Fehlertoleranz

Kurseinheit 1 - Prinzipien der Fehlertoleranz und Ausfallsicherheit

1.1 Fehlertoleranz vs. Systemsicherheit

- Fehlertoleranz: versucht die Effekte eines auftretenden Fehlers zu minimieren
- Systemsicherheit: Wahrscheinlichkeit des Auftretens eines Fehlers soll minimiert bzw. komplett ausgeschlossen werden
- **Definition Fehlertoleranz**: Ein System ist fehlertolerant, wenn es trotz des Auftretens unvorhergesehener Fehler weiterhin in der Lage ist seine Funktionen korrekt auszuführen.

Ein fehlertolerantes System hat die Eigenschaft, dass es bei Auftreten von Fehlern *graziös degeneriert*, d.h. möglicherweise mit verringerter Effizienz oder Genauigkeit weiterarbeitet.

- **Definition Systemsicherheit**: Ein System gilt als sicher, wenn die Wahrscheinlichkeit auftretender Fehler minimiert ist.

Das Konzept der Systemsicherheit befasst sich mit der Minimierung des Fehlerrisikos. Natürlich kann man das Auftreten von Fehlern nie völlig ausschließen.

Kategorien von Fehlern und Defekten

- **Fehler und Defekt** (generell): fehlerhaftes Verhalten von Systemkomponenten
- **Error**: wie sich ein Fehlverhalten eines Systems manifestiert

  Beispiel (Fehler bei der Implementierung): Die Funktion berechnet fälschlicherweise nur positive Funktionswerte. Dieser Fehler ist nur dann Error, wenn eigentlich ein negativer Wert rauskommen müsste. Der Error zeigt sich folglich **nicht** bei allen Eingaben.

1. **Permanente Fehler** sind Fehler, die bei Auftreten die jeweilige Komponente oder das System permanent außer Betrieb setzen.
2. **Transiente Fehler** sind Fehler, die dazu führen, dass eine Komponente oder ein System für ein bestimmtes Zeitintervall nicht korrekt arbeitet. Nach diesem Zeitintervall ist der Fehler nicht mehr präsent und das korrekte Systemverhalten ist wiederhergestellt.
3. **Sporadische Fehler** sind generell immer präsent, zeigen sich in unregelmäßigen Intervallen als Fehlverhalten des Systems.

Fehlerart 2. und 3. werden durch Testfälle abgedeckt, da schwer reproduzierbar.

- *unkritische* Fehler: Fehler generiert keine falschen Resultate
- *kritische* Fehler: verfälschte Werte werden an andere Komponenten weitergeben und erzeugen fehlerhafte Ergebnisse (byzantinische Fehler)

Redundanz

> Definition: Verfügbarkeit von mehr Ressourcen als für das Funktionieren des Systems minimal notwendig

Vermeidung eines `Single-Point-of-Failure`

- **Hardwareredundanz**: funktionskritische Elemente eines Systems werden dupliziert, bei Ausfall kann es zu einer Reduzierung der Qualität oder Effizienz kommen
- **statische Redundanz**: redundante Ressourcen sind aktiv (Fehler wird maskiert, da Last auf verbliebene Ressourcen verteilt wird)
- **dynamische Redundanz**: werden bei Bedarf als Ersatzkomponenten aktiviert (z.B. Netzteil Fault-Over)

Blockdiagramme

Komponenten **U1**….**Un** sind seriell angeordnet. Der Fehlerzustand **F** vom Gesamtsystem **S**:

```
F(S) = F(U1)   F(U2)   ...   F(Un)
```

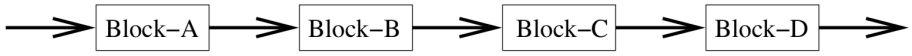

Figure 1: Diagramm serieller Systemkomponenten

Hier fällt auf, das ein Fehler in irgendeiner der vier Komponenten zu einem nicht funktionierenden System führt. Das heißt, dieses System enthält keine redundante Komponente, die im Fehlerfall die Funktionen der fehlenden Komponente übernehmen kann.

Im Gegensatz zu einem komplett seriellen System kann ein System so konzipiert sein, dass alle Komponenten parallel zueinander angeordnet sind und es somit einen hohen Grad an Redundanz aufweist.

```
F(S) = F(U1)   F(U2)   ...   F(Un)
```

Es ist möglich, serielle und parallele Strukturen zu verbinden um komplexere Systeme darzustellen.

F(S) = (F(A) F(B)) (F(C) F(D))

Theorem 1. Ein Gesamtsystem, bestehend aus Komponenten **ki**, kann genau dann durch ein SP Diagramm dargestellt werden, wenn der korrespondierende

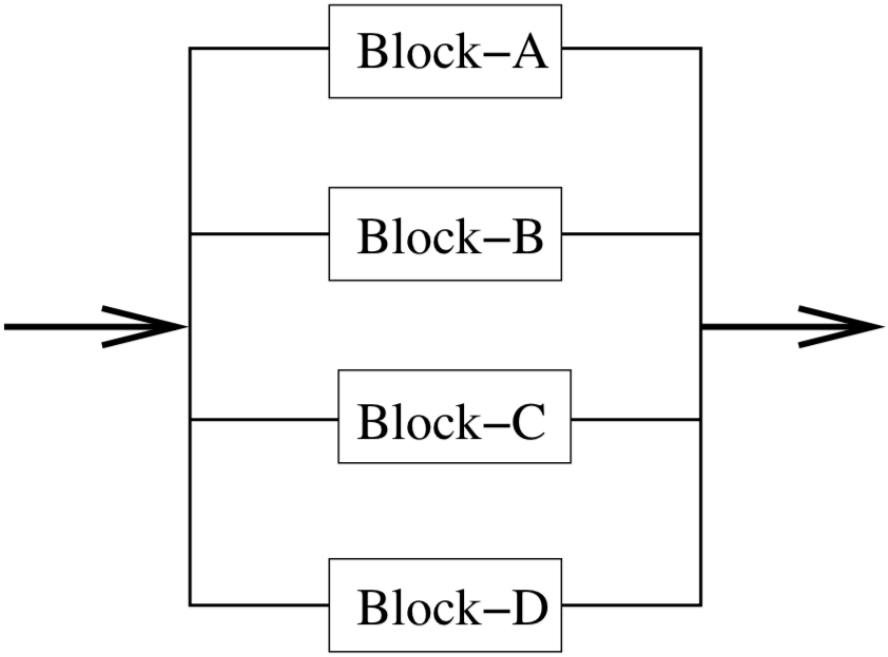

Figure 2: Diagramm paralleler Systemkomponenten

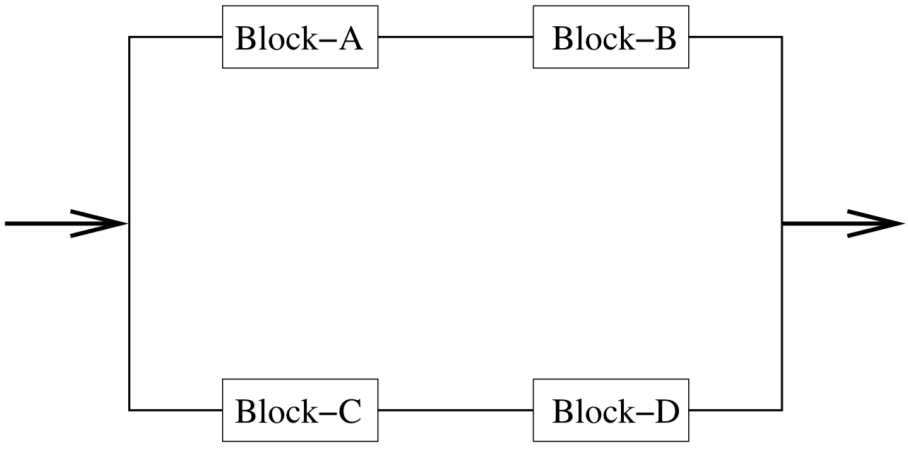

Figure 3: Serielle-Parallele Systemkomponenten

logische Ausdruck F(S), der den Fehlerstatus des Gesamtsystems ausdrückt, jedes **ki** *genau einmal* enthält.

