Verteilte Systeme
-Übungen-

21. Januar 2005

Harald Vogt

Aktualisiert am 18.02.05

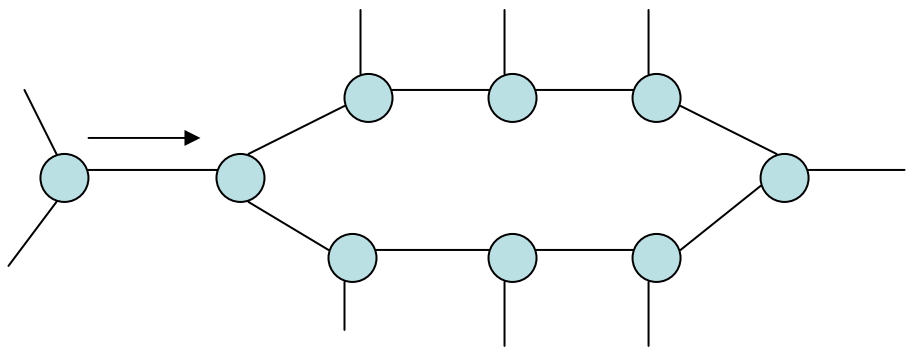
Bemerkungen zur Korrektur

- Bewertung:
  - ✓ : Korrekt bzw. „ich bin einverstanden“
  - (✓) : Im Prinzip korrekt, aber mit gewissen Mängeln, z.B. nur Spezialfall bewiesen
  - „Welle“: zweifelhafte Aussage nochmal drüber nachdenken!

- Achtung:
  - Volle Punktzahl garantiert keine 100%-ige Korrektheit/Vollständigkeit! Keine falsche Sicherheit!

A1: Topologien und Pfade

- Meistens ***kürzester*** Pfad gemeint
- 16x16-Gitter: max. Pfadlänge 30
- Hypercube mit 256 Knoten: Dimension 8 ➔ max. 8 Schritte
- Konstruktion des CCC: ersetze die Ecken des Hypercupe durch Ringe
- Maximale Pfadlänge im CCC:
  - Bei „regulärem“ CCC: Je Dimensionswechsel (ausser dem ersten) noch ein Schritt im Eckring ➔ 2*d-1
  - Bei „bösartigem“ CCC: Je Dimensionswechsel d/2 Schritte (ausser erstem): d+(d-1)*d/2
  - Obere Schranke: d+d*d/2, Beispiel: d = 8: 8+8*8/2 = 40 (Wirklich schlechteste obere Schranke?)
  - *Genaue Lösung hängt davon ab, wie man den CCC konstruiert, Erklärung notwendig!*
    → Viele verschiedene Lösungen wurden als korrekt akzeptiert (15, 16,18,32, 36, 40, ...), Punktabzug bei unklaren Annahmen bzw. Zahlenwerten obskurer Herkunft
  - **Siehe auch nächste Folie!**



A1. Topologien und Pfade

- Maximale Pfadlänge im CCC:
  - Jeder Knoten hat 3 Verbindungen:
    - 2 im Ring
    - 1 Verbindung führt in eine andere Dimension, d.h. jedem Knoten k kann eine Dimension dim(k) zugeordnet werden
  - Die maximale Pfadlänge ist zu finden zwischen zwei Knoten, die
    1. sich in allen Dimensionen (bezogen auf zugrundeliegendem Hypercube) unterscheiden
    2. und deren Ringpositionen maximal voneinander entfernt sind
  - Der maximale Pfad ergibt sich wie folgt:
    1. k = k0 (k0 ist der Startknoten, k* sei der Zielknoten)
    2. coord(x) seien die Hypercube-Koordinaten eines Knoten x. Alle Knoten auf einem Ring haben den gleichen coord-Wert.
    3. Wiederhole:
       1. Führe Dimensionswechsel dim(k) durch: neue Position k‘
       2. Falls coord(k‘) = coord(k*), so haben wir die Zieldimension erreicht. Breche die Schleife ab
       3. Gehe einen Schritt auf dem Ring zum Knoten k‘‘, so dass dim(k‘‘) = dim(k‘) + 1 mod d
       4. k := k‘‘
    4. k‘ und k* liegen auf dem gleichen Ring, sind aber maximal weit voneinander entfernt. Da der Ring d Element enthält, benötigen wir $\lfloor d/2 \rfloor$ Schritte, um k* von k‘ aus zu erreichen
  - Insgesamt werden d Dimensionswechsel sowie d-1 Zwischenschritte durchgeführt und im Zielring nochmals $\lfloor d/2 \rfloor$ Schritte, insgesamt also $d + d-1 + \lfloor d/2 \rfloor = \lfloor 5d/2 \rfloor - 1$ Schritte.
  - Für d=8 ergeben sich damit 19 Schritte
  - Gilt für d > 2

