

## 5. Foliensatz Computernetze

Prof. Dr. Christian Baun

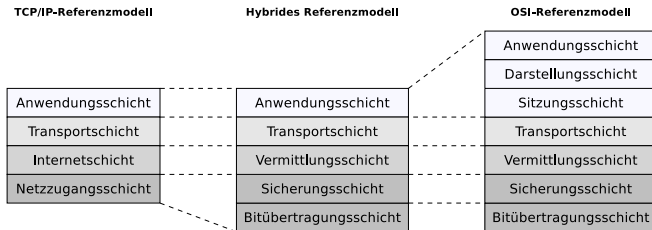
Frankfurt University of Applied Sciences  
(1971–2014: Fachhochschule Frankfurt am Main)  
Fachbereich Informatik und Ingenieurwissenschaften  
[christianbaun@fb2.fra-uas.de](mailto:christianbaun@fb2.fra-uas.de)

# Lernziele dieses Foliensatzes

- Sicherungsschicht (Teil 2)
  - Rahmen abgrenzen
    - Längenangabe im Header
    - Zeichenstopfen
    - Bitstopfen
    - Verstöße gegen Regeln des Leitungscodes
  - Rahmenformate aktueller Computernetze
  - Fehlererkennung
    - Zweidimensionale Parität
    - Zyklische Redundanzprüfung
  - Fehlerkorrektur
    - Hamming-Code
  - ~~Flusskontrolle~~
    - ~~Stop-and-Wait-Protokoll~~
    - ~~Schiebfensterprotokoll – Sliding Window Protokoll~~

# Sicherungsschicht

- Aufgaben der Sicherungsschicht (Data Link Layer):
  - Sender: Pakete der Vermittlungsschicht in Rahmen (Frames) verpacken
  - Empfänger: Rahmen im Bitstrom der Bitübertragungsschicht erkennen
  - Korrekte Übertragung der Rahmen innerhalb eines physischen Netzes gewährleisten durch Fehlererkennung mit Prüfsummen
  - Physische Adressen (MAC-Adressen) bereitstellen
  - Zugriff auf das Übertragungsmedium regeln



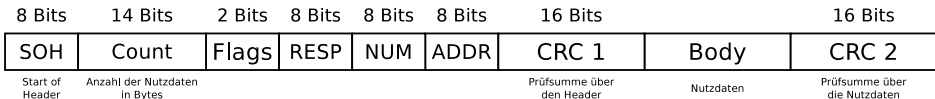
- Geräte: Bridge, Layer-2-Switch (Multiport-Bridge), Modem
- Protokolle: Ethernet, Token Ring, WLAN, Bluetooth, PPP

# Rahmen abgrenzen

- Um im Bitstrom der Bitübertragungsschicht die Rahmen zu erkennen und die Daten der Vermittlungsschicht in Rahmen zu verpacken, muss der Anfang jedes Rahmens markiert sein
  - Das ist nötig, damit der Empfänger die Rahmengrenzen erkennt
- Zur Markierung der Rahmen existieren verschiedene Möglichkeiten
  - **Längenangabe im Header**
  - **Zeichenstopfen**
  - **Bitstopfen**
  - **Verstöße gegen Regeln des Leitungscodes** der Bitübertragungsschicht mit ungültigen Signalen
- Jede der Möglichkeiten hat Vor- und Nachteile

# Längenangabe im Header

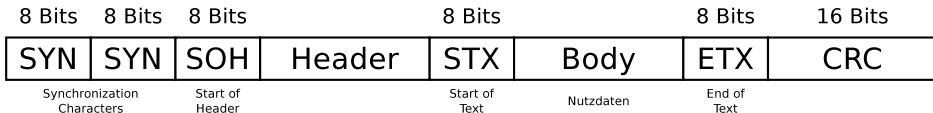
- Im Header des Rahmens steht die Anzahl der Zeichen im Rahmen
- Ein Protokoll der Sicherungsschicht, das die Rahmengrenze mit einer Längenangabe im Header angibt, ist das Byte-orientierte **Digital Data Communications Message Protocol** (DDCMP) des DECnet
- Im Rahmen befindet sich das Datenfeld Count, das die Anzahl der Nutzdaten in Bytes enthält



- Problem: Wird das Datenfeld Count während der Übertragung verändert, kann der Empfänger das Ende des Rahmens nicht mehr korrekt erkennen