Bewertung der Fehlertoleranz

Englisch	Deutsch	Abkürzung
Reliability	Ausfallsicherheit	R(t)
Availability	Verfügbarkeit	A(t)
Mean Time to Failure	mittlere ausfallfreie Zeit	`MTTF`
Mean Time to Repair	mittlere Ausfalldauer	`MTTR`
Mean Time Between Failures	mittlerer Ausfallabstand	`MTBF`

Definition 3 (Fehlerrate). Die zu erwartende Anzahl von auftretenden Systemfehlern (Fehlerrate), wird wie folgt berechnet:

```
= 1 / MTTF
```

Definition 4 (Reparaturrate). Die Anzahl der möglichen Reparaturen, die in einer Zeiteinheit durchgeführt werden können wird durch die Reparaturrate beschrieben. Da jede Reparatur im Durchschnitt `MTTR` Zeiteinheiten benötigt, wird die Reparaturrate mit Hilfe der mittleren Ausfalldauer wie folgt berechnet:

```
= 1 / MTTR
```

Definition 5 (mittlere Fehlerhäufigkeit). Die Häufigkeit, mit der Systemfehler auftreten können, werden von den Größen `MTTF` und `MTTR` bestimmt. Bei genauerer Betrachtung der Fehlergrößen wird klar, dass die Zeit zwischen zwei Fehlern nicht kleiner als `MTTF + MTTR` sein kann, denn während der Reparaturzeit ist das System nicht funktionsfähig. Daher wird die mittlere Fehlerhäufigkeit wie folgt berechnet:

```
= 1 / (MTTF + MTTR)
```

Mit diesen Definitionen, lässt sich nun die *Verfügbarkeit* bzw. die *Unverfügbarkeit* des Systems ausdrücken. Die stationäre Unverfügbarkeit **U** ist somit

U = `MTTR / (MTTF + MTTR)` bzw. U = `MTTR / MTBF`

Die *Unverfügbarkeit* ist das Verhältnis der durchschnittlichen Reparaturzeit und dem durchschnittlichen Zeitintervall, in dem kein Fehler auftritt. Damit lässt sich die (stationäre) Verfügbarkeit V direkt ausdrücken:

```
V = 1 - U
```

Die Verfügbarkeit beschreibt das Verhältnis der durchschnittlichen Zeit bis zum Auftreten eines Fehlers und dem durchschnittlichen Zeitintervall, in dem kein Fehler auftritt.

Kosten der Fehlertoleranz

Fehlertoleranz und Ausfallsicherheit sind auch wirtschaftstechnische Probleme (Preis-Leistungs- oder Preis-Risiko-Analyse erforderlich).

Weitere Kosten entstehen bei:

- Entwicklungskosten
- Hardware
- Zeit
- Platz
- Personal
- ungenutzte Kapazität durch Leistungsreserve für den Fehlerfall

  Für Systeme mit katastrophalen Konsequenzen durch Ausfall sind sehr hohe Kosten für die Bereitstellung redundanter Komponenten akzeptabel.

Der Aufwand für Zeit und Platz/Raum, notwendig um die Ausfallsicherheit und Verfügbarkeit zu erhöhen oder das Fehlerrisiko zu senken, lassen sich oft nur sehr schwer quantifizieren.

> Beispiel: Das zusätzliche Gewicht des Ersatzrads in einem PKW wird zu einem erhöhten Kraftstoffverbrauch beitragen

Fehlerhäufigkeit, Risiko, Daten-Verlust und -Sicherheit werden meist statistisch erfasst und verlangen daher eine Analyse (Wahrscheinlichkeitsrechnung)

Ausfallsicherheit und Verfügbarkeit

- **Ausfallsicherheit** ist ein (analytisches) vom System festgelegtes Leistungsmerkmal
- **Verfügbarkeit** ist ein vom Benutzer wahrgenommenes Leistungsmerkmal

5 x 9 Eigenschaft = 0,99999 Verfügbarkeit

%-Verfügbarkeit vs. akzeptable Ausfallzeit pro Jahr:

%-Verfügbarkeit V	maximale Ausfallzeit
99%	3 Tage, 15 Stunden, 36 Minuten
99.9%	8 Stunden, 45 Minuten
99.99%	52 Minuten, 36 Sekunden
99.999%	**5 Minuten, 15 Sekunden**
99.9999%	32 Sekunden

- 1 Jahr = 525960 Minuten
- Betriebszeit pro Jahr = V $\times$ 525960
- U = 525960 $-$ Betriebszeit

- Akzeptable Ausfallzeit = (1 − V) * Zeit

Beispiel 1

Angenommen die mittlere ausfallfreie Zeit (`MTTF`) des Netzteils eines Routers ist 18 Jahre. Um einen Fehler zu beheben kann das Netzteil entweder repariert oder ausgetauscht werden. Die Fehlersuche und Reparatur nehmen 24 Stunden in Anspruch (`MTTR_1` = 24h), während ein Austausch innerhalb von 30 Minuten ausgeführt werden kann (`MTTR_2` = 0.5h). Wie beeinflusst die Reparatur bzw. der Austausch des Netzteils die Verfügbarkeit?

`MTTF = 18Jahre = 18 × 365.25 × 24 = 157788h`

V1 = 157788 / (157788 + 24) = 0.99984792 99.985%

V2 = 157788 / (157788 + 0.5) = 0.999996831 99.99968%

Unterschiede zwischen Fehlertoleranz und Risikominimierung

Fehlertoleranz

Ist als Fehler das *Verpassen des Flugs* definiert, wird ein frühes Aufbrechen zum Flughafen die Toleranz eines solchen Fehlers nicht erhöhen. Wird ein *Stau auf der Autobahn* oder eine *Auto-Panne* als möglicher Fehler definiert, so kann die Maximierung der verfügbaren Zeit, den Flughafen rechtzeitig zu erreichen, zu einer Erhöhung der Fehlertoleranz führen. Das bedeutet, man kann den definierten Fehler bis zu einem bestimmten Grade tolerieren und trotz einer Verzögerung den Flughafen zeitgerecht erreichen. Wenn man ein System auf seine Fehlertoleranz hin untersuchen möchte, muss man zunächst dessen Ziele und Erwartungswerte genau festlegen.

Risikominimierung

Die Wahrscheinlichkeit des Auftretens eines Fehlers soll minimiert oder komplett ausgeschlossen werden. Die Sicherheit eines Systems wird durch den Designprozess bestimmt, in dem das System so konzipiert wird, dass die Mehrzahl der möglichen Fehlerquellen berücksichtigt werden. Diesbezüglich werden Parameter definiert, in deren Schranken das System fehlerfrei operieren muss. Grenzwerte für Temperatur, Feuchtigkeit, Strahlung, Druck, usw. sind Beispiele solcher Parameter. In der Informatik konzentrieren sich solche Parameter häufig auf Datenbereiche, Datenstrukturen, numerische Grenzwerte und Abweichungen, die für ein Programm oder ein System definiert werden.

- Beispiel **Black Box**: Darf bei Absturz nicht zerstört werden und muss so konzipiert werden, dass sie gegen auftretende Kräfte und mögliche Umwelteinflüsse geschützt ist.

Kurseinheit 2 - Wahrscheinlichkeit und Fehleranalyse - Theoretische Grundlagen

Da das Auftreten von Fehlern generell nicht voraussehbar ist, müssen mathematische Methoden eingesetzt werden, um das Fehlerrisiko eines Systems zu quantifizieren.

Wiederholung einfacher Kombinatorik

Fakultät

Der Ausdruck $1 \times 2 \times 3 \times \ldots \times (n-1) \times n = n!$ wird als Fakultät bezeichnet, wobei $0!$ per Definition den Wert 1 annimmt. Die Fakultät bildet eine Grundlage für das Abzählen von Permutationen, Permutationen mit Wiederholungen und Kombinationen.

Permutationen

- Eine Permutation *(P)* ist eine Anordnung von n Elementen in einer bestimmten Reihenfolge, die **alle** der n Elemente enthält.
- Eine r-Permutation (Pr) ist eine Anordnung von r Elementen aus einer Grundmenge von n Elementen.

Beispiel

Man berechne die Anzahl der Permutationen von *a, b, c, d, e, f* für je 3 Buchstaben:

6 x 5 x 4 = 120

Formel: P(n,r) = n(n−1)(n−2)…(n−r+1) = **n! / (n-r)!**

Permutation mit Wiederholung

Gebrauchtwagenhändler: Angenommen es stehen ein rotes (R), ein grünes (G) und drei blaue (B) Fahrzeuge zur Auswahl, in welcher Farbreihenfolge können diese vom Händler vorgeführt werden? Da hier nur die Farbe berücksichtigt werden, spielt die Reihenfolge der Vorführung der drei blauen Autos keine Rolle. Daher ist < G, B1, B2, R, B3 >=< G, B3, B1, R, B2 > äquivalent

Formel: P = n! / (n1! n2!…nr!)

Geordnete Stichproben mit Zurücklegen

Man ziehe k Elemente aus einem Eimer mit n Murmeln. Jedoch wird jede Murmel zurück in den Eimer gelegt bevor die nächste gezogen wird. Folglich kann jede Stichprobe n verschiedene Werte annehmen.

Formel: P = nˆk

Geordnete Stichproben ohne Zurücklegen

Für geordnete Stichproben ohne Zurücklegen wird die Anzahl der Ausgänge, wie bereits hergeleitet, durch den folgenden Ausdruck beschrieben:

P = n! / (n-r)!

Wiederholung der Binomialkoeffizienten

Der Binomialkoeffizient beschreibt die Anzahl von Auswahlmöglichkeiten und wird wie folgt berechnet:

n über r == n! / r(n-r)!

Kombinationen

Seien die Buchstaben *a*, *b*, *c* und *d* gegeben. Es sind die folgenden Kombinationen für drei Elemente möglich: (a, b, c), (a, b, d), (a, c, d), (b, c, d). Die Anzahl der Möglichkeiten drei aus vier Elementen auszuwählen wird wie folgt berechnet:

C(4,3) = 4 über 3 = 4! / 3(4-3)! = 4! / 3! = 24 / 6 = **4**

Welche Beziehung besteht zwischen Permutationen und Kombinationen?

