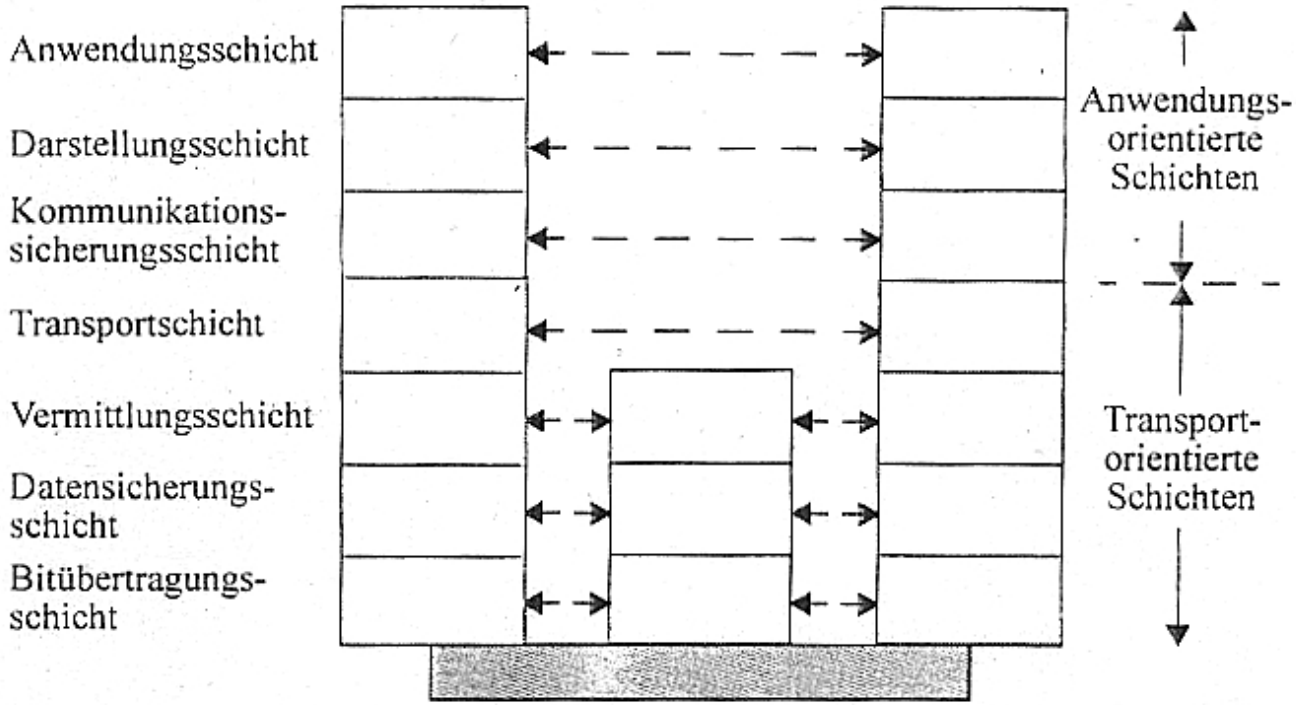
Data-Link Protokolle

N. Lynch, Distributed Algorithms, Kaufmann Publishers

Daniel Miesling
Email: daniel-miesling@arcor.de

1. Juli 2004

Motivation - Was sind Data-Link Protokolle

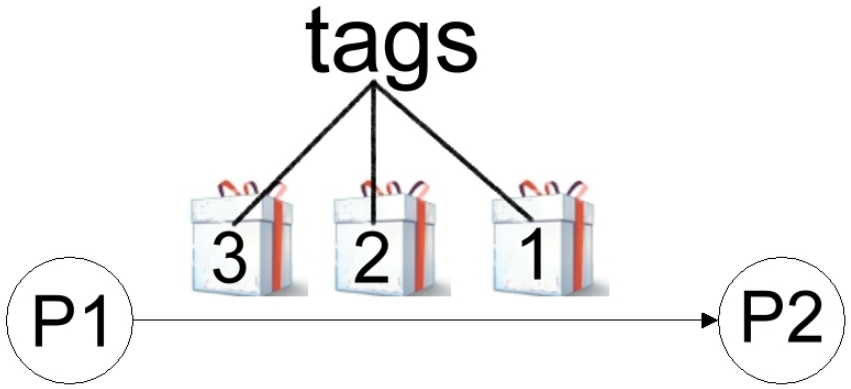


OSI-Referenzmodell

Die Datensicherungsschicht (data link layer,DL) sichert die korrekte Übertragung zusammenhängender Bitfolgen, die als Rahmen bzw. Frames bezeichnet werden, zwischen den über eine Punkt-zu-Punkt-Verbindung gekoppelten Netzknoten.
Zu den Aufgaben dieser Schicht gehört u.a. die Rahmenbildung, die Synchronisation der Rahmen sowie die Behandlung von Übertragungsfehlern.

König, H. (2003): Protocol Engineering

Grundlagen - tags, bounded, unbounded

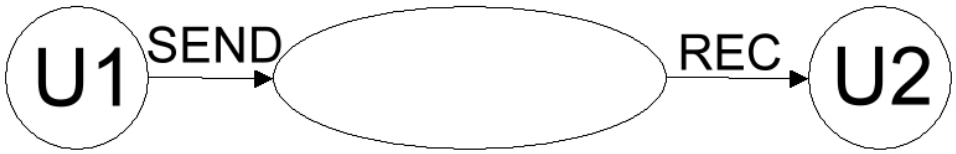


Übertragung der Datenpakete mit tags von P1 nach P2

bounded-tags:
die Menge der tags ist endlich

unbounded-tags:
es stehen unendlich viele tags zur Verfügung

Universal Reliable FIFO-Channel



Datenübertragung mit dem Universal Reliable FIFO-Channel

Spezifikation des Universal Reliable FIFO-Channel :

