

Willkommen bei Kommunikations- und Netztechnik!

-

*Von Kupferkabel, Glasfaser und Mikrowelle
über Telefon, Ethernet und TCP
zu E-Mail, Webserver und REST.*

-



-

Heute: **Verlässliche Übertragungen über fehlerbehafteten Kanal, Rechner über „Kabel“ verbunden.**

Übungsaufgaben

- In Gruppen bis 3 Personen
- Gruppenaufgaben einfach mit Name, da die Matrikelnummer nicht als öffentlich gilt

Wiederholung Bitübertragung I

- Nyquist Formel:

$$\text{maximum data rate} = 2B \log_2 V$$

- Shannon Formel:

$$\text{maximum number of bits/sec} = B \log_2(1 + S/N)$$

$$\text{SNR nach } \frac{S}{N}: \text{SNR} = 10 \log_{10}\left(\frac{S}{N}\right)$$

- duplex, simplex

■ Unterteilung Übertragungsmedien, Beispiele für Kategorien

- Kabelgebunden

- Kupfer

■ LWL

- Kabellos

- Modulation

Wiederholung Bitübertragung II

- Passband: FSK (Frequenz), ASK (Amplitude), PSK (Phase)
- Bandbreitenbedarf, Taktrückgewinnung, Gleichstromfreiheit
- Multiplexing
 - FDM (Frequenz), TDM (Zeit), CDMA (Code)
- CDMA
 - Walshcodes zur Encodierung mehrerer Signale in ein Signal

Weitere Fragen?

Ablauf heute

- Grundlagen
 - Dienste der Sicherungsschicht
 - Rahmen bilden
- Konkret
 - Fehlerkorrektur
 - Fehlererkennung
 - Grundlegende Protokolle
 - Schiebefensterprotokolle

Pause um etwa 14:15

Warum Sicherungsschicht?

- Bits zwischen verbundene Geräten übertragen
- Fehlerrate **genug reduzieren** für die Vermittlungsschicht
- Fehler frühzeitig abfangen: Optimierung für geringere Latenz

Ziele heute I

- Sie wissen, dass die Sicherungsschicht als Dienst Pakete aus Bits überträgt
- Sie wissen, dass die Sicherungsschicht als Protokoll Pakete in Rahmen verpackt und übermittelt.
- Sie verstehen, dass je nach Medium unterschiedliche Komplexität sinnvoll ist und können die dabei notwendigen Abwägungen erklären.
- Sie können mit einer Kurzbeschreibung aus (11,7) Hamming-Codierten Daten die korrigierten Nachrichtenbits extrahieren.
- Sie wissen, dass der Hamming-Abstand angibt, ab wievielen Bitfehlern Fehler unentdeckt bleiben können.
- Sie verstehen ein 1-Bit Schiebefensterprotokoll und erkennen verschiedene Optimierungen

Ziele heute II

- Sie haben einen Fehlerkorrekturcode programmiert (und wissen, dass sie es wieder tun könnten).

Versuch 1 (von 3): Auswirkungen von Fehlern

Erste Reihe

Sie haben ein Blatt mit 1-en und 0-en. Nehmen Sie jeweils die entsprechenden kleinen Zettel (Karten), drehen Sie sich nach hinten, schließen Sie die Augen und werfen Sie die Zettel einen nach dem anderen auf den Tisch der Person hinter Ihnen.

Mittlere Reihen

Nehmen Sie die Zettel. Wenn einer runterfällt, ersetzen Sie ihn durch einen zufälligen. Dann drehen Sie sich nach hinten, schließen Sie die Augen und werfen Sie die Zettel.

Letzte Reihe

Nehmen Sie die Zettel und decodieren Sie die Bitfolge mit der ASCII-Tabelle auf der nächsten Seite.

Etwas kalibrieren: 1-2 Fehler von erster zu letzter Reihe. Die Anzahl Fehler einer Satelliten-Übertragung in einem 100kiB Bild.

Corona-Pandemie-Zeit

Hochhalten und zeigen statt werfen!

