

**Eigenschaften mobiler und  
eingebetteter Systeme:**

# **Fehlertolerante Kommunikation**

**Dr.-Ing. Matthias Werner**

**Dipl.-Inf. Jan Richling**

**Wintersemester 2001/2002**

# Wiederholung

- Für Fehlertoleranz ist Redundanz notwendig

Allgemein zwei Aufgaben:

- Fehler entdecken
- Fehler(auswirkungen) beseitigen / maskieren

# Typische Fehlermodelle in der Kommunikation

- Keine Verbindung
- Stuck-at-X in Datenleitungen
- Stuck-at-X in Steuerleitungen
- OR- oder AND- Bridge
- Burst-Error

# Fehlerfolgen von Kommunikationsfehlern (I)

- Datenbitfehler:
  - Keine unmittelbaren Auswirkungen bei Kommunikation
  - Meist als Datenfehler (Berechnungsfehler) in höherer Schicht
- Adresstag-Fehler
  - Datenpaket erreicht nicht den korrekten Adressaten
  - Empfang falscher Daten
  - Verlust von Paketen
- Stuck-at-valid-Konfiguration
  - Datenpaket erreicht nicht den korrekten Adressaten
- Unterbrochene Verbindung
  - Kompletter Datenverlust
- Kurzschluß
  - Unerwünschter Multi-/Broadcasting-Effekt

# Fehlerfolgen von Kommunikationsfehlern (II)

Die Fehlerfolgen können in drei Gruppen zusammengefaßt werden:

- Verfälschte Daten
- Verlorene Daten
- Unerwartete Daten (Nachrichten)
- Als Abstraktion dienen *fehlerhafte Kanäle*, die Daten verfälschen, verlieren, verzögern oder generieren
- Viele (Konsens-)Probleme lassen sich unter der Annahme fehlerhafter Kanäle schlecht oder gar nicht lösen.

# Grundlegende Ansätze

- Der weitaus größte Teil von Übertragungsfehlern ist temporär
- ⇒ Bei Entdeckung kann erneute Übertragung (*retransmission*) erfolgen
- Bei permanenten Fehlern hilft nur ein alternativer Kanal
- Es ergeben sich zwei Problembereiche:
  - Entdeckung (und gegebenenfalls Korrektur) von Übertragungsfehlern
  - Konfigurationen für redundante Pfade

# Codes

- Ein Code dient zur Darstellung von Daten
- Daneben kann er noch weitere Eigenschaften haben.
- Formal:

**Code und Codierung** Seien  $A^*$  und  $B^*$  die *Wortmengen* über den Alphabeten  $A$  und  $B$ . Eine *Codierung* (der Wortmenge  $A^*$ ) ist eine Abbildung  $c : A^* \rightarrow B^*$ , die jedem Wort  $w \in A^*$  ein Wort über dem Alphabet  $B$  zuordnet. Die Menge aller Codewörter, d.h. das Bild  $c(A^*) \rightarrow B^*$  nennt man den *Code*.

# Codes für Fehlertoleranz

- Eine Fehlertoleranzmaßnahme, die häufig in der Kommunikation benutzt wird, ist die Verwendung spezieller Codes.
- Ziel:
  - Erkennung von Übertragungsfehlern (fehlererkennende Codes)
  - Behebung von Übertragungsfehlern (fehlerkorrigierende Codes)
- Ein Code kann sowohl fehlererkennend als auch fehlerkorrigierend ein.
- Typischerweise können mehr Fehler erkannt als behoben werden
- Fehlertoleranzcodes benötigen – wie jede FT-Maßnahme – Redundanz
- Redundanz heißt hier in der Regel, daß mehr Bits übertragen werden, als zur Darstellung gebraucht werden.