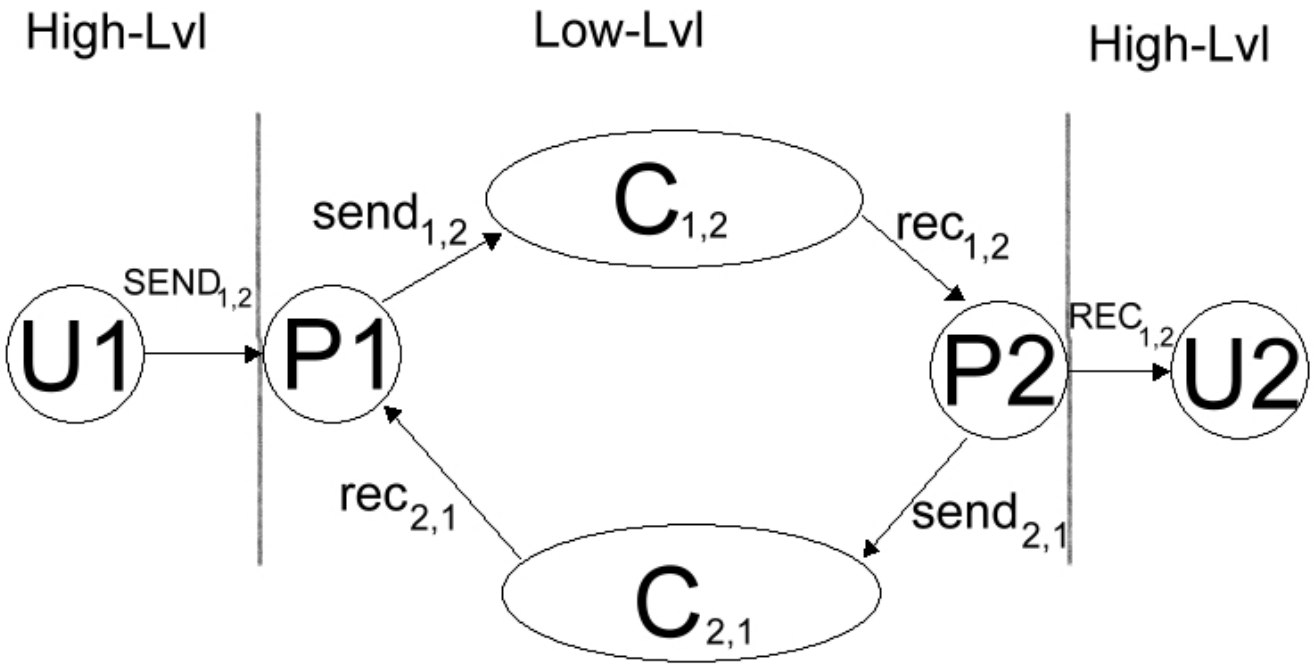
- zu jedem SEND(m) erfolgt irgendwann ein REC(m)
- es tritt kein REC(m) auf ohne, dass zuvor genau ein korrespondierendes SEND(m) aufgetreten ist.

  Bem.: Fasst man den Universal Reliable FIFO-Channel als Funktion auf, so ist diese bijektiv

Ziel:

Die Spezifikation des Universal Reliable FIFO-Channel zu implementieren

Architektur der Data-Link Protokolle



U1, U2: High-Level Prozesse
SEND, REC: High-Level Messages
rec, send: Low-Level Messages
P1, P2: Low-Level Prozesse
P1xP2: Parallelschaltung von P1 und P2
P1xP2 mit $C_{1,2}$und $C_{2,1}$: Implementierung eines Protokolls

P1

Signature:

Input:

$rec_{2,1}(world)$

Output:

$send_{1,2}(hello)$

States:

Transitions

$send_{1,2}(hello)$

Precontition: true

$rec_{2,1}(world)$

Tasks: $\{send_{1,2}(hello)\}$

P2

Signature:

Input:

$rec_{1,2}(hello)$

Output:

$send_{2,1}(world)$

States:

queue, ein FIFO queue mit Nachrichten als Elemente, initial leer

Transitions

$rec_{1,2}(hello)$

Effect: füge zum queue beliebig viele hello hinzu

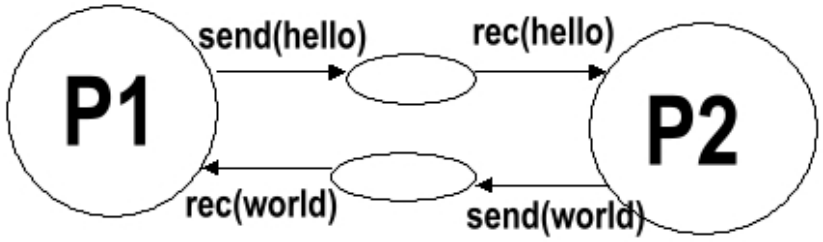
$send_{2,1}(world)$

Precondition: hello $\in$ queue

Effect: entferne oberes hello aus dem queue

Tasks: $\{send_{2,1}(world)\}$

Ausführung, traces, Projection, fairtraces, Tasks



Ein trace ist eine Folge von externen Aktionen
Beispiele für traces:

1. $send_{1,2}(hello), send_{1,2}(hello), rec_{1,2}(hello), send_{1,2}(world), rec_{2,1}(world)$

2. $send_{1,2}(hello), rec_{1,2}(hello), send_{1,2}(hello), rec_{1,2}(hello), \ldots$

Projection:
α trace wie im Bsp. 1
$\alpha\downarrow$P1 $= send_{1,2}(hello), send_{1,2}(hello), rec_{2,1}(world)$

