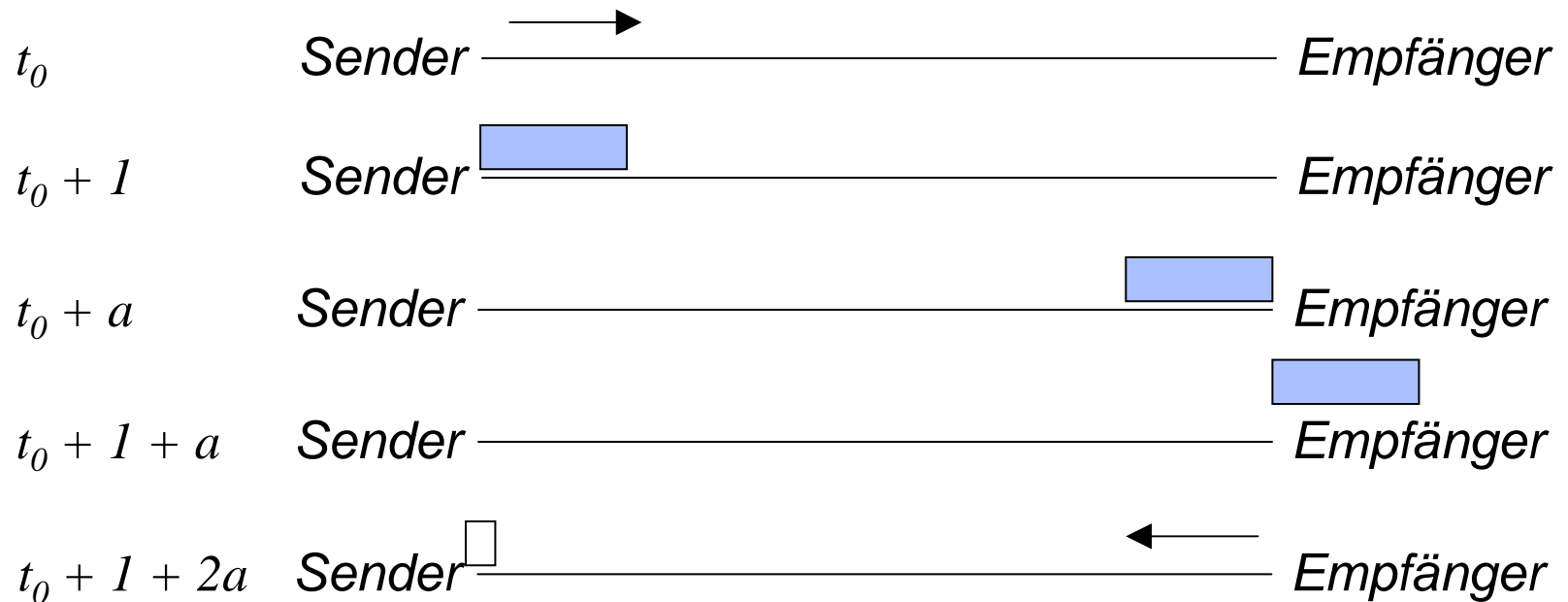
- Damit

$$a = \frac{m/v}{l/d} = \frac{md}{vl}$$

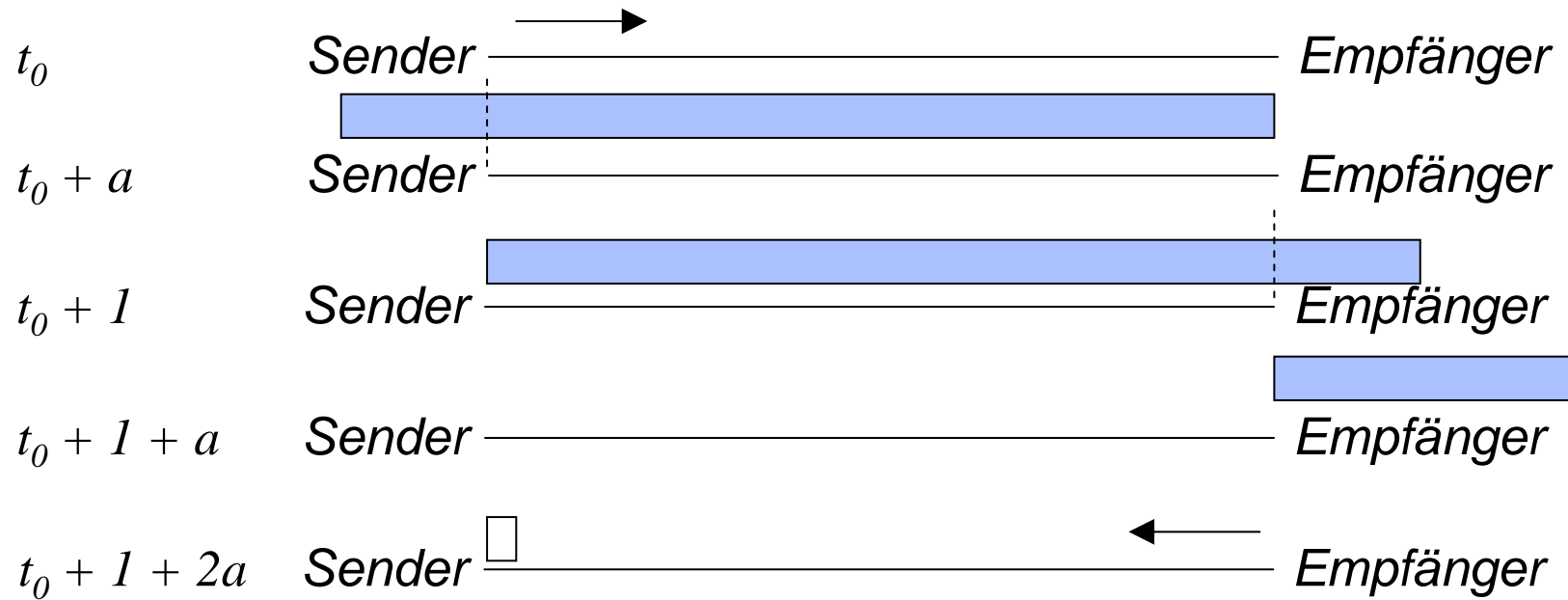
- Interpretation

- Bei fester Länge der Dateneinheit und fester Distanz zwischen den kommunizierenden Systemen
 - ▶ Parameter a ist proportional zum Produkt aus Datenrate und Länge des Mediums
- Parameter a repräsentiert Länge des Mediums in Bit ($m * d/v$) im Vergleich zur Länge der Dateneinheit (l)

- Normalisierung
 - Annahme: Sendezeit einer Dateneinheit betrage 1
- Damit gilt
 - $a = T_A$
 - Für $a > 1$: Ausbreitungsverzögerung $>$ Sendezeit



- Für $a < 1$: Ausbreitungsverzögerung $<$ Sendezeit



- Dateneinheiten der Länge 1000 Bit
- Fehlerrate vernachlässigbar
- Datenraten 1 kbit/s und 1 Mbit/s
- Ausbreitungsgeschwindigkeit:
 - (a) und (b) $2 * 10^8 \text{ m/s}$
 - (c) $3 * 10^8 \text{ m/s}$
 - ▶ (a) Verdrilltes Adernpaar, Länge des Mediums 1 km
 - ▶ Auslastung?
 - ▶ (b) Standleitung, Länge des Mediums 200 km
 - ▶ Auslastung?
 - ▶ (c) Satelliten-Verbindung, Länge des Mediums 50 000 km
 - ▶ Auslastung?