A2: Pfade im Hypercube

- k knotendisjunkte Pfade
- Beweis:
  - 1. Maximal k Pfade
    - Weil maximal k Nachbarn (auf dem Weg zum Ziel)
  - 2. Mindestens k Pfade existieren
    - Pfad repräsentiert durch Folge von Bitflips, z.B. 1,4,3,2 → Folge von Dimensionswechseln
    - Konstruktion von k solcher Folgen durch Verschieben, z.B: 1,2,3,4 / 2,3,4,1 / 3,4,1,2 / 3,1,2,3
      Dürfen niemals zwischendurch im gleichen Knoten landen!
    - Anderes Argument: **Präfixe der Pfade unterschiedlich** (sonst landet man im gleichen Knoten mehrfach); und solche Pfade gibt es genügend viele: (k *über/tief* j) > k (1 <= j < k)

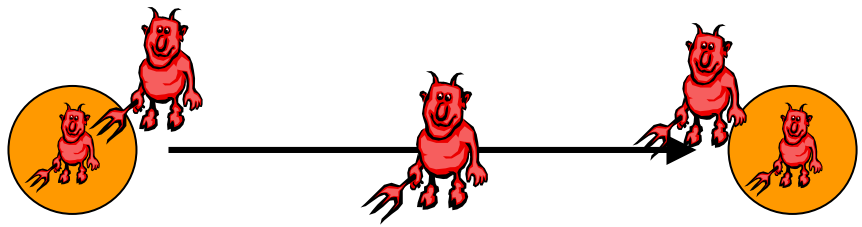
A3: Mittlere Pfadlänge im Hypercube

- Lösungen:
  - Wähle zufällig zwei Positionen, Abstand = Pfadlänge = # untersch. Bits. P(b=0)=P(b=1)=0.5 ➔ 0.5 * d
  - Summe über alle Pfadlängen...
  - Induktion: Für d=1...; für d‘=d+1 gilt: die Hälfte der Knoten in einem zusätzlichen Schritt erreichbar...
  - Erwartungswert der Binomialverteilung
- Achtung:
  - $l = (l_{max}+l_{min})/2$ gilt nur bei symmetrischer Verteilung (begründen!)

$$\sum_{k=0}^{n} \binom{n}{k} \cdot k = n \cdot 2^{n-1}$$

$$l' = \frac{2^d \cdot l + 2^d \cdot (l+1)}{2^{d+1}}$$

A4. Fehlermodelle



- Webseite nicht gefunden: **fail-stop** (Fehlermeldung wird zurückgegeben); **kein Fehler** (Meldung 404 ist erwünscht/korrekt)
- Instabiles WLAN: **Fehlerhaftes Übertragen**; **fail-stop** (je nach Layer); **Zeitfehler** (Verzögerungen durch retransmit); *≠Byzantinisch: kein beliebiges/absichtsvolles Verhalten*
- Virus: Byzantinischer Fehler
- ...

