Controller Area Network (CAN) Schedulability Analysis for Messages with Arbitrary Deadlines in FIFO and Work-Conserving Queues

Robert Davis[1] and Nicolas Navet[2]

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

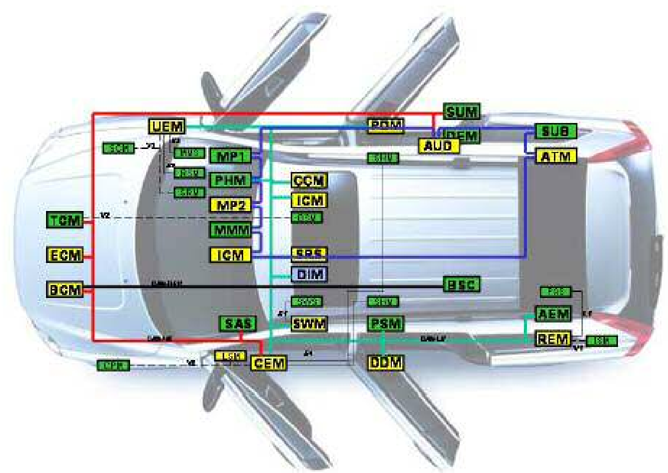
[2] INRIA / RTaW, Nancy, France

Outline

- Controller Area Network (CAN)
  - Background
- Motivation and work-conserving queues
- Scheduling model and analysis for priority queues
- Analysis for Work-conserving queues
- Priority assignment
- Case study
  - Impact of work-conserving queues
- Empirical investigation
- Summary and conclusions
- Recommendations

CAN Background

- Controller Area Network (CAN)
  - Simple, robust and efficient 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
  - CAN mandatory for cars and light trucks sold in USA since 2008 (On Board Diagnostics)
  - Today almost every new car sold in Europe uses CAN
  - Sales of microprocessors with CAN capability – approx 750 million in 2010.



Scheduling (Priority queues)

- 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

- Schedulability Analysis for CAN (assuming priority queues)
  - First derived by Tindell in 1994 [14,15] from earlier work on fixed priority pre-emptive scheduling
    - Calculates worst-case response times of all CAN messages
    - Used to check if all messages meet their deadlines in the worst-case
  - Significant flaws in the original analysis corrected by Davis et al. [6] in 2007.

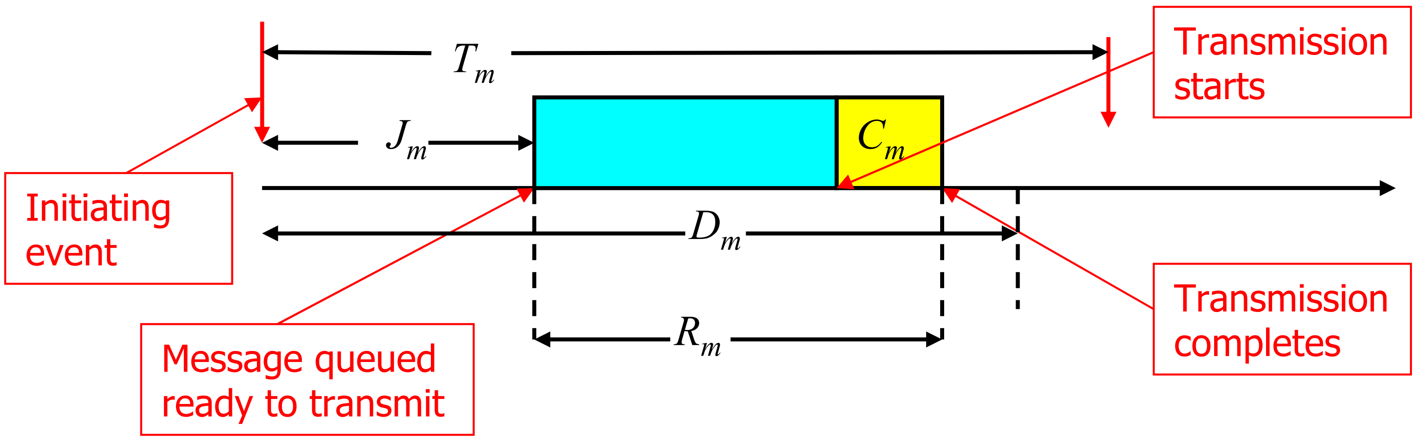
Motivation: Work-conserving and FIFO queues

