

Systeme II

7. Die Datensicherungsschicht

Thomas Janson[°], Kristof Van Laerhoven*, Christian
Ortolf[°]

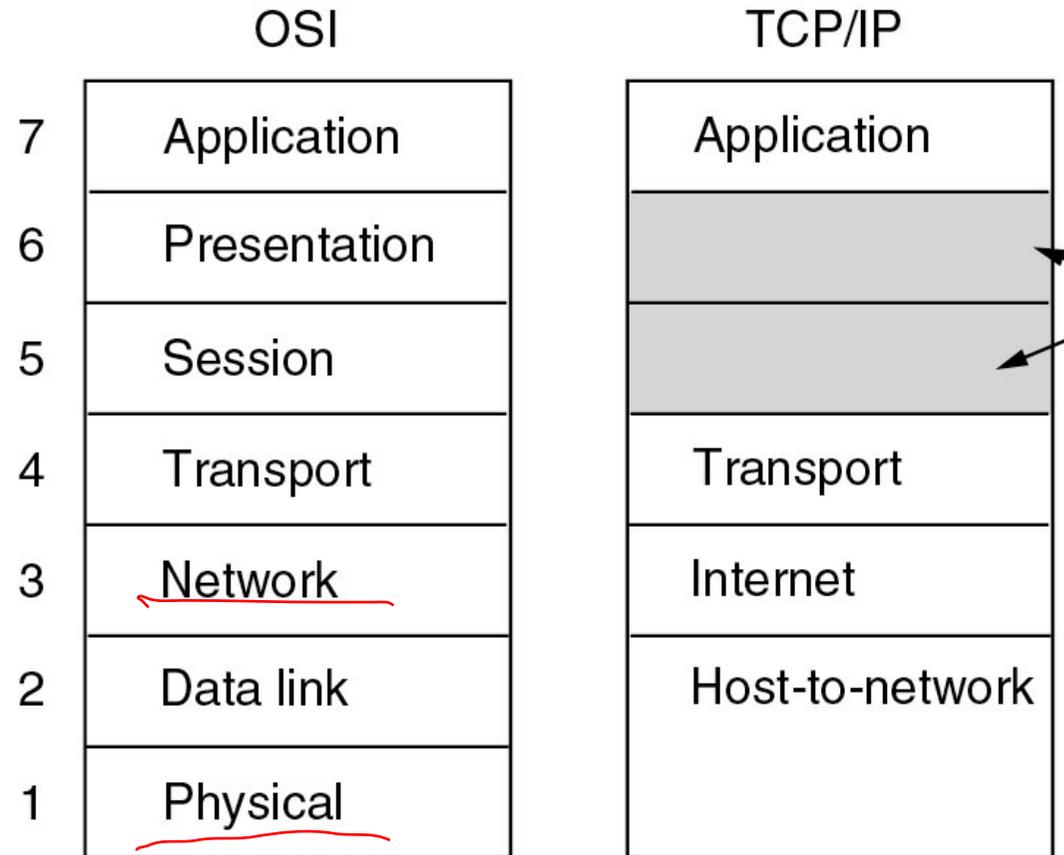
Folien: Christian Schindelhauer[°]

Technische Fakultät

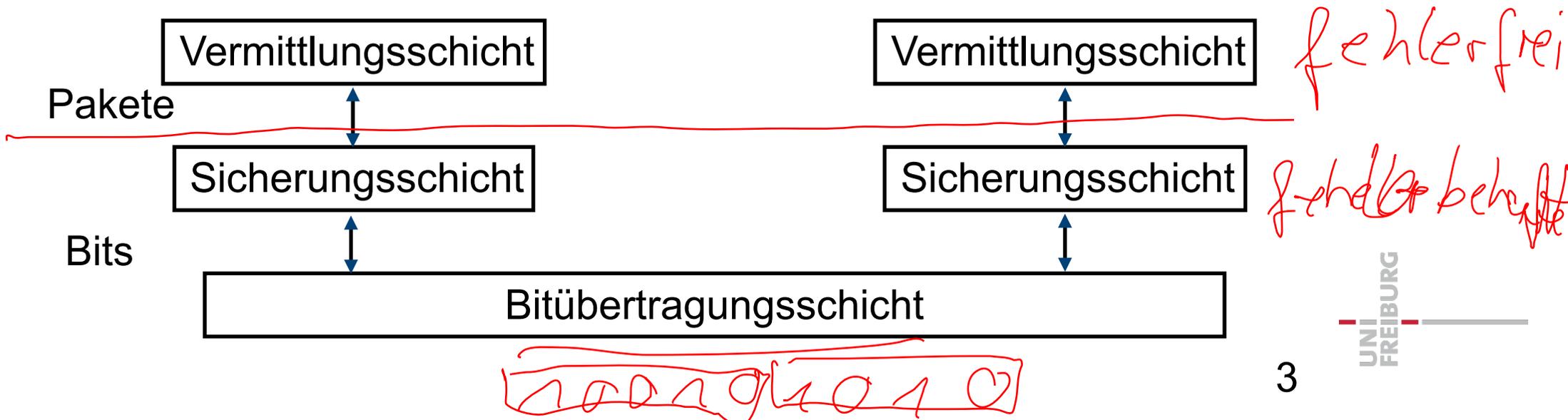
[°]: Rechnernetze und Telematik, *: Eingebettete Systeme

- Aufgaben der Sicherungsschicht (Data Link Layer)

- Dienste für die Vermittlungsschicht
- Frames
- Fehlerkontrolle
- Flusskontrolle



- Situation der Sicherungsschicht
 - Die Bitübertragungsschicht überträgt Bits
 - aber unstrukturiert und möglicherweise fehlerbehaftet
- Die Vermittlungsschicht erwartet von der Sicherungsschicht
 - fehlerfreie Übermittlung
 - Übermittlung von strukturierten Daten
 - Datenpakete oder Datenströme
 - störungslosen Datenfluss