A5. Fehlertoleranz/Verfügbarkeit

- Mit 3 Rechnern:
  - Verfügbarkeit $1 – (1 – 0.6)^3 = 0.936$
- Mit 6 Rechnern > 0.99
  - 0.99 Verfügbarkeit = 0.01 Nichtverfügbarkeit
    ➔ Ausfall **14.4 min/d** !!
    Über‘s Jahr sind das 87.6 h = **3.6 d**
- „five nines“ = 99.999% ➔ 5.3 min/a Ausfall

A6. Puffer

- Wie erwartet nehmen die Zeiten, in denen die Prozesse blockiert sind, mit der Puffergrösse ab
- Ohne Puffer:
  Sender blockiert für 1191 ms
  Empfänger blockiert für 1499 ms
- Mit einfachem Puffer:
  Sender blockiert für 920 ms
  Empfänger für 1250 ms
- Mit dreifachem, kaskadierten Puffer:
  Sender blockiert für 437 ms
  Empfänger für 766 ms
- (Zeiten können sich auf verschiedenen Plattformen leicht unterscheiden)

A7. Mars-Rover

- Signallaufzeit zwischen Erde u. Mars: 185 s
- Übertragungszeit für Bild u. Steuersignal nicht vergessen!
- Zeitbedarf für Reaktion und Steuerung:
  - 2 * 185 s + 4 s = 374 s
- Hinderniserkennung in 10 m
- Maximalgeschwindigkeit: 10 m / 374 s
  - 0.0267 m/s = ca. 96 m/h

A8. RPC

- Call-by-reference
  - Rechner haben getrennte Adressräume
  - Referenz (Pointer) also nicht gültig
  - Möglich: Verpacken des referenzierten Objekts
    - Achtung: zirkuläre Referenzen, grosse Datenstrukturen
  - Oder: Remote-Aufruf bei Dereferenzierungen
    - Bei jedem Zugriff auf Datenstruktur Aufruf zurück zum Client
    - Hoher Aufwand!
- Wiederholung von Anfragen bei fehlender Bestätigung
  - Mehrfache Ausführung potentiell schädlich
  - Sequenznummern: Erkennen von Mehrfachaufrufen
- Fehlerklasse
  - At-least-once
  - GET nicht immer idempotent!

A9. Broadcast

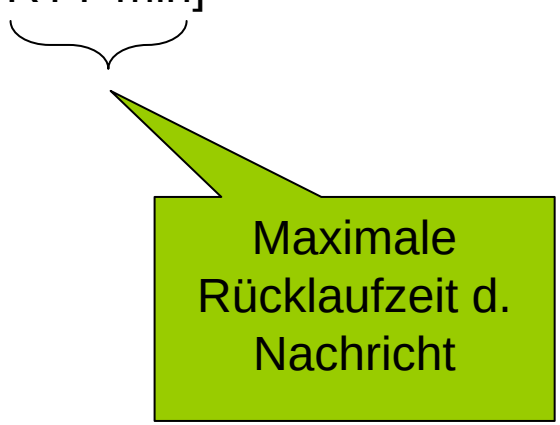
- Atomar: Empfangsreihenfolge überall gleich
  - Im Fall A gegeben
- Kausale Abhängigkeit
  - Im Fall A: ja, B: nein
- Totale Ordnung über Sequencer
  - FIFO-Kanäle
    - Nachrichten zu einem P überholen sich nicht
  - Acknowledgements
    - Implementierung der FIFO-Eigenschaft