- Bisher: fehlerfreie Übertragung angenommen
- Jetzt: Berücksichtigung von Fehlern

- Fehlerfall
 - Sender erhält keine korrekte Quittung

- Annahme: $(n - 1)$ konsekutive Sendewiederholungen
 - Benötigte Zeit hierfür
 - ▶ $T_{\text{Ges}} = T_S + (n - 1) (\text{Timeout} + T_S) + 2 T_A$
 - Annahme
 - ▶ Timeout entspricht zweifacher Ausbreitungsverzögerung T_A
 - ▶ Dann gilt: $T_{\text{Ges}} = n (T_S + 2 T_A)$
 - Damit gilt für die Auslastung

$$U = \frac{T_S}{T_{\text{Ges}}} = \frac{T_S}{n(T_S + 2T_A)}$$

- ... in der Regel n nicht fest: Erwartungswert hierfür?

- Annahmen

- p sei Wahrscheinlichkeit, dass eine einzelne Dateneinheit fehlerhaft übertragen wird
- Quittungen sind nie verfälscht

- Wahrscheinlichkeit für genau k Übertragungsversuche?

- $k - 1$ fehlerhafte Übertragungsversuche gefolgt von einer geglückten Übertragung
 - ▶ Wahrscheinlichkeit hierfür: $p^{k-1} (1 - p)$

- $n =$ Erwartungswert [Anzahl Übertragungen]

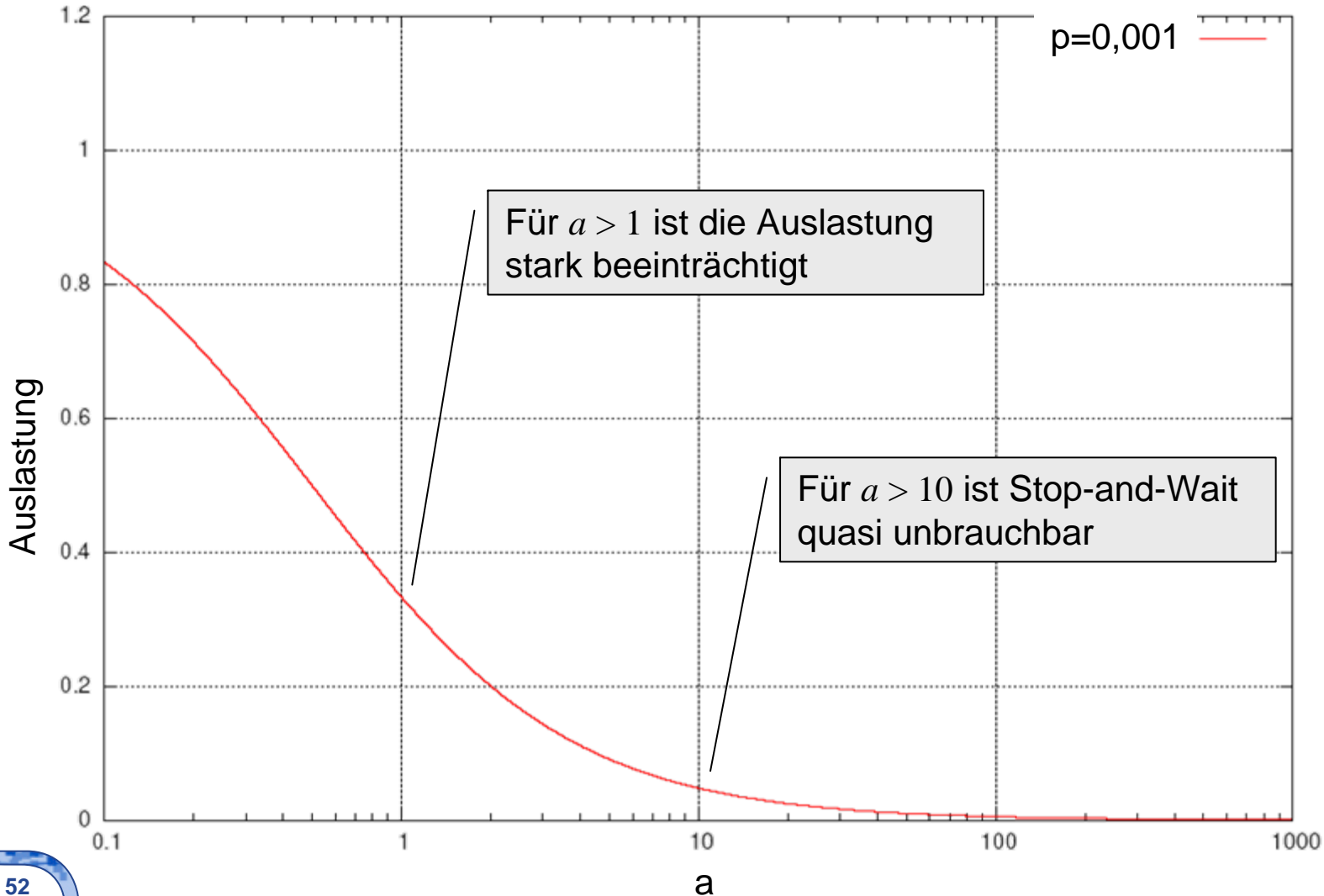
$$n = \sum_{i=1}^{\infty} (i \times p[\text{genau } i \text{ Übertragungen}]) = \sum_{i=1}^{\infty} ip^{i-1}(1-p) = \frac{1}{1-p}$$