ASCII-Tabelle

	000	001	010	011	100	101	110	111
0000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL
0001	BS	HT	LF	VT	FF	CR	SO	SI
0010	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB
0011	CAN	EM	SUB	ESC	FS	GS	RS	US
0100	SP	!	"	#	\$	%	&	'
0101	()	*	+	,	-	.	/
0110	0	1	2	3	4	5	6	7
0111	8	9	:	;	<	=	>	?
1000	@	A	B	C	D	E	F	G
1001	H	I	J	K	L	M	N	O
1010	P	Q	R	S	T	U	V	W
1011	X	Y	Z	[\]	^	_
1100	`	a	b	c	d	e	f	g
1101	h	i	j	k	l	m	n	o
1110	p	q	r	s	t	u	v	w
1111	x	y	z	{		}	~	DEL

ASCII-Steuerzeichen (zum Nachschlagen)

NUL	Null	DLE	Data Link Escape
SOH	Start of Heading	DC1	Device Control 1 (XON)
STX	Start of Text	DC2	Device Control 2
ETX	End of Text	DC3	Device Control 3 (XOFF)
EOT	End Of Transmission (EOF)	DC4	Device Control 4
ENQ	Enquiry (who are you?)	NAK	Negative ACK
ACK	Acknowledgement	SYN	Synchronous Idle
BEL	Bell: echo -en "\a";	ETB	End of Transmission Block
BS	Backspace	CAN	Cancel
HT	Horizontal Tab	EM	End of Medium
LF	Linefeed echo -en "\n";	SUB	Substitute
VT	Vertical Tab	ESC	Escape
FF	Form Feed	FS	File Separator
CR	Carriage Return	GS	Group Separator
SO	Shift Out	RS	Record Separator
SI	Shift In	US	Unit Separator

Dienste der Sicherungsschicht

Dienst-Arten

- Welche Arten von Diensten sind sinnvoll?
- Woran erinnern sie sich noch aus der Übersicht?

Je 6 Leute zusammen, 5 Minuten, dann sammeln wir.

Beispiel: Verbindungsorientiert, bytestrom, unbestätigt.

Stärken und Schwächen

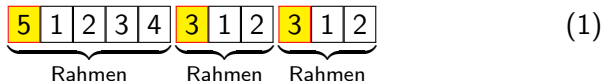
- Welche Charakteristiken haben diese Arten von Diensten?

Rahmen bilden

- Längenbyte
- Flagbyte
- Flagbits
- Codierungsverletzung
- Kombiniert

Längenbyte

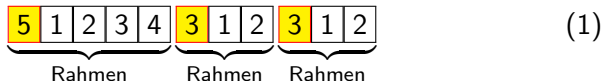
-



Brauchen canonical S-expressions einen fehlerfreien Kanal?
(4:this22:Canonical S-expression3:has1:55:atoms)

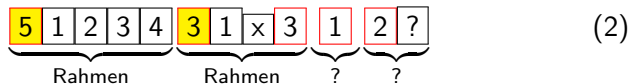
Längenbyte

-

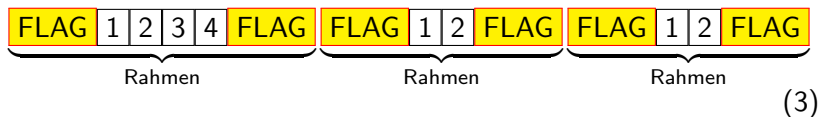


Brauchen **canonical S-expressions** einen fehlerfreien Kanal?
 (4:this22:Canonical S-expression3:has1:55:atoms)