A10. Uhrensynchronisation

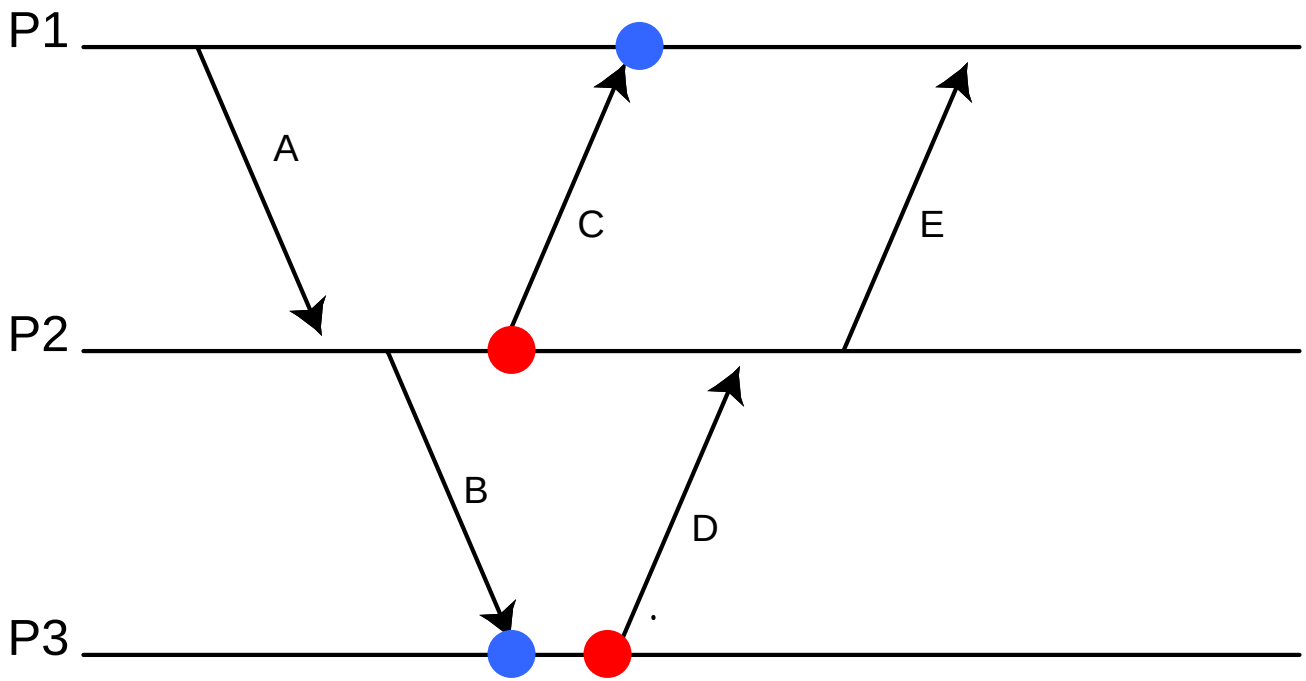
- Clock(A) ... 1000 ticks/s
- Clock(B) ... 990 ticks/s
- Nach 1 Minute:
  - Clock(A) ... 60‘000 ticks
  - Clock(B) ... 59‘400 ticks
- Unterschied: 600 ticks
- Wenn Clock(A) **korrekt** (1 tick = 1 ms) läuft, dann ist Clock(B) also zu **langsam**
  - Unterschied also 600 ticks = 600 ms
- Achtung: Ohne diese Annahme muss tick nicht gleich ms sein!
  - Dann wäre z.B. Clock(B) korrekt und Clock(A) zu schnell

A11. Uhrensynchronisation II

1. Varianz in der Nachrichtenlaufzeit
   - Dynamische Last im Netz
   - Unterschiedliche Routen
2. Mit einer Nachricht:
   - T ... Zeitstempel von M2
   - Clock(P1) := T + **10** ms + (**35**-10)/2 ms = T + 22.5 ms
   - Maximale Abweichung: 12.5 ms
3. Mit RTT:
   - Wenn M2 bei P1 ankommt, ist die Zeit bei P2 bereits weitergelaufen... Clock(P2) $\in$ [T+min,T+RTT-min]
   - Clock(P1) := T + RTT/2
   - Genauigkeit ± RTT/2 – min
   - Bei RTT = 80 ms und min = 10 ms:
     - [T + 10 ms, T + 70 ms]
     - Clock(P1) = T + 40 ms
     - Genauigkeit ± 30 ms