## Zeichenstopfen – „Character Stuffing“ (1/2)

- Ein Protokoll, das die Rahmengrenze mit speziellen Zeichen markiert, ist das Byte-orientierte Protokoll **Binary Synchronous Communication** (BISYNC) von IBM aus den 1960er Jahren



- Steuerzeichen („**Sentinel-Zeichen**“) markieren die Struktur der Rahmen
  - Den Anfang eines Rahmens markiert das SYN-Zeichen
  - Den Anfang des Headers markiert SOH (Start of Header)
  - Die Nutzdaten sind zwischen STX (Start of Text) und ETX (End of Text)
- Ein ETX im Nutzdatenteil (Body), muss in der Sicherungsschicht durch ein DLE-Zeichen (Data Link Escape) **geschützt** (*maskiert*) werden
- Ein DLE in den Nutzdaten wird durch ein **zusätzliches** DLE maskiert

## Zeichenstopfen – „*Character Stuffing*“ (2/2)

- Das Verfahren heißt **Zeichenstopfen**, weil der...
  - Sender zusätzliche Zeichen in die Nutzdaten **einfügt** (*stopft*)
  - Empfänger *gestopfte* Zeichen aus den empfangenen Nutzdaten **entfernt**, bevor er diese an die Vermittlungsschicht übergibt
- Nachteil dieses Verfahrens:
  - Enge Anlehnung an die ASCII-Zeichenkodierung
    - Aktuellere Protokolle dieser Schicht arbeiten nicht mehr Byte-orientiert, sondern Bit-orientiert, weil damit beliebige Zeichenkodierungen verwendet werden können

## Bitstopfen – „*Bit Stuffing*“

- Bei Bit-orientierten Protokollen hat jeder Rahmen ein spezielles Bitmuster als Anfangs- und Endsequenz
- Beispiele: Das Protokoll **High-Level Data Link Control** (HDLC) und das darauf aufbauende **Point-to-Point Protocol** (PPP)

- Jeder Rahmen beginnt und endet mit: 01111110

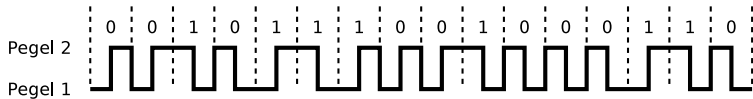


- Entdeckt das HDLC-Protokoll in der Sicherungsschicht. . .
  - des Senders 5 aufeinanderfolgende Einsen im Bitstrom von der Vermittlungsschicht, *stopft* es eine Null in den Bitstrom
  - des Empfängers 5 aufeinanderfolgende Einsen und eine Null im Bitstrom von der Bitübertragungsschicht, wird das *gestopfte* Null-Bit entfernt
- Vorteile:
  - Anfangs- und Endsequenz kommen garantiert nicht in den Nutzdaten vor
  - Beliebige Zeichensätze können verwendet werden

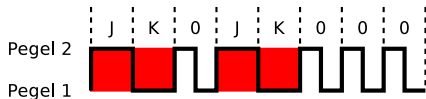


# Verstöße gegen Regeln des Leitungscode

- Abhängig vom verwendeten Leitungscode in der Bitübertragungsschicht können ungültige Signale Rahmen abgrenzen
  - Beispiel: Token Ring verwendet die differentielle Manchesterkodierung
    - Bei diesem Leitungscode findet in jeder Bitzelle ein Pegelwechsel zur Taktrückgewinnung statt



## Starting Delimiter bei Token Ring



- Bei Token Ring beginnen Rahmen mit einem Byte (**Starting Delimiter**), das **4 Verstöße gegen die Kodierregeln** enthält

Das vorletzte Byte (**Ending Delimiter**) eines Rahmens bei Token Ring enthält die gleichen 4 Verstöße gegen die Kodierregeln



# Rahmenformate aktueller Computernetze (2/8) – Ethernet



- Die Datenfelder für die physischen Adressen (MAC-Adressen) von Sender und Ziel sind jeweils 6 Bytes lang
- Der 4 Bytes lange optionale VLAN-Tag enthält unter anderem...
  - eine 12 Bits lange VLAN-ID (⇒ Foliensatz 4)
  - und ein 3 Bits großes Feld zur Priorisierung
- Das Datenfeld Typ enthält das verwendete Protokoll der nächsthöheren Schicht
  - Bei IPv4 hat das Datenfeld Typ den Wert 0x0800
  - Bei IPv6 hat das Datenfeld Typ den Wert 0x86DD
  - Enthalten die Nutzdaten eine ARP-Nachricht, hat das Datenfeld Typ den Wert 0x0806

