Inhalt

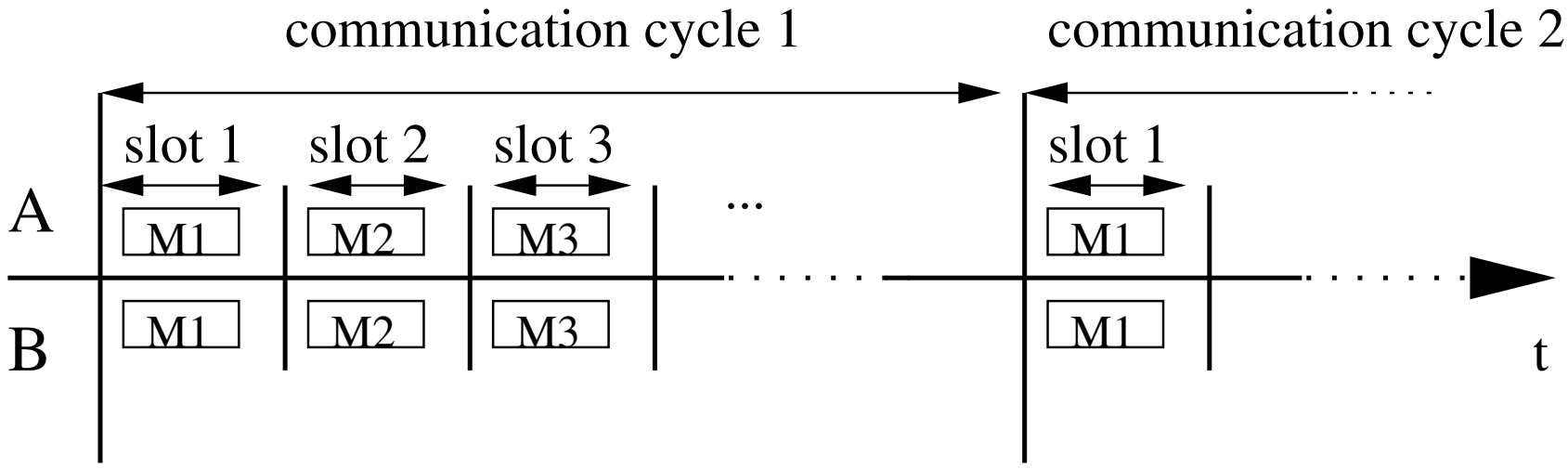
“A Flexible Slotting Scheme for TDMA-Based Protocols”

1. Einführung
2. Ziel
3. Lösung
   (a) Vermeiden des „Babbling idiot”
   (b) Übereinstimmungsalgorithmus
   (c) Korrektheitsbeweis
4. Zusammenfassung

Einführung

TDMA: “time division multiple access”

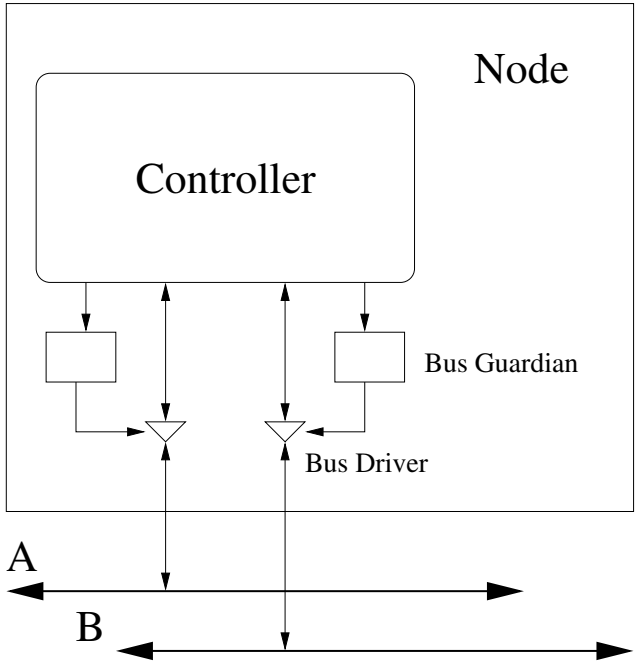
- Kommunikation in *Zyklen*
- Zyklen sind eingeteilt in *Slots*
- Sender sind den Slots eindeutig zugeteilt



Einführung

Festes Zeitraster

$\longrightarrow$ Bus kann durch Guardians geschützt werden



Guardian:
Verwehrt Zugriff auf den Bus, falls der Sender dem Slot nicht zugeteilt ist.

