

Datenkommunikation und Sicherheit

Kapitel 3: Sicherungsschicht

Klaus Wehrle

Communication and Distributed Systems

Chair of Computer Science 4

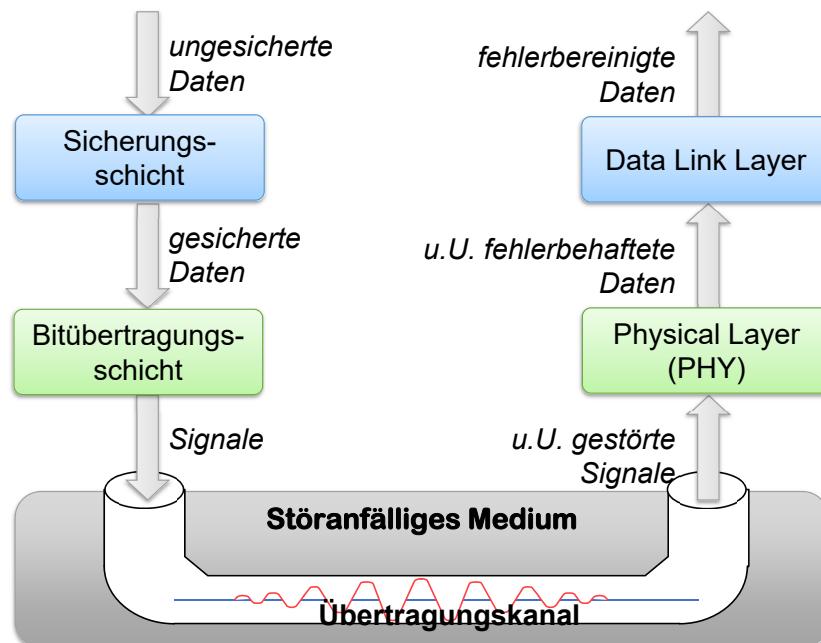
RWTH Aachen University

<http://www.comsys.rwth-aachen.de>



III-1

Bitübertragungsschicht und Sicherungsschicht



Zur Erinnerung: die beiden untersten Schichten sorgen gemeinsam für eine fehlerfreie Datenübertragung zwischen benachbarten Rechnern. „Benachbart“ heißt, dass die kommunizierenden Rechner durch ein physikalisches Medium direkt miteinander verbunden sind.

Die Bitübertragungsschicht realisiert einen nachrichtentechnischen Kanal zur Übertragung von Bitfolgen, allerdings können bei der Übertragung Fehler auftreten.

Die Sicherungsschicht erweitert einen nachrichtentechnischen Kanal zum eigenständigen, abstrakten Medium „gesicherter Kanal“. „Gesichert“ heißt hierbei: fehlerfrei. Notwendig dazu sind:

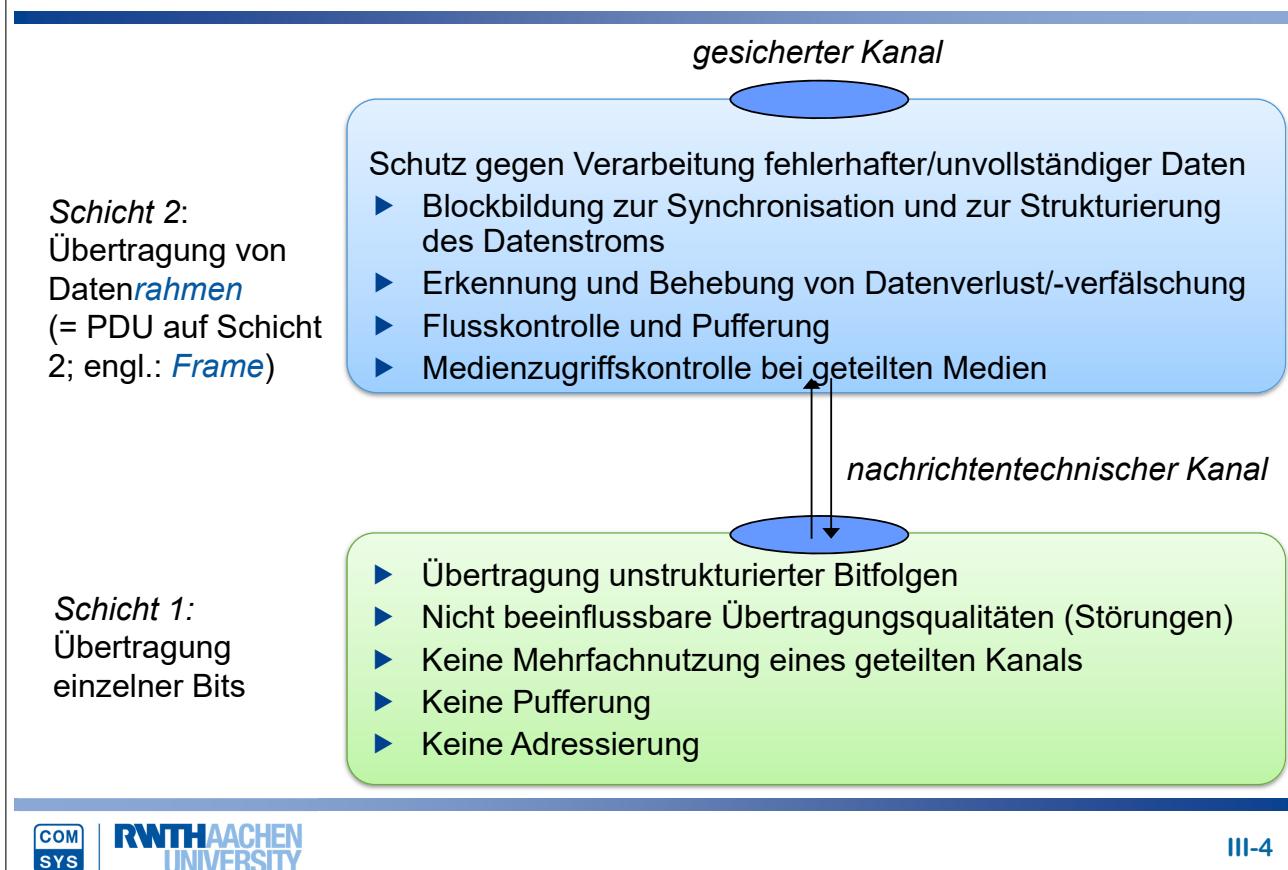
- Strukturierung des Bitstromes in Rahmen (Frames) als Basiseinheit der Übertragung
- Fehlererkennung und –behandlung (Quittungen und Übertragungswiederholungen, siehe Alternating Bit Protocol)
- Flusskontrolle (Vermeidung der Überlastung/Pufferüberlauf des Empfängers)
- Regelung des Zugriffs auf ein Medium, welches gemeinsam von mehreren Stationen verwendet wird.

Themenübersicht

• Datenkommunikation und Sicherheit

- ▶ Einführung, Begriffe und allgemeine Grundlagen
- ▶ Technische und nachrichtentechnische Grundlagen: Medien, Signale, Bandbreite, Leitungscodes und Modulation, Multiplexing
- ▶ Lokale Netze: Strukturierung des Datenstroms, Fehlererkennung/-behebung, Flusssteuerung, Medienzugriff, Ethernet
- ▶ Internet und Internet-Protokolle:
 - Vermittlungsschicht: IP, Routing
 - Transportschicht: TCP
- ▶ Grundlagen der Sicherheit in/von Kommunikationsnetzen
 - Grundlagen: Verschlüsselung, Authentifizierung, Integrität
 - Sichere Internet-Protokolle

Aufgaben der Sicherungsschicht



Die Sicherungsschicht hat die Aufgabe, höheren Schichten einen Übertragungskanal zur Verfügung zu stellen, über den Daten vollständig und fehlerfrei (= *gesichert*) übertragen werden. Der Sicherungsschicht steht aber nur ein ungesicherter Schicht-1-Dienst zur Verfügung, der unstrukturierte Bitfolgen ohne Kontrollmechanismen überträgt (*nachrichtentechnischer Kanal*).

Die Sicherungsschicht muss daher einen Schutz gegen Missinterpretation empfangener Bitfolgen bieten. Dazu können entweder einzelne Bits oder Blöcke von Bits gesichert werden. Im Allgemeinen ist die Sicherung von Blöcken effizienter und daher wird nur dieses Vorgehen behandelt. Der Empfänger kann durch Blockbildung innerhalb des empfangenen Datenstroms Muster identifizieren und Datenblöcke daraufhin als korrekt oder falsch klassifizieren. Die effiziente Erkennung und Behandlung von Fehlern setzt daher zwingend voraus, dass die Daten vor der Übertragung in Blöcke strukturiert werden.

Um verschiedene Arten von Fehlern oder Problemen zu erkennen und zu behandeln, hat die Sicherungsschicht die folgenden Aufgaben (die je nach Protokoll nicht unbedingt alle implementiert werden müssen):

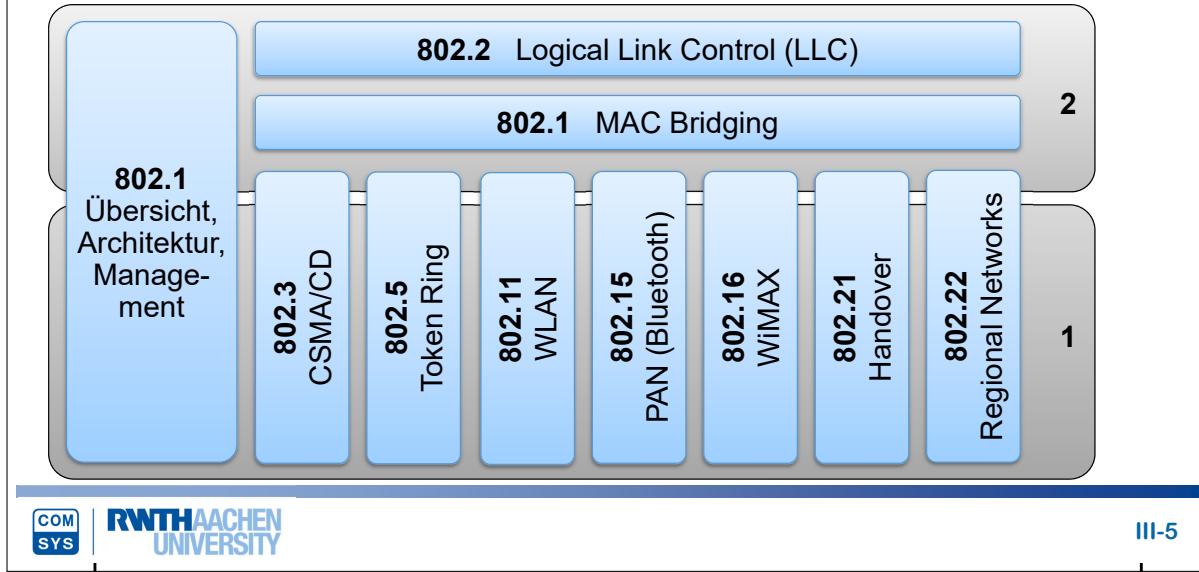
- Synchronisation, Strukturierung des Datenstroms/Erkennen von Blockgrenzen: Ganz allgemein müssen Zeichen bzw. Bitfolgen innerhalb eines unstrukturierten Bitstroms identifiziert werden können. Meist wird der Bitstrom dazu in sogenannte *Rahmen* unterteilt, um zur Bewältigung der restlichen Aufgaben eine fest definierte Struktur verwenden zu können. Die Anfänge eines Rahmens müssen erkannt werden, vor allem auch nach Übertragung fehlerhafter (bspw. abgeschnittener) Rahmen.
- Codetransparenz: Es muss sichergestellt werden, dass beliebige Zeichen- bzw. Bitfolgen übertragen werden können. Ob hierbei Probleme auftreten können, hängt von der Art der gewählten Strukturierung/Rahmenbildung ab und kann daher auch als Unterpunkt des vorherigen Punktes angesehen werden.
- Fehlererkennung und -behandlung: Es sollen sowohl Fehler bei der Datenübertragung als auch Fehler im Protokollablauf erkannt und behoben werden. Fehler bei der Datenübertragung können durch Suche nach Mustern in den empfangenen Blöcken erfolgen – die Behandlung eines Fehlers erfolgt z.B. einfach durch eine erneute Übertragung wie beim Alternating-Bit-Protokoll. Um Fehler im Protokollablauf erkennen zu können, müssen Protokolle vollständig spezifiziert werden und es muss auch definiert werden, wie in welcher Fehlersituation zu verfahren ist.
- Flusskontrolle und Pufferung: Überlastsituationen müssen verhindert werden – es soll nicht vorkommen, dass ein Empfänger überlastet wird, da er die empfangenen Daten nicht schnell genug verarbeiten kann und sie verwerfen

muss. Dazu müssen Daten gepuffert werden, aber es muss auch möglich sein, der Gegenstelle zu signalisieren, dass keine weiteren Daten mehr gepuffert werden können, wenn der Puffer voll ist.

- Koordinierter Medienzugriff: Vergabe von Senderechten/Medienzuteilung bei geteiltem Übertragungsmedium. Dies kann durch die bereits im letzten Kapitel vorgestellten Multiplexverfahren erreicht werden, doch sind diese i.A. zu unflexibel.
- Verfahren zur Steuerung der Übermittlung: z.B. Initialisierung, Terminierung, Identifikation, Halbduplex-/Vollduplexbetrieb.

LAN/MAN: Standardfamilie IEEE 802

- **IEEE: Amerikanischer Verband der Elektroingenieure**
 - ▶ Im Bereich von LAN/MAN Netzen sehr verbreitete Standards
 - ▶ Standardfamilie IEEE 802 unterteilt Schicht 2 in *Medium Access Control* (MAC) und *Logical Link Control* (LLC)



Die IEEE wurde bereits als die zentrale Standardisierungsorganisation im Bereich der lokalen Netze erwähnt.

Laut IEEE wird Schicht 2 in zwei Teilfunktionalitäten gegliedert. Der obere Teil ist unabhängig von dem konkreten Medium und stellt der nächsthöheren Schicht eine einheitliche Schnittstelle zum Zugriff auf das Netz zur Verfügung. Hier sind all die Funktionen definiert, die unabhängig vom konkreten Netzaufbau sind: Flusssteuerung, Quittierungsmechanismus, ... – zusammengefasst als 802.2, Logical Link Control.

Der untere Teil umfasst hardwarenähere Aspekte. Dies umfasst die Festlegung von Übertragungsmedien und Netztopologien – und damit zusammenhängenden Aufgaben. Vom Medium hängt es ab, welches Fehlererkennungs- oder -korrekturverfahren eingesetzt wird. (Bei Glasfaser treten kaum Bitfehler auf, weshalb eine einfache CRC oder gar ein Parity-Bit ausreicht, bei Funkübertragung sind Bitfehler die Regel, weshalb meistens CRC zusammen mit Fehlerkorrekturverfahren eingesetzt wird.) Vom Medium hängt es ab, welche Arten des Medienzugriffs möglich sind. (CSMA/CD beispielsweise, welches bei Ethernet über Kupfer- oder Glasfaserkabel eingesetzt wird, funktioniert nicht bei drahtloser Kommunikation – WLAN definiert deshalb eine Variante namens CSMA/CA.)

Sicherungsschicht – Überblick

Layer 2b:
Logical Link Control

1. Protocol Data Units

- ▶ Rahmenbildung
- ▶ Blockerkennung
 - Längenfeld
 - Layer-1-Codierung
 - Code-Violation
 - Character-Stuffing
 - Bit-Stuffing

2. Fehlererkennung und -behandlung

- ▶ Prüfsummen
 - Paritätsüberprüfung, CRC
- ▶ Proaktive Fehlerbehandlung
 - Hinzufügen von Redundanz
 - Hamming-Code
- ▶ Reaktive Fehlerbehandlung
 - Automatic Repeat Request (ARQ)

3. Flusskontrolle

- ▶ Stop and Wait
- ▶ Sliding Window

Layer 2a: Medium
Access Control

4. Medium Access Control

- ▶ LAN-Topologien
- ▶ Medium access control
 - Geregelter Zugriff (Polling, Token Ring)
 - Konkurrierender Zugriff (Aloha, CSMA/CD)
- ▶ Ethernet



Kapitel 3: Sicherungsschicht

- **Rahmenbildung**

- ▶ Blockbildung, Codetransparenz

- **Fehlererkennung/-behandlung und Flusskontrolle**

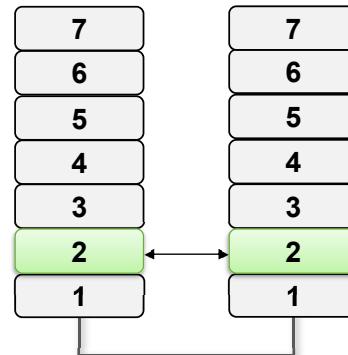
- ▶ Fehlererkennende Codes (CRC)
- ▶ Proaktive und reaktive Fehlerbehebung (FEC und ARQ)
- ▶ Flusskontrolle durch Sliding Window
- ▶ Beispielprotokoll: HDLC

- **Medienzugriff**

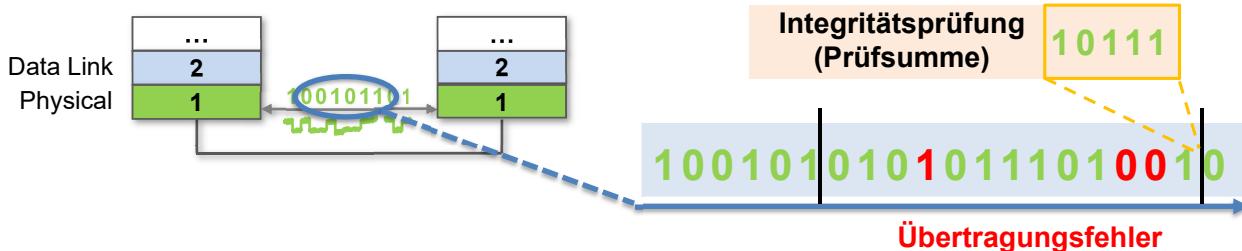
- ▶ Topologien
- ▶ Token Passing, CSMA/CD, CSMA/CA

- **Standards für lokale Netze**

- ▶ Ethernet



Rahmenbildung



- **Schicht 1: Unstrukturierter Bitstrom**

- Übertragungsfehler (gekippte Bits) können nicht erkannt werden

- **Benötigt:**

- *Integritätsprüfung* zur Erkennung von Fehlern → zusätzliche, redundante Daten (= *Prüfsumme*)
- *Struktur im Bitstrom* (= *Rahmen*)
 - um festzulegen, welche Bits von der Prüfsumme abgedeckt werden
 - um die Prüfsumme von den Datenbits abgrenzen zu können

Auf Schicht 2 wird nicht mehr die Übertragung einzelner Signale (einzelne Bits bzw. Bitkombinationen bei Verwendung mehrwertiger Signale) betrachtet, sondern die Übertragung von Bitfolgen fester oder variabler Länge. Dies ist z.B. notwendig, um Übertragungsfehler zu erkennen, die durch Störeinflüsse auf Schicht 1 zustande gekommen sind. Anhand eines einzelnen empfangenen Bits wird man nicht entscheiden können, ob es korrekt ist – dies wird erst möglich, wenn man Folgen von Bits betrachtet.

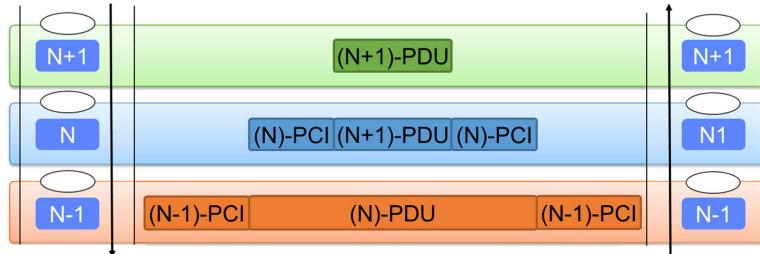
Aber diese Fehlererkennung ist nicht der einzige Sinn, den die Rahmenbildung hat. Während auf Schicht 1 zwar bestimmte Aufgaben umgesetzt werden müssen, wurden im ISO/OSI-Referenzmodell keine Protokolle für diese Schicht definiert; erst auf Schicht 2 werden Protokolle eingesetzt: bei der Betrachtung einzelner Bits können keine Protokollkontrollinformationen übertragen werden, da der Empfänger aufgrund eines einzelnen Bits nicht entscheiden kann, ob Nutzdaten oder Kontrollinformationen vorliegen. Bei Codes wie 4B/5B liegen einzelne Bitkombinationen für Kontrollinformationen vor, aber ihr Umfang ist relativ eingeschränkt. Erst wenn man die übertragenen Daten mit Struktur versieht, kann man einfach erkennen, was Kontroll- und was Nutzdaten sind.

Rahmenbildung bezeichnet also den Vorgang, einen zu übertragenden Bitstrom in Blöcke aufzuteilen, die einer festen Struktur folgen, um die für Protokolle nötigen Kontrollinformationen hinzufügen und die zur gesicherten Übertragung notwendigen Mechanismen (Prüfsumme) implementieren zu können.

Zur Erinnerung: PDUs

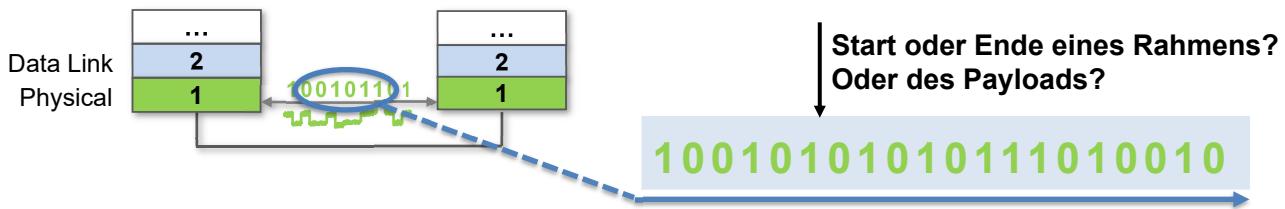
- Schicht 2 führt PDUs ein

- ▶ Nutzdaten (*Payload*) + Kontrollinformationen (zur Koordination der Protokolloperationen)
 - Hier: Hinzufügen einer Prüfsumme zur Fehlererkennung
 - Andere Kontrollinformationen:
 - Adressen, Sequenznummern, ...
 - Üblicherweise als *Header* den Daten vorangestellt
 - Auf Schicht 2 auch als *Trailer* an die Daten angehangen



- ▶ Herausforderung: Erkennung von *Beginn und Ende eines Rahmens*

Erkennung von Rahmenbeginn/-ende



- Wie können Start/Ende eines Rahmens erkannt werden?

- ▶ Payload darf nicht mit Begrenzungen durcheinandergebracht werden (Codetransparenz)

- *Implizite Codetransparenz*: Payload wird nicht modifiziert

- (1) Verwendung einer Längenangabe
 - (2) Spezielle Kontrollzeichen auf Schicht 1 oder 2
 - (3) Codeverletzung auf Schicht 1

- *Explizite Codetransparenz*: Payload muss modifiziert werden

- (4) Character-Stuffing
 - (5) Bit-Stuffing

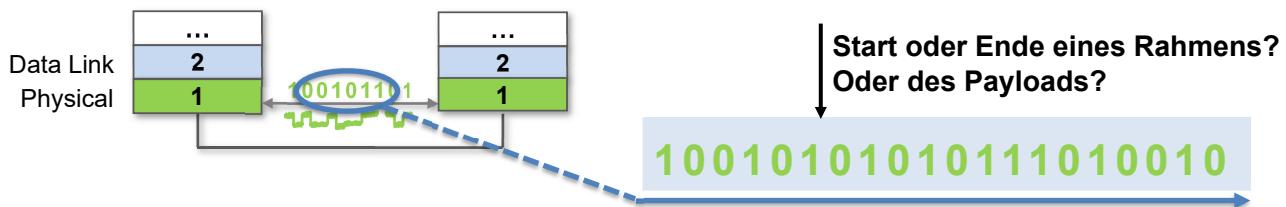
Statt den Begriff des Rahmens zu nutzen, verwenden wir im Folgenden oft den Begriff des Blocks – um darzustellen, dass uns eine weitere Rahmenstruktur noch nicht interessiert, sondern es zunächst nur um Kennzeichnung von Anfang und Ende einer zusammengehörigen Bitfolge geht. Ob diese nur aus Daten (Payload) oder auch aus weiteren Kontrollinformationen (Header) besteht, ist zunächst uninteressant.

Wir betrachten in dieser Vorlesung nur die synchrone Übertragung. Bei der synchronen Übertragung wird eine Folge von Zeichen/Bits innerhalb eines Blocks (bzw. Rahmens) übertragen. Anfang und Ende des Blocks sind durch bestimmte Zeichen-/Bit-Muster eindeutig festgelegt und können somit vom Empfänger erkannt werden. Eine Synchronisation ist somit nur am Anfang eines Blocks notwendig (dies wird dadurch erreicht, dass dem Blockstartmuster einige Zeichen/Bits (Präambel) vorangestellt werden, die nur der Synchronisierung dienen).

Als Sonderfall kann hierbei die Vorgehensweise betrachtet werden, nur den Anfang eines Blockes eindeutig zu kennzeichnen und die Länge des Nutzdatenfeldes durch ein zusätzliches, sich an das Blockstartmuster anschließendes Längenfeld zu spezifizieren.

Alternativ kann man auch den Leistungscode von Schicht 1 zur Blockbildung „missbrauchen“.

Codetransparenz: Längenangabe

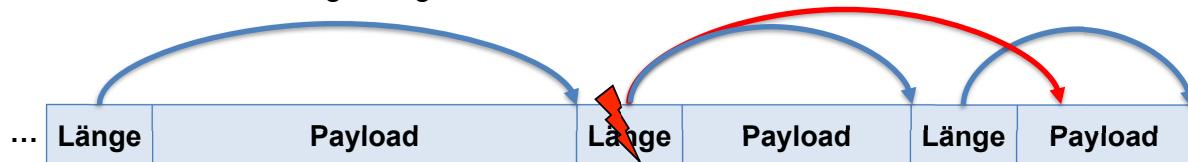


- Wie können Start/Ende eines Rahmens erkannt werden?

- ▶ Payload darf nicht mit Begrenzungen durcheinandergebracht werden (Codetransparenz)

- (1) Verwendung einer Längenangabe

- Einfach...
 - ... aber Probleme bei Verlust der Synchronisation, z.B. durch Bitfehler in Längenangabe



Bei dieser einfachen Methode wird eine weitere Kontrollinformation vor den Daten eingefügt, welche die Anzahl der Bits oder Bytes im Block angibt. Sobald die Implementierung der Sicherungsschicht im Empfänger dieses Feld liest, weiß sie, wie lang der Rahmen ist. Mehr Kontrollinformationen sind nicht nötig - was ein klarer Vorteil zu anderen Vorgehensweisen ist.

Bei dieser Methode kann es jedoch zu Problemen kommen, falls das Längenfeld während der Übertragung verfälscht wird. Der Empfänger kann zwar typischerweise erkennen, dass er einen verfälschten Block erhalten hat, kann aber den Anfang des nächsten Blocks nicht mehr korrekt erkennen.

Verwendung spezieller Kontrollzeichen

(2) Spezielle Kontrollzeichen auf Schicht 2

- ▶ Definition spezieller Zeichen im verwendeten Alphabet
- ▶ Beispiel: ASCII für text-basierte Codierung

← 02 72 69 76 76 79 32 87 79 82 76 68 03 02 ... 03 00 00 02 ... 03

STX (=02): Start of text
ETX (=03): End of text

- ▶ Markieren Blockbegrenzung
- ▶ Können nicht im Payload vorkommen

		000	001	010	011	100	101	110	111
0000	0	NUL (DLE)	SP	0	@	P	`	p	
0001	1	SOH (STX)	DC1	!	1	A	Q	a	q
0010	2	STX (ETX)	DC2	"	2	B	R	b	r
0011	3	EOT (ENQ)	DC3	#	3	C	S	c	s
0100	4	EOT (ACK)	DC4	\$	4	D	T	d	t
0101	5	ENQ (SYN)	(NAK)	%	5	E	U	e	u
0110	6	ACK (SYN)	(NAK)	&	6	F	V	f	v
0111	7	BEL (ETB)	(ETB)	-	7	G	W	g	w
1000	8	BS (HT)	CAN	(8	H	X	h	x
1001	9	HT (EM)	EM)	9	I	Y	i	y
1010	10	LF (SUB)	SUB	*	:	J	Z	j	z
⋮									

Bei zeichenorientierter Übertragung wird ein Datenstrom als Abfolge von Zeichen angesehen. Einige der Zeichen (hier grau/gelb hervorgehoben) werden als Steuerzeichen zur Regelung des Kommunikationsablaufs verwendet, z.B. STX (START OF TEXT) als Startmuster einer Übertragung, ETX (END OF TEXT) als Endmuster der Übertragung.

Verwendung spezieller Kontrollzeichen

(2) Spezielle Kontrollzeichen auf Schicht 1

- ▶ Definition spezieller Zeichen im verwendeten Code
- ▶ Beispiel: 4B/5B-Code

Layer 2: --- (Start Delimiter) 0001 0010 0011 (End) (Start Delimiter) ...