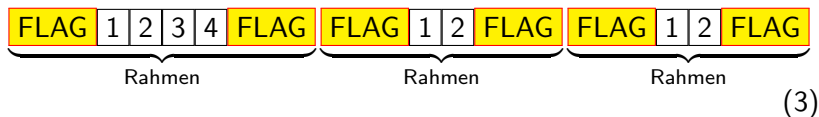
Byteverlust und Desynchronisierung



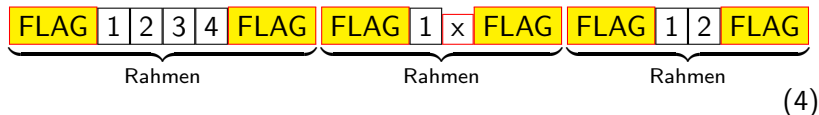
Flagbyte mit Bytestopfen



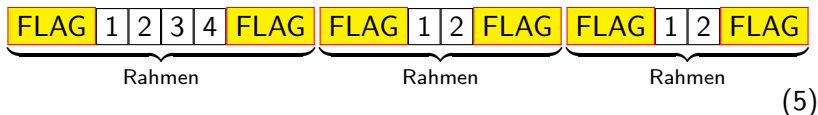
Flagbyte mit Bytestopfen



Byteverlust



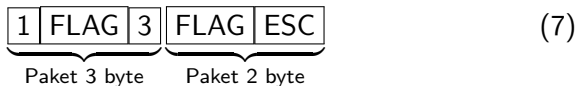
Flagbyte mit Bytestopfen



Flagbyteverlust



Flagbytes: Escaping nötig



Wie in Java-Strings:

```
String withQuote = "ab\"c";
```

Flagbits mit Bitstopfen

FLAG 01111110

Escape x011111x -> x0111110x

Unescape x0111110x -> x011111x



Codierungsverletzung (Abstraktionsbruch)

Codierung auf Bitübertragungsebene verwendet nur ein Subset der Zeichen, z.B. für Taktrückgewinnung.

Beispiel: 4B/5B

Signalisierung über verbotene Zeichen; unproblematisch, da selten.

Codierungsverletzung (Abstraktionsbruch)

Codierung auf Bitübertragungsebene verwendet nur ein Subset der Zeichen, z.B. für Taktrückgewinnung.

Beispiel: 4B/5B

Signalisierung über verbotene Zeichen; unproblematisch, da selten.

Aber Abstraktionsbruch:

Wechsel der Codierung erfordert Wechsel der Rahmenbildung.

Kombiniert

- Beispiel: Präambel + Längenfeld in WiFi (72 Bit Präambel!)

Frage: Wofür ist die Länge noch nützlich?

Zusammenfassung der abstrakten Grundlagen

Dienste

- Verbindungslos, unbestätigt
- Verbindungslos, bestätigt
- Verbindung, bestätigt, geordnet

*Bestätigung als Optimierung:
Früher korrigieren.*

Rahmen

- Längenfeld
- Flagbyte/-bit
- Codierungsverletzung

Fehlerkorrektur oder -erkennung

- Arten von Fehlern
- Erkennung vs. Korrektur
- Hamming-Abstand
- Fehlerkorrektur
- Fehlererkennung

Arten von Fehlern

Welche Arten von Fehlern gibt es? Welche Struktur haben die Veränderungen?

Arten von Fehlern

Welche Arten von Fehlern gibt es? Welche Struktur haben die Veränderungen?

-

- Verlorene Einzelbits
- Einzelne Bitflips
- Burst-Fehler (mehrere)

Erkennung vs. Korrektur

Effizienzfrage:

- Immer Zusätzliche Bandbreite
- Teure Neu-Übertragungen (Round-Trips) im Fehlerfall

Hamming-Abstand

Codewörter

- 0000000000
- 0000011111
- 1111100000
- 1111111111

$p=95\%$ korrekte Bits:

- HA 1: 40% beschädigt (1 / 3)
- HA 2: 8.6% beschädigt (1 / 12)
- HA 3: 1.2% beschädigt (1 / 87)
- Ha 5: 0.0096% beschädigt (1 / 10k)

Summe $\text{binom}(10, p, 11\text{-HA})$

→ <https://de.wikipedia.org/wiki/Binomialverteilung>

Hamming-Abstand

Codewörter

- 0000000000
- 0000011111
- 1111100000
- 1111111111

p=95% korrekte Bits:

- HA 1: 40% beschädigt (1 / 3)
- HA 2: 8.6% beschädigt (1 / 12)
- HA 3: 1.2% beschädigt (1 / 87)
- Ha 5: 0.0096% beschädigt (1 / 10k)

Summe binom(10, p, 11-HA)