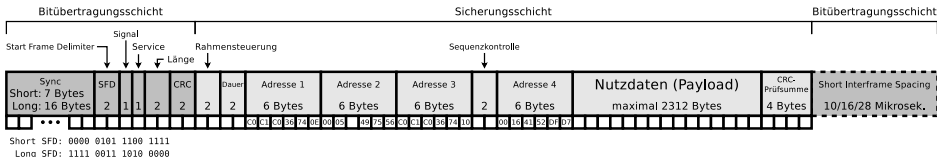


# Rahmenformate aktueller Computernetze (4/8) – Ethernet



- Das **Interframe Spacing** bzw. **Interframe Gap** ist der minimale zeitliche Abstand zwischen 2 gesendeten Rahmen auf dem Übertragungsmedium
- Die minimale Wartezeit entspricht 96 Bitzeiten (12 Bytes)
  - Bei 10 MBit/s sind es 9,6 Mikrosekunden
  - Bei 100 MBit/s sind es 0,96 Mikrosekunden
  - Bei 1 GBit/s sind es 96 Nanosekunden
- Einige Netzwerkgeräte erlauben es den Interframe Spacing zu verkürzen
  - Vorteil: Höherer Datendurchsatz ist möglich
  - Nachteil: Eventuell werden die Grenzen von Rahmen nicht korrekt erkannt (⇒ Kollisionen nehmen zu)

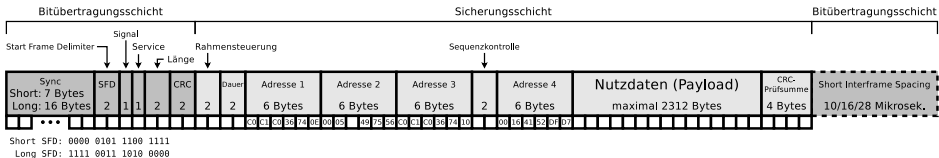
# Rahmenformate aktueller Computernetze (5/8) – WLAN



- In der Bitübertragungsschicht fügt das Protokoll folgendes hinzu:
  - eine Präambel zur Synchronisierung des Empfängers inklusive **SFD**
    - Das **Short Preamble Format** ist ein optionaler Standard den nicht alle Geräte unterstützen
  - ein Datenfeld **Signal**, das die Datenrate (MBit/s) angibt
  - ein Datenfeld **Service**, das spezielle Informationen zur verwendeten Modulation enthalten kann
  - ein Datenfeld **Länge**, das die zur Übertragung des Rahmens benötigte Zeit in Mikrosekunden enthält
  - ein Datenfeld **CRC**, das eine Prüfsumme über die Felder Signal, Service und Länge enthält

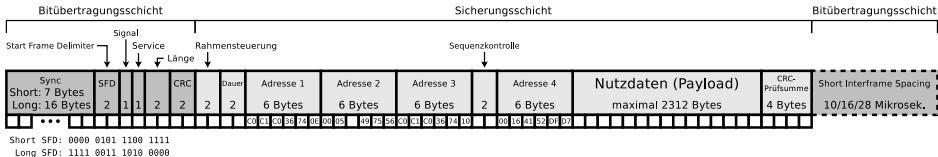
# Rahmenformate aktueller Computernetze (6/8) – WLAN

- Maximale Größe eines WLAN-Rahmen (nur der Teil der Sicherungsschicht): 2346 Bytes



- Die Rahmensteuerung (2 Bytes) enthält mehrere kleinere Datenfelder
  - Hier wird unter anderem die Protokollversion definiert und angegeben, um was für eine Art Rahmen (z.B. Datenrahmen oder Beacon) es sich handelt und ob der Rahmen mit dem WEP-Verfahren verschlüsselt ist
- Das 2 Bytes lange Datenfeld Dauer enthält einen Wert für die Aktualisierung der Zählvariable **Network Allocation Vector (NAV)**

