Scheduling in a Time-Triggered Protocol With Dynamic Arbitration

Jens Chr. Lisner

lisner@informatik.uni-essen.de

ICB / University of Duisburg-Essen

Germany

Introduction

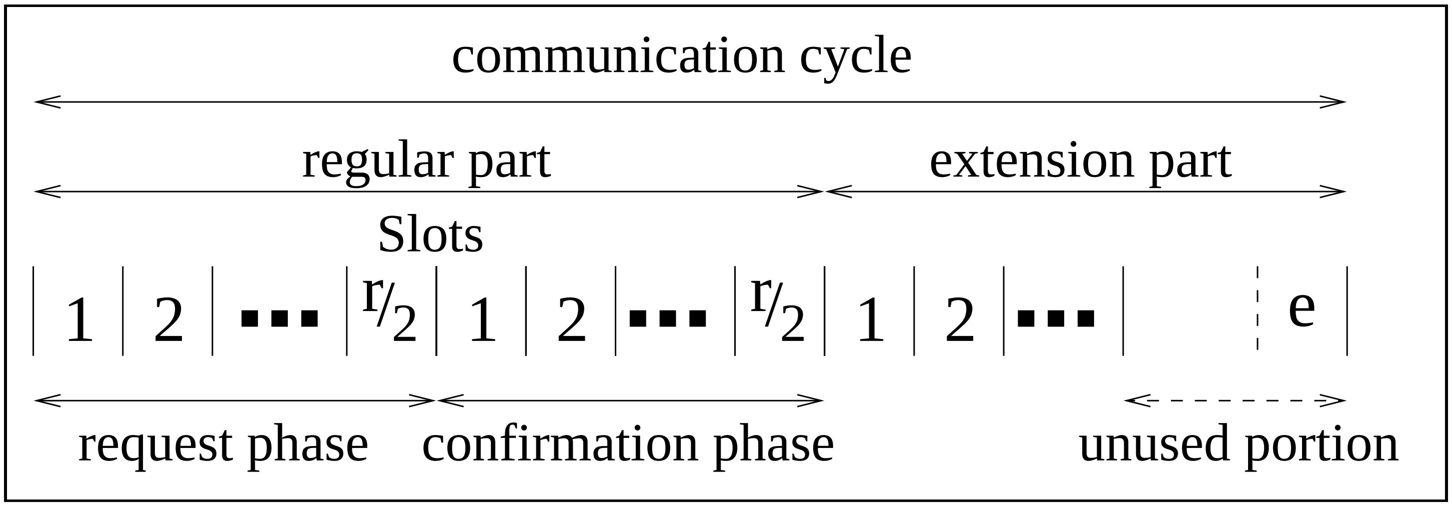
Two methods of arbitration in TDMA-based protocols

- Static arbitration
  - Schedule pre-configured
  - Slots have fixed length
  - Can be implemented in a fault-tolerant way
  - Example: TTP/C, FlexRay (“static segment”)
- Dynamic arbitration
  - Schedule determined at runtime
  - Slots have dynamic length
  - Fault-tolerant implementation difficult
  - Example: Byteflight, FlexRay (“dynamic segment”)

The *Tea* protocol aims to solve the problem of fault-tolerant dynamic arbitration.

Introduction

Tea uses a mixed-mode approach:



Regular part

Static slot length / static schedule

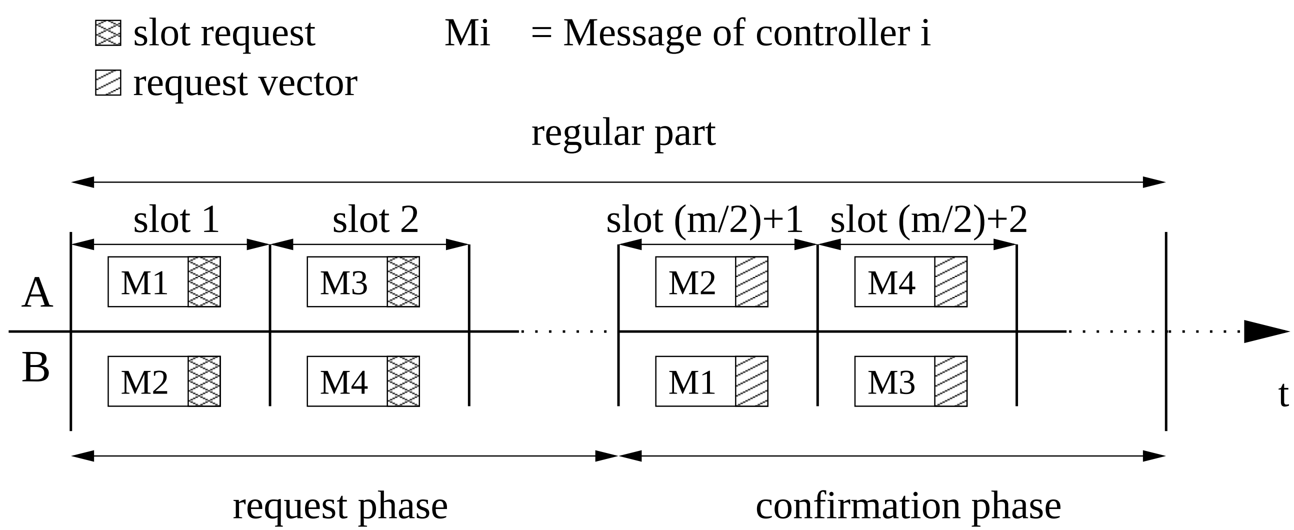
Extension part

Dynamic slot length / dynamic schedule

Every controller can request one additional slot in the extension part

Introduction

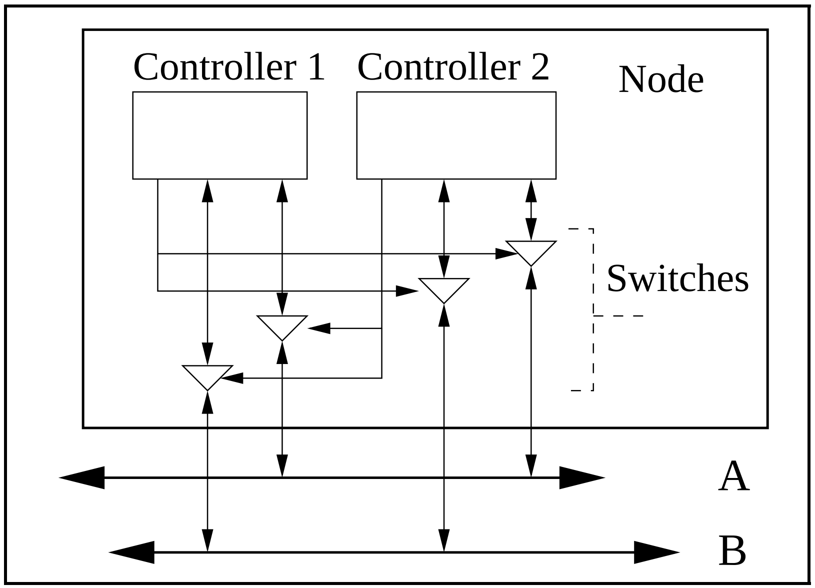
Schedule in extension provided by agreement algorithm



- Slots are shared by two controllers on two channels
- *Request phase:* Contains request bit (`request`, `no_request`)
- *Confirmation phase:* Contains vector of received requests (`request`, `no_request`, `corrupted`)
- Schedule to channels is reversed in confirmation phase

Introduction

Architecture for fault-tolerant operation



- Two completely independent controllers reside on one node
- Double broadcast channel
- Controllers guard each other by controlling the other's access to the bus
- Guaranteed fail-silent behavior

