Vorlesung Real-Time Systems

6/3. Zeit und Synchronisation

Übersicht

1. Was ist Zeit
2. Was beeinflusst die Zeit
3. Auswirkungen
4. Uhrensynchronisation
5. Literatur / Quellen

1. Was ist Zeit?

Zeit ist eine physikalische Größe, gemessen in Sekunde.

Sie ist – entgegen dem klassischen Weltbild – nicht konstant, sondern abhängig vom Bezugssystem und der „Umgebung“.

Konstant hingegen ist die Lichtgeschwindigkeit c_0 – unabhängig vom Bezugssystem.

Gemessen wir die Zeit mit so genannten Atomuhren, in denen der periodische Übergang zwischen zwei Energieniveaus des Cs-133 Atoms als Zeitnormal verwendet wird. Jedoch ist auch hier die theoretisch erreichbare Genauigkeit durch die *Heisenbergesche Unschärferelation* auf 10^{-15} begrenzt.
Praktische Genauigkeit: Δt ~1µs / a

2. Was beeinflusst die „Zeit“

Im klassischen Sinn ist die Zeit konstant. Dies gilt so lange, wie die Geschwindigkeit v gegeneinander bewegter Uhren sehr klein gegen c ist, also v << c

Beispiele: Schiff, Auto, Zug, Flugzeug, …

Bei großen Geschwindigkeiten v (SRT) gilt für die Zeiträume Δt einer bewegten und einer ruhenden Uhr

$$\Delta t_b = \sqrt{1 - \frac{v^2}{c^2}} \cdot \Delta t_r = \sqrt{1 - \beta^2} \cdot \Delta t_r$$

mit $\beta = \frac{v}{c}$

3. Welche Auswirkungen haben Ungenauigkeiten

Weg der Sonne über den Äquator in 1 s:

U = 40000 km, T = 24 h = 60 * 60 * 24 s

v = s / t ←→ s = v * t, v = 40000 km / 86400 s

1 Sekunde Fehler bei der Navigation: Ortsfehler = 463 m

GPS-Satelliten [Zöb08, S. 68 f]:

- Uhren gehen langsamer weil Satelliten in 20200 km Höhe 3874 m/s schnell sind (spezielle Relativitätstheorie).
- Uhren auf der Erde gehen langsamer wg. Höherer Gravitation (allgemeine Relativitätstheorie
- ➔ Uhren in Satelliten um $4.445 \cdot 10^{-10}$ verzögern, sonst mehrere hundert Meter Positionsfehler nach einer Stunde.

Flug einmal rund um die Erde: $\Delta t \sim 100$ ns (1971 Hafele, Keating)

Erddrehung durch Gezeiten gebremst:

Zwischen 1900 und 1996 hat sich Drehung um insgesamt 62 s verlangsamt.

Seit 1972 schon 23 mal Schaltsekunde eingeführt [Zöb08, S. 75]

4. Uhrensynchronisation

Für viele Anwendungen in verteilten Systemen ist es erforderlich, dass alle beteiligten Knoten eine gemeinsame Zeitbasis haben. Gründe sind z.B. (s. ABS-Beispiel):

- ein globaler Takt, um bestimmte Aktionen auf allen Knoten zeitgleich auszuführen
- die Möglichkeit, das "Alter" von Nachrichten zu bestimmen.

Die Uhren aller beteiligten Knoten sollten daher zu jedem Zeitpunkt so wenig wie möglich voneinander abweichen.

Allerdings sind Uhren in typischen Microcontrollern nicht sehr genau; schon gar nicht im µs- oder ns-Bereich. Die Uhren in einem verteilten System "driften" daher recht schnell auseinander, wenn sie nicht regelmäßig korrigiert werden. Darüberhinaus muss vermieden werden, dass eine fehlerbehaftete Uhr (etwa wegen eines HW-Defekts) die gesamte Zeitbasis verfälscht.