# Rahmenformate aktueller Computernetze (7/8) – WLAN



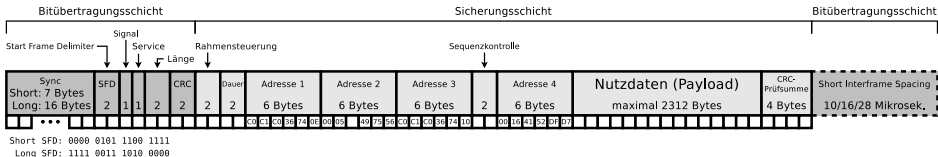
- Der Inhalt der 4 Adressfelder hängt davon ab, ob es ein WLAN im **Infrastruktur-Modus** oder im **Ad-hoc-Modus** ist und ob der Rahmen von einem Endgerät oder von einer Basisstation gesendet wurde
- Die MAC-Adressen heißen in diesem Kontext **Basic Service Set Identifier** (BSSID)

Mögliche Werte (MAC-Adressen) in den Adressfeldern sind

- Adresse des Senders (**Source Address**)
- Adresse des Empfängers (**Destination Address**)
- Adresse der nächsten Station auf dem Weg zum Ziel (**Receiver Address**)
- Adresse der letzten Station, die den Rahmen weitergeleitet hat (**Transmitter Address**)



# Rahmenformate aktueller Computernetze (8/8) – WLAN



- Das 2 Bytes große Datenfeld Sequenzkontrolle besteht aus einer 4 Bits langen Fragmentnummer und einer 12 Bits langen Sequenznummer
  - Wurde ein Rahmen in mehrere Fragmente unterteilt, ist die Sequenznummer für alle Fragmente gleich
- Abschließend folgt eine 32 Bits lange CRC-Prüfsumme, die aber nicht die Nutzdaten einschließt

# Fehlerursachen

- Bei der Übertragung von Bitfolgen kann es zu Fehlern kommen
- Typische Fehlerursachen sind
  - **Signalverformung**
    - Dämpfungseffekt des Übertragungsmediums
  - **Rauschen**
    - Thermisches und elektronisches Rauschen
  - **Nebensprechen**
    - Unerwünschter Einfluss benachbarter Kanäle (z.B. kapazitive Kopplung)
    - Kapazitive Kopplung nimmt mit wachsender Frequenz zu
  - **Kurzzeitstörungen**
    - Mehrere Bits hintereinander fehlerhaft (Burstfehler)
    - Kosmische Strahlung
    - Mangelhafte bzw. nicht ausreichende Isolierung
- Fehler müssen mindestens erkannt werden
  - Je nach Anwendungsfall kann es auch nötig sein, das Fehler korrigiert werden

# Fehlererkennung

- Zur Fehlererkennung wird jedem Rahmen vom Sender eine **Prüfsumme** angefügt
  - So werden fehlerhafte Rahmen vom Empfänger erkannt und verworfen

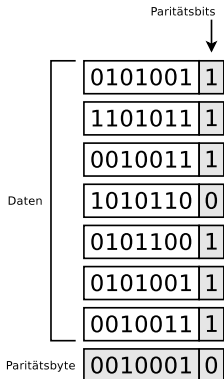
Das erneute Anfordern verworfener Rahmen durch den Empfänger sehen die Protokolle der Sicherungsschicht nicht vor

Diese Funktionalität erbringt das Transportprotokoll TCP ( $\implies$  Foliensatz 9)

- Möglichkeiten der Fehlererkennung:
  - **Zweidimensionale Parität**
  - Zyklische Redundanzprüfung – **Cyclic Redundancy Check (CRC)**

## Zweidimensionale Parität (1/2)

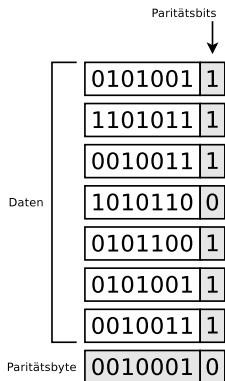
- Dieses Verfahren eignet sich dann besonders gut, wenn die 7-Bit-Zeichenkodierung US-ASCII verwendet wird



- Zu jedem 7-Bit-Abschnitt wird ein zusätzliches **Paritätsbit** addiert, um die Anzahl der Einsen im Byte auszugleichen
  - Definiert das Protokoll die gerade Parität, erhält das Paritätsbit bei Bedarf den Wert Eins oder Null, um eine gerade Anzahl von Einsen im Byte zu erwirken
  - Ist ungerade Parität gewünscht, erhält das Paritätsbit bei Bedarf den Wert Eins oder Null, um eine ungerade Anzahl von Einsen im Byte zu erwirken

