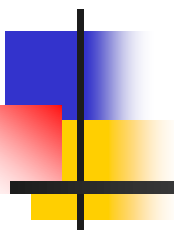


On Priority Assignment for Controller Area Network (CAN) when some Message Identifiers are Fixed

A decorative graphic on the left side of the slide, consisting of a vertical black line intersecting a horizontal black line. To the left of the intersection are three overlapping squares: a blue one on top, a red one on the left, and a yellow one on the bottom.

Robert I. Davis^{1,2}, Alan Burns¹, Victor Pollex³, Frank Slomka³

¹Real-Time Systems Research Group, University of York, UK

²AOSTE Team, INRIA, Paris-Rocquencourt, France

³Institute of Embedded / Real-Time Systems, Ulm University, Germany

A decorative graphic consisting of overlapping yellow, red, and blue squares with a black crosshair.

Outline

- Background to Controller Area Network (CAN)
- Two priority assignment problems **P1** and **P2**
- Recap of Scheduling model, analysis and priority assignment
- Simple solution to problem **P1**
- Why the simple solution doesn't work for problem **P2**
 - Counter example
- Solutions to problem **P2** with some constraints
- Case study – some experimental results
- Summary and conclusions

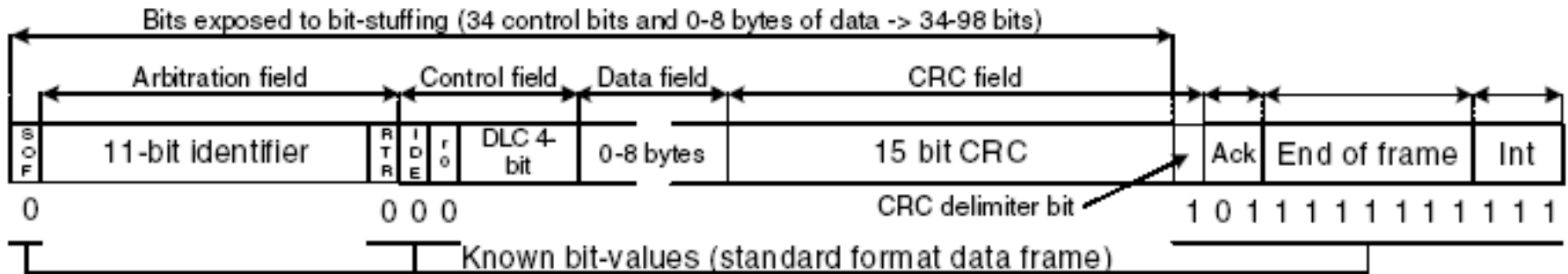
A decorative graphic on the left side of the slide, consisting of overlapping yellow, red, and blue squares with a black crosshair.

CAN Background

- Controller Area Network (CAN)
 - Simple, robust and efficient serial communications bus for in-vehicle networks
 - Average family car now has approx 25-35 Electronic Control Units (ECUs) connected via CAN
 - Today almost every new car sold uses CAN

- Information on CAN
 - Used to communicate 1000s of small signals packed into 100s of messages
 - Real-time constraints on signal transmission \sim 10ms to 1 sec

CAN Protocol: Data Frame Format



Key point: Message Identifier determines priority for access to bus (11-bit or 29-bit identifiers for messages which must be unique)

A decorative graphic on the left side of the slide, consisting of overlapping yellow, red, and blue squares with a black crosshair.

CAN Scheduling and Analysis

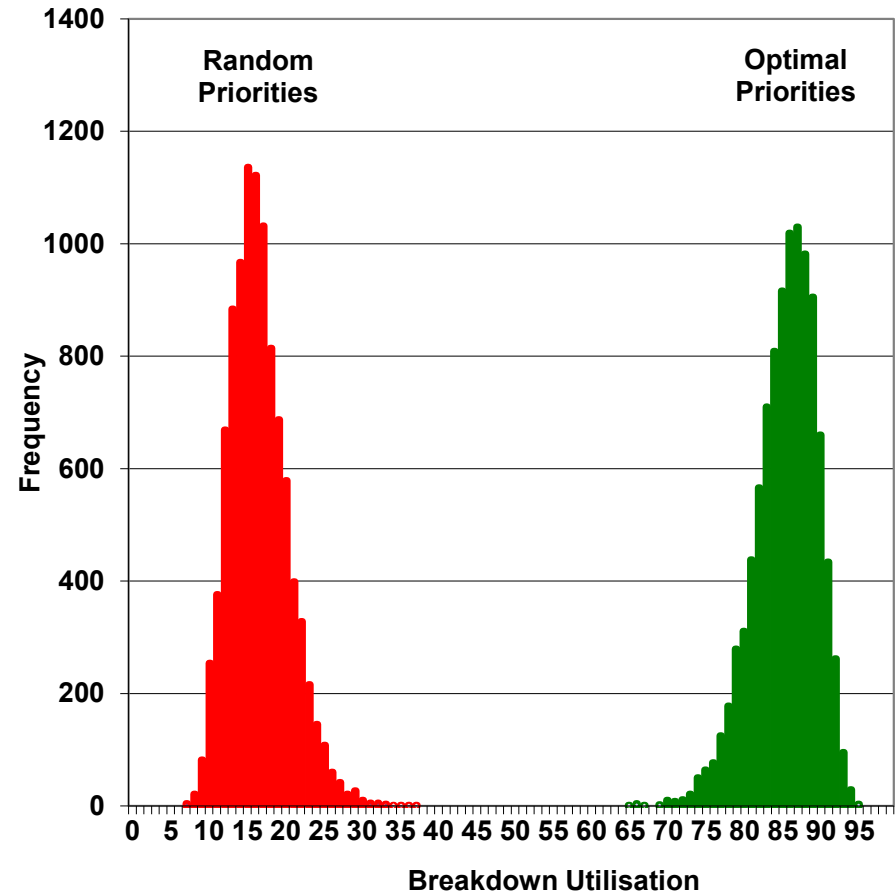
- CAN Scheduling
 - Messages compete for access to the bus based on message ID (priority)
 - With each node implementing a priority queue, network can be modelled as if there was a single global queue
 - Once a message starts transmission it cannot be pre-empted
 - Resembles single processor fixed priority non-pre-emptive scheduling