→ <https://de.wikipedia.org/wiki/Binomialverteilung>

Korrigierbar:

- 0000000000
- 0000000011
- 0000011111

Erkennbar:

- 0000000000
- 0000001111
- 0000011111

Hamming-Abstand:

- 0000000000
- 0000011111
- 0000011111

Fehlerkorrektur, Methoden

- Vollständige Suche
- Hamming-Code
- Binäre Faltungscodes
- Reed-Solomon-Codes
- LDPC-Codes (Low-Density Parity Check)

Vollständige Suche

- Komplexität: $O(N)$ (alle möglichen Codewörter vergleichen)

⇒ zu teuer.

⇒ Hinweise, wo der Fehler korrigiert werden kann.

Prüfbits Minimum, Einzelbitfehler

n gesamtbits, **m** Nachrichtenbits und **r** Prüfbits. $n = m + r$

m bits $\rightarrow 2^m$ mögliche Nachrichten \rightarrow Pro Nachricht zusätzlich n verbotene Codewörter mit Abstand 1 nötig.

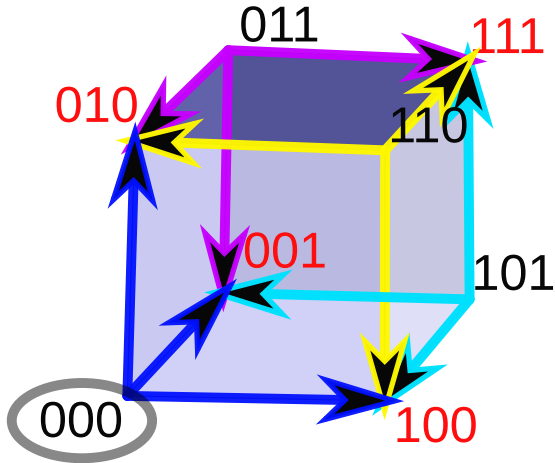
Verbotene Codewörter: Für jede erlaubte Nachricht jedes Bit flippen ($\Rightarrow n$ flips). $\Rightarrow n + 1$ Bitmuster für jede erlaubte Nachricht.

$$(n + 1) \cdot 2^m \leq 2^n = 2^{m+r} \quad (10)$$

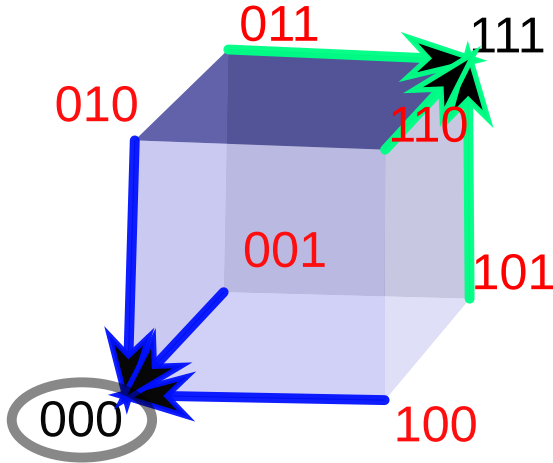
$$(m + r + 1) \leq 2^r \quad (11)$$

$$m \leq 2^r - r - 1 \quad (12)$$

Verbotene Wörter, 1 bit erkennen



Verbotene Wörter, 1 bit korrigieren



Hamming-Code

Prüfbits gemappt auf Bits

Reine Daten:

1	0	0	0	0	0	1
---	---	---	---	---	---	---

 (14)

Mit Prüfbits:

0	0	1	0	0	0	0	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---

 (15)

1 Prüfbit in Position 4, genutzt für Bit 5, 6, 7.

0	0	1	0	0	0	0	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---

 (16)

0	0	1	0	1	0	0	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---

 (17)

(11,7) Hamming: Hilfsfunktionen

```

define : mod2sum . numbers
  . "Modulo-2 sum, i.e. for even parity"
  ## : tests : test-equal 1 : mod2sum 1 0 1 1 0
  modulo (apply + numbers) 2

define H mod2sum ;; for brevity

define : flip numbers index
  . "flip the bit-number (0→1 or 1→0) at the index."
  ## : tests : test-equal '(1 0 1) : flip '(0 0 1) 0
  append
    take numbers index
    list : mod2sum 1 : list-ref numbers index
    drop numbers [index + 1]

```

Hacks, um auf Zahlen zu arbeiten. Sauberer: Bitvector.