Layer 1: 00000 11000 10001 01001 10100 10100 01101 11000 10001 ...

Symbol	Channel code	Payload value
0	11110	0000
1	01001	0001
2	10100	0010
3	10101	0011
4	01010	0100
5	01011	0101
6	01110	0110
7	01111	0111
8	10010	1000
9	10011	1001
A	10110	1010
B	10111	1011
C	11010	1100
D	11011	1101
E	11100	1110
F	11101	1111

Symbol	Channel code	Protocol command
Q	00000	Quiet
I	11111	Idle
H	00100	Halt
J	11000	1st of sequential SD-Pair
K	10001	2nd of sequential SD-Pair
T	01101	End
R	00111	
S	11001	

JK: Start of Frame
TT: End of Frame
▶ Markieren Blockbegrenzung
▶ Können nicht im Payload vorkommen

Alternative: je nach Leitungscode existieren solche speziellen Zeichen bereits auf Schicht 1.

Im vorherigen Kapitel wurde bereits der 4B/5B-Code vorgestellt, der entsprechende Kontrollzeichen mit sich bringt.

Nutzung weiterer Schicht-1-Informationen

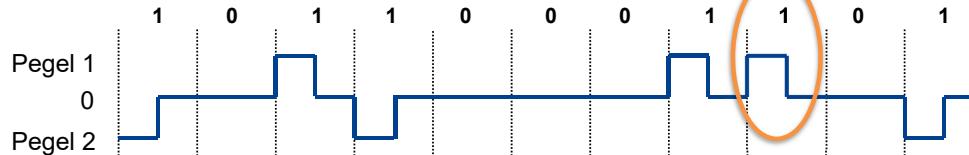
(3) Weitere nützliche Informationen auf Schicht 1

- ▶ Ruhepausen zwischen Blöcken
 - Beispiel: *Inter-Frame Gaps* bei Ethernet



▶ Code-Verletzungen

- Verletze Codierregeln auf Schicht 1, um Markierungen vorzunehmen
- Beispiel: Bipolarer RZ-Code



Bietet der Leitungscode auf Schicht 1 keine Steuerzeichen, kann man eventuell andere Informationen nutzen.

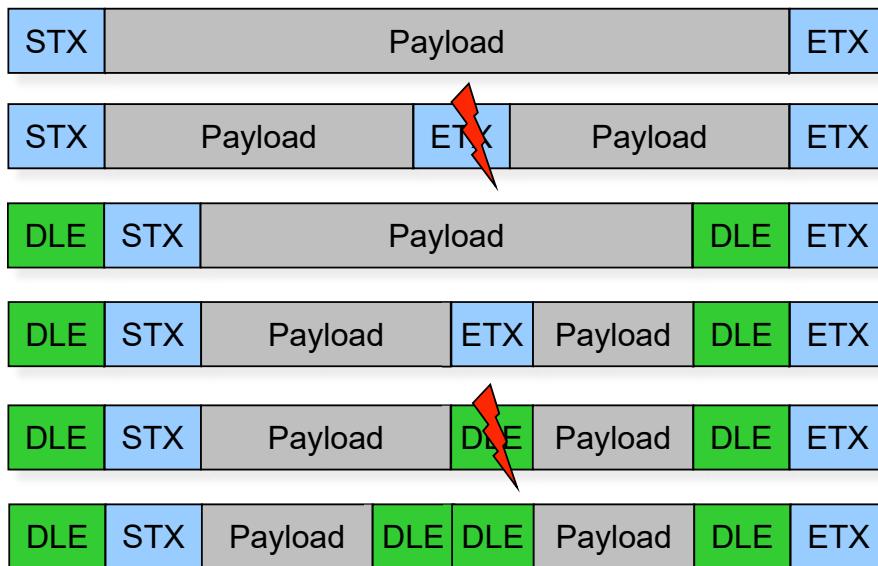
Z.B. kann man Ruhepausen zwischen je zwei übertragenen Blöcken lassen. Das Ende eines Blocks erkennt man dann dadurch, dass eine Ruhepause folgt; ebenso lässt sich der Beginn des nächsten Blocks einfach nach Ende der Ruhepause erkennen. Diese Technik wurde bereits oben zusammen mit der Längenangabe genannt. Diese einfache Technik der Blockbegrenzung wird z.B. bei Ethernet eingesetzt.

Eine andere Möglichkeit sind explizite Code-Verletzungen (Code Violation). Hier im Beispiel dargestellt ist ein vorher nicht behandelter Leitungscode, eine Variante des RZ-Codes. Bei dieser wird eine logischen Null durch eine Ruhepause dargestellt (Halten des Null-Pegels für eine definierte Zeit), die logische 1 abwechselnd durch einen Wechsel von hohem und niedrigem Pegel zum Null-Pegel. Verwendet man nun z.B. zwei Mal in Folge einen Wechsel vom hohen Pegel zum Null-Pegel, ist dies nicht im Code definiert – der Code wird verletzt. Der Sender kann diese Verletzung absichtlich hervorrufen, um einen Blockbeginn zu kennzeichnen.

Character-Stuffing

(4) Markierung von Kontrollsequenzen

- DLE (= Data Link Escape) markiert das folgende Zeichen als Kontrollinformation



Blöcke werden in Variante (2) durch die Zeichen STX und ETX begrenzt, welche dem Empfänger mitteilen, welche empfangenen Bits die Dateneinheit darstellen. Oben wurde gesagt, dass diese Zeichen nicht im Block vorkommen können.

Im Payload müssen aber beliebige Bit-/Bytefolgen übertragbar sein – und die Bitfolgen, die STX/ETX darstellen, können durchaus auch Teil der zu übertragenen Daten sein. In der zweiten Zeile dargestellt ist so ein Problemfall: das ETX-Zeichen ist zufällig im Nutzdatenteil enthalten. In diesem Fall wird der Empfänger es als die hintere Rahmenbegrenzung betrachten und die Daten für abgeschlossen halten. Wir benötigen hier explizite Codetransparenz – es muss sichergestellt sein, dass die übertragenen Daten nie mit Kontrollinformationen verwechselt werden. Dazu müssen im Datenfeld zufällig auftauchende Kontrollsequenzen maskiert werden.

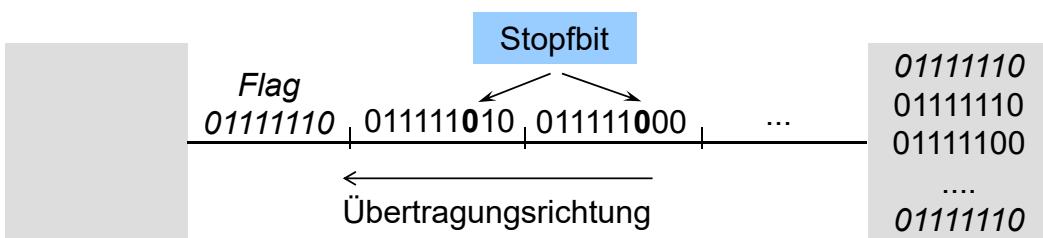
Ein Beispiel, wie explizite Codetransparenz erreicht werden kann, zeigt Zeile 3 der Abbildung auf der Folie: es wird festgelegt, dass einem Steuerzeichen immer ein DLE-Zeichen (Data Link Escape) vorangestellt sein muss. Nun ist das Auftreten eines ETX im Datenteil kein Problem mehr (Zeile 4). Das DLE sorgt hier also für die Codetransparenz – Steuerzeichen werden nur als solche erkannt, wenn ein DLE davor steht.

Dafür tritt ein neues Problem auf (Zeile 5): tritt die DLE-Bitfolge im Datenteil auf, wird sie als Beginn eines Steuerzeichens interpretiert, der Datenteil wird wieder nicht korrekt verarbeitet. Dies wird vermieden, indem der Sender ein weiteres DLE vor dieser Bitfolge im Datenteil einfügt. Ein doppeltes DLE (Zeile 6) bedeutet, dass das zweite DLE als Datenzeichen aufzufassen ist. Auf diese Art und Weise kann auch die DLE-Bitfolge als Teil der Daten übertragen werden.

Bit-Stuffing

(5) Maskierung von angeblichen Kontrollsequenzen

- ▶ Blockbegrenzung (Flag) ist eine ausgezeichnete Bitfolge (hier: 01111110)
 - *Bit-Stuffing* zur Vermeidung des Auftretens von 01111110 im Payload
 - Füge nach 5 aufeinanderfolgenden „1“-en eine „0“ ein, die der Empfänger wieder entfernt



Bit-Stuffing verfolgt das gleiche Prinzip wie bei der zeichenorientierten Übertragung: Anfang und Ende eines Rahmens werden durch eine spezielle Bitfolge (hier beispielhaft: 01111110) gekennzeichnet, die nicht innerhalb des Blocks auftreten darf. Um dies zu vermeiden, fügt man nun im Datenteil nach fünf aufeinanderfolgenden Einsen eine Null ein (auch dann, wenn gerade eine Null folgt). Somit erkennt der Empfänger deutlich an sechs aufeinanderfolgenden Einsen das Blockende und weiß, dass er bei einer Folge von fünf Einsen und einer Null die Null entfernen muss.

Kapitel 3: Sicherungsschicht

- **Rahmenbildung**

- ▶ Blockbildung, Codetransparenz durch Character/Bit Stuffing

- **Fehlererkennung/-behandlung und Flusskontrolle**

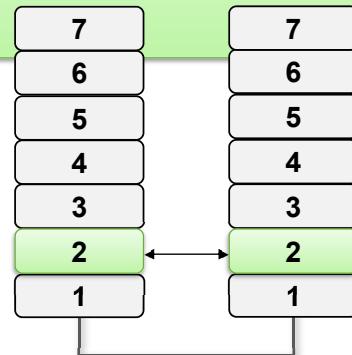
- ▶ Fehlererkennende Codes (CRC)
- ▶ Proaktive und reaktive Fehlerbehebung (FEC und ARQ)
- ▶ Flusskontrolle durch Sliding Window
- ▶ Beispielprotokoll: HDLC

- **Medienzugriff**

- ▶ Topologien
- ▶ Token Passing, CSMA/CD, CSMA/CA

- **Standards für lokale Netze**

- ▶ Ethernet



Fehlerursachen, Fehlertypen

• Übertragungsfehler

- ▶ Störeinflüsse führen zu *falsch detektierten Bits*
- ▶ *Hardwareinduzierte Fehler*, die auf dem Übertragungsmedium entstehen (seltener auch in der Anschlusselektronik der kommunizierenden Stationen)
 - Art und Häufigkeit von Übertragungsmedium und Datenrate abhängig
 - *Einzelbitfehler*
 - Bündelfehler (*Fehlerburst*)
- ▶ Des weiteren: Synchronisierungsfehler
 - Alle Bits werden falsch abgetastet

Störeinflüsse bei der Übertragung digitaler Daten führen normalerweise zur fehlerhaften Erkennung der Bits. Sowohl die Art der Störeinflüsse als auch die Häufigkeit, mit der Signale verfälscht werden, hängen von der Art des Mediums ab. Drahtlose Medien weisen im Allgemeinen deutlich höhere Fehlerraten auf als kabelgebundene Medien, Kupfermedien immer noch höhere Fehlerraten als Glasfaser.

Man unterscheidet generell zwei Fehlertypen:

- *Einzelbitfehler*: Fehler, bei denen nur ein Bit gekippt (= verfälscht) wurde, verursacht z.B. durch Spitzen im Rauschen.
- *Bündelfehler (Fehlerbursts)*: länger anhaltende Fehler, die über viele Bits in Folge gehen (z.B. durch längere Überspannungen oder Interferenz mit benachbarten Funkzellen). Achtung: ein Fehlerburst zeichnet sich dadurch aus, dass über eine längere Folge von Bits Verfälschungen aufgetreten sind; es müssen aber nicht notwendigerweise alle Bits der Folge verfälscht sein.

Ob ein Störeinfluss sich als Einzelbitfehler oder als Fehlerburst auswirkt, hängt von der Datenrate ab, mit der gesendet wird – je höher die Datenrate ist (je kürzer die Schrittdauer ist), desto mehr Bits können durch einen kurzen Störeinfluss verfälscht werden.

Eine weitere Fehlerklasse sind die *Synchronisierungsfehler*: der Empfänger ist falsch synchronisiert und tastet alle Bits falsch ab. Dieses Problem lässt sich lösen, indem einem Rahmen eine Präambel mit fest definierten Pegelwechseln vorangestellt wird und/oder selbsttaktende Leitungscodes eingesetzt werden. Diese Fehlerklasse soll hier nicht genauer betrachtet werden.

Fehlerwirkungen: Rechenbeispiel

- **Eine Störung von 20 ms führt ...**

- ▶ bei Telex (50 Bit/s, Signaldauer: 20 ms) zu einem Fehler von 1 Bit Länge → Einzelfehler
- ▶ bei ISDN (64 kBit/s, Signaldauer: 15,625 µs) zu einem Fehler von 1280 Bit Länge → Fehlerburst
- ▶ bei SDH (Weitverkehrsnetz, unterschiedliche Datenraten möglich)
 - 155 MBit/s, Signaldauer: 6,45 ns: zu einem Fehler von ca. 3,1 Mbit Länge
 - 622 MBit/s, Signaldauer: 1,61 ns: zu einem Fehler von ca. 12,4 Mbit Länge
 - 2,4 GBit/s, Signaldauer: 0,4 ns: zu einem Fehler von ca. 48 Mbit Länge
→ extreme Fehlerbursts!

(SDH: Synchronous Digital Hierarchy; analog dazu in den USA: Sonet. Weitverkehrstechnologie, nicht Fokus der Vorlesung.)

Fehlerhäufigkeiten

- Maß für die Fehlerhäufigkeit:

$$\blacktriangleright \text{Bitfehlerrate} \text{ (bit error rate, BER)} = \frac{\text{Summe gestörte Bits}}{\text{Summe übertragene Bits}}$$

- ▶ Abhängig von
 - Übertragungsmedium
 - Gesamtlänge des Übertragungswegs

- Typische Wahrscheinlichkeiten für Bitfehler:

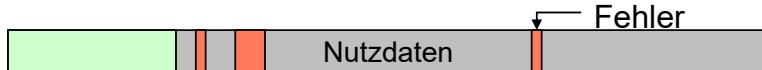
- ▶ Analoges Fernsprechnetz: $2 \cdot 10^{-4}$
- ▶ Funkstrecke: $10^{-3} - 10^{-4}$
- ▶ Ethernet (10Base2): $10^{-9} - 10^{-10}$
- ▶ Glasfaser: $10^{-10} - 10^{-12}$

Die Bitfehlerrate ist ein typisches Maß für die Wahrscheinlichkeit des Auftretens eines Übertragungsfehlers. Zu jedem Übertragungsmedium wird man eine Angabe zur BER finden.

Fehlerwirkungen

- Die Auswirkung eines Fehlers ist abhängig davon, welche Bits des Rahmens betroffen sind:

- ▶ (Nutz-)Datenfehler: Bits innerhalb der Nutzdaten



- ▶ Protokollfehler: Bits in Protokollkontrolldaten, Steuerzeichen, Adressen oder sonstigen protokollrelevanten Daten



Treten innerhalb der Nutzdaten Fehler auf, werden die Daten zwar korrekt zugestellt, aber die nächsthöhere Schicht des Empfängers verarbeitet fehlerhafte Daten, was letztlich zu Problemen auf den höheren Schichten oder in der empfangenden Anwendung führt.

Treten Fehler in den Kontrollinformationen auf, wird im Allgemeinen der Protokollablauf gestört; z.B. könnten die Daten dem falschen Rechner zugestellt oder auf dem empfangenden Rechner dem falschen Prozess zugestellt werden.

Beide Arten von Fehlern müssen gleichermaßen erkannt und behandelt werden.

Fehlererkennung und -behandlung

• Erste Notwendigkeit: Fehlererkennung

- ▶ Erzeuge „Muster“ im Block, anhand derer der Empfänger die Korrektheit / das Auftreten von Fehlern erkennen kann
 - Durch künstliches Hinzufügen von Redundanz zum Block beim Sender
 - *Error detecting codes*



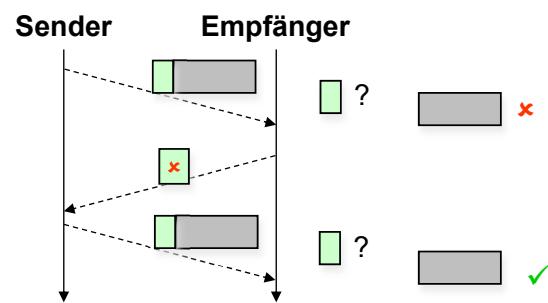
Teilweise kann man Fehler einfach ignorieren. Beispielsweise filtert das menschliche Hirn in gewissem Umfang Störungen heraus, so dass bei Übertragung von Audio und Video einzelne gestörte Bildteile oder Audiosequenzen keine Probleme verursachen.

Bei den meisten Daten allerdings ist es notwendig, sie fehlerfrei zu erhalten (Webseite, E-Mail, Datei, ...). Daher werden Techniken zur Fehlererkennung benötigt. Ganz generell gesprochen fügt der Sender Redundanzen zu einem zu übertragenden Block hinzu, anhand derer der Empfänger Übertragungsfehler erkennen lassen. Die Daten werden so codiert, dass eine Fehlererkennung möglich wird: man spricht von „Error Detecting Codes“.

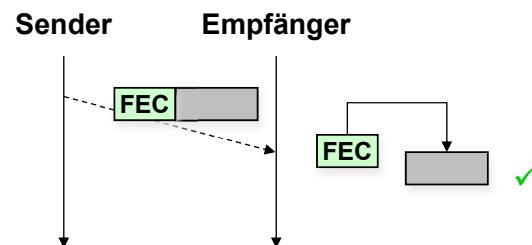
Fehlererkennung und -behandlung

• Nach Erkennung: Fehlerbehandlung

- ▶ Reaktiv: reagiere auf erkannten Fehler durch Neuübertragung
 - *Automatic Repeat Request (ARQ)*



- ▶ Proaktiv: Blöcke enthalten ausreichend Redundanzen zur Korrektur
 - *Error correcting codes (Forward Error Correction, FEC)*



Um einen erkannten Fehler zu beheben, gibt es zwei Möglichkeiten: proaktives oder reaktives Vorgehen.

Bei proaktivem Vorgehen geht man davon aus, dass Fehler auftreten werden und codiert die Daten proaktiv so, dass Fehler nicht nur erkannt, sondern sogar korrigiert werden können. Dazu werden deutlich größere Mengen an Redundanzen benötigt als zur reinen Fehlererkennung. Diese Art von Codes heißt „Error Correcting Codes“.

Treten Fehler relativ selten auf, bietet sich eher ein reaktives Vorgehen an: wird ein Fehler erkannt, wird der zugehörige Block verworfen und noch einmal übertragen. Die Klasse dieser Verfahren wird „Automatic Repeat Request“ genannt und das simpelste Verfahren ist das Alternating-Bit-Protokoll.

Fehlererkennung und -korrektur: Redundanzen

- **Hinzufügen von Redundanzen:**
 - ▶ Länge des gesamten Blocks: n Bit (2^n mögliche „Code“wörter)
 - ▶ Länge der Nachricht (Header und Payload):
 $m (< n)$ Bit (2^m mögliche Nachrichten ohne Redundanz)
 - Nachricht = Rahmen ohne Prüfbits
 - ▶ k Prüfbits ($k = n - m$)
- **Forme eine Sphäre von Codewörtern um jede erlaubte Nachricht**
 - ▶ *Hamming-Abstand D:*
 - Hamming-Abstand zweier Codewörter:
 - Anzahl der Positionen, in denen sich zwei Codewörter unterscheiden
 - Hamming-Abstand eines Codes:
 - Minimum der Hamming-Abstände aller möglichen Codewort-Paare
 - ▶ Dann:
 - $D \geq 2t + 1 \rightarrow$ Code kann t Fehler korrigieren
 - $D \geq t + 1 \rightarrow$ Code kann t Fehler erkennen

Die generellen Möglichkeiten, die man mit einem fehlererkennenden oder –korrigierenden Code erreichen kann, lassen sich mittels des Hamming-Abstands abschätzen. Als einfaches Beispiel kann man sich die Betrachtung einzelner Bits hernehmen: gültige Nachrichten sind „0“ und „1“. Um einen Fehler erkennen zu können, benötigt man einen Hamming-Abstand von 2, daher codiert man die „0“ als „00“ und die „1“ als „11“. Unter der Annahme, dass lediglich ein einzelner Fehler auftreten kann, kann der Empfänger bei Erhalt einer „00“ eine „0“ decodieren, bei Erhalt einer „11“ eine „1“. Werden eine „01“ oder eine „10“ empfangen, kann der Empfänger von einem Bitfehler ausgehen, da es keine gültigen Codeworte sind. (Allerdings funktioniert dies immer nur unter der Annahme, dass nur ein einzelner Fehler aufgetreten ist!)

Zur Korrektur von Fehlern benötigt man mehr Redundanz: um einen Fehler zu korrigieren, benötigt man einen Hamming-Abstand von 3. Betrachtet man wieder einzelne Bits, kann man eine „0“ als „000“ und eine „1“ als „111“ codieren. Wird nur z.B. eine „110“ empfangen, geht man davon aus, dass nur eine Stelle verfälscht worden sein kann und korrigiert deswegen zu „111“. Man legt sozusagen eine Sphäre von Codeworten um jede erlaubte Folge und interpretiert „000“, „100“, „010“ und „001“ als „0“, „111“, „110“, „101“, „011“ als „1“.

Fehlererkennung: Paritätsüberprüfung

• Paritätsüberprüfung

- ▶ Nur ein Bit Redundanz
- ▶ Gerade oder ungerade Parität
- ▶ Beispiel:

1101010	0110001	0001000	1000100	1101001	1110010	1101011
---------	---------	---------	---------	---------	---------	---------

- ▶ Parity-Bit bei gerader Parität: 1

- ▶ Block inklusive Redundanz:

1101010	0110001	0001000	1000100	1101001	1110010	1101011	1
---------	---------	---------	---------	---------	---------	---------	---

Die simpelste Form des Hinzufügens von Redundanz zur Fehlererkennung ist die Ergänzung der Daten um ein Paritätsbit (engl.: Parity Bit), welches mit zum Empfänger übertragen wird. Die Bildung des Parity-Bits lässt sich auf zweierlei Art und Weise vornehmen:

- *Gerade Parität*: Gesamtzahl der 1en einschließlich des Parity-Bits ist gerade.
- *Ungerade Parität*: Gesamtzahl der 1en einschließlich des Parity-Bits ist ungerade.

Im Allgemeinen findet gerade Parität verbreiteter Einsatz.

Durch dieses Verfahren kann die Verfälschung eines einzelnen Bits erkannt werden: unterscheiden sich zwei Nachrichten in nur einem Bit, haben sie einen Hamming-Abstand von 1. Durch das Hinzufügen des Parity-Bits wächst der Hamming-Abstand auf 2, so dass man einen einzelnen Fehler erkennen, aber keinen korrigieren kann.

Der Empfänger empfängt einen gesamten Rahmen (inklusive Parity-Bit) und berechnet selbst, welchen Wert das Parity-Bit aufgrund der empfangenen Bitwerte haben müsste. Diesen Wert vergleicht er mit dem empfangenen Parity-Bit-Wert. Sind die Werte gleich, geht der Empfänger von einem korrekten Empfang aus; sind die Werte unterschiedlich, hat der Empfänger einen Übertragungsfehler erkannt.

Allerdings kann der Fehler nicht korrigiert werden – dies kann man schon anhand des Hamming-Abstands sehen, sich aber auch intuitiv herleiten: egal, welches Bit verfälscht ist, das Parity-Bit hat den gleichen Wert. Es könnte sogar sein, dass alle „normalen“ Bits des Rahmens korrekt sind und das Parity-Bit verfälscht wurde. Zudem könnte es auch sein, dass 3, 5, 7, ... Bits verfälscht sind – dies hätte die gleiche Auswirkung auf das Parity Bit.

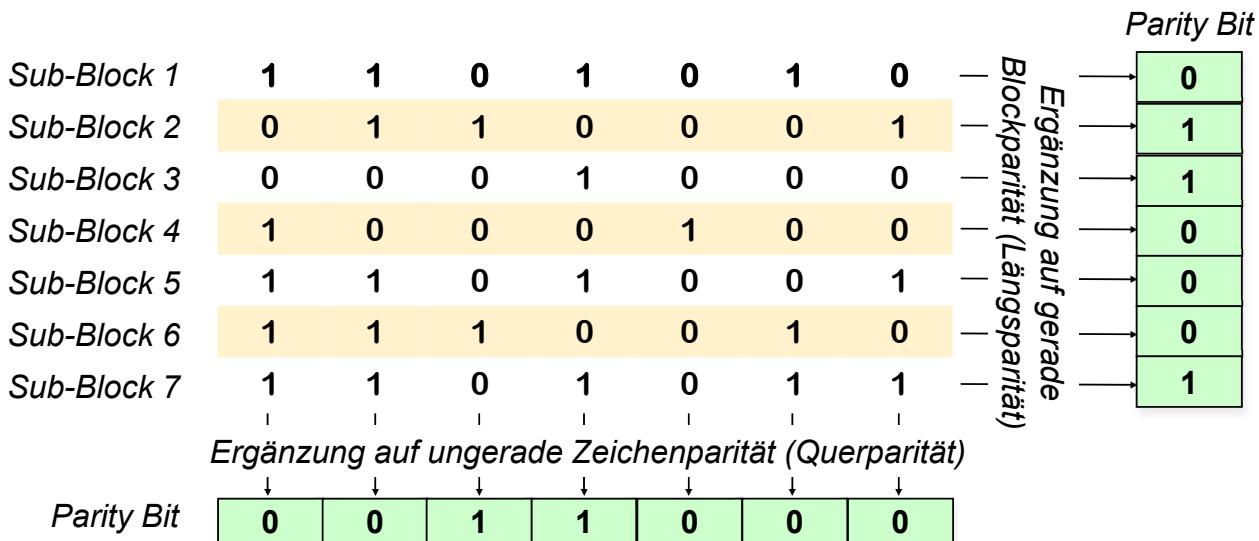
Schlimmer ist allerdings, dass sich zwei Fehler gegenseitig aufheben und Fehler in einer gerade Anzahl von Bits nicht erkannt werden können.

Fehlererkennung: Paritätsüberprüfung

- Paritätsüberprüfung

- Längs-, Quer- oder Kreuzparität

Beispiel: 1101010 0110001 0001000 1000100 1101001 1110010 1101011



Man kann zudem zwischen zwei Modi unterscheiden, die bei der Verwendung des Parity Bit zur Fehlererkennung eingesetzt werden können:

- Block- oder Längsparität:** Berechnung der Parität über Blöcke wie auf der vorherigen Folie.
- Zeichen- oder Querparität:** Sicherung von Einzelbits in aufeinanderfolgenden Blöcken. Bei Einsatz dieses Verfahrens muss man sich nur Gedanken machen, wie die Parity-Bits übertragen werden – eventuell als eigener Block.

Teilt man einen Block in Sub-Blöcke auf, ist auch mehr möglich – eine *Kreuzsicherung*: gleichzeitige Anwendung von Längs- und Querparität wie auf dieser Folie gezeigt. Über jeden Sub-Block wird die Längsparität berechnet (hier beispielhaft gerade Parität) und für jede Bitposition die Querparität über alle Sub-Blöcke (hier beispielhaft ungerade Parität).

Dies hat den Effekt, dass der Hamming-Abstand zwischen zwei Blöcken, die sich in nur einem Bit unterscheiden, auf drei anwächst - durch einen einzigen Bitfehler werden direkt zwei Paritätsbits verfälscht, die zusammen die Bestimmung der Position des Fehlers und somit seine Korrektur erlauben. Allerdings führt auch hier das Auftreten von zwei oder mehr Fehlern zu Fehlinterpretationen.

Eine Paritätsüberprüfung ist also nur dann sinnvoll, wenn man davon ausgehen kann, dass nur Einzelbitfehler auftreten können. In der Datenkommunikation ist dies selten der Fall – verwendet werden Parity-Bits aber zum Beispiel bei der Speicherverwaltung (RAM).

Fehlererkennung: Cyclic Redundancy Check (CRC)

• Verallgemeinerung der Paritätsüberprüfung

- ▶ Größere Zahl an Prüfbits, um mehr Fehler erkennen zu können
- ▶ Einsatz von CRC zur Berechnung einer Prüfsumme über den Block (*Frame Check Sequence, FCS*)



• Bildung der CRC-Prüfsumme:

1. Zu prüfende Bitfolge wird als Polynom aufgefasst
2. Erweiterung um 0-Folge (Anzahl 0en = Grad des Prüfpolynoms), Teilung durch vereinbartes Prüfpolynom (Generatorpolynom)
3. FCS ist Rest der Division, wird an die Bitfolge angehängt
4. Beim Empfänger wird neu dividiert (einschließlich Rest), bei fehlerfreier Übertragung muss das Ergebnis 0 sein

Das gängige Verfahren in der Datenkommunikation ist der Cyclic Redundancy Check. Dieses Verfahren ist in der Lage, auch Fehler-Bursts sehr zuverlässig zu erkennen. Die zu übertragenden Daten (Protokoll- und Nutzdaten) können beliebige Länge haben; mittels CRC wird eine Frame Check Sequence (FCS) berechnet, an letzter Stelle in den Rahmen eingefügt und mit übertragen. Anhand der FCS kann der Empfänger ermitteln, ob während der Übertragung Fehler in den vorhergehenden Daten aufgetreten sind.