- CAN Schedulability Analysis
 - Derived by Tindell in 1994
 - Calculates worst-case response times of all CAN messages
 - Used to check if all messages meet their deadlines in the worst-case
 - Corrected in 2007

Priority assignment is important!

- Example: CAN
 - Typical automotive config:
 - 80 messages
 - 10ms -1s periods
 - All priority queues
 - x10,000 message sets

- Breakdown utilisation
 - Scale bus speed to find util. at which deadlines are missed
80% v 30% or less



A decorative graphic on the left side of the slide, consisting of overlapping yellow, red, and blue squares with a black crosshair.

Priority assignment in practice

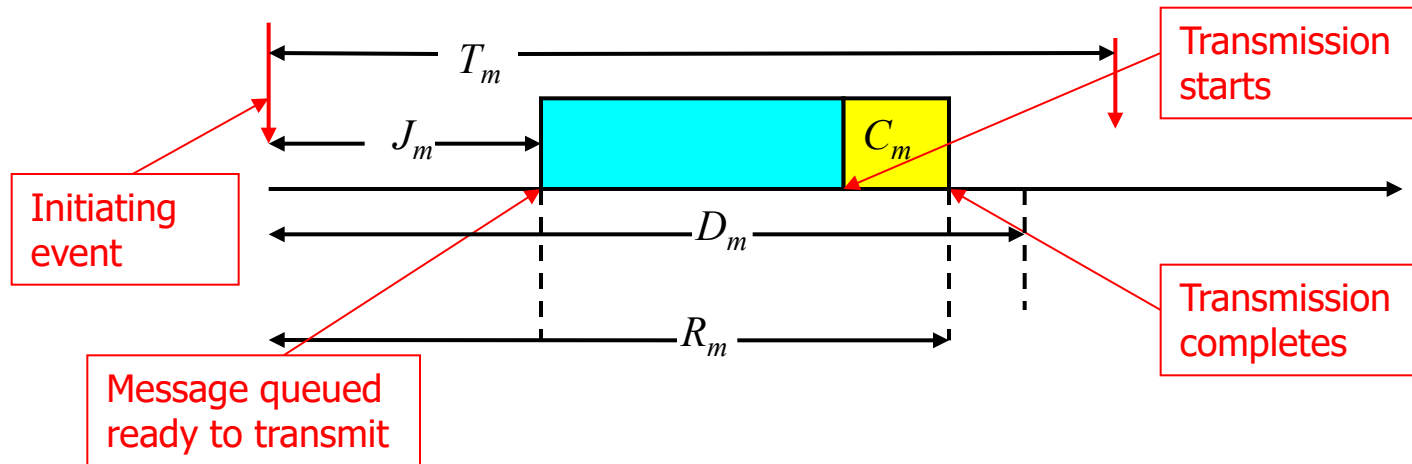
- Legacy systems
 - Very rarely (if ever) a clean sheet new design
 - Networks composed of some existing ECUs and some new ECUs
 - Identifiers of some messages may be fixed

- Priority assignment problem
 - How to assign the relative priorities of the new messages among the fixed ones

- Two variants of the problem:
 - **P1**: gaps between the identifiers of fixed messages are all large enough to accommodate all of the new messages (large gaps)
 - **P2**: the gaps aren't sufficient (small gaps)

These two problems are the focus of the talk

System Model



- Each CAN message has a:
 - Unique priority m (identifier)
 - Maximum transmission time C_m
 - Minimum inter-arrival time or period T_m
 - Deadline D_m
 - Maximum queuing jitter J_m
- Compute:
 - Worst-case queuing delay w_m
 - Worst-case response time

$$R_m = J_m + w_m + C_m$$
 - Compare with deadline

A decorative graphic on the left side of the slide, consisting of overlapping yellow, red, and blue squares with a black crosshair.