Generell ist es unmöglich, alle Uhren in einem verteilten System absolut zu synchronisieren. Eine gewisse Ungenauigkeit wird immer übrig bleiben.

4. Uhrensynchronisation

Für die Uhrensynchronisation gelten die folgenden Anforderungen und Annahmen:

Anforderungen:

- Zu jedem Zeitpunkt zeigen alle Uhren "ungefähr" den gleichen Wert.
  - d.h. eine maximale Abweichung Δ muss sichergestellt sein.
  - "Ereignis *A* vor Ereignis *B*", wenn $t(A) + \Delta < t(B)$
- Alle Uhren zeigen die "richtige" Zeit; d.h. möglichst nah an der "echten" Zeit.
- Die Zeit läuft immer vorwärts (Kausalität).
  - Langsame Uhren können vorgestellt werden.
  - Schnelle Uhren dürfen nicht zurückgestellt werden.
- Keine Zeitsprünge
- Möglichst wenig Overhead durch den Synchronisationsalgorithmus

Annahmen:

- Zu Beginn haben alle Uhren "etwa" die gleiche Zeit.
- Die Zeit für Nachrichtenübertragung und für den Synchronisationsalgorithmus selbst ist vernachlässigbar bzw. bekannt.
- Alle Uhren arbeiten fehlerfrei, d.h. mit einer definierten maximalen Driftrate δ

4. Uhrensynchronisation

Arten der Synchronisation:

- typischerweise periodisch
  - je ungenauer die Uhren, desto häufiger

- externe Synchronisation
  - alle Knoten verlassen sich auf eine externe, als perfekt angenommene Uhr (z.B. GPS).
- interne Synchronisation
  - basierend auf den einzelnen Zeitwerten wird eine korrekte globale Zeit ermittelt.

- zentrale Synchronisation
  - ein ausgewählter Knoten des Netzwerks koordiniert die Synchronisation.
    → fehleranfällig
- verteilte Synchronisation
  - jeder Knoten berechnet die globale Zeitbasis basierend auf den Zeitwerten aller anderen Knoten
    → robuster, aber hohes Datenaufkommen

4. Uhrensynchronisation

Wie oft sollten die Uhren synchronisiert werden?
Sei

- Δ : geforderte maximale Abweichung
- δ : maximale Driftrate einer Uhr, z.B. in Sekunden pro Sekunde (s/s)
- ***S*** : Zeit zwischen 2 Synchronisierungen
- Δ_s: Differenz nach der Synchronisation

Die maximale Differenz zwischen 2 Synchronisierungen beträgt damit: $S * 2\,\delta$

$\Rightarrow$ Wähle *S* so, dass gilt: $S * 2\delta + \Delta_s < \Delta$

$$\Leftrightarrow S < \frac{\Delta - \Delta_s}{2\delta}$$

Beispiel:

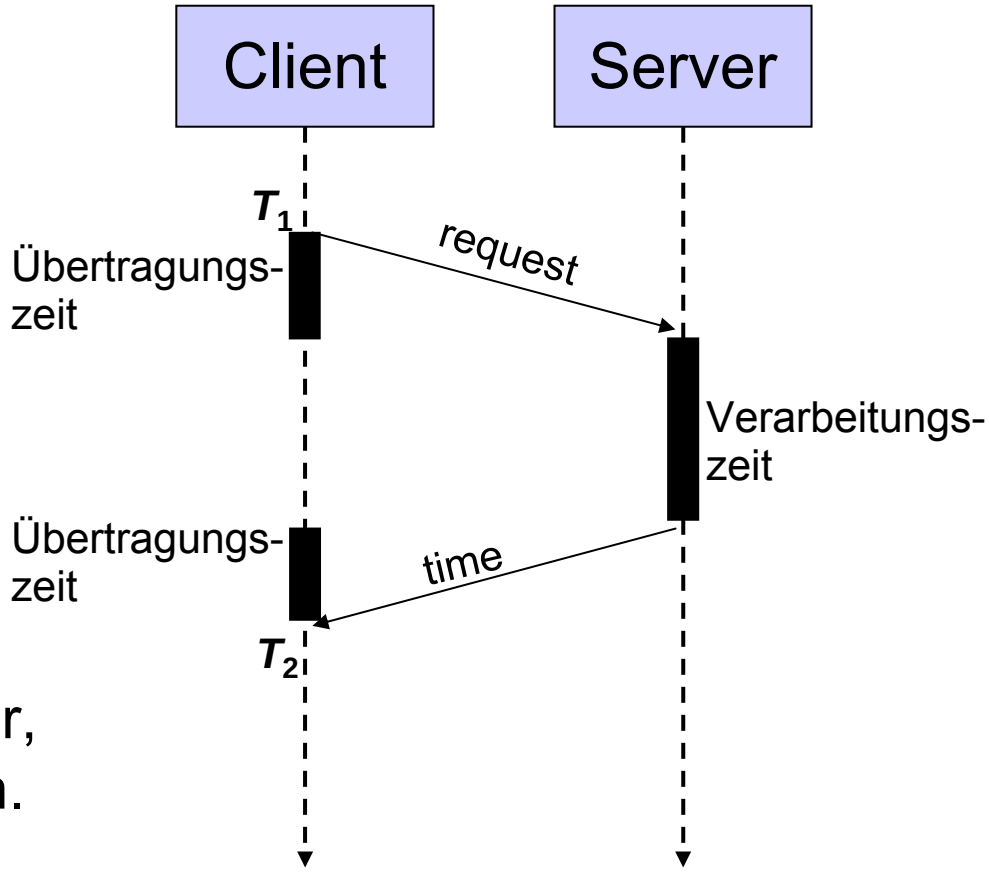
- $\Delta = 10^{-3}$ s (1 ms)
- $\delta = 2*10^{-5}$ s/s (~ 1.7 s pro Tag)
- $\Delta_s = 0$

$\Rightarrow$ ***S*** < 10^{-3} / 2*2*10^{-5} = **25s**

4. Uhrensynchronisation

Verteilte, externe Synchronisation:
Algorithmus von Cristian

- Es existiert ein externer Timeserver (z.B. GPS, UTC)
- Jeder Knoten schickt in regelmäßigen Abständen ein Request, das vom Server schnellstmöglich beantwortet wird.
- Um Zeitsprünge zu vermeiden, passt der Client seine Zeit graduell an (d.h. Beschleunigen bzw. Verlangsamen der Uhr, bis Client- und Server-Zeit übereinstimmen.
- Die Übertragungszeit des Requests kann, falls nicht vernachlässigbar und nicht bekannt, mit $(T_2 - T_1) / 2$ abgeschätzt werden



4. Uhrensynchronisation

Zentrale, interne Synchronisation: **Berkeley-Algorithmus**

Ein Knoten agiert als zentraler Timeserver.
Die Synchronisation läuft in 3 Phasen ab:

- Phase 1:
  - Der Timeserver sendet seine eigene Zeit an alle Knoten und erhält von diesen jeweils die Differenz zwischen der empfangenen und der lokalen Zeit als Antwort.
- Phase 2:
  - Der Timeserver berechnet aus allen Differenzen einen Durchschnittswert.
- Phase 3:
  - Der Timeserver teilt allen Knoten die jeweilige Differenz mit, um die die lokalen Uhren korrigiert werden müssen.
  - Auch die Uhr des Timeservers wird um den Durchschnittswert korrigiert.
  - Die Korrektur der Uhren erfolgt graduell, wenn Zeitsprünge vermieden werden sollen.