Für die Berechnung der FCS wird ein sogenanntes Generatorpolynom festgelegt. Der Sender fasst die Bitfolge der Daten als Polynom auf und teilt sie durch das Generatorpolynom (Modulo-2-Arithmetik). Der Rest, der bei der Division übrigbleibt, wird an die Bitfolge angehängt. Durch Anhängen des Rests an die Daten entsteht eine erweiterte Bitfolge (bzw. ein Polynom), welche Vielfaches des Generatorpolynoms ist.

Dadurch entsteht eine Art Codiervorschrift: eine übertragene Bitfolge ist nur dann gültig, wenn sie (als Polynom interpretiert) ein Vielfaches des Generatorpolynoms ist. Der Empfänger teilt die gesamte empfangene Bitfolge durch das gleiche Generatorpolynom. Bei korrektem Empfang darf bei der Division kein Rest bleiben.

Dieses Verfahren erkennt nicht nur mit sehr hoher Wahrscheinlichkeit fehlerhafte Datenblöcke, es ist auch einfach in Hardware zu implementieren, so dass die Berechnung der FCS effizient erfolgen kann und zu keiner spürbaren Verzögerung im Prozess des Versendes der Daten führt.

Fehlererkennung: CRC-Beispiel

- **Gegeben:**

- ▶ Zu sendender Block: 110011
- ▶ Generatorpolynom: $x^4 + x^3 + 1$
 - Polynom in Modulo-2-Binärarithmetik: 11001

- **Verfahren:**

- ▶ Addition/Subtraktion Modulo-2 entspricht bitweiser XOR-Verknüpfung
- ▶ Dividend ist teilbar durch Generatorpolynom, falls der Dividend mindestens so viele Stellen besitzt wie das Generatorpolynom (führendes Bit muss 1 sein)
- ▶ Länge der FCS = Grad des Generatorpolynoms = 4

Fehlererkennung: CRC-Beispiel – Senden

- **Gegeben:**

- ▶ Zu sendender Block: 110011
- ▶ Generatorpolynom: 11001

- **Berechnung der FCS:**

$$110011 \ 0000 \div 11001 = 100001$$

11001

000001 0000

1 1001

0 1001 = Rest

- Zu übertragender, erweiterter Block: 11 0011 **1001**

Fehlererkennung: CRC-Beispiel – Empfangen

- **Gegeben:**

- ▶ Zu sendender Block: 110011
- ▶ Generatorpolynom: 11001

- **Empfangen eines korrekten Blocks:**

$$\begin{array}{r} 110011 \ 1001 \div 11001 = 100001 \\ \underline{11001} \\ 000001 \ 1001 \\ \underline{1 \ 1001} \\ 0 \ \textcolor{blue}{0000} = \textbf{Rest} \end{array}$$

- **Kein Rest**

- ▶ Somit (mit sehr hoher Wahrscheinlichkeit) Daten fehlerfrei

Erhält der Empfänger bei Division durch das Generatorpolynom 0, geht er von einer fehlerfreien Übertragung aus.

Dies ist nicht notwendigerweise korrekt; wie bei der Paritätsüberprüfung gibt es auch hier Kombinationen von Fehlern, die nicht erkannt werden können.

Fehlererkennung: CRC-Beispiel – Empfangen

- **Gegeben:**
 - ▶ Zu sendender Block: 110011
 - ▶ Generatorpolynom: 11001
- **Empfangen eines verfälschten Blocks:**

$$\begin{array}{r} 11\textcolor{red}{1}111 \ 100\textcolor{red}{0} \div 11001 = 101001 \\ \underline{11001} \\ 001101 \ 1 \\ \underline{1100} \ 1 \\ 0001 \ 0000 \\ \underline{1} \ 1001 \\ 0 \ \textcolor{red}{1001} = \text{Rest} \neq 0 \end{array}$$

- Rest ungleich 0, somit Fehler in Übertragung

Fehlererkennung: CRC-Leistungsfähigkeit

- International genormte Prüfpolynome:

- ▶ CRC-12 $= x^{12} + x^{11} + x^3 + x^2 + x + 1$
- ▶ CRC-16 $= x^{16} + x^{15} + x^2 + 1$
- ▶ CRC-CCITT $= x^{16} + x^{12} + x^5 + 1$
- ▶ Ethernet: $= x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11}$
 $+ x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

- CRC-16 und CRC-CCITT entdecken

- ▶ alle Einzel- und Doppelfehler + alle Fehler ungerader Bitanzahl
- ▶ alle Fehlerbursts der Länge ≤ 16
- ▶ 99,997 % aller Fehlerbursts der Länge 17
- ▶ 99,998 % aller Fehlerbursts der Länge ≥ 18

→ Verbleibende Fehlerrate $< 0.5 * 10^{-5}$ der BER

Das CRC-Verfahren reduziert das Risiko, verfälschte Daten zu verarbeiten, kann aber nicht ganz ausschließen, dass verfälschte Daten als korrekt interpretiert werden. Die Wahrscheinlichkeit für so einen Fall ist allerdings extrem gering.

CRC wird nicht nur in der Datenkommunikation eingesetzt, sondern z.B. auch bei komprimierten Archiven.

Allerdings ist nur eine Erkennung von Fehlern möglich, keine Korrektur. Verfälschte Rahmen müssen verworfen werden, da man nicht feststellen kann, an welcher Stelle Fehler aufgetreten sind.

Vorwärtsfehlerkorrektur (Forward Error Correction)

- **Redundanz (durch Prüfsumme)**

- ▶ Bislang nur zur Überprüfung der Daten
- ▶ Jetzt: proaktive Fehlerbehandlung
 - Redundanz zur *Rekonstruktion verfälschter Datenblöcke*

- **Beispiel:**

- ▶ Zu senden sind die Blöcke

0101	- B1
1111	- B2
0000	- B3
1010	- B4

- ▶ Berechne vierten Block (XOR):

- ▶ Sende alle vier Blöcke an den Empfänger

- Bei korrektem Erhalt dreier Blöcke kann der vierte berechnet werden
- Schutz gegen Verlust eines ganzen Blockes

Für die Behebung von Übertragungsfehlern wird mehr Redundanz benötigt. Beispielsweise könnte eine zu übertragende Dateneinheit in Blöcke aufgeteilt werden, die mittels XOR verknüpft werden, um einen Prüfblock zu berechnen. Wird jeder dieser Blöcke zusätzlich mit einer CRC versehen, kann der Empfänger feststellen, welche der Blöcke korrekt übertragen wurden; wird einer der Blöcke als fehlerhaft identifiziert, kann er aus den anderen Blöcken rekonstruiert werden.

Forward Error Correction: Redundante Pakete

- Über wie viele Blöcke n soll ein XOR-Block berechnet werden?

► Tradeoff: Erhöhe n ...

- ... weniger Overhead durch seltene XOR-Blöcke
- ... größere Verzögerung bis zur Korrektur, falls ein Fehler auftritt
- ... mehr Buffer zum Zwischenspeichern von Blöcken zur Korrektur
- ... größeres Risiko, dass zwei oder mehr Blöcke korrupt sind

Hamming-Code

- **Idee:**

- ▶ Erlaube die Korrektur von Einzelbitfehlern
 - Nötig: Hamming-Abstand von 3
- ▶ Nutze dazu mehrere Paritätsbits, die jeweils (überlappend) mehrere Bits der Nachricht prüfen
 - Fehler können durch Kombination der Paritätsbits korrigiert werden

- **Realisierung eines „minimalen“ Codes**

- ▶ Jede natürliche Zahl kann durch eine Summe von Zweierpotenzen ausgedrückt werden
- ▶ Forme Codewörter $w = z_1, \dots, z_n$ durch Einschieben der Paritätsbits an den Positionen, die eine Zweierpotenz sind
- ▶ Jedes der Paritätsbits prüft alle anderen Positionen, zu deren Darstellung in Zweierpotenzen die Position des Paritätsbits nötig ist

Anderer Ansatz: erzeuge keine redundanten Blöcke, sondern – wie bei CRC – pro Block eine Redundanz, die die Korrektur von Fehlern in diesem Block ermöglicht. In dieser Vorlesung wird nur die einfachste Variante betrachtet: der Hamming-Code, der einen Bitfehler innerhalb eines Blockes korrigieren kann. Dazu ist es notwendig, zwischen je zwei beliebigen, möglichen Blöcken einen Hamming-Abstand von mindestens drei zu erreichen.

Hierzu muss man Regeln festlegen, um zusätzliche Bits berechnen zu können, damit man einen minimalen Overhead erzielt.. Beim Hamming-Code wird hierzu eine Menge an Parity-Bits verwendet, die jeweils einen Teil der Nachricht prüfen, so dass jedes Datenbit von mindestens 2 Parity-Bits geprüft wird und keine zwei Datenbits von der gleichen Kombination an Parity-Bits geprüft werden.

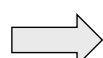
Achtung: der Hamming-Code wurde in der Vorlesung übersprungen und ist daher nicht relevant.

Hamming-Code: Beispiel

	ASCII-Code		Codewort																
	H	A	M	M	I	N	G	1	2	3	4	5	6	7	8	9	10	11	Position
H	1001000							1	2	3	4	5	6	7	8	9	10	11	
A	1100001							0	0	1	1	0	0	1	0	0	0	0	0
M	1101101							1	0	1	1	1	0	0	1	0	0	0	1
M	1101101							1	1	1	0	1	0	1	0	1	0	1	1
I	1101001							1	1	1	0	1	0	1	0	1	0	1	0
N	1101110																		
G	1100111																		

Position der Datenbits als Summe der Prüfbitpositionen:

3 = 1 + 2
5 = 1 + 4
6 = 2 + 4
7 = 1 + 2 + 4
9 = 1 + 8
10 = 2 + 8
11 = 1 + 2 + 8



Paritätsbit...	... prüft Datenbit
1	3, 5, 7, 9, 11
2	3, 6, 7, 10, 11
4	5, 6, 7
8	9, 10, 11

Der Empfänger der Daten berechnet alle Paritätsbits und prüft, welche davon falsch sind. Die Positionen der falschen Paritätsbits werden aufsummiert und ergeben dadurch die Position des Bits, welches falsch ist und korrigiert werden muss.

Hamming hat gezeigt, dass diese Form der Berechnung von Prüfbits einen Hamming-Abstand von 3 erzeugt. Problem daher: dieser Code eignet sich aufgrund des Hamming-Abstands nur zur Korrektur von Einzelbitfehlern. Die Korrektur oder eventuell sogar die Erkennung mehrerer Fehler ist nicht möglich.

Hamming-Code: Grenzen

- Unterschied zwischen empfangener und berechneter Prüfsumme in Positionen 1, 2, und 4!
 - Aufsummieren der Positionen 1, 2 und 4 → 7
 - Bit an Position 7 wird korrigiert

Einschränkungen des Hamming-Codes:

- a) Bit 4 und Bit 11 verfälscht:
 - Paritätsbits 1, 2, 4, 8 sind falsch
 - Bit 15 soll korrigiert werden, existiert aber nicht
 - b) Bit 2 und Bit 4 verfälscht
 - Paritätsbits 2, 4 falsch
 - Bit 6 wird fehlerhafterweise als falsch interpretiert und “korrigiert”
 - c) Bits 1, 8, 9 verfälscht
 - Alle Paritätsbits korrekt
 - Fehler nicht erkannt

Der Hamming-Code eignet sich nur zur Korrektur von 1-Bit-Fehlern.

In der Realität (speziell in drahtlosen Netzen mit deutlich hoher BER) treten aber meist Fehler-Bursts auf. In der Praxis eingesetzte Verfahren sind daher deutlich komplexer und verlangen mehr Redundanz. Daher werden solche Verfahren auch nur dann eingesetzt, wenn die BER so hoch ist, dass ohne Fehlerkorrektur keine Kommunikation mehr möglich wäre (drahtlose Netze) oder wenn eine Fehlerkorrektur die einzige Möglichkeit ist, Fehler zu beheben (Broadcast-Netze wie DVB-T, Speichermedien wie DVD).

In kabelgebundenen Netzen wird meist nur auf die deutlich einfachere Fehlererkennung zurückgegriffen (zumal heutige Netze eine relativ geringe BER aufweisen). Im Fall eines erkannten Fehlers kann eine Behebung z.B. durch erneute Übertragung der verfälschten Daten erfolgen.

Fehlerbehandlung: Sendewiederholungsverfahren

- **Sendewiederholungsverfahren**
(Automatic Repeat Request, ARQ)

- ▶ Reaktive Fehlerbehandlung
- ▶ Strategien:
 - Stop-and-Wait
 - Go-Back-N
 - Selective Repeat/Reject

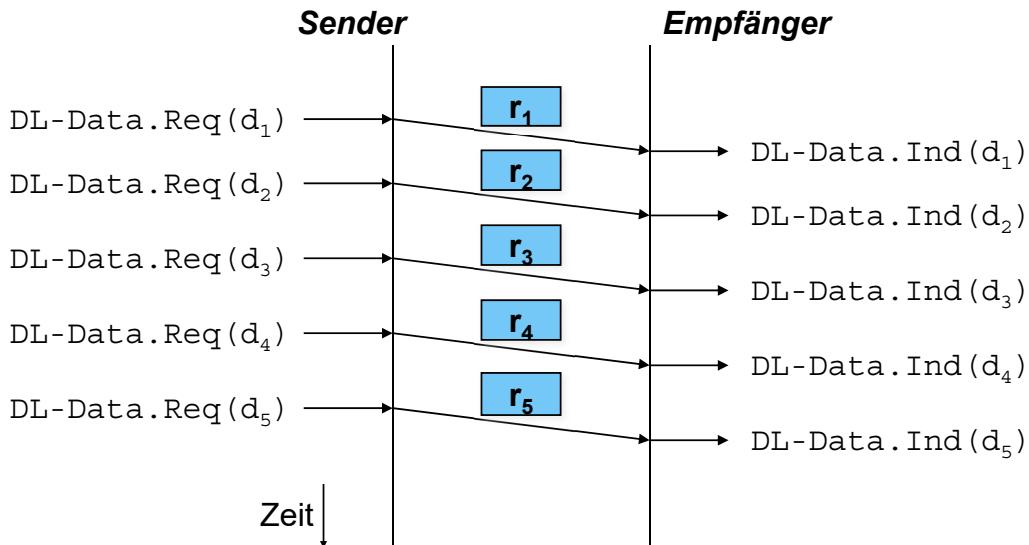


Wird eine Fehlererkennung verwendet, müssen als verfälscht erkannte Rahmen verworfen werden. Die einzige Möglichkeit zur Fehlerbehebung ist eine Neuübertragung der Rahmen. Während Rahmen auf Empfängerseite verworfen werden, muss eine Sendewiederholung durch den Sender erfolgen. Daher sind Protokollmechanismen notwendig, durch die der Empfänger dem Sender signalisieren kann, welche Rahmen korrekt angekommen sind und welche neu übertragen werden müssen. Solche Mechanismen werden als „Automatic Repeat Request“ (ARQ) bezeichnet.

Übertragung ohne Fehlerbehandlung

- **Sender und Empfänger sind immer bereit**

- ▶ Ständiger Datenfluss vom Sender zum Empfänger
- ▶ Übertragungsfehler werden nicht berücksichtigt



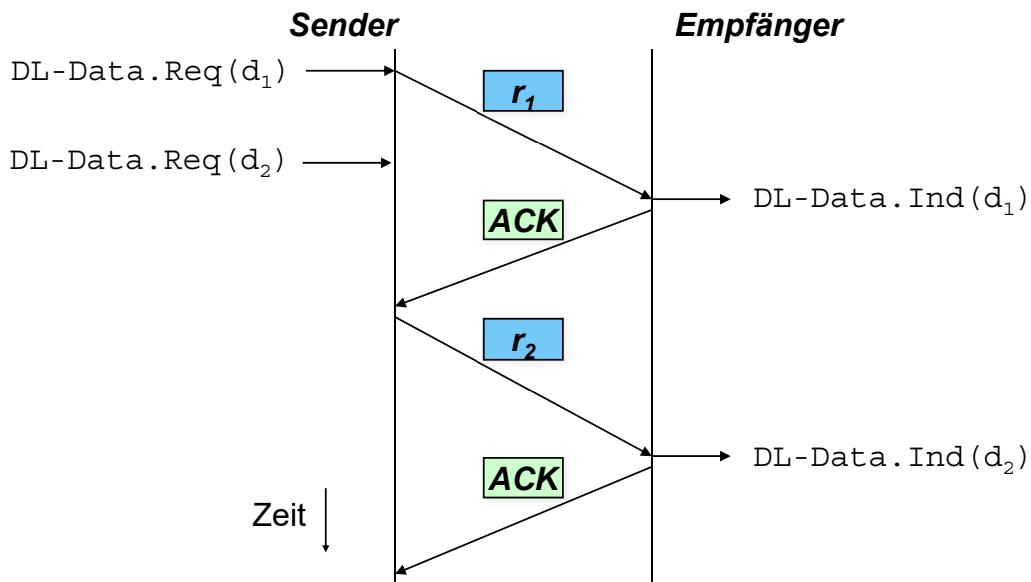
Protokolle auf Schicht 2 wären sehr einfach, wenn es keine Fehlersituationen gäbe. Können Fehler ignoriert werden (z.B. Audioinformationen) oder werden fehlerkorrigierende Codes eingesetzt, kann die Übertragung wie hier dargestellt erfolgen.

Notation: Auf Senderseite sollen Daten d_i übertragen werden; dazu werden sie an die Sicherungsschicht übergeben. Diese überträgt sie als Rahmen r_i – die Daten d_i zusammen mit z.B. einer CRC-Prüfsumme zur Fehlererkennung und eventuell weiteren Kontrollinformationen, die hier aber nicht von Bedeutung sind.

Fehlerbehandlung: Stop-and-Wait

- Empfang eines Rahmens muss *bestätigt* werden

- ▶ Sender muss auf Bestätigung (*ACK*) warten, bevor er den nächsten Rahmen sendet



Die einfachste Möglichkeit der Fehlerbehandlung wird durch die Bestätigung des Empfangs realisiert. Beim *Stop-and-Wait-Verfahren* hat der Empfänger jede Dateneinheit explizit durch eine Kontrollnachricht ACK positiv (oder durch NACK negativ) zu bestätigen. (Dies ist im Wesentlichen das Alternating Bit Protocol aus Kapitel 1.) Die Bestätigung erfolgt nur schichtenintern, sie wird nicht an den Dienstnutzer weitergeleitet. Es handelt sich also um einen unbestätigten, aber zuverlässigen Dienst.

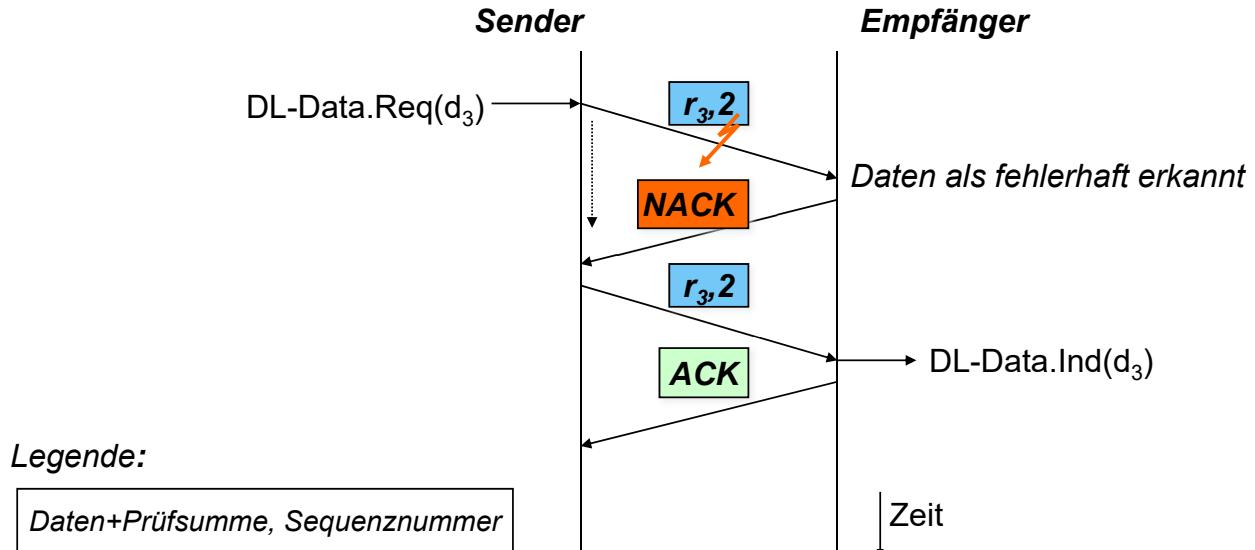
Als positiven Nebeneffekt der Quittierung kann man ansehen, dass der Empfänger nicht überlastet werden kann – bei unbestätigter Datenübertragung wie auf der vorherigen Folie könnte es vorkommen, dass der Empfänger die Rahmen nicht in der Geschwindigkeit verarbeiten kann, wie sie bei ihm ankommen und aufgrund dessen Daten verwerfen muss. Dies ist hier nicht mehr möglich.

Allerdings ist das Verfahren sehr ineffizient: einen großen Teil der Zeit verbringt der Sender mit dem Warten auf Bestätigungen, so dass die Kapazität der Leitung nicht ausgenutzt werden kann – insbesondere dann, wenn das Bandbreiten-Verzögerungs-Produkt groß ist, denn Bestätigungen erhält der Sender erst frühestens nach dem Doppelten der Latenz (plus Verarbeitungszeit des Rahmens durch den Empfänger und Sendezeit der Bestätigung, aber diese werden oft als verhältnismäßig klein angesehen und vernachlässigt).

Fehlerbehandlung: explizite Übertragungswiederholung

• Explizite Übertragungswiederholung

- Fehlerhafte Rahmen explizit durch **NACK** (Negative Acknowledgement) anfordern



Anmerkung zur Legende: r_i bezeichnet den Rahmen, also Daten + Kontrollinformationen der Schicht. Trotzdem wird die Sequenznummer (die auch eine Kontrollinformation ist) hier explizit zusätzlich angegeben, um die Abläufe besser darstellen zu können.

Wie auf der vorherigen Folie erwähnt, steht dem Empfänger je nach Implementierung auf einer Kontrollnachricht „NACK“ (negative acknowledgement, in Protokollen auch oft kurz als „NAK“ bezeichnet) zur Verfügung, über die er den Empfang eines fehlerhaften Rahmens signalisieren kann. Der Sender reagiert auf den Erhalt dieser Kontrollnachricht durch eine Wiederholung des vorherigen Rahmens.

Sollte der Sender weder eine Bestätigung durch eine ACK-PDU erhalten, noch eine Aufforderung zur Übertragungswiederholung durch eine NACK-PDU, so sendet er nach einer bestimmten Zeitspanne auch in diesem Fall den Rahmen erneut – denn auch die ACK/NACK-PDU kann durch eine Störung verfälscht werden, so dass der Sender nicht weiß, ob sein Rahmen korrekt angekommen ist.

Wichtig ist hier zudem, dass der Sender bei der Übertragung eines Rahmens als weitere Kontrollinformation im Header eine Sequenznummer angibt, damit der Empfänger eindeutig entscheiden kann, ob der empfangene Rahmen auch der erwartete ist: angenommen, der Empfänger empfängt Rahmen 3 (hier mit der Sequenznummer 2) korrekt und sendet ein ACK zurück. Durch eine Störung wird das ACK verfälscht und der Sender kann sich nicht sicher sein, ob Rahmen 3 angekommen ist und überträgt ihn erneut. Ohne Sequenznummer würde der Empfänger denken, er würde nun Rahmen 4 empfangen, da er Rahmen 3 korrekt bestätigt hat. Er würde die Nutzdaten fälschlicherweise noch einmal verarbeiten. Anhand der Sequenznummer kann der Empfänger diese Situation erkennen.

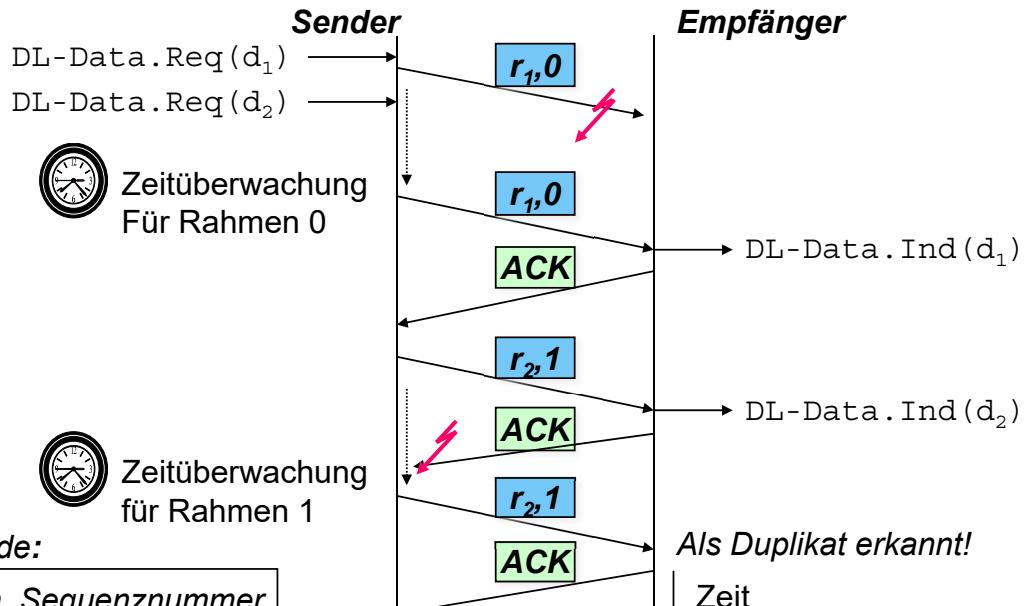
Die Sequenznummern werden oft als $N(S)$ bezeichnet; analog dazu können auch die Quittungen eine Bestätigungsnummer $N(R)$ enthalten, um dem Sender mitzuteilen, auf welche Sequenznummer sich eine Quittung bezieht.

Bei Vollduplexbetrieb können sowohl $N(S)$ als $N(R)$ im Header eines Rahmens angegeben werden, um zugleich in eine Richtung Daten zu übertragen und aus der anderen Richtung erhaltene Daten zu quittieren (Piggybacking).

Fehlerbehandlung: implizite Übertragungswiederholung

• Paketverlust

- Zeitüberwachung (*Timeout*) auf Senderseite & erneute Übertragung



Erhält der Sender nach einer bestimmten Zeitspanne (*Timeout*) keine Bestätigung für einen bestimmten Rahmen, so sendet er ihn erneut. Dieser Mechanismus ist (wie auf der letzten Folie erwähnt) notwendig, wenn negative Quittungen verwendet werden, kann aber darüber hinaus negative Quittungen auch ganz ersetzen: im Falle des korrekten Empfangs eines Rahmens sendet der Empfänger ein ACK, im Falle eines fehlerhaften Empfangs gibt er schlicht gar keine Rückmeldung. Die Erkennung eines Fehlers durch den Sender erfolgt somit nur dadurch, dass nach einer bestimmten Zeitspanne keine Antwort empfangen wurde – der Sender hat einen Timeout und überträgt den Rahmen neu.

Die Größe des Timeouts ist sehr wichtig:

- Der Timeout-Wert muss natürlich zumindest zwei mal so groß sein wie die Latenz auf der Leitung (plus Verarbeitungs- und Sendezeit des Empfängers) – vorher kann keine Quittung eintreffen.
- Ein zu niedriger Timeout (nur knapp über der zweifachen Latenz) führt dazu, dass der Sender sehr schnell beginnt, Rahmen zu wiederholen. Es kann durch schwankende Latenzen oder Verarbeitungszeiten auf Empfängerseite allerdings vorkommen, dass kein Rahmenverlust auftrat, sondern ein Rahmen oder dessen Bestätigung schlicht etwas später ankommt. Somit wird ein Rahmen (sinnlos) mehrfach übertragen, was zur Verschwendungen der Leitungskapazität führt.
- Ein zu hoher Timeout sorgt dafür, dass Verluste erst spät erkannt werden. Dies behindert die Kommunikation und führt zu einer Leistungssenkung.

