CoNe Freiburg

IIF INSTITUT FÜR INFORMATIK FREIBURG

Systeme II

7. Die Datensicherungsschicht (Teil 3)

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

Folien: Christian Schindelhauer°

Technische Fakultät

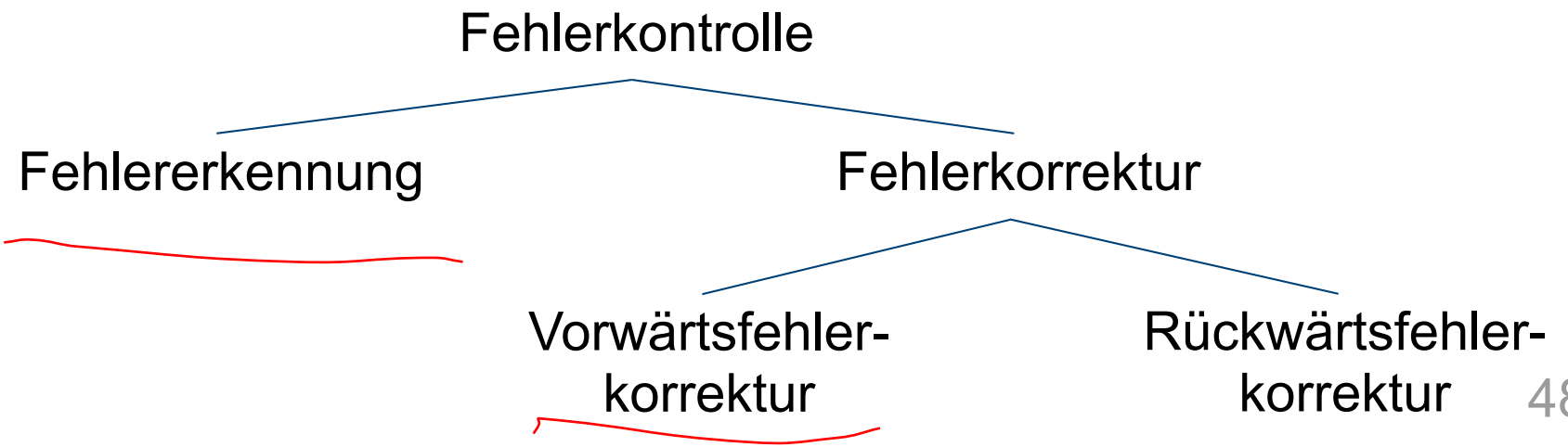
°: Rechnernetze und Telematik, *: Eingebettete Systeme

UNI FREIBURG

Fehlerkontrolle

CoNe Freiburg

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



UNI FREIBURG

CoNe Freiburg

Fehlererkennung: CRC

- effiziente Fehlererkennung mit zyklischer Redundanzprüfung (Cyclic Redundancy Check, CRC)
- praktisch häufig verwendeter Code
  - hoher Fehlererkennungsrate
  - effizient in Hardware umsetzbar
- Erstellung eines Prüfwerts (CRC-Wert) für jeden Datenblock
  - neben Fehlererkennung auch Korrektur möglich
- beruht auf Polynomarithmetik im Restklassenring Z_2
  - Zeichenketten sind Polynome
  - Bits sind Koeffizienten des Polynoms
- Beispiel:

Zeichenkette: 1 0 1 0

Polynom: $1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 0$

UNI FREIBURG

Rechnen in Z_2

- Rechnen modulo 2:
- Regeln:
  - Addition modulo 2 = Xor = Subtraktion modulo 2
  - Multiplikation modulo 2 = And

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	0

$\Leftrightarrow$

A	B	A - B
0	0	0
0	1	1
1	0	1
1	1	0

A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1

- Beispiel: $0 + (1 \cdot 0) + 1 + (1 \cdot 1) =$

Polynomarithmetik modulo 2

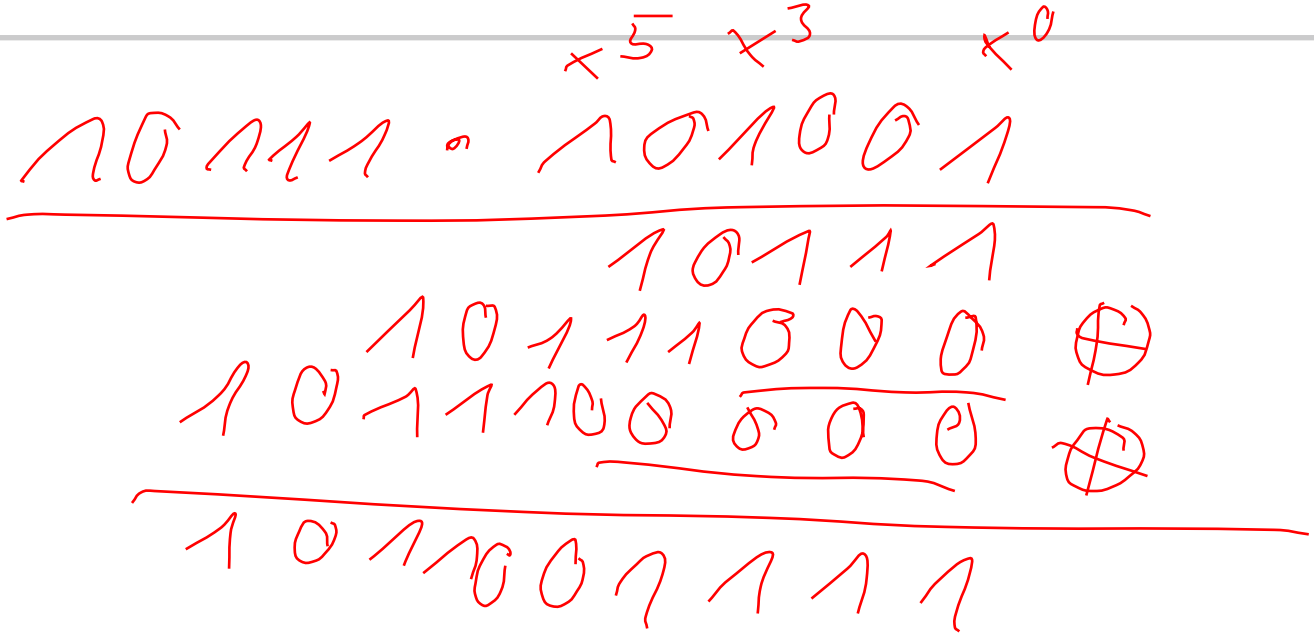
- Betrachte Polynome über den Restklassenring Z_2
  - $p(x) = a_n x^n + \ldots + a_1 x^1 + a_0$
  - Koeffizienten a_i und Variable x sind aus $\in \{0,1\}$
  - Berechnung erfolgt modulo 2
- Addition, Subtraktion, Multiplikation, Division von Polynomen wie gehabt

Zeichenketten und Polynomarithmetik

- Idee:
  - Betrachte Bitstring der Länge n als Variablen eines Polynoms
- Bit string: $b_n b_{n-1} \ldots b_1 b_0$

  Polynom: $b_n x^n + \ldots + b_1 x^1 + b_0$
  - Bitstring mit (n+1) Bits entspricht Polynom des Grads n
- Beispiel
  - A xor B = A(x) + B(x)
  - Wenn man A um k Stellen nach links verschiebt, entspricht das
    - B(x) = A(x) x^k
- Mit diesem Isomorphismus kann man Bitstrings dividieren

Polynomarithmetik: Multiplikation von Polynomen



Polynome zur Erzeugung von Redundanz: CRC

- Definiere ein Generatorpolynom $G(x)$ mit Grad g
  - Dem Empfänger und Sender bekannt
  - Wir erzeugen g redundante Bits
- Gegeben:
  - Frame (Nachricht) M, als Polynom $M(x)$
- Sender
  - berechne den Rest der Division $r(x) = x^g M(x) \bmod G(x)$
  - übertrage $T(x) = x^g M(x) + r(x)$
  - beachte: $x^g M(x) + r(x)$ ist ein Vielfaches von $G(x)$
- Empfänger
  - empfängt $m(x)$
  - berechnet den Rest: $m(x) \bmod G(x)$

$x^2 x^1 x^0$

$G = 111 \quad g = 2$

$M = 1000, M(x) = x^3$

$x^g \cdot M(x) = x^5$

100000

100000 : 111 = 1000

111000

1100

1110

100

111

11

Rest 11

100 10

1101

$T(x) = 100011$

$T(x) \bmod G(x) = 0$

CRC Übertragung und Empfang

- Keine Fehler:
  - T(x) wird korrekt empfangen
- Bitfehler: T(x) hat veränderte Bits
  - Äquivalent zur Addition eines Fehlerpolynoms E(x)
  - Beim Empfänger kommt T(x) + E(x) an
- Empfänger
  - Empfangen: m(x)
  - Berechnet Rest m(x) mod G(x)
  - kein Fehler: m(x) = T(x),
    - dann ist der Rest 0
  - mit Bitfehlern: m(x) mod G(x)
    = (T(x)+ E(x)) mod G(x)
    = T(x) mod G(x) + E(x) mod G(x)
    = 0 Fehlerindikator