$\alpha\downarrow$P2 $= rec_{1,2}(hello), send_{1,2}(world)$

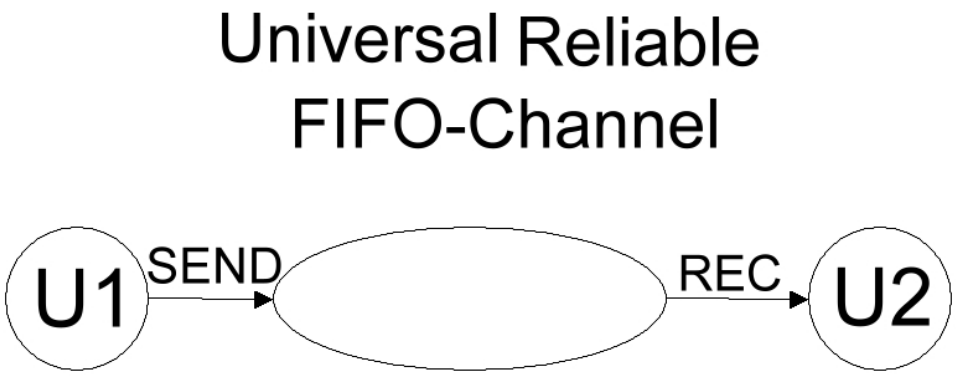
$\alpha\downarrow$P1 $\times$P2 $=$
$send_{1,2}(hello), send_{1,2}(hello), rec_{1,2}(hello), send_{1,2}(world), rec_{2,1}(world)$

fairtraces:

ein unendlicher trace α heißt fair gegenüber einem Task c $\Leftrightarrow$

- α enthält unendlich viele Vorkommen von c
  oder
- c ist unendlich oft nicht ausführbar

Korrektheit



Datenübertragung mit dem Universal Reliable FIFO-Channel

Korrektheit eines Protokolls:

- das Protokoll soll den Universal Reliable FIFO-Channel implementieren
  d.h. für alle fairen Ausführungen α des Protokolls soll gelten:
  $\alpha\downarrow \underbrace{ext(URFC)}_{\{SEND,REC\}} \in$ fairtraces(URFC)

Fehler die Lynch betrachtet:

- Prozesse werden zunächst als fehlerfrei angenommen

Channel können:

- eine Nachricht endlich oft duplizieren (finite duplication) - kurz duplication
- Nachrichten umordnen (reordering)
- endlich viele Nachrichten verlieren (limited loss) - kurz loss

Strong loss limitation(SLL):

Unendlich viele send(m) Ereignisse führen zu unendlich viele rec(m) Ereignissen.

(d.h. wird eine Nachricht oft genug versandt, so wird sie auch schließlich empfangen)

Weak loss limitation(WLL):

Unendlich viele send Ereignisse führen zu endlich vielen rec Ereignissen.

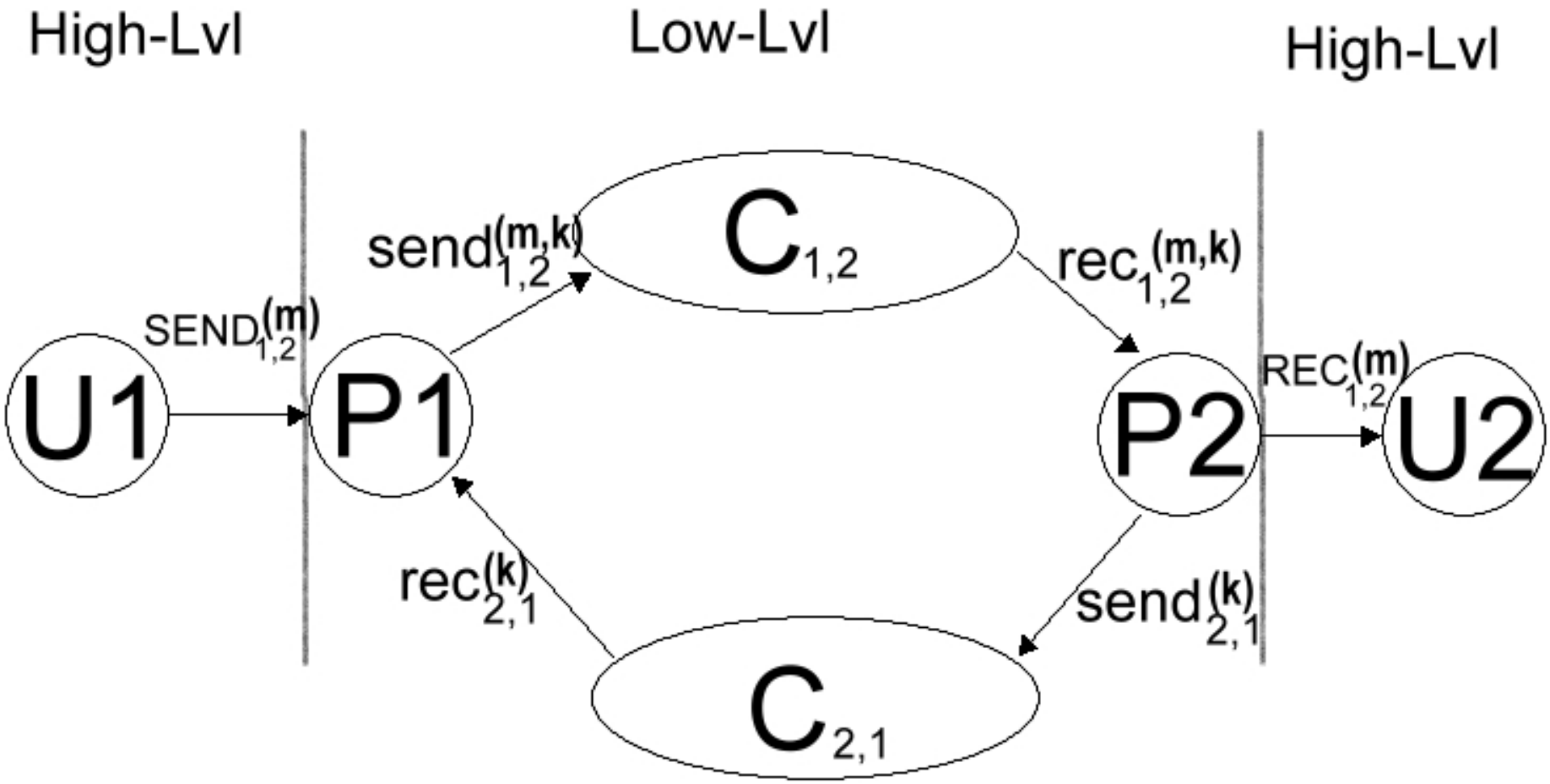
(unendlich viel versandt $\Rightarrow$ unendlich viel empfangen)