⇒ **eindimensionale Parität**

## Zweidimensionale Parität (2/2)



- Neben den Paritätsbits in jedem Byte existiert für jeden Rahmen noch ein zusätzliches **Paritätsbyte**

- Der Inhalt des Paritätsbyte ist das Ergebnis der Paritätsberechnung quer über jedes Byte des Rahmens

⇒ **zweidimensionale Parität**

- Zweidimensionale Parität kann alle 1-, 2-, 3-Bit-Fehler und die meisten 4-Bit-Fehler erkennen

Quelle: Computernetzwerke, *Larry L. Peterson, Bruce S. Davie*, dpunkt (2008)

- Das Verfahren wird unter anderem von BISYNC bei der Übertragung von Daten verwendet, die mit der 7-Bit-Zeichenkodierung US-ASCII kodiert sind

# Zyklische Redundanzprüfung

- Zyklische Redundanzprüfung – Cyclic Redundancy Check (CRC) basiert darauf, dass man Bitfolgen als Polynome mit den Koeffizienten 0 und 1 schreiben kann
- Ein Rahmen mit  $k$  Bits wird als Polynom vom Grad  $k - 1$  betrachtet
  - Das werthöchste Bit ist der Koeffizient von  $x^{k-1}$
  - Das nächste Bit ist der Koeffizient für  $x^{k-2}$
  - ...

- Beispiel: Die Bitfolge 10011010 entspricht also dem Polynom:

$$\begin{aligned} M(x) &= 1 * x^7 + 0 * x^6 + 0 * x^5 + 1 * x^4 + 1 * x^3 + 0 * x^2 + 1 * x^1 + 0 * x^0 \\ &= x^7 + x^4 + x^3 + x^1 \end{aligned}$$

- Das Senden und Empfangen von Nachrichten kann man sich als **Austausch von Polynomen** vorstellen

## Polynome in der Mathematik

Ein Polynom ist die Summe von Vielfachen von Potenzen mit natürlichzahligen Exponenten einer Variablen

# Zyklische Redundanzprüfung (Vorgehensweise)

- Das Protokoll der Sicherungsschicht legt ein **Generatorpolynom** bzw. **Divisor-Polynom**  $C(x)$  fest
  - Die Auswahl des Generatorpolynoms legt fest, welche Fehlerarten erkannt werden
- $C(x)$  ist ein Polynom vom Grad  $k$ 
  - Sei z.B.  $C(x) = x^3 + x^2 + x^0 = 1101$ , dann ist  $k = 3$ 
    - Das Generatorpolynom ist also vom Grad 3

Der Grad des Generatorpolynoms entspricht der Anzahl der Bits minus eins

- Soll für einen Rahmen die CRC-Prüfsumme berechnet werden, werden  $n$  Nullen an den Rahmen angehängt
  - $n$  entspricht dem Grad des Generatorpolynoms

## Auswahl gängiger Generatorpolynome

Name	$C(x)$	Anwendung
CRC-5	$x^5 + x^2 + x^0$	USB
CRC-8	$x^8 + x^2 + x^1 + x^0$	ISDN
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + x^0$	Ethernet

# Beispiel zur zyklischen Redundanzprüfung (1/4)

Generatorpolynom:	100110
-------------------	--------

- Das Generatorpolynom hat 6 Stellen
  - Also werden 5 Nullen ergänzt

Rahmen (Nutzdaten):	10101
Rahmen mit Anhang:	1010100000

- Der Rahmen mit Anhang wird von links nur mit XOR durch das Generatorpolynom dividiert
  - $1 \text{ XOR } 1 = 0$ ,  $1 \text{ XOR } 0 = 1$ ,  $0 \text{ XOR } 1 = 1$ ,  $0 \text{ XOR } 0 = 0$
  - Dabei fängt man immer mit der ersten gemeinsamen 1 an
  - Der Rest (Restpolynom) ist die **Prüfsumme**

Der Sender  
berechnet  
die Prüfsumme

```

1010100000
100110||||
-----vv||
   110000||
   100110||
   -----v|
      101100|
      100110|
      -----v
          10100 = Rest
    
```