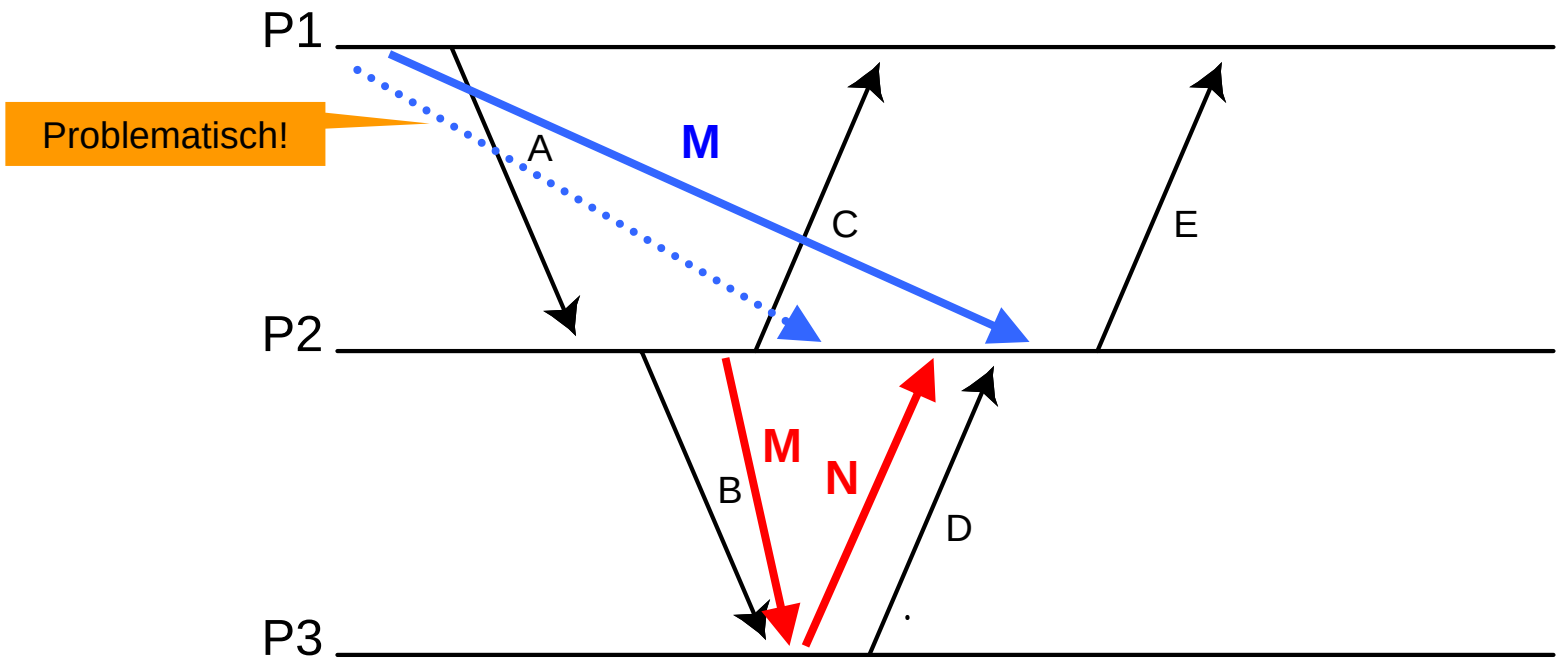
B1. Lamport-Zeit I



Achtung: Gefragt sind **Ereignisse**, keine **Nachrichten**!
1 Nachricht ≡ 2 Ereignisse!

Daher: Antwort C/D ist **falsch**

B1. Lamport-Zeit I



A.receive ≺ N.send ∧ C(N.receive) < C(E.send)