Fehlerbehandlung: erweiterte ARQ-Verfahren

• Stop-and-Wait

- ▶ Ineffizient: nach jedem Rahmen auf Quittung warten
 - Latenz zwischen Sender und Empfänger erzeugt lange Wartezeiten
 - Schlechte Ausnutzung der Leitung
- ▶ Sinnvoller: erlaube Übertragung mehrerer Rahmen ohne auf sofortige Quittierung zu warten

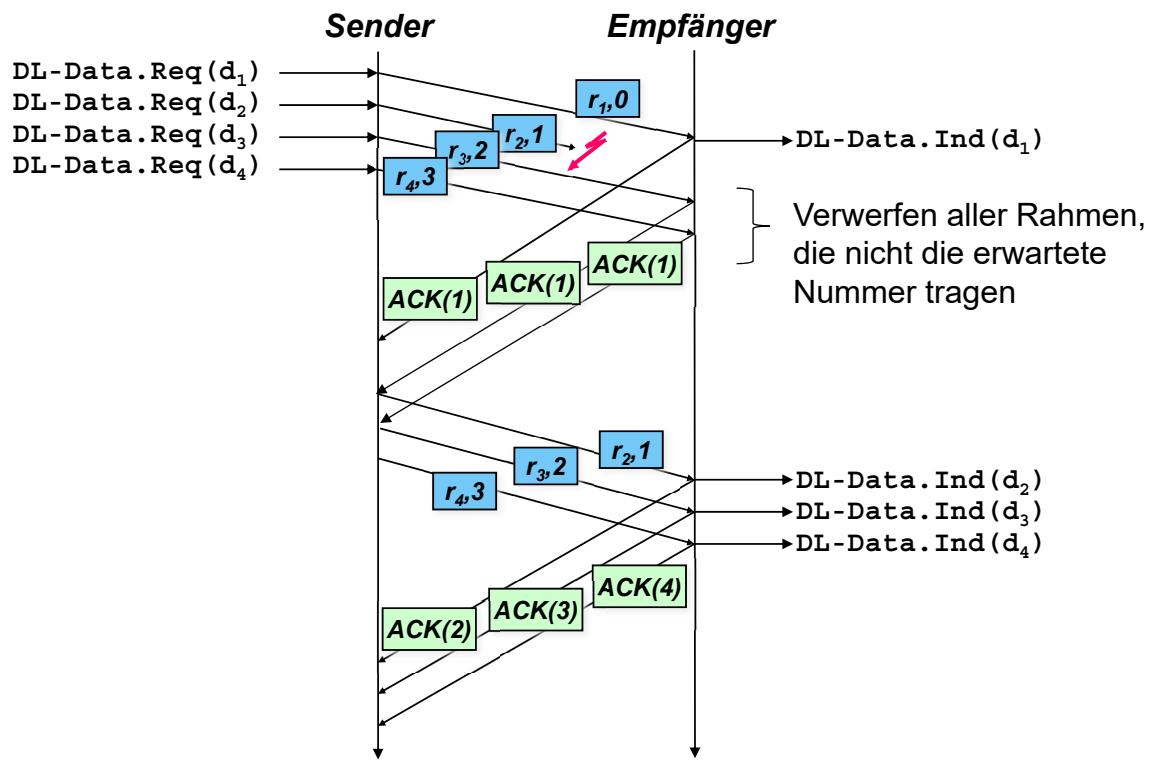
• Regelungen zur Quittierung bei mehreren Rahmen

- ▶ *Go-Back-N (Reject)-Verfahren*
 - Sämtliche Rahmen ab dem fehlerhaften müssen erneut übertragen werden
- ▶ *Selective Repeat*
 - Nur der als fehlerhaft angegebene Rahmen muss wiederholt werden



Da Stop-and-Wait sehr ineffizient ist, erlaubt man dem Sender, mehrere mit Sequenznummern versehene Rahmen zu versenden, ohne auf eine Quittung warten zu müssen. Falls nun allerdings ein Rahmen verfälscht ankommt, muss geregelt werden, wie der Empfänger mit diesem und allen folgenden noch eintreffenden Rahmen verfährt. Die gängigsten Mechanismen sind Go-Back-N und Selective Repeat.

Fehlerbehandlung: Go-back-N



Go-back-N

Der Empfänger bestätigt dem Sender jeden korrekt erhaltenen Rahmen. Bitte beachten: üblicherweise wird in der Praxis der Erhalt eines ACK(i) verstanden als „Bis i-1 ist alles korrekt angekommen; der nächste Rahmen, den ich erwarte, ist derjenige mit Nummer i“.

Wird ein Rahmen bei der Übertragung beschädigt, kann der Empfänger den Rahmen zwar (meist) als fehlerhaft erkennen, verwirft ihn aber. Es gibt verschiedene Varianten von Go-Back-N, je nach Quittungsverhalten des Empfängers in so einem Fall. Gemeinsam habe aber alle Varianten, dass der Sender alles ab dem fehlerhaften Rahmen neu übertragen muss.

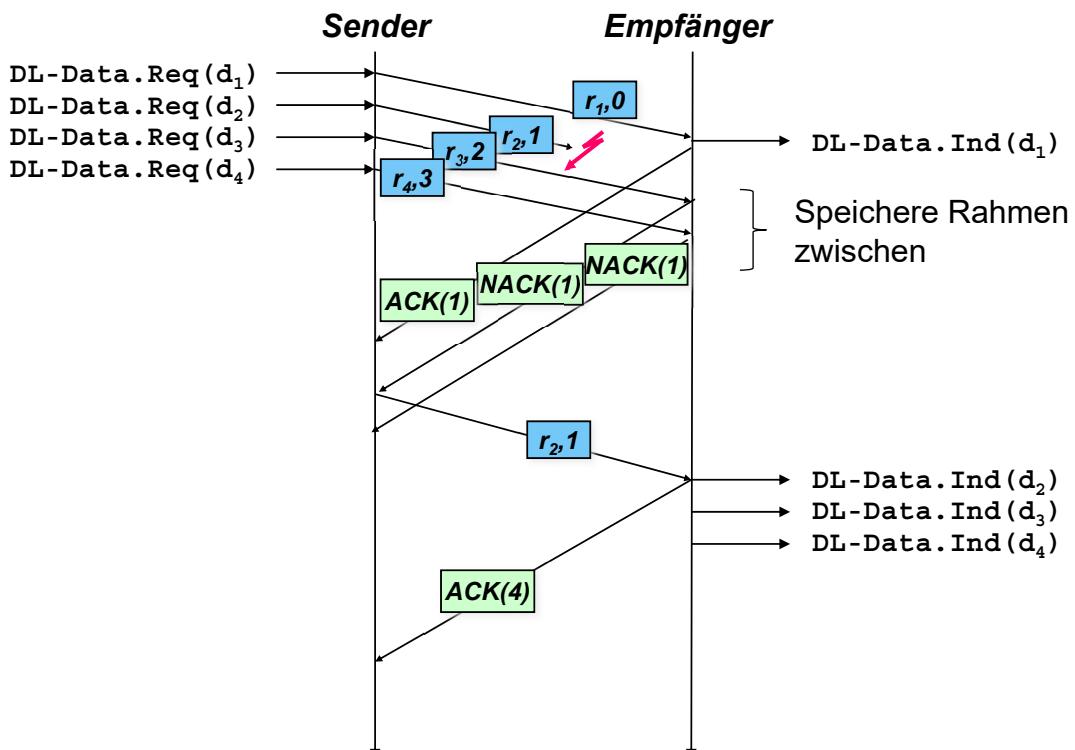
Varianten:

1. Auf der Folie dargestellt: bei Eintreffen eines Rahmens mit nicht erwarteter Nummer wiederholt der Empfänger die vorherige Bestätigung. Der Sender erkennt einen Rahmenverlust anhand einer doppelten ACK-PDU und wiederholt alles ab dem im ACK genannten Rahmen.
2. Durch eine NACK-PDU wird der Sender angewiesen, die Übertragung ab einer gewissen Rahmennummer erneut zu starten. Dieses Verfahren ist bis auf das unterschiedliche PDU-Format (NACK statt ACK) identisch mit Verfahren 1 (NACK(1) statt ACK(1) bei Erhalt von r_{3,2} im Beispiel auf der Folie).
3. Der Empfänger bestätigt ausschließlich korrekt empfangene Rahmen (in obiger Abbildung würde nur das erste ACK(1) gesendet, auf die nachfolgend eintreffenden Rahmen würde der Empfänger nicht reagieren). Der Sender erkennt Rahmenverluste nur durch Ablaufen eines Timeouts (fehlendes ACK) und muss nach dem zuletzt bestätigten Rahmen wieder aufsetzen.

Man sollte Go-back-N nur bei Leitungen mit einer geringen Latenz und/oder niedriger Datenrate einsetzen, damit nicht viele schon korrekt ausgelieferte Rahmen erneut übertragen werden müssen.

Des Weiteren bietet sich Go-Back-N an, wenn der Empfänger nicht viele Ressourcen zur Verfügung hat und daher nicht in der Lage ist, Rahmen zwischenzuspeichern, die nicht die erwartete Nummer tragen.

Fehlerbehandlung: Selective Repeat (Reject)



Selective Repeat

Bei diesem Verfahren wiederholt der Sender nur einzelne Rahmen. Allerdings muss der Empfänger die anderen Rahmen zwischenspeichern, um die Reihenfolgetreue zu erhalten (Empfängerbuffer wird benötigt).

Auch bei Selective Repeat gibt es zwei (bzw. drei) Ansätze:

1. *Explizite Übertragungswiederholung*: alle in korrekter Reihenfolge empfangenen Rahmen werden normal durch eine ACK-PDU bestätigt. Der Empfänger fordert fehlende/fehlerhafte Rahmen durch eine NACK-PDU erneut an. Diese Situation ist oben dargestellt. Wie bei Go-Back-N reagieren wir auf die negative Quittung mit einer Wiederholung, wiederholen allerdings nur den explizit als nicht korrekt empfangen gemeldeten Rahmen. Sobald dieser angekommen ist, schickt der Empfänger eine kumulative Quittung für alle bis zu diesem Zeitpunkt korrekt in Folge empfangenen Rahmen. (Wie auch bei Go-Back-N können auch positive Quittungen wiederholt werden, wenn keine NACK-PDU zur Verfügung steht. Es gibt also letztlich die gleichen drei Varianten wie bei Go-Back-N.)
2. *Implizite Übertragungswiederholung*: alle in korrekter Reihenfolge empfangenen Rahmen werden normal durch eine ACK-PDU bestätigt. Der Sender wiederholt nach einem Timeout für einen Rahmen nur diesen, der Sender schickt eine Quittung für diesen und alle folgenden bereits korrekt angekommenen Rahmen (kumulative Quittung). Bitte beachten: in obigem Beispiel würden unnötigerweise auch die Rahmen ab r_3 wiederholt werden, da es für diese kurz nach der Wiederholung von Rahmen r_2 auch Timeouts gibt. Wie viele Rahmen wir unnötigerweise wiederholen, hängt hier von der Latenz zwischen Sender und Empfänger ab, da die Wiederholungen abgebrochen werden können, sobald die kumulative Quittung eingeht. Daher ist die Verwendung von NACK-PDUs im Allgemeinen sinnvoller.

Flusskontrolle (Flow Control)

- **Synonyme Begriffe**

- ▶ Flusssteuerung
- ▶ Flussregulierung

- **Aufgabe**

- ▶ Der Empfänger wird vor einem zu großen Zufluss von Rahmen eines Senders geschützt

Flusskontrolle (Vermeidung des Überlaufens des Buffers des Empfängers)

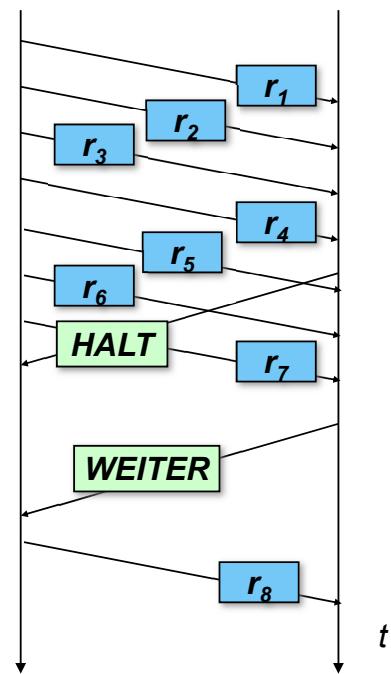
Schränkt man bei den vorherigen Verfahren nicht ein, wie viele unbestätigte Rahmen gleichzeitig unterwegs sein dürfen ohne quittiert worden zu sein, kann der Empfänger überlastet werden; auf Empfängerseite wird ein Buffer bereitgestellt, in dem empfangene Dateneinheiten zwischengespeichert werden, bis sie lokal verarbeitet werden können. Ist dieser Buffer voll, muss der Sender gezwungen werden, keine weiteren Rahmen mehr zu versenden, da diese beim Empfänger nicht zwischengespeichert werden können und man wiederum Rahmenverluste hat. Daher fügt man eine Beschränkung ein, um den Sender stoppen zu können. Dies nennt man Flusskontrolle oder auch Flusssteuerung: der Empfänger steuert, wie viel der Sender verschicken darf.

Flusskontrolle mit Halt-/Weiter-Meldungen

- **Einfachste Methode**

- ▶ Sender-Empfänger-Flusssteuerung
 - Meldungen
 - Halt
 - Weiter

Sender **Empfänger**



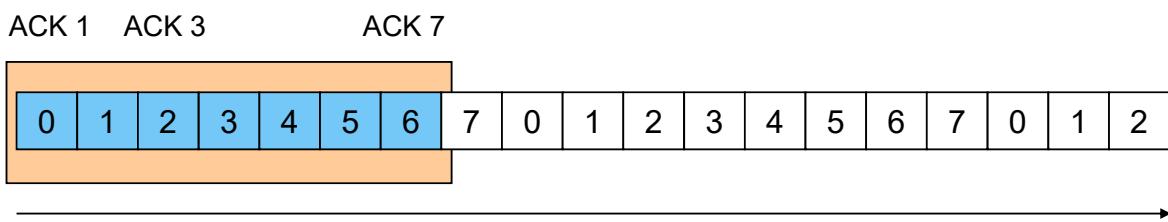
Die einfachste Methode der Flusskontrolle basiert auf zwei Meldungen:

- Mit der HALT-Meldung zeigt der Empfänger an, dass er nicht mit dem Sender Schritt halten kann und der Buffer volläuft. Diese Meldung sollte nicht zu spät gesendet werden, da der Sender bis zur Ankunft der HALT-Meldung noch weiter Rahmen senden kann.
- Mit der WEITER-Meldung wird später signalisiert, dass der Empfänger genügend Rahmen verarbeitet hat und so neue Rahmen annehmen kann.

Flusskontrolle: Sliding Window

- **Verwendung eines „Sendefensters“**

- ▶ Sender und Empfänger vereinbaren ein *Übertragungsfenster* (entspricht max. Größe des Pufferspeichers des Empfängers)
 - Sender nummeriert die zu versendenden Rahmen des Datenstroms im Bereich 0, 1, 2, ..., MODULUS-1, 0, ... mit $W < \text{MODULUS}$ durch
 - Fenstergröße W bedeutet: der Sender darf maximal W fortlaufend nummerierte Rahmen verschicken, ohne eine Bestätigung zu bekommen
 - Empfänger bestätigt empfangene Rahmen durch Quittungen (ACK)
 - Sender rückt Fenster vor, sobald ein ACK eintrifft, und darf neue Rahmen verschicken



Das gängige Verfahren zur Flusskontrolle ist Sliding Window: lege ein „Fenster“ über den zu versendenden Datenstrom, durch das der Sender nur einen Ausschnitt der Daten sehen kann. Nur diese Daten dürfen versendet werden, bevor der Sender auf eine Bestätigung warten muss. Die Menge der Rahmen, die ein Sender senden darf, ohne auf eine Bestätigung warten zu müssen, nennt man Fenstergröße. Die Fenstergröße sollte vom Empfänger eingestellt werden können, um dessen Buffergröße zu repräsentieren. Dadurch kann der Sender Netzkapazität wann immer möglich ausnutzen, während der Empfängerbuffer nicht überlastet wird.

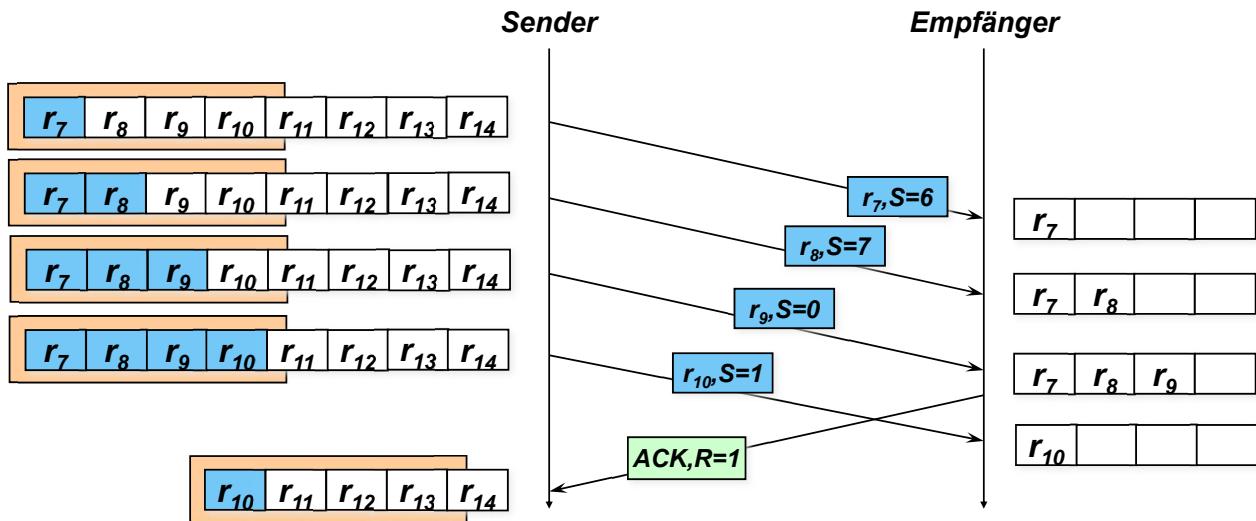
Ist die Fenstergröße statisch vorgegeben, kann der Empfänger lediglich Bestätigung für Rahmen verzögern, um nicht überlastet zu werden. Es ist jedoch nicht vorteilhaft, wenn während dieser Verzögerung der Timer beim Sender abläuft und dieser seine Rahmen wiederholt, obwohl sie beim Empfänger schon korrekt ankamen. Für solche Fälle gibt es bei manchen Protokollen eine Meldung wie RECEIVE NOT READY, welche die letzten Rahmen bestätigt, aber mitteilt, dass der Empfänger noch eine Weile braucht, um seine Buffer zu leeren.

Die Wahl der Fenstergröße wird dadurch beschränkt, dass im Header des Rahmens nur eine bestimmte Zahl n an Bit für die Sendefolgenummern reserviert werden und es daher nur 2^n unterschiedliche Sendefolgenummern gibt. Wird das Fenster zu groß gewählt, können zwei unbestätigten Rahmen mit gleicher Nummer unterwegs sein, so dass Quittungen nicht mehr eindeutig wären.

Sliding Window wird auch kreditbasierte Flusskontrolle genannt: der Sender hat Kredit für die Versendung einer bestimmten Zahl von Rahmen, bevor er auf eine Quittung warten muss.

Sliding Window

- Sliding Window mit Fenstergröße 4 für eine Senderichtung



S: Sequenznummer (des zuletzt gesendeten Rahmens)

R: Nächste erwartete Sequenznummer = Quittierung bis Folgenummer R-1

Beispiel: HDLC-Protokoll

• High-Level Data Link Control (HDLC)

- ▶ Bit-orientiertes Sicherungsschichtprotokoll
 - Anwendungszenario: Anschluss an Datennetze über Telefonnetze (X.25)
- ▶ Funktionen:
 - Framing durch spezielle Bitkombination, Codetransparenz durch Bitstuffing in Payload und CRC-Prüfsumme
 - Fehlerkorrektur durch CRC + ARQ, Piggybacking von Quittungen
 - Flusskontrolle (Sliding Window)
- ▶ Varianten zum HDLC-Protokoll:
 - LAPD (ISDN, Link Access Procedure, D-Kanal)
 - *LLC (IEEE 802.2, Logical Link Control)*
 - *PPP (Point-to-Point Protocol)*

Auf den folgenden Folien wird das HDLC-Protokoll vorgestellt, welches sehr simpel ist, aber alle bisher vorgestellten Aspekte der Sicherungsschicht implementiert und sich daher gut eignet, die Umsetzung zu erklären.

HDLC kam als ein Standardprotokoll in X.25-Netzen zum Einsatz, die ursprünglich von Telefonnetzbetreibern verwendet wurden, um Datenaustausch über Telefonnetze zu ermöglichen. X.25 wurde nach und nach von Frame Relay abgelöst, dieses dann von ATM und dieses wiederum von SDH – so dass HDLC heute vermutlich kaum noch im Einsatz ist. Varianten dieses Protokolls haben sich allerdings gehalten – beispielsweise als PPP, welches Internetprovider für die Einwahl der Kunden verwenden. Mit Hilfe von PPP teilt der Provider dem Rechner/Router des Kunden wichtige Daten mit, z. B. dessen IP-Adresse oder den zu verwendenden DNS-Server. Zunächst wurde PPP über Modem- oder ISDN-Verbindungen verwendet, heutzutage auch zum Datenaustausch über GPRS-/UMTS-Mobilfunkdatenverbindungen oder DSL-Verbindungen.

HDLC: Medienzugriff

- **HDLC baut auf einem Befehl/Antwort-Schema auf:**
 - ▶ *Leitsteuerung*: Aussenden eines Befehls
 - ▶ *Folgesteuerung*: Reaktion (Meldung) auf eine Leitsteuerung
- **Wer kann Leitsteuerung aussenden?**
 - ▶ HDLC definiert Leitstationen und Folgestationen
 - ▶ Asymmetrischer Fall mit zentraler Steuerung
 - Eine feste Leitstation, ein oder mehrere Folgestationen
 - Leitstation versendet Empfangsauftrag und Sendeaufrufe, kein spontaner Zugriff einer Folgestation auf das Medium
 - ▶ Symmetrischer Fall mit gleichberechtigten Stationen
 - Hybridstationen alternieren als Leit- und Folgestationen

(Ausgeblendete Folie – nur informativ, nicht wichtig für Vorlesung)

Der einzige bei HDLC implementierte Mechanismus, der noch nicht zuvor erläutert wurde, ist der Medienzugriff. Auch dieser ist bei HDLC einfach gehalten, bietet aber unterschiedliche Varianten. Generell basiert der Medienzugriff allerdings auf einer Koordination durch eine zentrale Station, die Leitsteuerung genannt wird. Alle anderen Stationen sind Folgestationen, die lediglich auf Sendeaufrüfe der Leitsteuerung reagieren (Polling).

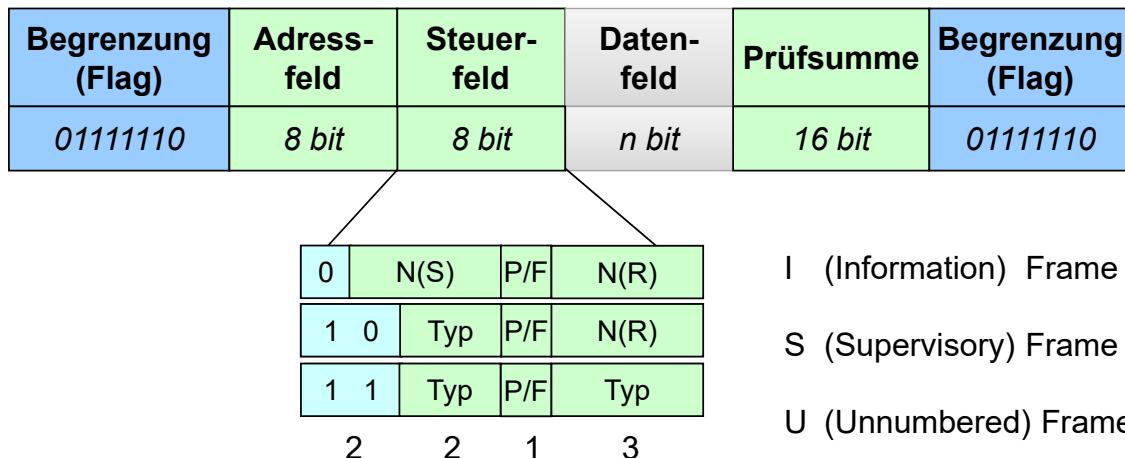
HDLC erlaubt mehrere Arbeitsbetriebsarten:

- *Aufforderungsbetrieb* (NRM — Normal Response Mode): entspricht Polling: eine Folgestation darf nur nach ausdrücklicher Erlaubnis durch die Leitstation Meldungen senden.
- *Spontanbetrieb* (ARM — Asynchronous Response Mode): eine Folgestation kann jederzeit Meldungen an Leitstation senden.
- *Gleichberechtigter Spontanbetrieb* (ABM — Asynchronous Balanced Mode): Beide Hybridstationen dürfen jederzeit Meldungen und Befehle übermitteln.

HDLC: Rahmenaufbau

- **Eigenschaften:**

- ▶ Synchronisation durch Rahmenbegrenzer **0111110**
 - Sequenz darf nicht im Rahmen vorkommen → Bitstuffing
- ▶ Quittierung, Flusskontrolle → Sequenz-/Bestätigungsnummern
- ▶ Fehlererkennung durch CRC → Prüfsumme



Zum Austausch von Daten definiert HDLC ein einheitliches Rahmenformat sowohl für den Datenaustausch als auch für Kontrollkommandos. HDLC unterscheidet dazu drei unterschiedliche Rahmentypen, die sich anhand der ersten Bits des Steuerfeldes unterscheiden lassen:

- Information (I-Frames): Datenübertragung
- Supervisory (S-Frames): Fehler-/Flusskontrolle
- Unnumbered (U-Frames): Verbindungsauf-/abbau

Neben den Sequenz-/Quittungsnummern N(S) und N(R) gibt es ein Typ-Feld, welches verschiedene Steuerbefehle unterscheidet (z.B. RR - Receive Ready) sowie das Poll/Final-Bit (P/F).

Da Rahmenbegrenzungen verwendet werden, muss auch Bitstuffing eingesetzt werden: dazu wird nach einer Folge von fünf Einsen immer eine Null eingefügt.

HDLC: Aufbau des Steuerfeldes

Steuerfeldformat für	Bit-Nummer							
	1	2	3	4	5	6	7	8
I-Rahmen (Nutzdatenrahmen) [I=Information]	0		N(S)		P/F		N(R)	
S-Rahmen (Steuerrahmen) [S=Supervisory]	1	0	S	S	P/F		N(R)	
U-Rahmen (Steuerrahmen) [U=Unnumbered]	1	1	M	M	P/F		M	M

- ▶ **I-Block:** Übertragung von Nutzdaten, N(S) ist aktuelle Rahmennummer
Piggybacking von Quittungen über N(R) möglich
- ▶ **S-Block:** Steuerrahmen mit Quittungsnummer N(R)
S-Bits des Typ-Felds spezifizieren den Rahmen, verschiedene Bestätigungen sowie Flusskontrolle implementiert
- ▶ **U-Block:** Steuerrahmen ohne Nummer
- ▶ **P/F Bit:** Übergibt Sendeberechtigung zwischen Leit- und Folgestationen

Das P/F-Bit dient dazu, einer anderen Station anzuzeigen, dass die Datenübertragung in ihre Richtung beendet ist. Grund hierfür ist, dass Rahmen nicht beliebig lang sein sollten – ein Bitfehler innerhalb eines Rahmens führt dazu, dass der Rahmen vom Empfänger verworfen wird, weshalb es in Abhängigkeit von der BER eines Mediums sinnvoll ist, die Nutzdaten auf mehrere Rahmen aufzuteilen. Sendet die Leitsteuerung nun eine Folgen von Rahmen an eine Folgestation, setzt sie im letzten Rahmen das P/F-Bit auf 1. Dies hat für die Folgestation die Bedeutung „Poll“, d.h. sie darf nun senden. Auch sie kann nun mehrere Rahmen an die Leitsteuerung senden und setzt im letzten Rahmen das P/F-Bit auf 1, was für die Leitsteuerung die Bedeutung „Final“ hat, d.h. nun ist sie wieder an der Reihe.

HDLC: Rahmentypen

Type	Name	Fields							
		1	2	3	4	5	6	7	8
I	I (Data Frame)	0		N(S)	P		N(R)		
S	RR (Receive Ready)	1	0	0	0	P/ F		N(R)	
	RNR (Receive not Ready)	1	0	1	0	P/ F		N(R)	
	REJ (Reject)	1	0	0	1	P/ F		N(R)	
	SREJ (Selective Reject)	1	0	1	1	P/ F		N(R)	
U	SABM (Set Asynchronous Balanced Mode)	1	1	1	1	P	1	0	0
	DISC (Disconnect)	1	1	0	0	P	0	1	0
	UA (Unnumbered ACK)	1	1	0	0	F	1	1	0
	CMDR (Command Reject)	1	1	1	0	F	0	0	1
	FRMR (Frame Reject)	1	1	1	0	F	0	0	1
	DM (Disconnect Mode)	1	1	1	1	F	0	0	0

Sende Rahmen $N(S)$, bestätige $N(R) - 1$ in die andere Richtung (Piggybacking)

Signalisierung „empfangsbereit“, bestätige $N(R) - 1$ in die andere Richtung (ACK, WEITER)

Signalisiere „temporär nicht empfangsbereit“, bestätige $N(R) - 1$ in die andere Richtung (HALT)

NACK für $N(R)$, ACK bis $N(R)-1$. Anforderung einer Übertragungswiederholung ab $N(R)$ (Go-back-N)

ACK bis $N(R)-1$, NACK nur für $N(R)$

Verbindaufbau

Ankündigung eines Verbindungsabbaus

Generelles ACK (z.B. im Verbindaufbau)

Rahmen/Kommado ungültig (falsches Rahmenformat, ungültige Sequenznummer, ...)