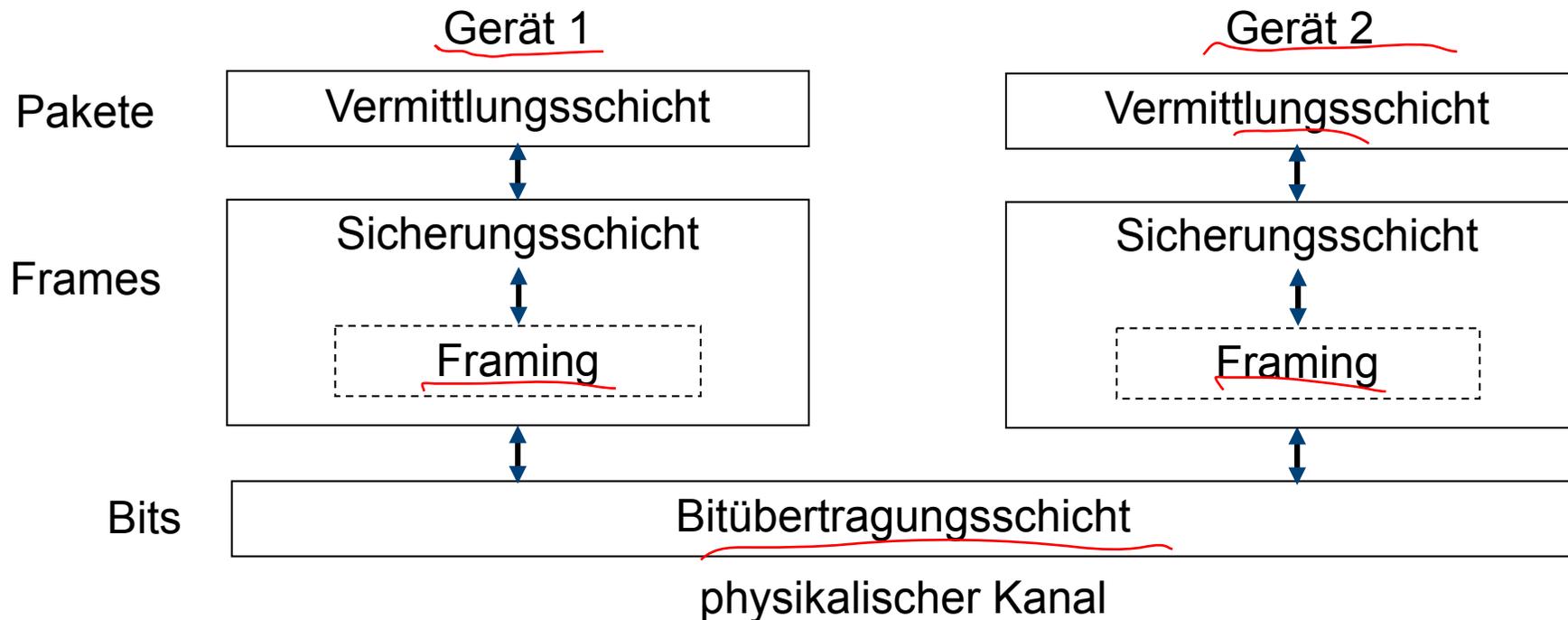


- Verlässlicher Dienst?
 - Das ausgelieferte und das empfangene Paket müssen identisch sein
 - Alle Pakete sollen (irgendwann) ankommen
 - Pakete sollen in der richtigen Reihenfolge ankommen
 - Fehlerkontrolle ist möglicherweise notwendig
- Verbindungsorientiert?
 - Ist die Punkt-zu-Punktverbindung in einem größerem Kontext?
 - Reservierung der Verbindung notwendig?
- Pakete oder Datenströme (Bitströme)?

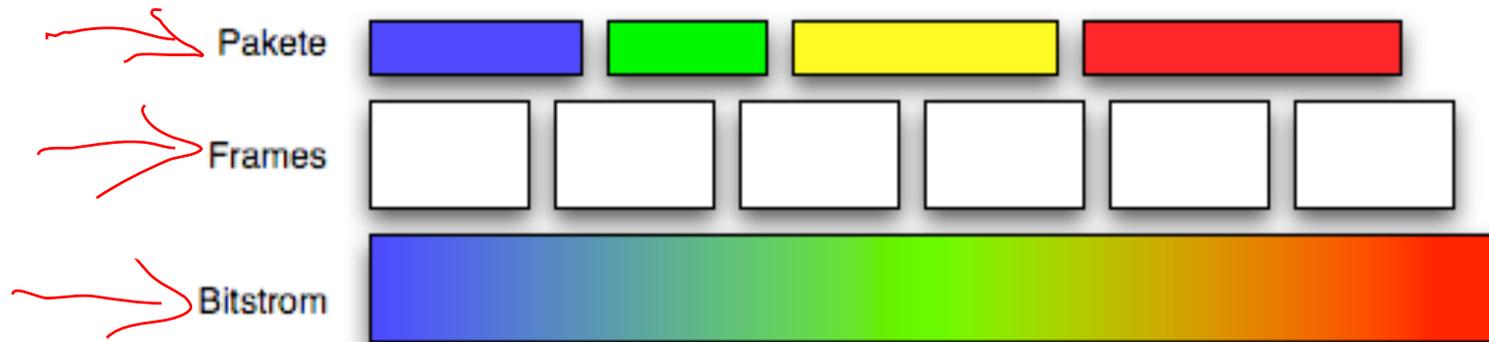
- Beispiel
 - Verbindungsloser und verlässlicher Dienst wird durch die Vermittlungsschicht gefordert
 - Sicherungsschicht verwendet intern verbindungsorientierten Dienst mit Fehlerkontrolle
- Andere Kombinationen sind möglich



- Der Bitstrom der Bitübertragungsschicht wird in kleinere "Frames" unterteilt
 - Notwendig zur Fehlerkontrolle
 - Frames sind Pakete der Sicherungsschicht
- Fragmentierung der Pakete aus Vermittlungsschicht in Frames notwendig



- Die Sicherungsschicht zwischen der Bitübertragungsschicht mit Bitstrom und der Vermittlungsschicht mit Paketen