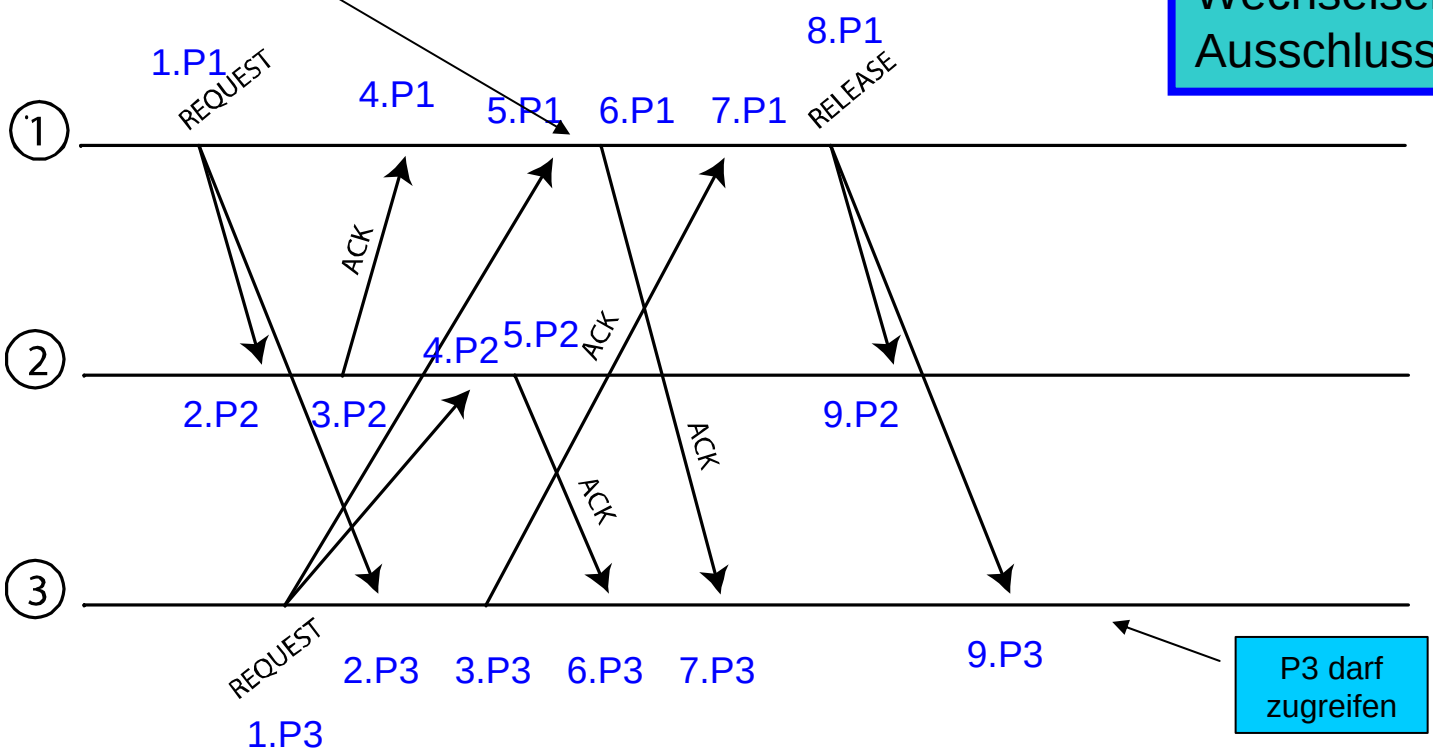
M.send ≺ C.send ∧ C(B.receive) < C(M.receive)

M von P1 nach P2 !!

B2. Lamport-Zeit II

P1 darf zugreifen

Wechselseitiger Ausschluss



P3 darf zugreifen

!

Empfangszeitstempel:
Erst ticken, dann zuweisen!

```
lokal := max(lokal,C(msg.send)) + 1
C(msg.recv) := lokal
```

Sorry – falsch korrigiert...

B2. Lamport-Zeit II

P1
REQ(1.P1)
REQ(1.P1) **P2**
REQ(1.P1); REQ(1.P3) **P2,P3**
REQ(1.P1); REQ(1.P3) **P2,P3**
REQ(1.P1); REQ(1.P3) **P2,P3**
REQ(1.P3)

P3
REQ(1.P3)
REQ(1.P1); REQ(1.P3)
REQ(1.P1); REQ(1.P3)
REQ(1.P1); REQ(1.P3) **P2**
REQ(1.P1); REQ(1.P3) **P2,P1**
REQ(1.P3) **P2,P1**

- Sendezeitstempel verwenden, da diese überall gleich sind
- REQ muss in der Warteschlange vorne sein und ältere Nachricht von P2 und P3 muss empfangen sein
- P1 kann nach der 3. Nachricht die Ressource benutzen
- P3 kann nach der 6. Nachricht zugreifen

B3. Namen

- Beispiele: Telefonbuch, DNS
- Caching bei häufigem Auflösen von Namen und relativ stabilen Bindungen
- Problem von Caching:
  - Zugriff auf veraltetes Objekt
  - Nameserver löst das Problem, aber dessen Cache könnte auch veraltet sein
  - Lösung: Aktive Propagierung von Bindungsänderungen; Cache-Eintrag ungültig machen; zeitliches Gültigkeitslimit für Cache
- Kriterien für Namen/Adressen
  - Träger von „höherer“ Information wie Typ, Ort (Namen)
  - Mehrere Namen im gleichen Kontext, aber nur eine Adresse (Ausnahmen, z.B. dynamischer DNS für Loadbalancing)
  - Eindeutigkeit
  - Zeitabhängigkeit (Adresse)
  - Änderungshäufigkeit (Namen fast nie, Adresse manchmal)
- Bemerkung:
  - TTL (=time to live) normalerweise Anzahl Hops
  - Was sagt TTL = 5 Sekunden? Wann endet die Lebenszeit? Referenzzeitpunkt notwendig!