Protokolle mit unbeschränkten tags

Beispiel: Stenning Protokoll

Zugelassene Fehler in den Channels beim Stenning Protokoll :

- (finite) duplication
- reordering
- (limited) loss , WLL


High-Lvl
Low-Lvl
High-Lvl
U1
P1
P2
U2
SEND(m)1,2
send(m,k)1,2
C1,2
rec(m,k)1,2
REC(m)1,2
rec(k)2,1
send(k)2,1
C2,1

$Stenning_1$ automaton (formal)

Signature:
Input:
$SEND_{1,2}$(m), m $\in$M
$rec_{2,1}$(k), k $\in\mathbb{N}$

Output:
$send_{1,2}$(m,k), m $\in$M, k $\in\mathbb{N}$

States:
buffer, a FIFO queue of elements of M, initially empty
tag $\in\mathbb{N}$, initially 1

Transitions:
$SEND_{1,2}$(m)
Effect: add m to buffer

$rec_{2,1}$(k)
Effect:
if k=tag then
remove first element
(if any) of buffer
tag:=tag + 1

$send_{1,2}$(m,k)
Precondition: m is first on buffer
k=tag
Effect: none

Tasks: {$send_{1,2}$(m,k):m $\in$ M, k $\in\mathbb{N}$}

$Stenning_2$ automaton (formal)

Signature:

Input:
$rec_{1,2}$(m,k), m $\in$M, k $\in\mathbb{N}$

Output:
$REC_{1,2}$(m), m $\in$M
$send_{2,1}$(k), k $\in\mathbb{N}$

States:
buffer, a FIFO queue of elements of M, initially empty
tag $\in\mathbb{N}$, initially 0

Transitions:
$REC_{1,2}$(m)
Pre: m is first on buffer
Eff: remove first element of buffer

$send_{2,1}$(m)
Pre: k=tag
Eff: none

$rec_{1,2}$(m,k)
Eff:
if k=tag+1 then
add m to buffer
tag:=tag +1

Tasks: {$REC_{1,2}$(m):m $\in$ M, k $\in\mathbb{N}$} {$send_{2,1}$(k):k $\in\mathbb{N}$}

Korrektheitsbeweis des Stenning Protokoll

$C_{1,2}$, $C_{1,2}$: duplication, reordering und loss mit WLL

Zu zeigen ist:

Das Stenning Protokoll mit Channel $C_{1,2}$, $C_{2,1}$ implementiert den Universal Reliable FIFO-Channel , d.h.

Für alle fairen Ausführungen α des Stenning Protokoll gilt:

$\alpha \downarrow$\{SEND, REC \} $\in$fairtraces(urfc)

Informell kann man sagen:

tag_1: tag der Nachricht, auf deren Empfangsbestätigung P_1 wartet

tag_2: tag der Nachricht, die zuletzt von P_2 akzeptiert wurde

Warum erfolgt kein reordering der High-Level Nachrichten ?

- $buffer_1$ und $buffer_2$ sind FIFO buffer
- High-Level–Nachrichten werden in der Reihenfolge des Eingangs durchnummeriert
- Eine Nachricht wird von P_2 nur akzeptiert, wenn sie die ist, auf die P_2 wartet

Warum erfolgt keine duplication der High-Level Nachrichten ?

- High-Level-Nachrichten erhalten von P_1 einen eindeutigen Identifyer

- Nachdem P_2 eine Nachricht mit Identifyer k akzeptiert hat, wird nur eine Nachricht mit Identifyer k+1 akzeptiert

Damit ist leicht einzusehen, dass für alle fairen Ausführungen α gilt:
$\alpha \downarrow\{SEND_{1,2}, REC_{1,2}\} \in$traces(urfc)

zu zeigen: zu jedem SEND(m) erfolgt irgendwann ein REC(m)

Annahme: Es existieren$SEND_{1,2}$(m) ohne $REC_{1,2}$(m) in α

Sei k das Tag des ersten m's, dass nicht nach U2 geliefert wurde
Schreibweise: m^k

m^k kann von P_2 niemals akzeptiert worden sein $\Longrightarrow tag_2 < k$

Falls k=1, so ist m^k nach Empfang des $SEND_{1,2}(m^k)$ vorne im $buffer_1$.
Aufgrund von WLL wird m^k dann auch nach P_2 übertragen und akzeptiert $\Longrightarrow Widerspruch$

Fall k $>$ 1:
Es werden alle vorherigen k-1 Messages nach P_2 geliefert.
Schließlich gilt dann $tag_2 = k - 1$ (für immer).

Aufgrund der fairness gilt:
P_2 sended laufend $send_{2,1}$(k-1) nach P_1.

Aufgrund der WLL gilt:
P_1 empfängt irgendwann die Bestätigungung für m^{k-1}
d.h. m^{k-1} wird vom $buffer_1$ entfernt und m^k ist im $buffer_1$ ganz vorne.

Aufgrund der fairness wird laufend $send_{1,2}(m, k)$ von P_1 nach P_2 gesendet und wegen der WLL auch von P_2 empfangen und akzeptiert.
$\Longrightarrow Widerspuch$

Überleitung zum Alternating Bit Protokoll

In allen erreichbaren Zuständen des Stenning Protokoll gilt:

T_1: Eine Folge von Integers passen zu den Tags in $queue_{1,2}$ in der Reihenfolge von Ersten zum Letzten im $queue_{1,2}$

Ergibt die Sequenz ... 1,2,2

T_2: Analog T_1für $queue_{2,1}$

Setzte $T := T_2 tag_2 T_1 tag_1$
In allen erreichbaren Zuständen des Stenning Protokoll gilt:

1. T ist aufsteigend

2. die Differenz zwischen der 1. und letzten Zahl in T ist max. 1

Beispiel:

$SEND_{1,2}(a), SEND_{1,2}(b), SEND_{1,2}(c) \Longrightarrow$

T_2	tag_2	T_1	tag_1
	0		1

$send_{1,2}(a,1)...send_{1,2}(a,1) \Longrightarrow$

T_2	tag_2	T_1	tag_1
	0	1...1	1

$rec_{1,2}(a,1) \Longrightarrow$

T_2	tag_2	T_1	tag_1
	1	1...1	1

$send_{2,1}(1), rec_{2,1}(1), send_{1,2}(b,2) \Longrightarrow$

T_2	tag_2	T_1	tag_1
1	1	1...12...2	2

$rec_{1,2}(a),...,rec_{1,2}(a), rec_{1,2}(b) \Longrightarrow$

T_2	tag_2	T_1	tag_1
1	2	2...2	2

Protokolle mit beschränkten tags

Beispiel: Das Alternating Bit Protokoll

Zugelassene Fehler in den Channels beim Stenning Protokoll :

- (finite) duplication
- (limited) loss , WLL

ABP_1 automaton (formal)

Signature

Input:
$SEND_{1,2}$(m), m ∈M
$rec_{2,1}$(b), b ∈{0,1}

Output:
$send_{1,2}$(m,b), m ∈M, b ∈{0,1}

States:

buffer, a FIFO queue of elements of M, initially empty
tag ∈{0,1}, initially 1

Transitions:

$SEND_{1,2}$(m)
Effect: add m to buffer

$send_{1,2}$(m,b)
Precondition:
m is first on buffer
b=tag
Effect: none

$rec_{2,1}$(b)
Effect:
if b=tag then
remove first element
(if any) of buffer
tag:=tag+1 mod 2

ABP_2 automaton (formal)

Signature

Input:
$rec_{1,2}$(m,b),m ∈M, b ∈{0,1 }

Output:
$REC_{1,2}$(m), m ∈M
$send_{2,1}$(b), b ∈{0,1}

States:
buffer, a FiFO queue of elements of M, initially empty
tag ∈{0,1},initially 0

Transitions:
$REC_{1,2}$(m)
Pre: m is first on buffer
Eff: remove first element of buffer

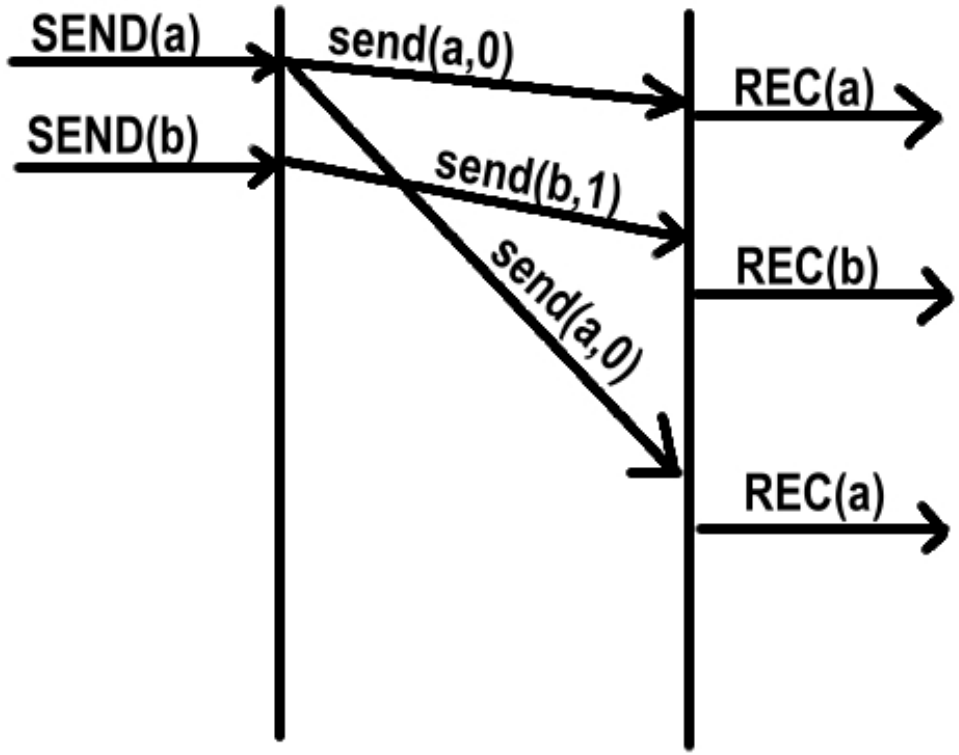
$rec_{1,2}$(m,b)
Effect:
if b ≠ tag then
add m to buffer
tag:= tag + 1 mod 2

$send_{2,1}$(b)
Precondition: b=tag
Effect: none

Hinweise zum Beweis

Beh: Das ABP mit lossy FIFO Channel ist eine Implementierung des Universal Reliable FIFO-Channel

Alternating Bit Protokoll funktioniert nicht mit reordering und duplication



reordering und duplication im ABP

Fragestellungen

Existieren bounded-tag Protokolle, die Channels mit reordering und duplication tolerieren ?

Existieren bounded-tag Protokolle, die Channels mit finite loss und reordering tolerieren ?

Es existieren keine bounded-tag Protokolle, die Channels mit reordering und duplication tolerieren.

Vorbereitung: konsistent und endlich konsistent

Eine Ausführung α von $P_1 \times P_2$ heißt konsistent mit $Q_{1,2}$, wenn

$$\alpha \downarrow \underbrace{ext(Q_{1,2})}_{\{send_{1,2}, rec_{1,2}\}} \in traces(Q_{1,2})$$

d.h. α eingeschränkt auf die Channel kann aus Channelsicht α auch wirklich vorkommen.