Scheduling Policies

Number of slots in the extension part is limited
$\longrightarrow$ scheduling policy required

Common criteria:

- Arrival time (cycle)
- Priority

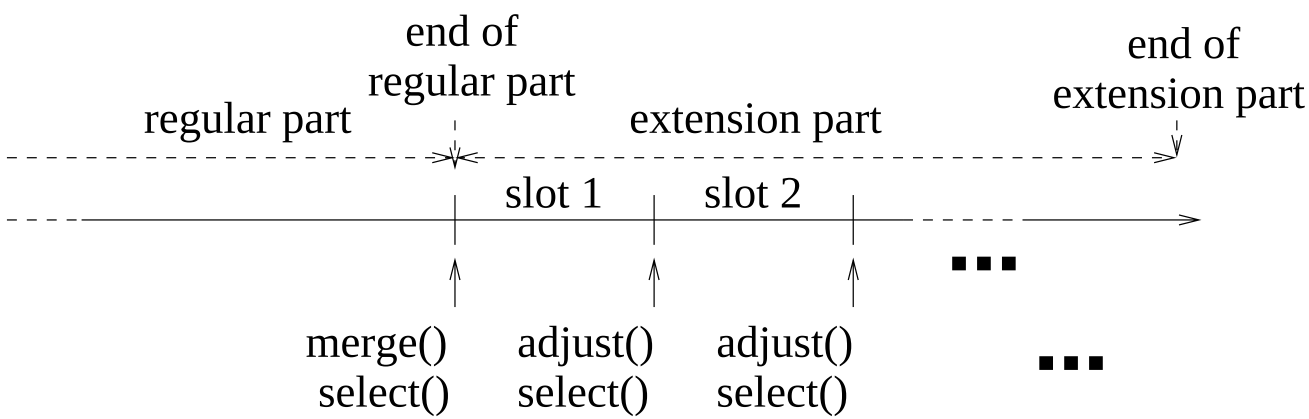
Common strategies:

- First-in-first-out
- Static priorities
- Priority-first
- FIFO-first

HW requirements should be minimized.

Basic Algorithm

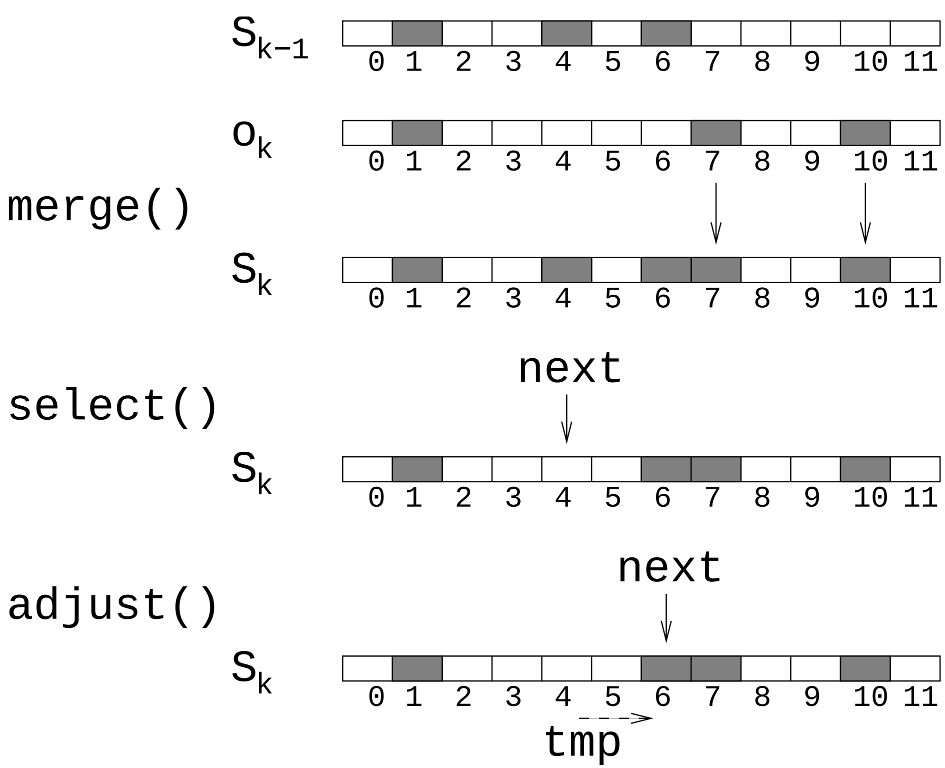
The basic scheduling algorithm consists of three subroutines.