Aufgabe:

Wenden Sie den Berkeley-Algorithmus auf die folgende Ausgangssituation an:

Timeserver

13:00

12:55

13:09

NTP – Network Time Protocol www.ntp.org

Die beiden obigen Algorithmen sind nur geeignet für relativ kleine, statische Netze. Für große, dynamische Netze (z.B. Internet) wurde das NTP entwickelt.

Die Grundstruktur von NTP ist ein hierarchisches Modell mit verschiedenen Schichten. Der Dienst wird durch ein verteiltes, hierarchisches Serversystem geleistet:

- Primäre Server sind direkt mit einer UTC-Quelle verbunden.
- Sekundäre Server synchronisieren sich mit primären Servern usw.
- Jede zusätzliche Schicht verursacht dabei einen zusätzlichen Zeitversatz.

Synchronisation bei fehlerbehafteten Uhren

Die bisherigen Algorithmen funktionieren nur unter der Voraussetzung, dass alle Uhren fehlerfrei arbeiten.

Im folgenden wird ein Algorithmus vorgestellt, der mit einer maximalen Anzahl von *f* fehlerbehafteten Uhren noch funktioniert.

Um einen "single point of failure" (wie z.B. beim Berkeley-Algorithmus) auszuschliessen, kommt hier nur ein verteilter Algorithmus in Frage.

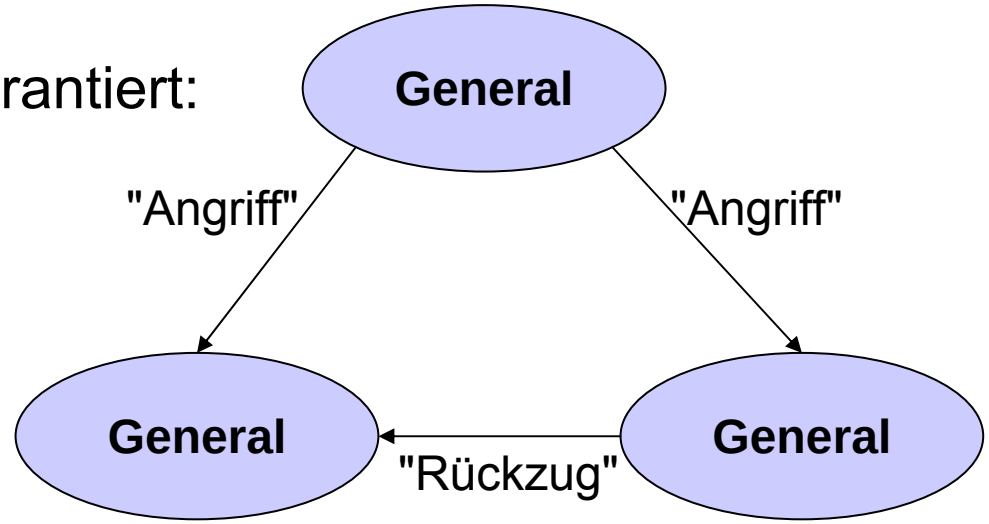
Dabei können auch sogenannte byzantinische Fehler auftreten, d.h. eine fehlerbehaftete Uhr läuft nicht einfach nur falsch, sondern liefert bei jeder Abfrage ein unterschiedliches Ergebnis („*babbling idiot*“)

Klassisches Problem: **Die byzantinischen Generäle**

Die Armee von Byzanz will mit mehreren Divisionen eine Stadt erobern. Jede Division wird von einem General befehligt. Die Generäle kommunizieren ausschließlich durch Boten, um sich auf ein konsistentes Verhalten ("Angriff", "Warten", "Rückzug") zu einigen.
Unter den Generälen können sich Verräter befinden, die durch irreführende Botschaften ein konsistentes Verhalten verhindern wollen. Die Boten sind allerdings verlässlich.

Gesucht: Ein Verfahren, das folgendes garantiert:

- Alle loyalen Generäle kommen zur gleichen, richtigen Entscheidung.
- Den Verrätern ist es nicht möglich, einen falschen Plan durchzusetzen.
- Verräter müssen nicht entlarvt werden.