B4. Client/Server

- **`http://www.amazon.de/exec/obidos/search-handle-form/ref=sr_sp_go_qs/028-6962200-8148522`**
  - Anderer Client könnte die Sitzung „übernehmen“, wenn er das Tag kennt
  - Abhilfe: IP-Adresse, Cookie (beim ersten Anmelden einen Authenticator speichern), jedesmal neu authentisieren
  - Verschlüsselung allein nützt noch nicht so viel... Tag darf nicht in der URL erscheinen
- Probleme mit abgestürzten Clients
  - Ressourcen auf dem Server bleiben belegt, evtl. sogar gelockt
- Vorteile
  - Zustandslos: Fehlertoleranz (Crash von Clients oder Server); kann u.U. mehr Clients bedienen (kein lokaler Speicher notwendig)
  - Zustandsbehaftet: Effizienter (weniger Kommunikationsaufwand; keine vollständige Beschreibung des Auftrags notwendig), Ressourcen können gesperrt werden

B5. Jini

- Ansatz, mit den allgemeinen Schwächen/Probleme von Rechnernetzen umzugehen
- Leases
  - Automatisches Freigeben von Ressourcen nach Ablauf einer Sperrzeit
  - Leases müssen vom Client explizit erneuert werden
  - Werden von JINI selbst benutzt (z.B. für Dienstverfügbarkeit, Benachrichtigungen), aber auch von Anwendungen nutzbar
- Mobiler Code
  - RMI überträgt Objektzustand; Code muss anderweitig beschafft werden (z.B. über Webserver)
  - Erlaubt „smart proxies“: Funktionalität zum Client verlagern; Vorverarbeitung
  - „Versteckt“ das Netzwerk-Protokoll
- RMI
  - „Call-by-value“: Serialisierung des Objektzustandes
  - „Call-by-reference“: falls als remote object deklariert; Kommunikation über Stub/Skeleton-Paar (ähnlich RPC)

B6. Sicherheit

- Einmalpasswörter mit Einwegfunktion (Hash)
  - Integrität des Startwerts muss gewährleistet sein → sonst denial-of-service
  - Und sogar: Angreifer kann sich selbst authentisieren, echter Client aber nicht
  - Geheim nicht notwendig → wird nicht zur Verschlüsselung o.ä. verwendet
- Versehentlich den falschen Wert übertragen
  - **Angreifer kann aus „mittlerem“ Wert die Folgewerte berechnen und diese zur Authentisierung verwenden**
  - (Angreifer kann diesen Wert direkt verwenden, wenn er an der Reihe ist)
- Kann man TGT missbrauchen?
  - Abgesichert durch:
    - TGT für Clients nicht lesbar (mit KC verschlüsselt)
    - TGT enthält Schlüssel K
    - K ist „echtem“ Client bekannt
    - TGS erhält AN mit K verschlüsselt
    - AN enthält Angaben zum Client
  - TGT ist also personalisiert (durch AN), d.h. nur „echter“ Client darf es benutzen → Spoofing-Angriff notwendig
  - K wird benötigt, um mit TGS zu kommunizieren
  - TGT allein reicht also nicht aus
- Trennung von KDC und TGS
  - Räumliche und zeitliche Trennung von Authentisierung und Service-Benutzung → Vorteile bei Mobilität
  - Wiederverwendung von TGT für mehrere ST
  - TGS benötigt keine Benutzerverwaltung etc. → ist einfacher zu implementieren
  - Root-Passwort wird seltener verwendet → weniger Informationen für kryptographische Angriffe