(11,7) Hamming: Encode

Body

Header

```
define : 11/7-encode numbers
```

```
##
```

```
tests
```

```
test-equal
```

```
. '(0 0 1 0 0 0 0 1 0 0 1)
```

```
11/7-encode
```

```
. '(1 0 0 0 0 0 1)
```

```
match numbers
```

```
: i3 i5 i6 i7 i9 i10 i11
```

```
list
```

```
H i3 i5 i7 i9 i11 ;; bit 1
```

```
H i3 i6 i7 i10 i11 ;; bit 2
```

```
. i3 ;; bit 3
```

```
H i5 i6 i7 ;; bit 4
```

```
. i5 i6 i7 ;; bit 5, 6, 7
```

```
H i9 i10 i11 ;; bit 8
```

```
. i9 i10 i11 ;; bit 9, 10, 11
```

(11,7) Hamming: Decode

```
define : 11/7-decode numbers
  define broken-bit
    match numbers
      : h1 h2 i3 h4 i5 i6 i7 h8 i9 i10 i11
        +
        * 1 : H h1 i3 i5 i7 i9 i11
        * 2 : H h2 i3 i6 i7 i10 i11
        * 4 : H h4 i5 i6 i7
        * 8 : H h8 i9 i10 i11
  define fixed
    if : zero? broken-bit
      . numbers
      flip numbers {broken-bit - 1}
  match fixed
    : h1 h2 i3 h4 i5 i6 i7 h8 i9 i10 i11
    list i3 i5 i6 i7 i9 i10 i11
```

Zum Nachhören

Eine schöne Beschreibung von Hamming-Codes mit einem 15/11 Beispiel finden Sie in den 3Blue1Brown Videos

- Hamming codes, how to overcome noise (<https://www.youtube.com/watch?v=X8jsijh1lIA>) und
- Hamming codes part 2, the elegance of it all (https://www.youtube.com/watch?v=b3NxrZOu_CE).

Weitere Codes

- Binäre Faltungscodes: Ursprünglich für die Voyager-Missionen der NASA, heute in WLAN. Erzeugt aus jedem Bit und internem Zustand zwei Bits, kann Unsicherheiten einbeziehen.
- Reed-Solomon: Polynome mit zusätzlichen Stützpunkten.
- LDPC-Codes (Low-Density Parity Check): Dünn besetzte Matrizen, iterativ

Fehlererkennung, Methoden

- Parität
- Prüfsummen
- Cyclic Redundancy Check (CRC)

Parität

Zusatzbits: Anzahl der 1-Bits gerade (oder ungerade, je nach Methode).

00101011(gerade)

10101010(gerade)

Versuch 2 (von 3): Fehlererkennung mit Hamming

Erste Reihe

Sie haben ein Blatt mit 1-en und 0-en. Berechnen Sie die Hamming-Encodierte Bitfolge. Nehmen Sie jeweils die entsprechenden kleinen Zettel (Karten), drehen Sie sich nach hinten, schließen Sie die Augen und werfen Sie die Zettel einen nach dem anderen auf den Tisch der Person hinter Ihnen.

Mittlere Reihen

Nehmen Sie die Zettel. Wenn einer runterfällt, ersetzen Sie ihn durch einen zufälligen. Hamming-Dekodieren Sie die Bitsequenz, mit Korrektur, falls nötig. Hamming-Enkodieren Sie sie erneut. Dann werfen Sie.

Letzte Reihe

Nehmen Sie die Zettel, Hamming-Dekodieren Sie die Bitfolge und berechnen Sie das entsprechende Zeichen mit der ASCII-Tabelle.

Corona-Pandemie-Zeit

Hochhalten und zeigen statt werfen!

Parity mit Versatz

Gegen Burstfehler: Versatz (Interleaving) \Rightarrow Parität
Byteübergreifend.