$\longrightarrow$ Fehlertolerante verteilte Echtzeit-Kommunikation möglich.

Einführung

Nachteil:

- Zuteilung muss statisch vorkonfiguriert werden
- Zusätzliche temporäre Slots sind für Sender
  - nicht verfügbar
  - nicht abzusichern

Beispiel: Ausnahmebehandlung
Ausnahmen sollten geworfen werden können, ohne die reguläre Kommunikation zu stören

Einführung

Mögliche Lösung mit TTP:

Konfigurieren zusätzlicher Slots

- Verschwendet Bandbreite
- Guardians müssen einer dynamischen Zuteilung folgen
  $\longrightarrow$ komplexere Komponenten benötigt

Mögliche Lösung mit FlexRay:

- *Statisches Segment*: TTP-artige Zuteilung
- *Dynamisches Segment*: Flexible Zuteilung möglich (Minislotting), aber Guardians werden abgeschaltet!
  $\longrightarrow$ Keine fehlertolerante Kommunikation!

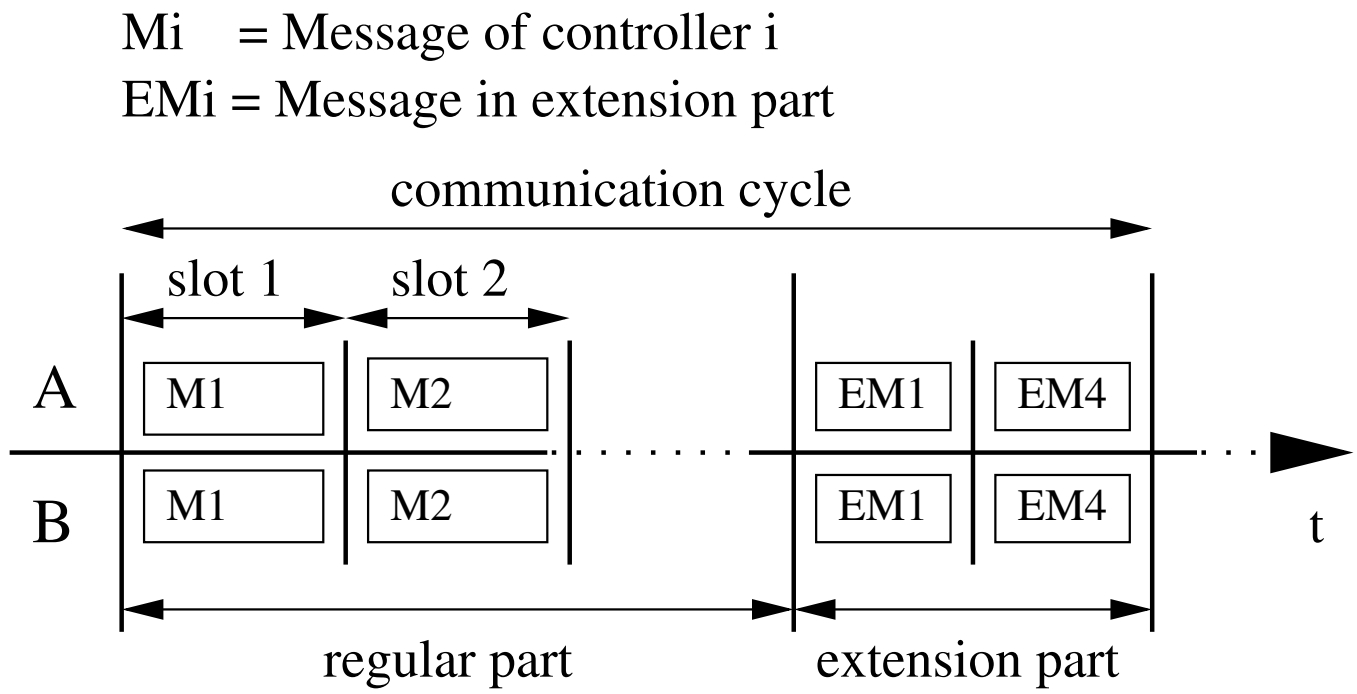
Ziel

1. Jeder Controller soll einen zusätzlichen Slot reservieren können
2. Fehlertolerante Kommunikation
3. Zu tolerierende Fehler
   - Einen fehlerhaften Kanal: Kann Nachrichten zerstören, Nachrichten werden nicht an alle Controller weitergeleitet (byzantinische Fehler möglich)
   - Ein fehlerhafter Controller
   - Sowohl ein fehlerhafter Kanal als auch ein fehlerhafter Controller