Optimal Priority Assignment

Definition: **Optimal Priority Assignment**

For a given system model, a priority assignment policy P is referred to as **optimal** if there are no systems, compliant with the model, that are schedulable using another priority assignment policy that are not also schedulable using policy P .

according to the test

according to the test

An optimal priority assignment policy can schedule any system that can be scheduled using any other priority assignment

May also consider priority assignment policies that are optimal with respect to a specific (sufficient) schedulability test

A decorative graphic on the left side of the slide, consisting of overlapping yellow, red, and blue squares with a black crosshair.

Robust Priority Assignment

- Drawback of (greedy) Optimal Priority Assignment
 - Arbitrary choice of schedulable messages at each priority level
 - May leave the system only just schedulable – i.e fragile not robust to minor changes
- In practice messages may be subject to **additional interference**
 $E(\alpha, w, i)$ function describes this
 - E.g. Interference on the bus causing errors and message re-transmission
- Want a **robust priority ordering**, able to tolerate the maximum amount of **additional interference**

A decorative graphic on the left side of the slide, consisting of overlapping yellow, red, and blue squares with a black crosshair.

Robust Priority Assignment

- Definition: **Robust Priority Assignment**

(with an additional interference function $E(\alpha, w, i)$)

For a given system model and additional interference function, a priority assignment policy P is referred to as **robust** if there are no systems, compliant with the system model, that are schedulable and can tolerate additional interference characterized by a scaling factor α using another priority assignment policy Q that are not also schedulable and can tolerate additional interference characterized by the same or larger scaling factor using priority assignment policy P .

Of all feasible priority assignments, the robust priority assignment tolerates the most additional interference (largest α)

Sufficient Schedulability Test **S1**

- Blocking $B_m = \max_{k \in lp(m)} (C_k)$
- Queuing delay

$$w_m^{n+1} = E_m(w_m^n + C_m) + \max(B_m, C_m) + \sum_{\forall k \in hp(m)} \left\lceil \frac{w_m^n + J_k + \tau_{bit}}{T_k} \right\rceil C_k$$

- Response time $R_m = J_m + w_m + C_m$

Robust Priority Assignment (RPA) for problem **P1**

- **P1**: Gaps are large enough to accommodate all freely assignable messages
- Simple solution based on Audsley's (greedy) OPA algorithm
- Works with Exact **E1** and sufficient **S1, S2** tests

```
for each logical priority level  $k$ , lowest first
{
    for each unassigned msg  $m$  from the set of NEW msgs
    and the lowest priority unassigned fixed msg
    {
        determine the largest value of  $\alpha$  for which msg  $m$ 
        is schedulable at priority  $k$  assuming that all
        unassigned msgs have higher priorities
    }
    if none of the above msgs are schedulable at priority  $k$ 
    {
        return unschedulable
    }
    else
    {
        assign the schedulable msg that tolerates the
        max  $\alpha$  at logical priority  $k$  to logical priority  $k$ 
    }
}
return schedulable
```

Why does this work for problem **P1**?

OPA (&RPA) algorithm applicability

- OPA algorithm provides optimal priority assignment w.r.t. any schedulability test S for fixed priority scheduling provided that three conditions are met...
 - Condition 1:** Schedulability of a task may, according to the test, be dependent on the set of higher priority tasks, but not on their relative priority ordering
 - Condition 2:** Schedulability of a task may, according to the test, be dependent on the set of lower priority tasks, but not on their relative priority ordering
 - Condition 3:** When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to the test, if it was previously deemed schedulable at the lower priority

Tests meeting these conditions referred to as **OPA-compatible**

Powerful idea as we have said very little
about the actual schedulability test
hence broad applicability

A decorative graphic on the left side of the slide, consisting of overlapping yellow, red, and blue squares with a black crosshair.

What about problem **P2**?

- **P2**: Recall some gaps are not large enough to accommodate all freely assignable messages
- Assume when constraint is broken => unschedulable
 - **P1** and **P2** can ignore cases where fixed ID messages have priorities swapped as this cannot give a valid and so “schedulable” ordering
 - **P2** swapping fixed and new messages in the logical ordering can violate the constraint (gap size) => the three conditions are **NOT** met

Suspicion that a greedy approach won't always work

Prior work by Schmidt [23] on Robust Priority Assignment for problem **P2** uses a greedy approach...