Kombinationen	Permutationen
a-b-c	a-b-c, a-c-b, b-c-a, b-a-c, c-b-a, c-a-b
a-b-d	a-b-d, a-d-b, b-d-a, b-a-d, d-a-b, d-b-a
a-c-d	a-c-d, a-d-c, c-a-d, c-d-a, d-a-c, d-c-a
b-c-d	b-c-d, b-d-c, c-b-d, c-d-b, d-c-b, d-b-c

Kombinationen in fehlertoleranten Systemen

Die Redundanz ist nicht nur in der Duplizität von unabhängigen Systemkomponenten manifestiert, sondern auch in der Fähigkeit des Systems, aus der Menge der existierenden Komponenten auszuwählen und diese zu nutzen.

> Beispiel: ein Kommunikationsnetzwerk in dem die Kommunikationswege, die die Knoten miteinander verbindet, von verschiedenen

Kommunikationspaaren gleichzeitig genutzt werden. Um die Verfügbarkeit eines solchen Systems zu quantifizieren, muss der Grad der Redundanz berechnet werden.

Beispiel

Gegeben sei ein Gitter der Größe `n × m`. Indem man sich an den Kanten der Gitter entlang bewegt, kann man Pfade finden, die mit jedem Schritt die verbleibende Distanz zum Zielpunkt verringern. Unter der Annahme der Startpunkt sei an Position (1,1) und der Endpunkt sei an Position (n,m), wie viele kürzeste Wege existieren zwischen Start- und Zielpunkt?

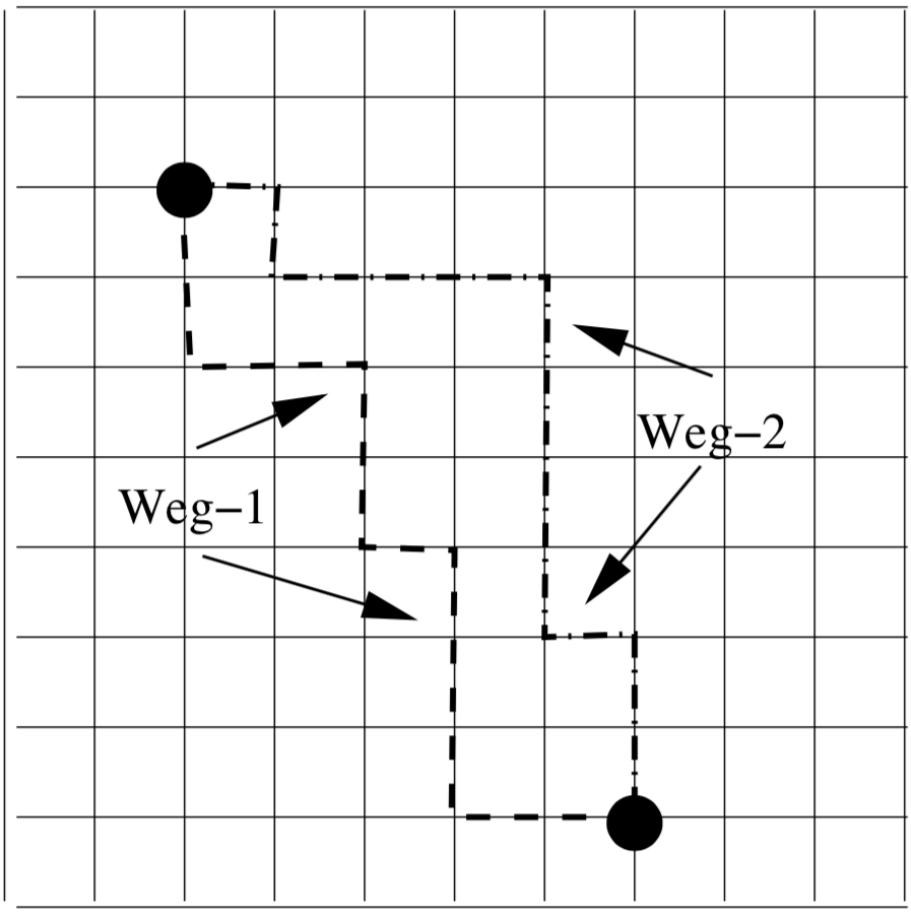

Figure 4: Kürzeste Wege in einem Gitter

In der obigen Abbildung nähert sich ein kürzester Weg dem Ziel in jedem Schritt und die Distanz zum Ziel erhöht sich niemals. Jeder der kürzesten Wege weist die gleiche Anzahl an Schritten auf, also genau so viele um die Distanzen in

X-Richtung und Y- Richtung zwischen dem Startknoten und dem Zielknoten zu überbrücken

Wahrscheinlichkeitsrechnung

Da Fehler in einem System nicht vorhersagbar/deterministisch sind, sind die Grundlagen der Wahrscheinlichkeitstheorie essentiell um die Fehlerhäufigkeit und somit das Fehlerverhalten eines System zu beschreiben.

Zufallsexperimente und Zufallsräume

Ein Zufallsexperiment ist eine Situation, in der mehr als ein Ereignis eintreffen kann und der Ausgang potentiell ungewiss ist.

- Zufallsraum Ω
- Wahrscheinlichkeitsexperiments `E`

E1 = Münzwurf, Kopf (K) oder Zahl(Z) $\Omega 1 = \{K,Z\}$

E2 = fairer Würfel $\Omega 2 = \{1,2,3,4,5,6\}$

Elementarereignisse und zusammengesetzte Ereignisse

Elementarereignisse in einem Zufallsraum stellen die feinkörnigste Partition dar, die ein Zufallsraum erlaubt. Zusammengesetzte Ereignisse bilden die Vereinigungsmenge von Elementarereignissen.

- Beispiel: Werfen eines fairen Würfels (E2)

Die Elementarereignisse sind 1, 2, 3, 4, 5 und 6. Die Ereignisse gerade = {2, 4, 6} und ungerade = {1, 3, 5} sind Beispiele für zusammengesetzte Ereignisse.

Ereignis	relative Häufigkeit
6	1/6
5	1/6
4	1/6
3	1/6
2	1/6
1	1/6
>3	3/6
gerade	3/6
ungerade	3/6
gerade (> 3)	4/6
gerade ungerade	1

Axiome und Konzepte der Wahrscheinlichkeitsrechnung

Für jedes Ereignis A in Ω treffen die folgenden Axiome der relativen Häufigkeit r(A) zu

1. **0 r(A) 1**
2. **r(Ω) = 1**
3. **Falls A B = , dann gilt r(A B) = r(A)+r(B)**

Gegeben sei ein Zufallsexperiment *E* mit dem Zufallsraum *Ω* und den Ereignissen *Ai*. Die Wahrscheinlichkeiten *Pr(Ai)* der Ereignisse genügen den folgenden drei Axiomen:

1. Pr(Ai) 0
2. Pr(Ω) = 1
3. Falls Ai Aj = , dann gilt r(Ai Aj) = Pr(Ai)+Pr(Aj)

Fehlererkennung und Fehlerkorrektur

Bitfehler können während des Ablaufs eines Programms in der CPU auftreten oder bei der Übertragung von Daten zu anderen Systemkomponenten durch das Übertragungssystem eingeführt werden. Eine andere Fehlerquelle ist die Langzeitarchivierung von Daten auf Magnetbändern oder Platten. Während den Bitfehlern in einem Computer meist ein Defekt in der Hardware zugrunde liegt, sind es Umwelteinflüsse wie Strahlung, Temperatur und magnetische Induktion die das Fehlverhalten eines Datenübertragungssystems und eines Langzeitspeichers beeinflussen

Zwei Arten:

- einfache Bitfehler
- Fehlerfolgen (Bursts)

Im Gebiet der Datenübertragung werden fehlerbehaftete Datenpakete generell eliminiert (und nicht versucht zu korrigieren, da der Overhead zu groß ist).

Fehlertoleranz durch adäquate Kodierung

Zusätzliche Bits, die zur Fehlererkennung oder Korrektur von Fehlern dienen, werden als Redundanzbits bezeichnet, da sie nicht zur Speicherung oder Übertragung der eigentlichen Information dienen.

Ein Code ist eine Menge *P* von Bitmustern, mit denen die zu repräsentierenden Informationsworte dargestellt werden können.

Paritätsbits und Paritätscodierung

Die Parität ist die Information darüber, ob ein bestimmtes Codewort eine gerade oder ungerade Anzahl an Bitstellen besitzt, die eine logische 1 enthalten. Ein Paritätsbit wird dem Datenwort hinzugefügt, welches den Wert 1 oder 0 enthält, je nachdem wie viele Datenbits den Wert 1 aufweisen und welche Parität man anwendet.