Lösung

Zyklus wird aufgeteilt:

- Regulärer Teil
- Erweiterter Teil



Lösung

Regulärer Teil:

- Alle Slots haben gleiche Länge
- In jedem Slot eine Nachricht
- Feste Anzahl von Slots
- Statische Länge
- Vorkonfigurierte Zuordnung
- Jeder Controller sendet mind. einmal
- Jeder Controller sendet Reservierungswunsch mit Nachricht in seinem Slot

Lösung

Erweiterter Teil:

- Alle Slots haben gleiche Länge
- In jedem Slot eine Nachricht
- Anzahl der Slots unterschiedlich
- Dynamische Länge
- Dynamische Zuteilung
- Ein Slot reserviert pro Controller falls angefordert

Lösung

Benötigt:

- Schutz des Übertragungsmediums: „Babbling idiot” muss vermieden werden
- Übereinstimmung über Zuteilung im dynamischen Teil

Vermeiden des „Babbling Idiot”

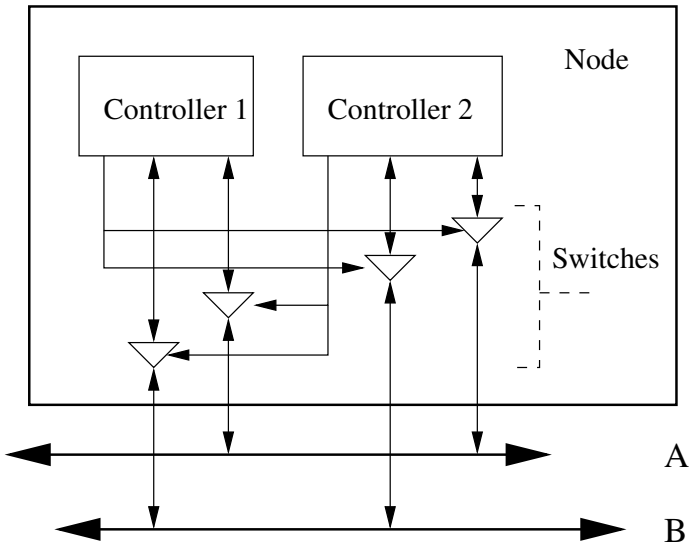
Lösung mit komplexeren Guardians?

Fehlerhafter Guardian $\xrightarrow{\text{kein Einfluß!}}$ Kanal

Controller müssen Information über Zuteilung austauschen
$\longrightarrow$ Guardian muß Nachrichten untersuchen
$\longrightarrow$ Guardian muß am Übereinstimmungsverfahren teilnehmen
Zusätzliche Funktionalität erforderlich
$\longrightarrow$ Teurere Komponenten
Guardians müssen synchronisiert werden
$\longrightarrow$ Zusätzlische Hardware (Quarz) notwendig

Vermeiden des „Babbling Idiot”

Lösung ohne Guardians:



Controller
überwachen sich gegenseitig!
Ein fehlerfreier Controller hält einen fehlerhaften Controller vom senden ab.

$\longrightarrow$ Kein Babbling idiot möglich.
$\longrightarrow$ Keine Guardians zu synchronisieren

Nachteil:
Fehlerhafter Controller kann fehlerfreien am senden hindern.