4. Uhrensynchronisation

Das Problem der byzantinischen Generäle, auch "Byzantinisches Agreement" genannt, ist lösbar in einem System mit *n* Knoten, von denen *f* fehlerhaft ("Verräter") sind, wenn gilt: ***n > 3f***.

[L. Lamport, R. Shostak, and M. Pease: "The Byzantine Generals Problem". *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, July 1982.]

Angewendet auf das Problem der Uhrensynchronisation: Sei

- ***n*** : die Gesamtzahl der Uhren
- ***f*** : die Anzahl der fehlerhaften Uhren
- Δ : die maximal zulässige Abweichung zwischen 2 Uhren

Algorithmus:

- Jeder Knoten liest die Uhren der anderen und setzt seine Uhr auf den Mittelwert.
- Ist die Abweichung zu einer anderen Uhr größer als Δ, so verwendet der Knoten stattdessen den Wert der eigenen Uhr.

Die fehlerfreien Uhren konvergieren mit jeder Iteration.

Annahme: Alle Uhren werden zeitgleich abgelesen, bzw. Differenzen sind vernachlässigbar.

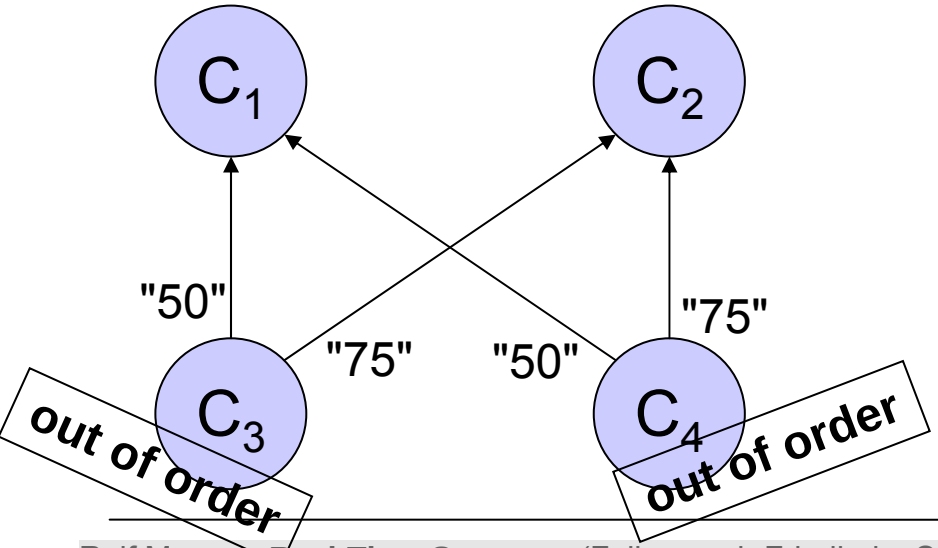
4. Uhrensynchronisation

Aufgabe:

Wenden Sie den Algorithmus auf folgende Situation an (der Einfachheit halber wird die Zeit in Minuten seit 12 Uhr mittags angegeben). Es sei Δ **= 10**.

Uhr	Zeit	1. Iteration	2.Iteration
C_1	60		
C_2	65		
C_3	55		
C_4	120		

Was passiert, wenn C_3 und C_4 falsche Werte liefern, wie im Bild dargestellt:



Uhr	Zeit	1. Iteration	2.Iteration
C_1	60		
C_2	65		
C_3	xxx		
C_4	xxx		

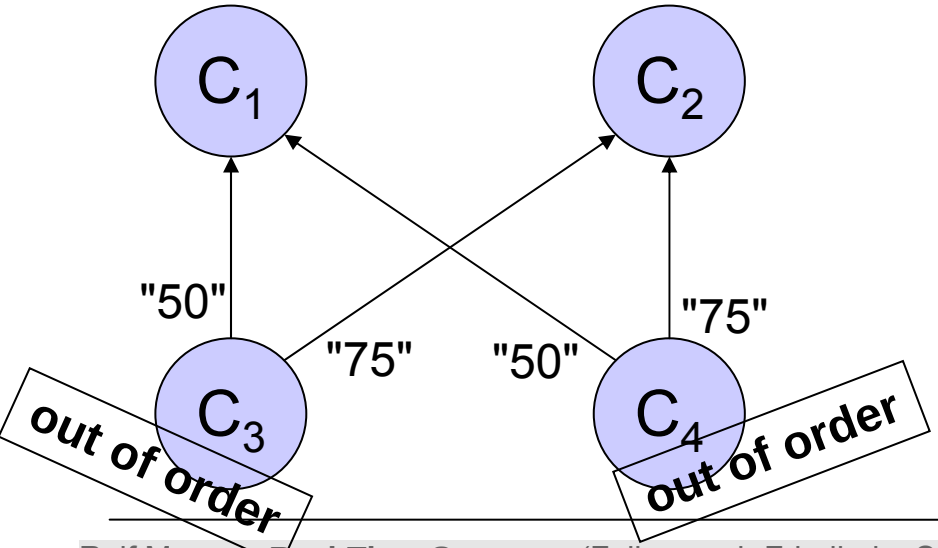
4. Uhrensynchronisation (Test)

Aufgabe:

Wenden Sie den Algorithmus auf folgende Situation an (der Einfachheit halber wird die Zeit in Minuten seit 12 Uhr mittags angegeben). Es sei Δ **= 10**.

Uhr	Zeit	1. Iteration	2.Iteration
C_1	60	60	60
C_2	65	61.25	60.3
C_3	55	58.75	59.69
C_4	120	120	120

Was passiert, wenn C_3 und C_4 falsche Werte liefern, wie im Bild dargestellt:



Uhr	Zeit	1. Iteration	2.Iteration
C_1	60	61.25	61.88
C_2	65	63.75	63.13
C_3	xxx	xxx	xxx
C_4	yyy	yyy	yyy

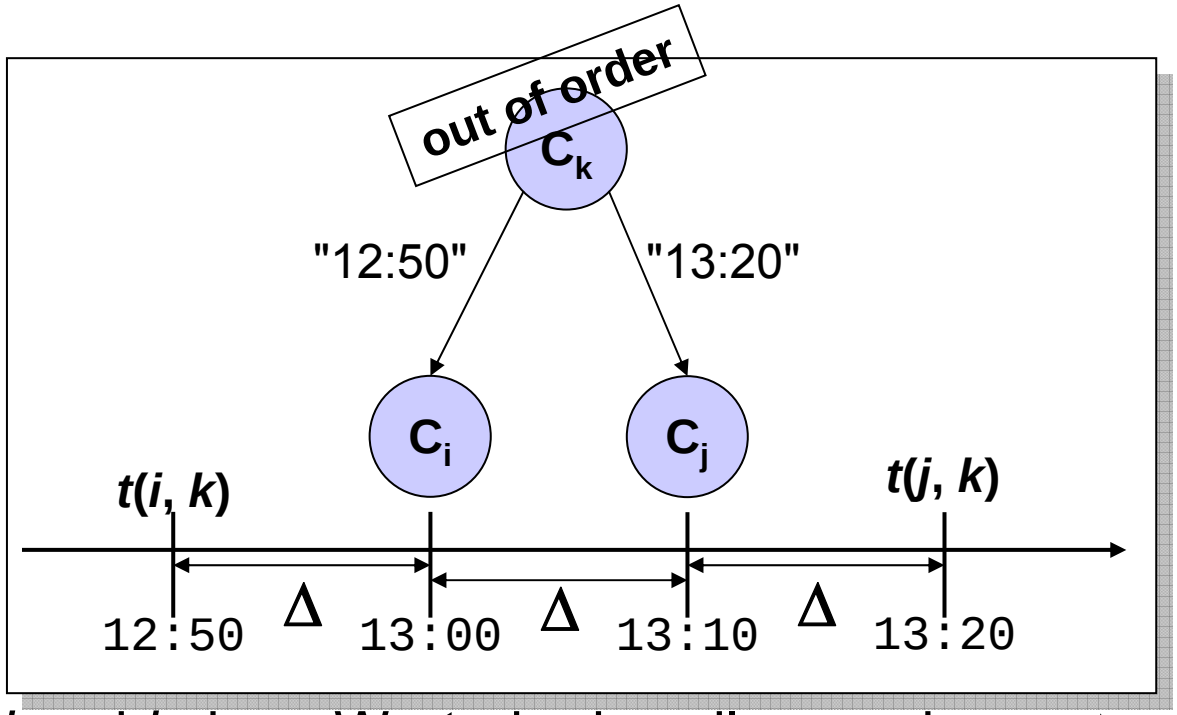
Der Algorithmus arbeitet erfolgreich, falls gilt: ***n > 3f.***

