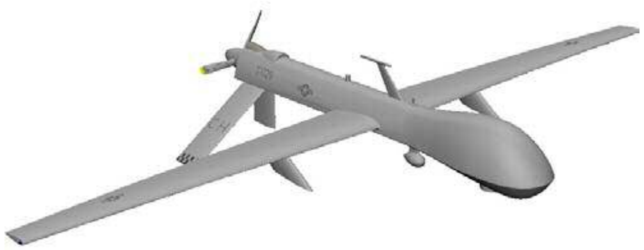
Mixed Criticality on Controller Area Network (CAN)

Alan Burns and Robert Davis

Real-Time Systems Research Group, University of York

Mixed Criticality Systems

- MCS
  - Applications of different criticality levels on the same HW platform
    - E.g. Safety Critical, Mission Critical, Non-critical
  - Driven by SWaP and cost requirements
- Examples
  - Aerospace: e.g. UAVs
    - Flight Control Systems v. Surveillance
  - Automotive:
    - Electronic Power Assisted Steering v. Cruise Control
- Typical research considers: Dual-Criticality Systems
  - Applications of HI and LO criticality



Mixed Criticality Systems

- Key requirements
  - *Separation* – must ensure LO-criticality applications cannot impinge on those of HI-criticality
  - *Sharing* – want to allow LO- and HI-criticality applications to use the same resources for efficiency

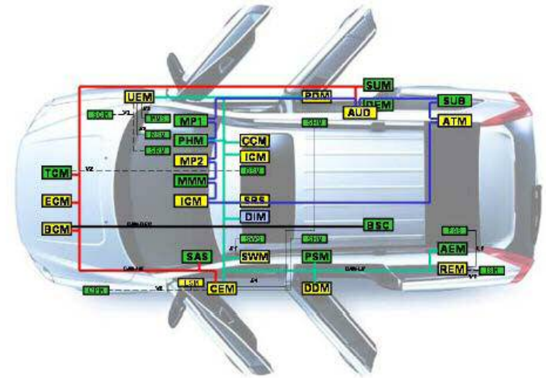
- Real-Time behaviour
  - Concept of a criticality mode (LO or HI)
  - System start in LO-criticality mode
  - LO and HI-criticality applications must meet their time constraints in LO-criticality mode
  - Only HI-criticality applications need meet their time constraints in HI-criticality mode

- Initial Research (Vestal 2007)
  - Idea of different LO- and HI-criticality WCET estimates for the same code
  - Certification authority requires pessimistic approach to $C(HI)$
  - System designers take a more realistic approach to $C(LO)$

Mixed Criticality Systems

- Most previous research (from Vestal 2007 on)
  - Examines processor schedulability
  - Assumes HI-criticality tasks have $C(LO)$ and $C(HI)$ estimates of WCET
  - Any HI-crit task executing for $C(LO)$ without signalling completion triggers transition to HI-criticality mode
  - In HI-crit mode all LO-crit tasks may be abandoned but HI-crit tasks must still meet their deadlines
- This research
  - Examines network schedulability
  - Addresses distributed MCS using Controller Area Network (CAN)
  - Assumes Hi-criticality messages have $T(LO)$ and $T(HI)$ minimum inter-arrival times, also $F(LO)$ and $F(HI)$ minimum number of tolerated faults
  - Uses Trusted Network Components to obtain separation
  - Develops a protocol for ensuring all nodes recognise the transition to HI mode (distributed system)

CAN Background



- Controller Area Network (CAN)
  - Simple, robust and efficient broadcast serial communications bus for in-vehicle networks
    - Developed originally by BOSCH in 1983, standardised in 1993 (ISO 11898)
    - Average family car now has approx. 25-35 Electronic Control Units (ECUs) connected via CAN
    - Today almost every new car sold in Europe uses CAN
- CAN Protocol
  - Messages compete for access to the bus based on priority (Message ID)
  - With priority queues in each node, 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