Eine mit $Q_{1,2}$ konsistente Ausführung α heißt endlich konsistent, falls α endlich ist.

analog für $Q_{2,1}$

M : High-Level Message Alphabet
M': Low-Level Message Alphabet

Im weiteren Verlauf gelte: M und M' sind endlich

Warum ist diese Einschränkung angemessen?
Wenn mit endlichen High-Level Alphabet M kein Protokoll existiert, dann insbesondere nicht mit unendlichem

M' endlich $\Longrightarrow$ die tags sind nicht unbounded
Kontraposition: unbounded tags $\Longrightarrow$ M' ist unendlich

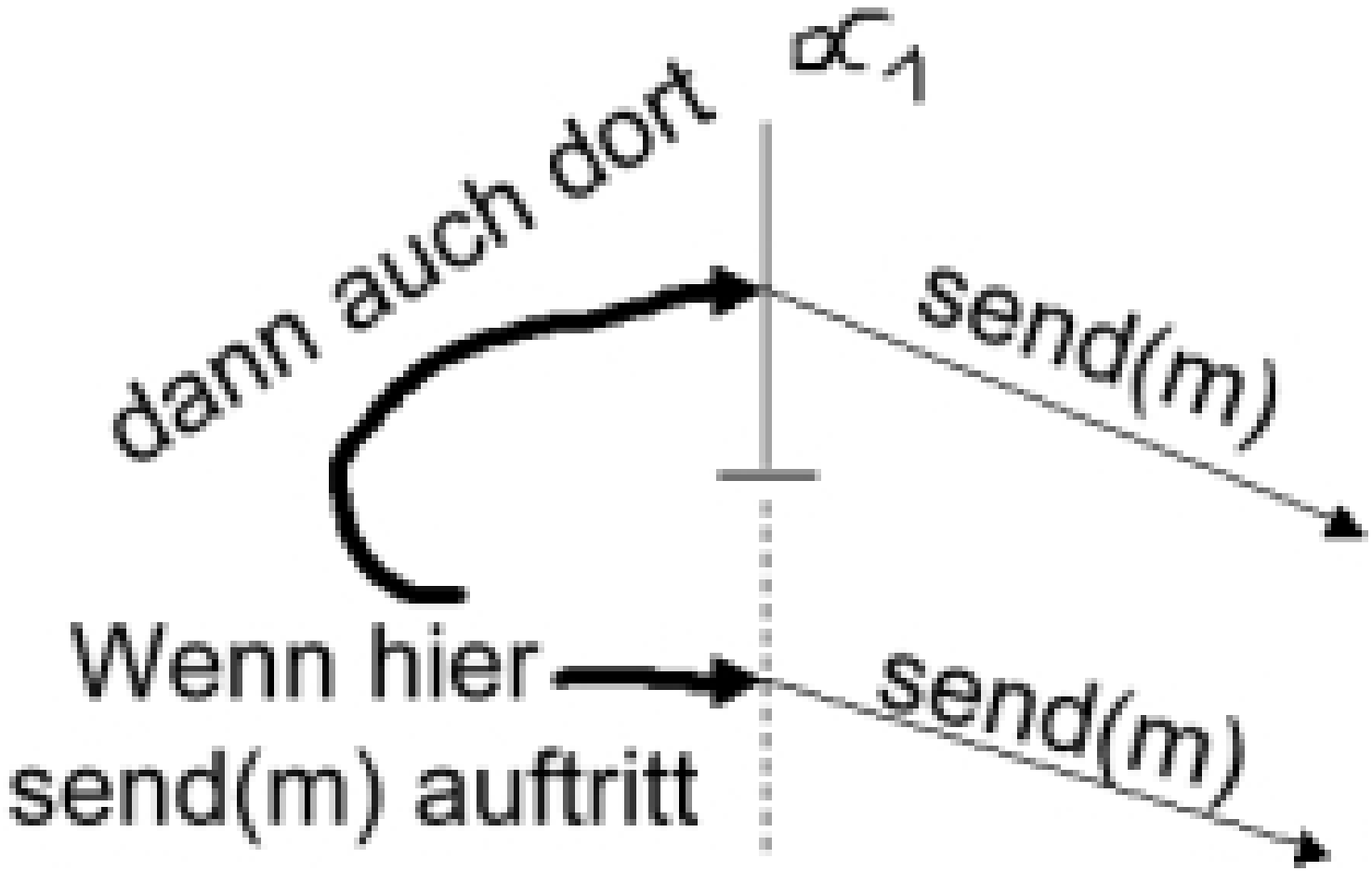
$Q_{1,2}$, $Q_{2,1}$ sind Channel bei denen reordering und duplication zugelassen sind.

Behauptung:
Es existiert kein bounded-tag Protokoll , dass den Universal Reliable FIFO-Channel implementiert und die Channel $Q_{1,2}$, $Q_{2,1}$ verwendet.

Annahme:
Es existiert eine Implementierung des Universal Reliable FIFO-Channel (P_1,P_2) mit den Channel $Q_{1,2}$ und $Q_{2,1}$, die reordering und duplication zulässt.
(d.h. f.a. konsistenten faire Ausführungen α von $P_1 \times P_2$
$\Longrightarrow \alpha \downarrow ext(urfc) \in fairtraces(urfc)$)

Man betrachte das System aus einem Startzustand heraus (gleich zu Anfang oder wenn keine neuen Ausführungen stattfinden können).

Sei α_1 eine Ausführung von $P_1 \times P_2$, endlich konsistent mit $Q_{1,2}$,$Q_{2,1}$ und so konstruiert, dass wenn man eine Fortsetztung von α_1 betrachtet, so gibt es für jedes Vorkommen eines $send_{1,2}$(m) in der Fortsetztung auch ein $send_{1,2}$(m) in α_1 .