- *merge*: Merges the vector of current requests into the vector with outstanding requests
- *select*: Selects the next controller before the start of a new slot
- *adjust*: Adjusts the index registers

Basic Algorithm

The following algorithm implements a basic round-robin-strategy



Implementation (Overview)

- FIFO $\Rightarrow$ by changing *merge*
- Priority $\Rightarrow$ by changing *select*
- FIFO-first $\Rightarrow$ by changing *merge* and *select*
- Priority-first $\Rightarrow$ by changing *merge* and *select*

HW requirements for registers in bit:

- $c + 3\lceil log_2 c \rceil$ for priority-first
- $c + 2\lceil log_2 c \rceil$ in all other cases

Controller Faults

- Neighbor prevents bus access of faulty controller
- Empty slots are possible, but can be ignored
- Input to scheduling algorithm is the value `unknown`

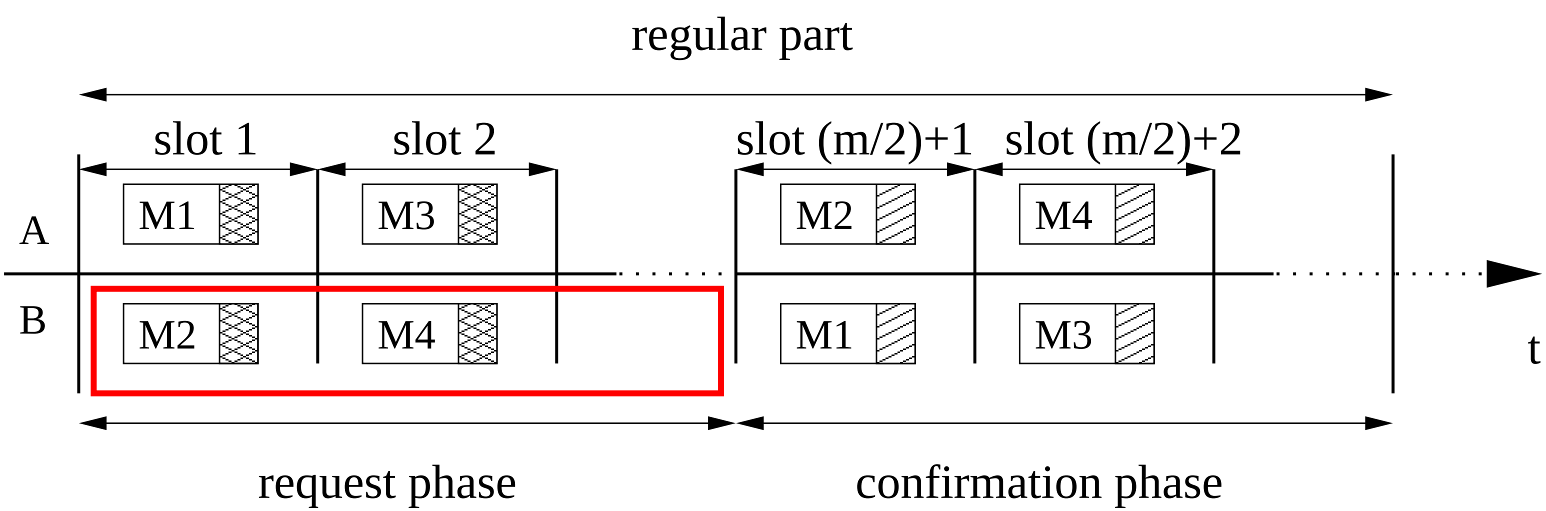
  Possible solutions:
  - Count as `no_request`: Clear the respective bit if set
  - Count as `request`: Set the respective bit if allowed
  - Leave bit unaffected