- Damit gilt für die Auslastung

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

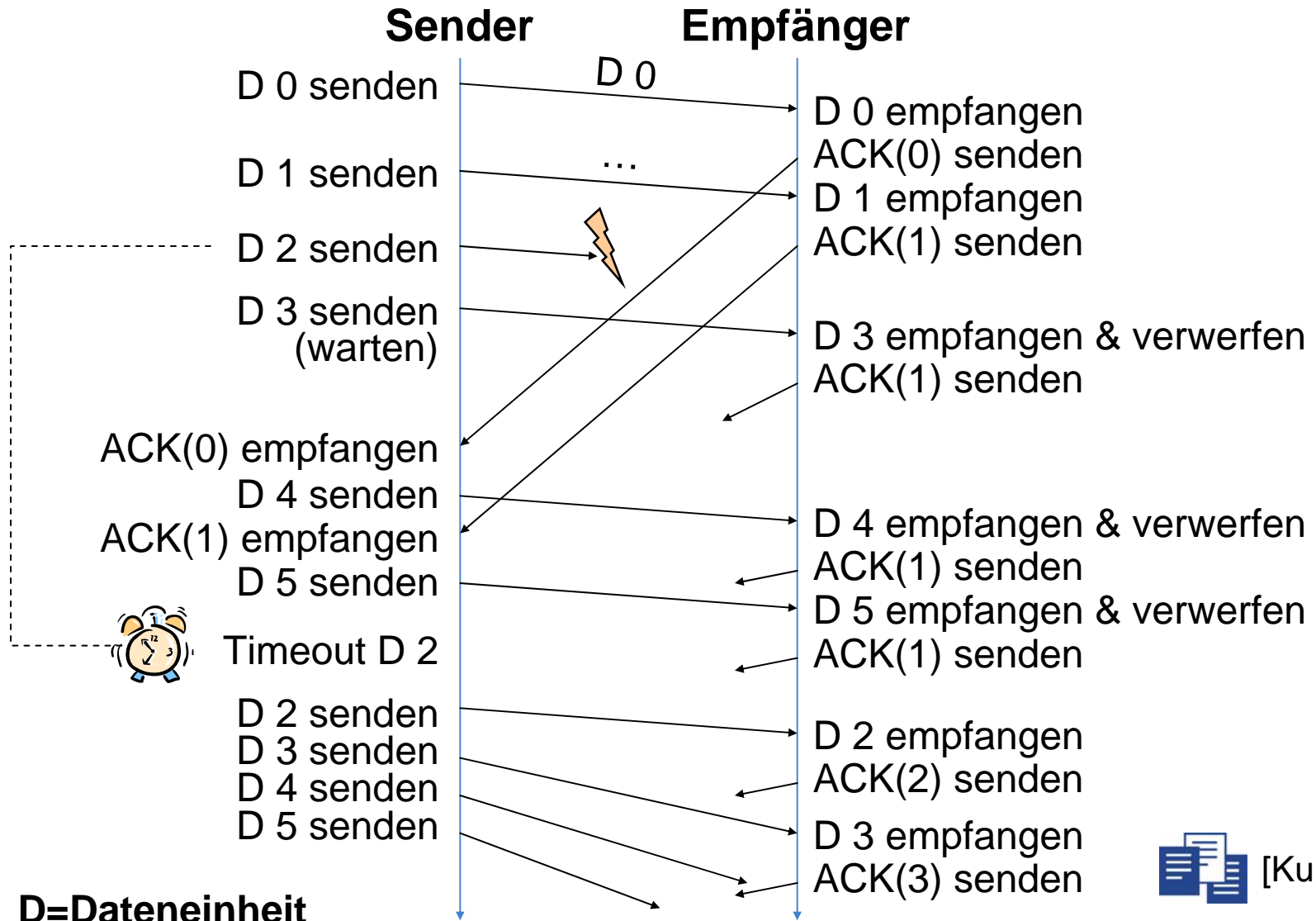
für $-1 < x < 1$ gilt :

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



- Ziel
 - Erhöhung der Leistungsfähigkeit im Vergleich zu Stop-and-Wait
 - ▶ Warten auf Quittung vor dem Senden der nächsten Dateneinheit vermeiden
- Verfahren
 - Der Sender kann mehrere Dateneinheiten senden bis er eine Quittung erhalten muss
 - Die maximale Anzahl der nicht quitierten Dateneinheiten ist begrenzt
 - ▶ typischerweise durch ein so genanntes **Sliding Window**
 - Verschiedene Varianten möglich (nächste Folie)

- Variante 1
 - Empfänger quittiert korrekt empfangene Dateneinheiten wie bei Stop-and-Wait
 - ▶ Die Quittung erfolgt kumulativ, d.h. für mehrere Dateneinheiten auf einmal
 - ▶ Kumulative Sequenznummer gibt an, bis wohin die Daten korrekt empfangen wurden, d.h. es handelt sich um positive Quittungen
 - ▶ Es kann auch jede Dateneinheit quittiert werden
- Variante 2
 - Nicht korrekt empfangene Dateneinheiten werden mit einer negativen Quittung (NACK) bestätigt
 - ▶ Sender wiederholt daraufhin **ab** dieser Sequenznummer alle gesendeten Dateneinheiten
 - ▶ Go-Back N, wobei N die Sequenznummer in der negativen Quittung ist
- Frage
 - Wo ist eine Pufferung der Dateneinheiten erforderlich? Wie viele müssen gepuffert werden?



D=Dateneinheit

- Bemerkung
 - Fenstergröße W begrenzt die Anzahl gesendeter und noch nicht quittierter Dateneinheiten
 - Parameter und Normalisierung wie bei Stop-and-Wait
- Annahme
 - Fehlerfreie Übertragung
- Zwei Fälle müssen unterschieden werden
 - 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

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

- Jetzt: Berücksichtigung von Übertragungsfehlern
- Herleitung ähnlich wie bei Stop-and-Wait, aber
 - Im Fehlerfall werden K Dateneinheiten wiederholt anstelle einer einzigen
 - n = Erwartungswert [Anzahl übertragener Dateneinheiten für eine geglückte Übertragung]
 - ▶ $f(i)$: Komplette Anzahl übertragender Dateneinheiten, falls die ursprünglich gesendete Dateneinheit i mal übertragen werden muss