Counter example for P2 (Exact test E1)

- Assume just 4 priorities possible
- Message parameters
- Computed values of α (additional interference tolerated)

Task	C	T	D
M_A	125	1000	750
M_F	125	1000	350
M_B	125	1000	750
M_C	75	1000	1000

Priority	Message			
	M_A	M_F	M_B	M_C
4	300	NS	300	550
3	300	NS	300	-
2		NS	375	-
1		-	-	-

- Exact test **E1**
 - Priority 4: All messages have WCRT of 450
 - Priority 3: Again all messages have WCRT of 450
 - Priority 2: Forced to assign fixed message M_F but it's not schedulable!
 - Algorithm gives up => unschedulable

Order M_C (highest), M_F , M_A , M_B is schedulable

Response times of 200, 325, 450, 450 respectively ($\alpha = 800, 25, 300, 300$)

Counter example for P2 (Sufficient test S1)

- Assume just 4 priorities possible
- Message parameters
- Computed values of α (additional interference tolerated)

Task	C	T	D
M_A	125	1000	750
M_F	125	1000	350
M_B	125	1000	750
M_C	75	1000	1000

Priority	Message			
	M_A	M_F	M_B	M_C
4	175	NS	175	475
3	250	NS	250	-
2		NS	375	-
1		-	-	-

- Sufficient test **S1**
 - Same issues slightly different numbers
 - Order M_C (highest), M_F , M_A , M_B is schedulable**
 - Response times of 200, 325, 450, 525 respectively ($\alpha = 800, 25, 300, 175$)**
- Real systems
 - More priority levels but same problem arises**

A decorative graphic on the left side of the slide, consisting of overlapping yellow, red, and blue squares with a black crosshair.

Partial solution to Problem **P2**

- Make an approximation **A1**:
 - In the analysis consider all messages to have the same max length (need not actually be the case)
- With test **S1** and approximation **A1** we can prove (in the paper) that if a schedulable ordering exists there are optimal and robust orderings where all freely assignable messages are in D-J monotonic priority order (partial order)
- Problem reduced to finding a merge between D-JMPO for new messages and the fixed order of existing (fixed) messages

Many possible merges are possible preserving order of both sets

OPA Algorithm for Problem P2

- Builds a logical priority order of new and fixed messages
- Prefer to assign new messages to the lowest possible priorities
- Proved optimal w.r.t. test **S1** and approx. **A1** in the paper
- Complexity $O(n^2)$ single message schedulability tests

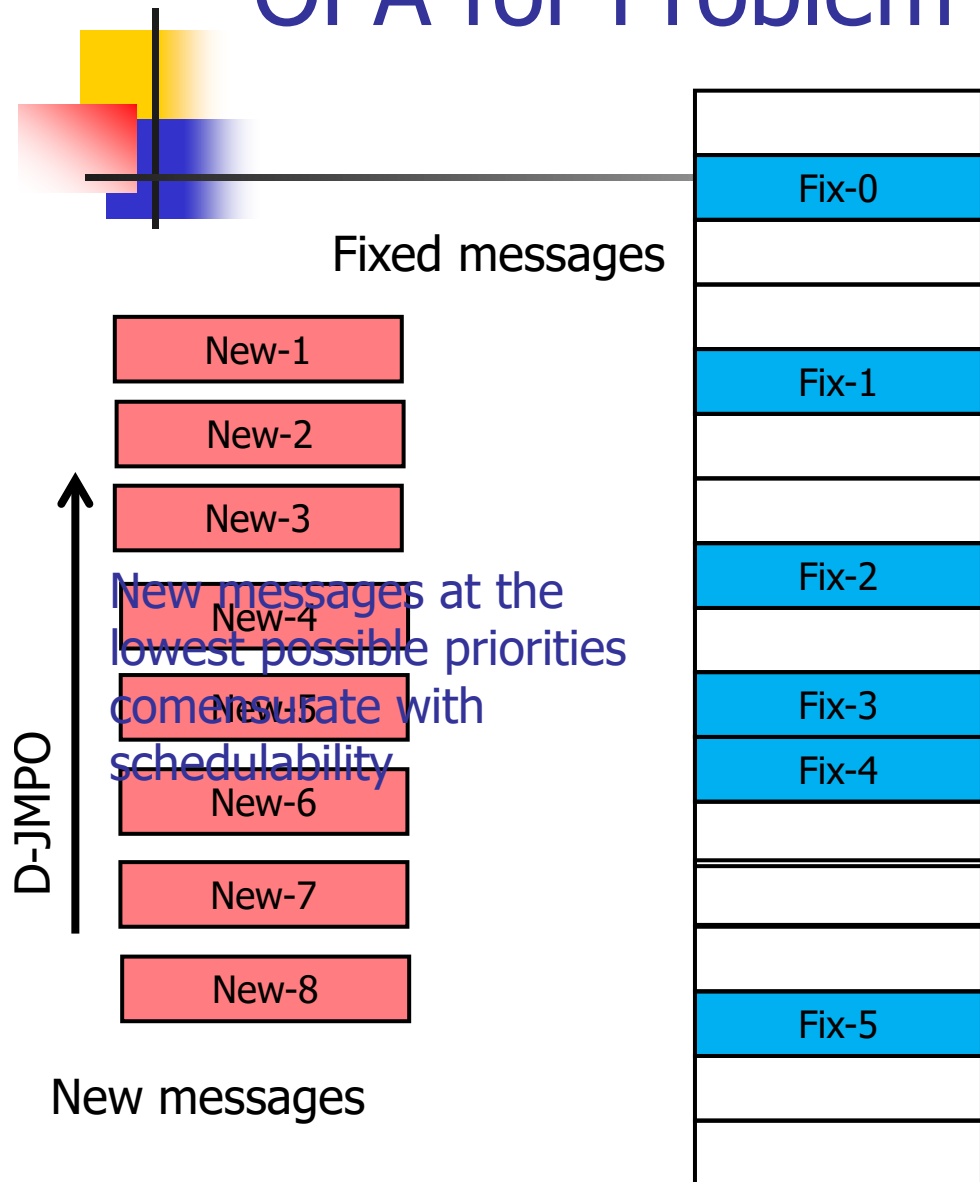
```

All messages in  $N$  and  $M$  are assumed to be unassigned
Sort the new messages in  $N$  into DJMPO
Sort the fixed messages in  $M$  by their IDs, highest first
NextID = Highest_CAN_ID()
while (there are unassigned messages) {
     $m$  = unassigned message from  $M$  with the highest ID
    if (fixed ID of  $m$  is NextID) {
        if ( $m$  is schedulable with all unassigned messages
            assumed to have higher priorities) {
            assign message  $m$  with ID = NextID.
        }
        else {
            return unschedulable
        }
    }
    else {
         $k$  = unassigned message from  $N$  with the largest value of
            deadline minus jitter
        if ( $k$  is schedulable with all unassigned messages
            assumed to have higher priorities) {
            assign message  $k$  with ID = NextID.
        }
        else if ( $m$  is schedulable with all unassigned messages
            assumed to have higher priorities) {
            NextID = ID of message  $m$ 
            assign message  $m$  with ID = NextID.
        }
        else {
            return unschedulable
        }
    }
    NextID = Next_higher_CAN_ID (NextID)
}
return schedulable