Original

1	0	0	1	1	1	0
1	1	0	0	1	0	1
1	1	1	0	1	0	0
1	1	1	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	0	1	0
1	1	0	1	0	1	1
1	0	1	1	1	1	0

Mit Burstfehlern

1	0	0	1	1	1	0
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	0	1	0
1	1	0	1	0	1	1
1	0	0	0	0	0	0

Burst: auch kurze Störung paralleler Übertragung.

Prüfsummen

Verallgemeinerte Parität, Internetprüfsumme in IP im
 Einerkomplement: Überlauf wird auf niedrigstes Bit addiert $\Rightarrow 100$
 $+ 100 = 001$

Cyclic Redundancy Check (CRC)

Die Bitfolge geteilt durch Generatorpolynom (das CRC-Polynom) mod(2) (Polynomdivision); behält Rest. Rest = CRC-Wert → anhängen.

Bsp: 110101 → $1x^5 + 1x^4 + 0x^3 + 1x^2 + 0x^1 + 1$

Verifizieren: Daten + CRC durch CRC-Polynom teilen. Rest → Fehler!

Fehlerkorrektur allgemeiner: Fountain codes

- Fountain Codes:
https://en.wikipedia.org/wiki/Fountain_code
- Patente auf Raptor-Codes sind freigegeben.

*Wiederherstellung von Daten mit ausreichendem Subset.
A fountain code is optimal if the original k source symbols
can be recovered from any k encoding symbols*

In [Freenet](#) / [Hyphanet](#) und [gnunet](#) wird FEC (Forward Error Correction) genutzt, bei der beliebige 50% der Daten gebraucht werden.

Fehlerkorrektur allgemeiner: PAR2

- PAR2: <http://parchival.sourceforge.net/>

Easily verify and regenerate single missing parts out of a set of data-blocks

Aus den Usenet-Zeiten. Heute meist nicht mehr nötig, weil Fehler schon bei Übertragung abgefangen werden.

*Könnte ähnliches bei Latenz-Jitter helfen? — bei den **langsamsten Antworten** oder **p90**, die alles blockieren?*

Zusammenfassung

- Hamming Abstand: Ab wie vielen Fehlern nicht mehr erkannt
- Erkennbare Bits: Abstand - 1
- Korrigierbare Bits: 0.5 * Erkennung
- $m \leq 2^r - r - 1$

Grundlagen der Protokolle

- Trennung von höheren Schichten: Sicherungsschicht läuft teils auf der Netzwerkkarte
- Rahmen liefert Zusatzinformationen (z.B. Prüfsumme)

Simplexprotokoll

```
void Sender() {
    Frame toSend;
    Packet buffer;
    while (true) {
        buffer = from_network_layer();
        toSend.info = buffer;
        to_physical_layer(toSend);
    }
}

void Receiver() {
    Frame received;
    while (true) {
        wait_for_event();
        received = from_physical_layer();
        to_network_layer(received.info);
    }
}
```

Stop and Wait

```
void Sender() {  
    Frame toSend;  
    Packet buffer;  
    while (true) {  
        buffer = from_network_layer();  
        toSend.info = buffer;  
        to_physical_layer(toSend);  
        wait_for_event();  
    }  
}  
  
void Receiver() {  
    Frame received, wakeSender;  
    while (true) {  
        wait_for_event();  
        received = from_physical_layer();  
        to_network_layer(received.info);  
        to_physical_layer(wakeSender.info);  
    }  
}
```

1 bit Sliding Window: implementation

```
define : 1-bit-sliding-window
  let loop : : event : wait-for-event
    when : is-frame-arrived? event
      let*
        : received : from-physical-layer
        seq : frame-seq received
        ack : frame-ack received
      when : = seq : flip-bit : frame-ack to-send
        to-network-layer : frame-packet received
        frame-ack-flip! to-send
        format #t "received: ~a, ack it: ~a\n" received to-send
      when : = ack : frame-seq to-send
        stop-timer ack
        frame-packet-set! to-send : from-network-layer
        frame-seq-flip! to-send
        format #t "~a acked ours, send next: ~a\n" received to-send
      to-physical-layer to-send
      start-timer : frame-seq to-send
      loop : wait-for-event
```