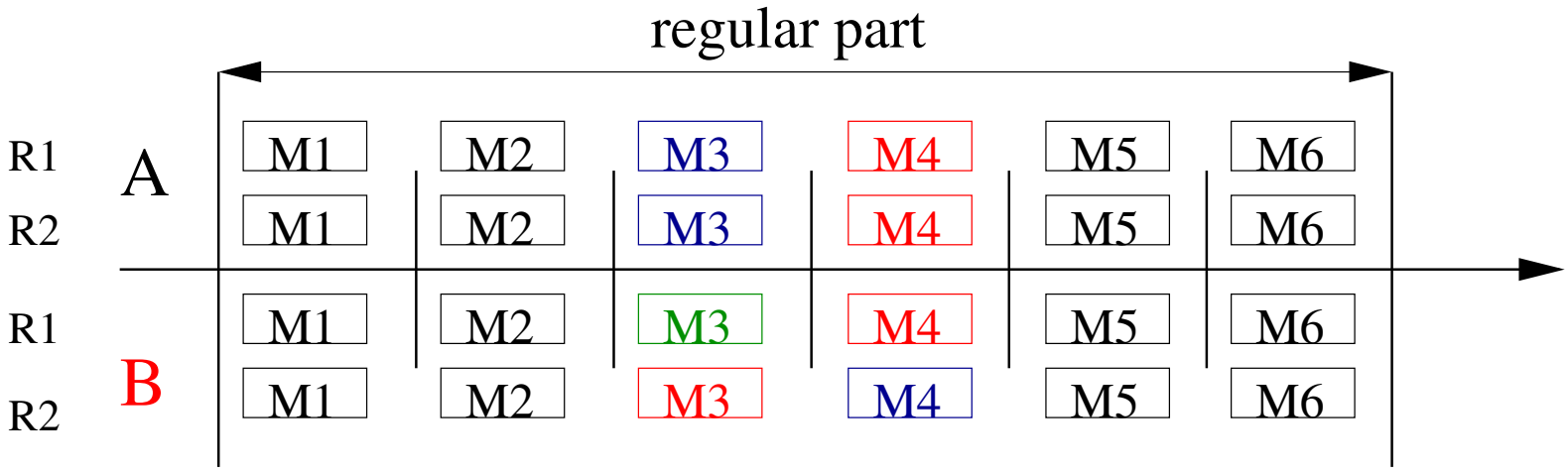
Fehlerannahme

- Fehlerhafter Controller
  - kann ungültige Nachrichten versenden
  - kann sich anders als angekündigt verhalten
  - Kann außerhalb des zugeteilten Slots (wird von fehlerfreien „Nachbarn” verhindert)
  - Kann einen fehlerfreien Controller blockieren
- Fehlerhafter Kanal
  - kann Nachrichten zerstören
  - generiert keine Nachrichten
  - kann keine Nachricht verändern ohne sie zu zerstören (entsp. Schutzmaßnahmen vorausgesetzt z.B. CRC)
  - Nicht alle Controller empfangen Nachricht

Übereinstimmung

Übereinstimmung (blau oder grün) kann nicht immer erzielt werden:

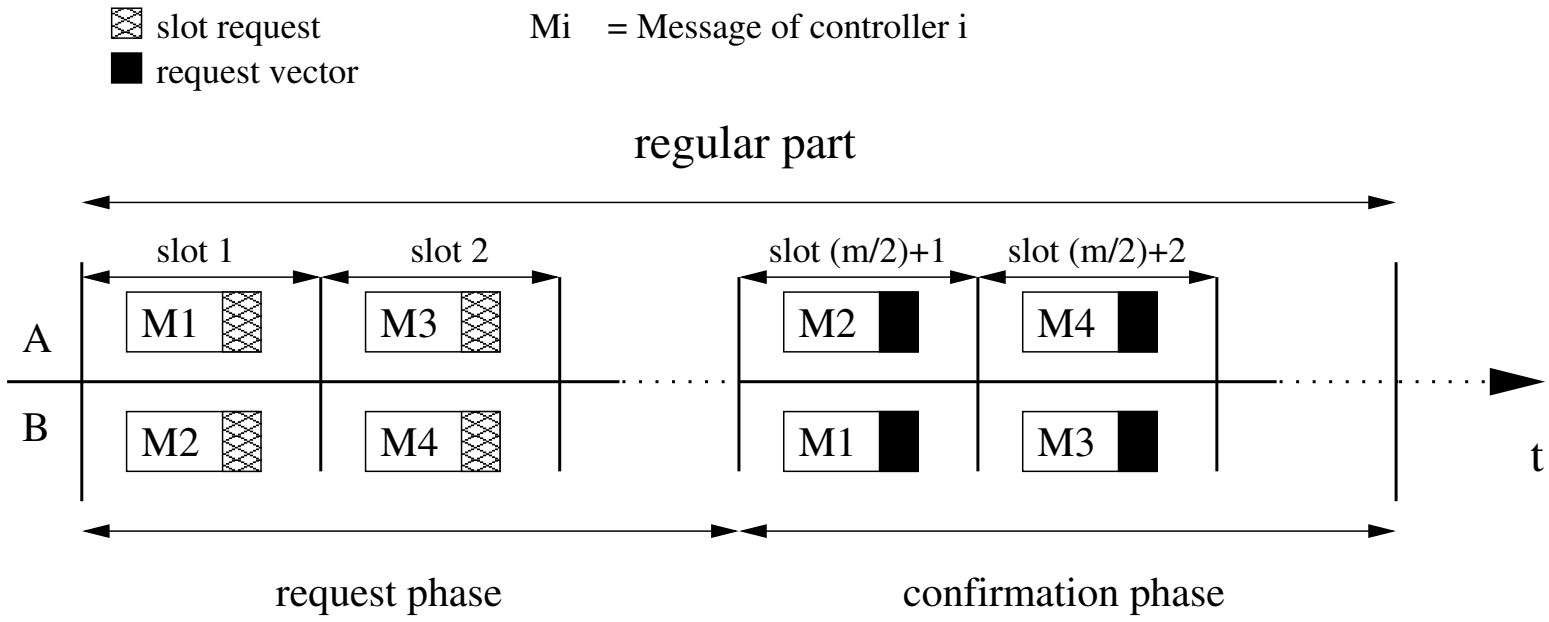
- Fehlerhafter Controller (3) blockiert fehlerfreien Controller (4)
- Kanal fehlerhaft (B)



R1:	Widerspruch für C3	Keine Informationen für C4
R2:	Blau für C3	Blau für C4

Übereinstimmungsalgorithmus

- Regulären Teil aufspalten:
- Controller senden auf beiden Kanälen in verschiedenen Slots
  - Anforderungsphase: Slot anfordern
  - Bestätigungsphase: Sende alle erhaltenen Anforderungen



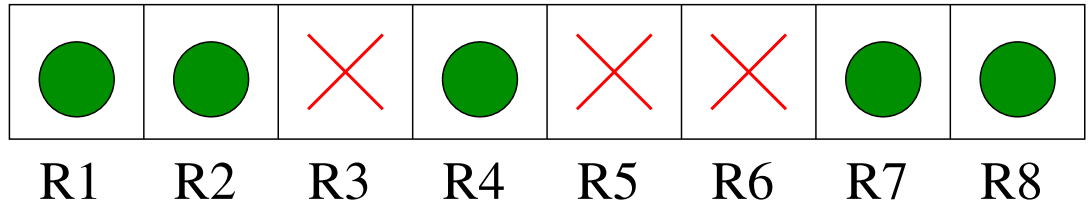
Übereinstimmungsalgorithmus

- Anforderungsphase
  - Sende Anforderung
  - Sammle empfangene Informationen
- Bestätigungsphase
  - Sende Vektor von empfangenen Anforderungen
  - Ersetze alle Anforderungen durch “unknown” falls ursprüngliche Nachricht ungültig
- Ende des regulären Teils
  - Bilde Mehrheit aus bestätigten Anforderungen für Controller
  - Endgültige Entscheidung über Reservierung
- Mindestens 8 Controller nötig