Verbindungsabbau

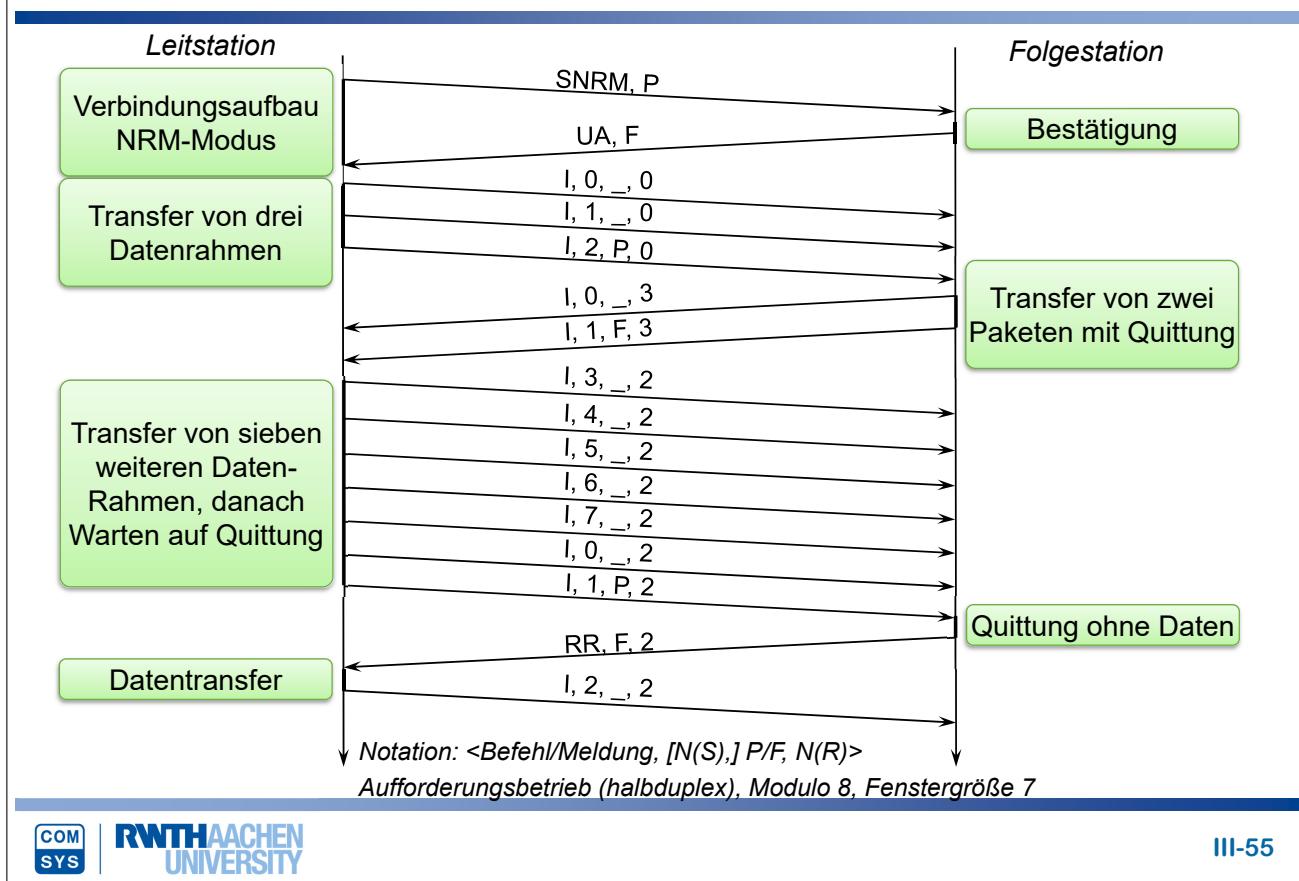
Bitte beachten: die Liste der U-Frames ist nicht vollständig, sondern soll einfach ein paar Beispiele zeigen, was für Kommandos gegeben werden können.

Siehe auch http://de.wikipedia.org/wiki/High-Level_Data_Link_Control.

Wir sehen hier ein paar Mechanismen, die zuvor erläutert wurden:

- Quittungsmechanismus mit ACK und NACK
- RNR entspricht einer „HALT-Meldung“

HDLC: Beispielablauf



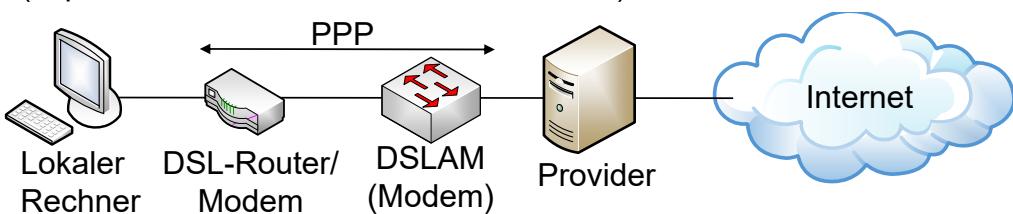
PPP (Point-to-Point Protocol)

- **Großteil des Internets: Punkt-zu-Punkt Verbindungen**

- ▶ Verbindungen im WAN zwischen Routern
- ▶ Heimanbindung über Modem und Telefonleitung (DSL)

- **PPP (Vereinfachte Version von HDLC)**

- ▶ Effiziente Anpassung an verschiedene PHYs + unterstützt diverse Schicht-3-Protokolle und deren Optionsverhandlung
 - Steuerprotokoll (LCP, Link Control Protocol) zum Verbindungsauftbau, Verbindungstest, Verbindungsverhandlung, Verbindungsabbau
 - Verhandlung von Schicht-3-Optionen unabhängig vom Schicht-3-Protokoll (separates Network Control Protocol, NCP)



„Einloggen“ über Modem bzw. Terminal-Emulation auf einem Server ergibt lediglich (textorientierte) Terminal-Funktionalität und eröffnet nicht die gesamte Anwendungsbreite des Internets (WWW, FTP etc.), da der eigene Rechner nicht als vollwertiger Internet-Host auftritt. Dies wurde erst durch die Protokolle SLIP (Serial Line IP, standardisiert als RFC 1055) und PPP (Point-to-Point Protocol, standardisiert als RFC 1661/1662) ermöglicht, die die transparente Übertragung von IP-Paketen erlauben.

Das aus den 80er Jahren stammende SLIP hatte einige Nachteile (keine Fehlererkennung, nur IP, keine dynamische Adresszuweisung, keine Authentifikation), so dass es 1994 durch das leistungsfähigere PPP ersetzt wurde.

Typisches Szenario beim Zugriff eines PCs auf das Internet via Modem

- Anruf beim Internetprovider via Modem und Aufbau einer physikalischen Verbindung
- Anrufer sendet mehrere LCP-Pakete im PPP-Rahmen zur Auswahl der gewünschten PPP-Parameter, Authentifizierung
- Austausch von NCP-Paketen, um Vermittlungsschicht zu konfigurieren; z.B. kann hier dynamisch eine IP-Adresse zugewiesen werden
- Der Anrufer kann nun genauso wie ein festverbundener Rechner Internet-Dienste nutzen
- Zur Beendigung der Verbindung wird via NCP die IP-Adresse wieder freigegeben und die Vermittlungsschichtverbindung abgebaut
- Über LCP wird die Schicht 2-Verbindung beendet, schließlich trennt das Modem die physikalische Verbindung

PPP – Rahmenformat

- Rahmenformat an HDLC angelehnt

1	1	1	1 oder 2	variabel	2 oder 4	1	Byte
Flag 01111110	Address 11111111	Control 00000011	Protocol	Payload	Checksum	Flag 01111110	

- ▶ Zeichenorientiert (Länge des Payload endet immer an Byte-Grenze)
- ▶ Codetransparenz durch Character Stuffing
- ▶ Typischerweise nur *Unnumbered*-Frames, bei hohen Fehlerraten (Mobilkommunikation) kann jedoch auch der zuverlässigeren Modus mit Sequenznummern und Bestätigungen gewählt werden
- ▶ „Protocol“ definiert Art des Payloads, z.B. IPv4, IPv6
- ▶ Falls nicht anderweitig verhandelt, ist die maximale Länge der Nutzlast auf 1500 Byte begrenzt

PPP hat im wesentlichen das Rahmenformat von HDLC übernommen (Kompatibilität). Es werden die gleichen Flags zur Kennzeichnung des Rahmenbeginns und -endes verwendet. Das Adressfeld ist immer auf 255 gesetzt, das Steuerfeld (Control) hat üblicherweise den Wert 3 (unnummerierte Datenübertragung). Die Checksumme kann wie HDLC 2 Byte lang sein, ist heutzutage aber 4 Byte lang (wie bei allen modernen Protokollen).

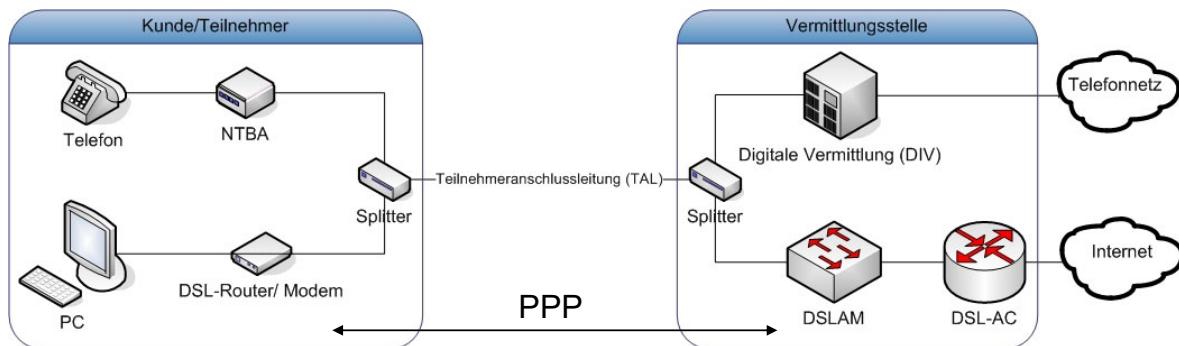
Zudem wird hinter dem Steuerfeld ein Feld eingefügt, welches dem Empfänger anzeigt, welches höhere Protokoll im Datenteil verwendet wird, damit der Empfänger weiß, mit welcher Protokollinstanz er die Daten weiterverarbeiten muss.

Durch zusätzliche Verhandlung kann der Header verkleinert werden.

PPP und DSL: Systemaufbau

• Komponenten des DSL-Systems

- ▶ **DSL-Modem** für alle Unterkanäle
- ▶ **Splitter** zur Trennung des Telefonkanals (bis 120 kHz) vom Datenkanalbereich (ab 138 kHz)
- ▶ **DSLAM** (Digital Subscriber Line Access Multiplexer) als Modembank zum Anschluss mehrerer Haushalte beim DSL-Anbieter, misst Kanalqualitäten



Das DSL-Grundprinzip ist das gleiche wie beim ursprünglichen Modem. Auf Seiten des Heimnetzes wird das DSL-Modem die zu übertragenen Daten auf die verwendbaren Unterkanäle aufteilen und modulieren. Auf der Seite des Providers existiert für jeden Haushalt eine Gegenstelle, d.h. auch ein Modem. Die Gesamtheit der Modems formt den DSLAM, der die Daten aller Haushalte demoduliert und auf eine gemeinsame Leitung zur Weiterleitung ins Internet multiplext.

Um parallel Telefonie über die Anschlussleitung zu ermöglichen, wurde der untere Bereich des Frequenzbands freigelassen (POTS/ISDN). Ein sogenannter Splitter trennt bzw. vereint die nieder- und die hochfrequenten Anteile der übertragenen Signale.

Dieser Splitter ist heutzutage meist nicht mehr nötig, da die Sprachübertragung für das Telefon bei den Providern auch über IP läuft und die Sprach-Pakete wie alle anderen Datenpakete an das Modem übertragen und erst dort getrennt werden.

Um einen Kommunikationskanal zwischen den Modems auf beiden Seiten einzurichten, wird PPP eingesetzt.

Bilder Telefon"verkabelung" und DSLAM



Bilder Telefon"verkabelung" in New Delhi



DSL-Varianten

- **Symmetrische Varianten**

- ▶ High Data Rate Digital Subscriber Line (HDSL)
- ▶ Symmetric Digital Subscriber Line (SDSL)

- **Asymmetrische Varianten**

- ▶ Asymmetric Digital Subscriber Line (ADSL)
 - Frequenzband bis 1,1 MHz (Up: 1 Mbit/s, Down: 8 Mbit/s)
- ▶ ADSL2/ADSL2+
 - Frequenzband bis 2,2 MHz (Up: 1 Mbit/s, Down: 24 Mbit/s)
- ▶ Very High Data Rate Digital Subscriber Line (VDSL)
 - Frequenzband bis 12 MHz (Up: 3,5 Mbit/s, Down: 25 Mbit/s)
- ▶ VDSL2
 - Frequenzband bis 30 MHz (Up/Down: bis zu 200 Mbit/s)

(Ausgeblendete Folie)

Je nach Aufteilung der Unterkanäle kann man symmetrische Varianten umsetzen (gleiche Datenrate in beiden Richtungen) oder asymmetrische Varianten (höhere Datenrate im Downstream). Asymmetrische Varianten werden typischerweise von den DSL-Anbietern angeboten, da Privatnutzer typischerweise mehr Daten herunter- als hochladen.

Die Verbesserungen der neuen Varianten werden durch Verwendung einer größeren Bandbreite erreicht – VDSL ist über alte Telefonverkabelung dabei aber nicht mehr zu realisieren. Die Distanz zur Vermittlungsstelle muss drastisch verkürzt werden, wozu die DSL-Anbieter Outdoor-DSLAMs installieren, die die Entfernung zu den Privathalshalten auf wenige hundert Meter reduzieren. Mittlerweile werden die DSLAMs sogar innerhalb von Privathäusern selbst installiert und von dort mit Glasfaserkabel mit den Standorten der Provider verbunden, um die Übertragungsstrecke über Telefonkabel noch weiter zu reduzieren und somit die nutzbare Bandbreite noch einmal drastisch zu erhöhen.

Kapitel 3: Sicherungsschicht

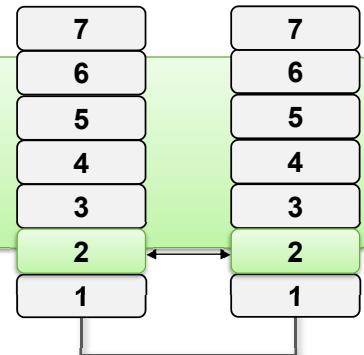
- **Rahmenbildung**
 - ▶ Blockbildung, Codetransparenz durch Character/Bit Stuffing
- **Fehlererkennung/-behandlung und Flusskontrolle**
 - ▶ Fehlererkennende Codes (CRC)
 - ▶ Proaktive und reaktive Fehlerbehebung (FEC und ARQ)
 - ▶ Flusskontrolle durch Sliding Window
 - ▶ Beispielprotokoll: HDLC

• Medienzugriff

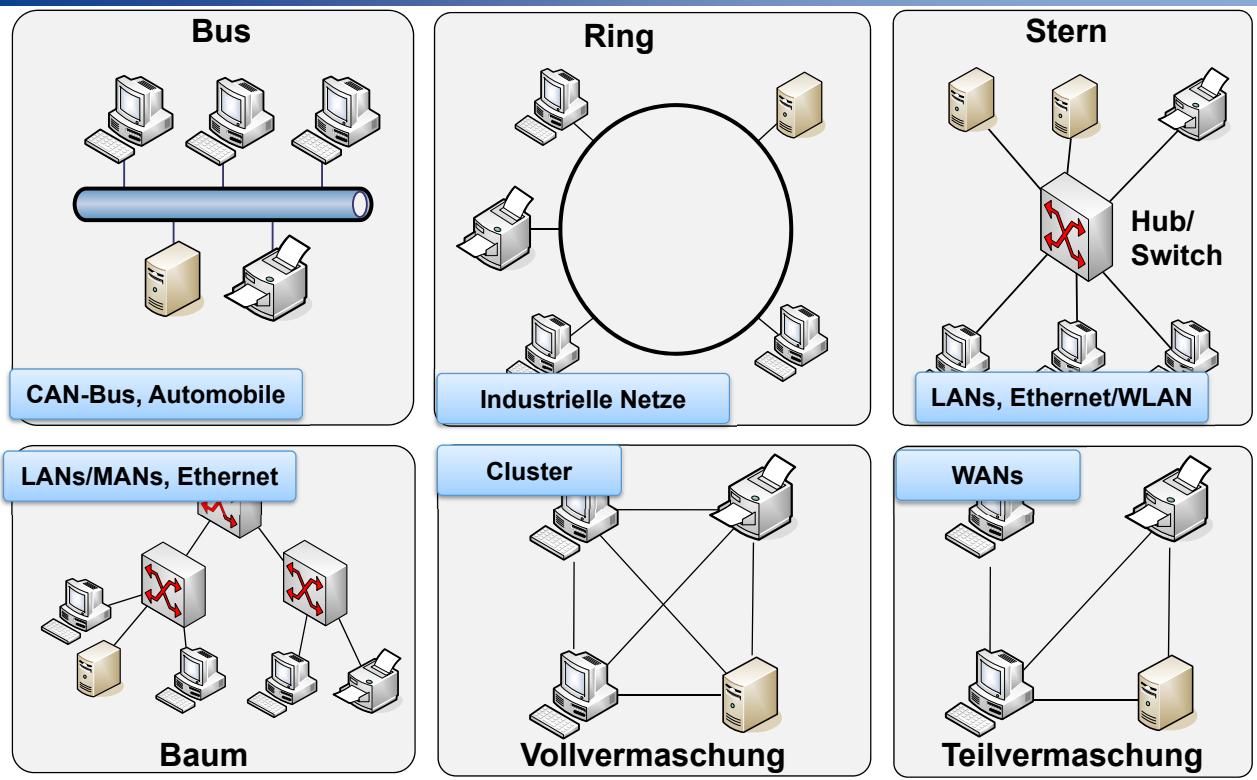
- ▶ Topologien, Shared Medium
- ▶ Token Passing, CSMA/CD, CSMA/CA

• Standards für lokale Netze

- ▶ Ethernet



Verbindungstopologien



LANs (lokale Netze) haben nur eine geringe Ausdehnung von wenigen zehn Metern bis zu sehr wenigen Kilometern. Dadurch sind aber sehr große Datenraten im Bereich von 100 Mbit/s bis 10 Gbit/s möglich. LANs werden meist von privaten Organisationen/Firmen und Privatleuten betrieben. Ihre Ausdehnung ist deshalb meist auf ein (Privat-) Grundstück beschränkt. Man verwendet klassischerweise meist Busse, Sterne oder Ringe, um schnelle Erreichbarkeit zwischen jedem Paar von Rechnern zu ermöglichen. Heutzutage sind vorwiegend Sterne oder Bäume (kaskadierende Sterne) im Einsatz.

Ein MAN (Nahverkehrsnetz) erstreckt sich meist über das Gebiet einer Stadt (oder eines Campus'). Ein MAN soll bei sehr vielen angeschlossenen Knoten (bzw. LANs) auch eine sehr hohe Übertragungsrate und sehr geringe Ausfallzeiten garantieren. Eingesetzt werden meist Ringe und Bäume.

WANs (Weitverkehrsnetze) sind in ihrer Ausdehnung nicht begrenzt. Sie werden meist von staatlichen Einrichtungen oder TK-Gesellschaften betrieben. In vielen Fällen sind die einzelnen WANs auf Staatsgrenzen beschränkt, da jede Gesellschaft den allgemeinen Standard etwas anders benutzt. Trotzdem ist es möglich, länderübergreifend zu kommunizieren (über bestimmte Übergangsknoten). Ein WAN besteht meist aus einem teilvermaschten Netz, da Leitungen bedarfsgesteuert verlegt werden; oft kann man aber auch (hierarchische) Ringstrukturen erkennen.

Vollvermaschung ist in Rechnernetzen nicht üblich – dieses Konzept bietet sich bei Rechenclustern an, um direkten schnellstmöglichen Datenaustausch ohne Kollisionsrisiko zwischen jedem einzelnen Rechnerpaar zu ermöglichen.

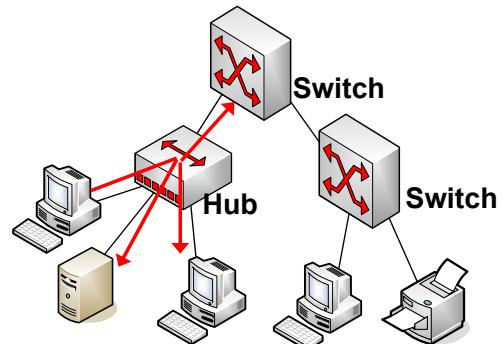
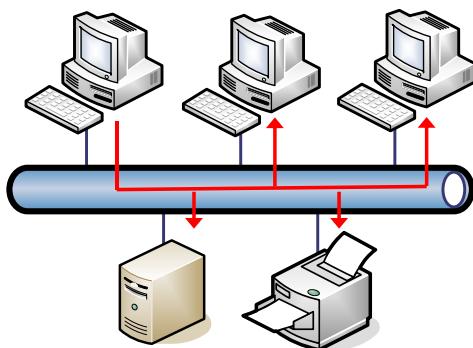
Einen Überblick über die Topologien samt Vor-/Nachteilen bietet [https://de.wikipedia.org/wiki/Topologie_\(Rechnernetz\)](https://de.wikipedia.org/wiki/Topologie_(Rechnernetz)).

Broadcast-Netze

- Bus, Ring und eventuell Stern und Baum sind

Broadcast-Netze

- ▶ Geteiltes Medium (Shared Medium)
- ▶ Sendet eine Station, erhalten alle anderen Stationen die Signale
 - Ein Netzsegment ist eine *Kollisionsdomäne*
- ▶ Gleichzeitiges Senden zweier Stationen führt zu einer *Kollision*
 - Regelung des Medienzugriffs nötig



Bus und Ring sind sogenannte Broadcast-Netze – sendet eine Station auf dem Medium, erhalten alle angeschlossenen Stationen die Signale. Diese Zustellung der Daten an alle nennt man „Broadcast“. Der Stern (und auch der Baum) sind Broadcast-Netze, wenn als zentrale Komponente ein Hub eingesetzt wird.

Voll- und Teilvermaschte Netze beinhalten nur direkte Verbindungen zwischen genau zwei Rechnern/Netzkomponenten – daher sind dies keine Broadcast-Netze, wenn das Medium vollduplex ist. Ist es nur halbduplex, hat man eine Richtungskonkurrenz. (Ein Beispiel hierfür ist WLAN: Laptop und Access Point kommunizieren auf einem Funkkanal miteinander.)

Ein Broadcast-Netz (und auch ein Halubduplex-Kanal) bildet eine Kollisionsdomäne: es wird ein geteiltes Medium verwendet und wenn zwei oder mehr Stationen gleichzeitig senden, tritt eine Kollision auf. Die Signale überlagern sich auf dem Medium, es kann keine Übertragung mehr identifiziert bzw. korrekt dekodiert werden.

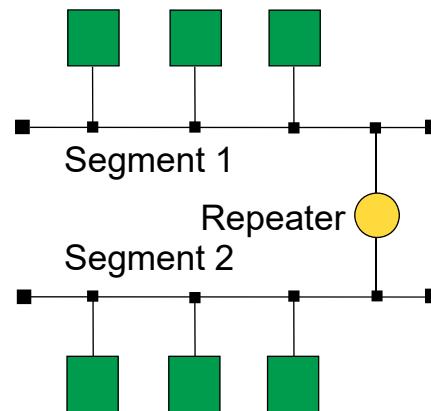
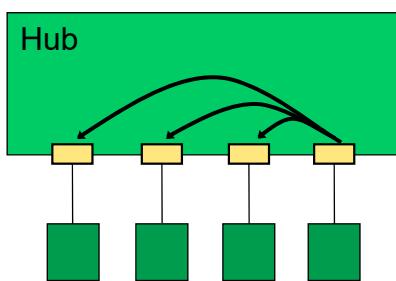
Durch Komponenten wie Switches werden Kollisionsdomänen aufgetrennt: ein Switch kennt die Adressen der angeschlossenen Stationen und kann Rahmen gezielt weiterleiten.

Daher ist in solchen Netzen eine Regelung des Medienzugriffs nötig. Welche Zugriffsregelungen Sinn machen, hängt stark davon ab, wie die Rechner untereinander verbunden sind (Topologie).

Zentrale Komponente beim Stern: Hub

- Nur Bitübertragungsschicht

- ▶ Empfang und Auffrischung von Signalen (wie Repeater)
 - Kann auch zur Vergrößerung des Netzes verwendet werden
- ▶ Keine Unterbrechung der Kollisionsdomäne
 - Hub broadcastet empfangenes Signal auf allen anderen Ports
 - Nur eine Station zur Zeit kann senden



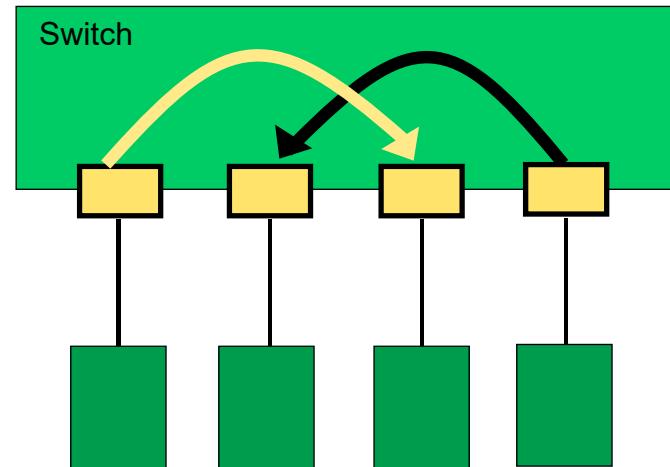
Die erste Komponente, die als zentrale Einheit im Stern eingesetzt wurde, war der Hub. Wie auch die Repeater hat er eine einfache Funktionalität: er empfängt auf einem Port (= Anschluss) Daten und leitet sie auf allen anderen weiter. Da die Signale auf dem empfangenen Port abgetastet und auf den anderen Ports neu erzeugt werden, nimmt er eine Signalauffrischung vor. Damit arbeitet der Hub auf der Bitübertragungsschicht.

Wird ein Hub als zentrale Komponente einer Sterntopologie (oder in einem Baum) eingesetzt, hat man ein Broadcast-Netz und damit die gleichen Probleme wie bei Bus und Ring.

Zentrale Komponente: Switch

- Bitübertragungs- und Sicherungsschicht

- ▶ *Punkt-zu-Punkt-Verbindungen* zwischen je zwei Stationen
- ▶ Lernt die Adressen angeschlossener Stationen
- ▶ Puffer für jeden Port
 - Trennt Kollisionsdomänen
 - Keine Kollisionen mehr!
- ▶ Höherer Durchsatz als Hub
- ▶ Realisiert durch hochratigen internen Bus



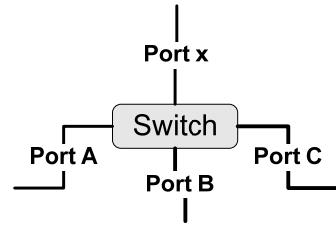
Der Switch wird mittlerweile (hoffentlich) überall als zentrale Komponente eingesetzt. Er verwaltet eine Weiterleitungstabelle und lernt die Adressen der angeschlossenen Stationen. Er kann auch zur Kopplung ganzer Netzsegmente eingesetzt werden (allerdings ohne Protokollwandlung, alle Segmente müssen das gleiche Protokoll einsetzen!). Für eine zusätzliche Protokollwandlung würde man eine sogenannte Brücke benötigen), wird aber üblicherweise zur Kopplung einzelner Stationen eingesetzt – als Resultat können keine Kollisionen mehr auftreten und verschiedene Paare von Stationen können simultan Daten austauschen. Der Gesamtdurchsatz des Netzes wird gegenüber dem Hub deutlich erhöht.

Achtung, Falle: im heutigen Sprachgebrauch wird sehr viel als Switch bezeichnet, was eigentlich gar kein Switch mehr ist. Ein sogenannter „Layer-3-Switch“ ist ein Switch, der zusätzlich zu den hier dargestellten Funktionen auch über eine integrierte Routing-Funktionalität verfügt (siehe nächstes Kapitel). Wenn im weiteren Verlauf der Vorlesung/Übungen von „Switch“ die Rede ist, ist stets die eigentliche, hier dargestellte Ursprungsvariante gemeint.

Aufbau von Weiterleitungstabellen

- Automatisches Lernen von Adressen angeschlossener Stationen

Rahmen kommt auf Port x an:



Existiert Weiterleitungeintrag für Ziel in der Tabelle?

Vorhanden und $\neq x$?

Leite Rahmen auf entsprechendem Port weiter

Vorhanden und $= x$?

Verwerfe Rahmen

Nicht vorhanden?