    (best solution in connection with channel faults)
- A faulty controller can block a fault-free neighbour

Channel Faults

Problem:

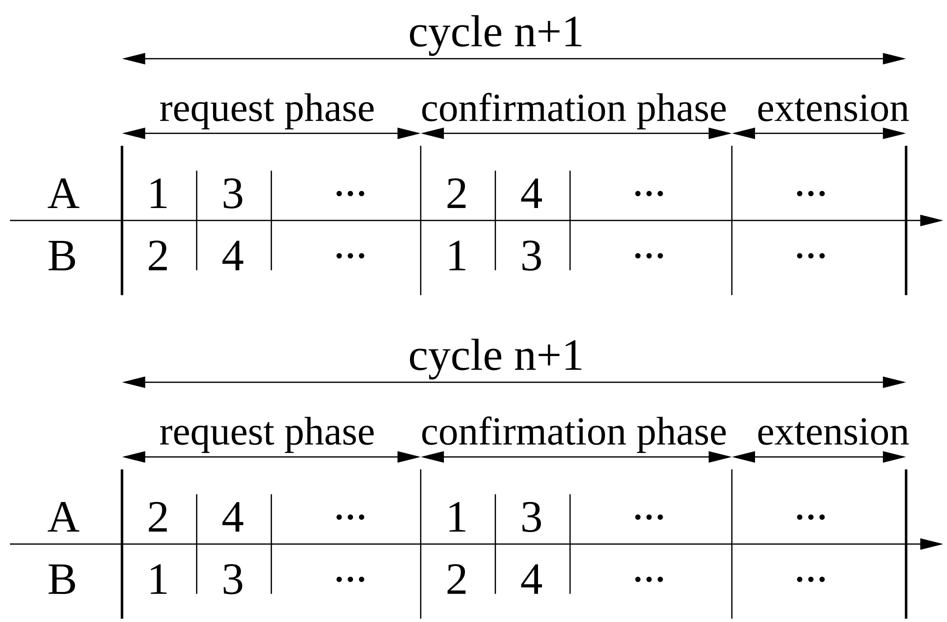
Channel faults may lead to the value `unknown` for requests of *fault-free* controllers



Channel Faults

Solution 1: **Double cycle**

Reverse schedule of controllers in the regular part every two cycles

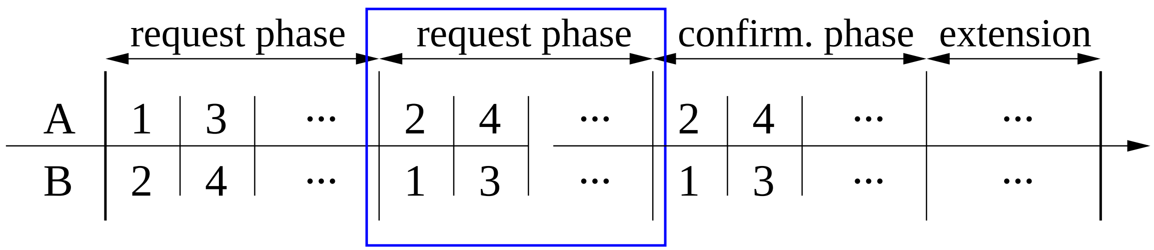


- Fault-free controllers can successfully request a slot within a double-cycle
- Can cause further delays