Schedulability Analysis for CAN

- Initially developed by Tindell et al. 1994, flaws later corrected by Davis et al. 2007
- Sufficient schedulability test for priority queued messages
  - Blocking $B_m = \max_{k \in lp(m)} (C_k)$
  - Queuing delay $R_m^s = \max(B_m, C_m) + \sum_{\forall k \in hp(m)} \left\lceil \frac{R_m^s + J_k + \partial}{T_k} \right\rceil C_k$
  - Response time $R_m = R_m^s + C_m + J_m$
  - Message m schedulable if $R_m \leq D_m$
- With faults
  - Queuing delay

$$R_m^s = \max(B_m, C_m) + \sum_{\forall k \in hp(m)} \left\lceil \frac{R_m^s + J_k + \partial}{T_k} \right\rceil C_k + F(Err_{\max} + \max_{\forall k \in hep(m)} (C_k))$$

Mixed Criticality on CAN

- Model
  - Messages categorised as HI-crit or LO-crit
  - For a HI-criticality message Period, Fault tolerance, and Transmission time may vary between criticality levels: $T(HI) \leq T(LO)$, $F(HI) \geq F(LO)$, $C(HI) \geq C(LO)$
  - Assume no change in deadlines, or jitter (use smallest D, largest J)

- Change in criticality mode
  - When a HI-crit message attempts to exceed its LO-crit parameters
    - Request for transmission not complying with $T(LO)$ and J
    - Request for transmission from a message with $C(LO) = 0$
    - Fault count exceeds $F(LO)$
  - Needs to be communicated – distributed system

  No longer need to transmit LO-crit messages

Mixed Criticality on CAN

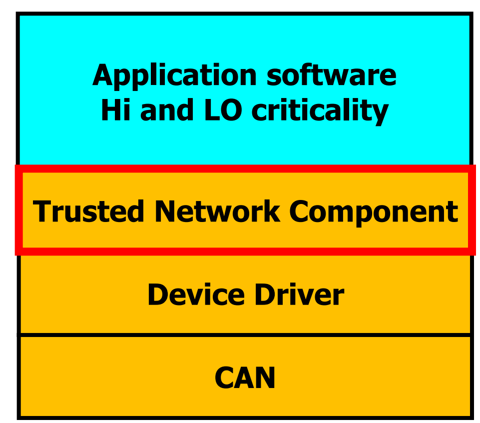
- Trusted Network Component (TNC) on each node
  - Developed to standards required for HI-criticality components
  - Monitors and controls access to the bus
  - Uses *sporadic invariant* to police queuing requests e.g. $T(LO) - J$ or $T(LO)$ since last instance of the same message was queued
  - If a TX request comes too soon from a LO-crit message

    ➡ blocked by the TNC

  - If a HI-crit message exceeds its LO-crit parameters or a larger number of faults detected than are tolerated in LO-crit mode

    ➡ Initiates transition to HI-crit mode at local node and by queuing a *trigger message* to broadcast the change

  - If a HI-crit transmit request exceeds its parameters – design decision what to do



Trusted Network Component (output)

```
output(M) is  -- called by application code
  t := clock
  if not Valid(M) then return <invalid> end if
  if Crit_Level = LO then
    if Trigger(M) then
      send(M)
      Crit_Level := HI
      flushALL
    else if t < G[M] then -- too early for LO mode
      if Crit(M) = HI then
        send(Go_HI)
        send(M)
        Crit_Level := HI
        flushALL
      else
        return <invalid, too early>
      end if
    else
      G[M] := max(G[M], t - J[M]) + T[M]
      send(M)
    end if
  else  -- in HI mode
    if Crit(M) = HI then send(M) end if
  end if
  return <success>
```

Broadcast message that system going to HI-crit mode

TX Abort of all LO-crit messages

Set up next permitted TX request time for message

Trusted Network Component (input)

```
input(M) is -- called by interrupt handler
  if Crit_Level = LO then
    if Errors_High then
      Crit_Level := HI
      flushALL
      send(Go_HI)
    else if Trigger(M) then
      Crit_Level := HI
      flushALL
      Flush(Go_HI)
    end if
  end if
  if (Crit_Level = LO) or
     (Crit_Level = HI and Crit(M) = HI) then
    Receive(M)
  end if
```