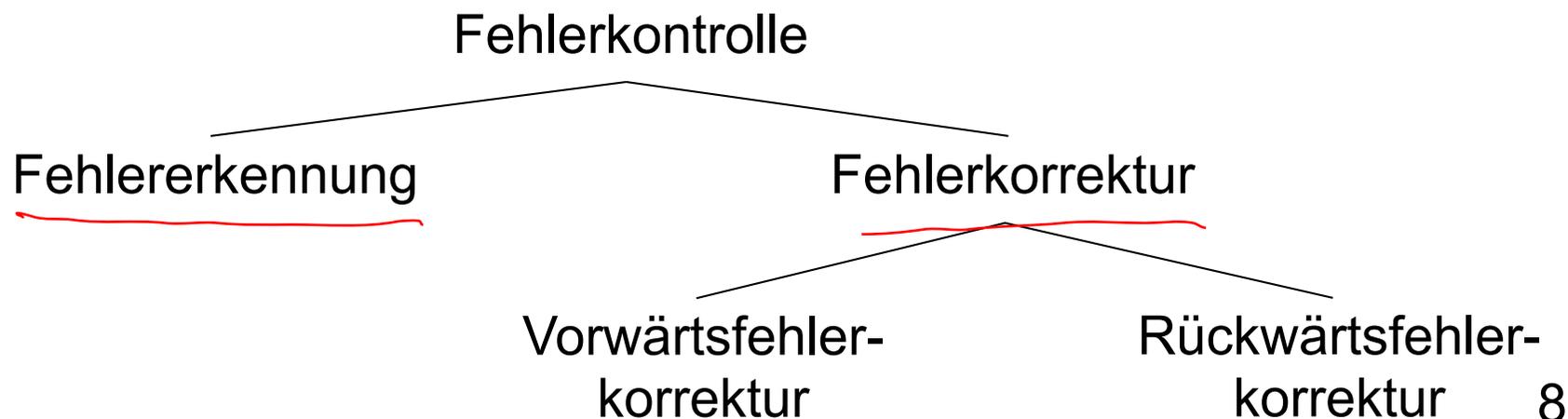
- Pakete werden in Framegröße fragmentiert



- Defragmentierung auf Empfängerseite



- Zumeist gefordert von der Vermittlungsschicht
 - Mit Hilfe der Frames
- Fehlererkennung
 - Gibt es fehlerhaft übertragene Bits?
- Fehlerkorrektur
 - Behebung von Bitfehlern
 - Vorwärtsfehlerkorrektur (Forward Error Correction)
 - Verwendung von redundanter Kodierung, die es ermöglicht Fehler ohne zusätzliche Übertragungen zu beheben
 - Rückwärtsfehlerkorrektur (Backward Error Correction)
 - Nach Erkennen eines Fehlers, wird durch weitere Kommunikation der Fehler behoben



- Verbindungsaufbau ebenfalls

- Sitzungsschicht/Anwendungsschicht, z.B. Login beim Server
- Transportschicht zwischen Endpunkten

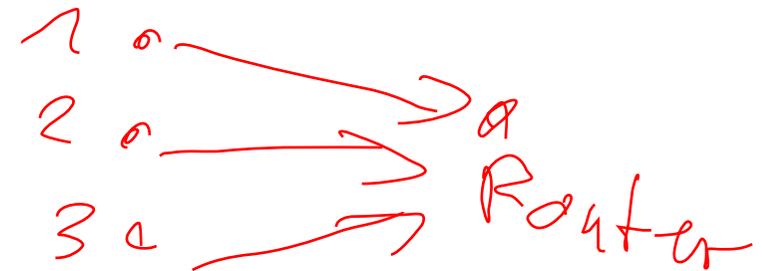
- Nutzen von Verbindungen

- Kontrolle des Verbindungsstatus
 - Korrektheit des Protokolls
- Fehlerkontrolle
 - Verschiedene Fehlerkontrollverfahren vertrauen auf gemeinsamen Kontext von Sender und Empfänger



- Aufbau und Terminierung von Verbindungen

- “Virtuelle Verbindungen”
 - Es werden keine Schalter umgelegt
 - Interpretation des Bitstroms
- Kontrollinformationen in Frames
- Besonders wichtig bei drahtlosen Medien

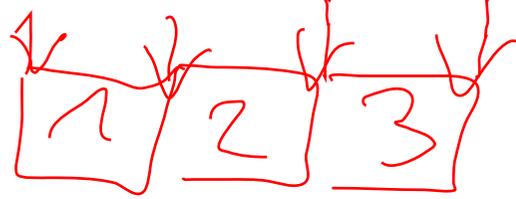


- Problem: Schneller Sender und langsamer Empfänger
 - Der Sender lässt den Empfangspuffer des Empfängers überlaufen
 - Übertragungsbandweite wird durch sinnlosen Mehrfachversand (nach Fehlerkontrolle) verschwendet
- Anpassung der Frame-Sende-Rate an dem Empfänger notwendig