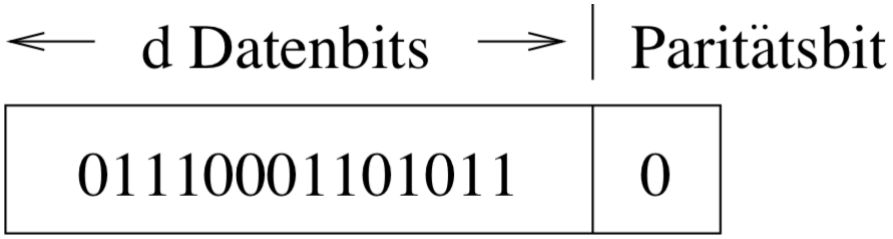

Figure 5: Datenbits mit gerader Parität

Das Paritätsbit lässt sich für das gerade Paritätsprinzip durch XOR-Verknüpfung aller Datenbits berechnen. Es ist nur möglich, einen *Fehler zu erkennen*, nicht zu beheben (da man nicht weiss, welches Bit geflippt ist). Flippen mehrere Bits (gerade Anzahl), kann dieser Fehler nicht erkannt werden.

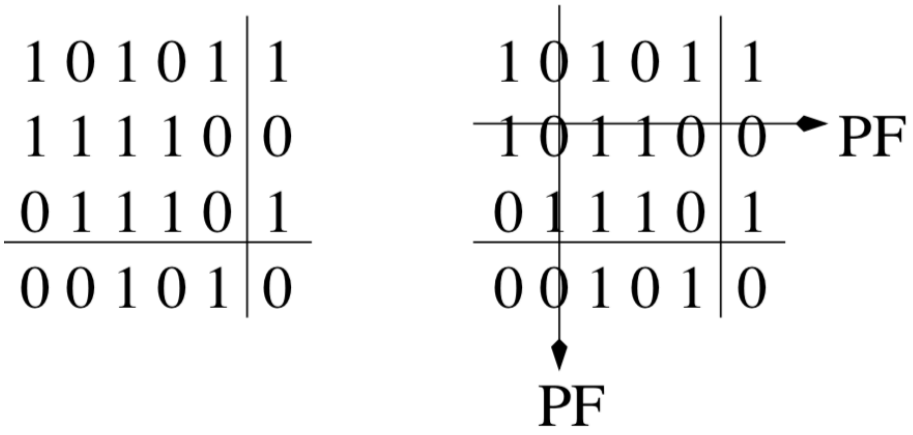

Figure 6: Fehlerkorrektur durch 2-dimensionale Parität

Mehrere Codeworte werden zu einem Datenblock zusammengefasst, die Parität jeder Spalte berechnet und in einem Paritätswort dem Datenblock hinzugefügt. Ein aufgetretener Bitfehler ändert somit die Parität des jeweiligen Datenwortes (Querparität, QP) und die Parität der jeweiligen Spalte des Datenblocks (Längsparität, LP). Allerdings kann es auch zu Bitflips bei den Paritätsbits kommen!

Redundanz

Codeeffizienz:

1. Je *kürzer* das zu kodierende Datenwort ist, desto höher ist die Anzahl der speicherplatzbenötigenden P-Bits.
2. Je *länger* das zu kodierende Datenwort ist, desto höher ist die Wahrscheinlichkeit von Bitfehlern.

Hammingdistanz und Hammingcodierung

Der Hammingabstand zwischen zwei Codeworten ist die Anzahl der Bitstellen, an denen sich zwei Codeworte unterscheiden

Theorem 5. Um k Fehler zu erkennen, muss der Hammingabstand des gesamten Codes $k + 1$ betragen, zur Korrektur von k Fehlern, muss er $2k + 1$ betragen.

Man betrachte m Datenbitstellen, r Redundante Bits und somit n = m + r Bit Codeworte. Damit können insgesamt 2^n Codeworte dargestellt werden.

$(m + r + 1)$ ist eine untere Grenze für die Anzahl der redundanten Stellen, die benötigt werden, um einen Code mit m Nachrichtenbitstellen, der die Korrektur eines einfachen Bitfehlers erlaubt, zu erstellen.

Hamming-Code

Grundgedanke: Verteile die Prüfbits so, dass jedes Datenbit in mindestens zwei Prüfbits berücksichtigt wird.

- Jedes Bit im Codewort erhält eine fortlaufende Nummer beginnend bei 1. Diese Positionsnummern werden binär dargestellt. Das Codewort besteht zunächst aus dem Quellwort.
- Das *k-te* Paritätsbit wird an der Position $2k$ eingeordnet. Die binäre Darstellung dieser Position hat (von rechts gesehen) das *k-te* Bit gesetzt, alle anderen Bits haben den Wert 0.
- Also:
  - 00 .. 0001 = Position 1
  - 00 .. 0010 = Position 2
  - 00 .. 0100 = Position 3
  - 00 .. 1000 = Position 4
- Der Wert des *k-ten* Paritätsbits ergibt sich aus der Quersumme aller Bits des Codeworts, deren Position in der binären Darstellung das *k-te* Bit gesetzt haben.
- Jedes Datenbit wird folglich über zwei Paritätsbits abgedeckt.

Das Codewort (Datenwort + Paritätsbits) wird übertragen und beim Empfänger nach dem selben Verfahren decodiert.

1. Ergibt sich bei der Prüfung kein Fehler, wurde das Codewort ohne (1-Bit-)Fehler übertragen.
2. Nur **ein** Paritätsbit passt nicht: Das Datenwort wurde korrekt übertragen und ein Paritätsbit ist geflippt. (Egal, da Datenwort korrekt)
3. **Mehrere** Paritätsbits passen nicht: Ein Datenbit ist verfälscht. Summe der inkonsistenten Paritätsbits ist die Position des verfälschten Datenbits.

Beispiel

Für m = 7 (z.B., ASCII) erhalten wir (7 + r + 1) 2r Das kleinste r, für das diese Bedingung erfüllt, ist r = 4

Datenübertragung

zwei Arten der Fehlerüberwachung:

1. vorwärts gerichtete Fehlerüberwachung (Forward Error Control): Empfänger kann Fehler erkennen und korrigieren
2. rückwärts gerichtete Fehlerüberwachung (Backward Error Control): Erneutes Übertragen ist erforderlich (z.B. Ausbleiben Empfangsbestätigung)

Die meisten Datenübertragungsprotokolle eine rückwärtsgerichtete Fehlerkontrolle. (weniger En-/Decodierung Overhead) Fehler wird erkannt und Datenpaket eliminiert, bevor es weitergeleitet wird.

Zyklische Redundanzprüfung - CRC

G(x) kann generell frei gewählt werden solange die folgenden Bedingungen erfüllt sind:

- Sowohl hochwertige und niedrigwertige Stellen (high- and low-order bits) von G(x) müssen eine 1 aufweisen.
- Um die Prüfsumme für einen Rahmen mit m Bits zu berechnen, die dem Polynom M(x) entspricht, muss es länger als G(x) sein.

Drei Polynome wurden zu internationalen Standards: CRC-12, CRC-16, CRC-CCITT.

CRC	G(x)
CRC-8	x8 +x2 +x1 +1
CRC-10	x10 +x9 +x5 +x4 +x1 +1
CRC-12	x12 +x11 +x3 +x2 +x1 +1
CRC-16	x16 +x15 +x2 +1
CRC-CCITT	x16 +x12 +x5 +1
CRC-32	x32 +x26 +x23 +x22 +x16 +x12 +x11 +x10 +x8+ +x7 +x5 +x4 +x2 +x+1

Zusammenfassung

- Die Datenbits D werden als Binärzahl betrachtet;
- Man wähle ein (r+1) Bitpattern (Generatorpolynom) G;
- Ziel: Wähle CRC Bits, R, so dass < D, R > genau durch G(modulo 2) teilbar ist;
- Der Empfänger kennt G, teilt < D, R > durch G. Falls ein Rest entsteht wurde ein Fehler erkannt;
- Es können somit alle Burstfehler mit weniger als (r) Bits erkannt werden;

Figure 7: Mathematische Formel: D × 2 R

Datenspeicherung (RAID)

RAID ist die Abkürzung für *Redundant Array of Inexpensive Disks*, also ein redundantes Array kostengünstiger Plattenspeicher (später wurde aus inexpensive independent).

Striping (RAID 0)

In dieser Konfiguration simuliert das RAID einen alleinstehenden Plattenspeicher, dessen Speicherkapazität in Streifen (engl. stripes) mit je **k** Sektoren unterteilt ist.

Während RAID-0 eine Lösung für das Problem des großen Speicherbedarfs darstellt, hilft es **nicht**, die Fehlertoleranz des Plattenspeichers zu erhöhen:

- keine Erhöhung der Redundanz
- Erhöhung der Fehlerwahrscheinlichkeit aufgrund individueller Fehlerraten der einzelnen Laufwerke

Beispiel:

Eine Festplatte hat eine `MTTF` von 20000h. 4 Festplatten im RAID-Verbund haben eine `MTTF` von (20000h / 4) = 5000h.

> Bei RAID0 mit 4 Platten geht alle 5000h *eine* Platte kaputt und **alle** Daten sind weg.