# HAMMING-Distanz und Abdeckung

- Die HAMMING-Distanz beschreibt Fähigkeit eines Codes, fehlererkennend oder fehlerkorrigierend eingesetzt zu werden
- Die HAMMING-Distanz ist kleinste Anzahl von Bitstellen, in denen sich zwei beliebige Codewörter unterscheiden.
- „Normale“ Codes (z.B. binäres Positionssystem) haben eine HAMMING-Distanz von 1.  
Sie sind damit nicht fehlererkennend oder korrigierend

# Abdeckung

- Zur Bewertung der Fehlertoleranzeigenschaften von Codes bzw. dazugehöriger Testverfahren ist das Maß *Abdeckung* gebräuchlich
- Die Abdeckung (*coverage*) gibt den Anteil von einer gegebenen Grundgesamtheit von Fehlern an, in dem die Fehlererkennung oder Fehlerkorrektur erfolgreich ist
- Beispiel: Gegeben sei ein 8-Bit Datum, das mit einem Paritätsbit versehen wird.
  - 100% aller Ein-Bit- und Drei-Bit-Fehler werden entdeckt (Abdeckung in diesen Fällen:  $C = 1.0$ )
  - Keine Fehlerkorrektur oder Entdeckung von Zwei-Bit-Fehlern möglich ( $C = 0$ )

# Häufig benutzte Codes in der fehlertoleranten Kommunikation

- In der Kommunikation werden folgende Codes häufig eingesetzt:
  - Checksummen
  - Paritäten
  - CRC
  - HAMMING-Codes
- Betrachten die letzten beiden (ersten beiden sind wahrscheinlich bekannt)

# CRC-Code (I)

- CRC steht für *cyclic redundancy check*
- Der CRC-Algorithmus erlaubt eine automatische Generierung eines Codewortes aus dem entsprechenden Datenwort
- $(n, k)$ -Code: Das Codewort ist  $n$  Bit lang, die Daten  $k$ -Bit
- Grundidee:
  - Datenwort wird als Polynom aufgefaßt
  - Bildung des Codewortes so, daß es durch Multiplikation mit einem anderen Polynom (Generatorpolynom) entsteht
  - Der Test erfolgt durch Teilung des Codewortes durch das Generatorpolynom: Sie muß den Rest 0 haben.
  - Körper mit vereinfachten Operationen (kein Übertrag)

# CRC-Code (II)

Mathematische Grundlagen:

- Sei  $\mathcal{B} = \{0, 1\}$
- Es werden die Verknüpfungen „ $\cdot$ “ und „ $+$ “ so definiert, daß kein Übertrag berücksichtigt wird.
- $(\mathcal{B}, \cdot, +)$  bilden einen Körper  $\mathcal{B}^+$
- Über  $\mathcal{B}^+$  werden Polynome definiert. Die Menge aller Polynome  $\mathcal{B}[x]$  bilden einen *Ring*  $\Rightarrow$  es sind darin Addition, Subtraktion und Multiplikation definiert
- Division ist nicht definiert, aber Division mit Rest und damit auch die Modulo-Operation

# CRC-Code (III)

Algorithmus:

- Gegeben
  - Datenwort  $M$  der Länge  $k$  (Aufgefaßt als Polynom  $M(x)$  mit dem Grad  $k$ )
  - Generatorpolynom  $G(x)$  mit dem Grad  $r$ ,  $r = n - k$
- Hänge an  $M(x)$   $r$  Nullen an (Multipliziere  $M(x)$  mit  $x^r$ ,  $M(x) := M(x) \cdot x^r$ )
- Ermittle den Rest der Division (im Polynomring!) von  $M(x)$  durch  $G(x)$ :  
 $T(x) = M(x) \bmod G(x)$
- Addiere  $T(x)$  zu  $M(x)$ , das Ergebnis ist das Codewort