- Analysis in [6] only holds if every node can always enter its highest priority ready message into bus arbitration
- This may not always be the case:
  - Device drivers may implement FIFO rather than priority queues
    - Simpler to implement, less code / lower CPU load
  - It may not be possible to abort a lower priority message in a transmit buffer
    - An issue if there are fewer transmit buffers than transmitted messages
  - The CAN controller may enter messages into bus arbitration based on transmit buffer number rather than message ID (priority)
    - May result in high priority messages being delayed by lower priority ones placed in transmit buffers with lower numbers
- Precise queuing policy used may be difficult to quantify / analyse

Work-conserving queues

- Recognise that the system integrator may lack complete information (e.g. when CAN nodes supplied by 3rd parties)
- Work-conserving queues
  - Assume only that the comms stack on a node ensures that if there are ready messages (queued by an API call but not yet transmitted) then one of them will be entered into arbitration whenever arbitration start on the bus
- Work-conserving: re-ordering not permitted (WQN)
  - Arbitrary work-conserving queue – but nevertheless ensures that instances of the same message are not re-ordered
  - Example: FIFO
- Work-conserving: re-ordering permitted (WQR)
  - Arbitrary work-conserving queue – may re-order instances of the same message
  - Undesirable in practice – but is the most general case
  - Example: LIFO

Schedulability Analysis: 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
- Additional notation for work-conserving queues
  - Group $M(m)$ set of messages transmitted by the node that transmits message m
  - L_m lowest priority of any message in group $M(m)$
  - f_m *buffering time* – longest time that an instance of message m can take from being queued to being able to enter into priority based arbitration ($f_m = 0$ for priority queued messages)

Impact of buffering delay

- High priority messages delayed from entering priority based arbitration due to a work-conserving rather than priority based queuing policy can impact the schedulability of messages sent by other nodes
  - From the perspective of other nodes on the network, such a message k can be modelled as having additional jitter equal to its maximum buffering time f_k
  - Allows analysis to be derived from the case where all nodes use priority queues

Schedulability Analysis for Priority Queues

- Schedulability test (derived from [6]) for message m sent via a priority queue on a heterogeneous network
  - Worst-case response time for an instance of message m occurs within a priority-level m busy period
  - Assuming instances of all higher priority messages released at the start of the busy period with maximum jitter, with subsequent instances of these messages released as soon as possible
  - Blocking at the start of the busy period due to longest lower priority message: $B_m = \max_{k \in lp(m)} (C_k)$
  - Busy period: $$v_m^{n+1} = B_m + \sum_{\substack{\forall j \in hep(m) \\ \wedge j \in M(m)}} \left\lceil \frac{v_m^n + J_j}{T_j} \right\rceil C_j + \sum_{\substack{\forall k \in hp(m) \\ \wedge k \notin M(m)}} \left\lceil \frac{v_m^n + J_k + f_k}{T_k} \right\rceil C_k$$
  - Number of instances of message m ready in the busy period: $$Q_m = \left\lceil \frac{v_m^{n+1} + J_m}{T_m} \right\rceil$$

Schedulability Analysis for Priority Queues

- Interval from start of busy period to when instance q of message m begins transmission

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

- Iteration starts with $w_m^0(q) = B_m + qC_m$ ends when $w_m^{n+1}(q) = w_m^n(q)$ or when $J_m + w_m^{n+1}(q) - qT_m + C_m > D_m$
- Response time of instance q of message m :

$$R_m(q) = J_m + w_m(q) - qT_m + C_m$$

- Worst-case response time of message m :

$$R_m = \max_{q=0..Q_m-1} (R_m(q))$$

- Note the impact of a message k sent by another node implementing a work-conserving queue. The queuing policy can delay message k from entering priority based arbitration by up to f_k and hence the impact on message m can be modelled as message k having additional jitter equal to f_k

Analysis for work-conserving queues

- Aim:
  - To obtain an upper bound on the worst-case response time for message m in group $M(m)$ sent by a node implementing a work-conserving queuing policy in a heterogeneous network
- Strategy: Make (pessimistic) worst-case assumptions:
  - Assume that all messages in $M(m)$ are transmitted at the lowest priority L_m of any message in the group
    - This cannot result in shorter response times for any of the instances of messages from $M(m)$
    - The node sending messages in $M(m)$ can then be modelled as implementing a priority queue, albeit with just one priority
    - Assume that instances of message m lose ties in this priority queue to instances of other messages in the group
  - Model messages sent by other nodes as being priority queued with additional jitter of f_k