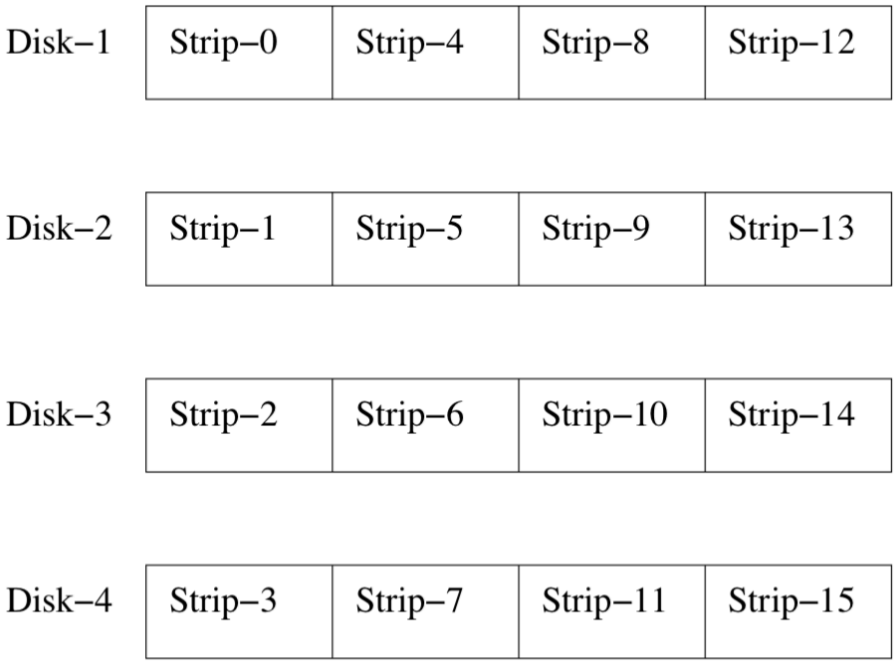

Figure 8: Raid 0

Redundanz mit RAID-1

Duplizieren von Daten auf zusätzliche Plattenlaufwerke

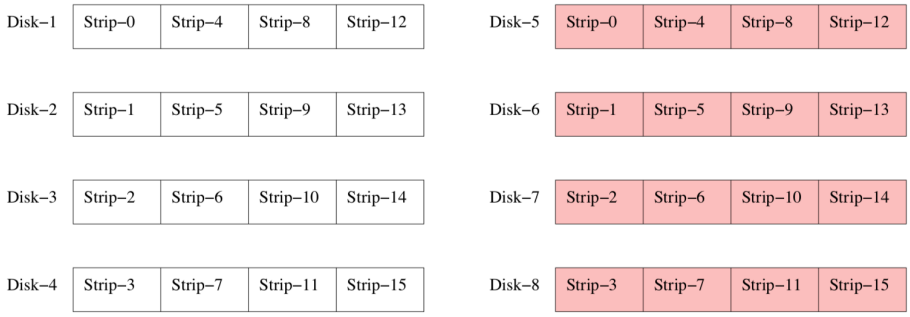

Figure 9: Raid 1

RAID-1 hat den Vorteil, dass das Versagen eines Laufwerkes nur eine Kopie der Daten zerstört und man die Möglichkeit hat, das defekte Laufwerk zu reparieren oder auszutauschen. Die defekte Komponente muss sofort getauscht werden, um die Redundanz wieder herzustellen.

Beispiel:

- Austauschzeit def. Platte: 1h (`MTTR`)
- Verfügbarkeit:
  - pro Komponente: `MTTF` / (`MTTF` + `MTTR`) = 5000 / (5000 + 1) = 99,98%
  - Im Parallelbetrieb also: ‘1 - (1 - 0,9998) * (1 - 0,9998) = 1 - (1 - 0,9998)^2 = 0,99999996 = 99,99996 %

Andere RAID-Konfigurationen

Um eine bessere Leistung des Plattenspeichers zu erzielen, speichert man die Daten in Worten oder Bytes anstatt in Streifen mit Sektoren.

RAID-2

Daten in 4-Bit nibbles, oder halbe Bytes zerlegen und den jeweiligen *Hamming-Code* generieren. Somit werden die 4-Bit Daten in ein 7-Bit Wort umgewandelt, welches auf 7 verschiedenen Laufwerken abgelegt werden kann. Wenn man nun die 7 Laufwerke so synchronisiert, dass die zu einem Wort gehörenden Daten auf gleichen Zylindern liegen, können 7-Bit Bitmuster parallel geschrieben und gelesen werden.

Nachteile:

- Laufwerke müssen synchronisiert sein, um die parallelen Eigenschaften des RAID-2 auszunutzen und den Durchsatz zu erhöhen
- Konfiguration mit 32 Datenlaufwerken und sechs Paritätslaufwerken eine effektive Datennutzung von nur 81%

RAID-3

Wie RAID-2, aber Paritätsbits nur einem Laufwerk zugeordnet. Mit diesem Ansatz reduziert man natürlich den Grad der Redundanz und somit ist der Overhead reduziert.

RAID-4 und RAID-5

Stellen eine Erweiterung der RAID-0 und RAID-1 Konfigurationen dar, wobei nicht jedes Datenwort mit einem Paritätsbit abgesichert wird, wie in RAID-2 und RAID-3.

In der RAID-4 Konfiguration muss auf das Paritätslaufwerk jedes mal zugegriffen werden, wenn Daten geschrieben werden. Somit kann dieses Laufwerk zu einem Flaschenhals werden, das den Durchsatz (=Effizienz) des System verringert

RAID-5 wirkt diesem durch die gleichmäßige Verteilung der Paritätsstreifen über die verfügbaren Laufwerke entgegen.

Die Datenspeicherung mit RAID-Konfigurationen ist ein Beispiel für den Kompromiss, den man treffen muss, wenn Fehlertoleranz und Performance im System berücksichtigt werden müssen.

Kurseinheit 4 - Protokollbasierte Fehlertoleranz

In Kommunikationsnetzen und verteilten Systemen ist das Auftreten von Fehlern häufig nicht mit dem Ausfall einer Systemkomponente verbunden, sondern hängt von der momentanen Nutzlast und der Verfügbarkeit notwendiger Systemressourcen ab.

Ein fehlertolerantes System muss so konzipiert werden, dass ein temporäres Fehlverhalten aufgrund fluktuierender Lastverteilung und Ressourcenverfügbarkeit transparent ist und nur durch Variationen in der jeweiligen Antwortzeit, nicht aber durch die Korrektheit des Resultats wahrnehmbar ist.

Das 2 Armeen Problem

Kriegssitutation:

Armee **A** (Bataillone **A1** und **A2**) kesseln Armee **B** ein.