Broadcaste Rahmen, d.h. sende ihn auf allen Ports außer x

Ein Switch verwaltet eine Weiterleitungstabelle, in der festgehalten wird, welche Rechner (bzw. Netzwerkkarten mit bestimmten MAC-Adressen) sich hinter welchem Anschluss (Port) befinden. In der Tabelle wird bei jedem Eintrag neben der MAC-Adresse und dem zugehörigen Port auch das Alter des Eintrags festgehalten.

Die Weiterleitungstabelle des Switches muss nicht manuell konfiguriert werden. Statt dessen lernt der Switch, welche Stationen angeschlossen sind, im laufenden Betrieb. Empfängt der Switch einen Rahmen, schaut er sich zunächst die Zieladresse des Rahmens an und durchsucht seine bisherige Weiterleitungstabelle, ob er einen Eintrag für die entsprechende Adresse hat. Ist dies der Fall, wird der Rahmen über den im Eintrag angegebenen Port weitergeleitet. Eine Ausnahme ist, dass der Rahmen über den Port weitergeleitet werden müsste, über den der Switch ihn empfangen hat – in diesem Fall hat sich der Rahmen bereits in dem Segment, in dem das Ziel zu finden ist, ausgetragen und braucht nicht noch einmal ausgesendet zu werden.

Ist kein Eintrag vorhanden, wird der Rahmen auf allen Ports (außer dem, über den er empfangen wurde) weitergeleitet, damit er auf jeden Fall beim Ziel ankommt.

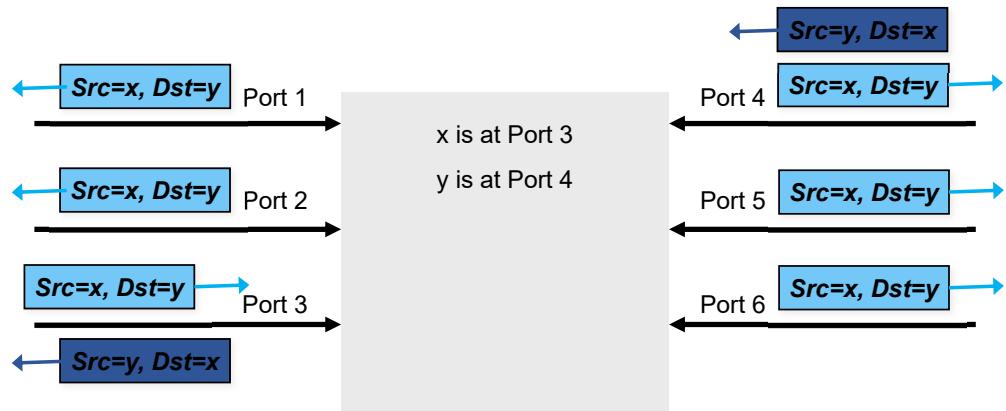
Gleichzeitig untersucht der Switch auch bei jedem Rahmen, ob er einen neuen Weiterleitungeintrag anlegen muss, indem er sich die Absenderadresse des Rahmens anschaut. Existiert zu dieser Adresse bereits ein Eintrag, wird das Alter des Eintrags zurückgesetzt. Ist im Eintrag ein anderer Port abgespeichert als der, über den der aktuelle Rahmen empfangen wird, wird auch diese Angabe aktualisiert. Existiert noch kein Eintrag, wird ein neuer Eintrag mit der Absenderadresse und dem Port, auf dem der Rahmen empfangen wurde, angelegt.

Das Alter eines Eintrags wird mitgeführt, damit veraltete Einträge nach bestimmter Zeit automatisch gelöscht werden.

Lernen von Adressen

- Daten für eine unbekannte Adresse auf einem Port empfangen

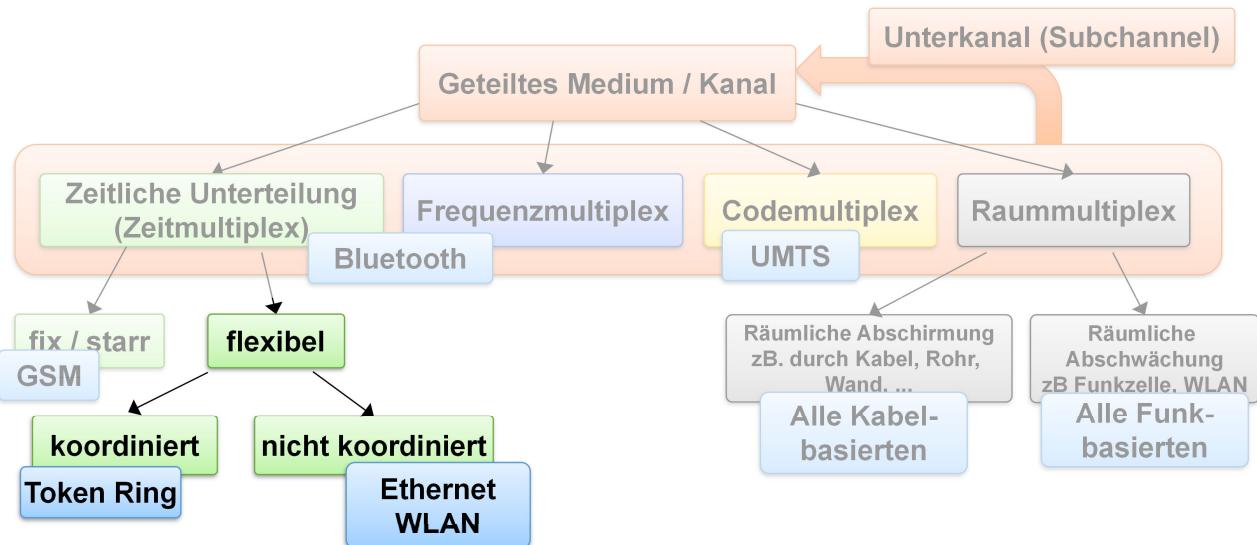
- ▶ Broadcast des Rahmens auf alle anderen Ports
- ▶ Absenderadresse des Rahmens kann für den Port, über den er empfangen wurde, gespeichert werden



Medienzugriff

• Problem:

- Mehrere Stationen als Dienstnutzer eines einzigen physikalischen Mediums (*Shared Medium*) können *Kollisionen* verursachen



Sind mehrere Stationen an ein physikalisches Medium angeschlossen (z.B. Twisted Pair, Koaxialkabel, aber auch ein gemeinsames Frequenzband bei drahtlosen Netzen), muss geregelt werden, wer das Medium zu einem bestimmten Zeitpunkt belegen darf. Die Grundlagen der Verfahren *Frequenzmultiplex (FDM)*, *Zeitmultiplex (TDM)* und *Codemultiplex (CDM)* wurden schon in Kapitel 2 (Bitübertragungsschicht) vorgestellt. Diese Verfahren kann man nutzen, um einen physikalischen Kanal statisch auf mehrere Stationen aufzuteilen. Eine Kombination aus FDM und synchronem TDM wird beispielsweise bei GSM eingesetzt, CDM bei UMTS – Netze, bei denen eine garantierter Datenrate (zur Telefonie) bereitgestellt werden soll.

Der Nachteil eines statischen Multiplexings ist allerdings, dass nicht auf Änderungen in der benötigten Datenrate einer Station eingegangen werden kann. Internet-Verkehr ist oft *burstartig*, d.h. die Übertragung von Daten erfolgt in unregelmäßigen Zeitintervallen, aber dann wird direkt eine große Menge an Daten auf einmal übertragen. Wir benötigen also die Möglichkeit, spontan bei Bedarf quasi beliebige Mengen an Daten übertragen zu können, während ein Multiplexing uns dauerhaft eine konstante Datenrate zur Verfügung stellt.

Die Lösung ist ein dynamisches Multiplexing. Vor allem für LANs (aber auch für MANs) existiert eine Reihe von Varianten des *asynchronen Zeitmultiplex* (asynchron=keine feste Einteilung in Rahmenstruktur bzw. Zeitschlitz), die im Folgenden beschrieben werden. (Die Multiplexingverfahren können auch kombiniert werden – so wird z.B. bei WLAN die gesamte Bandbreite zunächst statisch in verschiedene Frequenzbereiche unterteilt, dann aber auf jedem dadurch entstehenden Frequenzkanal wieder dynamisches Zeitmultiplexing eingesetzt.)

Der größere Teil der existierenden Zugriffsprotokolle ist dezentral organisiert. Grundsätzlich lassen sich die Verfahren in zwei Gruppen einteilen:

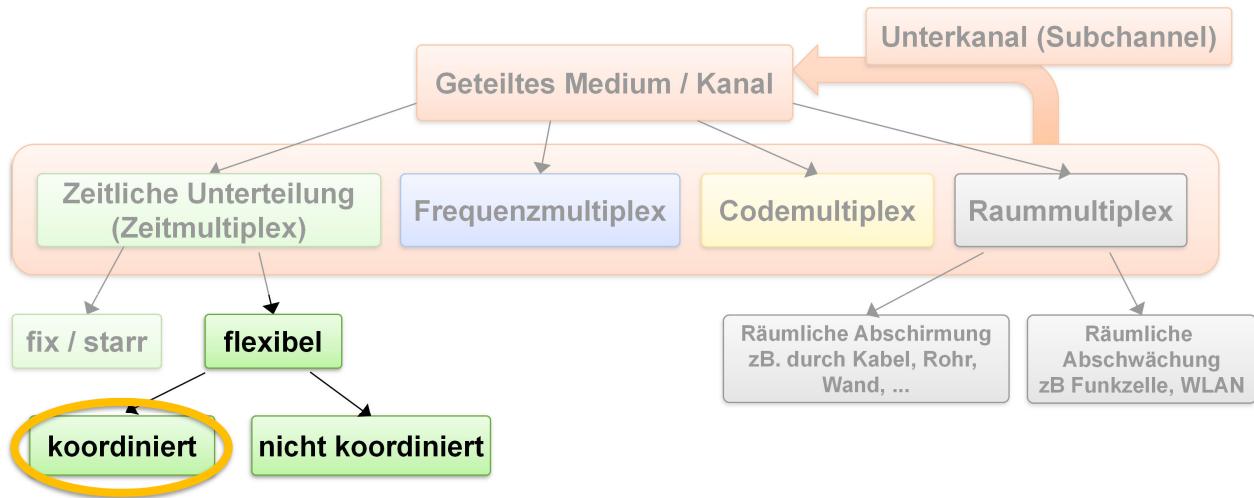
- *Konkurrierender Zugriff (Contention)*: Hier gibt es keine spezielle Zuteilung, d.h. wenn eine Station senden will, so prüft sie bei Bedarf gegebenenfalls, ob das Medium frei ist, und sendet dann. Es kann natürlich zu Kollisionen kommen, falls mehrere Stationen gleichzeitig anfangen zu senden – die Signale aller Stationen überlagern sich auf dem Medium und werden dadurch unbrauchbar. Kollisionen werden aufgelöst, indem zu einem späteren Zeitpunkt eine erneute Übertragung versucht wird. In diese Kategorie fallen z.B. CSMA/CD (Ethernet) und CSMA/CA (WLAN).
- *Geregelter Zugriff* (= kontrollierte Zuteilung): es gibt keine Kollisionen und keine dadurch bedingte Sendewiederholungen. Die Stationen teilen unter sich die Sendeberechtigung auf. Es existieren wiederum mehrere Möglichkeiten:
 - Bei der zyklischen Zuteilung wandert ein Bitmuster, welches das Senderecht repräsentiert (Token), von einer Station zur nächsten. Möchte eine Station senden, so muss sie warten, bis sie das Token erhält, dann kann sie eine gewisse Datenmenge senden (z.B. bei Token Ring).
 - Die Zuteilung kann auch *zentral* durch Aufforderung erfolgen
 - Andere Konzepte kombinieren Verfahren, z.B. das Token-Ring-Prinzip mit konkurrierendem Zugriff, indem zusätzliche Buffer verwendet werden.

Während die bisher in diesem Kapitel behandelten Verfahren bei jedem Netz benötigt werden, ist der Medienzugriff nur dann notwendig, wenn man Netze mit geteiltem Medium hat – und welches Verfahren sinnvoll ist, hängt sowohl vom Medium und der Topologie des Netzes ab, als auch von den Anforderungen der sendenden Stationen. Daher gibt es in diesem Bereich eine Vielzahl an Verfahren und Standards für lokale Netze unterscheiden sich vorwiegend in diesem Punkt.

Medienzugriff

- **Problem:**

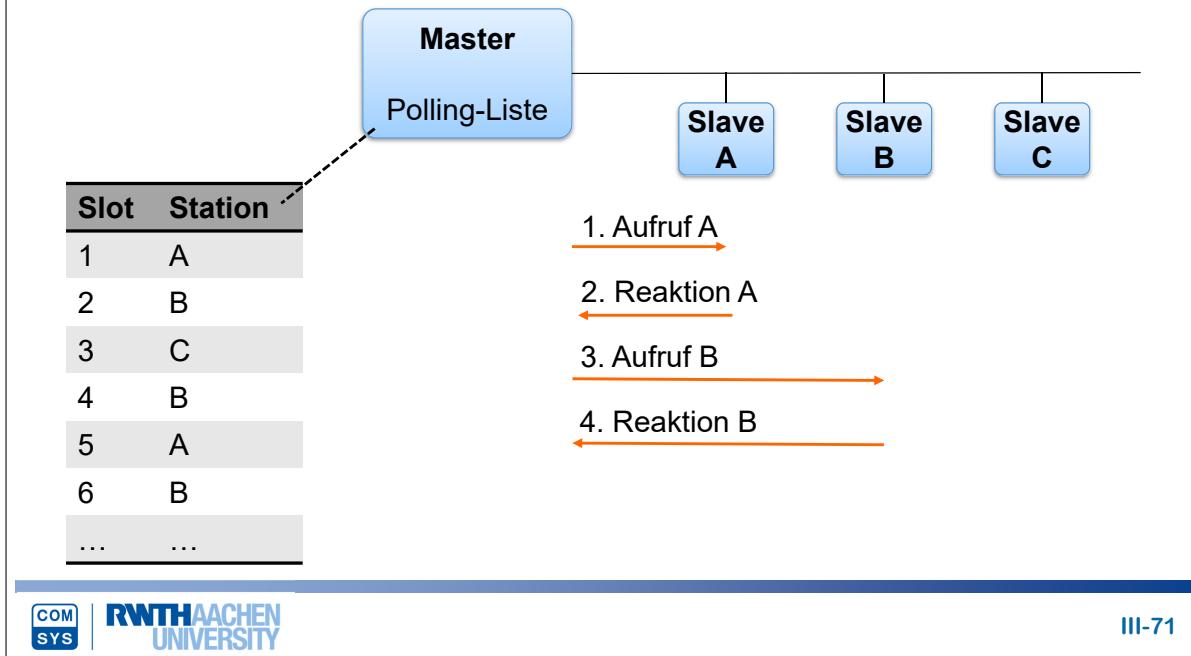
- ▶ Mehrere Stationen als Dienstnutzer eines einzigen physikalischen Mediums (*Shared Medium*) können *Kollisionen* verursachen



Medienzugriff: Zentrale Protokolle

- **Polling**

► Z.B. Bluetooth, USB



Polling (Poll/Select – Roll call polling)

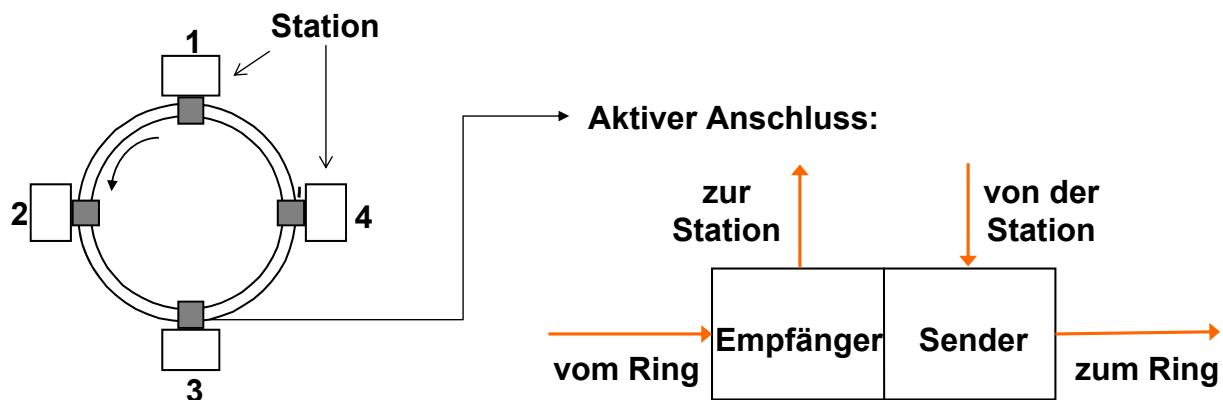
Hier unterscheidet man einen zentralen Rechner (Leitstation, Master) und viele angeschlossene Stationen (Folgestationen, Slave). Der zentrale Rechner entscheidet, welche Station wann das Senderecht bekommt. Dieses Prinzip wird z.B. bei Bluetooth eingesetzt: ein Master fragt zyklisch alle anderen Geräte ab, jedes Gerät hat danach seinen reservierten Zeitslot, um Daten zu senden.

Abfragen mit gemeinsamer Busleitung: als Variante zum obigen Mechanismus kann eine gemeinsame Leitung (gesonderter Kanal) verwendet werden, auf der Stationen dem Master einen Sendewunsch mitteilen können, damit dieser seine Aufrufliste anpassen kann. Eine Variante, hierbei sogar ohne Master auszukommen, ist das Daisy-Chaining bei kabelgebundenen Netzen; mehr Varianten existieren allerdings für drahtlose Netze.

Medienzugriff: Token Passing (Token Ring, IEEE 802.5)

- **Token Ring: Garantierter Medienzugriff innerhalb einer bestimmten Zeitperiode**

- ▶ Stationen sind Punkt-zu-Punkt zu Ring verbunden
- ▶ Stationen sind aktiv an das Medium gekoppelt (*Repeater*)



Bei Token Ring sind alle teilnehmenden Stationen physikalisch in einer Ringstruktur verbunden. Man kann auch alle Stationen auf andere Art verschalten und lediglich einen logischen Ring durch alle Stationen ziehen – dies wurde beispielsweise basierend auf einer Bus-Topologie bei 802.4 (Token Bus) vorgenommen.

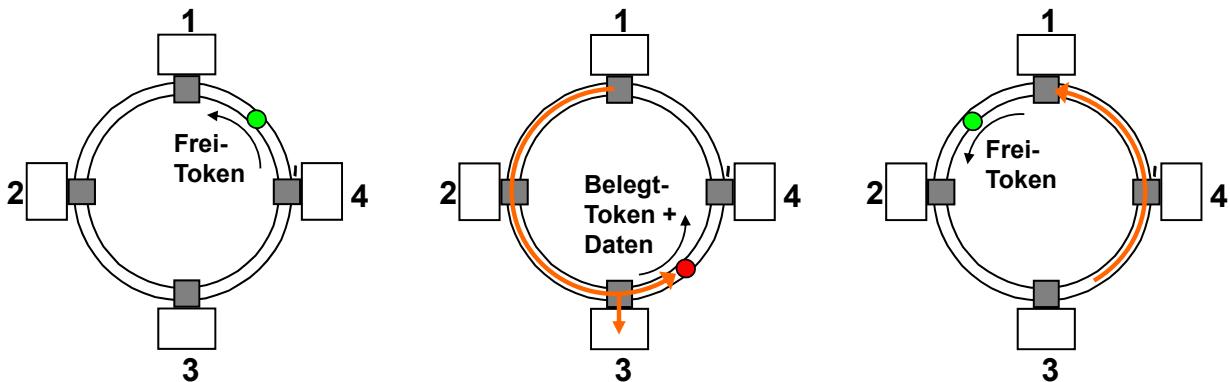
Auf dem Ring wird eine erlaubte Senderichtung festgelegt; damit hat jede Station eindeutig einen Vorgänger und einen Nachfolger. Die Stationen sind durch sogenannte Repeater aktiv an das Netz gekoppelt: die einkommende Signale werden rücktransformiert in ihre digitale Darstellung; falls sie nicht für die empfangene Station selber sind, sondern über die andere Leitung weitergeleitet werden müssen, werden sie neu in die Übertragungssignale transformiert. Damit nimmt jede Station eine Signalaufrischung vor (sie arbeitet als Repeater), so dass ausgedehnte Netze entstehen können.

Token Ring basiert zwar auf Koaxialkabel (mit Twisted Pair in 1/4/16 Mbit/s) dennoch ist es für verschiedene Physical Layer implementiert. Zur Datencodierung wird der Manchester-Code verwendet. Zudem wurde eine Variante standardisiert, die Glasfaser benutzt (FDDI) und damit noch größere Abstände zwischen den einzelnen Stationen erlaubt. Dadurch wurde FDDI eine Zeitlang zum beliebten Netztyp für MANs (auch an der RWTH).

Ablaufbeispiel Token-Ring

• Regelung des Zugriffs durch Token Passing

- Zeitliche Obergrenze für Medienzugriff durch Token Holding Time



- „Frei“-Token kreist
- 1 hat Sendewunsch

- 1 hat Token belegt
- 1 sendet an 3
- 3 kopiert

- 1 vernichtet Daten
- 3 setzt Quittungsbits (Piggybacking)
- 1 setzt Token auf „frei“

Der Medienzugriff bei Token Ring ist koordiniert und kontrolliert: nur wer eine bestimmte Bitfolge, das Token empfängt, darf senden.

Durch zyklische Weitergabe des Senderechts nach der eigenen Übertragung und die Festlegung einer maximalen Sendedauer pro Station (Token Holding Time, per Default 10ms) erlangt jede Station nach einer berechenbaren Zeit spätestens wieder das Senderecht – einer Station wird dadurch eine maximale Wartezeit bis zum sendebeginn sowie eine feste Datenrate zugesichert.

Erhält eine Station das Senderecht, sendet sie nicht das Token weiter über den Ring, sondern ihre eigenen Rahmen: durch Modifikation eines bestimmten Bits im Token wird aus dem Token der Anfang eines Datenrahmens. Es können beliebig viele Rahmen versendet werden, aber maximal für die Dauer der THT.

Die Rahmen umrunden den gesamten Ring – der Empfänger der Daten wird sie lediglich kopieren, aber trotzdem über den Ring weiterleiten. Dadurch kommen die Daten irgendwann wieder beim Sender an, der sie vom Ring nimmt und stattdessen wieder ein Token erzeugt und auf den Ring setzt.

Token Ring – Mechanismen

- **Zugriffskontrolle durch Token Passing**

- ▶ Modifikation eines Bits im Token → Rahmenbeginn
- ▶ Empfänger kann Quittungsbits an Rahmen anhängen
- ▶ Priorisierung einzelner Stationen möglich (Reservierung)

- **Token-Management nötig**

- ▶ Zentralisierte Station zur Überwachung (*Monitor*)
 - Erzeugung eines neuen Tokens nach Tokenverlust
 - Erkennung endlos kreisender Rahmen
- ▶ Reaktion auf verdoppelte Token
- ▶ Reaktion auf Ausfall des Monitors
- ▶ Überbrückung bei Ausfall einer Netzschaltung

Das Token ist ein 3 Byte langes Bitmuster, das zwischen den Stationen weitergeleitet wird. Will eine Station senden, kippt sie einfach ein bestimmtes Bit innerhalb des Tokens und macht das Token damit zum Beginn eines Rahmens. Ein „Belegt“-Token gibt es also eigentlich gar nicht, das „Frei“-Token wird einfach durch einen korrekten Rahmen ersetzt.

Die Quittierung des korrekten Empfangs eines Rahmens ist bei Token Ring direkt in den Medienzugriff integriert: da ein Rahmen den Ring vollständig umrundet und erst durch den Empfänger wieder vom Netz genommen wird, kann der Empfänger eine Quittung an den Rahmen anhängen. Der Sender erhält auch diese Quittung und weiß, ob die Daten korrekt empfangen wurden.

Durchläuft ein Rahmen eine Station, kann diese im Header bestimmte Reservierungsbits setzen, um sich selbst eine höhere Priorität zuzuweisen. Als Effekt wird bei der nächsten Erzeugung eines „Frei“-Tokens ein Token höherer Priorität erzeugt, welches andere Stationen weiterleiten müssen, ohne selbst Daten zu senden; erst eine Station höherer Priorität (im Normalfall die, die die Priorität gesetzt hat) darf das Token nehmen.

Beim Token Ring können mehrere Fehlersituationen auftreten, die speziell durch die Verwendung eines Tokens zustande kommen:

- Verlust des Tokens: Eine Monitorstation überwacht immer, ob bei ihr ein Token vorbeikommt. Wenn nach einer bestimmten Zeit kein Token mehr vorbeikommt, erzeugt sie ein neues Token.
- Endlos kreisender Rahmen: Fällt ein Sender aus, bevor er seinen Rahmen wieder vom Ring nehmen kann, wird dieser endlos auf dem Ring weiter übertragen und keine andere Station

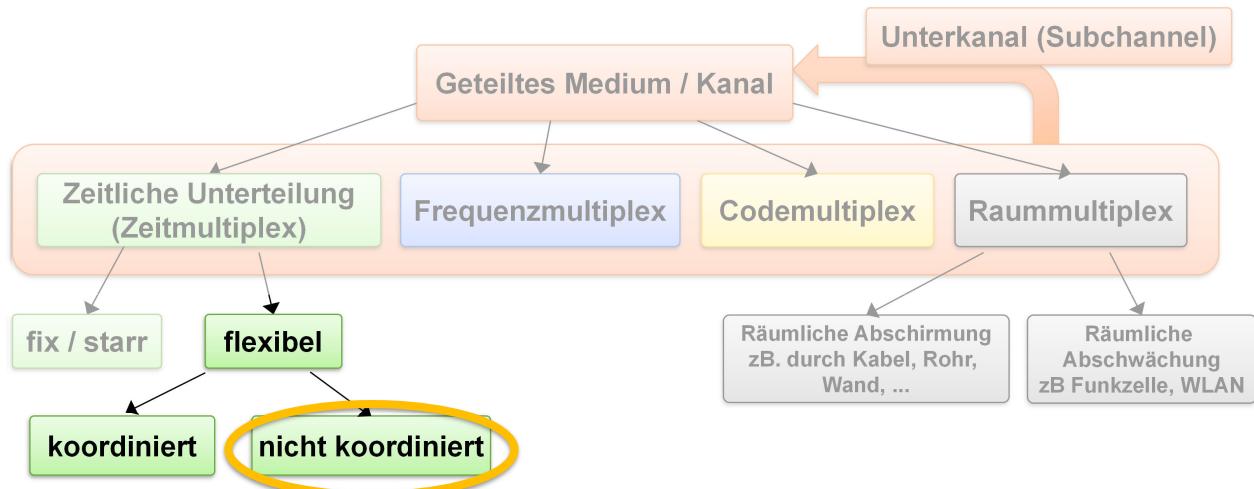
kann mehr senden. Um dies zu verhindern, setzt der Monitor bei jedem Rahmen, welcher vorbeikommt, das sogenannte M-Bit. Wenn nun ein Rahmen mit gesetztem M-Bit ankommt, macht dieser Rahmen bereits eine zweite Umrundung, was nur bei einer Fehlfunktion des Senders der Fall sein kann. Die Monitorstation leert den Ring und erzeugt ein neues Token.

- Doppelte Token: Wenn eine Station, die gerade sendet, einen Datenrahmen bekommt, in welchem nicht ihre Adresse als Sendeadresse steht, bricht sie die Sendung ab. Man hat jetzt kein Token mehr und verfährt nach Fall 1.
- Ausfall des Monitors: Wenn der Monitor ausfällt, so kann jede Station zum neuen Monitor werden. Den Ausfall des Monitors erkennt man daran, dass die obigen Fehler nicht behandelt werden. Wenn dies geschieht, so senden alle Stationen bestimmte Steuerrahmen. Jede Station lässt nur solche Steuerrahmen durch, deren Quelladresse kleiner als die eigene ist. So erhält nur eine Station ihren Rahmen wieder. Diese ist der neue Monitor.
- Ausfall einer Netzschmittstelle: Wenn eine Netzschmittstelle ausfällt, so schließt ein Relais. Somit ist die Leitung überbrückt und zwar eine Station nicht mehr erreichbar, alle anderen können aber noch kommunizieren.