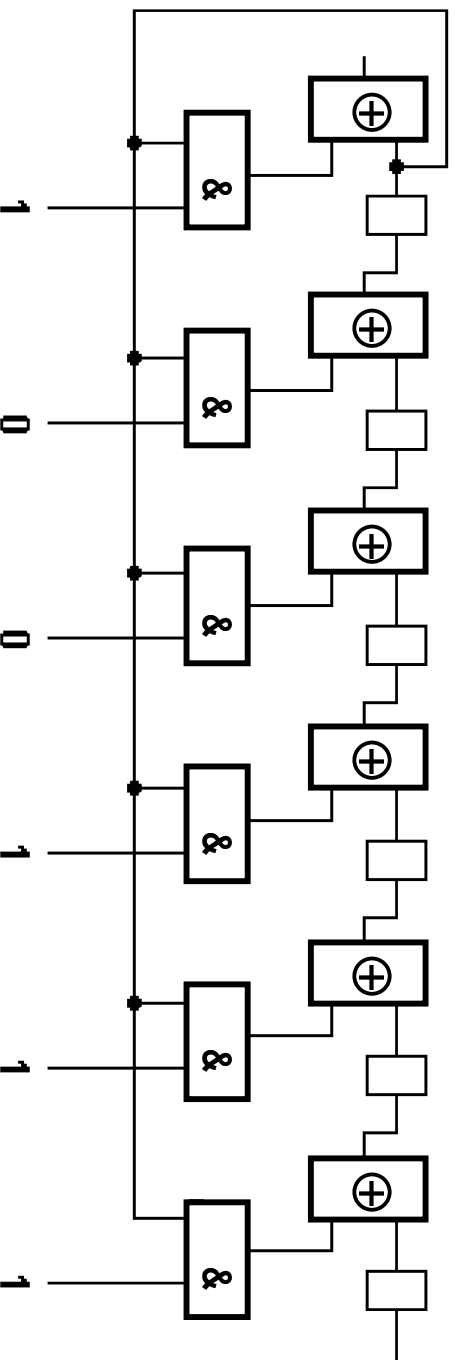
# CRC-Code (VI)

Beispiel:

- Datenwort als Bitstream: 1101011011
- Generator:  $G(x) = x^4 + x + 1$ , d.h. als Bitstream: 10011
- Datenbits + Nullen: 11010110110000
- Modulo-Operation  $11010110110000 \bmod 10011 = 1110$
- Codewort: 11010110111110

# CRC-Code (V)

- Der Vorteil der Polynommathematik: Leichte Implementation.
- Schieberegister mit XOR- und AND-Gattern



- Dient sowohl auf Sender- als auch auf Empfängerseite zur Divisionsrestermittlung
- Bei festem Generatorpolynom können die entsprechenden Gatter für 0en weggelassen werden

# CRC-Code (VI)

Was wird mit CRC erreicht?

- Ist  $x + 1$  Teiler von  $G(x)$ , so werden alle ungeraden Bitfehler erkannt
- Jeder Burst-Fehler der Länge  $\leq r$  wird erkannt
- Abhängig von  $G(x)$  werden noch mehr Fehler erkannt

Typische Generatorpolynome:

- **CRC-16 (Magnetband):**  $x^{16} + x^{15} + 1$
- **CRC-CCITT (Disketten):**  $x^{16} + x^{12} + x^5 + 1$
- **CRC-Ethernet:**  $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

# HAMMING-Codes (I)

- Idee: Codes mit einer Hamming-Distanz  $H > 1$
- Bei  $H = 2$  können alle Ein-Bit-Fehler **erkannt** werden
- Bei  $H = 3$  können alle Ein-Bit-Fehler **korrigiert** werden
- Alternativ können bei  $H = 3$  alle Ein- und Zwei-Bit-Fehler **erkannt** werden
- Es besteht also ein Tradeoff zwischen Fehlererkennung und Korrektur
- Fehlerkorrektur erfolgt durch Ersetzung des fehlerhaften Codewortes durch das „nächstliegende“ gültige Codewort

# HAMMING-Codes (II)

- Typisch:  $H = 3$
- Wie lang muß ein Codewort sein, um ein Datenwort der Länge  $m$  zu codieren?
  - HAMMING-Ungleichung:
$$2^k \geq m + k + 1$$
  - $k$  : Anzahl der zusätzlichen Bits.
- Beispiel: Für  $m = 4$  ist die Ungleichung bei  $k \geq 3$  erfüllt
- Konstruktion eines Hamming-Codes mit  $m = 4$ 
  - Drei Check-Bits,  $c_1, c_2$  und  $c_3$
  - Ein Check-Bit bildet jeweils die Parität von 3 Datenbits.