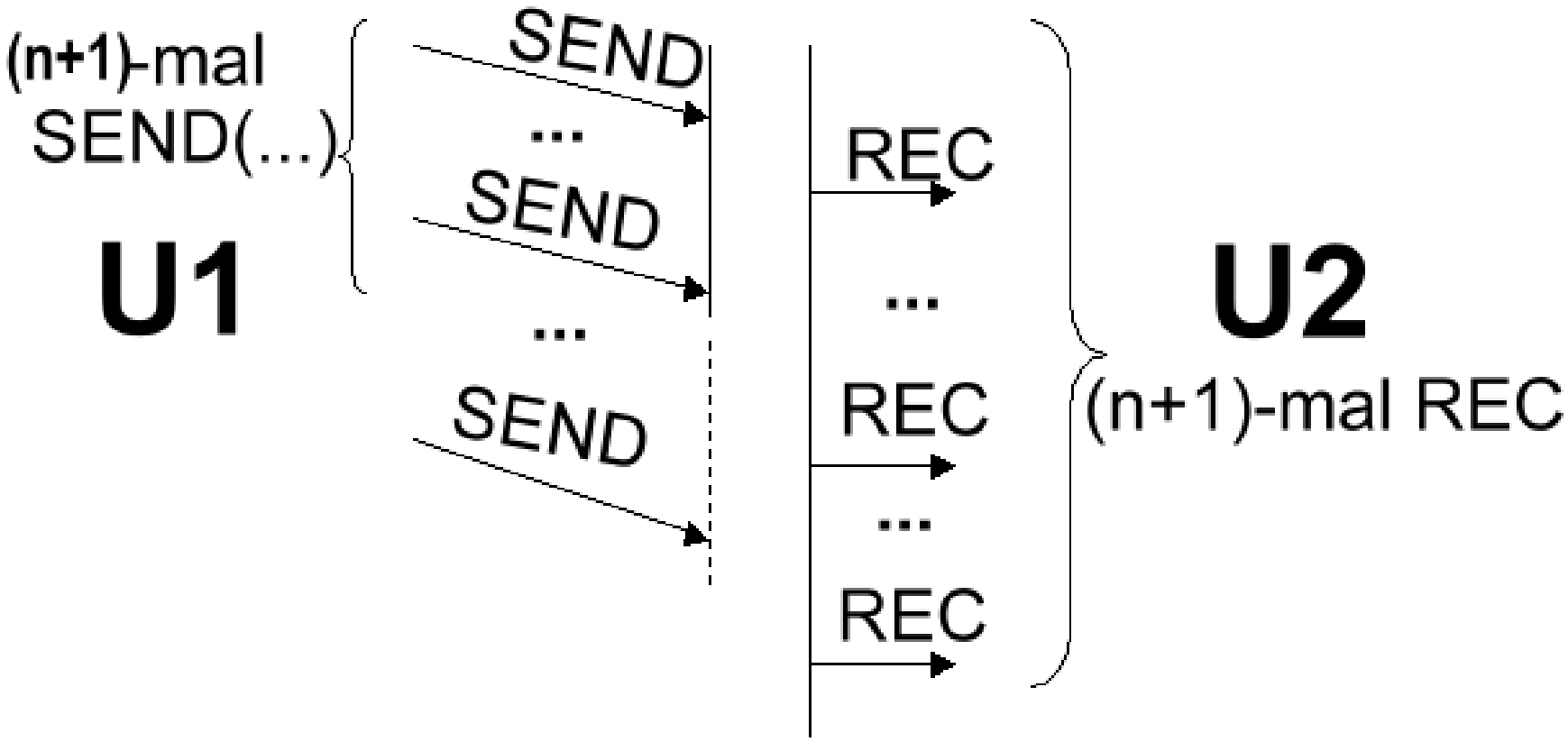
Tritt in der Erweiterung von α_1 ein $send_{1,2}$(m) auf, so auch in α_1

M' wurde als endlich angenommen, also ist die Anzahl der Konstruktionsmöglichkeiten von α_1 endlich (da α_1 endlich ist).
Informell: α_1 "alle tags sind verbraucht, so dass in der Fortsetzung von α_1 die tags wiederholt werden.

Sei n die Anzahl der Ausführungen von $SEND_{1,2}$ in α_1 .

α_2 Erweiterung von α_1 , fair, konsistent und α_2 enthalte genau ein zusätliches $SEND_{1,2}$ Ausführungen,
d.h. α_2 enthält n+1 $SEND_{1,2}$ Ausführungen.

Nach Korrektheit werden alle Nachrichten, die von U1 gesended wurden, schließlich zu U2 übertragen.
Also existieren in α_2 (n+1) Ausführungen von $REC_{1,2}$

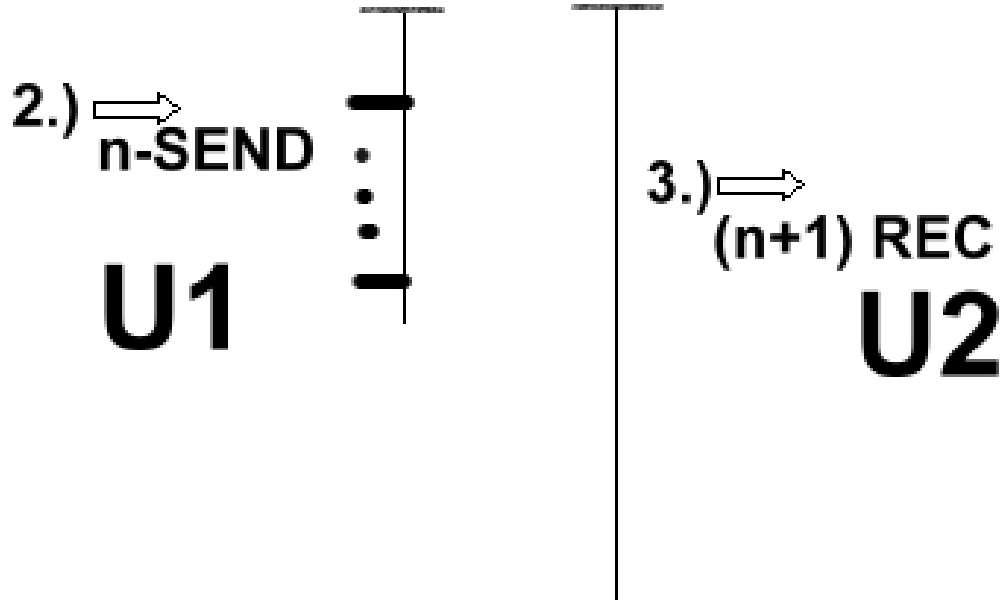


Sei α_3 nun Teil von α_1 α_2 derart, dass α_3 direkt hinter dem n+1'ten $REC_{1,2}$ aufhört

Wir konstruieren nun α_4 wie folgt:

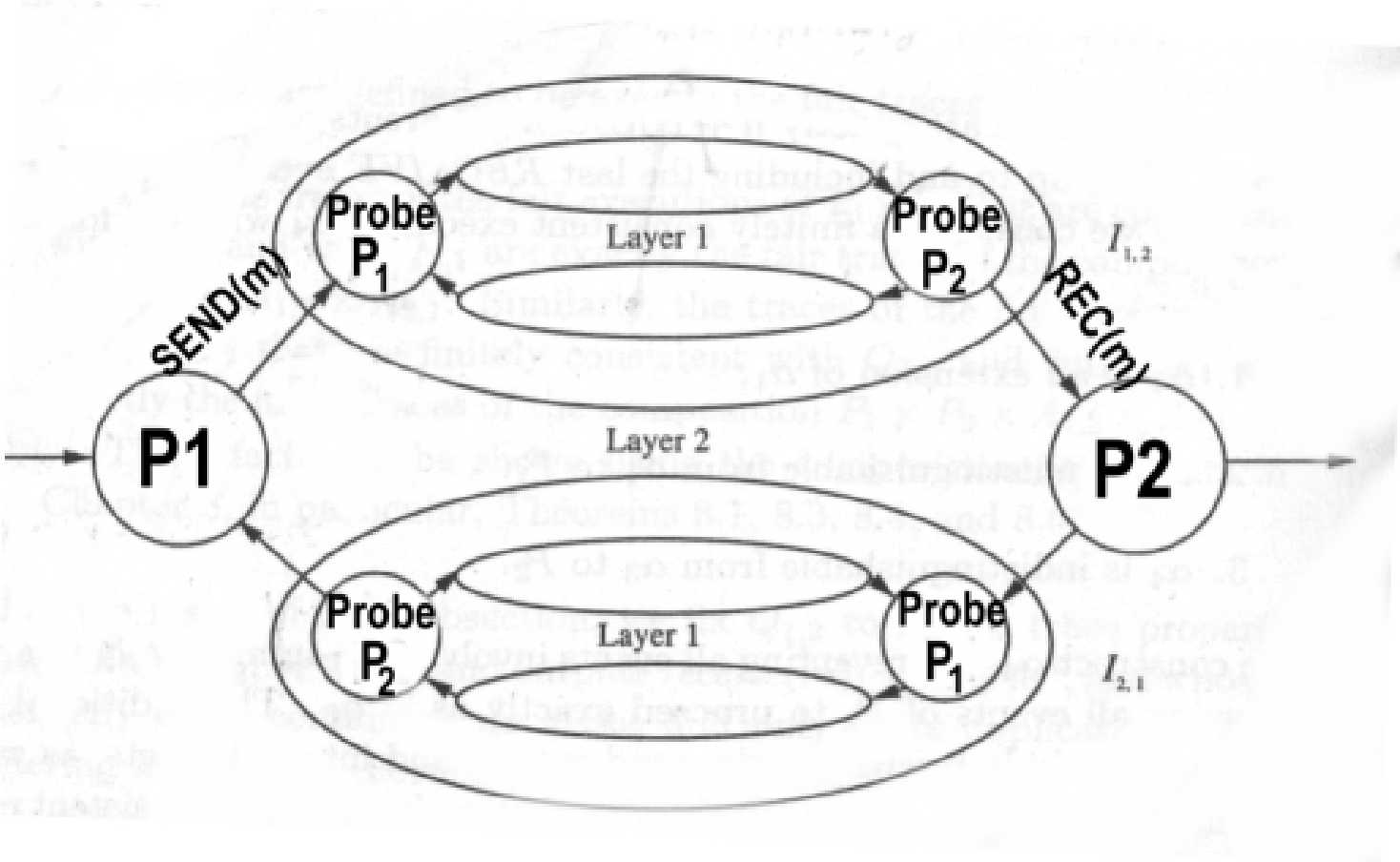
1. α_4 ist eine Erweiterung von α_1
2. $\alpha_4 \downarrow P_1 = \alpha_1 \downarrow P_1$
3. $\alpha_4 \downarrow P_2 = \alpha_3 \downarrow P_2$

α_4 enthält nun n $SEND_{1,2}$ und (n+1)- $REC_{1,2}$ Ausführungen.



Beh: Es existiert ein bounded-tag Protokoll, dass Channels mit finite loss und reordering toleriert ?

Bew: Probe-Protokoll



Layered structure of the Probe-Protokoll

Layer 1:

Channels $C_{1,2}$ und $C_{2,1}$: keine duplication, reordering, loss - SLL

Probe P1, Probe P2 mit $C_{1,2}$, $C_{2,1}$: bounded-tag Protokoll, dass duplication, loss und kein reordering erlaubt

Channels $I_{1,2}$ und $I_{2,1}$: duplication, kein reordering, loss

Layer 2:

P1, P2: Ein bounded-tag Protokoll, dass mit loss und duplication umgehen kann z.B. Alternating Bit Protokoll

Probe Layer 1, process P_1:

Signature:
Input:
$SEND_{1,2}$(m), m $\in$M
$rec_{2,1}$(probe)

Output:
$send_{1,2}$(m), m $\in$M

States:
latest $\in$ M $\cup$ {null}, initially null
unanswered $\in \mathbb{N}$, initially 0

Transitions:
$SEND_{1,2}$(m)
Effect: latest:=m

$rec_{2,1}$(probe)
Effect:
unanswered:=unanswered+1

$send_{1,2}$(m)
Precondition:
unanswered $>$ 0
m=latest
Effect:
unanswered:=unanswered-1

Probe Layer 1, process P_2:

Signature:

Input:
$rec_{1,2}$(m), m $\in$M

Output:
$REC_{1,2}$(m), m $\in$M
$send_{2,1}$(probe)

States:
pending $\in\mathbb{N}$, initially 0
old $\in\mathbb{N}$, initially 0
for every m $\in$M: cound(m) $\in\mathbb{N}$, initially 0

Transitions:

$REC_{1,2}$(m)
Precondition: count(m) $>$ old
Effect:
for all m' $\in$M do
count(m'):=0
old:=pending

$send_{2,1}$(probe)
Precondition: true
Effect: pending:=pending+1

$rec_{1,2}$(m)
Effect: count(m):=count(m)+1

Warum reicht die WLL nicht aus ? - Stichwort Multiplexen