TX Abort of all LO-crit messages

Broadcast message that the system is going to HI-crit mode

Get rid of this node's Go_HI message if we have just seen one

Analysis of MixedCAN

- Analysis needs to cover
  - LO-crit mode
  - Analysis of the transition to HI-crit mode (and the HI-crit mode)

  follows the approach for tasks in Baruah et al. 2011.
- LO-crit mode:

$$R_m^s(LO) = \overline{B_m}(LO) + \sum_{\forall k \in hp(m)} \left\lceil \frac{R_m^s(LO) + J_k + \partial}{T_k(LO)} \right\rceil C_k(LO) + \overline{F}(LO)$$

$$R_m(LO) = R_m^s(LO) + C_m(LO) + J_m$$

$$R_m(LO) \leq D_m$$

Analysis of MixedCAN

- HI-crit mode: conservative assumptions (sufficient analysis)
  - HI-crit messages have their HI-crit parameters from time 0
  - LO-crit messages have their LO-crit parameters from time 0, and are aborted / not sent after the mode change
  - Max sized message used to communicate the mode change
  - Max sized LO-crit message is sent after the mode change
  - Max sized LO-crit message is sent after the $F(LO)+1$ fault occurs, but before the HI-criticality mode can be signalled (only if $F(HI) > F(LO)$)

$$R_m^s(HI) = C_m^F + C_m^{Mode} + \overline{B_m}(LO) + \sum_{\forall k \in hpH(m)} \left\lceil \frac{R_m^s(HI) + J_k + \partial}{T_k(HI)} \right\rceil C_k$$

$$+ \sum_{\forall k \in hpL(m)} \left\lceil \frac{R_m^s(LO) + J_k}{T_k(LO)} \right\rceil C_k + \overline{F_m}(HI)$$

$$C_m^F = \max_{\forall k \in hpL(m)} (C_k) \qquad C_m^{Mode} = C_{Go\_HI} + \max(C_{Go\_HI}, \max_{\forall k \in hpL(m)} (C_k))$$

$$R_m(HI) = R_m^s(HI) + C_m + J_m$$

Example Analysis of MixedCAN

- Message set:

$\mathcal{M}$	Crit	T(HI)	T(LO)	D	C	Pri	$R^s(LO)$	$R(LO)$
τ_1	HI	∞	-	5	2	1	-	-
τ_2	HI	12	24	12	2	4	7	9
τ_3	LO	-	11	11	2	3	4	6
τ_4	LO	-	6	6	1	2	3	4
τ_5	HI	18	36	18	3	5	9	12

  - Note deadlines in DMPO
- Analysis of HI-crit mode
  - See paper for detailed working

$R_5(HI) = 18$ ✓ **schedulable**

$R_2(HI) = 13$ ✗ **unschedulable**

- Does not mean the system is unschedulable for all priority orderings…

…can Audsley's OPA algorithm provides optimal priority assignment

Example with OPA

- LO-criticality mode:

$\mathcal{M}$	Crit	T(HI)	T(LO)	D	C	Pri	$R^s(LO)$	$R(LO)$
τ_2	HI	12	24	12	2	2	3	5
τ_3	LO	-	11	11	2	4	7	9
τ_4	LO	-	6	6	1	3	5	6
τ_5	HI	18	36	18	3	5	9	12

✓

- HI-criticality mode:

$\mathcal{M}$	Crit	T(HI)	T(LO)	D	C	Pri	$R^s(HI)$	$R(HI)$
τ_1	HI	∞	-	5	2	1	3	5
τ_2	HI	12	24	12	2	2	7	9
τ_5	HI	18	36	18	3	5	15	18

✓

All messages are schedulable with OPA

Message 2 has a higher priority than in DMPO

Basic MixedCAN (BMC)

- Trade-off
  - MixedCAN introduces extra messages (Go_HI) to communicate the criticality mode change
  - Prevents LO-crit messages being sent following the mode change
  - If there are not many high priority LO-criticality messages this trade-off may not be worthwhile

- Basic MixedCAN: A simpler protocol
  - TNC simply prevents LO-crit messages from being sent too soon
  - No flushing / prevention of LO-crit messages being sent in HI-crit mode
  - No broadcast of the criticality mode change
  - LO-crit messages do <u>not</u> have to meet their deadlines in HI-crit mode