## Beispiel zur zyklischen Redundanzprüfung (2/4)

- Die Prüfsumme wird an die Nutzdaten angehängt
  - Der Rest muss aus  $n$  Bits bestehen
  - $n$  ist der Grad des Generatorpolynoms
- Ergebnis: 10100 wird an den Rahmen angehängt
- Übertragener Rahmen inkl. Prüfsumme (Codepolynom): 1010110100

Generatorpolynom:	100110
Rahmen (Nutzdaten):	10101
Rahmen mit Anhang:	1010100000
Rest (Restpolynom):	10100
Übertragener Rahmen (Codepolynom):	1010110100

### Den Rest auffüllen

- Der Rest muss aus  $n$  Bits bestehen und  $n$  ist der Grad des Generatorpolynoms
- Wenn der Rest kürzer ist als  $n$  muss mit Nullen *aufgefüllt* werden
- Beispiel: Der Rest ist 101 und  $n$  ist 5, dann ist die anzuhängende Prüfsumme 00101

## Beispiel zur zyklischen Redundanzprüfung (3/4)

Übertragener Rahmen (Codepolynom):	1010110100
Generatorpolynom:	100110

Der Empfänger  
testet, ob die  
Übertragung  
korrekt war

- Der Empfänger des Rahmens kann überprüfen, ob dieser korrekt angekommen ist
- Mit einer Division (ausschließlich via XOR) durch das Generatorpolynom werden fehlerhafte Übertragungen erkannt
  - Bei der Division mit XOR immer mit der ersten gemeinsamen 1 anfangen!
- Ist der Rest der Division gleich null, war die Übertragung fehlerfrei

```

1010110100
100110||||
-----vv||
      110101||
      100110||
      -----v|
          100110|
          100110|
          -----v
              0
    
```

# Beispiel zur zyklischen Redundanzprüfung (4/4)

Übertragener Rahmen (Codepolynom):	1 <b>1</b> 10110100
Generatorpolynom:	100110
<b>Korrekte Übertragung:</b>	1010110100

- Ist im übertragenen Rahmen 1 Bit fehlerhaft, ist der Rest der Division mit XOR durch das Generatorpolynom ungleich null
- Das Verfahren erkennt neben allen Einbitfehlern auch jede ungerade Anzahl fehlerhafter Bits und alle Bündelfehler der Länge  $n$ 
  - $n$  ist der Grad des Generatorpolynoms

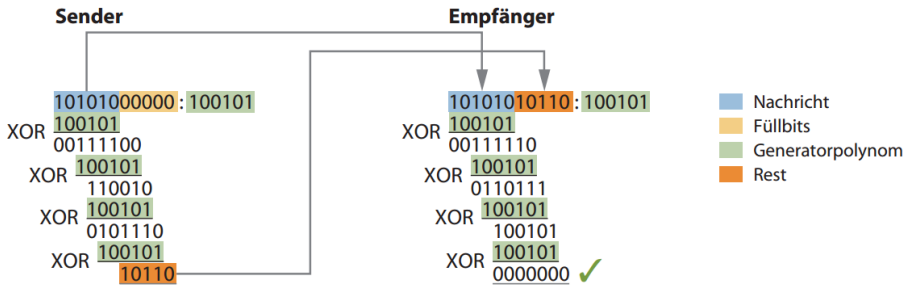
```

1110110100
100110|||
-----v|||
111010|||
100110|||
-----v||
111001||
100110||
-----v|
111110|
100110|
-----v
110000
100110
-----
10110
  
```

## Quellen

- Hompel, Büchter, Franzke. *Identifikationssysteme und Automatisierung*. Springer. 2008. S.219-221
- Meinel, Sack *Digitale Kommunikation: Vernetzen, Multimedia, Sicherheit*. Springer. 2009. S.122-124

# Zusammenfassung zur CRC-Vorgehensweise



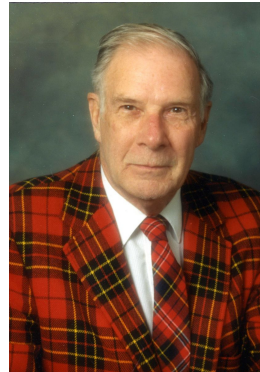
Der CRC-Prüfwert ist der Rest der Division der Nachricht durch das Generatorpolynom. Dieselbe Rechnung mit der Nachricht plus angehängtem CRC ergibt beim Empfänger 0, wenn kein Übertragungsfehler aufgetreten ist.