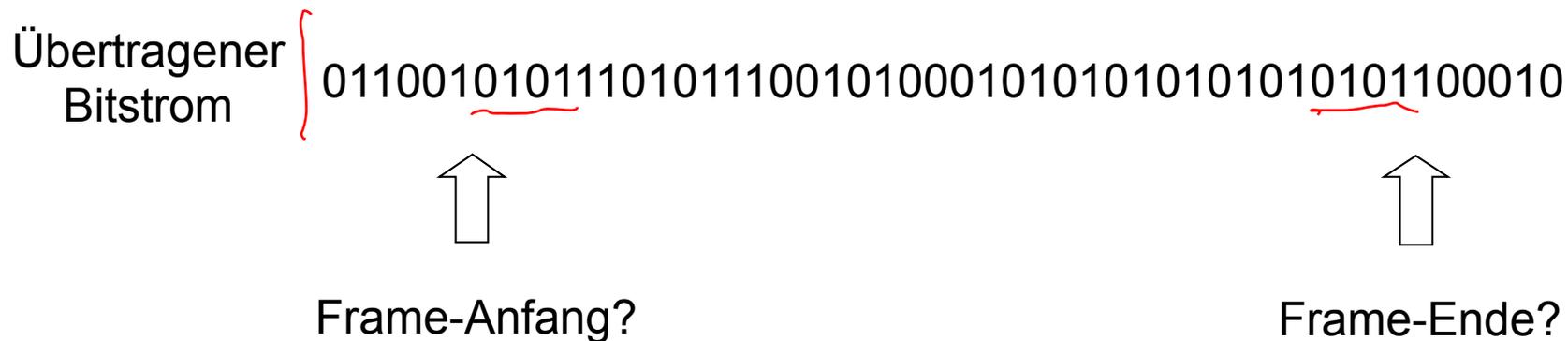
Langsamer Empfänger



Schneller Sender

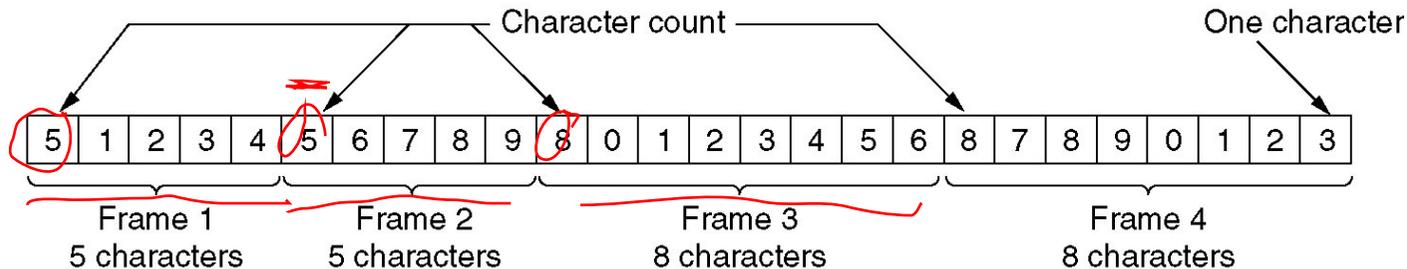


- Wo fängt der Frame an und wo hört er auf?
- Achtung:
 - Die Bitübertragungsschicht kann auch Bits liefern, wenn der Sender tatsächlich nichts sendet
 - Der Empfänger
 - könnte das Rauschen auf dem Medium interpretieren
 - könnte die Folge 00000000... liefern
 - Daten oder Kontrollinformation?

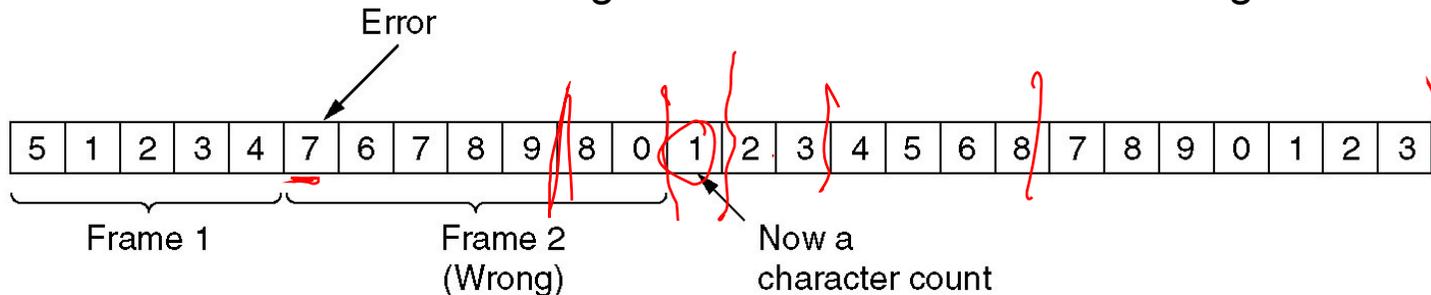


Frame-Grenzen durch Paketlängen?

- Idee: Ankündigung der Byteanzahl im Frame-Header

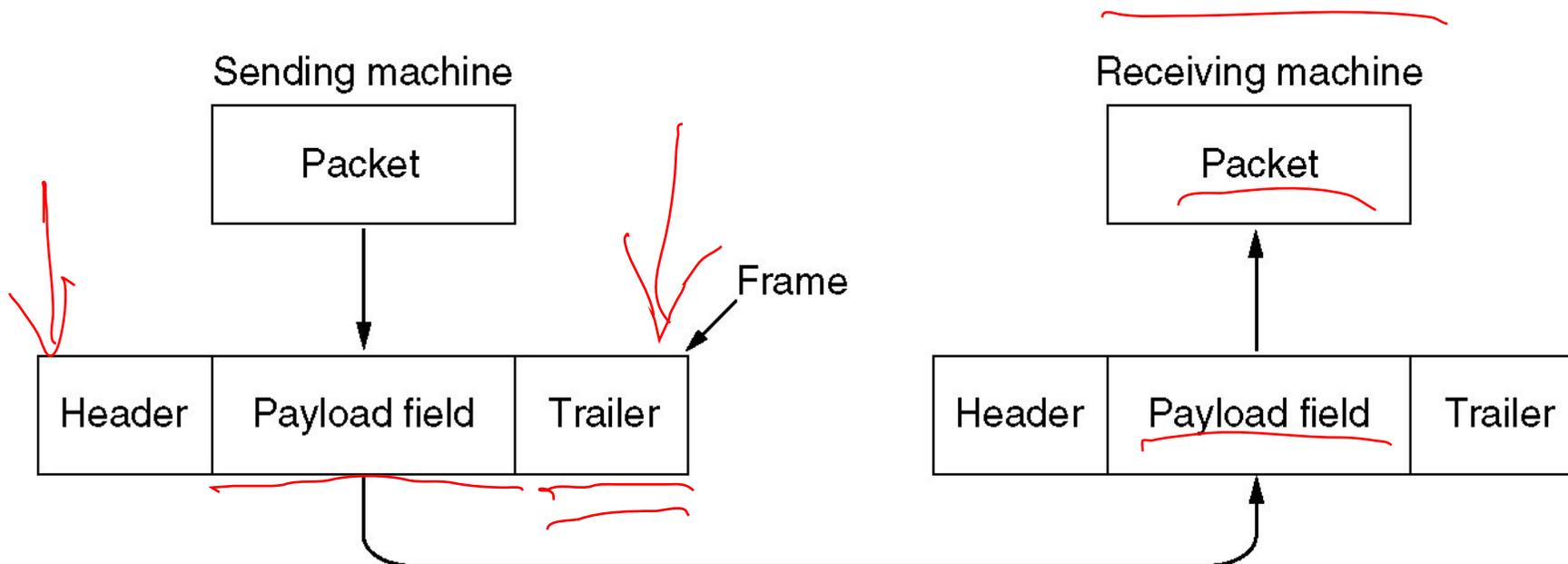


- Problem: Was, wenn die Frame-Länge fehlerhaft übertragen wird?
 - Erkennung der Framelänge vor Fehlerkorrektur
 - Empfänger verpasst Anfang des nächsten Frames
 - irgendein Daten-Byte wird als Frame-Länge interpretiert und folgende Frames falsch
 - Variable Frame-Größen mit Längeninformaton sind daher kein gutes Konzept



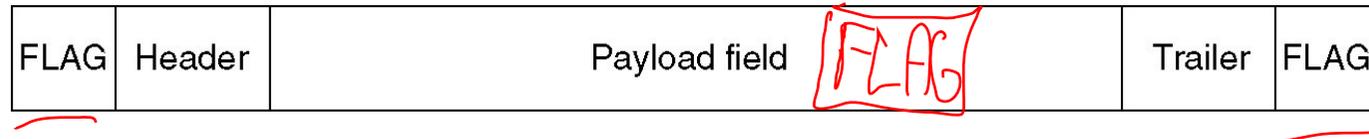
■ Header und Trailer

- Zumeist verwendet man Header am Anfang des Frames, mitunter auch Trailer am Ende des Frames
- signalisieren den Frame-Beginn und das Frame-Ende
- tragen Kontrollinformationen
 - z.B. Sender, Empfänger, Frametypen, Fehlerkontrollinformation