Analysis of Basic MixedCAN

- Analysis of LO-crit mode as before
- Analysis of HI-crit mode:
  - LO-crit messages continue to be sent, but no additional Go_HI messages

$$R_m^s(HI) = \overline{B_m}(LO) + \sum_{\forall k \in hpH(m)} \left\lceil \frac{R_m^s(HI) + J_k + \partial}{T_k(HI)} \right\rceil C_k$$
$$+ \sum_{\forall k \in hpL(m)} \left\lceil \frac{R_m^s(HI) + J_k + \delta}{T_k(LO)} \right\rceil C_k + \overline{F_m}(HI)$$

$$R_m(HI) = R_m^s(HI) + C_m + J_m$$

  - Audsley's OPA algorithm is also optimal with respect to the schedulability test for BMC

Evaluation

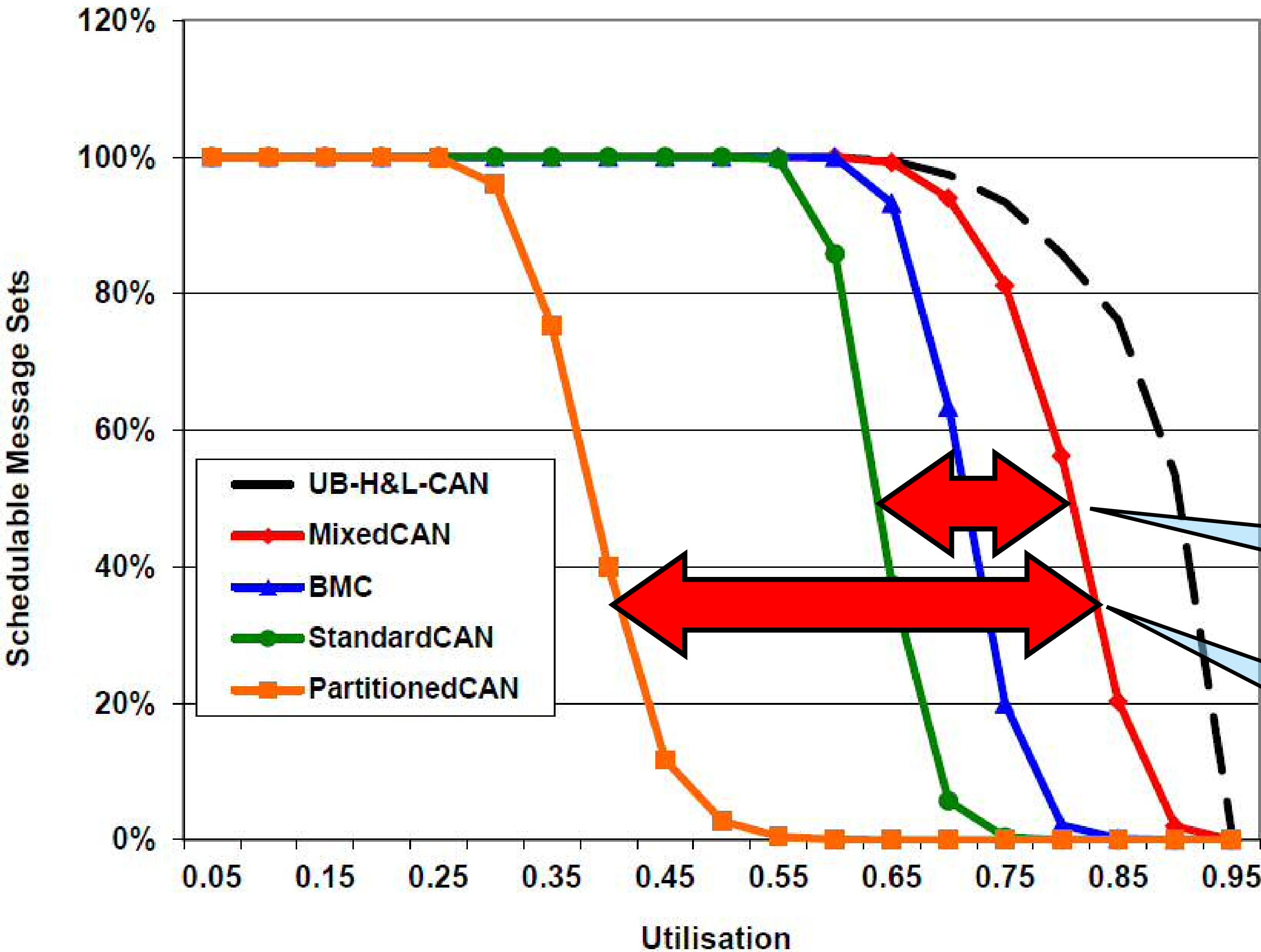
- Compared the following schemes:
  - **PartitionCAN**: Assigns HI-crit messages higher priorities – uses DMPO within the HI- and LO-crit subsets
  - **StandardCAN**: Assumes the worst-case parameters – ignores criticality
  - **BMC**: Determines schedulability of LO-crit messages in LO-crit mode, and HI-crit messages in both modes – uses OPA
  - **MixedCAN**: Protocol described earlier – uses OPA
  - **UB-H&L-CAN**: A necessary test – gives an upper bound on schedulability – checks LO-crit messages are schedulable in LO-crit mode and that HI-crit messages are schedulable in HI-cit mode – uses DMPO