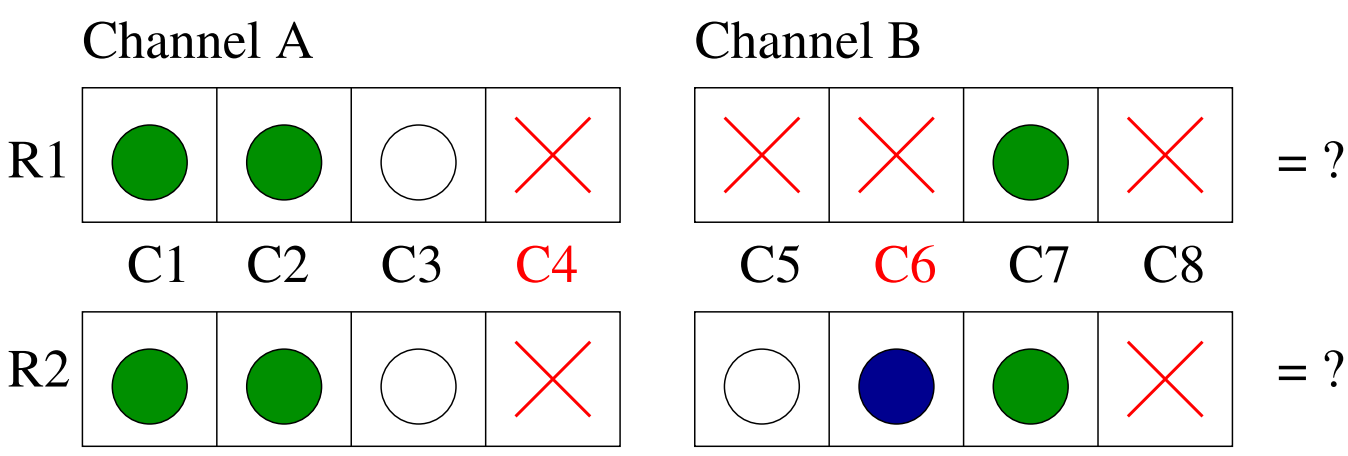
Übereinstimmungsalgorithmus

Request phase

Controller 1 sends on faulty channel B



Confirmation phase



Voting

Funktion “majority(c, channel)”

- Zähle Bestätigungen pro Kanal
  - request_slot_counter: Anzahl der Bestätigungen mit Anforderungen
  - request_no_slot_counter: Anzahl der Bestätigungen ohne Anforderungen
  - Ignoriere “unknown”
  - Ignoriere ungültige Nachrichten
- gib “unknown” zurück falls beide Zähler kleiner zwei
- sonst gib Mehrheit zurück