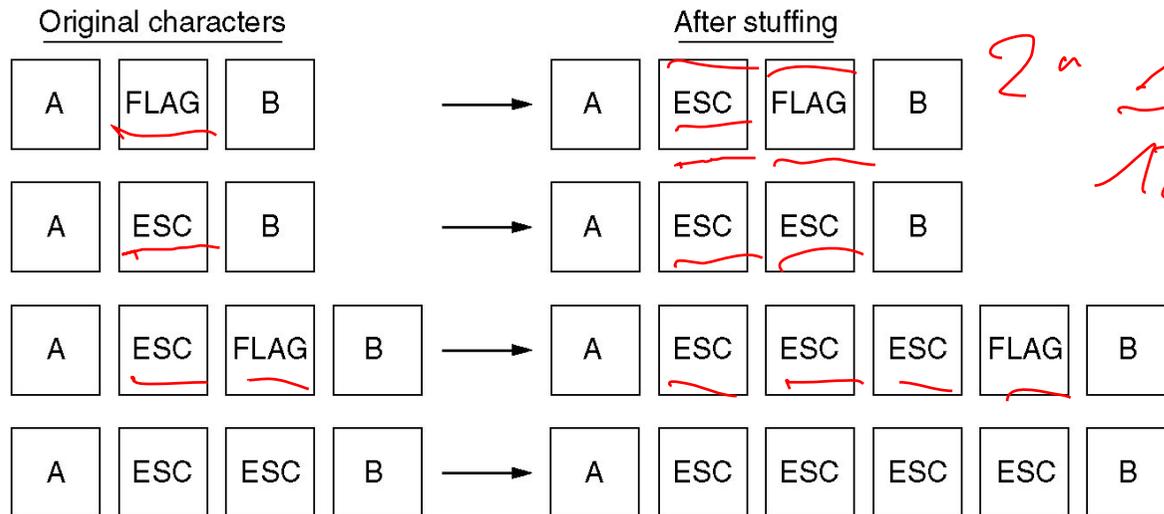


Flag Bytes und Bytestopfen

- Besondere “Flag Bytes” markieren Anfang und Ende eines Frames



- Falls diese Marker in den Nutzdaten vorkommen
 - Als Nutzdatenbyte mit Sonderzeichen (Escape) markieren
 - Bytestopfen (byte stuffing)
 - Falls Sonderzeichen und “Flag-Byte” erscheinen, dito,
 - etc., etc.



n Bytes
2 * n Bytes

$$2^n \frac{1}{128} + 2 \left(1 - \frac{1}{128} \right)$$

$$= 1 + \frac{1}{128}$$

- Bytestopfen verwendet das Byte als elementare Einheit
 - Das Verfahren funktioniert aber auch auf Bitebene
- Flag Bits und Bitstopfen (bit stuffing)
 - Statt flag byte wird eine Bit-Folge verwendet
 - z.B.: 01111110 *6 x 1er präfixfrei*
 - Bitstopfen
 - Wenn der Sender eine Folge von fünf 1er senden möchte, wird automatisch eine 0 in den Bitstrom eingefügt
 - Außer bei den Flag Bits
- Der Empfänger entfernt eine 0 nach fünf 1ern



Originale Nutzdate (a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Nach dem Bitstopfen (b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0

Stuffed bits



Diagram showing bit stuffing with arrows pointing to stuffed 0s.

Nach der "Entstopfung" (c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0



Diagram showing the original bit sequence after de-stuffing.

000000 0 0V 1 5V

- Möglicher Spielraum bei Bitübertragungsschicht bei der Kodierung von Bits auf Signale

- Nicht alle möglichen Kombination werden zur Kodierung verwendet

0 → 01 1 → 10

- Zum Beispiel: Manchester-Kodierung hat nur tief/hoch und hoch/tief-Übergang



- Durch "Verletzung" der Kodierungsregeln kann man Start und Ende des Rahmens signalisieren

- Beispiel: Manchester – Hinzunahme von hoch/hoch oder tief/tief

- Selbsttaktung von Manchester gefährdet?

Bits kodiert
0 01
1 10

- Einfache und robuste Methode

- z.B. verwendet in Ethernet

- Kosten? Effiziente Verwendung der Bandbreite?

1101 101010101010 000000



■ Aufgaben

- Erkennung von Fehlern (fehlerhafte Bits) in einem Frame
- Korrektur von Fehlern in einem Frame

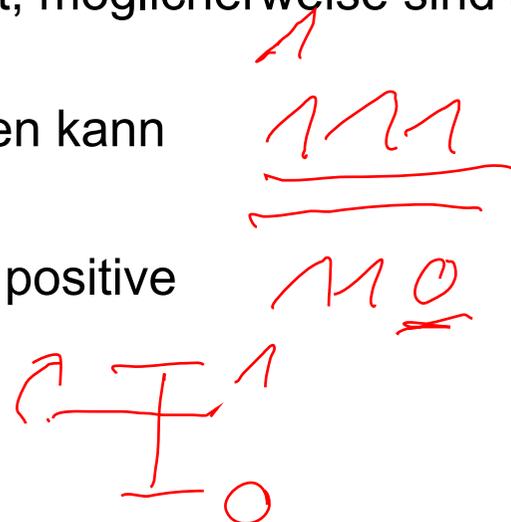
■ Jede Kombination dieser Aufgaben kommt vor

- Erkennung ohne Korrektur

- Löschen eines Frames ohne weiter Benachrichtigung (drop a frame)
- ~~Höhere~~ Schichten müssen sich um das Problem kümmern

- Korrektur ohne Erkennung

- Es werden bestmöglich Bitfehler beseitigt, möglicherweise sind aber noch Fehler vorhanden
- Sinnvoll, falls Anwendung Fehler tolerieren kann
 - Beispiel: Tonübertragung
- Prinzipiell gerechtfertigt, weil immer eine positive Restfehlerwahrscheinlichkeit bleibt



- Redundanz ist eine Voraussetzung für Fehlerkontrolle
- Ohne Redundanz
 - Ein Frame der Länge m kann 2^m mögliche Daten repräsentieren
 - Jede davon ist erlaubt