Note on use of DMPO – it is optimal for the parameters used in the evaluation (all TX times are identical, as are blocking times, jitter is zero, and the simple sufficient test is used)

Evaluation

- Message set generation:
  - **Message sets contained 10-120 messages (default 80)**
  - LO-crit message periods $T(LO)$ followed a Log-uniform distribution 10ms – 1000ms
  - $\boldsymbol{T(HI) = T(LO) / CF}$ **Criticality factor e.g.** $\boldsymbol{CF = 2.0}$
  - Deadline = $T(LO)$ for LO-crit and = $T(HI)$ for HI-crit messages
  - TX times equated to the maximum time for an 8 data byte message (max bit stuffing = 135 bits)
  - Maximum blocking factor for all messages (i.e. assuming some soft real-time messages at lower priorities)
  - Probability of a message being HI-crit, $CP = 0.5$ (by default)
  - Bus speed adjusted to give the desired utilisation (utilisation computed according to LO-crit parameters)
  - **Faults tolerated** $\boldsymbol{F(LO) = 0}$**,** $\boldsymbol{F(HI) = 15}$ **(default)**
  - Additional Go_HI messages in the analysis of MixedCAN

Success ratio: Periods



Criticality reflected in varying message periods

80 messages
CP = 0.5
CF = 2.0
No fault tolerance

Significant improvement over StandardCAN

Very large improvement over PartitionedCAN

Weighted schedulability

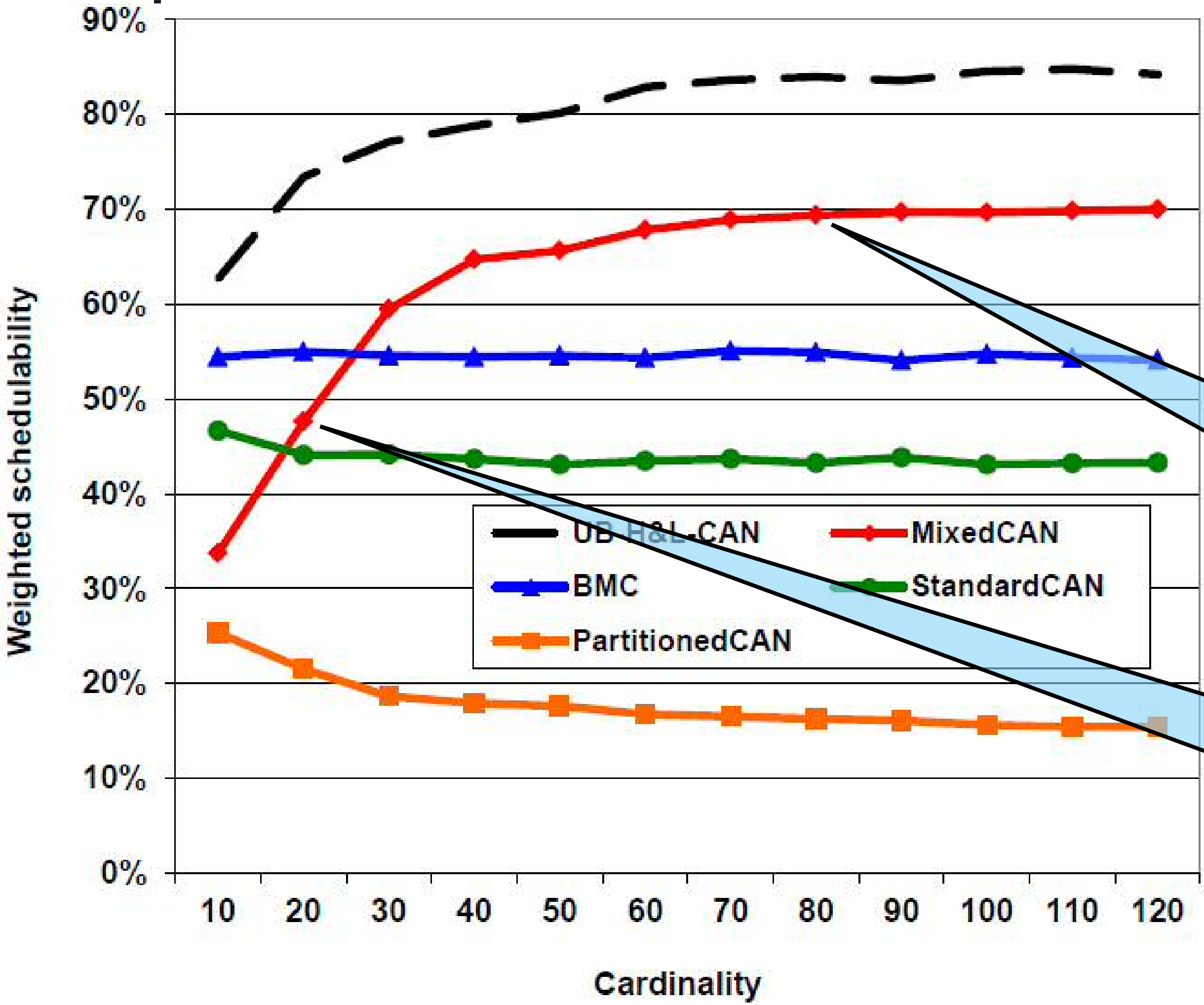
- Weighted schedulability
  - Enables overall comparisons when varying a specific parameter (not just utilisation)
  - Combines results from all of a set of equally spaced utilisation levels
  - Weighted schedulability:

$$Z_y(p) = \frac{\sum_{\forall \tau} S_y(\tau).U(\tau)}{\sum_{\forall \tau} U(\tau)}$$

  - Collapses all data on a success ratio plot for a given method, into a single point on a weighted schedulability graph

Weighted schedulability is effectively a weighted version of the area under a success ratio curve biased towards scheduling higher utilisation message sets