```

Algorithm 2: OPA for problem P2
with fixed message IDs and small gaps

OPA for Problem P2 example



```

All messages in  $N$  and  $M$  are assumed to be unassigned
Sort the new messages in  $N$  into DJMPO
Sort the fixed messages in  $M$  by their IDs, highest first
NextID = Highest_CAN_ID()
while (there are unassigned messages) {
   $m$  = unassigned message from  $M$  with the highest ID
  if (fixed ID of  $m$  is NextID) {
    if ( $m$  is schedulable with all unassigned messages
        assumed to have higher priorities) {
      assign message  $m$  with ID = NextID.
    }
    else {
      return unschedulable
    }
  }
  else {
     $k$  = unassigned message from  $N$  with the largest value of
        deadline minus jitter
    if ( $k$  is schedulable with all unassigned messages
        assumed to have higher priorities) {
      assign message  $k$  with ID = NextID.
    }
    else if ( $m$  is schedulable with all unassigned messages
        assumed to have higher priorities) {
      NextID = ID of message  $m$ 
      assign message  $m$  with ID = NextID.
    }
    else {
      return unschedulable
    }
  }
  NextID = Next_higher_CAN_ID (NextID)
}
return schedulable

```

Algorithm 2: OPA for problem P2 with fixed message IDs and small gaps

RPA Algorithm for Problem P2

- OPA for P2
 - Suffers from the classic OPA problem – the assignment may be fragile – only just schedulable
- RPA Algorithm for P2
- Intuition
 - Start with OPA assignment (new messages at lowest priorities)
 - Identify message that constrains robustness to smallest α
 - Move it up in logical priority above next higher priority fixed message
 - Ripple all higher priority new messages up to maintain D-J partial order
 - Repeat until no longer possible to increase robustness (e.g. constrained by spacing or a fixed message)

```

CurrentOrder = Priority order from Algorithm 2 with  $\alpha = 0$ .
if (CurrentOrder not valid) { // no schedulable ordering
    return unschedulable
}
RobustOrder = CurrentOrder
Determine max value of  $\alpha$  for each message in CurrentOrder
MaxAlpha = min  $\alpha$  over all messages in CurrentOrder
while (true) {
    x = message with min  $\alpha$  in CurrentOrder // break ties in
    // favour of fixed messages, then lower priority messages
    if (x is a fixed message) {
        return schedulable
    }
    if (there is no fixed message at a higher priority than x in
        CurrentOrder) {
        return schedulable
    }
    NewOrder = RippleUpwards(x, CurrentOrder)
    if (NewOrder not valid) { // i.e. cannot create a valid
        // NewOrder due to no space at highest physical priorities
        return schedulable
    }
    Determine max value of  $\alpha$  for each message in NewOrder
    NewMaxAlpha = min  $\alpha$  over all messages in NewOrder
    if (NewMaxAlpha  $\geq$  MaxAlpha) {
        RobustOrder = NewOrder
        MaxAlpha = NewMaxAlpha
    }
    CurrentOrder = NewOrder
}