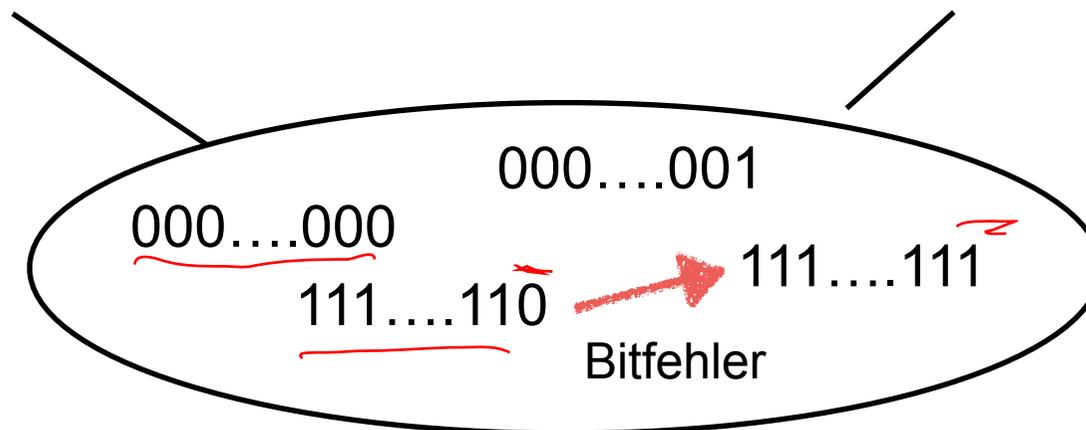
$m = 3$

000
001
010
011
100
101
111

- Ein fehlerhaftes Bit ergibt einen neuen Dateninhalt

Menge legaler Frames

Menge möglicher Frames



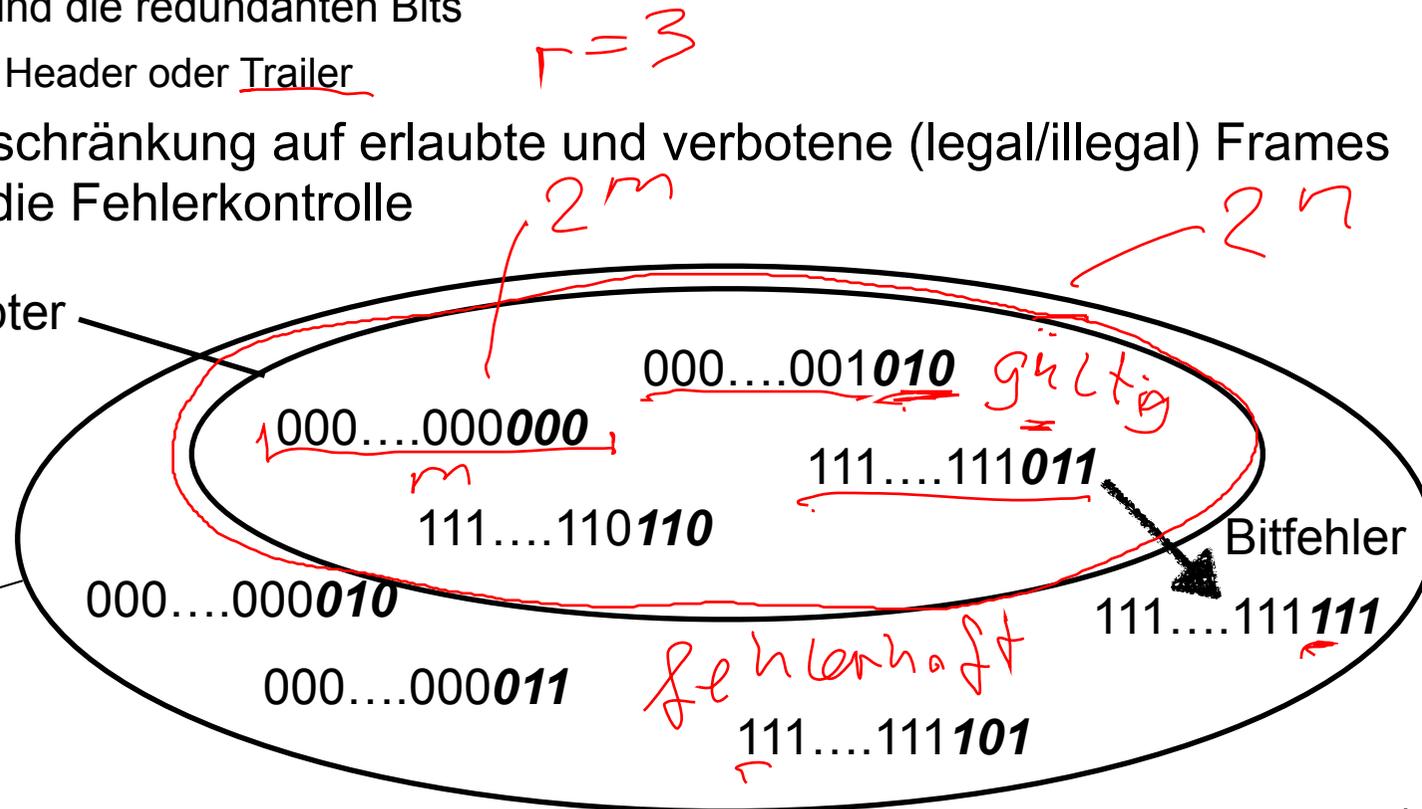
■ Kernidee:

- Einige der möglichen Nachrichten sind verboten
- Um dann 2^m legale Frames darzustellen
 - werden mehr als 2^m mögliche Frames benötigt
 - Also werden mehr als m Bits in einem Frame benötigt
- Der Frame hat also Länge $n > m$
- $r = n - m$ sind die redundanten Bits
 - z.B. Im Header oder Trailer

■ Nur die Einschränkung auf erlaubte und verbotene (legal/illegal) Frames ermöglicht die Fehlerkontrolle

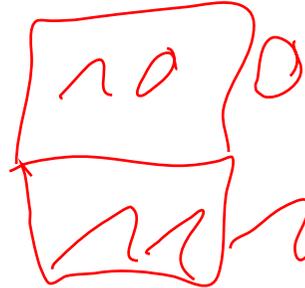
Menge erlaubter Frames

Menge aller Frames



Einfachste Redundanz: Das Paritätsbit

- Eine einfache Regel um ein redundantes Bit zu erzeugen
(d.h. $n=m+1$)