$T(x) \bmod G(x) = 0$

T: 100011

E: 010000

m: 110011

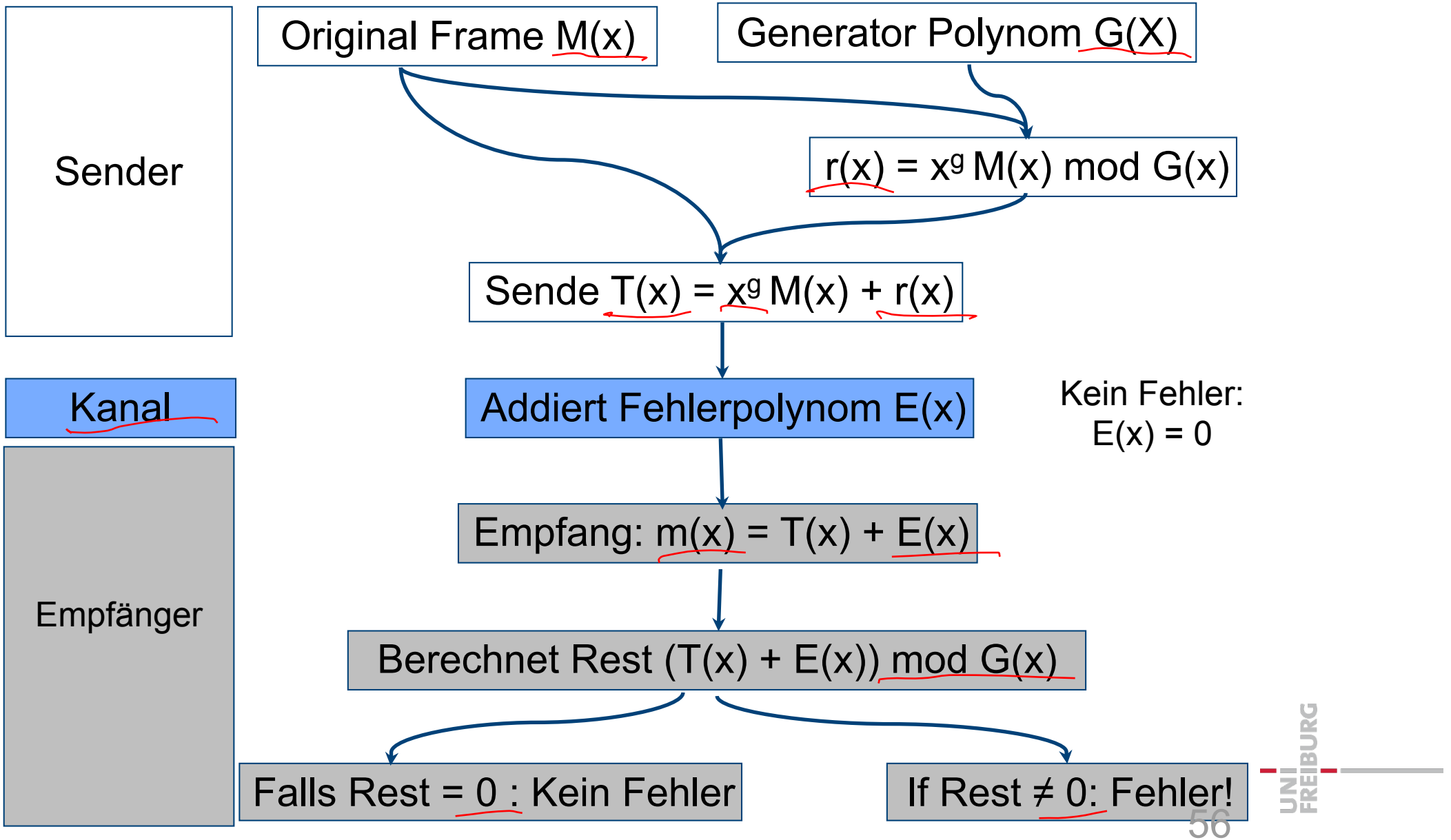
$G(x) = x^3 + x^2 + x + 1$

$x^3 + x^2 + x + 1 : x = x^2 + x + 1$

$x^2 + x + 1 : x = x + 1$ Rest 1

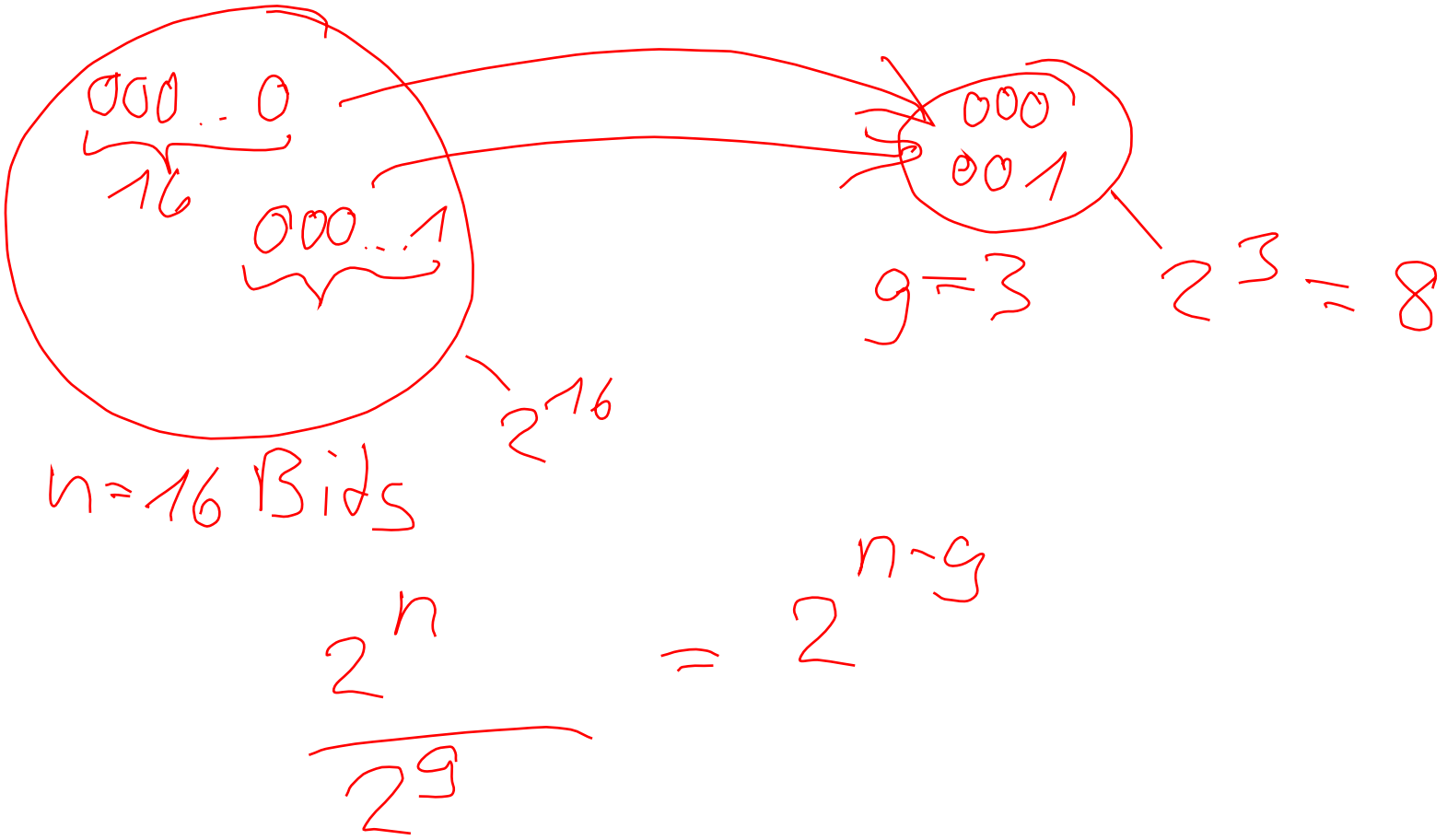
$0 \cdot G(x), 1 \cdot G(x), x \cdot G(x), (x+1)G(x)$

CRC – Überblick



CoNe Freiburg

Abbildung Daten auf CRC



Der Generator bestimmt die CRC-Eigenschaften

- Bit-Fehler werden nur übersehen, falls E(x) ein Vielfaches von G(x) ist
- Die Wahl von G(x) ist trickreich:
- Einzel-Bit-Fehler: $E(x) = x^i$ für Fehler an Position i
  - falls G(x) mindestens zwei Summenterme hat, dann ist E(x) kein Vielfaches von G(x) ist
- Zwei-Bit-Fehler: $E(x) = x^i + x^j = x^j (x^{i-j} + 1)$ für i>j
  - G(x) darf nicht $(x^k + 1)$ teilen für alle k bis zur maximalen Frame-Länge
- Ungerade Anzahl von Fehlern:
  - E(x) hat nicht (x+1) als Faktor
  - Gute Idee (?): Wähle (x+1) als Faktor von G(x)
    - Dann ist E(x) kein Vielfaches von G(x)
- Bei guter Wahl von G(x):
  - kann jede Folge von r Fehlern erfolgreich erkannt werden
- Häufig:
  - G(x) wird als irreduzibles Polynom gewählt, dass heißt es ist kein Vielfache eines anderen (kleineren) Polynoms

CRC Übertragung: Bit zuviel

CoNe Freiburg

b a

10110 10110 0 $\hat{=} a + b \cdot x^n$ $n = 6$

$T(x) = a + b \cdot x^n \bmod G(x) = 0$

0

mit Fehler: $m(x) = a + b \cdot x^{n+1} + 0 \cdot x^n$

$a + b \cdot x^{n+1} \bmod G(x) \neq 0$

$- a + b \cdot x^n$

$b \cdot (x^{n+1} + x^n) \bmod G(x) \neq 0$

Faktor

UNI FREIBURG

CRC in der Praxis

- Verwendetes irreduzibles Polynom gemäß IEEE 802:
  - $x^{32} + x^{23} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
- Achtung:
  - Fehler sind immer noch möglich
  - Insbesondere wenn der Bitfehler ein Vielfaches von G(x) ist.
- Implementation:
  - Für jedes Polynom x^i wird $r(x,i) = x^i \bmod G(x)$ berechnet
  - Ergebnis von B(x) mod G(x) ergibt sich aus
  - $b_0\, r(x,0) + b_1\, r(x,1) + b_2\, r(x,2) + \ldots + b_{k-1}\, r(x,k-1)$
  - Einfache Xor-Operation