```

Algorithm 3: RPA for problem P2
with fixed message IDs and small gaps

RPA for Problem P2

Fixed message

New message



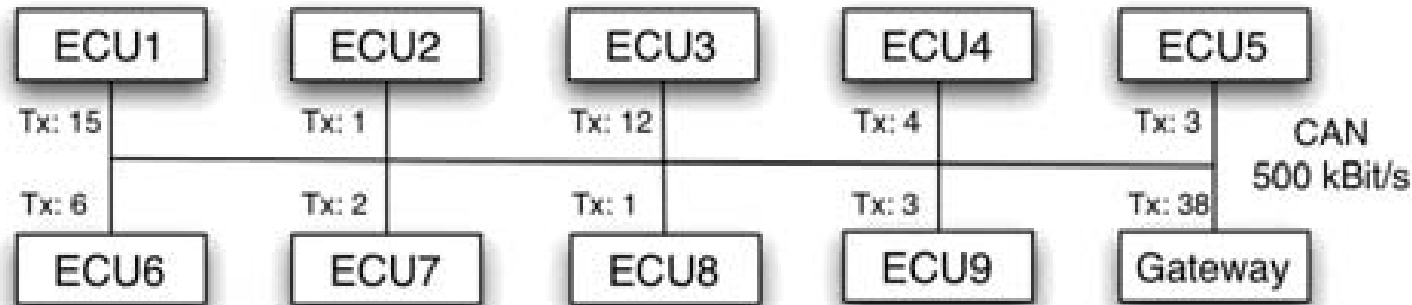
```

CurrentOrder = Priority order from Algorithm 2 with  $\alpha = 0$ .
if (CurrentOrder not valid) { // no schedulable ordering
    return unschedulable
}
RobustOrder = CurrentOrder
Determine max value of  $\alpha$  for each message in CurrentOrder
MaxAlpha = min  $\alpha$  over all messages in CurrentOrder
while (true) {
    x = message with min  $\alpha$  in CurrentOrder // break ties in
    // favour of fixed messages, then lower priority messages
    if (x is a fixed message) {
        return schedulable
    }
    if (there is no fixed message at a higher priority than x in
        CurrentOrder) {
        return schedulable
    }
    NewOrder = RippleUpwards(x, CurrentOrder)
    if (NewOrder not valid) { // i.e. cannot create a valid
        // NewOrder due to no space at highest physical priorities
        return schedulable
    }
    Determine max value of  $\alpha$  for each message in NewOrder
    NewMaxAlpha = min  $\alpha$  over all messages in NewOrder
    if (NewMaxAlpha  $\geq$  MaxAlpha) {
        RobustOrder = NewOrder
        MaxAlpha = NewMaxAlpha
    }
    CurrentOrder = NewOrder
}
    
```

Algorithm 3: RPA for problem P2 with fixed message IDs and small gaps

Case Study: Automotive

- Specific configuration from an automotive supplier
- 10 ECUs, 85 messages, nominally 500 Kbit/s



- Message periods in range 10 to 1000 ms
- 1-8 data bytes in each message (60 of max length)