Medienzugriff

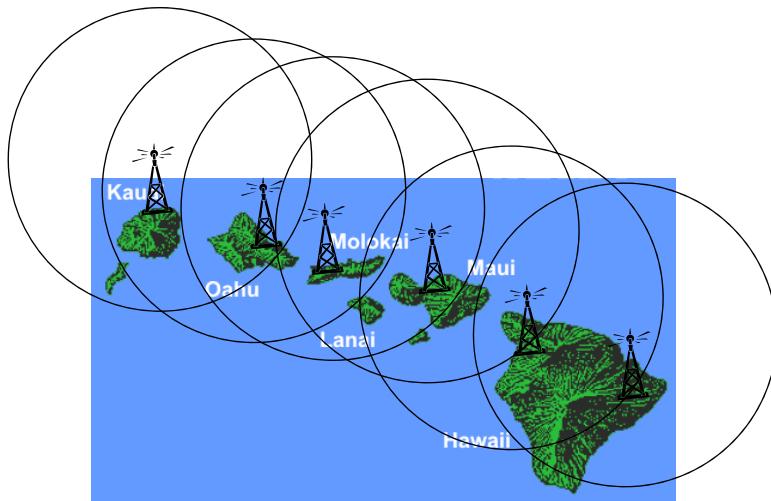
• Problem:

- ▶ Mehrere Stationen als Dienstnutzer eines einzigen physikalischen Mediums (*Shared Medium*) können *Kollisionen* verursachen



MAC-Protokolle mit konkurrierendem Zugriff: ALOHA

- Stationen übertragen Daten jederzeit nach Bedarf
 - ▶ Nicht alle Stationen in Reichweite
 - ▶ Koordination zwischen Stationen zu aufwändig

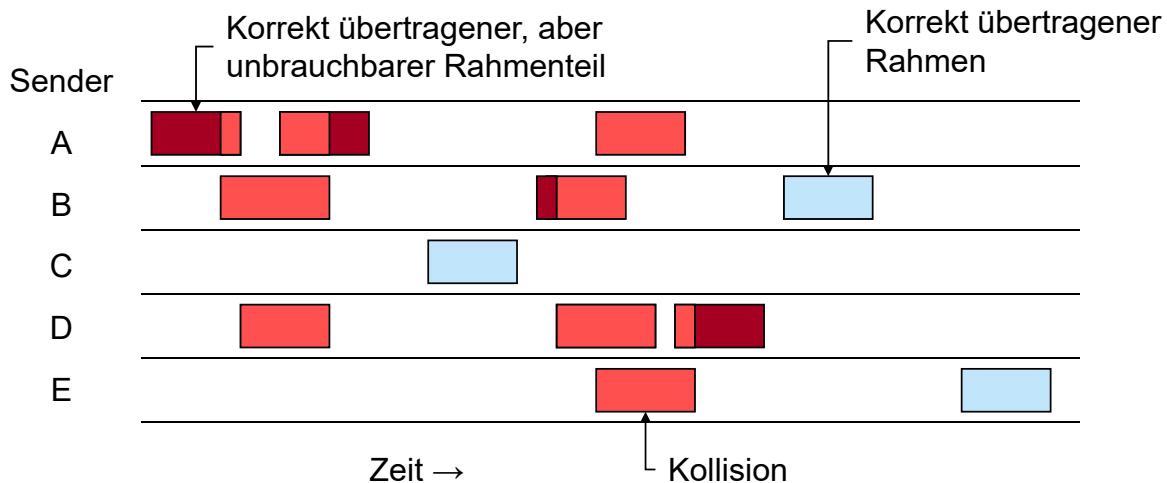


ALOHA

Dieses Protokoll wurde für die Kommunikation zwischen Stationen auf verschiedenen Inseln (Hawaii) mittels Funk entwickelt. Die Stationen können die anderen Stationen bis zu einer bestimmten Entfernung empfangen (z.B. nur die direkten Nachbarn), aber nicht weiter entfernte Stationen. Da eine Koordination zwischen den Stationen sehr aufwändig gewesen wäre, hat man komplett drauf verzichtet: wenn ein Sender etwas zu senden hat, so sendet er.

MAC-Protokolle mit konkurrierendem Zugriff: ALOHA

- Stationen übertragen Daten jederzeit nach Bedarf
 - ▶ *Kollisionen* führen zu gestörten Rahmen
 - ▶ Empfänger schickt Bestätigung, wenn Rahmen korrekt empfangen
 - ▶ Maximale Kanalauslastung 18% (analytischer Wert)



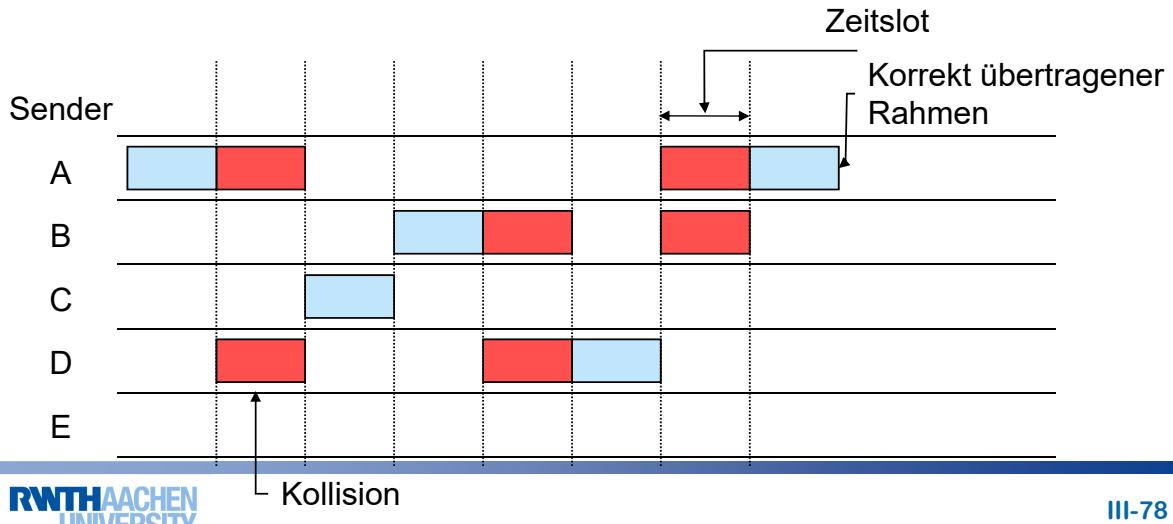
Das spontane Senden kann natürlich zu Kollisionen führen, welche der Sender nicht sofort bemerkt, da sich die verschiedenen Sender gegenseitig zum Teil nicht hören können.

Im schlimmsten Fall überlappen sich die gesendeten Rahmen in nur einem Bit und beide sind trotzdem unbrauchbar – damit ist die maximale Kollisionszeit also zweimal so lang wie die Dauer der Übertragung eines Rahmens maximaler Länge. Der (analytisch ermittelte) maximale Durchsatz liegt hier etwa bei 18% (nicht sehr effektiv). Damit man bei Kollisionen wenigstens nur die Übertragungszeit eines Rahmens verliert, hat man Slotted Aloha konzipiert.

Konkurrierender Zugriff: Slotted ALOHA

- Übertragung der Rahmen in festen *Zeitslots*

- ▶ Rahmenübertragung nur zu Beginn eines Zeitslots
 - ▶ Nur total überlappende Kollisionen treten auf
 - ▶ Kollisionszeit: Übertragungszeit von ein statt zwei Rahmen
 - ▶ Maximale Kanalauslastung auf 36% verbessert



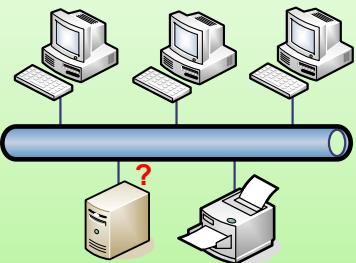
Slotted ALOHA

Durch die Vorgehensweise, Zeit in *Slots* fester Lnge einzuteilen und das Senden eines Rahmen nur zu Beginn eines Slots zu gestatten, wird die maximale Kanalauslastung auf 36% gesteigert. Dies ist immer noch sehr gering, erforderte aber zustzlich noch eine Synchronisationen zwischen allen Stationen, was die Komplexitt des Netzes erhhte.

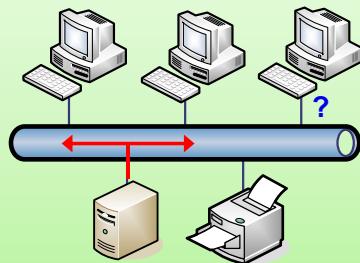
- Verbessertes ALOHA-Prinzip für LANs

- ▶ Ursprünglich entwickelt für Bustopologie (Broadcast-Netz)
- ▶ Prinzip: Listen before Talk (*Carrier Sense Multiple Access, CSMA*)
 - Standardisiert in IEEE 802.3, eingesetzt in Ethernet
 - Konkurrierendes Zugriffsverfahren
 - Sende nur, wenn Medium aktuell frei ist (Kollisionsvermeidung)

1. Ist das Medium frei?
(Carrier Sense)



2. Übertragen



Aus ALOHA ging CSMA/CD hervor. Während ALOHA für Funknetze entwickelt wurde, bei denen nicht alle Stationen sich gegenseitig hören können, ist CSMA/CD eine Modifikation für kleine Netze (mit geringen Latenzen), in denen sich alle angeschlossenen Stationen gegenseitig hören können.

Beim CS (Carrier Sense)-Verfahren hören die Stationen im Gegensatz zu ALOHA das Medium ab, bevor sie zu senden beginnen. Dadurch lässt sich vermeiden, dass eine sendende Station gestört wird.

Ist das Medium frei, wartet die Station noch 9,6 µs (Interframe Spacing) bevor sie zu senden beginnt. Dieses Spacing soll vermeiden, dass eine Station, die erfolgreich Medienzugriff erlangt hat, beliebig viele Rahmen in Folge senden kann. Durch die erzwungene Wartezeit zwischen zwei Rahmen können andere Stationen mit Sendewunsch das Medium als frei erkennen und auch zum Senden kommen.

Es kann passieren, dass eine Station bereits sendet während eine zweite Station aufs Medium lauscht um festzustellen, ob sie senden kann. Durch die endliche Ausbreitungsgeschwindigkeit auf dem Medium kann es vorkommen, dass die zweite Station das Medium als frei erkennt, obwohl die erste Station bereits sendet (Abbildung rechts auf der Folie). In diesem Fall fängt auch die zweite Station guten Gewissens an zu senden.

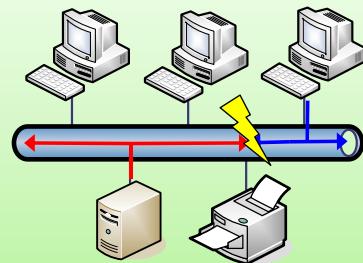
- Verbessertes ALOHA-Prinzip für LANs

- ▶ Listen while Talk (with *Collision Detection, CD*)

- Empfang von Signalen während der eigenen Aussendung von Daten bedeutet, dass eine Kollision aufgetreten ist:
 - Erkennbar an Spannungsspitzen
 - Jamming-Signal; Abbruch der Sendung
 - Erneuter Versuch nach statistisch verteilter Verzögerungszeit

3. Prüfe auf Kollision (Collision Detection)

Falls ja: sende Jamming-Signal und breche ab. Danach weiter mit z.B.
“Binary Exponential Backoff “-Algorithmus



In diesem Fall treffen sich die Signale irgendwo auf dem Medium und überlagern sich (= Kollision). Die beiden Übertragungen löschen sich dabei nicht gegenseitig aus, wie es diese Folie scheinbar zeigt. Die Signale beider Stationen breiten sich über das gesamte Medium aus und jede angeschlossene Station empfängt die Überlagerung. Die beiden sendenden Stationen oder auch beliebige andere an das Medium angeschlossene Stationen bemerken die Kollision an Spannungsspitzen (die durch die Überlagerung entstehen).

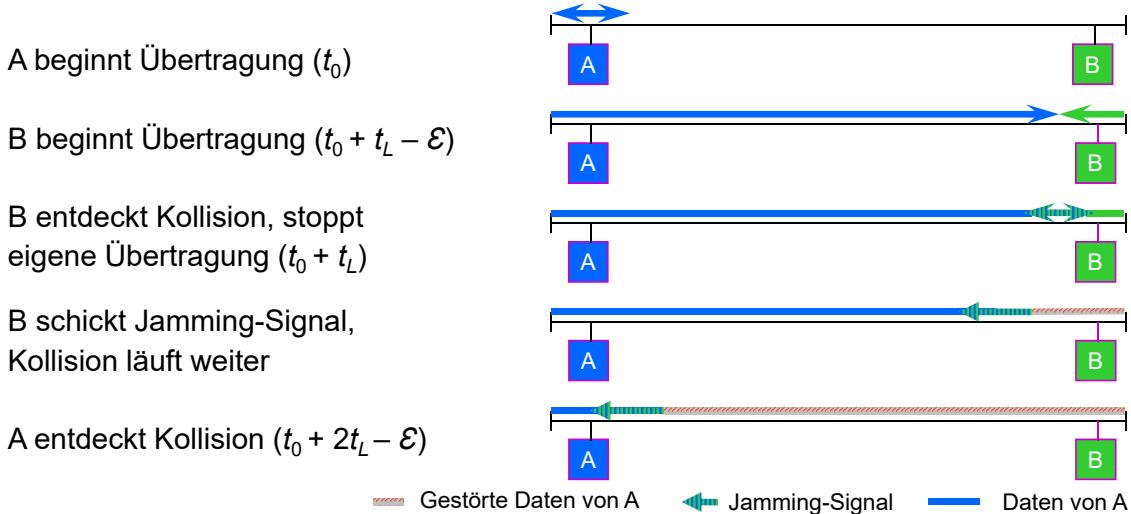
So eine Kollision sollte schnellstmöglich erkannt werden, damit das Medium nicht für längere Zeit unnütz belegt wird. Dies wird durch das CD ermöglicht: sendende Stationen hören während des gesamten Sendevorgangs mit, ob sie andere Signale außer ihren eigenen hören.

Wenn eine (beliebige) Station bemerkt (an Spannungsspitzen und Codeverletzungen), dass eine Kollision auftritt, so sendet sie ein sogenanntes *Jamming-Signal*. Dieses Signal ist eine definierte Bitfolge, welche mit höherer Spannung ausgesendet wird, um alle anderen Signale auf dem Medium zu übertönen und auf jeden Fall erkannt zu werden. Dieses soll allen Stationen ermöglichen, die Kollision zu erkennen. Nach Erhalt des Jamming-Signals stoppen alle Sender ihre Sendung.

CSMA/CD – Beispielablauf

- Zu berücksichtigen: Laufzeiten**

- t_L : Signallaufzeit von A nach B (Propagation Delay)
- $2t_L$: Signallaufzeit von A nach B und zurück (Round Trip Delay)



Wichtig bei CSMA/CD ist das Propagation Delay: die Laufzeit eines Signals von einem bis zum anderen Ende eines Übertragungsmediums. Diese beträgt in obigem Beispiel t_L . Sie entspricht der Latenz, wobei die Latenz üblicherweise als Ausbreitungsverzögerung zwischen zwei Stationen angesehen wird, das Propagation Delay hier die Verzögerung von einem äußersten Ende des Netzes zum anderen bezeichnet. Die Latenz zwischen je zwei Stationen im Netz ist abhängig von der Position der Stationen, das Propagation Delay ist das Maximum der Latzenzen im Netz.

Im Beispiel beginnt B zu senden (zum Zeitpunkt $(t_0 + t_L - \epsilon)$), kurz bevor zum Zeitpunkt $(t_0 + t_L)$ das Signal von A die Station B erreicht. B erkennt die Kollision praktisch unverzögert nach deren Auftreten. B stoppt die begonnene Sendung und bringt ein starkes Störsignal auf den Bus (Jamming-Signal). Die von der Kollision und dem Jamming-Signal herrührende Signalstörung läuft nach A zurück und wird zum Zeitpunkt $(t_0 + 2t_L - \epsilon)$ von A entdeckt. A stoppt seine Sendung ebenfalls; der bisher gesendete MAC-Teilrahmen ist zerstört.

Damit A erkennt, dass das detektierte Kollisionssignal den von ihm abgesandten MAC-Rahmen betrifft, darf die Sendung nach $2t_L$ (Round Trip Delay) noch nicht beendet sein. 802.3-Rahmen müssen also eine Mindestlänge haben, damit eine Kollision zuverlässig erkannt wird!

Nachdem beide kollidierenden Stationen die Übertragung gestoppt haben, erfolgt ein erneuter Anlauf nach einer gewissen Verzögerungszeit.

Kollisionserkennung bei Ethernet (10 Base2)

- ***Slot Time* (= Round Trip Delay)**
 - ▶ Minimale Dauer für das Senden eines Rahmens zur zuverlässigen Kollisionserkennung
- „Willkürliche“ Festlegung der maximalen Netzausdehnung
 - ▶ Maximales Round Trip Delay berechenbar
 - 2800 Meter beim klassischen Ethernet
 - 51,2 µs Slot Time
 - ▶ Bei Datenrate von 10 MBit/s: in 51,2 µs werden 64 Byte versendet
 - ▶ Es müssen immer mindestens 64 Byte versendet werden, um Kollisionen zuverlässig erkennen zu können
 - Sonst: Padding auf Mindestlänge



Um die Dauer berechenbar zu machen, die ein Sender aktiv bleiben muss, bis er sicher sein kann, dass keine Kollision mehr auftreten wird, wurde die maximale Ausdehnung beim anfänglichen Ethernet auf 2800 Meter festgelegt. Diese Entfernung war mit dem verwendeten Koaxialkabel aufgrund der Dämpfung nicht zu überbrücken, daher musste alle 500 Meter ein Repeater eingefügt werden, der das Signal wieder auffrischte. Die Ausbreitungszeit eines Signals von einem Ende des größtmöglichen Netzes bis zum anderen inklusive der Verzögerungen durch Verarbeitung in den Repeatern betrug 51,2µs (bzw. etwas weniger – man hat so aufgerundet, dass man innerhalb dieser Zeit eine Zweierpotenz an Bits versenden kann). Bei der Datenrate von 10 MBit/s, die erreicht werden sollte, können in dieser Zeit 64 Byte versendet werden. Daher muss eine Station bei Ethernet immer Rahmen mit mindestens 64 Byte aussenden, auch wenn sie eigentlich gar nicht so viel zu senden hat. Muss eigentlich nur weniger gesendet werden, wird Padding vorgenommen; der Payload des Rahmens wird mit Füllbits auf die nötige Anzahl Bytes gebracht.

Binary Exponential Backoff

- **Vorgehen nach Kollision**

- ▶ Beide (bzw. alle) Stationen brechen Übertragung ab
- ▶ Vermeidung von Folgekollisionen?
 - Stationen ziehen zufällige Wartezeit bis Übertragungswiederholung

- **BEB-Algorithmus:**

- ▶ Anpassung der Wartezeit an die Zahl wartender Stationen
 - Nach der i -ten Kollision: ziehe Wartezeit x aus $[0, 2^i - 1]$
 - Wartezeit bis zum nächsten Carrier Sense: $x \cdot 51,2 \mu\text{s}$
 - Nach 10 – 15 Kollisionen konstantes Intervall $[0, 2^{10} - 1]$
 - Nach 16 Kollisionen gebe auf
- ▶ Jede Station verwaltet ihr i selbst
 - Einzelne Stationen können benachteiligt werden!

Wenn eine sendende Station eine Kollision erkannt hat und aufhört zu senden, wartet sie eine gewisse Zeit lang und hört dann das Medium wieder ab, ob sie ihre Übertragung wiederholen kann. Es kann natürlich weiterhin zu Kollisionen kommen, vor allem weil während des Wartens noch einige sendewillige Stationen dazukommen können. Die Wartezeit einer Station nach einem erfolglosen Versuch errechnet sich nach dem exponentiellen Backoff-Algorithmus. Dieser berechnet die Anzahl der Slot-Times (die Slot-Time von $51,2 \mu\text{s}$ entspricht 64 Byte), die gewartet werden muss. Dies garantiert, dass, falls eine Station anfängt, das Medium zu nutzen, nach Ablauf einer Slot-Time auf jeden Fall alle anderen Stationen wissen, dass das Medium in Benutzung ist und weiterhin warten.

Man wartet somit immer ein Vielfaches der Slot-Time. Wenn zwei Stationen die gleiche Zufallszahl ziehen, dann gibt es wieder eine Kollision. Der exponentielle Anstieg des Bereichs, aus dem Zufallszahlen gezogen werden, garantiert, dass bei wenigen Stationen diese nicht allzu lange warten müssen (wir ziehen nur kurze Wartezeiten, die sich hoffentlich trotzdem unterscheiden). Warten viele Stationen, so wird das Intervall der Wartezeiten nach jeder Kollision vergrößert, so dass nach einigen Schritten jede voraussichtlich eine andere Zufallszahl ziehen: nach der ersten Kollision zieht die Station eine Zufallszahl aus dem Bereich $[0, 1]$ und wartet diese Anzahl von Slots. Nach der i -ten Kollision zieht sie eine Zufallszahl aus dem Intervall $[0, 2^i - 1]$ und wartet diese Anzahl von Slots. Ab dem 10ten Wiederholungsversuch bleibt das Intervall allerdings immer bei $[0, 2^{10} - 1]$. Nach 16 erfolglosen Versuchen bricht die Station ab und verwirft das Paket.

Vergleich von CSMA/CD und Token Ring

CSMA/CD

- Vorteile

- ▶ Einfaches Protokoll
- ▶ Installation im laufenden Betrieb einfach möglich
- ▶ Passive Kabel
- ▶ Nur geringe Verzögerung bei niedriger Last

Token Ring

- Vorteile

- ▶ Sehr guter Durchsatz und hohe Effizienz unter hoher Last
- ▶ Automatische Erkennung und Elimination von Kabelbruch
- ▶ Prioritäten möglich
- ▶ Kurze Rahmen möglich, Rahmenlänge nur durch THT begrenzt
- ▶ Echtzeitbetrieb möglich

Vergleich von CSMA/CD und Token Ring

CSMA/CD

- **Nachteile**

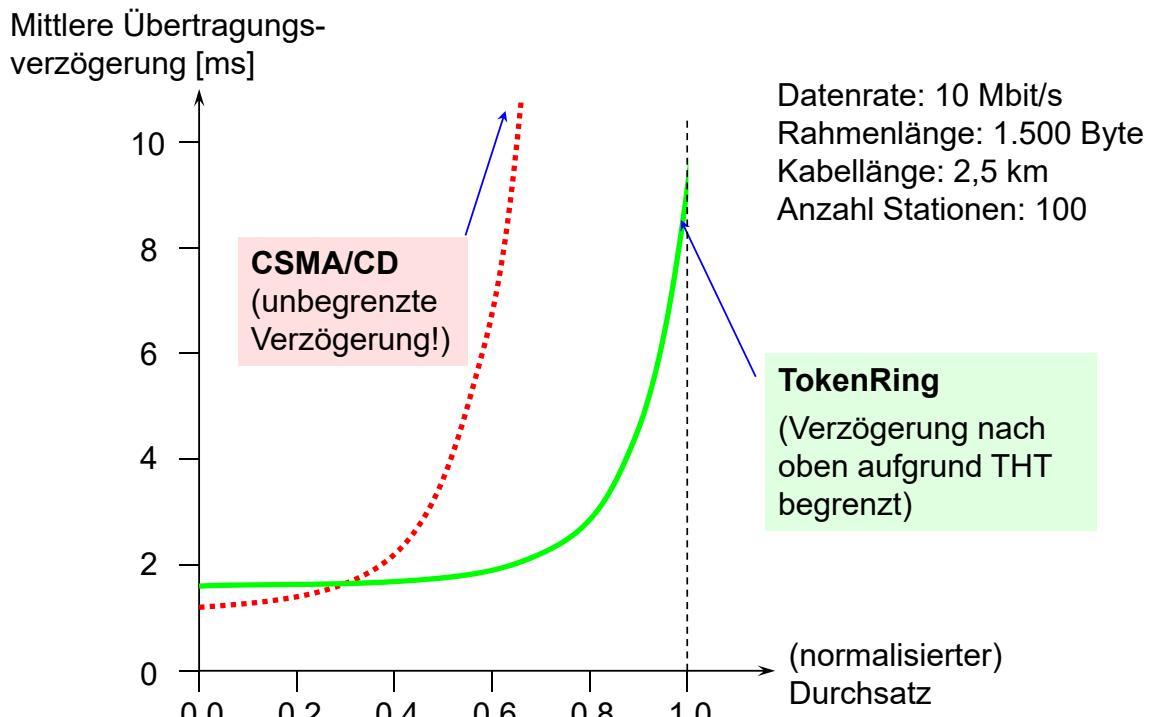
- ▶ Kollisionserkennung benötigt minimale Rahmenlänge
- ▶ Keine Prioritäten
- ▶ Nicht deterministisch, deshalb kein Echtzeitbetrieb möglich
- ▶ Begrenzte Netzausdehnung
- ▶ Geringe Effizienz durch viele Kollisionen, problematisch bei höherer Last

Token Ring

- **Nachteile**

- ▶ Komplexes Fehlermanagement (Token-Verlust, Monitorausfall)
- ▶ Unnötige Verzögerung unter niedriger Last

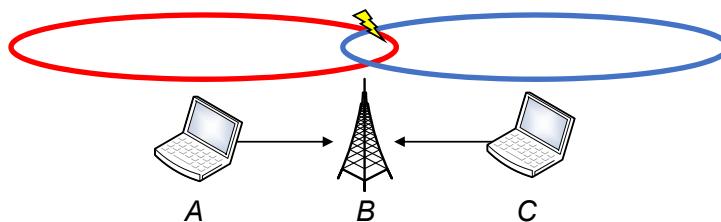
Verzögerungen von CSMA/CD vs. Token Ring



- Kann CSMA/CD auch in drahtlosen Netzen eingesetzt werden?

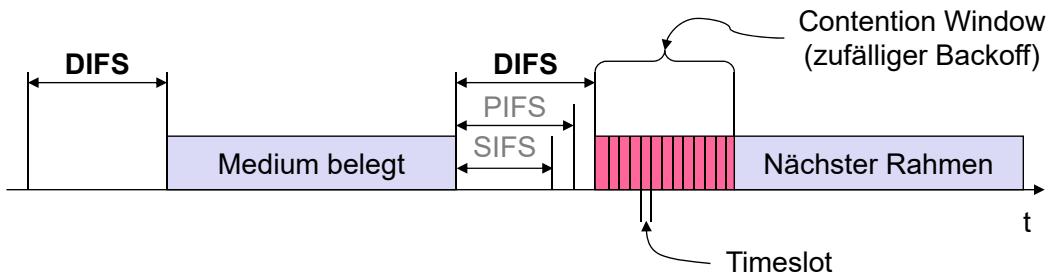
► *Hidden-Station-Problem*

- A sendet an B, C ist nicht in Reichweite von A
- C möchte an B senden und führt CS aus – Medium frei
 - Carrier Sense schlägt fehl
- Kollision bei B, A stellt Kollision nicht fest
 - Collision Detection schlägt fehl
- A ist gegenüber C versteckt (*hidden* für C), ebenso C gegenüber A



In drahtlosen Netzen tritt das ALOHA-Problem auf: es sind nicht notwendigerweise alle Stationen in Reichweite aller anderen Stationen. Ein typisches Beispiel ist Wi-Fi: mobile Geräte können überall rund um den Access Point verteilt sein und sich gegenseitig nicht sehen.

- **Modifikation: Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)**
 - ▶ Verzichte auf Kollisionserkennung
 - ▶ Kollisionsvermeidung durch zufällige Wartezeit vor Sendungsbeginn
 - Ähnlich BEB

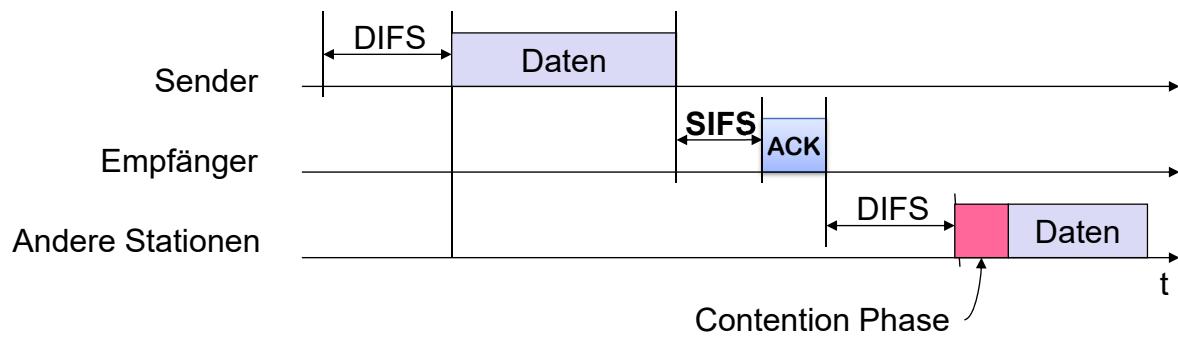


CSMA/CA versucht Kollisionen zu vermeiden, indem Sendewünsche zufällig verteilt werden. Eine sendewillige Station führt nach wie vor CSMA durch und wartet danach einen Inter-Frame Space wie bei Ethernet. Dieser Space ist hier mit DIFS bezeichnet: Distributed Coordination Function Inter Frame Space. Ist das Medium immer noch frei, würde eine Station bei Ethernet nun anfangen zu senden. Bei CSMA/CA hingegen hängt es davon ab, ob das Medium zuvor belegt war oder nicht. War es vorher belegt, gibt es ein nicht zu vernachlässigendes Risiko, dass auch andere Stationen warten und gleichzeitig anfangen würden zu senden. Daher wird nun zufällig eine zusätzliche Wartezeit aus einem vorgegebenen Intervall gezogen (dynamisches Intervall, ähnlich zu BEB). Ist das Medium nach Ablauf der Zufallszeit immer noch frei, beginnt eine Station zu senden. Garantierte Kollisionsvermeidung erreicht man dadurch nicht.