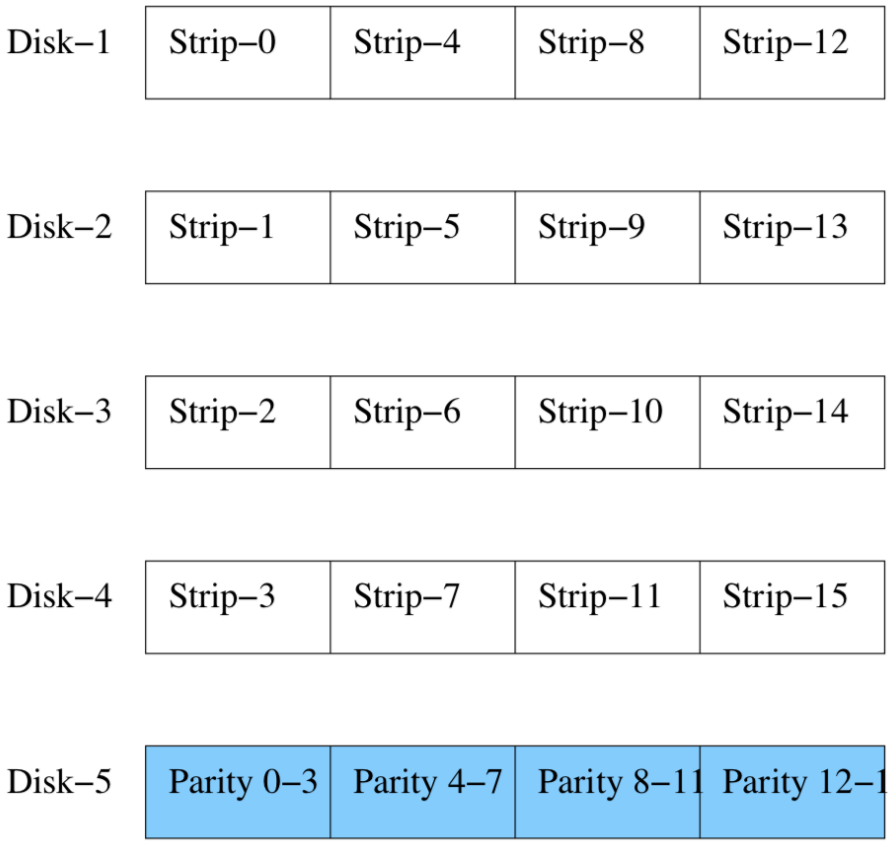

Figure 10: Raid 4

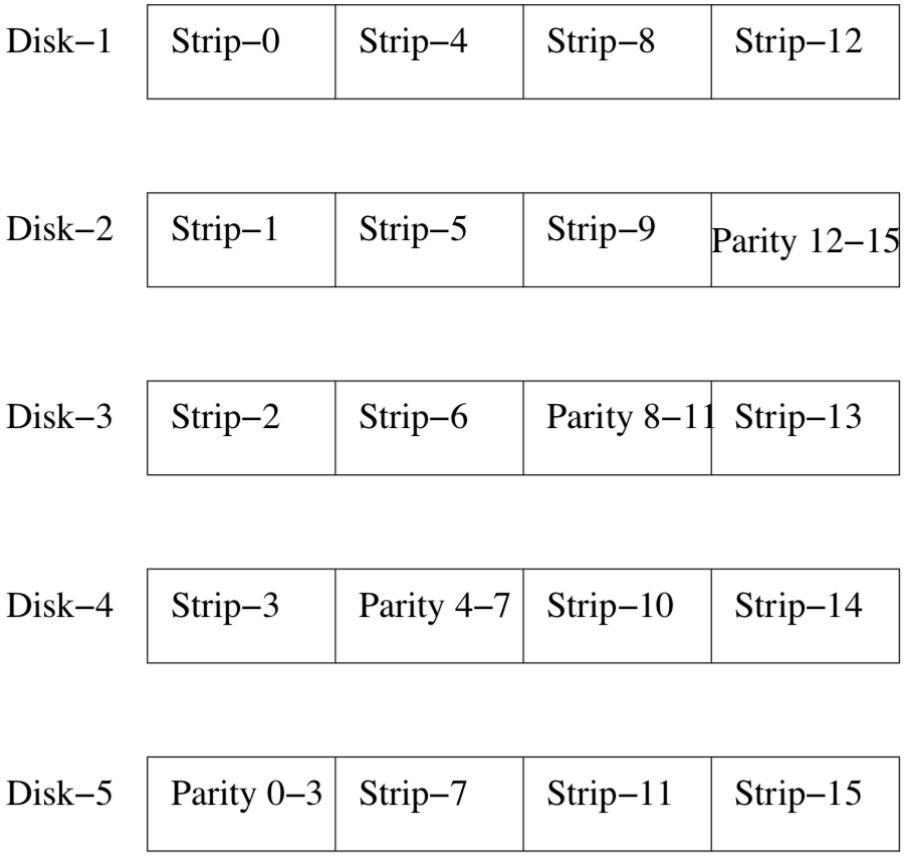

Figure 11: Raid 5

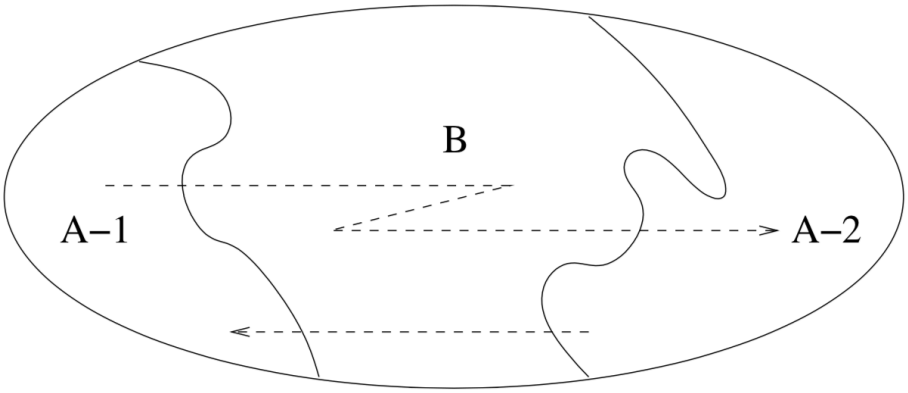

Figure 12: Das 2-Armeen Problem

- **A** kann nur gewinnen, wenn sich **A1** und **A2** synchronisieren und zeitgleich (beidseitig) angreifen.
- sonst gewinnt **B**

Kommandanten der Bataillone müssen kommunizieren, um sich auf präzise Angriffszeit zu einigen. Die einzige Möglichkeit der Kommunikation ist die Absendung eines Boten durch das von **B** besetzte Gebiet.

Man nehme an, dass **A1** zum Zeitpunkt `t1` eine Nachricht `M` der Form zu **A2** sendet

Für dieses Problem existiert **keine** Lösung. Es existieren folgende Fälle:

1. Der von **A1** gesendete Bote und somit `M1` erreicht **A2**: **A2** weiß, dass **A1** plant, am Samstag um 5:30 anzugreifen, **A1** kann allerdings nicht sicher sein, dass `M1` von **A2** erfolgreich empfangen wurde. **A2** ist sich dessen bewusst und wird daher nicht angreifen. (Angriff nur, wenn beide Bataillone sich **sicher** sind, dass das jeweilige andere Bataillon auch angreift).
2. Der von **A1** gesandte Bote wird von **B** gefangen genommen - `M1` erreicht niemals **A2**: **A1** kann diesen Fall nicht vom ersten Fall unterscheiden und wird somit nicht angreifen.
3. Der von **A1** gesandte Bote und somit `M1` erreicht **A2** später als der in `M1` geplanten Angriffszeit: In diesem Fall wird **A2** niemals angreifen, da die Koordinationszeit vergangen ist. **A1** kann diesen Fall nicht von den ersten beiden Fällen unterscheiden und wird nicht angreifen.

Da **A1** nur angreifen darf, wenn es sicher ist, dass **A2** zur gleichen Zeit angreift, muss im Fall 1 **A2** eine Empfangsbestätigung `M2` der Form per Bote zu **A1** senden. `M2` unterliegt den gleichen Bedingungen wie `M1`, somit ändert sich keine der obigen Fälle.

Abstraktion eines Kommunikationskanals wobei **A1** und **A2** die Kommunikationspartner darstellen. Armee **B** repräsentiert das Kommunikationsmedium mit möglichen Fehlerquellen, die die Daten während der Übertragung zerstören oder modifizieren können. Anhand des *2 Armeen Problems* kann man verschiedene Arten von Fehlern definieren, die von verschiedenen Protokollelementen abgedeckt werden.

Fehlertoleranz in den OSI-Schichten

Schicht 1 (physical layer)

Schicht 1, die Bitübertragungsschicht, bezieht sich auf die technischen Eigenschaften des Kommunikationssystems. Für eine bestimmte Klasse von Netzwerken (z.B. Ethernet oder Tokenring) werden die technischen und physikalischen Eigenschaften in der ersten Schicht festgelegt und standardisiert.

Die Spezifikationen bestimmter Signalkodierungsverfahren sind Beispiele von Designkomponenten, die die Fehlertoleranz des Übertragungssystems beeinflussen können

Beispiel:

Durch physikalische und elektrische Einflüsse verändern sich die Signalform und die Signalstärke

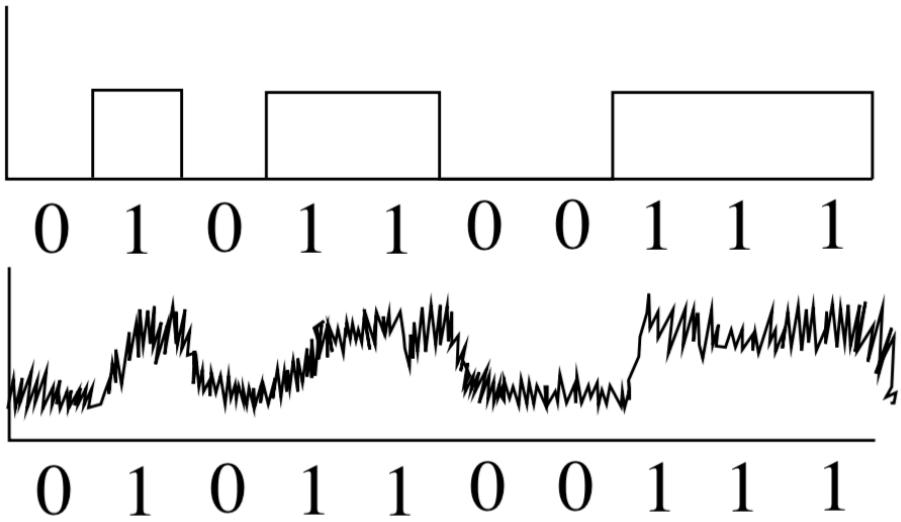

Figure 13: Signale werden während der Übertragung degeneriert und verzerrt

- Sender **S** und Empfänger **E** müssen sich synchronisieren -> Kommunikation wird durch spezielles Bitmusters eingeleitet.
- Sync erfolgt über einen Takt, welcher **S** und **E** selber generieren.
- Während der Kommunikation ist auch fortlaufende Synchronisierung notwendig, da **E** und **S** nicht exakt den gleichen Timer haben.
- Das Signal soll immer in der Mitte eines Takts abgegriffen werden

Spezifikation der Datensignalkodierung hat Einfluss auf die Fehlertoleranz des Übertragungssystems.