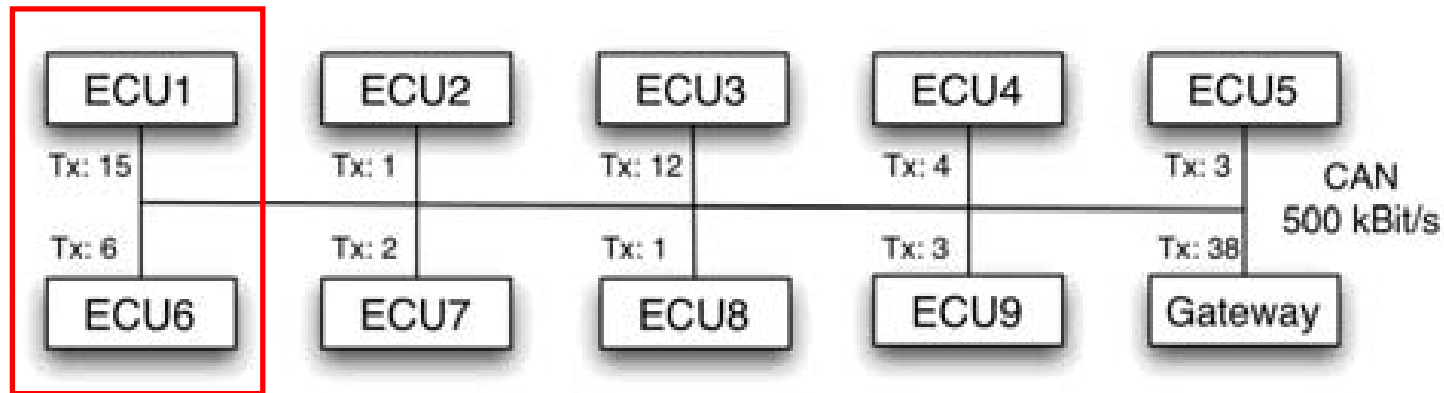
Case Study: Automotive

- Default, all freely assignable messages

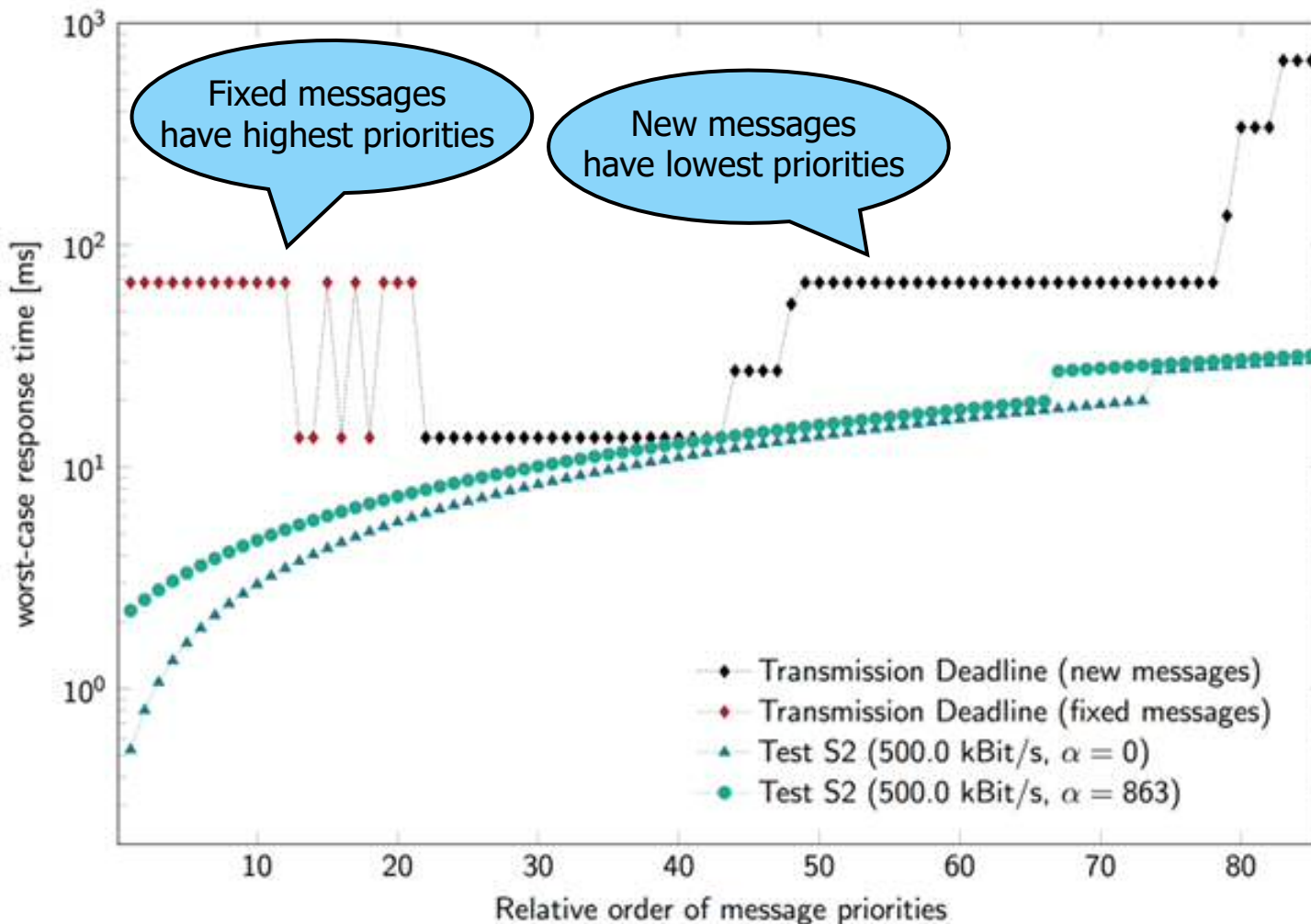
Expt.	Schedulability test	Priority order	Min. bus speed	Max. bus Utilisation
1	Exact E1	OPA	275.8 Kbit/s	84.7%
2	Sufficient S2	D-JMPO	276.3 Kbit/s	84.5%
3	Sufficient S2 & A1	D-JMPO	302.5 Kbit/s	77.2%
4	Exact E1	Specified	750.9 Kbit/s	31.1%

Case Study: Automotive

- 10 ECUs, 85 messages, nominally 500 Kbit/s
- Messages sent by ECUs 1 and 6 fixed IDs