Channel Faults

Solution 2: **Double request phase**

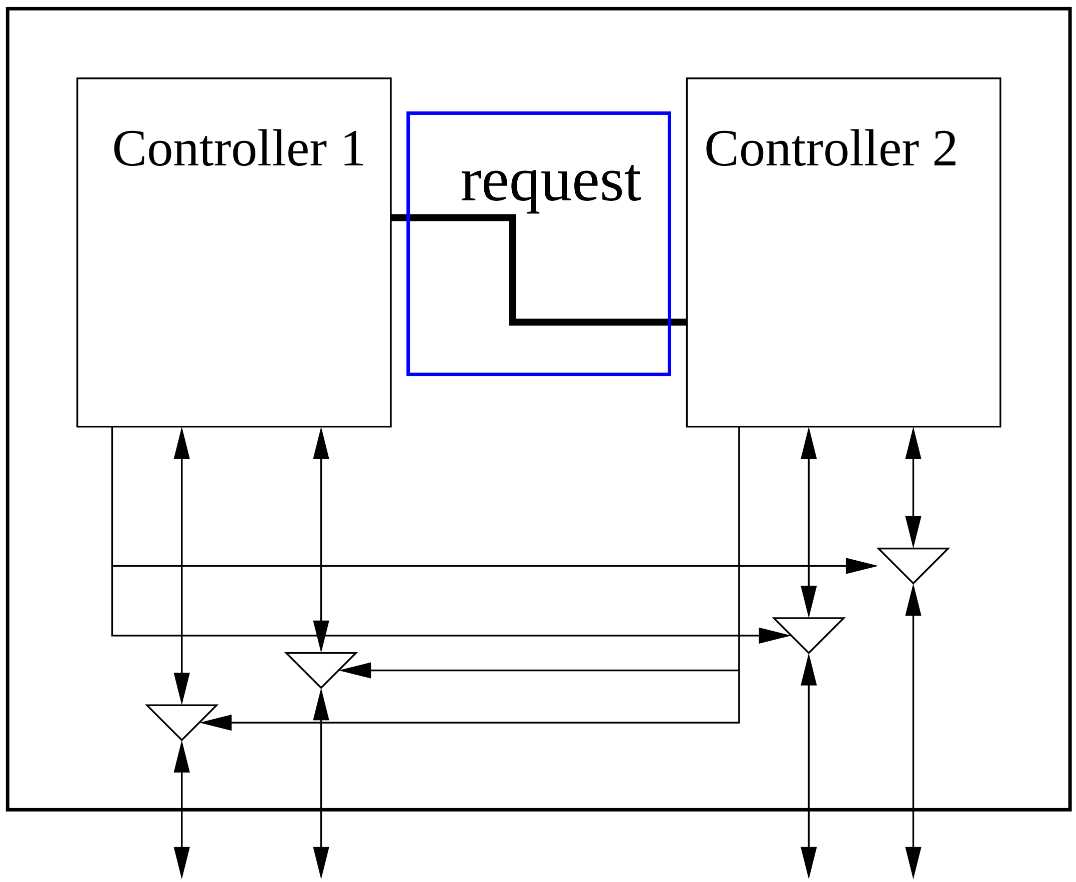
Reverse schedule of controllers in two consecutive request phases



- Request of fault-free controllers are guaranteed within a cycle
- Cycle length grows by $\frac{c}{2}$ static slots permanently

Channel Faults

Solution 3: **Additional link between both controllers in a node**



Channel Faults

Solution 3: **Additional link between both controllers in a node**

- Controller must also provide request for neighbor (extra bit necessary in request phase)
- Both controllers must be scheduled for different channels
- Request of fault-free controllers are guaranteed within a cycle
- No need to extend cycle

Conclusion

- A fault-tolerant solution for dynamic allocation in time-triggered protocols is provided by the *Tea* protocol
- Fault-tolerance can be assured
- Dynamic allocation requires dynamic scheduling
- Well known policies are available with low effort in hardware registers
- Requests can be guaranteed in case of channel faults
- Requests cannot be guaranteed for a fault-free neighbor of a faulty controller