1 bit Sliding Window: data

```

import : srfi :9 records
define-record-type <frame>
  make-frame seq ack packet
  . frame?
  seq frame-seq frame-seq-set!
  ack frame-ack frame-ack-set!
  packet frame-packet frame-packet-set!

define : flip-bit bit
  if : zero? bit
    . 1 0

define-syntax-rule : flip-bit! rec getter setter
  setter rec : flip-bit : getter rec

define : frame-ack-flip! frame
  flip-bit! frame frame-ack frame-ack-set!
define : frame-seq-flip! frame
  flip-bit! frame frame-seq frame-seq-set!

```

1 bit Sliding Window: tooling

```
define : from-physical-layer ;; fake random frame
  make-frame (random 2) (random 2) #f

define to-send : make-frame 0 1 #f
;; stubs
define (from-network-layer) #f ;; fake packet (data)
define (to-network-layer packet) #f
define (to-physical-layer frame) #f
define (start-timer bit) #f
define (stop-timer bit) #f
define (wait-for-event) #f
define (is-frame-arrived? event) #t
```

Versuch 3 (von 3): Übertragung mit 1 bit window

Erste Reihe

Sie haben ein Blatt mit 1-en und 0-en. Nehmen Sie die passenden Zettel, drehen Sie sich nach hinten, schließen Sie die Augen und werfen Sie **einen Zettel**. Wenn Sie eine Bestätigung erhalten, werfen Sie den nächsten. Wenn Sie nach 5 Sekunden keine Bestätigung erhalten, schreiben Sie den Zettel neu.

Mittlere Reihen

Nehmen Sie den Zettel und bestätigen Sie mit OK. Wenn einer runterfällt, bestätigen sie **nicht**. Nach Empfang aller bits, drehen Sie sich nach hinten, schließen Sie die Augen und werfen Sie die wie die erste Reihe.

Letzte

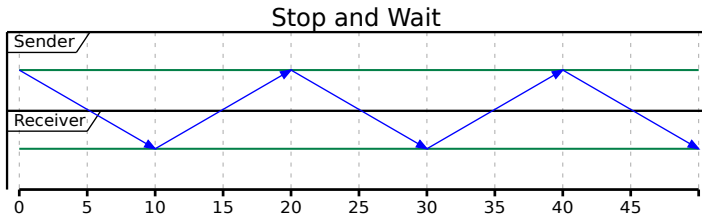
Reihe

Bestätigen Sie wie die zweite Reihe. Nach Abschluss nehmen Sie die Zettel und decodieren Sie die Bitfolge mit der ASCII-Tabelle.

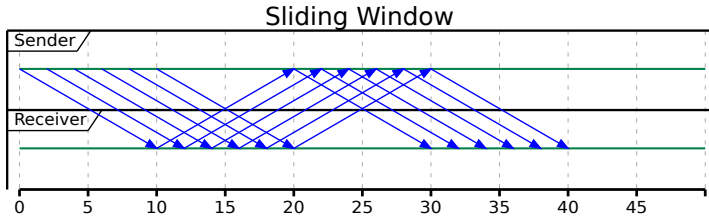
Corona-Pandemie-Zeit

Hochhalten und zeigen statt werfen!

Stop and wait



Optimierung



Zusammenfassung

- Bestätigungen im Rahmen (ack), mit Sequenznummern
- Sliding Window beidseitig

Zusammenfassung

- Dienst der Sicherungsschicht: Pakete aus Bits übertragen
- Protokoll der Sicherungsschicht: Pakete in Rahmen verpacken und übermitteln.
- Aufgaben der Sicherungsschicht: Rahmenbildung, Fehlerkorrektur und -erkennung, Flusskontrolle
- Fehlerkorrektur erhöht Latenz. Der richtige Grad an Fehlerkorrektur minimiert die durchschnittliche Gesamtzeit bis zur Korrekten Übertragung.
- Hamming-Codes haben Korrekturbits auf Zweierpotenzen.
- Hamming-Abstand: Abstand in Bitflips zwischen den zwei ähnlichsten Codewörtern. Ab dieser Zahl Bitfehler können Fehler unentdeckt bleiben.
- 1-Bit Schiebefensterprotokoll: Übertrage nur nach Freigabe via Bestätigung der Rahmennummer.