- Nonreturn-to-Zero Level (NRZ-L)
- Nonreturn-to-Zero Inverted (NRZI)
- Manchaster Kodierung
- Differential Manchaster

Schicht 2 (data link layer)

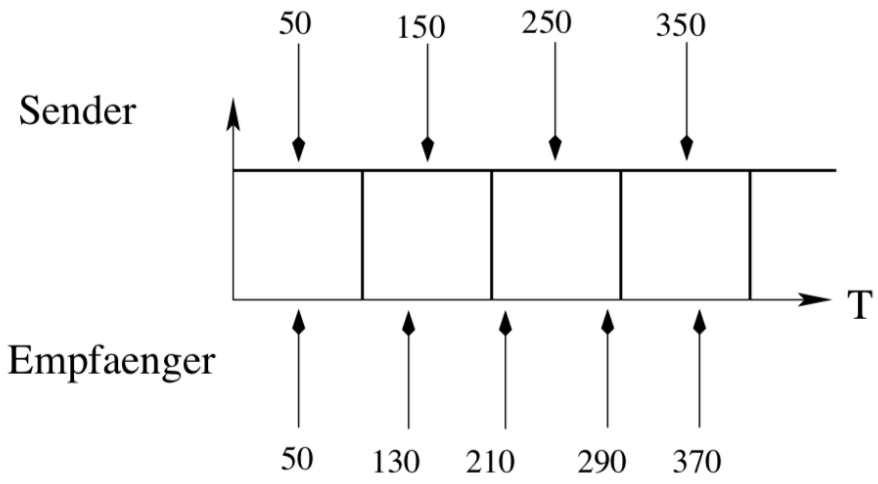

Figure 14: Drifting - Abweichung des Empfangstakts vom Sendetakt

Stop-and-Wait Protokoll

Ziel des Stop-and-Wait Protokolls ist, Sender und Empfänger so zu koordinieren, dass der Sender nur einen neuen Frame sendet wenn garantiert ist, dass der vorherige Frame korrekt empfangen wurde, um somit eine fehlertolerante Datenübertragung zu realisieren.

naive Implementierung:

- Empfänger sendet nach Empfang ein `ACK`. Stellt der Empfänger einen Bitfehler fest, sendet er `NACK`
- Sender startet beim Absendung einen Timer und wiederholt die Übertragung, wenn kein `ACK` kommt.
- Problem: Sendet der Empfänger ein `ACK` und der Timer ist gerade abgelaufen, sendet der Sender erneut, bevor das `ACK` bei ihm eintrifft. Der Empfänger weiss nichts mehr von dem vorher empfangenen Paket und würde den Rahmen in den Protokollstack hochreichen.
  - Lösung: jeder Datenübertragungseinheit wird eine Sequenznummer zugewiesen wird, damit der Empfänger Reihenfolge bilden kann.
  - Auch die `ACK`s bekommen Sequenznummer, damit der Sender diese zuordnen kann.

Algorithmus

Sender:

1. Initialisiere eine Integer-Variable SN = 0
2. Warte auf Daten von den höheren Schichten

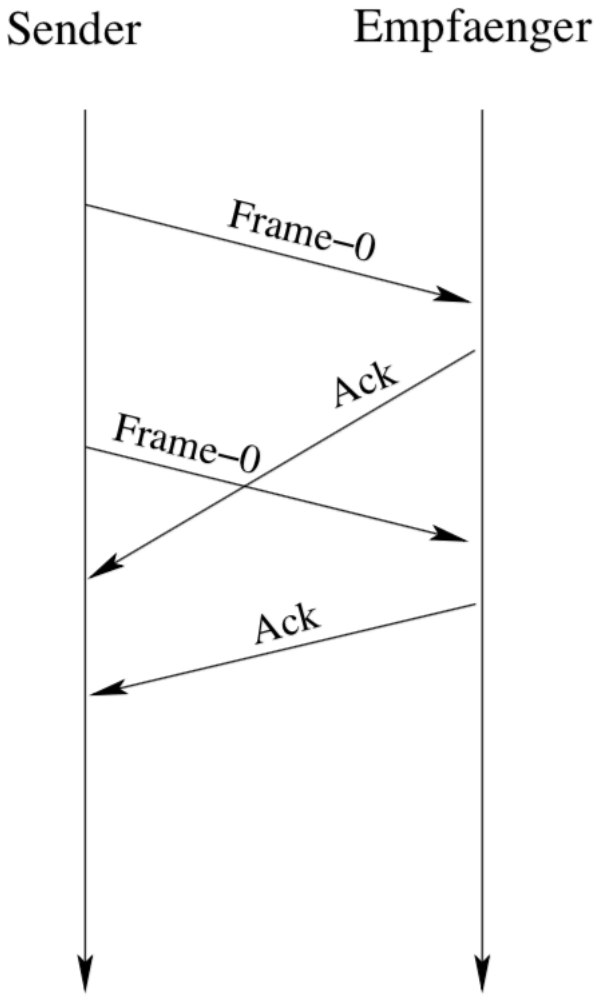

Figure 15: Verzögerung in der Übertragung des ACK

3. Wenn Daten verfügbar sind, weise der Datenübertragungseinheit (Frame) die Nummer SN zu
4. Übertrage den Frame SN mit SN in einem speziellen SeqNr. Feld
5. Wenn ein `ACK` Frame mit Anforderungsnummer RN fehlerfrei eintrifft setze SN = RN und wiederhole Schritt 2; Wenn kein fehlerfreier `ACK` Frame innerhalb der gesetzten Zeitgrenze eintrifft oder ein `NACK` empfangen wird, wiederhole Schritt 4;

Empfänger:

1. Initialisiere eine Integer-Variable RN = 0;
2. Wenn ein fehlerfreier Frame mit SN = RN (vom Sender) empfangen wird, leite den Frame an die höheren Protokollschichten weiter.
3. Setze RN = RN + 1;
4. Zu einem beliebigen Zeitpunkt (aber im Rahmen eine definierten Zeitintervalls), sende ein `ACK` Frame mit RN in einem speziellen SeqNr. Feld zu A und wiederhole Schritt 2.

   Garantiert die fehlertolerante Kommunikation zwischen zwei Kommunikationspartnern, die direkt über ein gemeinsames Medium kommunizieren

Schicht 3 (network layer)

Routing: Kommunikationspfad zwischen zwei Kommunikationspartnern (nicht direkt über selbes Medium verbunden).

Bester Pfad: kürzester Weg basierend auf

- Anzahl der notwendigen Übertragungsschritte oder
- minimaler Latenzzeit

2 Verfahren:

- Globale Routingalgorithmen (Link State Routing) -> Dijkstra
- Dezentralisierte (Verteilte) Routingalgorithmen -> Bellmann-Ford

Schicht 4 (transport layer)

Stellt Protokolle zur fehlerfreien Kommunikation zwischen zwei **Prozessen** bereit. Die Protokolle habe kein Kontrolle darüber, durch welche Teilnetze und welche Netzwerkarchitekturen die einzelnen Kommunikationseinheiten auf ihrem Weg von Sende- zu Empfangsprozess gesendet werden.

Übertragungszeit abhängig von:

- Last im Netzwerk (Buffer)
- Welchen Pfad die Pakete nehmen
- Ausfall von Stationen
- etc.

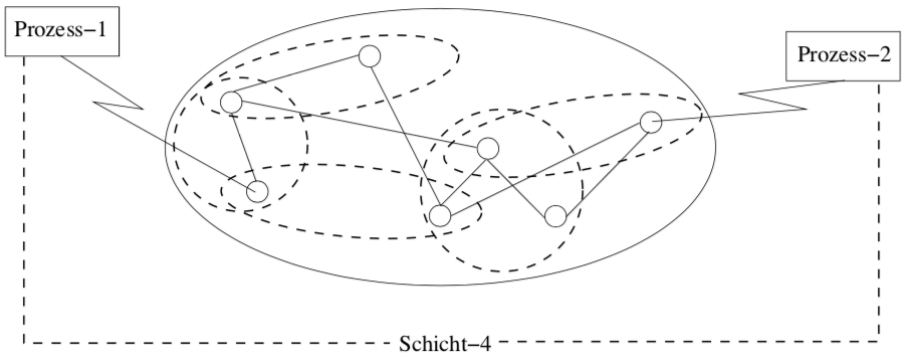

Figure 16: Prozessorientierte Protokolle der 4. Schicht

Verbindungsaufbau über SYN/`ACK`. ein Datenpaket muss wiederholt übertragen werden, wenn keine Empfangsbestätigung (`ACK`) innerhalb eines Zeitintervalls `T_ack` beim Sender eintrifft. `T_ack` muss sich allerdings ständig den Gegebenheiten des Netzwerks anpassen:

- Angenommen ein Paket wurde zur Zeit `t0` gesendet, dann erwartet der Sender eine Empfangsbestätigung bis zur Zeit `t0` + `T_ack`.
- Ist `T_ack` zu klein wird eine kleine Verzögerung in der Übertragung des Pakets oder des `ACK`s eine wiederholte Datenübertragung initiieren(unnötigerweise wird Netzwerklast erhöht und Bandbreite verschwendet).
- Ist `T_ack` zu groß, muss der Sender bis zum Timeout `t0` + `T_ack` warten, bis ein fehlerbehaftetes oder verlorenes Paket wiederholt übertragen werden kann. Somit wird der Sendevorgang unnötigerweise verzögert.

`T_ack` muss daher *fortlaufend* abgeschätzt werden, indem man die Zeit eines Roundtrips (RTT), also das Zeitintervall zwischen dem Senden des Paketes und dem Empfang eines `ACK`, misst und in die Schätzung miteinbezieht.