OPA using **S1** and **A1**



Min bus speed
408.6 Kbit/s

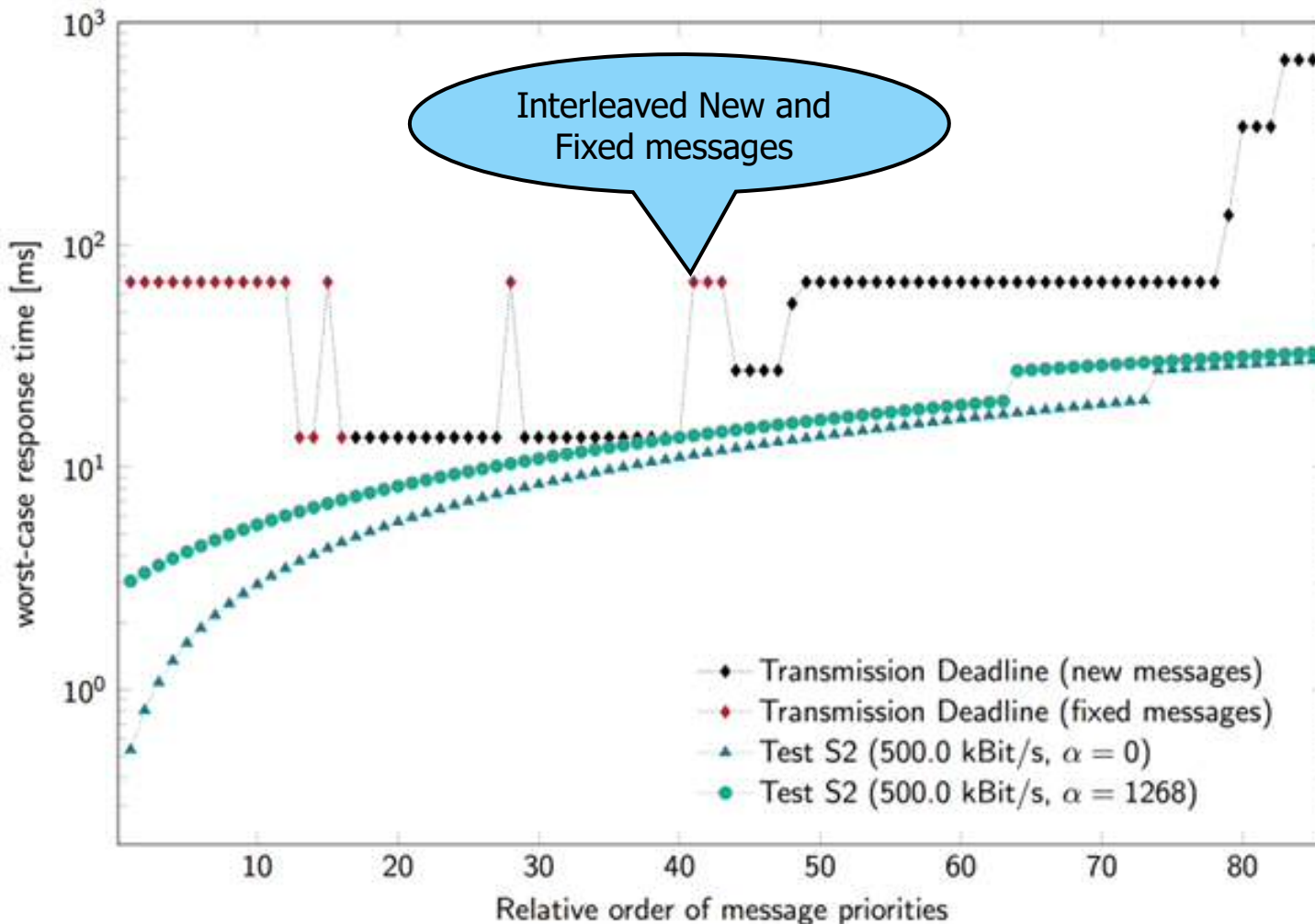
Max bus Util.
57.1%

Robustness

$\alpha = 863$
bit times

(5 msg re-tx)

RPA using **S1** and **A1**



Min bus speed
378.8 Kbit/s

Max bus Util.
61.7%

Robustness

$\alpha = 1268$
bit times

(7 msg re-tx)

A decorative graphic on the left side of the slide, consisting of overlapping yellow, red, and blue squares with a black crosshair.

Summary and Conclusions

- Investigated problem of priority assignment when some message IDs (priorities) are fixed
 - Very common problem in practice due to integration of legacy ECUs / ECUs which cannot be easily reprogrammed to use different message IDs

- Two variants of the problem
 - **P1:** gaps between fixed messages are enough to accommodate all freely assignable messages
 - **P2:** gaps are not large enough

A decorative graphic on the left side of the slide, consisting of overlapping yellow, red, and blue squares with a black crosshair.

Summary and Conclusions

- **Problem P1** admits a simple solution using an exact test (or sufficient tests)

- **Problem P2** is harder to solve
 - Greedy approach doesn't work – counter example given to prior work on Robust Priority Assignment
 - Can solve the problem when using sufficient test and an approximation in the analysis (messages of same length)
 - Finding a tractable solution for exact test or even sufficient tests without the approximation A1 is open and an interesting area for further research
 - Brute force approach is no use – case study has over a googol (10^{100}) different priority orderings possible

Recommendations

- Avoid problems **P1** and **P2** occurring in the first place
 - Use CAN middleware that allows message identifiers to be re-programmed post production
- Spread out the Message IDs used
 - The assignment problem is then **P1** and can be solved easily
- Use a Robust Priority Assignment
 - Gives maximum tolerance to interference
- Don't use ad-hoc methods of priority assignment
 - The resulting max bus utilisation is very poor (An easy way to waste half the bandwidth)



A decorative graphic consisting of overlapping yellow, red, and blue squares with a black crosshair.

Questions?