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

- ▶ Daraus ergibt sich für den Erwartungswert n

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

- ▶ Einsetzen von $f(i)$

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

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

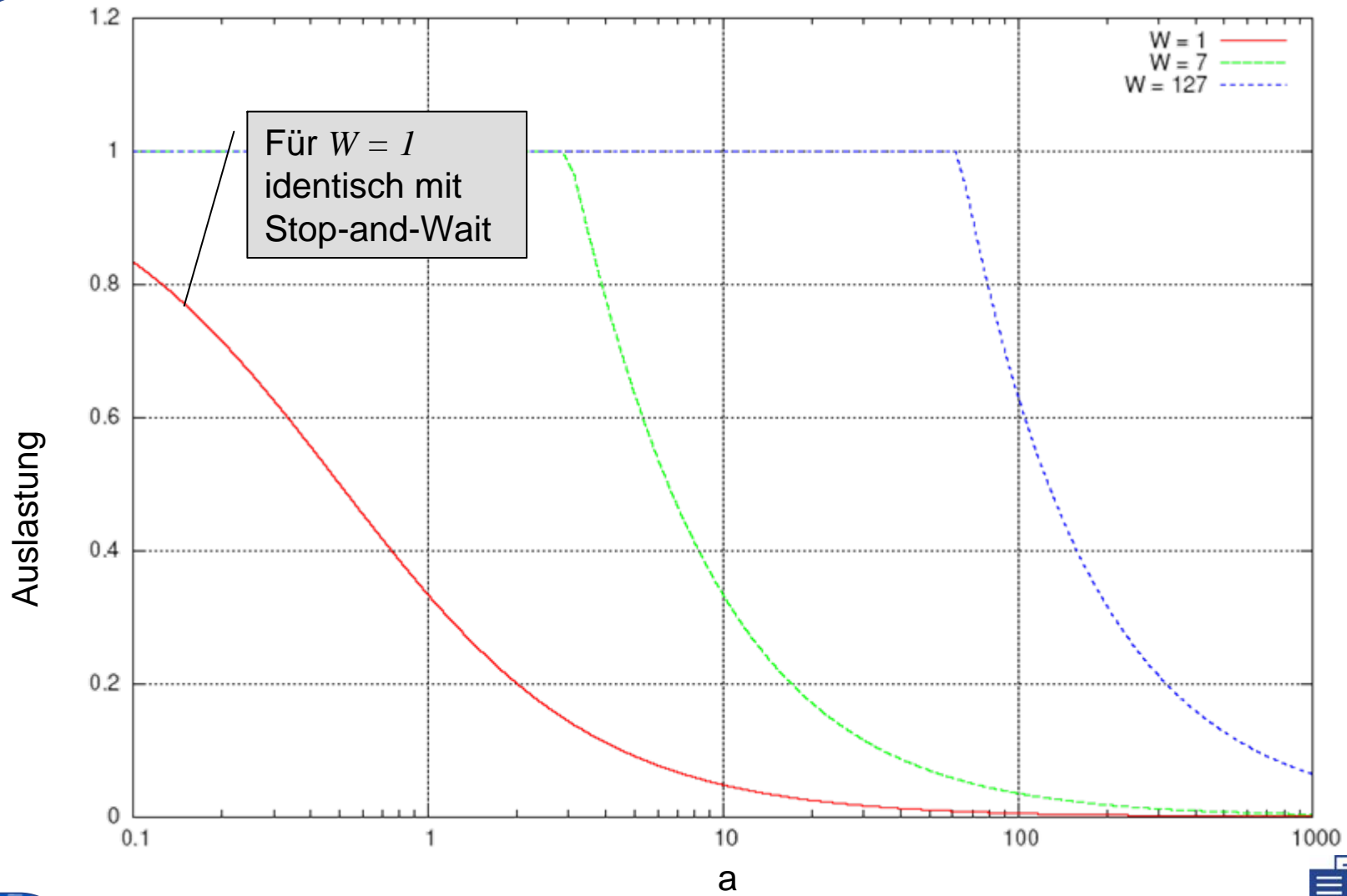
$$n = 1 - K + \frac{K}{1-p} = \frac{1-p + Kp}{1-p}$$

für $-1 < x < 1$ gilt:

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

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

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



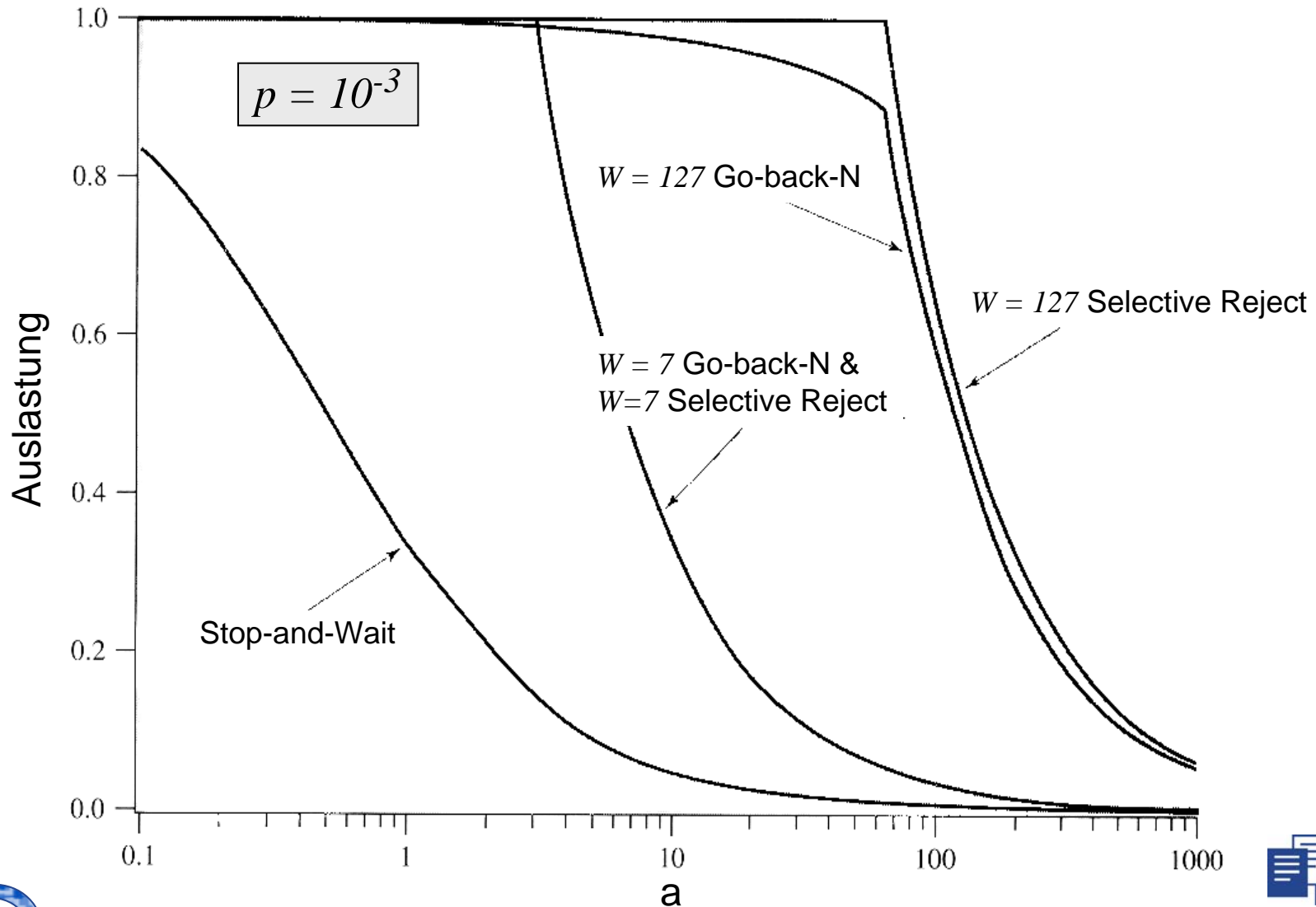
- Ziel
 - Erhöhung der Auslastung im Vergleich zu Stop-and-Wait
 - Reduzierung des Datenaufkommens im Vergleich zu Go-Back-N
- Verfahren
 - Der Sender kann mehrere Dateneinheiten senden, bis er eine Quittung erhalten muss
 - Die maximale Anzahl der nicht quittierten Dateneinheiten ist begrenzt (wie bei Go-Back-N)
 - ▶ Der Empfänger sendet eine negative Quittung, wenn er einen Fehler erkennt
 - ▶ Diese Quittung bezieht sich auf eine einzelne Dateneinheit
 - ▶ Der Sender wiederholt genau die Dateneinheit mit der bei der negativen Quittung angegebenen Sequenznummer
 - ▶ Nur die nicht korrekt empfangenen Dateneinheiten werden vom Sender wiederholt
- Fragen
 - ▶ Wo ist eine Pufferung der Dateneinheiten erforderlich? Wie viele müssen gepuffert werden?
 - ▶ Vor- und Nachteile von Go-Back-N und Selective Reject im Vergleich?