Die anderen IFSs „Point Coordination Function IFS“ und „Short IFS“ dienen dazu, bestimmte Arten von Übertragungen priorisieren zu können, indem sie eher und ohne Zufallswartezeit Medienzugriff bekommen.

- Hohe Bitfehlerrate – hohes Risiko Rahmenverlust

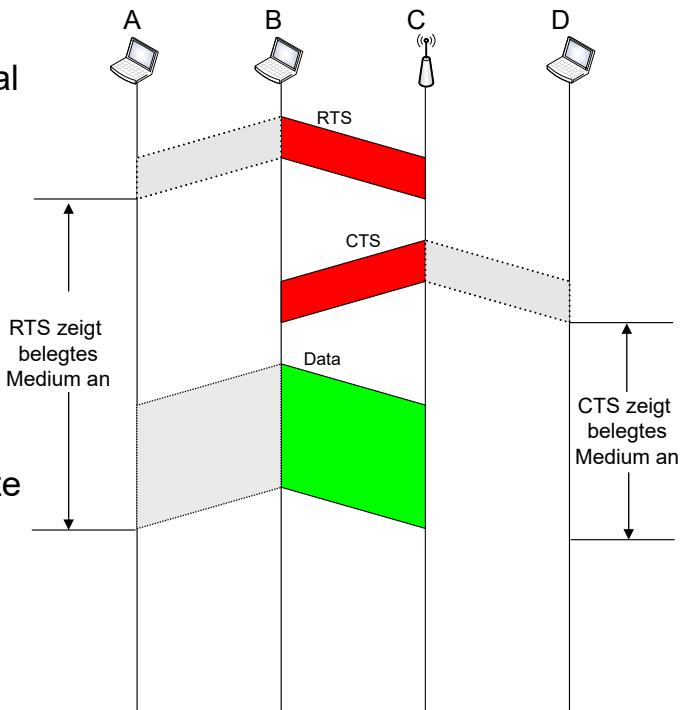
- ▶ Quittungsmechanismus integriert
- ▶ Quittung wird priorisiert (SIFS)



Alternative MACA – Multiple Access with Collision Avoidance

- **MACA:**

- ▶ Vor Übertragung wird Kanal reserviert
- ▶ Request to send (RTS) wird verwendet, um Empfänger nach Empfangsbereitschaft zu fragen
- ▶ Clear to send (CTS) teilt Senderecht zu
- ▶ Alle Stationen in Reichweite des Empfängers erhalten das CTS und wissen, dass sie "hidden" sind und nicht senden sollten



Sollen Kollisionen vermieden werden, kann man MACA verwenden. Eine Station führt immer noch CSMA durch, aber falls der Kanal nicht belegt ist, beginnt die Station nicht mit Versendung des Rahmens, sondern sendet zunächst einen Kontrollrahmen: ein RTS, mit dem sie den Empfänger nach Empfangsbereitschaft fragt.

Der Empfänger antwortet mit einer anderen Kontrollnachricht (CTS), falls er nicht bereits andere anstehende Anfragen hat. Der Sender darf nach Erhalt eines CTS anfangen zu senden. Erhält er kein CTS, muss er seine Übertragung zurückstellen.

Der Sinn des CTS ist auch, dass alle Stationen um den Empfänger – potentielle Hidden Stations – von der anstehenden Übertragung erfahren und selbst keinen Übertragungsversuch starten. Gleiches gilt für alle Stationen rund um den Sender (die das RTS erhalten) – diese sollten auch nicht senden, da der Sender typischerweise auch eine Quittung erhält, die nicht gestört werden soll.

Kapitel 3: Sicherungsschicht

- **Rahmenbildung**

- ▶ Blockbildung, Codetransparenz durch Character/Bit Stuffing

- **Fehlererkennung/-behandlung und Flusskontrolle**

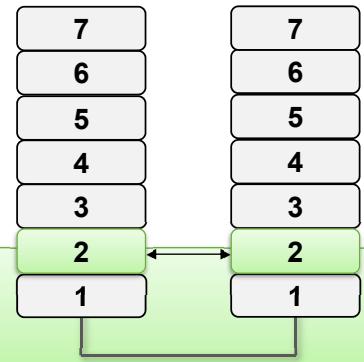
- ▶ Fehlererkennende Codes (CRC)
- ▶ Proaktive und reaktive Fehlerbehebung (FEC und ARQ)
- ▶ Flusskontrolle durch Sliding Window
- ▶ Beispielprotokoll: HDLC

- **Medienzugriff**

- ▶ Topologien
- ▶ Token Passing, CSMA/CD, CSMA/CA

- **Standards für lokale Netze**

- ▶ Ethernet



- **Implementierung von CSMA/CD in der Praxis**

- ▶ 70er Jahre: experimentelles Netzwerk, Koaxialkabeln, Datenrate von 3 MBit/s; entwickelt von der Xerox Corporation als ein Protokoll für LANs mit sporadischem aber burst-artigem Verkehrsverhalten
- ▶ 1980: gemeinsame Weiterentwicklung durch DEC, Intel und Xerox zu einer 10 MBit/s-Variante („klassisches“ Ethernet)
- ▶ Bus-Topologie mit einer maximalen Segmentlänge von 500 Metern, Anschlussmöglichkeit für maximal 100 passive Stationen; Repeater zum Zusammenschluss mehrerer Segmente
- ▶ Gängiges Medium: Kupferkabel; aber auch Glasfaser-Kabel kommen zum Einsatz (erhöht Segmentlänge)
- ▶ Frühe 90er Jahre: Bus-Topologie wird mehr und mehr von Stern-Topologie mit Hub oder Switch verdrängt (auf Twisted-Pair oder Glasfaserkabel basierend)

In den Anfangszeiten der lokalen Netze haben sich zwei Konzepte prominent entwickelt: 802.5 wurde unter dem Namen „Token Ring“ vertrieben, 802.3 mit leichten Variationen unter dem Namen „Ethernet“. Bitte beachten: Die 802.3/5 definieren den MAC-Layer – für konkrete Produkte werden auch Festlegungen für den Physical-Layer benötigt!

Rahmenformat nach IEEE 802.3 / Ethernet

PR 7 Byte	SD 1 Byte	DA 2/6 Byte	SA 2/6 Byte	Länge/ Typ 2 Byte	Payload ≤1500 Byte	PAD 0-46 Byte	FCS 4 Byte
--------------	--------------	----------------	----------------	-------------------------	-----------------------	------------------	---------------

- PR : Präambel zur Synchronisation (7 x 10101010)
 SD : *Start-of-frame Delimiter* zeigt Blockbeginn an (10101011)
 DA : *Destination Address*
 SA : *Source Address*
 Länge : Anzahl der Bytes im Datenfeld (IEEE 802.3), Typ des Payloads im Datenfeld (Ethernet)
 Payload: Datenfeld, das maximal 1500 Byte umfassen darf
 PAD : *Padding*, um zu kurze Datenfelder auf die nötige Länge zu ergänzen
 FCS : *Frame Check Sequence*, CRC

802.3 – Rahmenformat

- Präambel: Dient zur anfänglichen Synchronisation der Stationen durch konstant auftretenden Pegelwechsel.
- Start Delimiter: zeigt den Beginn eines Rahmens an: 10101011.
- Zieladresse, Absenderadresse: laut ursprünglichem Standard: 2 oder 6 Byte; heute: 6 Byte MAC-Adresse.
- Längenfeld: gibt die Länge des Datenfelds (in Byte) an. Bei der Implementierung von 802.3 in Ethernet wird allerdings dieses Feld für einen anderen Zweck missbraucht: der Empfänger muss wissen, welcher Typ von Daten im Datenteil enthalten ist. Laut IEEE steht diese Angabe im LLC-Header, mit dem der Datenteil beginnt. In der Praxis verzichtet man aber auf den Einsatz von LLC, daher muss die Typangabe im 802.3-Rahmen selbst untergebracht werden. Bei Ethernet wird daher die Typangabe in das Längenfeld eingetragen; es werden nur Werte größer 1500 erlaubt, um eine Verwechslung mit einer Längenangabe zu vermeiden.
- Payload: Die Daten werden ohne Bitstuffing o.ä. angefügt, da kein End Delimiter verwendet werden muss (Längenangabe). Bei Verwendung des Längenfelds für die Angabe des Typs von Daten erhält man die Datenfeldlänge implizit dadurch, dass es ein „Interframe Spacing“ gibt; eine Station, die feststellt, dass das Medium frei ist, muss noch für 9,6µs warten, bevor sie zu Senden anfangen darf. Zwei Rahmen können also nicht direkt aufeinander folgen. Die letzten 4 empfangenen Byte müssen also die FCS gewesen sein.
- PAD: Dieses Feld enthält eventuell Padding, d.h. wahllos eingefügt Bits. Zu kurze Datenfelder werden dadurch aufgestockt, damit die Rahmenlänge (ohne Präambel) mindestens 64 Byte beträgt, die vorgeschriebene Mindestlänge zur Kollisionserkennung.
- FCS: Dieses Feld dient zur Fehlererkennung: CRC-Verfahren mit 32 Bit-Polynom

Als Leitungscodierung verwendet Ethernet den Manchester-Code. Bei Überlagerung von Signalen kommt es zu Codeverletzungen, so dass alle Stationen Kollisionen erkennen können.

Ethernet

- **Ethernet: der heutige LAN-Standard**

- ▶ Einfach zu verstehen, umzusetzen und zu überwachen
- ▶ In der Anschaffung billig
- ▶ Bringt Flexibilität bzgl. der Topologie mit sich
- ▶ Über die Zeit Entwicklung von 4 Klassen von Ethernet-Varianten:

- (Klassisches) Ethernet → 10 Mb/s
- Fast Ethernet → 100 Mb/s
- **Gigabit Ethernet** → 1000 Mb/s
- 10Gigabit-Ethernet → 10000 Mb/s

- ▶ Des Weiteren:
 - 40G-Ethernet, 100G-Ethernet

„Ethernet“ bezeichnet nicht einen einzelnen Standard, sondern wird im Sprachgebrauch für eine ganze Standardfamilie verwendet. Die Grundprinzipien sind bei allen Unterstandards gleich, sind aber auf unterschiedliche Medien angepasst und können durch weitere Anpassungen unterschiedliche Datenraten erzielen. Diese Unterschiede betreffen also im Wesentlichen die Umsetzung des Physical-Layers, während der MAC-Layer größtenteils gleich umgesetzt wird (~802.3).

Datenrate und Kollisionserkennung

- Festlegung der minimalen Rahmenlänge im klassischen Ethernet auf 64 Byte
 - ▶ Kompatibilität: behalte minimale Rahmenlänge bei Fast Ethernet bei
- Fast Ethernet (IEEE 802.3u)
 - ▶ Datenrate von 100 MBit/s: 64 Byte werden in 5,12 µs versendet!
 - ▶ Reduktion der Ausdehnung auf gut 200 Meter notwendig, um Kollisionen zuverlässig erkennen zu können
- Gigabit Ethernet (IEEE 802.3z)
 - ▶ Datenrate von 1 GBit/s: 64 Byte werden in 0,512 µs versendet!
 - ▶ Reduktion der Ausdehnung auf 20 Meter?

Umsetzung bei Ethernet

	Ethernet	Fast Ethernet	Gigabit Ethernet
Maximale Ausdehnung	bis zu 2800 Meter	205 Meter	200 Meter
Datenrate	10 MBit/s	100 MBit/s	1000 MBit/s
Minimale Rahmenlänge	64 Byte	64 Byte	520 Byte
Maximale Rahmenlänge	1526 Byte	1526 Byte	1526 Byte
Signal-darstellung	Manchester-Code	4B/5B-Code, 8B/6T-Code, ...	8B/10B-Code, ...
Topologie	Bus oder Stern	Stern	Stern
Maximale #Repeaters	5	2	1

Für einen konkreten Netzstandard werden die unteren beiden Schichten also gemeinsam standardisiert. Dies betrifft auf dem Physical-Layer die Art des physikalischen Mediums, seine Länge, Steckertypen zum Anschluss von Stationen, Topologie des Netzes, Codierung, Schrittgeschwindigkeit, ... - hier dargestellt ist nur eine Teilmenge der Möglichkeiten. Z.B. gibt es beim Classical Ethernet nicht nur die Möglichkeit, den Manchester-Code zu verwenden, sondern auch Modulationsverfahren. Hier dargestellt sind die gängigsten Varianten.

Dargestellt ist hier auch ein Aspekt des MAC-Layers: die erlaubte Rahmenlänge. Durch die Wahl von CSMA/CD als MAC-Verfahren ergibt sich die Notwendigkeit einer minimalen Rahmenlänge.

Die maximale Ausdehnung ist auch abhängig von CSMA/CD. Eine maximale Ausdehnung von 2800 Metern bei Ethernet auf Basis von Kupferkabel allerdings wäre zunächst nicht möglich, wenn auch durch CSMA/CD erlaubt: durch Signalabschwächung auf dem Medium können Kupferkabel nur über wenige 100 Meter die notwendige Bandbreite zur Verfügung stellen. Sollen größere Ausdehnungen erzielt werden, ist der Einsatz von Repeatern zur Signalauffrischung notwendig. Prinzipiell kann durch Repeater auch eine beliebig große Ausdehnung erzielt werden – aber durch CSMA/CD sind keine größeren Ausdehnungen erlaubt.

Die einzige „Optimierung“ beim klassischen Ethernet waren sogenannte *Brücken* (*Bridges*), die Kollisionsdomänen auf unterschiedlichen Segmenten trennen sollten.

Man setzt zur Verbindung zweier Segmente nicht mehr einen Repeater zur puren Signalauffrischung ein, sondern implementiert in den Brücken auch Schicht-2-Protokolle. Hierdurch ist eine Brücke in der Lage, MAC-Adressen zu verstehen. Die Brücke kann lernen, welche Stationen mit welchen Adressen über welches der Segmente, die sie koppelt, erreichbar sind. Damit kann sie auch entscheiden, ob ein auf einem Segment empfangener Rahmen auf das andere Segment weitergeleitet werden muss oder nicht. Die Brücke führt also eine Weiterleitungstabelle zu allen ihr bekannten MAC-Adressen. Damit wird die Zahl von Kollisionen auch in ausgedehnten Netzen stark reduziert. Trotzdem darf die maximal erlaubte Ausdehnung nicht überschritten werden, da miteinander kommunizierende Stationen natürlich auch in den am weitesten auseinanderliegenden Segmenten platziert sein können und die Rahmen auch durch die Brücken über alle Segmente hinweg weitergeleitet werden.

Gigabit-Ethernet: Rahmenformat

- **Beibehaltung der Segmentlänge**

- ▶ Minimale Rahmenlänge von 64 Byte beibehalten → Reduktion der Netzausdehnung auf 20 Meter wäre erforderlich!
- ▶ Daher: größere minimale Rahmenlänge aber Beibehaltung des alten Rahmenformats durch eine *Carrier Extension*



- Padding auf 64 Byte innerhalb des Rahmens
- Padding auf 520 Byte außerhalb des Rahmens
- Entfernung des „nodata“-Anteils bei Übergang von Gigabit- in Fast-Ethernet

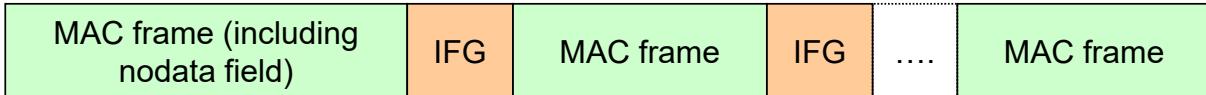
Um zu vermeiden, dass bei erneuter Steigerung der Datenrate um den Faktor 10 die Ausdehnung des Netzes für CSMA/CD um den Faktor 10 reduziert werden muss, hat man zu einem schmutzigen Trick gegriffen: führe ein zweites Padding-Feld außerhalb des Rahmens ein. Ist ein Rahmen kleiner als 64 Byte, wird das innere Padding-Feld genutzt, um den Rahmen auf 64 Byte aufzufüllen. Danach (oder falls ein Rahmen zwischen 64 und 520 Byte Größe hat) wird Padding hinter dem Rahmen vorgenommen (Nodata-Feld). Dieses Vorgehen wird Carrier Extension genannt.

Hierdurch ermöglicht man gemischten Betrieb von Fast- und Gigabit-Ethernet. Wird ein Rahmen < 520 Byte von einer Fast-Ethernet-Station über einen Switch (der beide Varianten spricht) an eine Gigabit-Station weitergeleitet, fügt diese ein Nodata-Feld an. Umgekehrt wird solch ein Feld beim Übergang von einem Gigabit- in ein Fast-Ethernet entfernt.

Da mittlerweile wohl nur noch Switches eingesetzt werden, ist eine Kollision mehr möglich, aber der Standard führt trotzdem noch diesen Trick für die Kollisionserkennung mit sich...

Gigabit-Ethernet: Rahmenformat

- **1500 Byte maximale Rahmenlänge sehr beschränkt**
 - ▶ *Frame Bursting*: sende mehrere Rahmen ohne erneutes CSMA
 - ▶ Senden von bis zu 5 Rahmen in Folge möglich, verbunden durch *Inter-Frame Gaps* (IFGs)
 - Medium scheint für andere Stationen belegt
 - Beibehaltung des vorherigen Rahmenformats



Um den Effekt von Carrier Extension bei Versendung vieler kleiner Frames zu reduzieren, hat man Frame Bursting eingeführt: sende mehrere Rahmen, ohne zwischendurch erneut CSMA durchführen zu müssen (was im Normalfall sowieso überflüssig ist, da Switches eingesetzt werden). Statt dessen sendet man mehrere Rahmen in Folge und um diese voneinander zu trennen, werden bestimmte Bitfolgen als spezieller Interframe-Gap eingefügt. Damit scheint das Medium für konkurrierende Stationen durchgehend belegt. Ab dem zweiten Rahmen ist nun kein Carrier Extension mehr notwendig.

Benennung von Ethernet-Varianten

- **Benennung von Ethernet-Varianten:**
 - ▶ Datenrate in MBit/s (10, 100, **1000** oder 10G)
 - ▶ Übertragungstechnik (z.B. **Base** für Basisband, Broad für Broadband)
 - ▶ Maximale Segmentlänge á 100 Metern oder **Medientyp**
- **Beispiele:**
 - ▶ 10Base-T: 10 MBit/s, Basisband, Twisted-Pair-Kabel
 - ▶ 100Base-T2: 100 MBit/s, Basisband, 2 Adern des Twisted-Pair-Kabels
 - ▶ 1000Base-X: 1000 MBit/s, Basisband, Glasfaser-Kabel
- **Von der Variante hängen noch Parameter wie z.B. min. Rahmenlänge ab (Ausbreitungsgeschwindigkeiten):**
 - ▶ 1000Base-X: minimale Rahmenlänge 416 Bytes
 - ▶ 1000Base-T: minimale Rahmenlänge 520 Bytes

Ethernet-Varianten: klassisches Ethernet

Bezeichnung	Kabel/ Topologie	Segment- länge	Knoten pro Segment
10Base5 (Thick Ethernet)	Dickes Koax (Bus)	Max. 500m	Max. 100
10Base2 (Thin Ethernet)	Dünnes Koax (Bus)	Max. 185m	Max. 30
10BaseT	Twisted Pair (Stern)	Max. 100m	Max. 1.024
10BaseF	Glasfaser (Stern)	Max. 2.000m	Max 1.024



Technische Realisierungen von CSMA/CD

- **10Base5 Thick Ethernet:** Bus, dickes gelbes Koax-Kabel, 10 MBit/s Übertragungsrate, Segmentlänge = 500m, 100 Rechner pro Segment, max. 5 Segmente = 2500m Länge (wegen Signallaufzeit bzw. Slot-Time), über Repeater verbunden. Medienzugriff über Transceiver, den man ins Kabel einsticht. Extra Leitung (Attachment Unit Interface, AUI) zum Rechner (Transceiverkabel, max. 50m).
- **10Base2 Thin Ethernet (Cheapernet):** Bus, dünnes Koaxkabel, 10 Mbit/s Übertragungsrate, 185m pro Segment (höhere Dämpfung), max. 5 Segmente, 30 Rechner pro Segment, Anschluss direkt an Karte über T-BNC-Stecker (Transceiver in Adapterkarte integriert).
- **10BaseT Twisted Pair:** Sternförmige Kopplung über zentrale Komponente (z.B. Hub, Switch). 10 MBit/s Übertragungsrate, UTP-Kabel mit max. 100m von Station zur zentralen Komponente, theoretisch max. 1024 Stationen pro Segment, maximal 5 Segmente.

Ethernet-Varianten: Fast Ethernet (IEEE 802.3u)

- **Einschränkung der Topologie**

- ▶ Stern topologie
- ▶ Datenrate 100 MBit/s → Reduktion der Ausdehnung

Bezeichnung	Kabel	Segmentlänge	Vorteile
100Base-T4	Vier verdrillte Adernpaare Cat3	Max. 100m	Ältere Kabel wiederverwendbar
100Base-TX	Zwei verdrillte Adernpaare Cat5	Max. 100m	Vollduplex bei 100 Mbit/s
100Base-F	Glasfaser	Max. 800m	Vollduplex, längere Strecken

Bei Fast Ethernet wurde das Konzept von IEEE 802.3 (insbesondere das Zugriffsverfahren) übernommen und die Datenrate auf 100 MBit/s erhöht. Bei einer Übertragungsrate von 100 MBit/s müsste man in einer Slot-Time von 51,2 µs nun 640 Byte senden, um das ganze Kabel zu belegen. Diese minimale Rahmengröße ist aber zu groß. Deshalb wurde die maximale Ausdehnung auf 100m beschränkt und nur noch eine sternförmige Netztopologie (Anschluss erfolgt an einen Hub oder Switch - 100m Länge zum Hub entsprechen dann 200m Netzausdehnung) zugelassen. Folgende Kabelarten werden unterstützt:

- 100 Base-T4: 4 Adernpaare (UTP-3/4/5-Kabel). Drei Adernpaare werden zur Datenübertragung genutzt, eines zur Übertragung von Kontrollinformationen. Auf jedem Adernpaar erfolgt eine Übertragung mit 33.33 Mbit/s (8B6T Code). Die Übertragung erfolgt im Halbduplex-Modus.
- 100 Base-TX: 2 Adernpaare (UTP-5-Kabel, STP-1/2-Kabel), eins pro Richtung. Man benutzt die 4B/5B-Codierung zur Übertragung (vollduplex).
- 100 Base FX: Hier verwendet man Multimode-Glasfaserkabel (jeweils eines pro Richtung) zur Vollduplexübertragung. Die maximale Länge der Übertragungsstrecke zum Hub beträgt 400m – was eigentlich länger ist als durch CSMA/CD erlaubt! Allerdings werden bei Verwendung von Glasfaser nur noch Switches eingesetzt, so dass keine Kollisionen mehr auftreten können und man auf Kollisionserkennung keine Rücksicht mehr nehmen muss.

► Weitere Steigerung der Datenrate

- Stern topologie
- Datenrate 1.000 MBit/s
- Auch UTP möglich (802.3ab)

Bezeichnung	Kabel	Segmentlänge
1000Base-T	4 ungeschirmte Adernpaare (UTP-5)	100m
1000Base-CX	2 geschirmte Adernpaare	25m
1000Base-SX	Multimode-Glasfaser	2 - 550 m
1000Base-LX	Multi-/Monomode-Glasfaser	2 - 5.500 m

Auch hier zu sehen: bei Glasfaserverkabelung wird vorausgesetzt, dass keine Kollisionen mehr verursacht werden können und die Kollisionserkennung wird ignoriert.

Gigabit Ethernet: 1000Base-T

- Wegen Verbreitung von UTP-5-Kabeln: Gigabit-Ethernet soll auch für diesen Kabeltyp möglich sein
 - ▶ Kompatibilität: Länge von bis zu 100 m soll beibehalten werden
 - ▶ 8B/10B-Codierung: 1.250 MBaud zur Übertragung notwendig!
 - ▶ Probleme mit Übersprechen, Dämpfung, ...
- Lösung: *parallele Nutzung aller vier Doppeladern*
 - ▶ Steuerung des Senders durch Switch
 - ▶ Modulationsverfahren: PAM5 (Pulsamplitudenmodulation mit fünf Zuständen)
 - Über die 4 Adernpaare kann pro Signalschritt 1 Byte übertragen werden
 - ▶ Duplexbetrieb durch Echokompensation
 - Reduktion auf 125 MBaud pro Adernpaar



Probleme beim Übergang zu Gigabit-Ethernet:

- Kompatibilität zu älteren Ethernet-Varianten sollte beibehalten werden, um ein gemischtes Zusammenschalten von Stationen zu ermöglichen. Damit darf sich auch das Rahmenformat nicht ändern, wodurch wieder die Netzausdehnung reduziert werden müsste (auf ca. 20 Meter... „Very local area network“???).
- Erreichen der nötigen Datenrate über relativ schlechte Kabel – was ermöglicht wurde, indem auf allen vier Adernpaaren parallel ein ganzes Byte codiert wird. Um Duplexbetrieb zu erreichen, werden alle vier Adernpaare simultan in beide Richtungen genutzt, was spezielle Filter nötig macht, damit die eigene Sendung nicht Störeinflüsse auf die empfangenen Signale ausübt.

Aktuelle Ethernet-Entwicklungen

- **10-Gigabit-Ethernet (802.3ae)**

- ▶ Nur noch Switch als zentrale Komponente erlaubt
 - Keine Notwendigkeit mehr für CSMA/CD
 - Aus Kompatibilitätsgründen mit implementiert (Rahmenformat, Größen)
- ▶ Zwei Varianten der Bitübertragungsschicht
 - 10 Gbit/s für LAN
 - 9,615 Gbit/s zum Übergang zu SDH (WAN)
- ▶ Anfangs nur Glasfaser, mittlerweile auch Twisted Pair

- **Mittlerweile**

- ▶ 40G-Ethernet, 100G-Ethernet
 - Beibehaltung des Rahmenformats
 - 7 Meter über Twisted Pair, bis zu 40 km über Glasfaser
- ▶ EtherCAT, Ethernet Powerlink, ...



Bei 10G-Ethernet müsste man erneut die Ausdehnung verringern, da für die höheren Datenraten eine höhere Bandbreite benötigt wird – und höhere Frequenzen werden stärker gedämpft. Ziel war: 50m bei CAT6, 100m bei CAT7. Um dies mit 10 Gigabit zu erreichen, hat man einige Änderungen auf der Bitübertragungsschicht eingeführt: zum einen wird nicht mehr ein Signal über ein Adernpaar geschickt, sondern nur noch über eine Ader. Damit kann man die Daten auf 8 Adern verteilen. Als Resultat wird aber der Einfluss von Störungen durch das Verdrillen nicht mehr reduziert – daher fügt man dem Signal vor der Übertragung ein Rauschen hinzu: die Signale werden auf Senderseite so verzerrt, dass sie nach Dämpfung und Störeinflüssen auf Empfängerseite geschätzt so ankommen, wie es gewünscht ist. Zusätzlich wird PAM16 verwenden, also ein Code mit 16 Zuständen, so dass 4 Bit pro Signal pro Ader übertragen werden können.

Des weiteren hat Ethernet sich mittlerweile auch in vielen anderen Bereichen durchgesetzt – beispielweise auch in Busnetzen, bei denen die Zustellung von Daten zeitkritisch ist oder priorisiert werden können muss (wozu früher Token Bus konzipiert wurde). Hierzu gibt es eigene Mechanismen (beispielsweise EtherCAT, Ethernet Powerlink), die zumindest das Rahmenformat von Ethernet übernehmen.

Zusammenfassung

• Aufgabe der Sicherungsschicht

- ▶ Gesicherte Übertragung von Daten zwischen benachbarten Stationen
 - Umfasst Rahmenbildung, Fehlererkennung/-behebung, Flusskontrolle, Medienzugriff
- ▶ Standardverfahren wie CRC (Fehlererkennung), Go-Back-N/Selective Repeat (Fehlerbehebung), Sliding Window (Flusskontrolle)
- ▶ Vielfältige Mechanismen zum Medienzugriff
 - Dezentral, geregelt/konkurrierend
 - Nur notwendig in Netzen mit geteiltem Medium
 - Sinnvoller Mechanismus hängt von Medium und Topologie ab
- ▶ Standard für lokale kabelgebundene Netze: Ethernet

Lessons Learned

- **Wichtige Begriffe/Konzepte des Kapitels**

- ▶ Rahmenbildung notwendig für Fehlererkennung
- ▶ Codetransparenz durch Character-/Bitstuffing
- ▶ Fehlererkennung durch Redundanzen; bevorzugtes Verfahren: CRC
- ▶ Fehlerkorrektur durch FEC oder ARQ
- ▶ Go-Back-N vs. Selective Repeat/Reject als prominente ARQ-Verfahren
- ▶ Flusskontrolle durch Sliding Window
- ▶ Medienzugriff angepasst an Medium und Topologie
 - Bevorzugt: asynchrones Zeitmultiplex, dezentral koordiniert
 - Ethernet / CSMA/CD erfolgreichstes Verfahren
 - In drahtlosen Netzen angepasste Mechanismen (CSMA/CA) notwendig