Daten $b_{k-1} \ldots b_2 b_1 b_0$

$b_{k-1}x^{k-1} + \ldots + b_2x^2 + b_1x + b_0 \bmod G(x)$

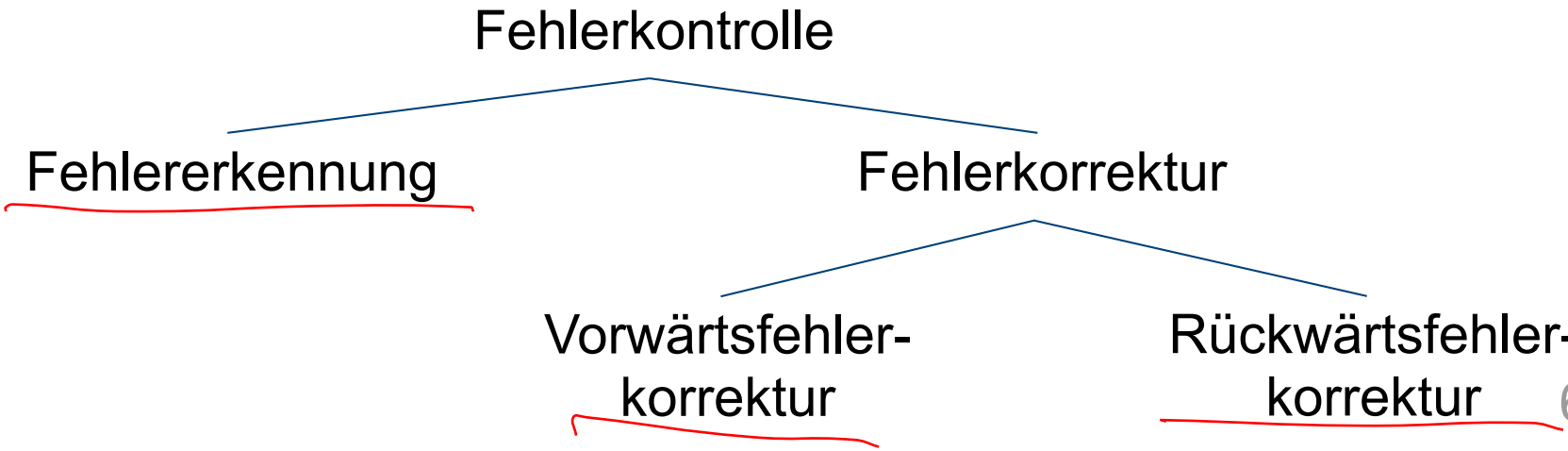
$a = b \bmod m,\ c = d \bmod m \Rightarrow a + c = b + d \bmod m$

$(b_{k-1}x^{k-1} \bmod G) \oplus \ldots \oplus (b_2x^2 \bmod G) \oplus (b_1 \ldots$

CoNe Freiburg

Fehlerkontrolle

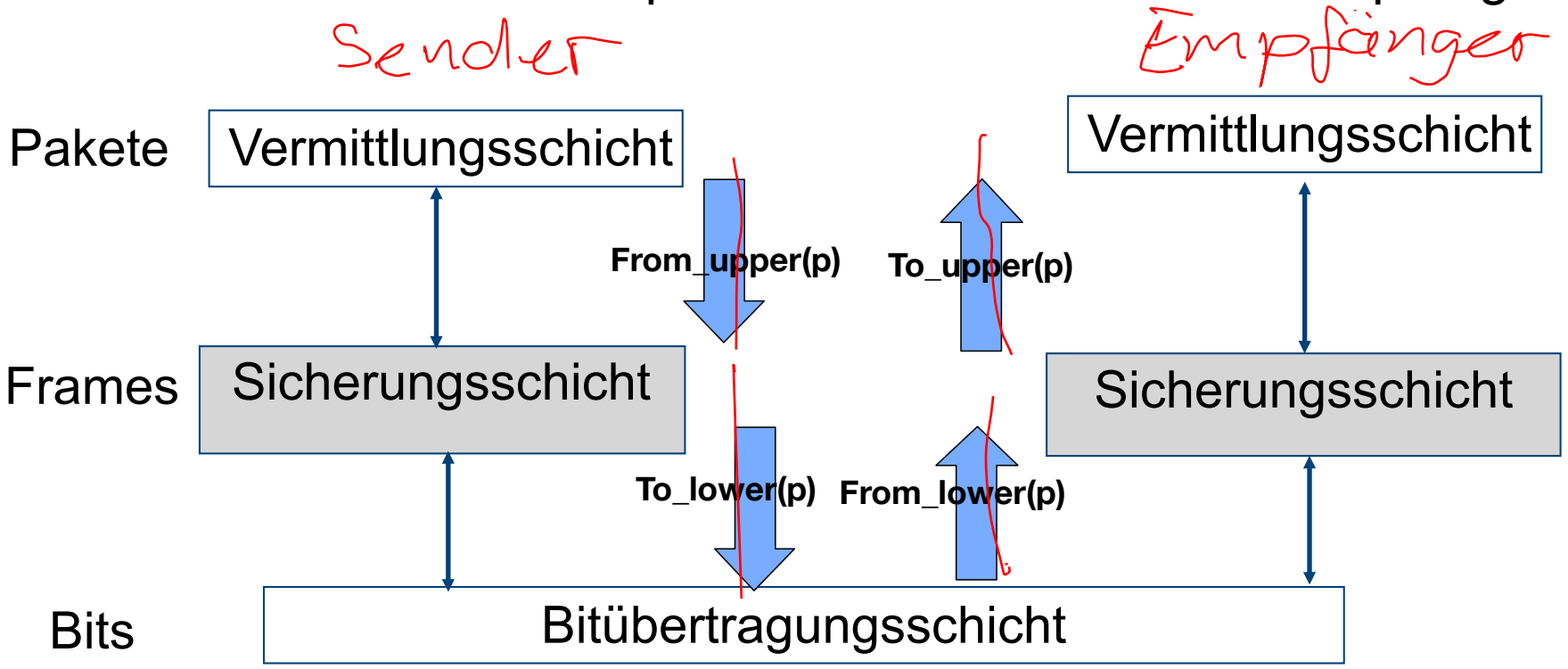
- Zumeist gefordert von der Vermittlungsschicht
  - Mit Hilfe der Frames
- Fehlererkennung
  - Gibt es fehlerhaft übertragene Bits?
- Fehlerkorrektur
  - Behebung von Bitfehlern
  - Vorwärtsfehlerkorrektur (Forward Error Correction)
    - Verwendung von redundanter Kodierung, die es ermöglicht Fehler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



UNI FREIBURG

Rückwärtsfehlerkorrektur

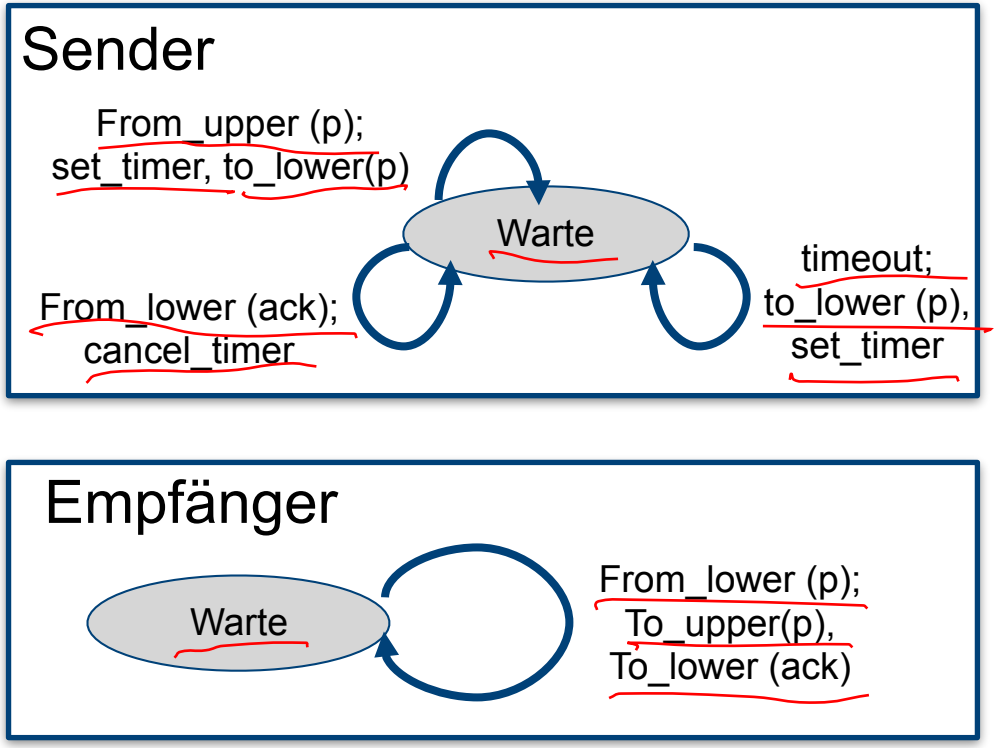
- Bei Fehlererkennung muss das Frame nochmal geschickt werden
- Wie ist das Zusammenspiel zwischen Sender und Empfänger?