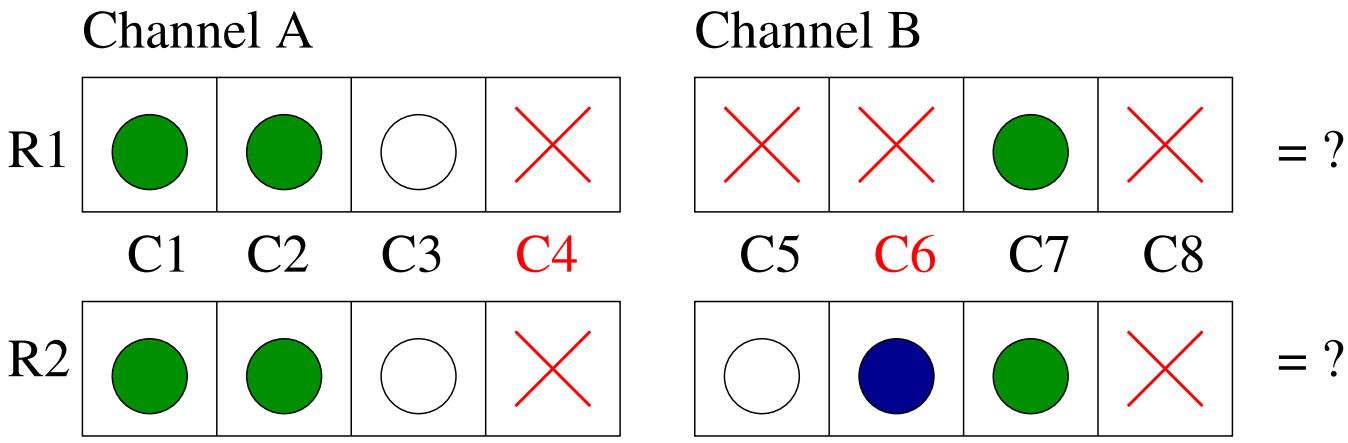
Voting

R1 Channel A — return

request_slot_counter = 2
request_no_slot_counter = 0
} > 1 → ●

Channel B

request_slot_counter = 1
request_no_slot_counter = 0
} < 2 → ○



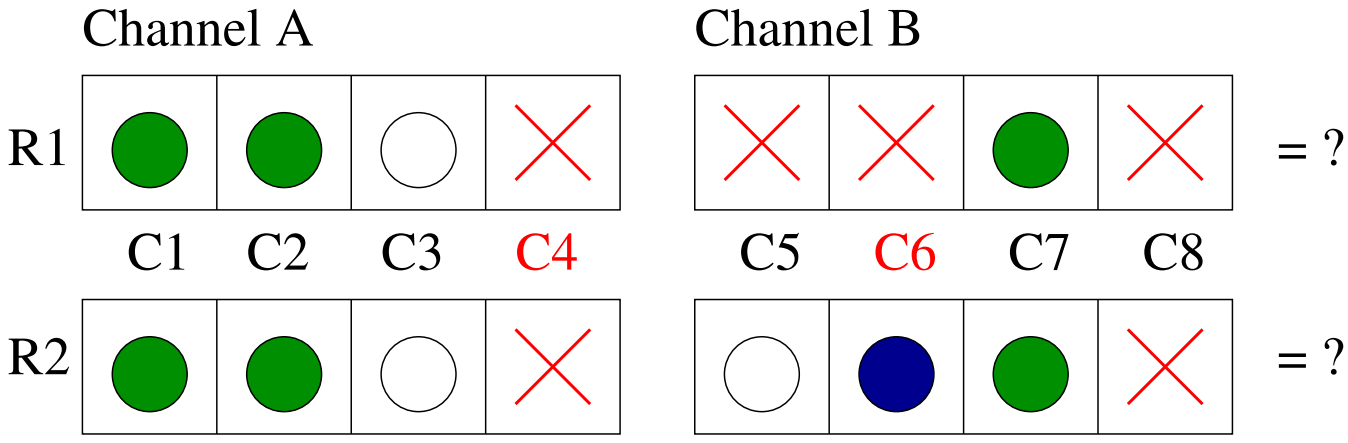
Voting

R2 Channel A — return

request_slot_counter = 2
request_no_slot_counter = 0
} > 1 

Channel B

request_slot_counter = 1
request_no_slot_counter = 1
} < 2 



Entscheidung

Auf dem Kanal auf dem der Controller in der Bestätigungsphase sendet zähle:

- Anzahl der “unknown”-Einträge
- Anzahl Wiederspruch

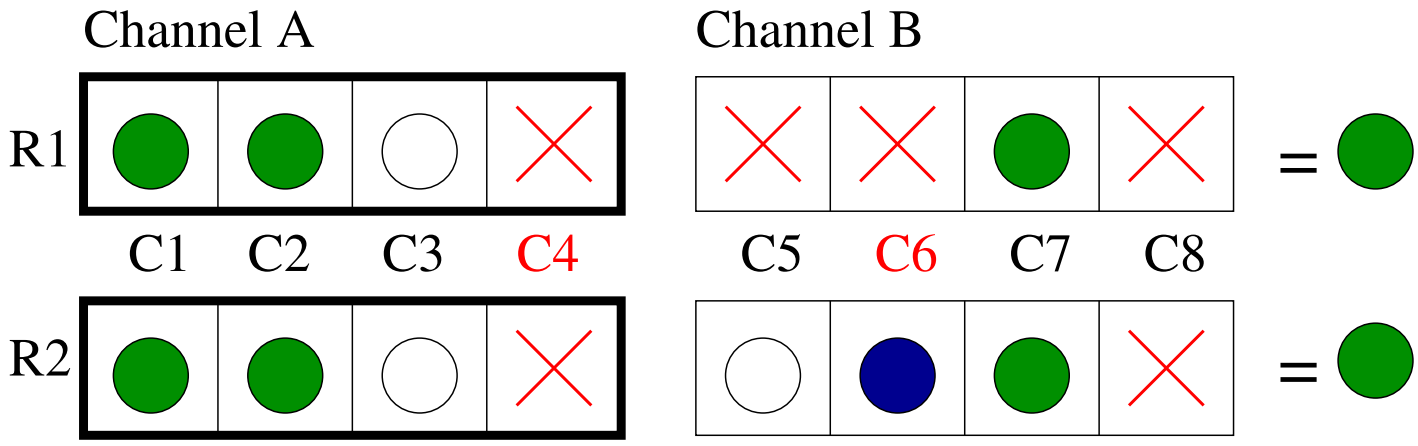
Reserviere Slot, falls ...