# HAMMING-Codes (III)

- Beispiel für Paritäts-Matrix:  $P =$

$$\begin{array}{cccc|cccc} & \underbrace{d_1} & \underbrace{d_2} & \underbrace{d_3} & \underbrace{d_4} & \underbrace{c_1} & \underbrace{c_2} & \underbrace{c_3} \\ \hline & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}$$

- Daraus ergibt sich:
  - $C_1 = b_1 \oplus b_2 \oplus b_3$
  - $C_2 = b_1 \oplus b_3 \oplus b_4$
  - $C_3 = b_2 \oplus b_3 \oplus b_4$

# HAMMING-Codes (IV)

- Beispiel für einen  $(7, 4)$ -Hamming-Code
- Keine zwei Codewörter in der dritten Spalte haben eine Hamming-Distanz kleiner 3.

Wert	binär	Hamming
0	0000	0000000
1	0001	0001011
2	0010	0010111
3	0011	0011100
4	0100	0100101
5	0101	0101110
6	0110	0110010
7	0111	0111001
8	1000	1000110
9	1001	1001101
10	1010	1010001
11	1011	1011010
12	1100	1100011
13	1101	1101000
14	1110	1110100
15	1111	1111111

## HAMMING-Codes (V)

- Ein empfangenes Codewort  $h$  wird überprüft durch Multiplikation modulo 2 mit der Paritäts-Matrix
- Den entstehenden Vektor  $S$  nennt man *Syndrom*
- $P \cdot h = S$
- Ist das Syndrom ein Null-Vektor, so ist die Übertragung fehlerlos (wenn die Fehlerannahme zutrifft)
- Bei einem Einzelfehler stimmt das Syndrom mit der Spalte von  $P$  überein, die der fehlerhaften Bitstelle in  $h$  entspricht.

# HAMMING-Codes (VI)

- Beispiel 1: kein Fehler

$$- d = 1110, h = H(d) = 1110100$$

$$- S = P \cdot h = \begin{vmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{vmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = [000]$$

# HAMMING-Codes (VII)

- Beispiel 1: Fehler

- $d = 1110, h = H(d) = 1110\underline{1}0$

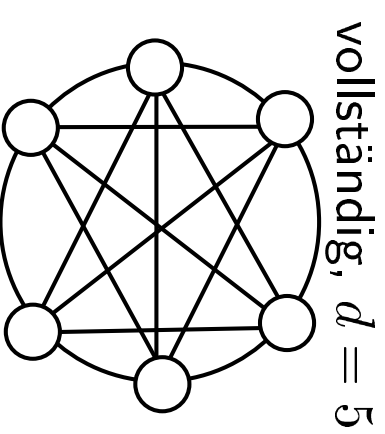
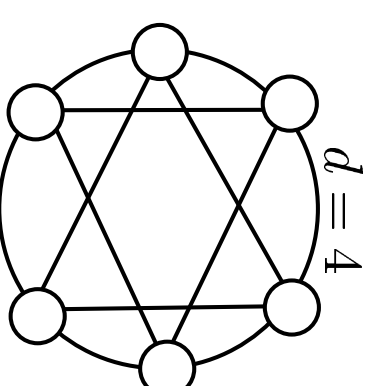
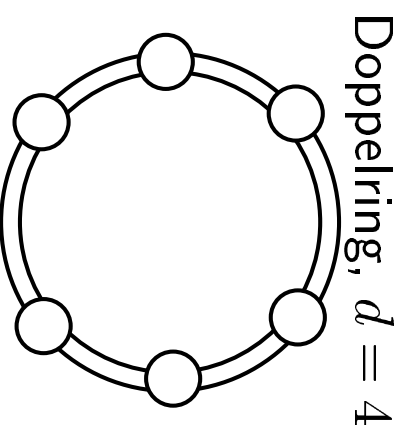
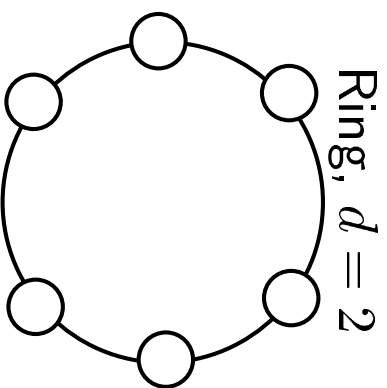
- $S = P \cdot h = \begin{vmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{vmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = [010]$