Weighted schedulability: Number of messages



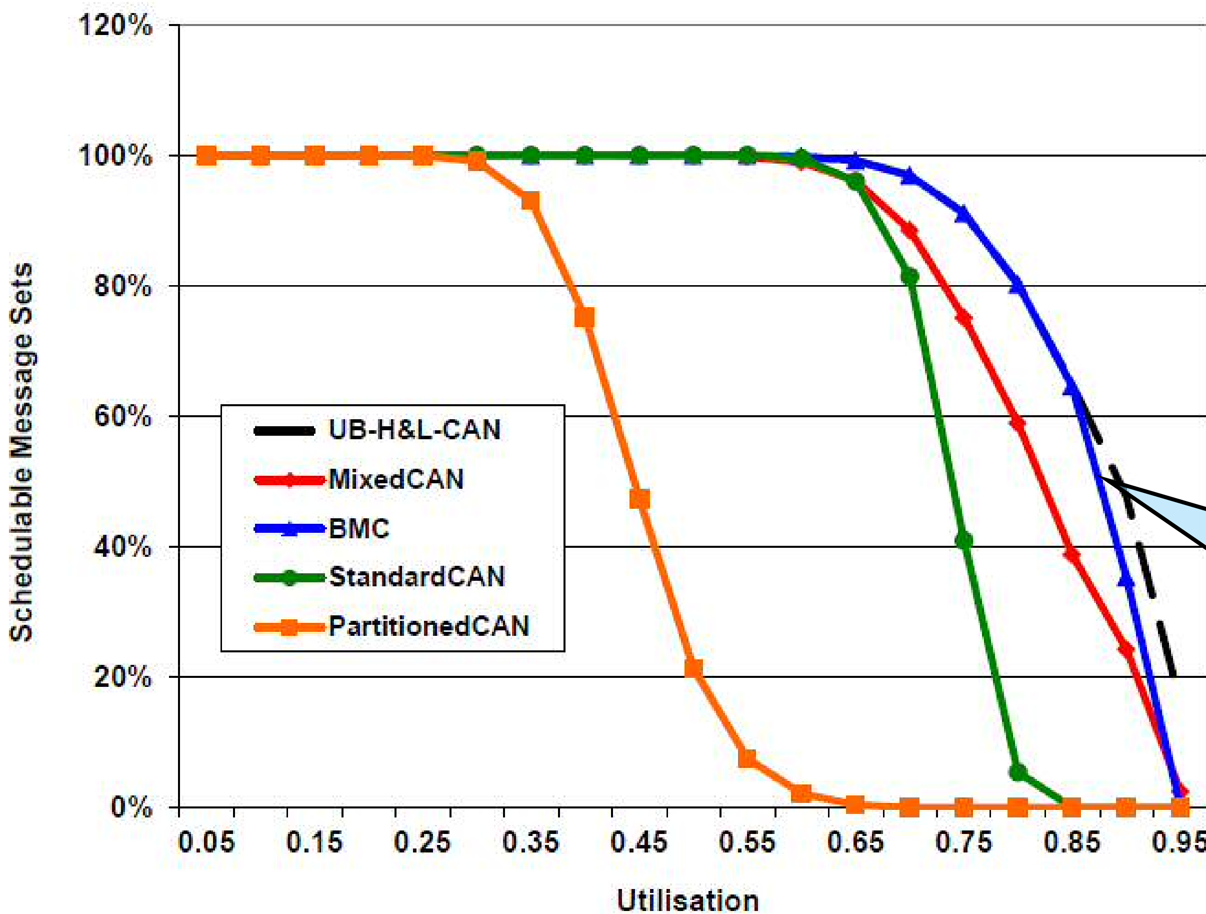
Criticality reflected in varying message periods

10 to 120 messages
CP = 0.5
CF = 2.0
No fault tolerance

With many messages broadcasting the criticality change is effective; MixedCAN better than BMC

With few messages broadcasting the criticality change is not worthwhile; BMC better than MixedCAN