- Herleitung wie bei Stop-and-Wait
 - Es wird immer nur eine Dateneinheit wiederholt

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

- Fallunterscheidung wegen Fenster

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

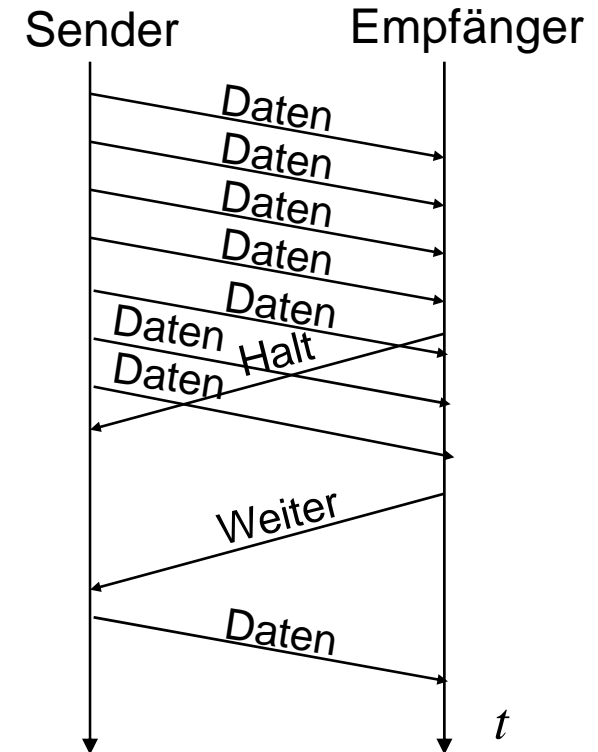


- Problem
 - Wie viele Dateneinheiten dürfen vom Sender hintereinander gesendet werden, ohne dass der Speicher beim Empfänger überläuft (d.h. Verluste von Dateneinheiten entstehen)?
- Anforderungen
 - Einfachheit
 - Möglichst geringe Nutzung von Netzressourcen
 - Fairness
 - Stabilität
- Varianten
 - Closed Loop
 - ▶ Rückkopplung, um zu verhindern, dass Empfänger „überschwemmt“ wird ⇒ Quelle adaptiert ihren Datenstrom entsprechend
 - Open Loop
 - ▶ Beschreibung des Verkehrs mit anschließender Ressourcenreservierung und Überwachung des eingehenden Verkehrs
 - ▶ Hier nicht weiter behandelt

- Varianten

- 1. Generation: Berücksichtigt die Leistungsfähigkeit des Kommunikationspartners
 - ▶ Halt-und-Weiter
 - ▶ Stop-and-Wait
 - ▶ Kombiniert Fehler- und Flusskontrolle
 - ▶ Kreditbasierte Flusskontrolle: statisches Fenster
- 2. Generation: Berücksichtigt auch die Leistungsfähigkeit des Netzes
 - ▶ Messungen
 - ▶ Explizit vs. Implizit
 - ▶ Kontrollvariante
 - ▶ Dynamisches Fenster vs. dynamische Rate
 - ▶ Kontrollpunkt
 - ▶ Ende-zu-Ende vs. Hop-by-hop

- Sehr einfache Methode
 - Meldungen
 - ▶ Halt
 - ▶ Weiter
 - Kann der Empfänger nicht mehr Schritt halten, schickt er dem Sender eine Halt-Meldung
 - Ist ein Empfang wieder möglich, sendet der Empfänger die Weiter-Meldung
- Bewertung
 - Nur auf Vollduplex-Leitungen verwendbar
 - Bei hohen Verzögerungen nicht effektiv
 - Probleme bei Verlust der Halt-Meldung
- Beispiel: Protokoll XON/XOFF
 - Für serielle Leitungen
 - Mit ISO 7-Bit-Alphabetzeichen
 - ▶ XON ist DC₁ (Device Control 1)
 - ▶ XOFF ist DC₃ (Device Control 3)