➡ **use analysis derived from that for priority queues**

Schedulability Analysis for Work-Conserving Queues

- Schedulability test for message m sent via a work-conserving queue on a heterogeneous network
  - Pessimistic assumption that all messages in $M(m)$ are transmitted at priority L_m
  - Blocking at the start of the busy period due to longest lower priority message: $B_m = \max_{k \in lp(L_m)} (C_k)$
  - Busy period: $$v_m^{n+1} = B_{L_m} + \sum_{j \in M(m)} \left\lceil \frac{v_m^n + J_j}{T_j} \right\rceil C_j + \sum_{\substack{\forall k \in hp(L_m) \\ \wedge k \notin M(m)}} \left\lceil \frac{v_m^n + J_k + f_k}{T_k} \right\rceil C_k$$
  - Number of instances of message m ready in the busy period: $$Q_m = \left\lceil \frac{v_m^{n+1} + J_m}{T_m} \right\rceil$$

Schedulability Analysis for Work-Conserving Queues

- General framework:
  - Compute length of the interval from start of busy period to when instance q of message m begins transmission

$$w_m^{n+1}(q) = B_{L_m} + qC_m + I_m^*(q, w_m^n) + \sum_{\forall k \in hp(L_m) \wedge k \notin M(m)} \left\lceil \frac{w_m^n + J_k + f_k + \tau_{bit}}{T_k} \right\rceil C_k$$

  - Iteration starts with $w_m^0(q) = B_{L_m} + qC_m$ ends when $w_m^{n+1}(q) = w_m^n(q)$ or when $J_m + w_m^{n+1}(q) - qT_m + C_m > D_m$
  - Response time of instance q of message m :

$$R_m(q) = J_m + w_m(q) - qT_m + C_m$$

  - Worst-case response time of message m :

$$R_m = \max_{q=0..Q_m-1} (R_m(q))$$

Need to instantiate the interference term $I_m^*(q, w_m^n)$ for other messages $M(m)$ sent by the same node for the work-conserving queuing policy

Schedulability Analysis for Work-Conserving Queues

- For work-conserving queues (WQN) that do not permit re-ordering of instances of the same message:

$$I_m^{WQN}(q, w_m^n) = \sum_{j \in M(m) \wedge j \neq m} \left\lceil \frac{w_m^n + J_j + \tau_{bit}}{T_j} \right\rceil C_j$$

- With re-ordering of instances (WQR):

$$I_m^{WQR}(q, w_m^n) = \sum_{\substack{j \in M(m) \\ \wedge j \neq m}} \left\lceil \frac{w_m^n + J_j + \tau_{bit}}{T_j} \right\rceil C_j + \underbrace{\max\left(0, \left\lceil \frac{w_m^n + J_m + \tau_{bit}}{T_m} \right\rceil - (q+1) \right) C_m}_{\text{later instances of the same message}}$$

Schedulability test for Work-Conserving Queues

- Buffering delay:
  - Upper bound given by

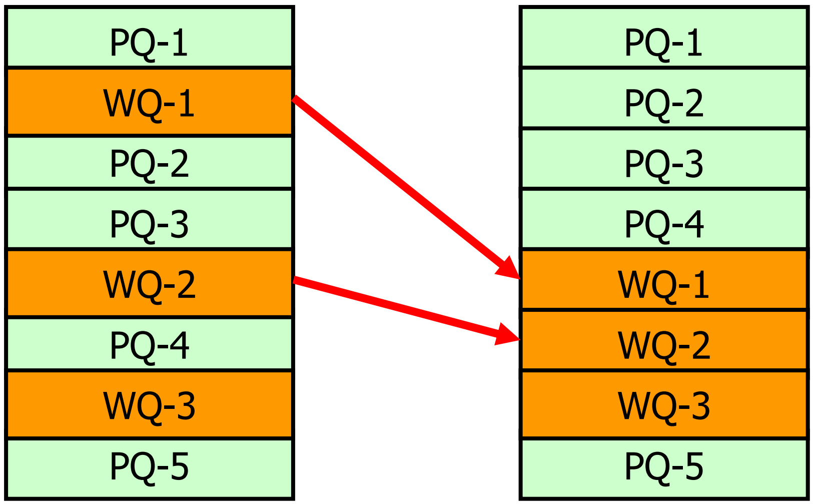
$$f_m = R_m - J_m - C_m$$

- Problem:
  - If priorities of message groups are interleaved, then buffering delay of one message can depend on the response time of another message and vice-versa
  - Resolved by noting that buffering delays are monotonically non-decreasing w.r.t. response times and vice-versa

```
repeat = true
initialise all f_m = 0
while(repeat){
    repeat = false
    for each priority m, highest first{
        if (m belongs to a WQN- or WQR-node){
            calc R_m via:
            (7) to (10) & (11) – WQN-node,
            (7) to (10) & (12) – WQR-node.
            if( R_m > D_m ) {
                return unschedulable
            }
            if( f_m != R_m − J_m − C_m ){
                f_m = R_m − J_m − C_m
                repeat = true;
            }
        }
        else { // m belongs to a PQ-node
            calc R_m via (2) to (6).
            if( R_m > D_m ) {
                return unschedulable
            }
        }
    }
}
return schedulable
```