- Das Muster 0 1 0 stimmt mit der 6. Spalte von  $P$  überein, folglich liegt der Fehler bei 6. Bit von  $h$

# Redundante Pfade

- Problem: Permanente Ausfälle eines Kanals in einem Netzwerk sollen toleriert werden
- Lösung: Kommunikation über alternative Pfade
- Wieviele Kanalfehler kann ein Netzwerk tolerieren?
- $\Rightarrow$  Graphentheoretischer Ansatz: Konnektivität des Verbindungsgraphen (bei regelmässigen schlingenfremen Graphen gleich dem Grad des Graphen)

# Topologien für fehlertolerante Netzwerke

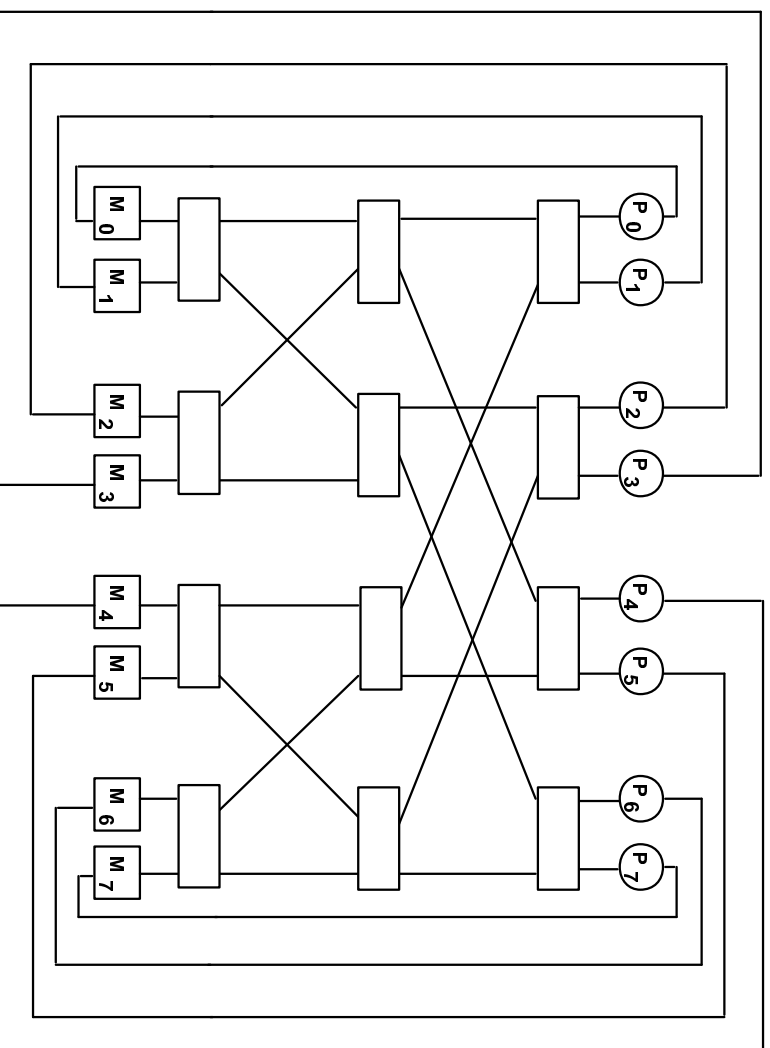


# Unregelmässige Netze

- Bei unregelmässigen Pfadgraphen ist der Grad der Fehlertoleranz nicht offensichtlich
- Lösung: Rückführung auf graphentheoretisches Standardproblem *minimaler Schnitt*
- Ein (*Kanten-*)*Schnitt* ist eine Menge von Kanten, bei deren Entfernung ein Graph nicht mehr zusammenhängend ist
- Bei Berücksichtigung von Last (Echtzeit)  $\Rightarrow$  *Flußproblem*

# Fehler in Switches (I)

- Fehler in Switches führen zu falschem „Routing“ von Nachrichten
- Test von Pfaden
- Was ist die minimale Anzahl von nötigen Test?  $\Rightarrow$  Parallelisierung
- Beispiel: Banyan



# Fehler in Switches (II)

- Mit lediglich zwei Konfigurationen können alle Switches getestet werden