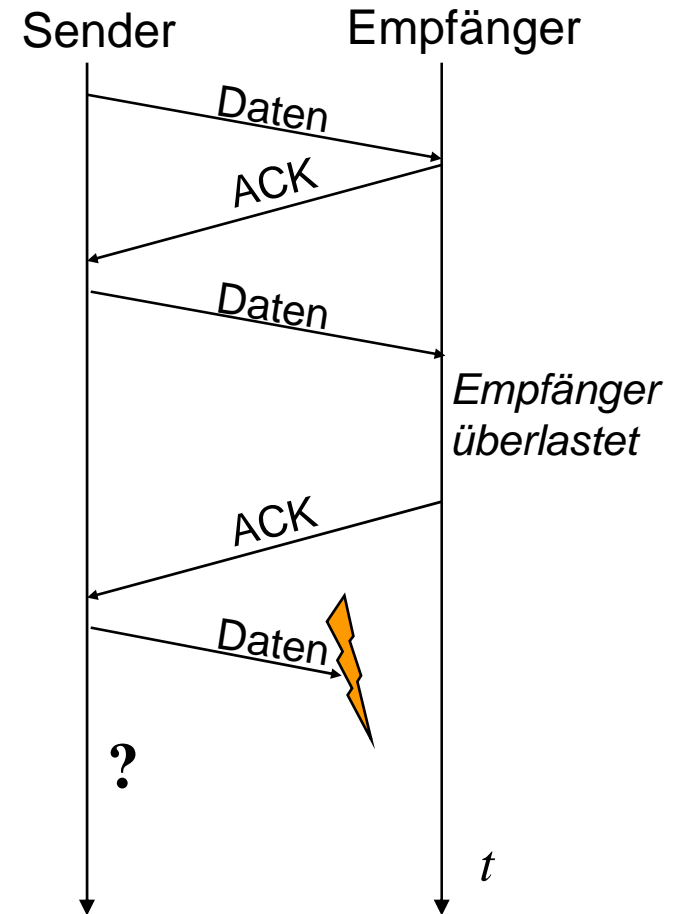


- Funktionsweise

- Durch Zurückhalten der Quittung (z.B. ACK/NACK) kann der Sender gebremst werden
- Das bedeutet, dass ein Verfahren zur Fehlererkennung für die Flusskontrolle mitbenutzt wird
- Beispiel: **Stop-and-Wait**

- Problem

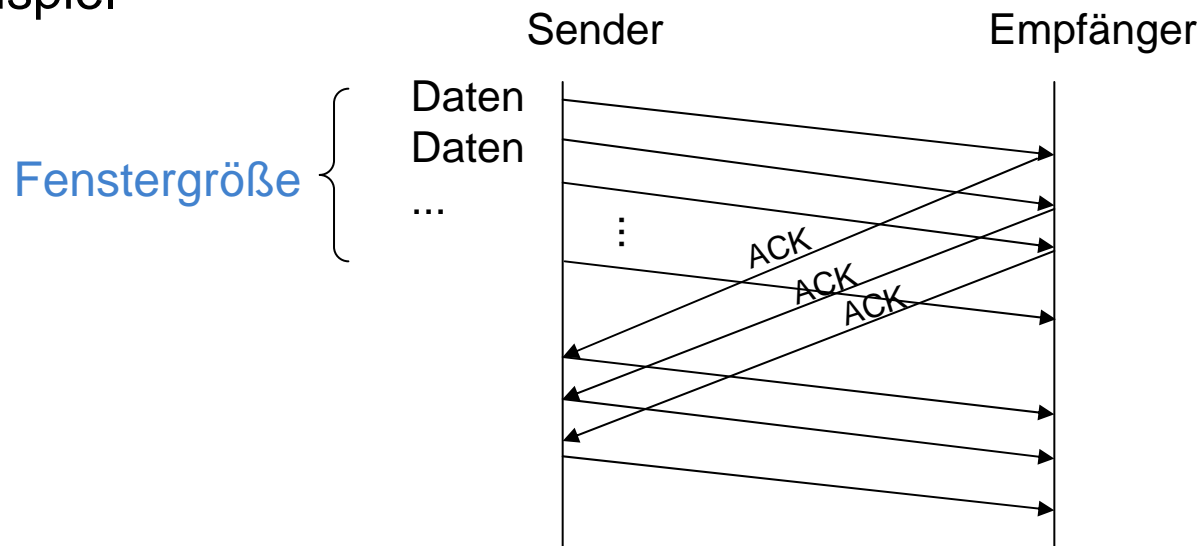
- Der Sender kann nicht mehr unterscheiden,
 - ▶ ob seine Dateneinheit verloren ging, oder
 - ▶ ob der Empfänger die Quittung wegen Überlast zurückgehalten hat