Adjacent priority ordering

- **Adjacent priority ordering**:
  - Messages within a group $M(m)$ have adjacent priorities – no interleaving with other messages
- **Optimal partial ordering**:
  - If a priority ordering Q exists that is schedulable according to the previous schedulability test, then a schedulable adjacent priority ordering also exists
  - Regardless of the priority ordering of priority queued messages, all messages sharing a work-conserving queue should have adjacent priorities (but not necessarily consecutive values)



Adjacent priorities

- With **Adjacent Priorities**:
  - No need to account for buffering time so $f_m = 0$ for all messages sent via work-conserving queues
  - This is because if a message *m* is of higher priority than message *k*, then crucially, so are all of the other messages that share the queue with *m*, hence all contribute to the queuing delay of message *k*, and the order in which they are actually sent on the bus is irrelevant
  - Setting $f_m = 0$ for all messages:
    - simplifies the analysis (no repeats of the while loop – just calculate the message response times)
    - Removes a significant amount of pessimism

Optimal priority assignment

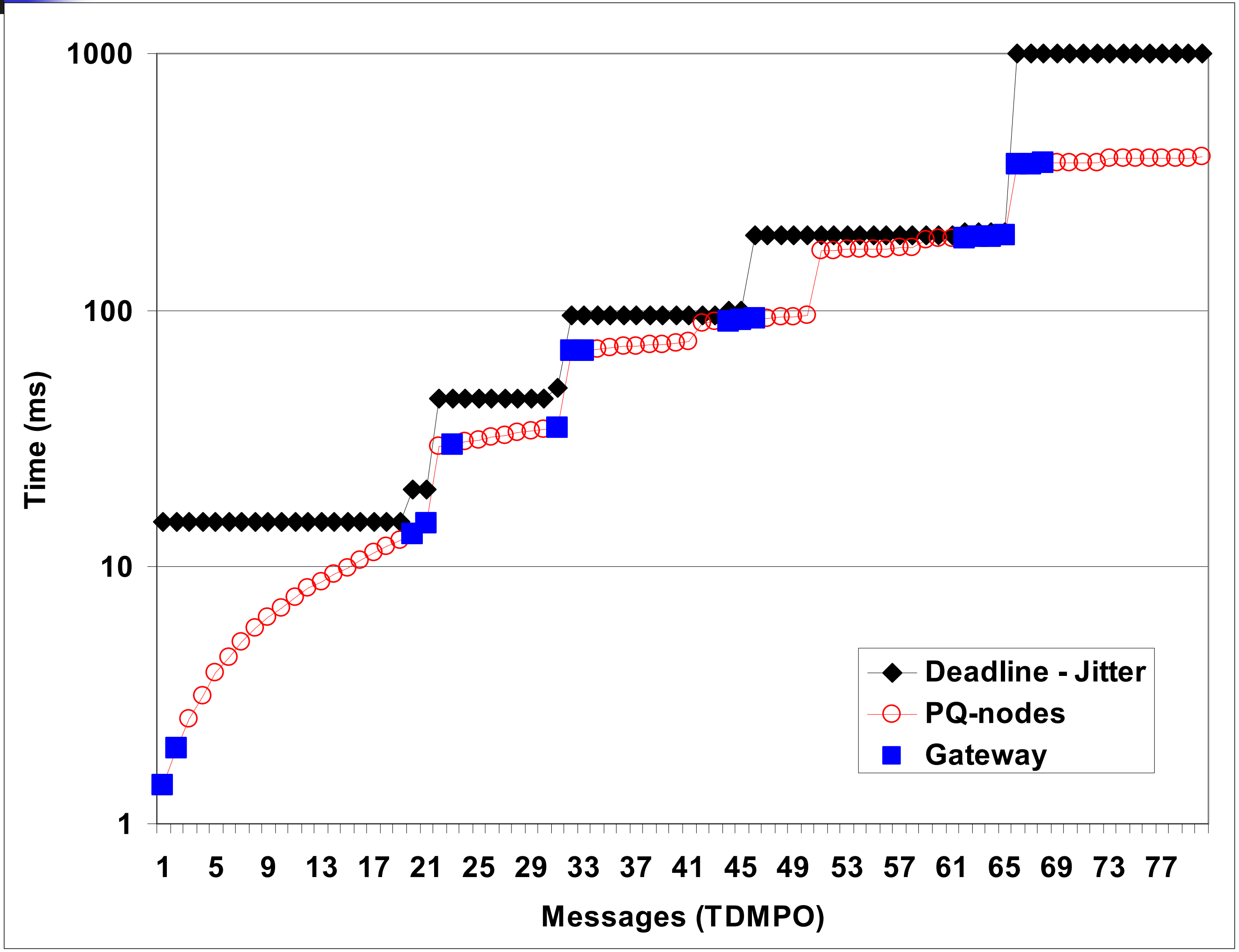
- OPA-FP/WQ algorithm:
  - Based on Audlsey's greedy Optimal Priority Assignment (OPA) algorithm
  - Optimal for networks with a mix of priority queues and work-conserving queues w.r.t. the schedulability test presented

```
for each priority band k, lowest first
{
    for each message msg in the initial list {
            check the schedulability of msg in priority band
            k, with all unassigned priority-queued messages
            and messages in other WQN or WQR groups
            assumed to be in higher priority bands
            if msg belongs to a WQN- or WQR-node,
            similarly check the schedulability of all other
            messages from the same group in priority band k.
            if all messages checked are schedulable
            {
                  assign them to priority band k (with adjacent
                  priorities)
                  break (continue outer loop)
            }
    }
      return unschedulable
}
return schedulable
```