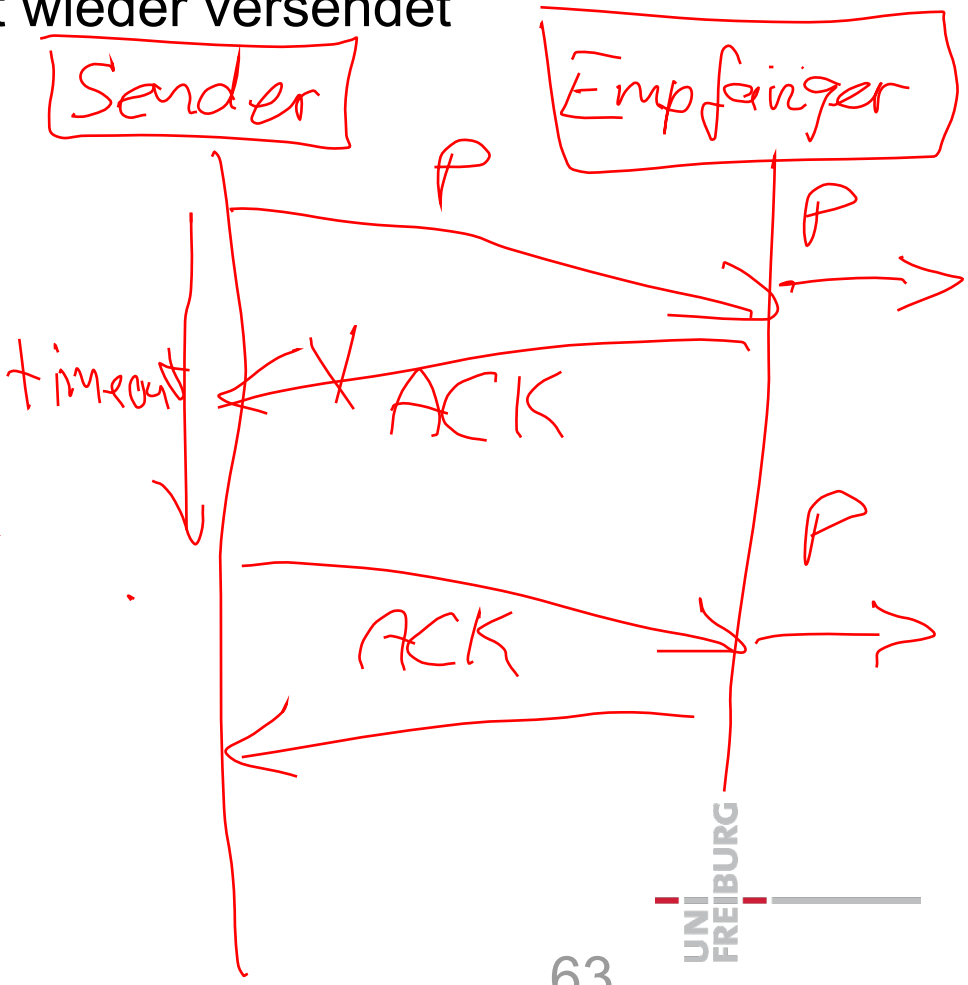


to_lower, from_lower beinhalten CRC
oder (bei Bedarf) Vorwärtsfehlerkorrektur

Einfaches Simplex-Protokoll mit Bestätigungen

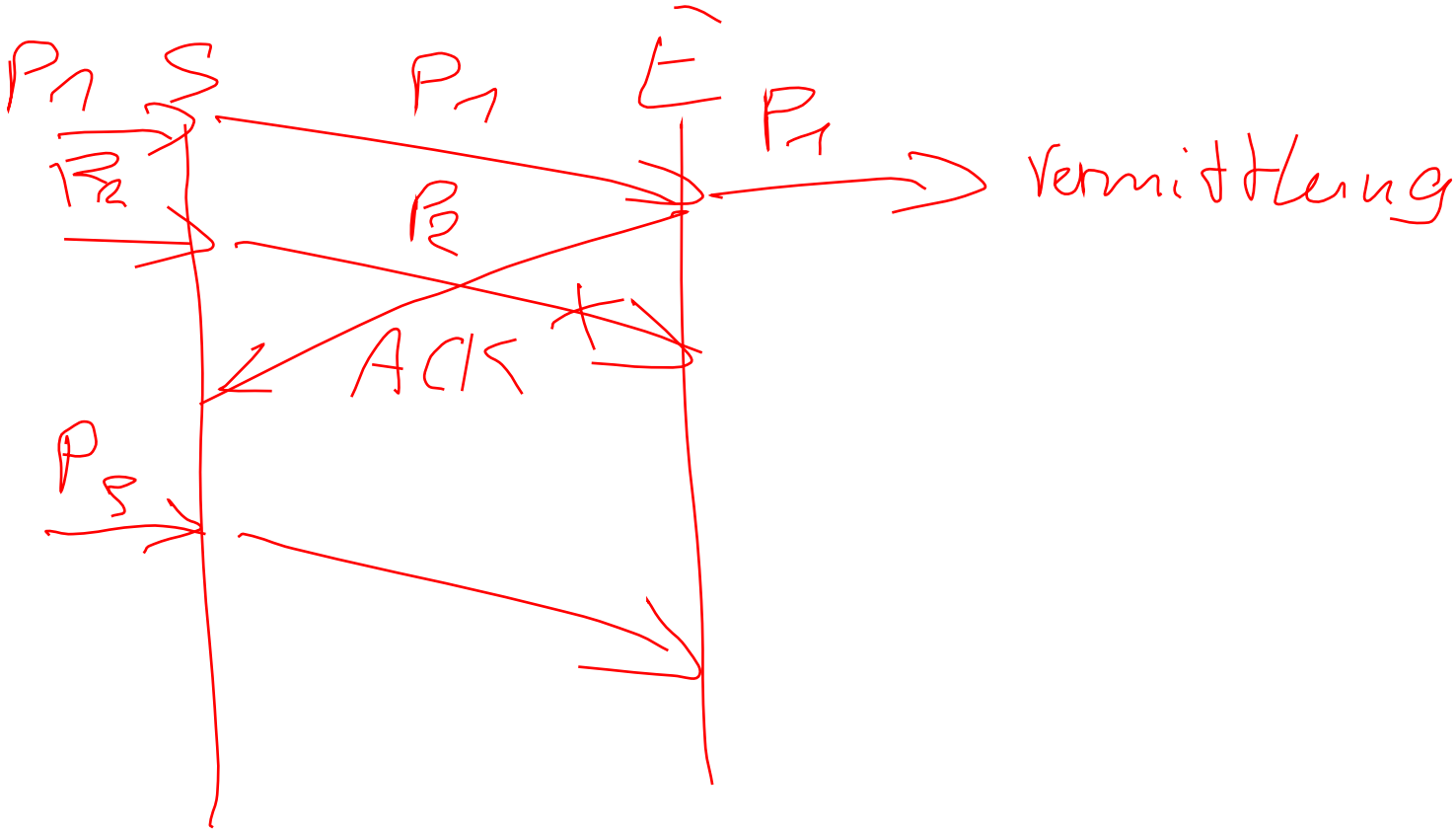
- Empfänger bestätigt dem Sender den Paketempfang
  - der Sender wartet für eine bestimmte Zeit auf die Bestätigung (acknowledgment)
  - nach Ablauf der Zeit wird das Paket wieder versendet
- erster Lösungsansatz:





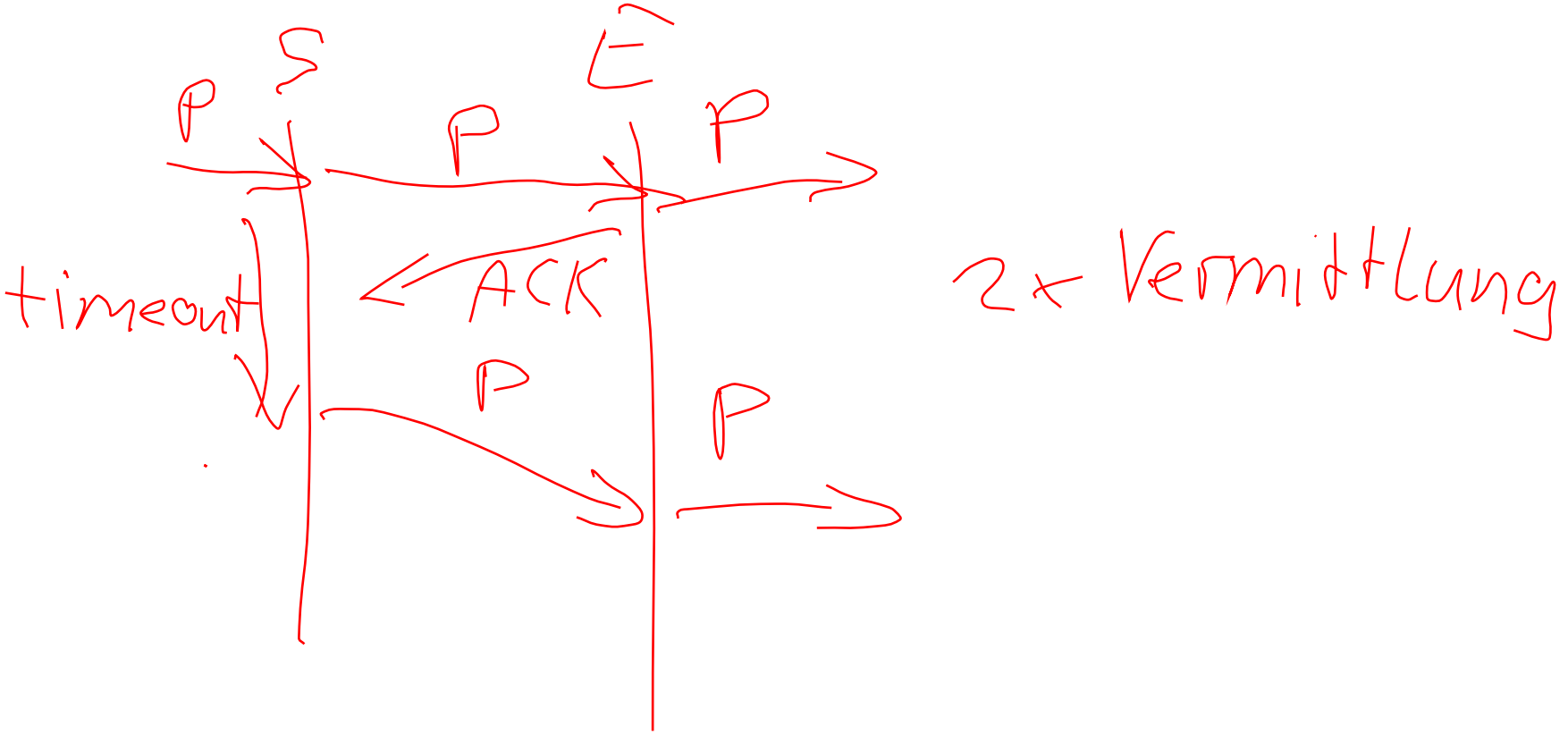
Diskussion

- Problem:
  - Sender ist schneller als Empfänger



Diskussion (2)

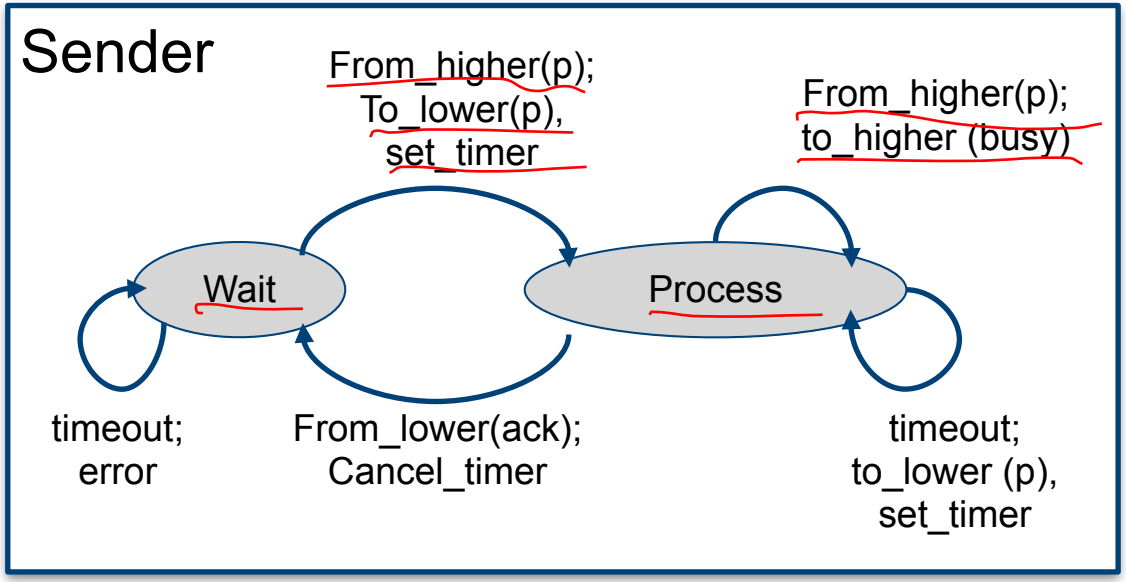
- Probleme
  - Was passiert, wenn Bestätigungen verloren gehen?

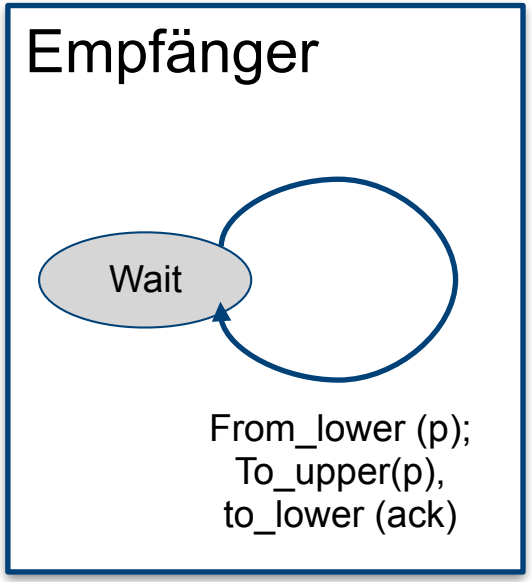


CoNe Freiburg

2. Versuch

- Lösung des Problems „schnellerer Sender als Empfänger“
  - Ein Paket nach dem anderen



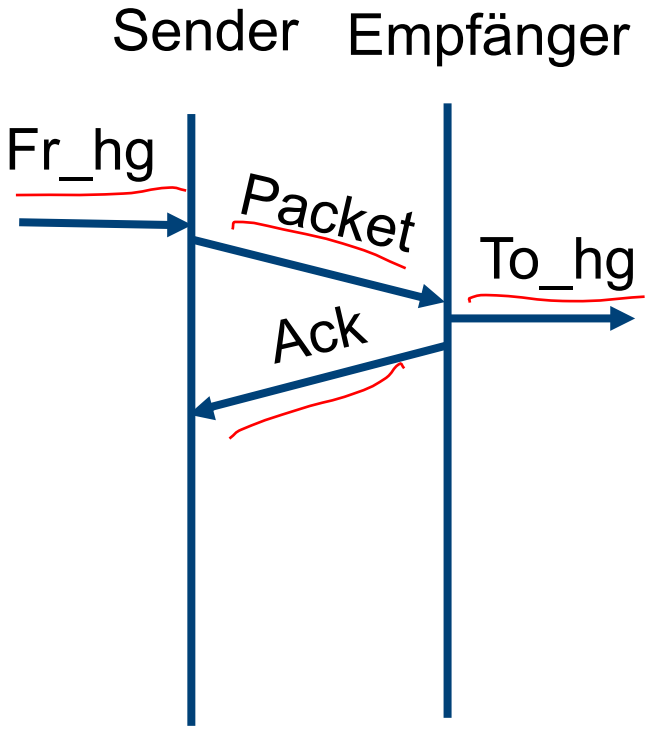


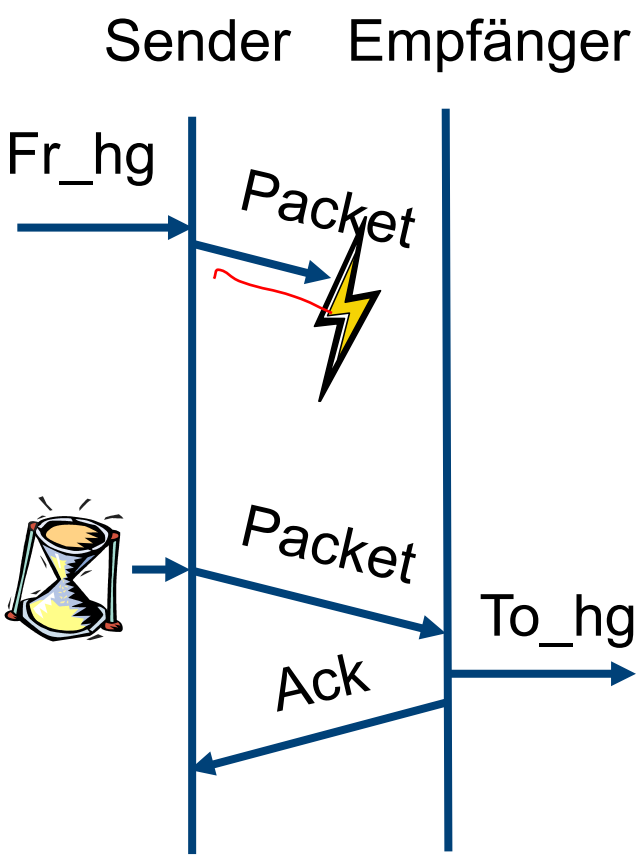
UNI FREIBURG

CoNe Freiburg

Diskussion

- Protokoll etabliert elementare Flusskontrolle

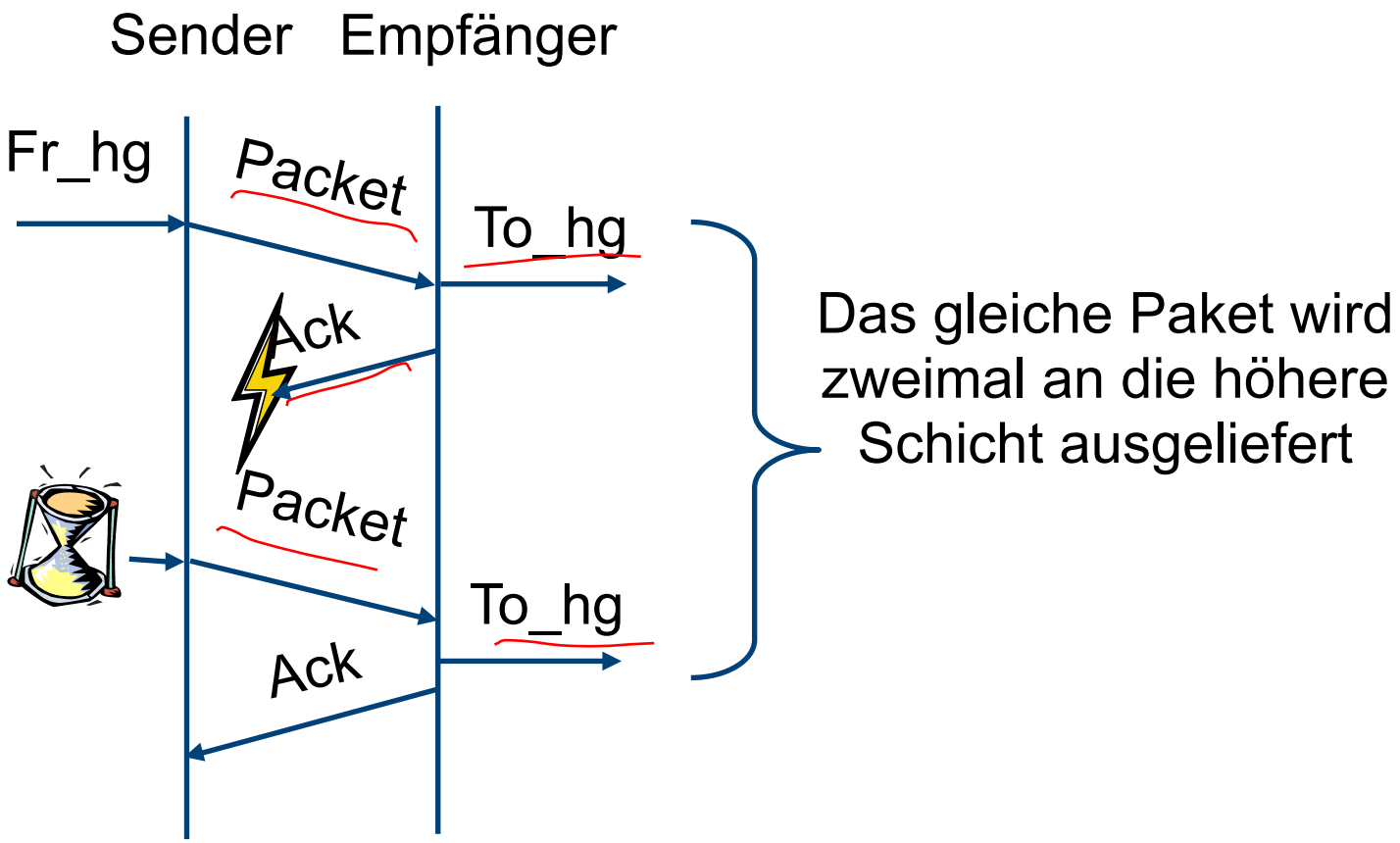




UNI FREIBURG

Diskussion

- 2. Fall: Verlust von Bestätigung

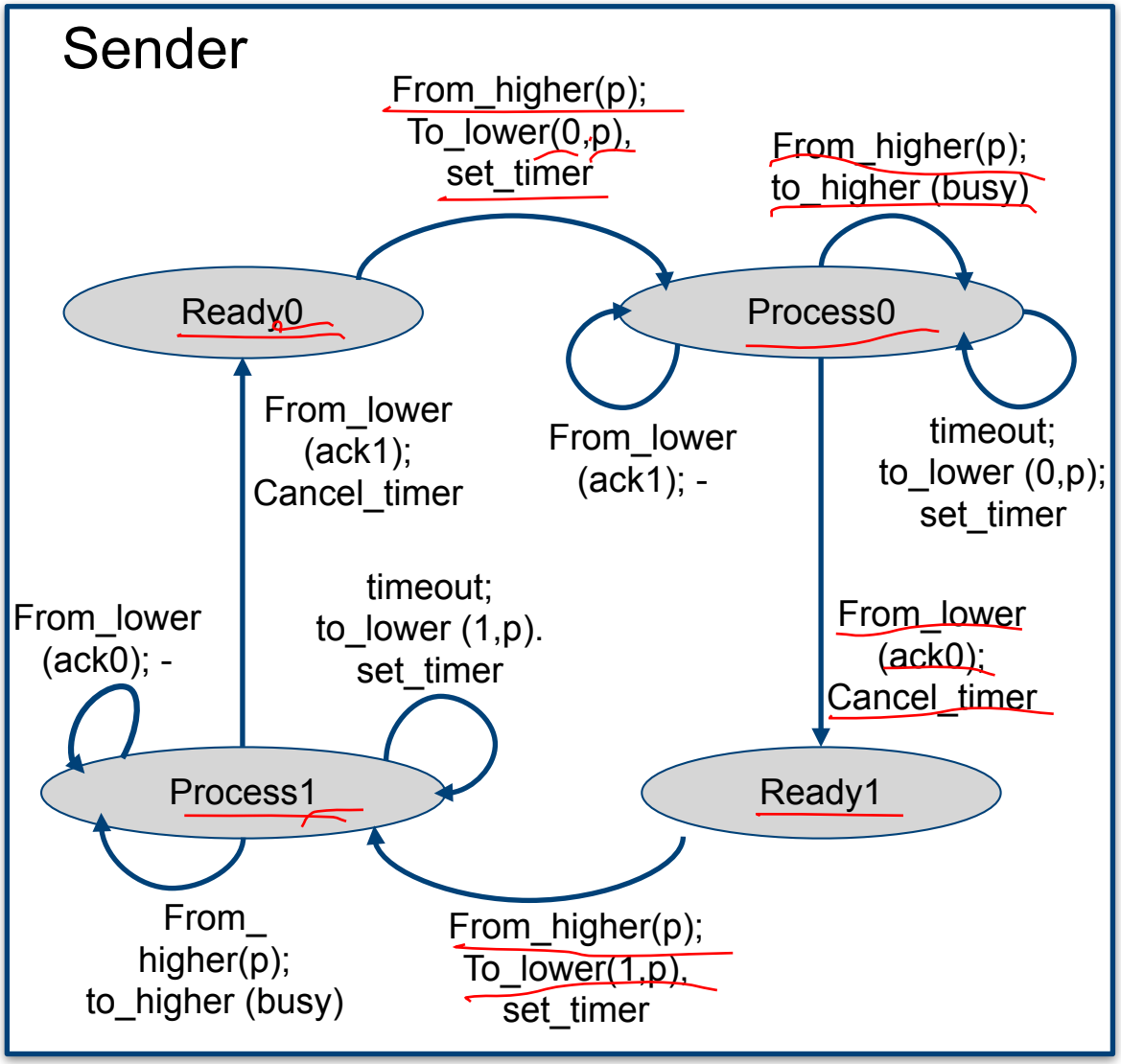


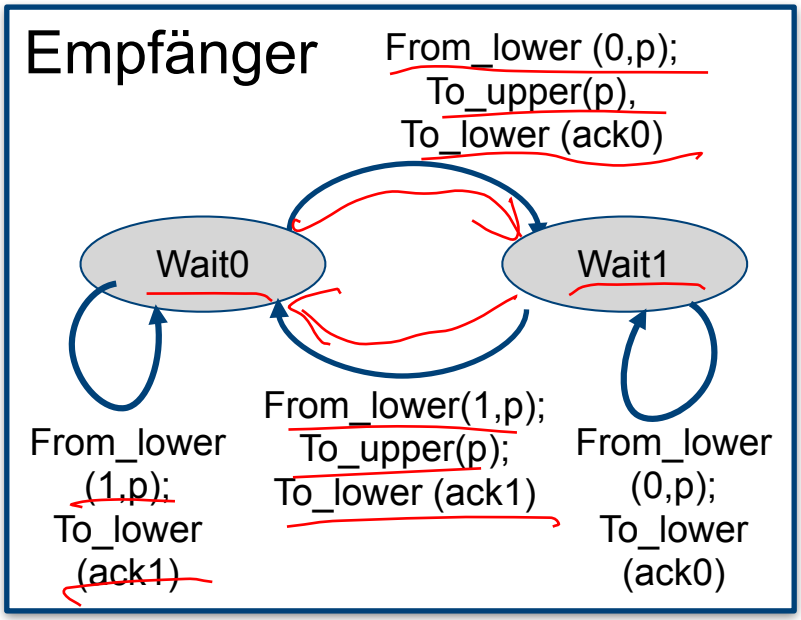
CoNe Freiburg

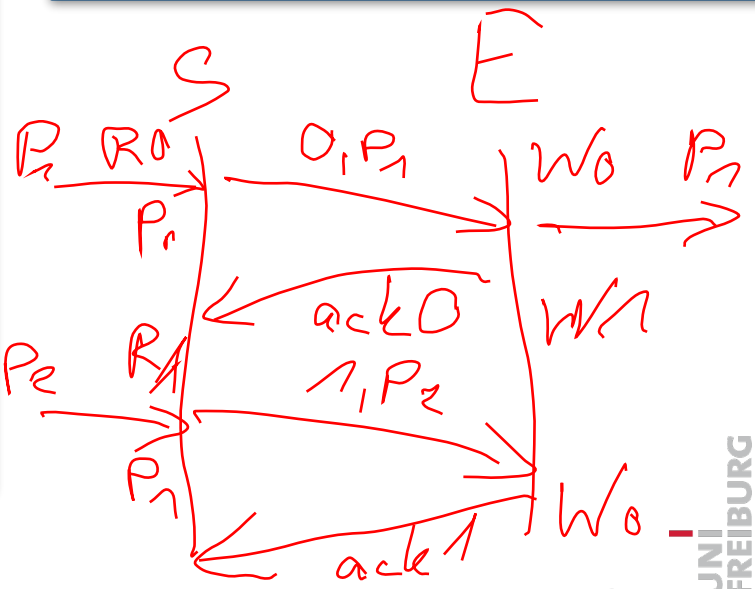
Probleme der 2. Version

- Sender weiß bei fehlender Bestätigung nicht, ob das Paket oder die Bestätigung verloren ging
  - Paket muss neu versendet werden
- Empfänger kann nicht zwischen Paket und redundanter Kopie eines alten Pakets unterscheiden
  - Zusätzliche Information ist notwendig
- Idee:
  - Pakete erhalten Sequenznummer zur Unterscheidung beim Empfänger
  - Sequenznummer ist im Header jedes Pakets
  - hier: nur 0 oder 1
- notwendig in Paket und Bestätigung
  - In der Bestätigung wird die Sequenznummer des letzten korrekt empfangenen Pakets mitgeteilt (reine Konvention)

3. Versuch: Bestätigung und Sequenznummern

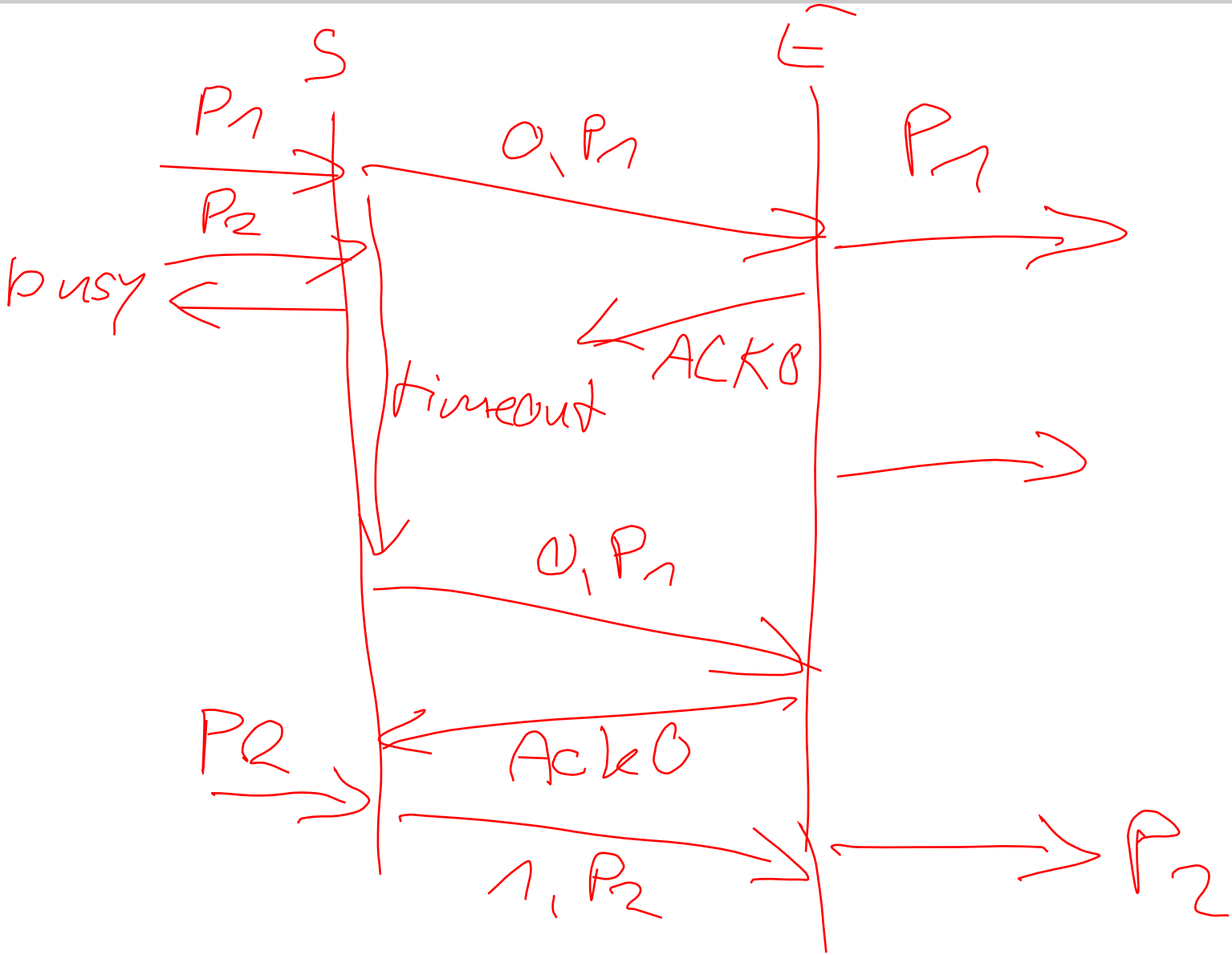






CoNe Freiburg

3. Versuch: Beispiel



CoNe Freiburg

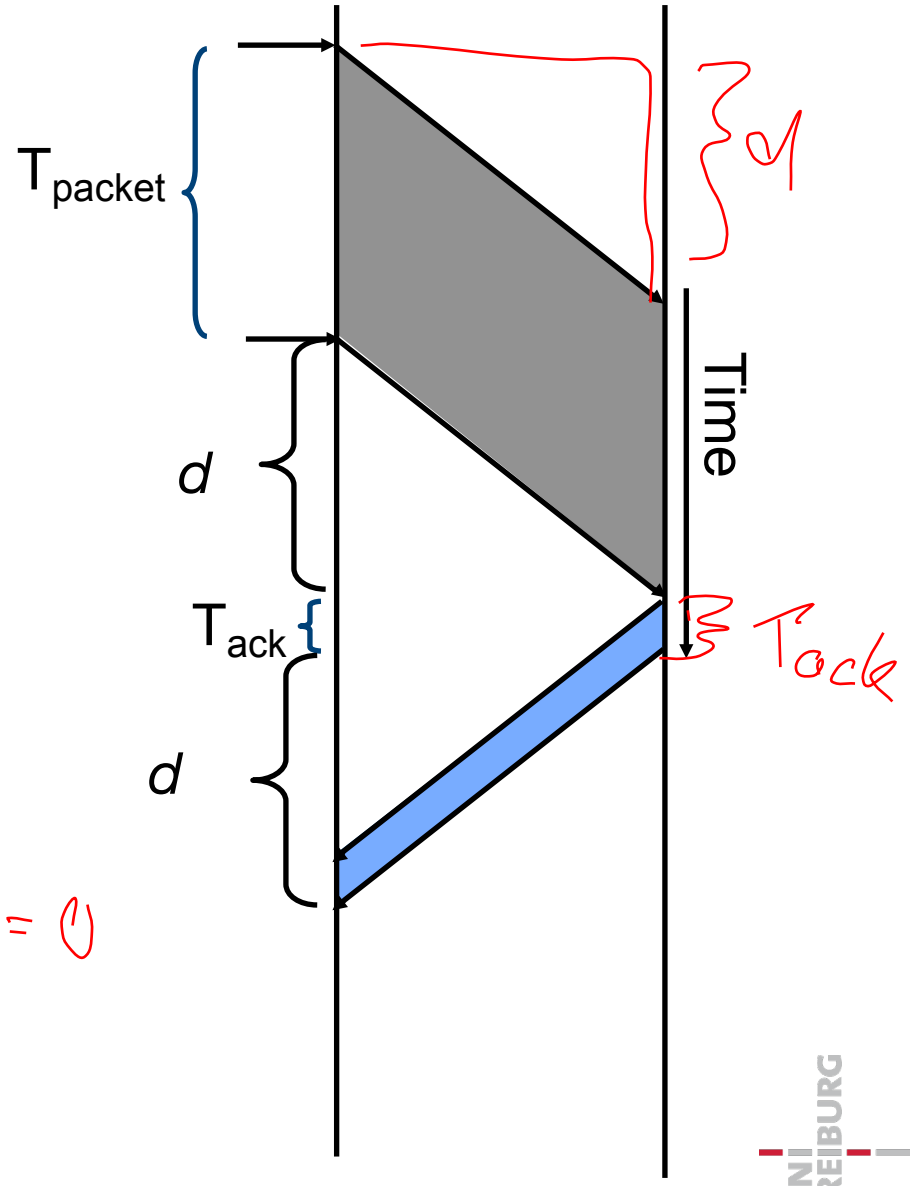
3. Version
Alternating Bit Protocol

- Die 3. Version ist eine korrekte Implementation eines verlässlichen Protokolls über einen gestörten Kanal
  - Alternating Bit Protokoll
  - aus der Klasse der Automatic Repeat reQuest (ARQ) Protokolle
  - beinhaltet auch eine einfache Form der Flusskontrolle
- Zwei Aufgaben einer Bestätigung
  - Bestätigung, dass Paket angekommen ist
  - Erlaubnis ein neues Paket zu schicken

Alternating Bit Protocol – Effizienz

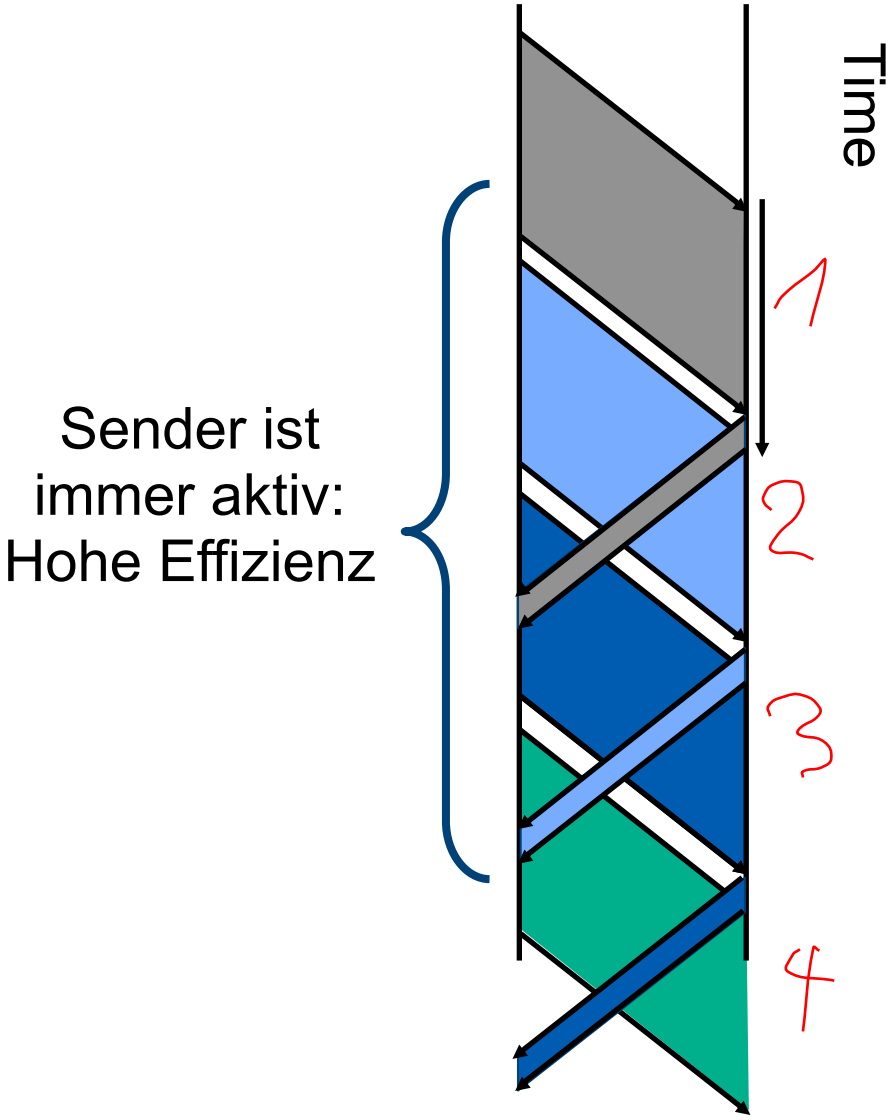
- Effizienz η
  - Definiert als das Verhältnis zwischen
    - der Zeit um zu senden
    - und der Zeit bis neue Information gesendet werden kann
    - (auf fehlerfreien Kanal)
  - $\eta = T_{packet} / (T_{packet} + d + T_{ack} + d)$
- Bei großen Delay ist das Alternating Bit Protocol nicht effizient

$\eta = 1$ für $d = 0$, $T_{ack} = 0$



Verbesserung der Effizienz

- Durchgehendes Senden von Paketen erhöht Effizienz
  - Mehr “ausstehende” nicht bestätigte Pakete erhöhen die Effizienz
  - “Pipeline” von Paketen
- Nicht mit nur 1-Bit-Sequenznummer möglich



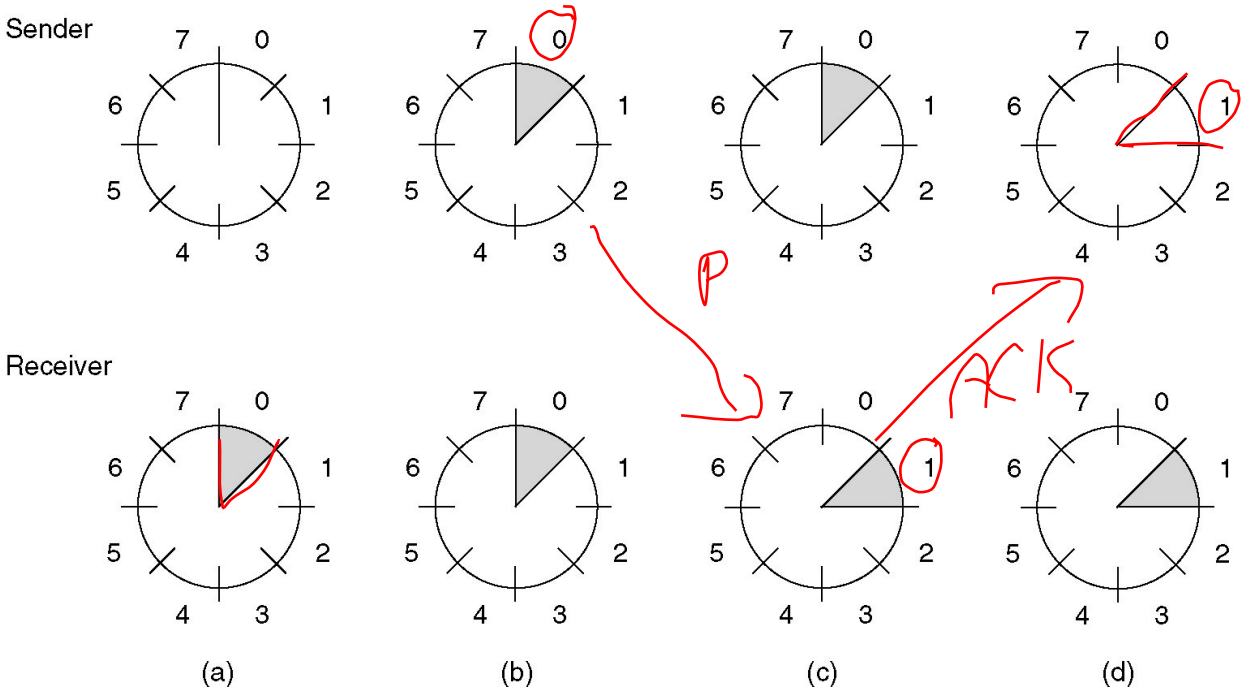
CoNe Freiburg

Gleitende Fenster

- Der Raum für Sequenznummern wird vergrößert
  - auf n Bits oder 2^n Sequenznummern
- Nicht alle davon können gleichzeitig verwendet werden
  - auch bei Alternating Bit Protocol nicht möglich
- “Gleitende Fenster” (sliding windows) bei Sender und Empfänger behandeln dieses Problem
  - Sender: Sende-Fenster
    - Folge von Sequenznummer, die zu einer bestimmten Zeit gesendet werden können
  - Empfänger: Empfangsfenster
    - Folge von Sequenznummer, die er zu einer bestimmten Zeit zu akzeptieren bereit ist
  - Größe der Fenster können fest sein oder mit der Zeit verändert werden
  - Fenstergröße entspricht Flusskontrolle

Beispiel

- “Sliding Window”-Beispiel für n=3 und fester Fenstergröße = 1
- Der Sender zeigt die momentan unbestätigten Sequenznummern an
  - Falls die maximale Anzahl nicht bestätigter Frames bekannt ist, dann ist das das Sende-Fenster



a. Initial: Nichts versendet

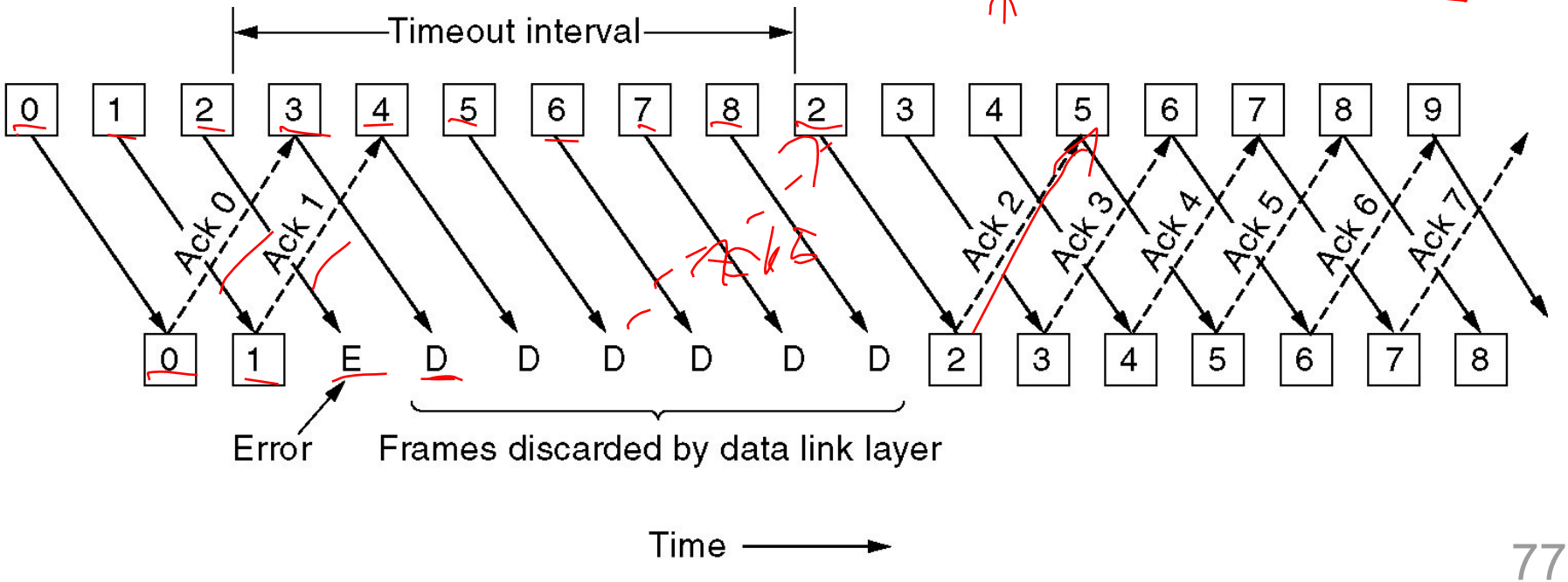
b. Nach Senden des 1. Frames mit Seq.Nr. 0

c. Nach dem Empfang des 1. Frame

d. Nach dem Empfang der Bestätigung

Übertragungsfehler und Empfangsfenster

- Annahme:
  - Sicherungsschicht muss alle Frames korrekt in der richtigen Reihenfolge verschicken
  - Sender “pipelined” Paket zur Erhöhung der Effizienz
- Bei Paketverlust:
  - werden alle folgenden Pakete ebenfalls fallen gelassen



CoNe Freiburg

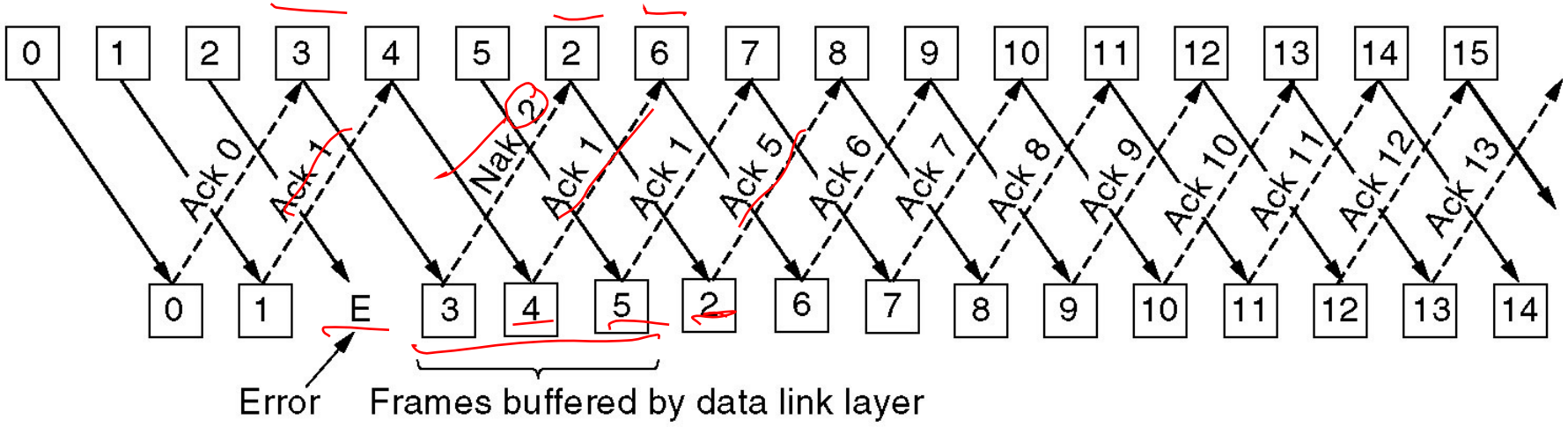
Go-back-N

- Mit Empfangsfenster der Größe 1 können die Frames, die einem verloren Frame folgen, nicht durch den Empfänger bearbeitet werden
  - Sie können einfach nicht bestätigt werden, da nur eine Bestätigung für des letzte korrekt empfangene Paket verschickt wird
- Der Sender wird einen “Time-Out” erhalten
  - Alle in der Zwischenzeit versandten Frames müssen wieder geschickt werden
  - “Go-back N” Frames!

- Kritik
  - Unnötige Verschwendung des Mediums
  - Spart aber Overhead beim Empfänger

Selektierte Wiederholung

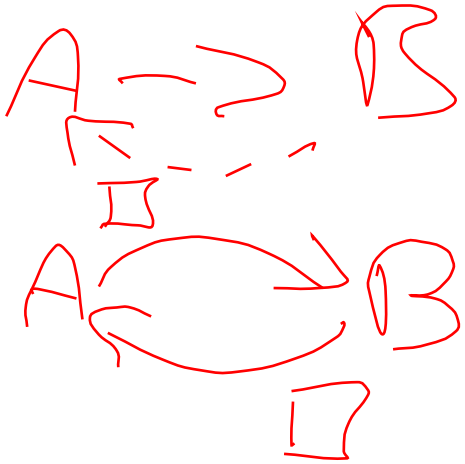
- Angenommen
  - der Empfänger kann die Pakete puffern, welche in der Zwischenzeit angekommen sind
  - d.h. das Empfangsfenster ist größer als 1
- Beispiel



  - Der Empfänger informiert dem Sender fehlende Pakete mit negativer Bestätigung
  - Der Sender verschickt die fehlenden Frames selektiv
  - Sobald der fehlende Frame ankommt, werden alle (in der korrekten Reihenfolge) der Vermittlungsschicht übergeben

Duplex-Betrieb und Huckepack

- Simplex
  - Senden von Daten in eine Richtung
- Duplex
  - Senden von Daten in beide Richtungen
- Bis jetzt:
  - Simplex in der Vermittlungsschicht
  - Duplex in der Sicherungsschicht
- Duplex in den höheren Schichten
  - Nachrichten und Datenpakete separat in jeder Richtung
  - oder Rucksack-Technik
    - Die Bestätigung wird im Header eines entgegen kommenden Frames gepackt



Piggyback