Beweis:

C_i, C_j seien 2 fehlerfreie Uhren, C_k eine beliebige Uhr.

Sei $t(i, k)$ die Uhrzeit, die Knoten i von Knoten k erhält, also:

$$t(i,k) = \begin{cases} C_i, & falls \left| C_i - C_k \right| > \Delta \\ C_k, & sonst \end{cases}$$



Wenn C_k fehlerfrei arbeitet, gilt:

$t(i, k) = t(j, k) = C_k$

Andernfalls gilt:

$| t(i, k) - t(j, k) | \leq 3\Delta$

denn:

- $|C_i - C_j| \leq \Delta$
- im schlechtesten Fall liefert k an i und j einen Wert, der jeweils gerade um Δ von C_i bzw. C_j abweicht (s. Abb.).

4. Uhrensynchronisation

Jeder Knoten *i* stellt seine Uhr auf den Mittelwert aller *t*(*i*, *k*):

$$C_i = \frac{\sum_{k=1}^{n} t(i,k)}{n}$$

Für die Differenz zwischen C_i und C_j nach der Synchronisation ergibt sich:

$$\delta(C_i, C_j) = \left| \frac{\sum_{k=1}^{n} t(i,k) - \sum_{k=1}^{n} t(j,k)}{n} \right| \leq \frac{f * 3\Delta}{n} < \Delta$$

jeder der *f* fehlerhaften Knoten produziert einen Fehler von max. 3Δ (s.o.)

⇔ ***n > 3f***

Bemerkungen

- Die byzantinischen Fehler entstehen nicht unbedingt nur durch fehlerhafte Uhren, sondern auch durch Übertragungsfehler bei der Kommunikation.
- Das gleichzeitige Auslesen aller anderen Uhren für jeden Knoten ist in der Praxis natürlich nicht möglich. Ähnlich wie beim Berkeley-Algorithmus ist es ggf. sinnvoller, wenn statt der absoluten Zeitwerte jeweils die Differenzen zu den anderen Uhren für die Mittelwertbildung benutzt werden.
- Weicht eine Uhr um mehr als Δ von allen anderen Uhren ab, dann wird sie vom Algorithmus ignoriert und hat damit kaum noch eine Chance, wieder in die Synchronisation mit einbezogen zu werden.
- Der Algorithmus kann also nur funktionieren, wenn zu Beginn alle Uhren synchronisiert sind – aber das ist ein separates Problem.

5. Literatur

[Zöb08] Dieter Zöbel: „Echtzeitsysteme – Grundlagen der Planung“, Springer Verlag, 2008

[LSP82] L. Lamport, R. Shostak, and M. Pease: "The Byzantine Generals Problem". ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3, July 1982