- Parität
 - Odd parity
 - Eine Eins wird hinzugefügt, so dass die Anzahl der 1er in der Nachricht ungerade wird (ansonsten eine Null)
 - Even parity
 - Eine Eins wird hinzugefügt, so dass die Anzahl der 1er in der Nachricht gerade wird (ansonsten eine Null)

101 110

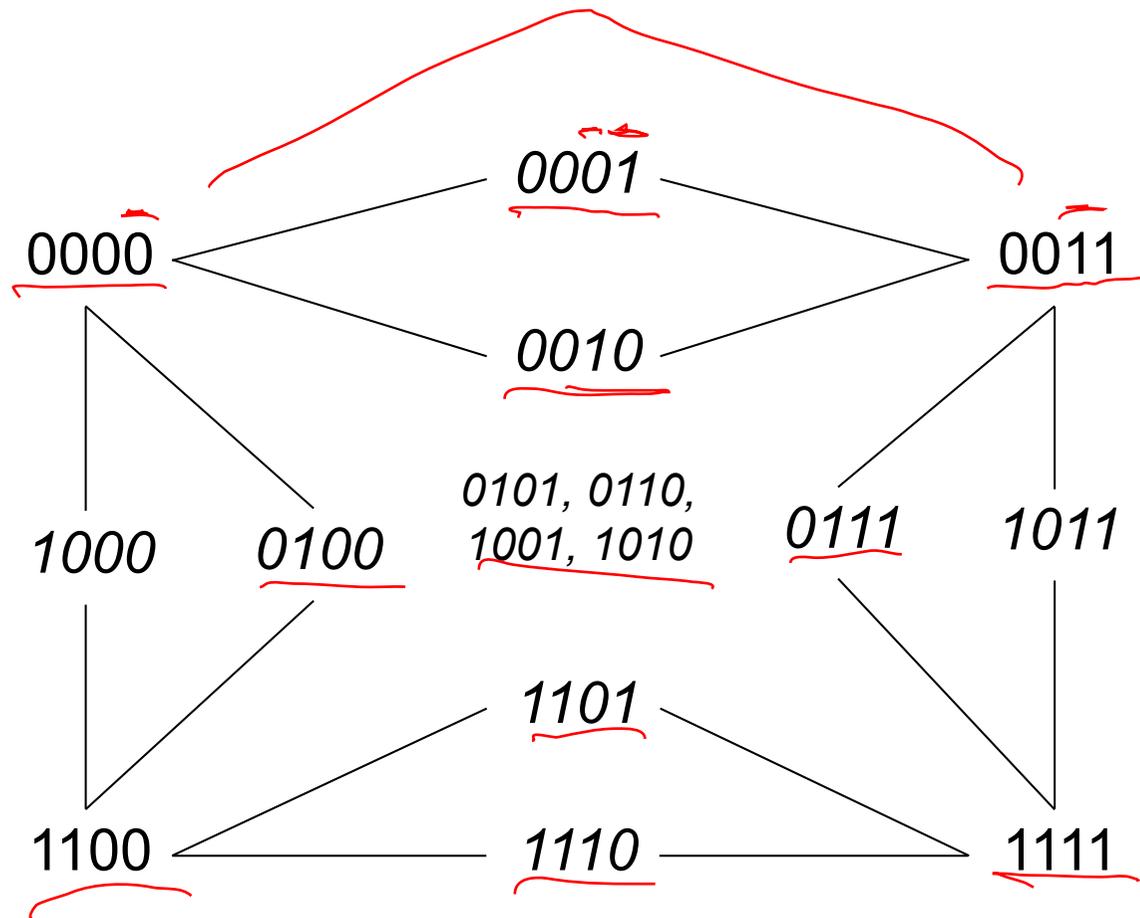
- Beispiele:

- empfangen für even parity: 111
- empfangen für even parity: 110

110, 011, 101
000

- Der Sender sendet nur erlaubte Frames
- In der Bitübertragungsschicht könnten Bits verfälscht werden
- Hoffnung:
 - Legale Frames werden nur in illegale Nachrichten verfälscht
 - Und niemals ein legaler Frame in einen anderen Legalen
- Notwendige Annahme
 - In der Bitübertragungsschicht werden nur eine bestimmte Anzahl von Bits verändert
 - z.B. k Bits pro Frame
 - Die legalen Nachrichten sind verschieden genug, um diese Frame-Fehlerrate zu erkennen

- Angenommen die folgenden Frames sind erlaubt: 0000, 0011, 1100, 1111



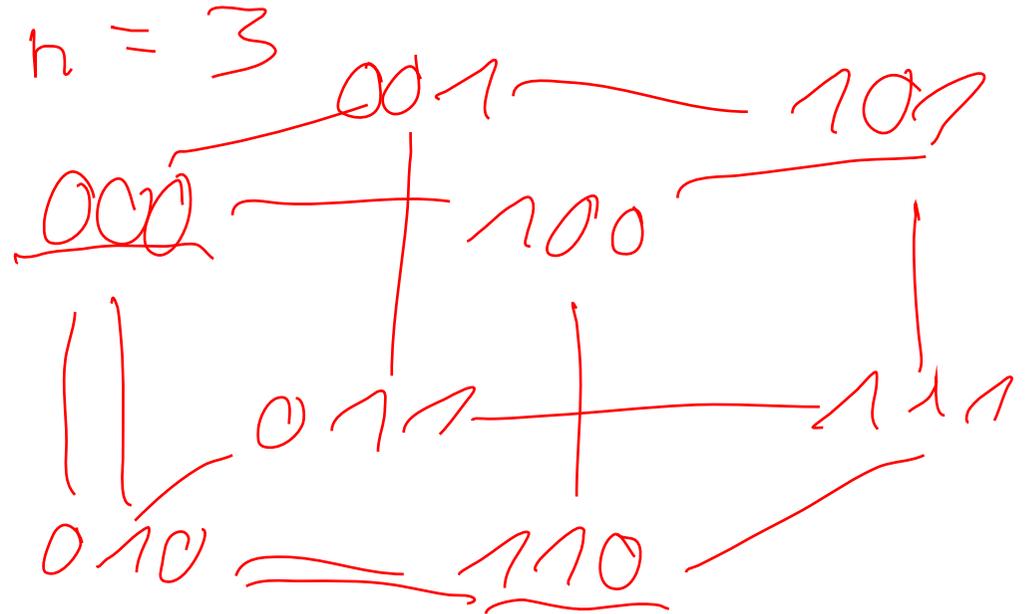
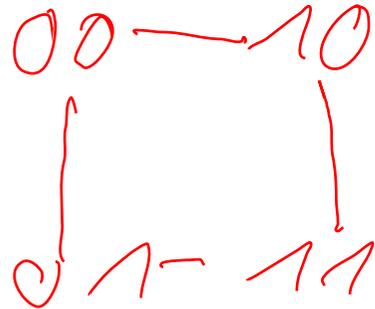
Kanten verbinden Frames, die sich nur in einem Bit unterscheiden