Case Study: Automotive

- Generated using NETCARBENCH
- 16 ECUs, 80 messages, nominally 500 Kbit/s, load 36.5%
- Message periods 20, 50, 100, 200, or 1000 ms
- 5-8 data bytes in each message
- Deadlines equal to periods, queuing jitter 5ms,
- 18 messages sent via a gateway
  - 9 of which come from another network. The deadline of these messages is 2x period, and jitter = period
- Experiments
  - *Config. 1*: All priority queues, message priorities in transmission deadline monotonic order (TDMPO) i.e. D-J order
  - *Config. 2*: Gateway assumed to use a work-conserving queue, message priorities again TDMPO
  - *Config. 3*: As Config 2, but using OPA-FP/WQ algorithm to set message priorities

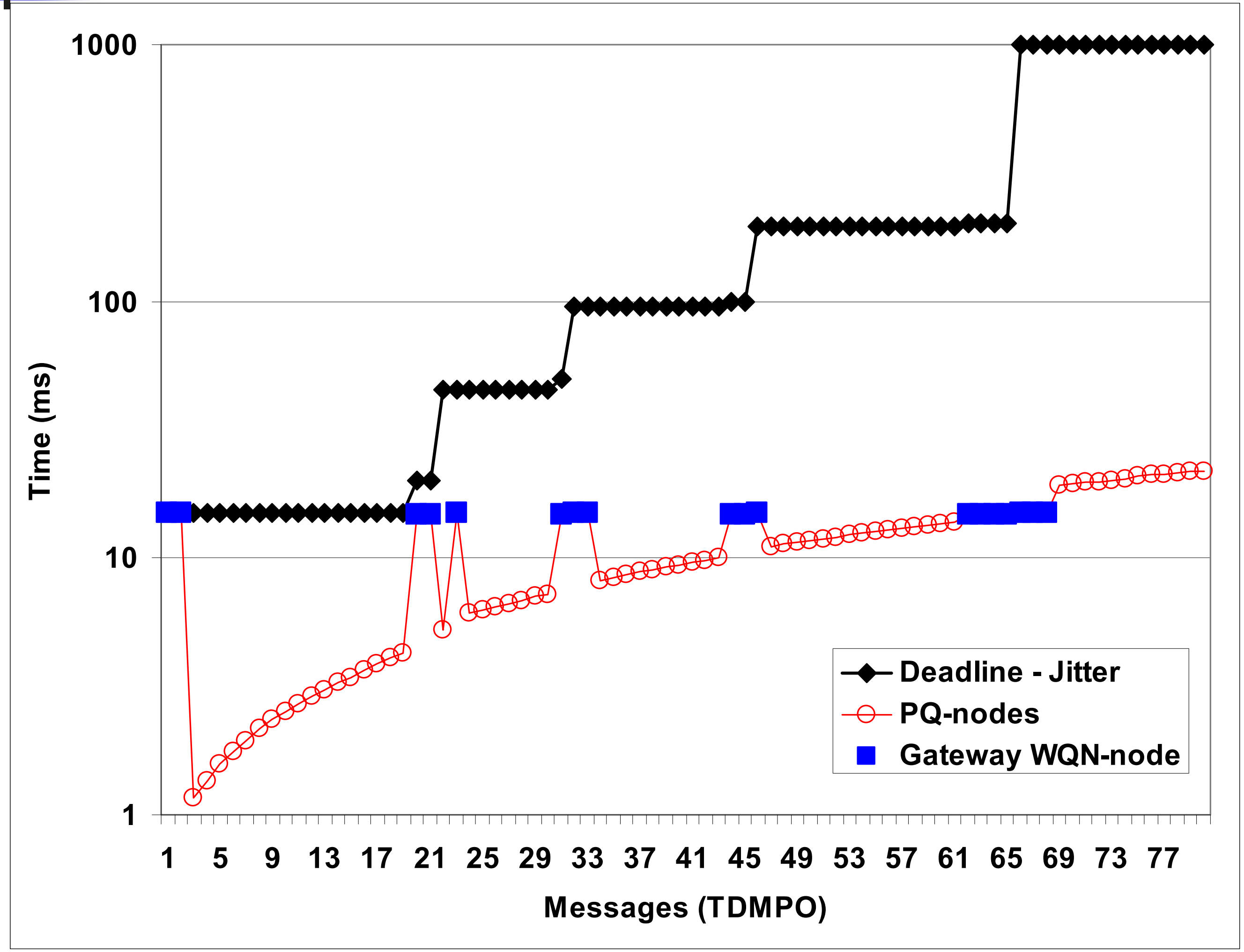
Config 1: All priority queues



Min bus speed
191 Kbit/s

Max bus Util.
95.8%

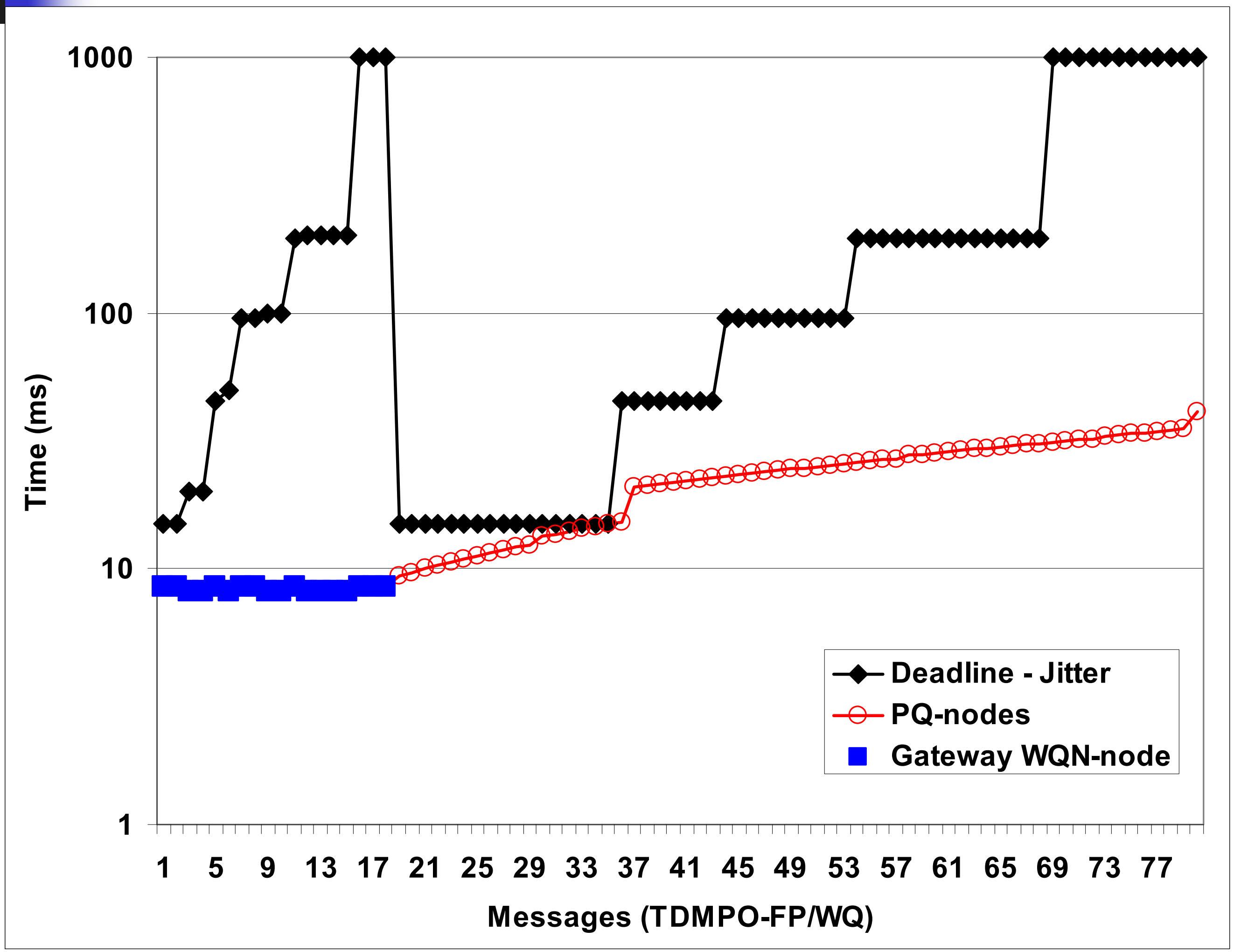
Config 2: Gateway WQN



Min bus speed
624 Kbit/s
(+230%)

Max bus Util.
29.3%

Config 3: Gateway WQN



Min bus speed
393 Kbit/s
(+105%)