Schicht 5-7

Diese Protokollelemente sind weitgehend unabhängig von dem realen Netzwerk, dass durch die Schichten 1 - 4 definiert sind und beziehen sich auf die Kooperation verteilter Prozesse, die das verteilte System darstellen.

Protokollelemente:

- Computersicherheit (Security): Methoden, die den Datenaustausch der Prozesse durch kryptografische Methoden schützen.
- Zeitsynchronisation: Methoden, mit denen sichergestellt werden kann, dass die Zeit aller im verteilten System partizipierenden Prozesse nicht

mehr als voneinander abweichen. Dadurch kann eine geordnete Abhandlung individueller Transaktionen durch verteilte Prozesse stattfinden
- Datensynchronisation: Methoden, die gewährleisten, dass duplizierte Daten synchronisiert sind und somit den gleichen Wert aufweisen. Dazu gehören die verschiedenen Commit-Protokolle, die in verteilten Datenbanksystemen ihre Anwendung finden.
- Agreement-Protokolle: Diese Klasse von Protokollen soll garantieren, dass Prozesse in einem verteilten System kooperieren und ihre Aktionen synchronisieren. Das Auftreten einer limitierten Anzahl von Systemfehlern darf dabei diese Kooperation nicht negativ beeinflussen.

Agreement-Protokolle nehmen in der Fehlertoleranz eine besondere Stellung ein, da sie häufig die Grundlage einer Mehrheitsentscheidung formen, die es den Prozessen erlaubt trotz Fehlern ihre Aktionen zu koordinieren:

Das Problem der byzantinischen Generäle

- Eine byzantinische Armee logiert außerhalb einer feindlichen Stadt
- Eine Armee aus mehreren Divisionen, die je von einem eigenen General angeführt werden
- Die Kommunikation findet ausschließlich durch Boten statt.
- nicht alle Generäle sind loyal, sondern es befinden sich Verräter unter ihnen.
- Ungeachtet der Handlungen der Verräter soll stets für ein gemeinsames, sinnvolles Vorhaben entschieden werden
- Handlungsvorhaben, für die sich ein General entscheiden kann: **Angriff** oder **Rückzug**

Bedingungen für sinnvolle/konsistente Handlungen:

- Alle loyalen Generäle entscheiden sich für den selben Plan
- Eine geringe Anzahl von Verrätern kann nicht dazu führen, dass die loyalen Generäle einen schlechten Plan annehmen.

Weitere Annahmen:

- Jede gesendete Nachricht, wird korrekt zugestellt
- Der Empfänger einer Nachricht weiß, wer der Absender war
- Das Fehlen einer Nachricht kann festgestellt werden

Problem

Fall 1:

Fall 2:

Leutnant 1 kann die Fälle 1 und 2 nicht voneinander unterscheiden, da er in beiden Fällen unterschiedliche Befehle bekommt.

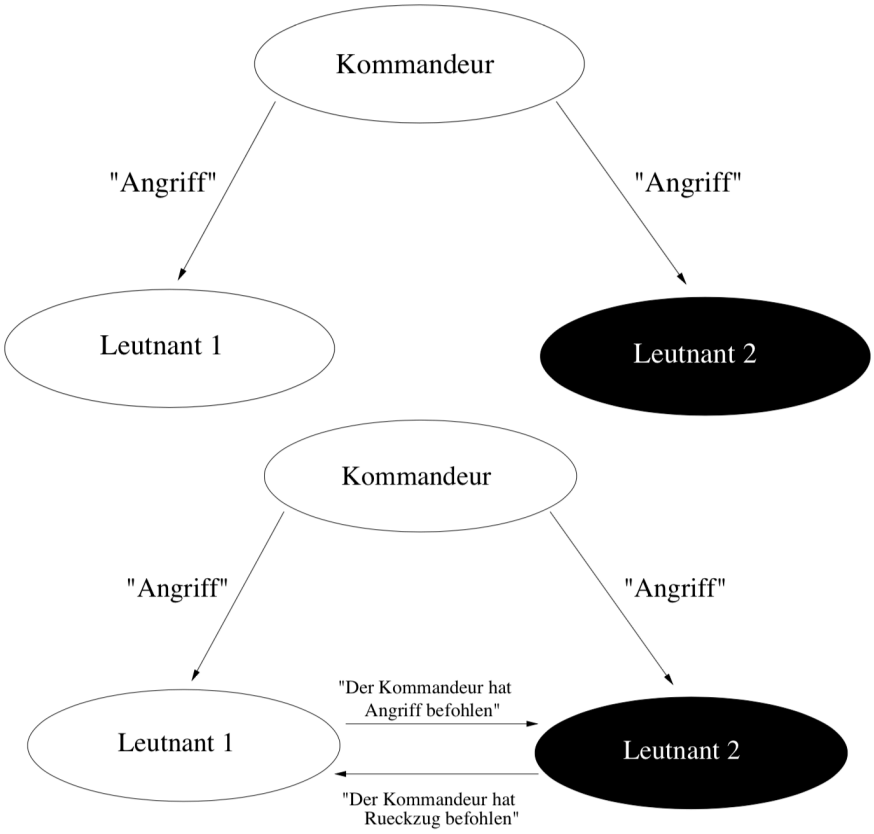

Figure 17: Leutnant 2 ist ein Verräter

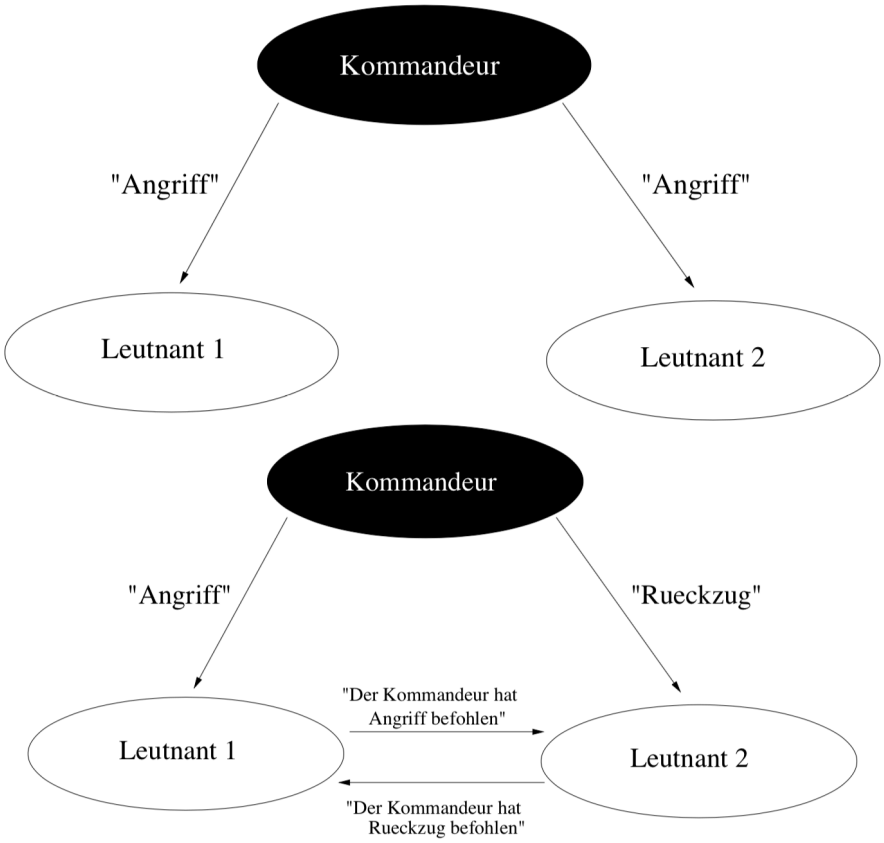

Figure 18: Der Kommandeur ist ein Verräter

Lösung

Das Problem der byzantinischen Generäle kann nur für `3m + 1` Generäle in der Anwesenheit von höchstens `m` Verrätern gelöst werden

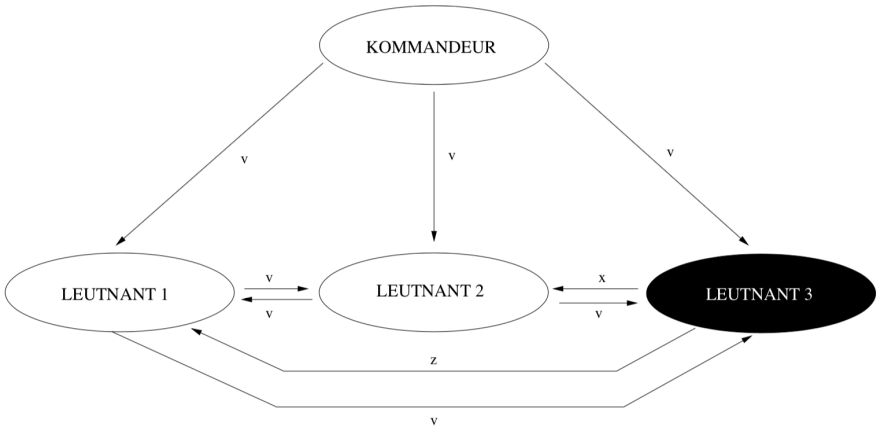

Figure 19: Beispiel für vier Generäle, wovon ein Leutnant nicht loyal ist