Quelle: CRCs berechnen mit Intrinsic. Oliver Lau. c't 9/2013. S.184-185

# Fehlerkorrektur

Bildquelle: <http://amturing.acm.org/images/hamming-3.jpg>

- Übertragungsfehler können mit CRC-Prüfsummen erkannt werden
  - Sollen Fehler auch korrigiert werden können, müssen die zu übertragenden Daten entsprechend kodiert werden
- Fehlerkorrektur kann man mit dem **Hamming Code** realisieren
  - Benannt nach dem Mathematiker Richard Wesley Hamming (1915-1998)



Richard Hamming (1915 – 1998)

Kein etabliertes Protokoll der Sicherungsschicht ermöglicht Fehlerkorrektur

Die etablierten Protokolle ermöglichen nur Fehlererkennung!

# Hamming-Code – Vorgehen: Position der Prüfbits

In dieser Vorlesung wird eine vereinfachte Version des Hamming-Codes demonstriert, die an manchen Stellen Hamming-ECC heißt

<http://www.mathcs.emory.edu/~cheung/Courses/455/Syllabus/2-physical/errors-Hamming.html>

<http://de.wikipedia.org/w/index.php?title=Fehlerkorrekturverfahren>

Andrew Tanenbaum. *Computernetzwerke*. Pearson (2000). 3. Auflage. S. 208-209

- Bits einer Bitfolge werden beginnend mit 1 von links durchnummeriert
  - Bits, die Potenzen von 2 sind (1, 2, 4, 8, 16, usw.) sind Prüfbits
    - Die übrigen Bits sind die Nutzdaten
- Beispiel:
  - 8 Bit Nutzdaten: 01001100

Position:	1	2	3	4	5	6	7	8
Nutzdaten:	0	1	0	0	1	1	0	0

Position:	1	2	3	4	5	6	7	8	9	10	11	12
zu übertragende Daten:	?	?	0	?	1	0	0	?	1	1	0	0

- Nun muss der Sender die Werte der Prüfbits berechnen

## Hamming-Code – Vorgehen: Werte der Prüfbits (1/2)

- Jeder Bit-Position in der Übertragung wird die Positionsnummer zugeordnet
- Die Positionsnummer ist in diesem Beispiel 4-stellig, da wir 4 Prüfbits haben
- Beispiele:

Position: 1     $\implies$     Wert: 0001

Position: 2     $\implies$     Wert: 0010

Position: 3     $\implies$     Wert: 0011

Position: 4     $\implies$     Wert: 0100

Position: 5     $\implies$     Wert: 0101

...

# Hamming-Code – Vorgehen: Werte der Prüfbits (2/2)

- Als nächstes werden die Werte derjenigen Nutzdatenpositionen, die 1 in unserer Übertragung sind, mit XOR zusammen gerechnet
  - Im Beispiel Positionsnummer 5, Positionsnummer 9 und Positionsnummer 10

	Position:	1	2	3	4	5	6	7	8	9	10	11	12
zu übertragende Daten:		?	?	0	?	1	0	0	?	1	1	0	0

0101	Position 5
1001	Position 9
XOR 1010	Position 10
-----	
=	0110

- Das Ergebnis sind die Werte der Prüfbits
  - Diese werden in die Übertragung eingefügt

	Position:	1	2	3	4	5	6	7	8	9	10	11	12
zu übertragende Daten:		0	1	0	1	1	0	0	0	1	1	0	0



## Hamming-Code – Vorgehen: Empfangene Daten prüfen (1/2)

- Der Empfänger kann überprüfen, ob eine Bitfolge korrekt ist
  - Er verknüpft via XOR die berechneten und empfangenen Prüfbits
    - Die Prüfbits sind Position 1, 2, 4 und 8

empfangene Daten: 1 2 3 4 5 6 7 8 9 10 11 12  
                           0 1 0 1 1 0 0 0 1 1 0 0

```

0101   Position 5
1001   Position 9
XOR 1010   Position 10
-----
0110   Prüfbits berechnet
XOR 0110   Prüfbits empfangen
-----
= 0000   => Korrekte Übertragung
    
```