Max bus Util.
46.6%

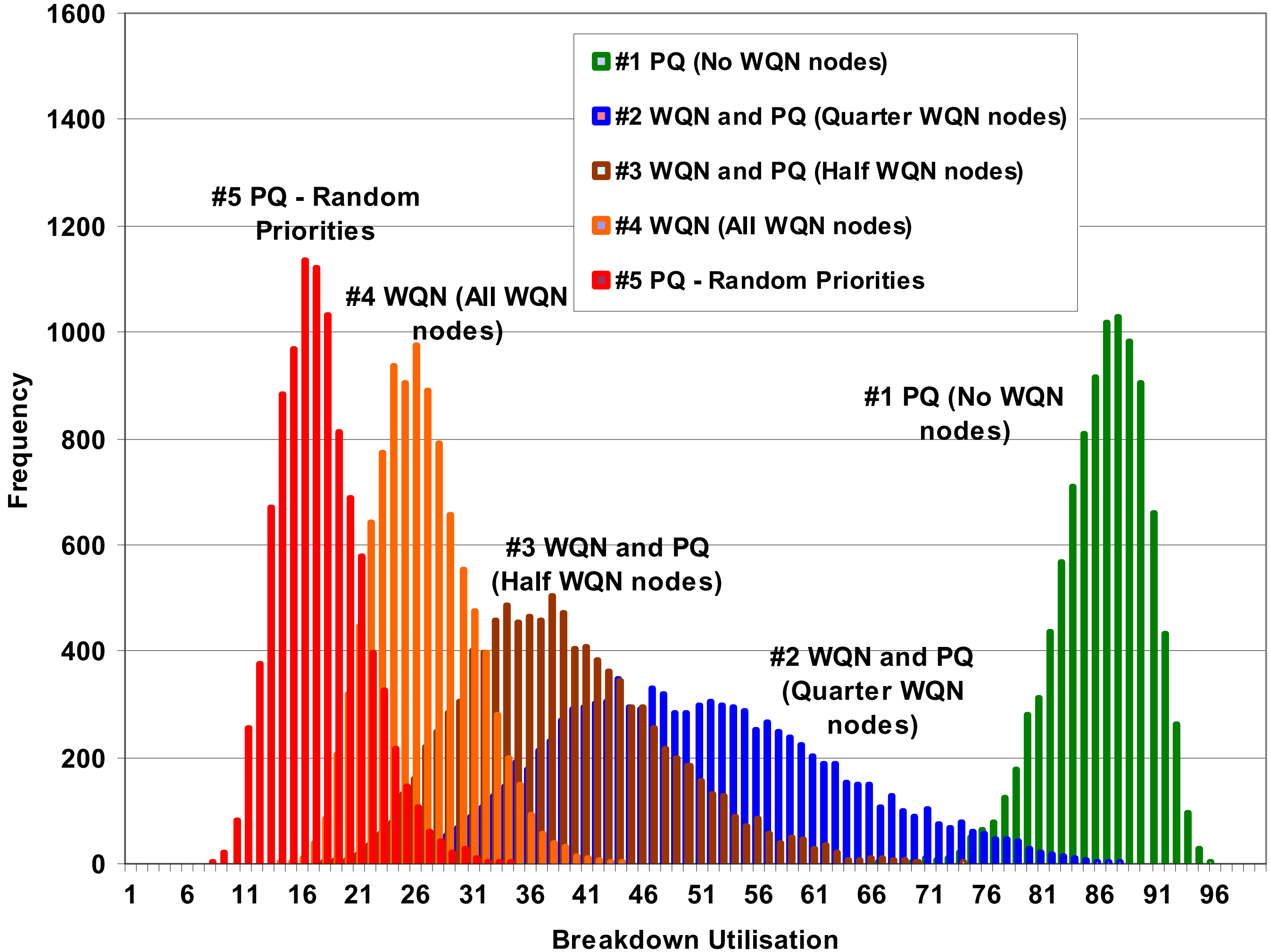
Case Study: Summary

Config.	Node type	Priority order	Min. bus speed	Max. bus Utilisation
1	All PQ	TDMPO	191 Kbit/s	95.8%
2	Gateway WQN	TDMPO	624 Kbit/s (+230%)	29.3%
3	Gateway WQN	OPA-FP/WQ	393 Kbit/s (+105%)	46.6%

Empirical evaluation

- Examined 10,000 randomly generated sets of messages:
  - 80 messages in each set, 8 data bytes per message
  - 8 nodes on the network
  - Random allocation of messages to nodes
  - Log-uniform distribution of message periods 10ms – 1000ms
  - First node assumed to be a gateway:
    message deadlines = 2x period, jitter = period
  - Other nodes: message deadlines = period, jitter (uniform distribution 2.5 – 5ms)
  - 11-bit identifiers
- Configurations
  - *Config. 1*: All PQ nodes - TDMPO
  - *Config. 2*: Two WQN nodes – TDMPO-WQ/FIFO
  - *Config. 3*: Four WQN nodes – TDMPO-WQ/FIFO
  - *Config. 4*: All WQN nodes – TDMPO-WQ/FIFO
  - *Config. 5*: All PQ nodes – random priorities

Empirical results



Evaluation

- Empirical evaluation of 10,000 message sets
  - 8 nodes, 80 messages, 8 data bytes per message
  - periods 10-1000ms (log uniform distribution)
  - jitter 2.5-5ms (uniform distribution)
  - 1 gateway: deadlines = 2x period, jitter = period

Config.	**Node type**	**Priority order**	**Average Max. bus utilisation**
1	All PQ	TDMPO	**85.5% (89.5%)**
2	2 WQN, 6 PQ	TDMPO-FP/WQ	49.9% (62.7%)
3	4 WQN, 4PQ	TDMPO-FP/WQ	38.0% (44.9%)
4	All WQN	TDMPO-FP/WQ	**25.5% (28.4%)**
5	All PQ	Random	16.4% (18.4%)

  - Figures in brackets are without larger deadlines and jitter for messages sent by the gateway

Summary and Conclusions

- Introduced sufficient schedulability test for CAN networks with a mix of nodes using work-conserving (e.g. FIFO) and priority queues