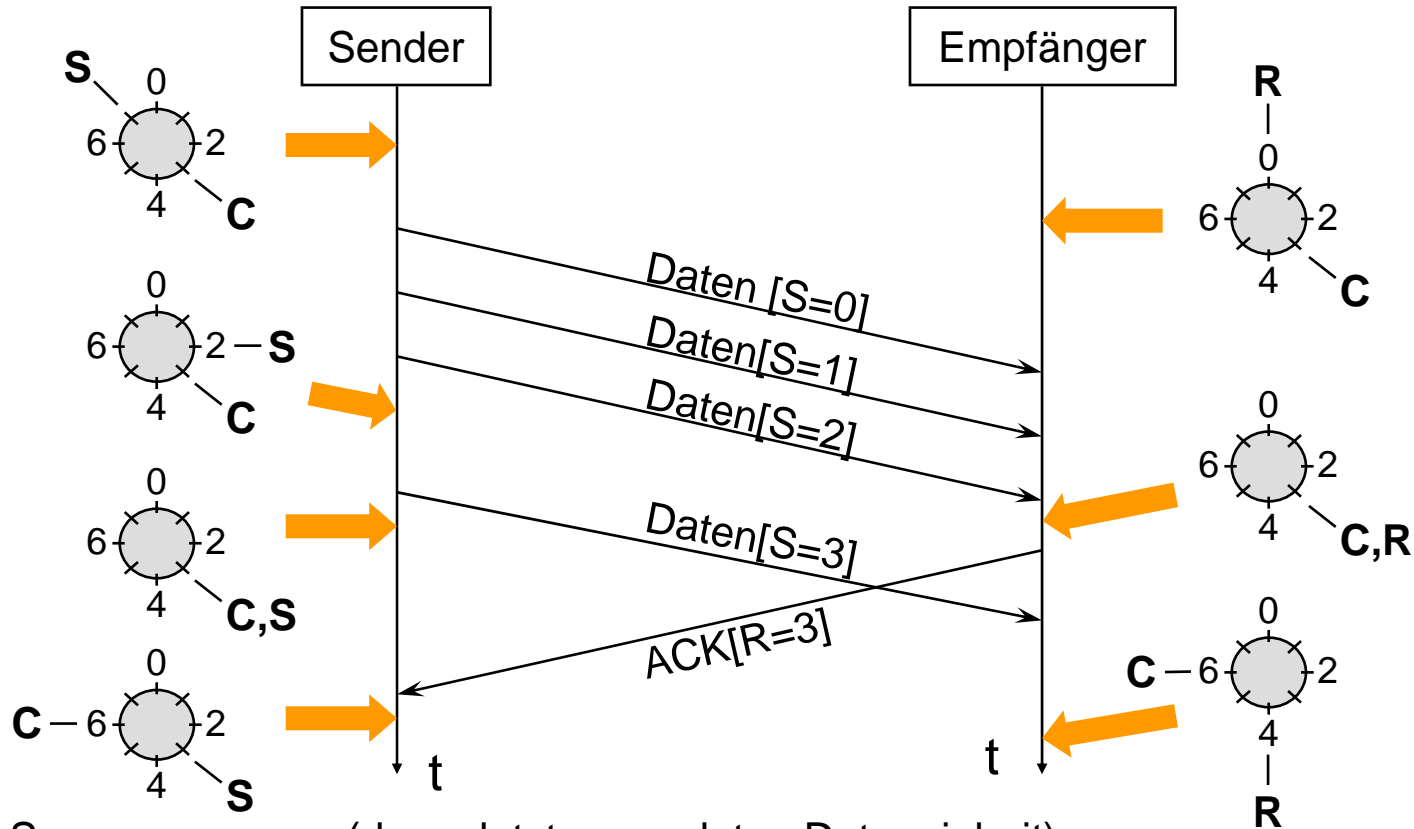
- Prinzip

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

- Beispiel



- 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)

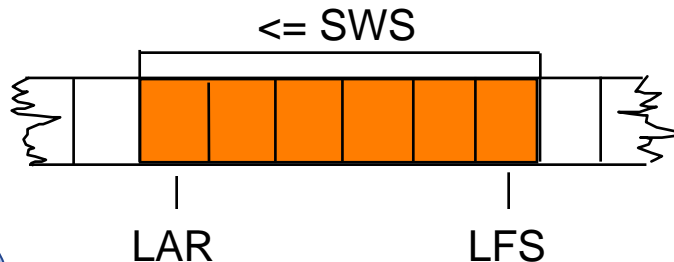
- **Nachteil: Kopplung von Fluss- und Fehlerkontrolle**

- Sender

- **SWS: Send Window Size**
(max. Anzahl ausstehender Dateneinheiten bzw. Bytes)
- **LAR: Last ACK Received**
(Sequenznummer der letzten quittierten Dateneinheit bzw. Bytes)
- **LFS: Last Frame Sent**
(Sequenznummer der letzten gesendeten Dateneinheit bzw. Bytes)

- Invariante

- $LFS - LAR + 1 \leq SWS$

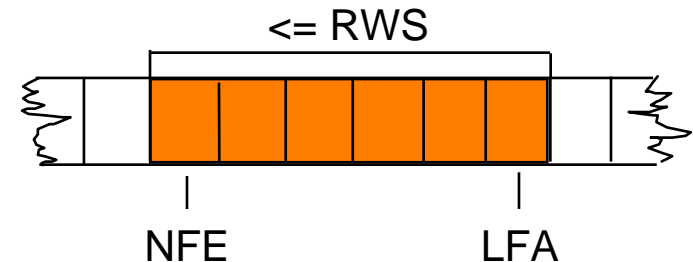


- Empfänger

- **RWS: Receiver Window Size**
(max. Anzahl nicht in Reihenfolge empfangener Dateneinheiten bzw. Bytes)
- **LFA: Last Frame Acceptable**
(Sequenznummer der letzten empfangbaren Dateneinheit bzw. Bytes)
- **NFE: Next Frame Expected**
(Sequenznummer der nächsten in Reihenfolge erwarteten Dateneinheit bzw. Bytes)

- Invariante

- $LFA - NFE + 1 \leq RWS$



http://www.tm.uka.de - Network Simulator - Mozilla Firefox

Speed: ▶ ⏸ ⏩ ⏹ Slow Fast 🚗 📊 📉 📈 Time: 72 x 10 ms

Scenario: Demo_GoBackN_Err Protocol: SlidingWindowGoBackN

<http://www.tm.uka.de/lehre/aktuell/vl/telematik/sim/simulation.html>

Java-Applet: Selective Reject und Flusskontrolle

http://www.tm.uka.de - Network Simulator - Mozilla Firefox

Speed: Slow Fast Time: 133 x 10 ms

Scenario: Demo_SelRequest Protocol: SlidingWindowSelectiveRepeatNack

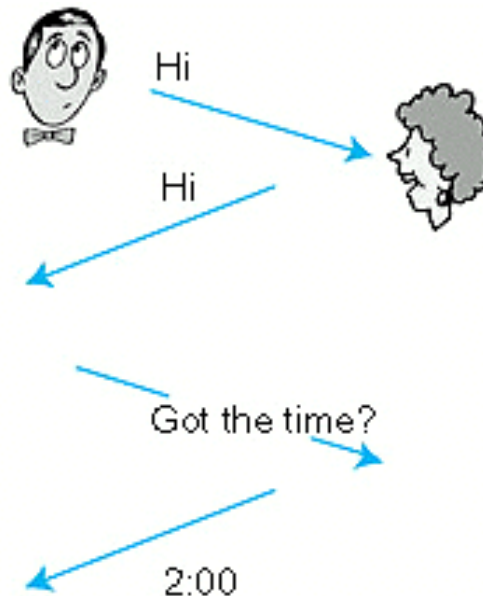
seneca (19) Out → 26 → 25 → 24 → 23 → moloch (23) In