- die Mehrheit auf mindestens einem Kanal eine Anforderung ist und
- der “unknown”-Zähler kleiner als zwei ist

Entscheidung

R1: ch_unknown = 1 < 2

R2: ch_unknown = 1 < 2



Eigenschaften von “majority”

Matrix für Empfänger r

$$
\begin{aligned}
rm^r_{i,j} &= 1 && \text{falls } sg^j(i) = rq(i) \text{ und } rq(c) \neq \times \\
rm^r_{i,j} &= 0 && \text{falls } sg^j(i) = 0 \\
rm^r_{i,j} &= \times && \text{falls Nachricht von } j \text{ ungültig} \\
rm^r_{i,j} &= -1 && \text{falls } sg^j(i) \neq rq(i)
\end{aligned}
$$

D Menge der Controller, die in der Anforderungsphase auf fehlerfreiem Kanal gesendet haben

$\overline{D}$ Menge der Controller, die in der Anforderungsphase auf fehlerhaften Kanal gesendet haben

A fehlerfreier Kanal

B fehlerhafter Kanal

Eigenschaften von “majority”

$majority(c, ch)$ gibt niemals -1 zurück

$\sigma_c^r(1)$:
Summe der Einträge mit $rm_{c,j}^r = 1$ für alle j auf Kanal ch in $majority(c, ch)$

In jedem Fall: $\sigma_c^r(-1) \leq 1$

Falls $c \in D$, c fehlerfrei und $ch = A$:
$\sigma_c^r(1) \geq \frac{m}{2} - 2 \Rightarrow majority(c, ch) = 1$

In allen anderen Fällen:
Falls $\sigma_c^r(1) \leq 1 \Rightarrow majority(c, ch) = 0$
Falls $\sigma_c^r(1) \geq 2 \Rightarrow majority(c, ch) = 1$

Eigenschaften von “ch_unknown”

Falls $c \in D$:
Entweder ein „unknown” oder ein Wiederspruch
$\Rightarrow ch\_unknown \leq 1$ für alle r

Falls $c \in \overline{D}$:
Falls ein „unknown” oder ein Wiederspruch:
$\Rightarrow ch\_unknown \leq 1$ und $\sigma_c^r(1) \geq 2$
$\Rightarrow majority(c, A) \neq 0$

In allen anderen Fällen: $ch\_unknown > 1$ für alle Empfänger
(default-Wert wird eingesetzt)

$\Longrightarrow$ Übereinstimmung für alle Controller

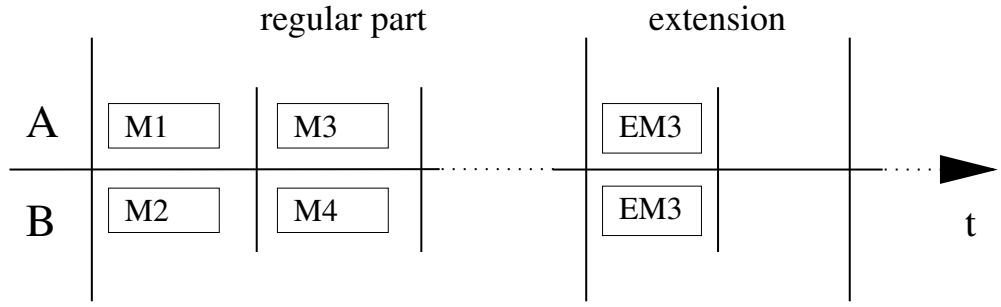
Slot Overloading

Bedingung für Echtzeit-Anwendungen:

Feste Zyklenlänge!

Slot overloading:

- Erweiterter Teil hat feste Länge
- Slots in erweitertem Teil können auf Anfrage benutzt werden
- Zuteilung in erweitertem Teil dynamisch



Zusammenfassung

- Anwendungen, die sowohl flexibel als auch fehlertolerant sein sollen sind mit bestehenden fehlertoleranten TDMA-basierten Protokollen nicht realisierbar.
- Dynamische Zuteilung ist für zeitkritische Anwendungen möglich.
- Fehlertoleranz kann garantiert werden.