  - Extended previous analysis for FIFO queues and constrained deadlines to work-conserving queues and arbitrary deadlines
  - Analysis reduces to that for FIFO queues when all messages have constrained deadlines (see paper)
  - FIFO analysis from ECRTS'11 holds for arbitrary work-conserving queues when all messages have constrained deadlines.
  - For work-conserving queues and the analysis presented, Adjacent priority ordering is optimal within each message group
  - Modified OPA algorithm provides an optimal priority ordering (w.r.t. our analysis) for a set of messages sent via work-conserving priority queues

Summary and Conclusions

- Examined performance of work-conserving queues / analysis via case study and empirical evaluation
  - Significant reduction in performance – increased bus speed is required and a large decrease in max. bus utilisation (e.g. 80% down to 30%)
  - Mainly caused by unavoidable priority inversion
- Why are FIFO queues used
  - Make the device driver more efficient (less processor load)
  - Easier to implement
- But
  - local gain comes at a cost – undermining priority based arbitration on CAN – significant performance penalty

Recommendations

- To obtain the best possible performance
  - Use an **appropriate priority ordering** (e.g. based on transmission deadlines)
  - **Use priority queues**
  - **Avoid using work-conserving / FIFO queues** whenever possible

- Arbitrary work-conserving / FIFO queues can cause significant performance degradation
  - When there are many messages in a queue, with a range of transmission deadlines that interleave with those of other messages on the network – result is significant priority version

Questions?