19 ✓ 20 ✓ 21 ✓ 22 ✓

<http://www.tm.uka.de/lehre/aktuell/vl/telematik/sim/simulation.html>

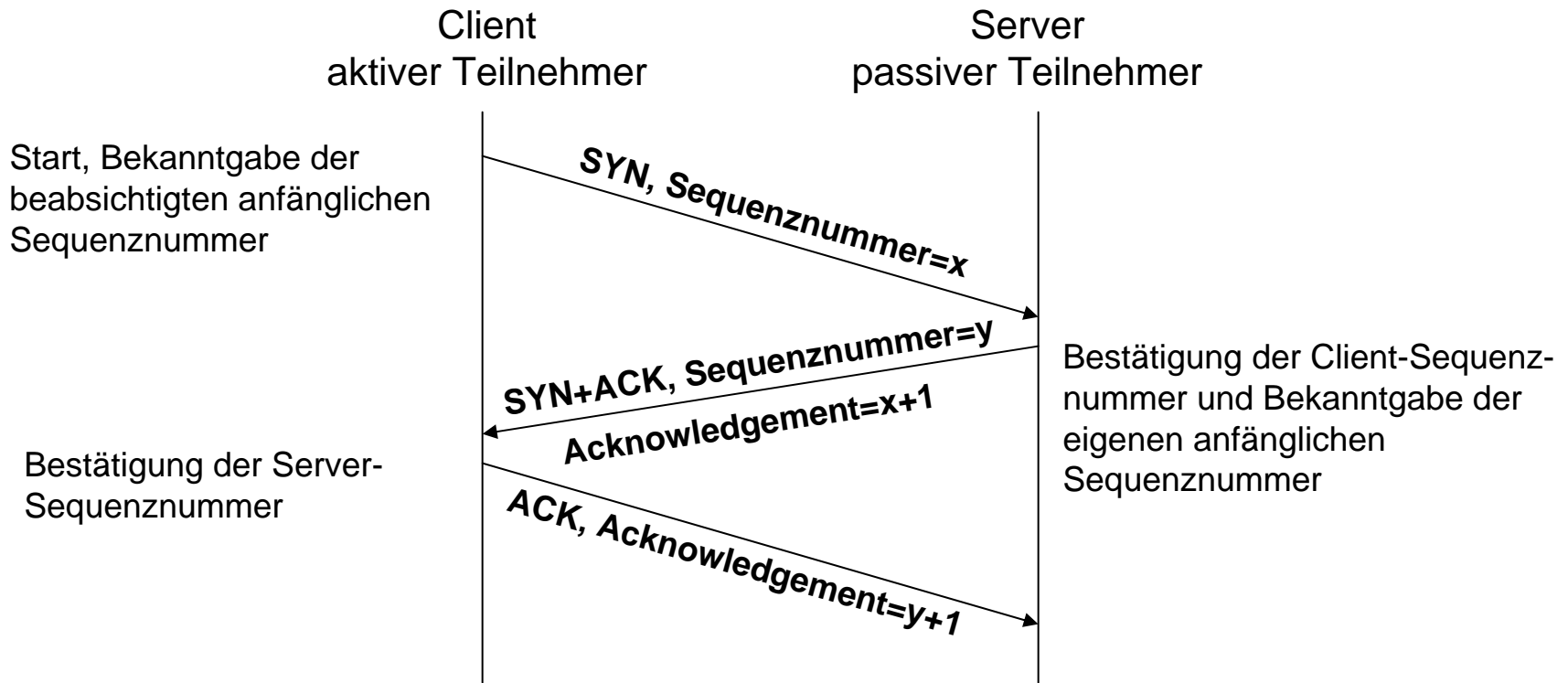
- Verbindungen
 - stellen eine Kommunikationsbeziehung zwischen den Kommunikationspartnern her
 - dienen dem zuverlässigen Austausch von Informationen (Daten)
 - Es wird bei jedem Kommunikationspartner ein sogenannter Verbindungskontext etabliert
 - ▶ Sequenznummern, Fenstergröße etc.

- Beispiel



- Verbindungslose Kommunikation
 - Informationen werden versendet, ohne vorherigen Aufbau einer Verbindung
 - ▶ im Beispiel würde also die Begrüßung wegfallen
 - Vorteil: schnelle Datenversendung möglich
 - Nachteil: keine Möglichkeit der Kontrolle, ob der Kommunikationspartner überhaupt zuhört bzw. zuhören kann
- Verbindungsorientierte Kommunikation
 - dem Informationsaustausch geht Aufbau einer Verbindung voraus
 - Vorteil: Aushandlung von Kommunikationsparametern möglich (3-Wege-Handshake)
 - ▶ sonst: benötigte Fenstergrößen, verwendeter Code, verwendete Fehlerkontrollmechanismen, Sequenznummern ...
 - Nachteile
 - ▶ eigentlicher Datenaustausch verzögert
 - ▶ Overhead der Verbindungsetablierung und -verwaltung kann höher sein als der eigentliche Datenaustausch (kurze Daten)

3-Wege-Handshake



- Das Feld **Acknowledgement** identifiziert jeweils die nächste erwartete Sequenznummer

- Grundlegende Protokollmechanismen
 - Werden uns als Bausteine in vielen Protokollen immer wieder begegnen
- Java Applets: Beispiele für Simulation
 - Animation des Ablaufs
 - Dokumentation des Ablaufs
 - Leistungsbewertung
- Grundlegende Techniken der Leistungsbewertung
 - Viele Vereinfachungen, aber grundlegende Aussagen möglich

Fehlerkontrolle

ARQ-Verfahren:

Stop-and-Wait
Go-Back-N
Selective Reject

Mechanismen:

Sequenznummern
Zeitgeber
Quittungen
Sendewiederholungen

Flusskontrolle

Halt/Weiter
Sliding Window

Verbindungs- management

3-Wege-
Handshake

- 3.1 Es sind Daten im Umfang von 1 MByte zu übertragen. Die Größe einer Dateneinheit betrage 2000 Byte, die Ausbreitungsverzögerung betrage 20 ms. Vergleichen Sie die erzielbare Auslastung von Stop-and-Wait mit Go-Back-N.
- 3.2 Wann versendet der Empfänger jeweils Quittungen?
- 3.3 Welches Ziel hat die Flusskontrolle?
- 3.4 Konstruieren Sie ein Schieberegister, das Sie für die Division durch $x^7 + x^5 + x^4 + x + 1$ verwenden können.
- 3.5 Berechnen Sie den CRC für die Dateneinheit 111011000110101 mit dem Generatorpolynom 110011
- 3.6 Knoten A sendet mit Go-Back-N und Sliding-Window Daten an Knoten B. Die Fenstergröße betrage 4 Bit. Zeichnen sie die Fenster auf beiden Seiten
- Bevor A anfängt Daten zu senden.
 - Nachdem A die Dateneinheiten 0, 1 und 2 gesendet und B 0 und 1 quittiert hat.
 - Nachdem A die Dateneinheiten 3, 4 und 5 gesendet hat und B 4 quittiert hat.
- 3.7 Welche Fehlerarten kennen Sie?
- 3.8 Eine Störung von 10 ms führt bei einer Datenrate von 100 Mbit zu wieviel gestörten Bits?

- [Benv05] Ch. Benvenuti; Understanding Linux Network Internals; O'Reilly, 2005
- [Hals05] F. Halsall; Computer Networking and the Internet; Addison-Wesley, 2005
 - Kapitel 1.4, Anhang C
- [Holz91] G. J. Holzmann; Design and Validation of Computer Protocols; Prentice Hall, 1991
Auch online unter: <http://spinroot.com/spin/Doc/Book91.html>
- [KoBu94] W. P. Kowalk, M. Burke; Rechnernetze; Teubner Verlag, 1994
- [KuRo07] J. F. Kurose, K. W. Ross: Computer Networking: A Top-Down Approach, Addison-Wesley, 2007
- [Stal06] W. Stallings; Data and Computer Communications, Prentice Hall, 2006
 - Kapitel 7