Ein einfacher Bitfehler kann erlaubte Frames nicht in einen anderen erlaubten Frame umformen!

uvxy – erlaubt

abcd – verboten

Struktur der Frames durch Bitfehler



Hyperwürfel

- Der “Abstand” der erlaubten Nachrichten zueinander war immer zwei Bits
- Definition: Hamming-Distanz
 - Seien $x = x_1, \dots, x_n$ und $y = y_1, \dots, y_n$ Nachrichten
 - Dann sei $d(x,y)$ = die Anzahl der 1er Bits in $x \text{ XOR } y$
- Intuitiver: die Anzahl der Positionen, in denen sich x und y unterscheiden

$$\begin{array}{l} x = 0000 \\ y = 0011 \end{array} \quad d(x,y) = 2$$

- Die Hamming-Distanz ist eine Metrik

- Symmetrie

- $d(x,y) = d(y,x)$

- Dreiecksungleichung:

- $d(x,y) \leq d(x,z) + d(z,y)$

- Identität

- $d(x,x) = 0$ und
 $d(x,y) = 0$ gdw. $x = y$

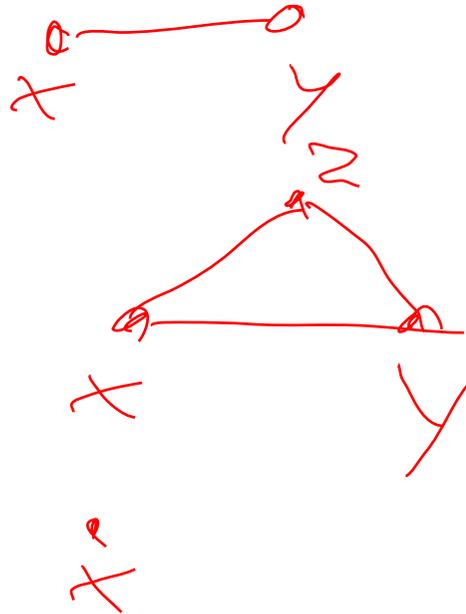
- Beispiel:

- x = 0011010111

- y = 0110100101

- x XOR y = 0101110010

- $d(x,y) = 5$



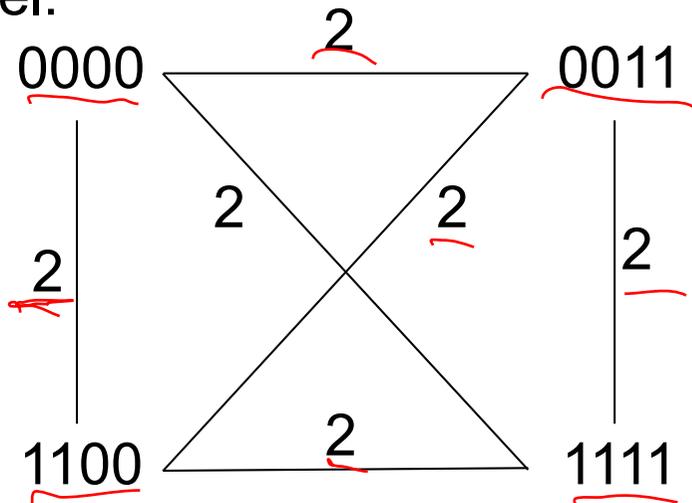
- Die Hamming-Distanz einer Menge von (gleich langen) Bit-Strings S ist:

$$S = \{0000, 0011, 1100, 1111\}$$

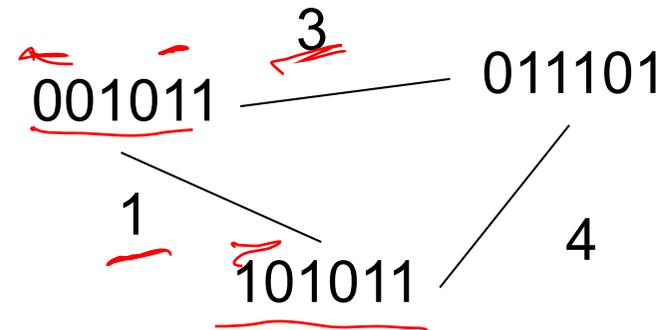
$$d(S) = \min_{x,y \in S, x \neq y} d(x, y)$$

- d.h. der kleinste Abstand zweier verschiedener Wörter in S

Beispiel:

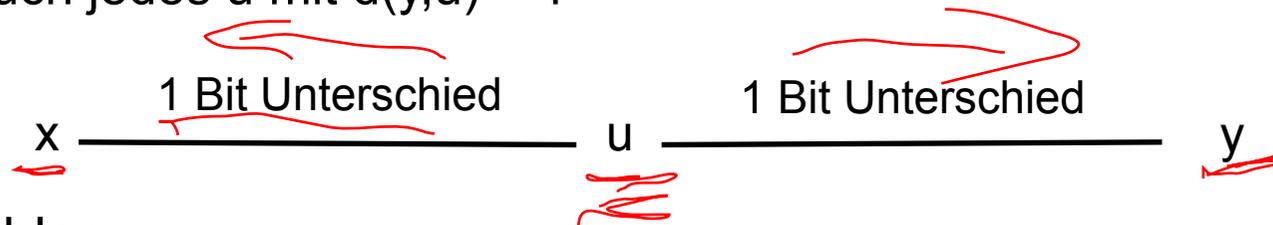


Alle Abstände sind 2



Ein Abstand ist 1!

- 1. Fall $d(S) = 1$
 - Keine Fehlerkorrektur
 - Legale Frames unterscheiden sich in nur einem Bit
- 2. Fall $d(S) = 2$
 - Dann gibt es nur $x, y \in S$ mit $d(x,y) = 2$
 - Somit ist jedes u mit $d(x,u) = 1$ illegal,
 - wie auch jedes u mit $d(y,u) = 1$



- 1-Bit-Fehler
 - können immer erkannt werden
 - aber nicht korrigiert werden
 - z.B. Parity Bit