Fragen für die Prüfung?

Ideensammlung:

- Hamming-Code anwenden
- Ein Dienst und ein Protokoll nennen (+ erläutern)
- Gängige Fehler(-typen) nennen.
- Vor- und Nachteil der Fehlerkorrektur
- Korrektur vs. Erkennung

Selbststudium diese Woche I

- Geben Sie Ihre Namen in den folgenden Sprachpaargenerator ein, um ein **Sprachpaar** zu erhalten:
<https://www.draketo.de/software/vorlesung-netztechnik#nummer-zu-sprache> (läuft clientseitig in Ihrem Browser)
- Schreiben Sie ein Programm, das die folgende Bitfolge mit dem 11,7 Hamming-Code aus der Vorlesung **encodiert**. Verwenden Sie dafür **Sprache A** aus dem Sprachpaar.
- Schreiben Sie ein Programm, das aus der folgenden 11,7 Hamming-codierten Bitfolge die Nachricht **extrahiert**. Verwenden Sie dafür die **Sprache B** aus dem Sprachpaar.
- Sind Sie bereits mit beiden Sprachen vertraut (oder wollen sie aus anderen Gründen wechseln), hängen Sie ein X an Ihre Namen an. Schreiben Sie das bitte bei der Abgabe.

Selbststudium diese Woche II

- Zeit: 4 Stunden.
- Bis zu drei Leute pro Gruppe.
- 7 Bit zum codieren: 0110001
- 11 Bit zum extrahieren: 00011110000

Einstieg
oooooooooooo

Dienste
o

Rahmen
oooooooooooo

Fehlerkorrektur
oooooooooooooooooooooooooooooooooooo

Protokolle
oooooooooooo

Zusammenfassung
ooo●

Viel Erfolg bei den Übungen!



Binomialverteilung für die Effizienz der Fehlerkorrektur I

Bei 10 Bits mit 95% Wahrscheinlichkeit individuell korrekt übertragen zu werden ist die Wahrscheinlichkeit, dass alle korrekt übertragen werden:

$$0.95^{10} \approx 0.60 = 60\%$$

Die Wahrscheinlichkeit, dass 9 von 10 korrekt übertragen werden, wird durch die Binomialverteilung angegeben.

<https://de.wikipedia.org/wiki/Binomialverteilung>

$$B(k|p, n) = \binom{n}{k} p^k \cdot (1 - p)^{n-k}$$

Gibt es bei 10 bits genau 9 korrekt übertragene?

$$\binom{10}{9} 0.95^9 \cdot 0.05^{10-9}$$

Binomialverteilung für die Effizienz der Fehlerkorrektur II

$$\binom{10}{9} = \frac{10!}{9! \cdot (10-9)!} = 10$$

$$0.95^{**}(9) * 0.05^{**}(10 - 9) = 0.03151247048623045$$

31.5%, dass es genau 9 korrekt übertragene gibt. Dazu 60% Wahrscheinlichkeit, dass es genau 10 korrekt übertragene gibt.

$60\% + 31.5\% = 91.5\%$, dass die Übertragung erfolgreich ist oder ein Fehler erkannt wird.

Das ist für Hammingabstand 2.

Bei Hammingabstand 5 ist hier die Wahrscheinlichkeit, dass ein Fehler unerkannt durchgeht nur 1 zu 15 701 (im Vergleich zu 1 zu 3 ohne Fehlererkennung).

Binomialverteilung für die Effizienz der Fehlerkorrektur III

Wir haben also als Kosten 250-Fache Erhöhung der Bandbreite pro Übertragung, dafür aber eine um Faktor 5000 reduzierte Wahrscheinlichkeit, dass der Rest des Netzes ein kaputtes Paket weiterträgt.

Verweise I

Bilder: