

# Protokolle zur Fehlerbehebung und Flußkontrolle



<https://iuk.one/1010-1018>

Clemens H. Cap

ORCID: [0000-0003-3958-6136](https://orcid.org/0000-0003-3958-6136)

Department of Computer Science  
University of **Rostock**  
Rostock, Germany  
[clemens.cap@uni-rostock.de](mailto:clemens.cap@uni-rostock.de)

Version 1



Wie wirken **Protokolle** und **Codierung** zusammen?

Wie bauen wir Protokolle zur **Fehlerbehebung** und zur **Flußkontrolle**?

Welche **Probleme** können dabei auftreten?

## 1. Überblick

**Ziel:** Wir wollen die Probleme erst einmal grundsätzlich einsortieren.

## 1. Überblick

2. Alternating Bit Protokoll

3. Protokoll-Verifikation

4. Deadlocks und Livelocks

5. Protokoll-Effizienz

### **Fehlerkorrektur** ([Forward] Error Correction)

- Erfordert fehlerkorrigierenden Code (Bsp: SEC Code).
- Einsatz bei hoher Sicherheitsanforderung, wenn erneute Übertragung vermieden werden muß.
- Bsp: Raumfahrt, Speichermedien.

### **Fehlererkennung und erneute Übertragung** (Error detection and retransmission)

- Erfordert fehlererkennenden Code (Bsp: DED Code).
- Separates Protokoll sichert erneute Übertragung.
- Einsatz in den meisten Kommunikationsprotokollen.

### **Zusammenwirken** von Codierung und Protokoll:

- Wird Fehler durch Codierung nicht behoben, muß Protokoll erneut übertragen.
- Bei vollständigem Daten-Verlust ebenso Protokoll-Aktivität benötigt.

# Mechanismen der Fehlerkontrolle

### Fragen:

- Wie erkennt Sender im Verlustfall, daß beim Empfänger etwas nicht geschehen ist?
- Wie unterscheidet Empfänger *gewollte Doppelübertragung* von *erneuter Übertragung wegen Verlust*?

### Mechanismen:

- **Bestätigung:** Positives Acknowledgement
- **Beschwerde:** Negatives Acknowledgement
- **Zeitüberwachung:** Time Out

### Wichtige Protokolle der Fehlerkorrektur:

- **Stop-and-Wait:** Warte jeweils, bis ein Paket korrekt übertragen.
- **Go-Back-N:** Gehe zurück bis zum Fehler, wiederhole ab dort alles.
- **Selective Repeat:** Wiederhole nur die fehlerhaften Pakete.

**Beachte:** Broadcast, Multicast & Anycast erfordern besondere Maßnahmen.

**Problematik:** Sender sendet schneller als Empfänger annehmen kann.

- Layer 1 im Empfänger erzeugt Interrupt, wenn Rahmen empfangen.
- Wenn Rahmen im Buffer nicht rechtzeitig ausgelesen, dann wird er überschrieben.

**Verfahren 1: Echtzeitverhalten** in Interrupt-Routine sicherstellen.

- Erfordert Realzeit Verhalten auf allen nachgeordneten (höheren) Schichten
- Sehr schwierig zu gewährleisten.
- Außerhalb spezieller Geräte daher meist Verfahren 2: Sender-Drosselung.

### Verfahren 2: Sender-Drosselung

Dazu gibt es drei Ansätze:

**Receiver-not-Ready:** Explizites Aushandeln der Empfangsbereitschaft.

- Empfänger sendet: RR: Receiver Ready Nachricht
- Empfänger sendet: RNR: Receiver Not Ready Nachricht
- Sender sendet: ENQ: Enquiry Nachricht

**Stop-and-Wait:** Nutze das Warten im Stop-and-Wait Fehlerprotokoll.  
Verzögerung der Bestätigung verzögert auch das Senden.

**Sliding Window:** Angabe einer Fenstergröße für maximal ausstehende Bytes.  
In Zusammenarbeit mit dem Fehlerprotokoll.

## 2. Alternating Bit Protokoll

### 2.1. Bestätigungsprotokoll

### 2.2. Phasenprotokoll

### 2.3. Alternating Bit Protokoll

**Ziel:** Das Alternating Bit Protokoll ist das einfachste Protokoll zur Behebung von Fehlern. Wir konstruieren das Protokoll und lernen dabei Probleme kennen, die bei der Konstruktion von Protokollen auftreten.

### 1. Überblick

### 2. Alternating Bit Protokoll

### 3. Protokoll-Verifikation

### 4. Deadlocks und Livelocks

### 5. Protokoll-Effizienz



# Aufgabenstellung

**Aufgabe:** Erstelle ein Protokoll zur Kommunikation zwischen Sender und Empfänger.

**Annahmen:**

- Es gibt einen Sender und einen Empfänger.
- Bei jeder Übertragung können Fehler auftreten.
- Fehler werden durch eine Codierung erkannt und signalisiert.
- Ein Paket kann auch gänzlich verloren gehen, was durch einen Timeout erkannt und signalisiert wird.
- Das Protokoll soll auf die erkannten Fehler reagieren und eine erneute Übertragung anstoßen.

# Landkarte unseres Vorgehens

- ① **Lösung 1:** Bestätigungsprotokoll.
- ② **Problem:** Erzeugt Duplikate.
- ③ **Lösung 2:** Phasenprotokoll.
- ④ **Problem:** Fehlfunktion bei Timeout.
- ⑤ **Lösung 3:** Alternating Bit Protokoll.

**Hinweis:** Bestätigungsprotokoll und Phasenprotokoll sind rein didaktische Protokolle, die nicht funktionieren und in der Praxis nicht genutzt werden.

Das **Alternating Bit Protokoll** funktioniert.  
Es wird (in optimierter Form) real eingesetzt.

# Beschreibung des Bestätigungsprotokolls

### Funktionsweise:

- Sender sendet ein Paket und wartet dann.
- Wenn Bestätigung kommt: Fortsetzen des Sendens.
- Wenn Fehlermeldung kommt: Erneutes Senden.
- Wenn Timeout kommt: Erneutes Senden

### Bewertung:

- Einfachstes **Stop-and-Wait** Protokoll: Tue etwas und warte auf Bestätigung.
- Auftretende Fehler werden behoben.
- Leistet zugleich auch noch Flußkontrolle.
- **Aber:** Verlorene Bestätigungen können Duplikate bewirken.

# Beschreibung des Phasenprotokolls

### Idee:

- Man muß einen Weg finden, Erstpaket und Wiederholungspaket zu unterscheiden.
- Der Sender vergibt eine von 2 sich abwechselnden Phasen.
- Phasenwechsel unterscheidet ein neues Paket von einem Wiederholungspaket.

### Umsetzung:

- Phasen sind Farbmarkierungen: **Rote** und **blaue** Pakete.
- Implementiert als 1 Bit im Header der entsprechenden Schichte.

### Bewertung:

- Das Phasenprotokoll kann Pakete verschlucken.
- **Es ist grob falsch.**

# Alternating Bit Protokoll

### Idee:

- Der Fehler entstand durch eine falsche Zuordnung von Bestätigungen.
- Um das zu vermeiden fügt man Phasen auch bei den Bestätigungen hinzu.

## 3. Protokoll-Verifikation

**Ziel:** Wir erkennen, daß der Entwurf korrekter Protokolle nicht so ganz einfach ist.

1. Überblick
2. Alternating Bit Protokoll
- 3. Protokoll-Verifikation**
4. Deadlocks und Livelocks
5. Protokoll-Effizienz

# Grundlegende Probleme

**Aspekt 1:** Wie beschreiben wir ein Protokoll genau?

Bisher: Unpräzise, sprachliche Beschreibung.

**Aspekt 2:** Was bedeutet es, daß ein Protokoll korrekt arbeitet?

Bisher: Unklare Vorstellung, was ein Protokoll leisten soll.

**Aspekt 3:** Wie kann man das nachweisen?

Benötigen ein formales Werkzeug (Logik, Algebra, usw.) für Beweise.

## Programm-Korrektheit

Ein Programm  $P$  **berechnet** eine Funktion  $f: \mathbb{N} \rightarrow \mathbb{N}$  **korrekt**, wenn gilt:

Gibt man den Wert  $x$  dem Programm  $P$  als Eingabe,  
dann hält das Programm nach einiger Zeit an und gibt den Wert  $f(x)$  aus.

**Hinweis:** Diese Begrifflichkeit kann und muß noch nachgebessert werden.

- Was ist ein "Programm"?
- Was bedeutet "Eingabe", "Anhalten" und "Ausgeben" genau.

Diese Aufgabe wird in der theoretischen Informatik gelöst.

Was bedeutet nun die **Korrektheit** eines Protokolls?



# Protokoll-Korrektheit

Um die Aufgabe zu vereinfachen, teilt man sie in 2 Aspekte auf:

### Sicherheitseigenschaften

Das Protokoll macht **nichts Falsches**, dh. es tut nur Dinge im Sinne der Spezifikation.

### Lebendigkeitseigenschaften

Das Protokoll macht **immer mal wieder etwas Gutes**, dh. es bewirkt immer mal wieder einen Fortschritt im Sinne der Spezifikation.

Bsp **Sicherheitseigenschaft**: Protokoll verliert keine Daten.

Bsp **Lebendigkeitseigenschaften**:

- **Hat kein Deadlock**: Es tritt kein dauerndes, produktivloses Warten ein.
- **Hat kein Livelock**: Es tritt keine dauernde, produktivlose Verwaltungsarbeit ein.

## 4. Deadlocks und Livelocks

### 4.1. Deadlocks

### 4.2. Livelocks

**Ziele:** Deadlocks und Livelocks sind wichtige Formen von Protokollfehlern. Letztere führen auch zum Begriff der **Fairness**.

1. Überblick
2. Alternating Bit Protokoll
3. Protokoll-Verifikation
4. **Deadlocks und Livelocks**
5. Protokoll-Effizienz

# Anekdote der Dining Philosophers

### Situation:

- Vier Philosophen sitzen um runden Tisch.
- In der Mitte eine Schüssel Spaghetti.
- Spaghetti kann man nur mit 2 Gabeln essen.
- Zwischen jeweils 2 Philosophen liegt 1 Gabel.

Können die Philosophen der Tätigkeit des Denkens nachgehen,  
ohne dabei zu verhungern?

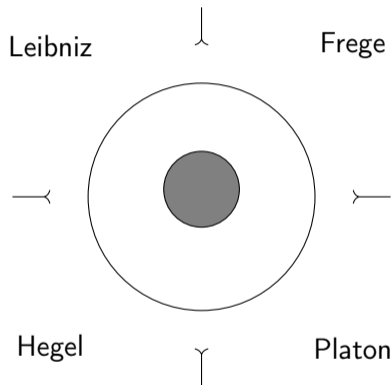


Abb. 1: Situation der Dining Philosophers.

# API-Struktur der Dining Philosophers

```
void think()           // Philosoph denkt
void eat()             // Philosoph ißt

void pickUpLeft()     // Philosoph hebt linke Gabel auf
                    // Wenn nicht da: Wartet bis erfolgreich aufgehoben
                    // Nachbedingung: Hat Gabel

void putDownLeft()   // Philosoph legt linke Gabel ab
void pickUpRight()   // Analog für rechte Gabel
void putDownRight()  // Analog für rechte Gabel
bool isLeftFree()    // Ist linke Gabel frei?
bool isRightFree()   // Ist rechte Gabel frei?

bool hungry()        // Ist Philosoph hungrig?
                    // Axiom: hungry() wird immer mal wieder wahr
```

**Src. 1:** Für jeden Philosophen bestehen diese API-Aufrufe.

## 4.1 Deadlocks

### Erstes Protokoll: Erzeugt Deadlocks

```
while (true) {  
    while ( !hungry() )    { think(); } // denke bis Hunger kommt  
    pickUpLeft();         // hebe erst linke Gabel auf; ggf warten  
    pickUpRight();        // hebe dann rechte Gabel auf  
    while ( hungry() )    { eat(); }   // esse solange hungrig  
    putDownLeft();  
    putDownRight();  
}
```

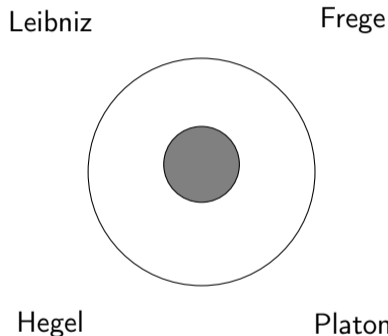
**Src. 2:** Das erste Protokoll der Philosophen führt auf ein Deadlock.

# Anekdote der Dining Philosophers

### Situation:

- Leibniz wird hungrig, hebt linke Gabel auf.
- Platon wird hungrig, hebt linke Gabel auf.
- Frege wird hungrig, hebt linke Gabel auf.
- Hegel wird hungrig, hebt linke Gabel auf.
- Alle Gabeln sind jetzt besetzt.
- Leibniz wartet auf seine rechte Gabel, die nicht kommt, denn Hegel wartet noch auf seine rechte Gabel, vorher kann er nicht essen und seine linke Gabel zurücklegen, auf die Leibniz wartet.
- Leibniz wartet auf Hegel, der auf Platon wartet, der auf Frege wartet, der auf Leibniz wartet.
- Leibniz wartet also auf Leibniz.

### Die Philosophen verhungern!



**Abb. 2:** Situation der Dining Philosophers, nachdem jeder Philosoph eine (seine linke) Gabel aufgehoben hat.

# Bedingungen für ein Deadlock

### Theorem von Coffman:

Ein Deadlock tritt auf, wenn die folgenden **4 Bedingungen** erfüllt sind:

- ① **Mutual Exclusion:** Betriebsmittel werden exklusiv genutzt.
- ② **No Preemption:** Kein Entzug von außen, keine vorzeitige Rückgabe.
- ③ **Hold and Wait:** Inkrementelle Reservation one-by-one.
- ④ **Circular Wait:** Prozeß-wartet-auf-Prozeß ist zyklisch.

**Beispiel:** Die Situation der Dining Philosophers.

- ① Gabel werden während des Essens nicht ausgetauscht.
- ② Gabeln werden erst am Ende des Essens zurückgelegt.
- ③ Gabeln werden erst-links-dann-rechts aufgehoben.
- ④ Es besteht im Deadlock eine zyklische Warte-Relation.

# Deadlock – Eigenschaften und Lösung

**Eigenschaften** des Deadlock:

- Ist inhärent stabil und daher (relativ) leicht zu detektieren.
- Ist leicht zu vermeiden.

**Lösungsansätze:** Sofern möglich: Eine der 4 Bedingungen aufbrechen.

**Beispiel 1:** Nur **gemeinsame Reservation aller** Betriebsmittel erlauben.

Pessimistische Lösung, da sie im Vorfeld Aufwand betreibt, etwas zu verhindern.

**Nachteil:** Schlechte Performanz. Alles reserviert, auch wenn nicht sofort benötigt.

**Beispiel 2:** **Deadlocks erkennen** und Gabel wieder wegnehmen.

Optimistische Lösung, da sie erst im Nachhinein auf ein Problem reagiert.

**Nachteil:** Geht nicht immer so einfach. (Bsp: Habe schon mit Drucken begonnen).



# Grundsätzliche Strategie

**Optimistische Strategie:** Wenn etwas passiert ist: Repariere es.

- **Vorteil:** Kein Aufwand für die Vermeidungs-Strategie.
- **Nachteil:** Gelegentlich Aufwand für die Reparatur-Strategie.

**Pessimistische Strategie:** Verhindere stets, daß etwas passiert.

- **Vorteil:** Kein Aufwand für die Reparatur-Strategie.
- **Nachteil:** Immer Zusatzaufwand für die Vermeidungs-Strategie.

**Entscheidung:** Wahl der richtigen Strategie hängt ab von:

- Kosten der Vermeidungs-Strategie und der Reparatur-Strategie.
- Wahrscheinlichkeit des Eintretens eines Problems.
- Wenn Schaden oft eintritt, dann besser die pessimistische Strategie.
- Adaptive Vorgehensweise möglich, um Kosten dynamisch zu optimieren.

# Idee zur Behandlung von Deadlocks

Es gibt etliche weitere Strategien zur Behandlung von Deadlocks.

Mehr dazu in einem Kurs zu Betriebssystemen.

Hier: Eine spezielle Strategie betrachten und daran weiteres Phänomen lernen.

**Beobachtung:** `pickUpLeft()` und `pickUpRight()` können blockieren.

**Idee 1:** Command überwachen; nur ausführen, wenn es nicht blockieren wird.

**Idee 2:** Hold-and-Wait Bedingung zerstören.  
Bei Gefahr eines Deadlock, Betriebsmittel zurückgeben.

### Zweites Protokoll: Erzeugt Livelocks

```
while (true) {  
    while ( !hungry() ) { think();      }  
    if (leftFree())      { pickUpLeft(); }  
    else                 { continue;    } // "verbrät" leer CPU Leistung  
    if (rightFree())    { pickUpRight(); }  
    else                 { putDownLeft(); // erlaubt Kollegen den Gabelzugriff  
                        continue; }    // "verbrät" leer CPU Leistung  
    while ( hungry() ) { eat(); }  
    putDownLeft();  
    putDownRight();  
}
```

**Src. 3:** Das zweite Protokoll der Philosophen hat etliche Nachteile: Es verbraucht sinnlos CPU-Leistung in Warteschleifen (busy wait). Als weiteres, wichtiges Problem tritt hier ein Livelock auf.

# Livelock

- Die Philosophen werden fast zur selben Zeit hungrig.
- Sie heben alle die linke Gabel auf.
- Sei legen alle die linke Gabel wieder zurück
- Sie heben alle die linke Gabel auf.
- Sie legen alle die linke Gabel wieder zurück.
- ...

**Problem:** Alle machen dauernd nur interne Verwaltungsarbeit

**Eigenschaften** des Livelock:

- Ist meistens inhärent instabil und daher schwer zu detektieren.
- Schwer zu vermeiden.
- Echtes Problem meist nur, wenn Realzeitverhalten gefordert ist

Elegante Lösungen erst mit fortgeschrittenen Konzepten von Betriebssystemen.

### Nicht-Determinismus

Das zukünftige Verhalten eines Systems kann nicht aus seinem aktuellen Zustand und auch nicht aus der gesamten Zustandshistorie vorhergesagt werden.

# Arten des Nicht-Determinismus

### Realer Nicht-Determinismus:

- Wir unterstellen die traditionellen Modelle der Physik (es gibt auch andere!)
- Dann: Im Bereich der Quantenphysik treten nicht-deterministische Effekte auf.
- **Bsp Radioaktivität:** Kann nicht vorhersagen, wann ein Uran-Atom zerfällt.

### Modellierungs Nicht-Determinismus:

- Wird in der Beschreibung genutzt, wenn wir zu wenig über ein komplexes System wissen, um es vollständig beschreiben zu können.
- **Bsp: Münzwurf:** Kann nicht vorhersagen, auf welche Seite eine Münze fällt.
- **Bsp: Betriebssystem:** Kann nicht vorhersagen, wann ein Programm terminiert.
- **Bsp: Benutzer:** Weiß nicht, was nächste Anforderung des Benutzers ist.
- **Bsp: Kanal:** Weiß nicht, wann ein Kanal einen Paketfehler macht.

# Motivation für Fairness

- Frage:** Wie modelliert man einen Kanal?
- Ansatz 1:** Der Kanal überträgt jedes zehnte Datenpaket falsch.
- Problem:** Ein Protokoll, das jeweils nur die Pakete 1–9 nutzt würde in Theorie funktionieren aber in Praxis versagen.
- Ansatz 2:** Ein Kanal hat immer die nicht-deterministische Auswahl zwischen "Funktionieren" und "Fehler Machen".
- Problem:** Wird der Kanal so spezifiziert, dann "darf" er *immer* "Fehler machen". Dann gibt es auch kein Protokoll zur Behebung von Fehlern.
- Ansatz 3:** Der Kanal muß fair nicht-deterministisch auswählen.  
Dh: Er muß beide Optionen immer mal wieder nutzen.

### Fairness

Eine nicht-deterministische Auswahl-situation heißt fair, wenn alles, was geschehen kann schließlich auch einmal geschehen wird.

Beachte: Diese anschauliche Definition wird in der Theorie nicht-deterministischer Systeme deutlich präziser gemacht.

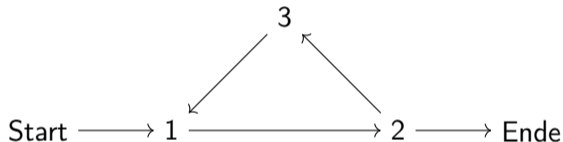
#### **Beispiel Münzwurf:**

Ein Münzwurf sei als nicht-deterministische Auswahl "Kopf oder Zahl" modelliert.

Die Auswahl ist fair, wenn die Münze immer mal wieder Kopf und immer mal wieder Zahl anzeigt.



# Fairness und Termination



**Abb. 3:** Endlicher Automat, mit nicht-deterministischer Auswahl in Zustand 2: Er kann terminieren oder in Zustand 3 weiter machen. Wenn die Auswahl in Zustand 2 fair ist, dann wird der Automat anhalten. Davor kann er endlich oft das Dreieck durchlaufen haben. Ist die Auswahl nicht fair, dann ist auch ein Verhalten zulässig, bei dem der Automat dauernd "im Dreieck läuft".

# Problem der verschworenen Kanäle

Im Alternating Bit Protokoll gibt es 2 Kanäle: Datenkanal und Bestätigungskanal. Beide Kanäle sind "fehlerhaft" oder "funktionsfähig" in fairer Auswahl.

**Problem:** Es kann sein, daß niemals ein Paket korrekt übertragen wird. Datenkanal und Bestätigungskanal können sich verschwören zu:

- ① Datenkanal sendet Paket, macht dabei Fehler.
- ② Bestätigungskanal sendet Fehler-Nachricht, diese kommt an.
- ③ Datenkanal sendet Paket (nochmal), dieses kommt an.
- ④ Bestätigungskanal sendet Bestätigung, diese kommt nicht an.

**Praktisch** nicht relevant, weil reale Kanäle sich nicht so verschwören.

**Aber:** Theoretisches Problem & keine Garantie für Realzeit-Verhalten möglich.

# Globale und lokale Fairness

**Lokale Fairness:** Jede einzelne Systemkomponente verhält sich fair.

**Globale Fairness:** Jedes globale Systemverhalten, das durch Interaktion von Komponenten immer wieder möglich erscheint, wird auch immer wieder einmal realisiert.

**Beachte:** Wenn jede Komponente lokal fair ist, dann bewirkt das nicht immer globale Fairness.

**Beispiel:** Zwei nebeneinander stehende Personen wollen miteinander reden.

- **Lokal fair:** Jeder sieht abwechselnd nach links und nach rechts.
- **Global unfair:** Wenn beide Personen stets gleichzeitig nach links oder gleichzeitig nach rechts schauen, können sie nie miteinander reden.

## 5. Protokoll-Effizienz

**Ziele:** Es genügt nicht, nur korrekte Protokolle zu entwickeln. Sie müssen auch leistungsfähig sein. Wir betreiben Leistungsanalyse und weisen auf Optimierungsmöglichkeiten von Protokollen hin.

1. Überblick
2. Alternating Bit Protokoll
3. Protokoll-Verifikation
4. Deadlocks und Livelocks
5. Protokoll-Effizienz

**Framezeit:** Zeit zur Übertragung eines Frames vom Buffer auf Draht oder umgekehrt.

Kleinstes Ethernet Paket            67.2 [ $\mu$ s]

Größtes Ethernet Paket            1'230.0 [ $\mu$ s]

Größtes Fast Ethernet Paket        46.7 [ $\mu$ s]

**Signallaufzeit:** Zeit zur Ausbreitung des Signals vom Sender zum Empfänger.

Licht im Vakuum            299'790 [km/s]

Signal in Twisted Pair        177'000 [km/s]

1 [m]    5.6 [ns]

1 [km]    5.6 [ $\mu$ s]

1'000 [km]    5.6 [ms]

35'000 [km]    196 [ms]

## 5. Protokoll-Effizienz

### Leistungsanalyse

Betrachte Stop-and-Wait Protokoll ohne Fehler, vergleiche mit theoretischem Maximum.  
Treffe Annahmen für LAN (1 [km] Abstand) und WAN (10'000 [km] Abstand).

	LAN	WAN
$t_d$ Framezeit für ein Datenpaket	1'230 [ $\mu$ s]	1'230 [ $\mu$ s]
$t_s$ Signallaufzeit zwischen Sender und Empfänger	5.6 [ $\mu$ s]	56'000 [ $\mu$ s]
$t_e$ Bearbeitungszeit beim Empfänger	10 [ $\mu$ s]	10 [ $\mu$ s]
$t_a$ Framezeit für ein Bestätigungs-Paket	67.2 [ $\mu$ s]	67.2 [ $\mu$ s]
$t_s$ Bearbeitungszeit beim Sender	10 [ $\mu$ s]	10 [ $\mu$ s]

$$\text{Effizienz} = \frac{t_d}{t_d + t_s + t_e + t_a + t_d + t_s}$$

Im LAN:0.92

Im WAN:0.01

# Interpretation der Leistungsanalyse

**Limitierender Faktor:** Zeit bis zum Eintreffen der Bestätigung bleibt ungenutzt.

Im **WAN:** Signallaufzeit sehr groß im Vergleich zum Rest.

Im **LAN:** Stop-and-Wait kann Sinn machen.

### Lösungsansätze:

- Andere Protokollmechanismen: Die Zeit bis zum Eintreffen der Bestätigung nutzen.
- Pakete größer machen (sog. Jumbo Frames).
  - Aber: Wahrscheinlichkeit eines Fehlers steigt.
  - Aber: Aufwand bei erneuter Übertragung steigt.

# Stop-and-Wait

**Konzept:** Sender sendet ein Paket und wartet auf Bestätigung.

### Mögliche Mechanismen:

- Positiv: Empfänger versendet Bestätigung über korrekten Empfang.
- Negativ: Empfänger sendet Beschwerde bei Fehler.

**Problem:** Negatives Verfahren versagt bei Paketverlust.

**Lösung 1:** Permanent senden, ggf. leer, und Timeouts nutzen.

**Nachteil:** Unnützes Blockieren des Mediums mit Leer-Paketen!

**Lösung 2:** Sequenznummern und Beschwerde spätestens beim nächsten Paket.  
Letztes Paket braucht Ausnahmelösung.

**Aber:** Mangelnde Performance im WAN bleibt zentrales Problem.



# Go-Back-N

- Konzept:** Maximal  $n$  ausstehende Bestätigungen.
- Pipelining:** Sender kann mehrfach senden bevor Bestätigung eintreffen muß.
- Fehlerfall:** Erneute Übertragung ab dem fehlerhaften Rahmen  
inklusive aller folgenden (korrekten oder bereits bestätigten) Rahmen.

### Bewertung:

- Einfache Strategie.
- Effizienter als Stop-and-Go.
- Verschwenderisch, da ggf. korrekte Pakete verworfen werden.
- Bei hoher Fehlerrate: Sehr verschwenderisch.
- Real: Kein besonderes Problem, da meistens Burst-Fehler auftreten und dann die folgenden Pakete ebenfalls fehlerhaft sind.

# Selective Repeat

### Konzept:

- Empfänger verwirft nur das fehlerhafte Paket.
- Sender läuft in ein Time Out **für dieses Paket**.
- Sender überträgt jeweils das älteste nichtbestätigte Paket.
- Die anderen Pakete sind ggf. schon bestätigt.
- Empfänger erhält das fehlerhafte Paket neu.
- Empfänger kann nun ganze Reihe fehlerfreier Pakete an oberen Layer abliefern.

### Bewertung:

- Hohe Effizienz.
- Protokoll sehr komplex.

# Variante der Rückmeldungen

### **Kumulatives Acknowledgement:**

- Bestätige nicht jeden Rahmen separat sondern kumuliere
- Bsp: Alle Rahmen bis Rahmen X wurden empfangen
- Parameter: Wie lange bis zu einem ACK warten

### **Piggyback Acknowledgement:**

- Bestätigung wird gemeinsam mit normalem Datenpaket versandt.
- Sinnvoll, wenn häufiger Wechsel der Transferrichtung erfolgt.

### **Sofortige Beschwerden:**

- Bei einem fehlerhaften Paket sofort protestieren (NAK, negative acknowledgement).
- Erlaubt sofortige Neuübertragung des fehlerhaften Pakets.

# Anhang

Übersicht

Programmquellenverzeichnis

Prog

Verzeichnis aller Abbildungen

Abb

Rechtliche Hinweise

§

Zitierweise dieses Dokuments

→

Verzeichnis aller Folien



1	API-Struktur der Dining Philosophers .....	20
2	Erstes Protokoll der Philosophen .....	21
3	Zweites Protokoll der Philosophen .....	27

1	Situation der Dining Philosophers (1) .....	19
2	Situation der Dining Philosophers (2) .....	22
3	Endlicher Automat mit nicht-deterministischem Übergang .....	33

Die hier angebotenen Inhalte unterliegen deutschem Urheberrecht. Inhalte Dritter werden unter Nennung der Rechtsgrundlage ihrer Nutzung und der geltenden Lizenzbestimmungen hier angeführt. Auf das Literaturverzeichnis wird verwiesen. Das **Zitatrecht** in dem für wissenschaftliche Werke üblichen Ausmaß wird beansprucht. Wenn Sie eine Urheberrechtsverletzung erkennen, so bitten wir um Hinweis an den auf der Titelseite genannten Autor und werden entsprechende Inhalte sofort entfernen oder fehlende Rechtsnennungen nachholen. Bei Produkt- und Firmennamen können Markenrechte Dritter bestehen. Verweise und Verlinkungen wurden zum Zeitpunkt des Setzens der Verweise überprüft; sie dienen der Information des Lesers. Der Autor macht sich die Inhalte, auch in der Form, wie sie zum Zeitpunkt des Setzens des Verweises vorlagen, nicht zu eigen und kann diese nicht laufend auf Veränderungen überprüfen.

Alle sonstigen, hier nicht angeführten Inhalte unterliegen dem Copyright des Autors, Prof. Dr. Clemens Cap, ©2020. Wenn Sie diese Inhalte nützlich finden, können Sie darauf verlinken oder sie zitieren. Jede weitere Verbreitung, Speicherung, Vervielfältigung oder sonstige Verwertung außerhalb der Grenzen des Urheberrechts bedarf der schriftlichen Zustimmung des Rechteinhabers. Dieses dient der Sicherung der Aktualität der Inhalte und soll dem Autor auch die Einhaltung urheberrechtlicher Einschränkungen wie beispielsweise **Par 60a UrhG** ermöglichen.

Die Bereitstellung der Inhalte erfolgt hier zur persönlichen Information des Lesers. Eine Haftung für mittelbare oder unmittelbare Schäden wird im maximal rechtlich zulässigen Ausmaß ausgeschlossen, mit Ausnahme von Vorsatz und grober Fahrlässigkeit. Eine Garantie für den Fortbestand dieses Informationsangebots wird nicht gegeben.

Die Anfertigung einer persönlichen Sicherungskopie für die private, nicht gewerbliche und nicht öffentliche Nutzung ist zulässig, sofern sie nicht von einer offensichtlich rechtswidrig hergestellten oder zugänglich gemachten Vorlage stammt.



# Zitierweise dieses Dokuments

Wenn Sie Inhalte aus diesem Werk nutzen oder darauf verweisen wollen, zitieren Sie es bitte wie folgt:

Clemens H. Cap: Protokolle zur Fehlerbehebung und Flußkontrolle. Electronic document.  
<https://iuk.one/1010-1018> 17. 1. 2021.

**Bibtex Information:** <https://iuk.one/1010-1018.bib>

```
@misc{doc:1010-1018,  
  author      = {Clemens H. Cap},  
  title       = {Protokolle zur Fehlerbehebung und Flußkontrolle},  
  year        = {2021},  
  month       = {1},  
  howpublished = {Electronic document},  
  url         = {https://iuk.one/1010-1018}  
}
```

## Typographic Information:

Typeset on January 17, 2021

This is pdfTeX, Version 3.14159265-2.6-1.40.21 (TeX Live 2020) kpathsea version 6.3.2

This is pgf in version 3.1.5b

This is preamble-slides.tex myFormat©C.H.Cap

- 1 Titelseite
- 2 Ziel

## 1. Überblick

- 4 Fehlerkontrolle
- 5 Mechanismen der Fehlerkontrolle
- 6 Flußkontrolle
- 7 Mechanismen

## 2. Alternating Bit Protokoll

- 9 Aufgabenstellung
- 10 Landkarte unseres Vorgehens

### 2.1. Bestätigungsprotokoll

- 11 Beschreibung des Bestätigungsprotokolls

### 2.2. Phasenprotokoll

- 12 Beschreibung des Phasenprotokolls

### 2.3. Alternating Bit Protokoll

- 13 Alternating Bit Protokoll

## 3. Protokoll-Verifikation

- 15 Grundlegende Probleme
- 16 Programm-Korrektheit
- 17 Protokoll-Korrektheit

## 4. Deadlocks und Livelocks

### 4.1. Deadlocks

- 19 Anekdote der Dining Philosophers
- 20 API-Struktur der Dining Philosophers
- 21 Erstes Protokoll: Erzeugt Deadlocks
- 22 Anekdote der Dining Philosophers
- 23 Bedingungen für ein Deadlock
- 24 Deadlock – Eigenschaften und Lösung
- 25 Grundsätzliche Strategie

### 4.2. Livelocks




- 26 Idee zur Behandlung von Deadlocks
- 27 Zweites Protokoll: Erzeugt Livelocks
- 28 Livelock
- 29 Nicht-Determinismus
- 30 Arten des Nicht-Determinismus
- 31 Motivation für Fairness
- 32 Fairness
- 33 Fairness und Termination
- 34 Problem der verschworenen Kanäle
- 35 Globale und lokale Fairness

## 5. Protokoll-Effizienz

- 37 Zeiten
- 38 Leistungsanalyse
- 39 Interpretation der Leistungsanalyse
- 40 Stop-and-Wait
- 41 Go-Back-N
- 42 Selective Repeat

## 43 Variante der Rückmeldungen

### Legende:

-  Fortsetzungsseite
-  Seite ohne Überschrift
-  Bildseite