Success ratio: Fault tolerance



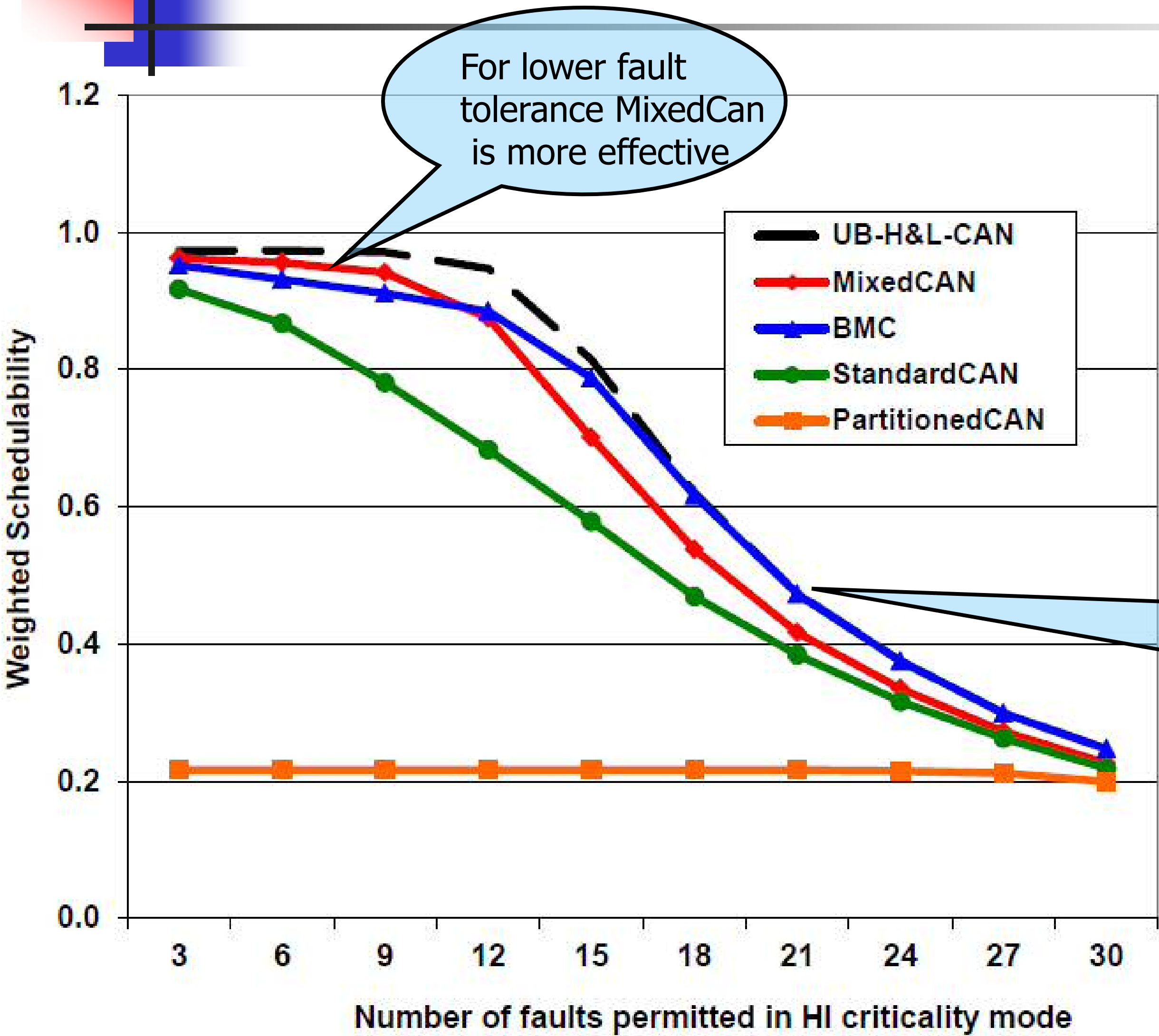
Criticality reflected in varying number of faults tolerated

$F(HI) = 15, F(LO) = 0$

80 messages
CP = 0.5
CF = 1.0 ($T(HI) = T(LO)$)

BMC is more effective because high fault tolerance forces HI-crit messages to have high priorities

Weighted schedulability: # faults in HI-crit mode



Criticality reflected in varying number of faults tolerated $F(HI), F(LO) = 0$

80 messages
CP = 0.5
CF = 1.0 ($T(HI) = T(LO)$)

Again BMC more effective when high fault tolerance forces HI-crit messages to have high priorities

Summary and Conclusions

- Main contributions
  - Set out support for mixed criticality on Controller Area Network (CAN) – addressing changes in period and fault tolerance between criticality levels
  - Trusted Network Components necessary to obtain *separation* between criticality levels
  - MixedCAN enables effective *sharing* of bandwidth between HI- and LO-crit messages
  - MixedCAN broadcasts a criticality mode change switching off LO-crit messages, but has additional overheads
  - Basic MixedCAN (BMC) allows LO-crit messages to continue in HI-crit mode trading lower overheads (no broadcast of the mode change) for additional interference
  - Introduced simple sufficient but effective schedulability analysis for both MixedCAN and BMC
  - Evaluation showed the methods to be highly effective for representative configurations with circa 80 messages

Questions?