## Hamming-Code – Vorgehen: Empfangene Daten prüfen (2/2)

empfangene Daten: 1 2 3 4 5 6 7 8 9 10 11 12  
                          0 1 0 1 0 0 0 0 1 1 0 0

```

      1001   Position 9
XOR 1010   Position 10
-----
      0011   Prüfbits berechnet
XOR 0110   Prüfbits empfangen
-----
=   0101   => Bit 5 ist falsch!
    
```

- Mögliche Ergebnisse der Berechnung:
  - Positionsnummer des veränderten Bits
  - 0 wenn die Übertragung fehlerfrei war
- Wurden  $\geq 2$  Bits verändert, kann nur noch eine Aussage darüber getroffen werden, dass Bits verändert wurden
  - Die Positionen können so nicht ermittelt werden

## Fazit zur Fehlererkennung

- Kodierungen mit Fehlerkorrektur sind da sinnvoll, wo Sendewiederholungen nicht angefordert werden können
  - Beispiel: Das Netzwerk bietet nur einen Kanal und die Übertragungsart ist Simplex
- Üblicherweise wird die reine Fehlererkennung bevorzugt
  - Wird ein Fehler erkannt, wird der Rahmen einfach verworfen
    - Ein erneutes Anfordern eines fehlerhaften Rahmens sehen die etablierten Protokolle der Sicherungsschicht nicht vor!
    - Erneutes Anfordern fehlerhafter Übertragungen realisiert das Transportprotokoll TCP
- Bei **Fehlererkennung** müssen **weniger Prüfbits übertragen** werden
  - Darum ist **Fehlererkennung effizienter als Fehlerkorrektur**

Siehe Modellrechnung zur Fehlererkennung/-korrektur auf der nächsten Folie

# Modellrechnung zur Fehlererkennung/-korrektur

- Blockgröße: 1.000 Bits
- 1 Megabit Daten (1.000 Blöcke) sollen übertragen werden
- **Fehlererkennung**
  - Zum Erkennen eines fehlerhaften Blocks genügt am Ende jedes Blocks ein Paritätsbit
    - Dadurch vergrößert sich jeder Block um ein Bit auf 1.001 Bits
  - Nach je 1.000 Blöcken muss ein Extrablock (1.001 Bits) übertragen werden
  - Ein Megabit Daten erfordert daher 2.001 Prüfbits
- **Fehlerkorrektur (Hamming-Code)**
  - Um 1.000 Bits Daten korrigieren zu können, sind 10 Prüfbits nötig
    - Die Prüfbits haben die Positionen 1, 2, 4, 8, 16, 32, 64, 128, 256 und 512
  - Ein Megabit erfordert daher 10.000 Prüfbits

## Quelle

Andrew Tanenbaum, David Wetherall *Computernetzwerke*. Pearson (2012). 6. Auflage. S.253

# Übung

- ① Sie haben 8 Bits Nutzdaten (10011010) vorliegen
  - Ermitteln Sie die zu übertragenden Daten (Nutzdaten inklusive Prüfbits)
- ② Überprüfen Sie, ob die folgenden Nachrichten korrekt übertragen wurden:
  - ① 001111101
  - ② 101110100010
  - ③ 001101100100
  - ④ 0001101100101101

⇒ Übungsblatt 3

# Zuverlässige Übertragung durch Flusskontrolle

- Via Flusskontrolle steuert der Empfänger die **Sendegeschwindigkeit des Senders** dynamisch und stellt so und die **Vollständigkeit der Datenübertragung** sicher
    - Langsame Empfänger sollen nicht mit Rahmen überschüttet werden
      - Dadurch würden Rahmen verloren gehen
    - Während der Übertragung verlorene Rahmen werden erneut gesendet
  - Vorgehensweise: **Sendewiederholungen**, wenn diese nötig sind
    - Grundlegende Mechanismen:
      - **Bestätigungen** (Acknowledgements, ACK) als Feedback bzw. Quittung
      - **Zeitschranken** (Timeouts)
    - Konzepte zur Flusskontrolle:
      - **Stop-and-Wait** ( $\implies$  Foliensatz 9)
      - **Schiebefenster** (Sliding-Window) ( $\implies$  Foliensatz 9)
- Ethernet bietet keinen Mechanismus zur Flusskontrolle auf der Sicherungsschicht
  - In der Praxis wird Flusskontrolle, da wo es erfolgreich ist, vom Transportprotokoll TCP realisiert