


Today :- Real-time Networks.



Real time systems are about predictability.

Today we will look at the CAN bus which is a priority driven network. Which allows predications to be made about message times.

The analysis that we do will be similar to response time calculations.

Controller Area Network (CAN)



- From Bosch (mid '80s) for automotive applications
- Used in most automotive systems (Volvo).
- Lots of off the shelf controllers available. Philips, Intel, NEC, Siemens.
- Possible to have one Sender and many receivers
- Highly robust (error mechanism to overcome disturbance on the bus).
- Medium Speed, Max 1Mbit/sec; typically used from 35 Kbit/sec upto 500Kbit/sec. Typical max length 50m.

CAN Frame layout



- Small Frame size 0 to 8 Bytes
- Relatively high overhead. >100bits to send 64bits.
- Id number 11 bits, gives 2048 priority levels.

CAN Arbitration Mechanism



- Shared broadcast bus.
- Bus behaves like a large AND-gate
 - If all nodes send 1 then the bus becomes 1, otherwise 0.
- A frame is tagged by an identifier, used for arbitration as well as identity.
- Bit-wise arbitration
 - Each message has a unique priority \Rightarrow node with message with the lowest id wins arbitration.

CAN Arbitration Mechanism



Every node on the network implements the following algorithm:

- Wait until the bus is free.
- Start transmitting the bits of the message identifier, if at any time the bit on the bus is different from the bit you send go back to the waiting condition.

Because the bus acts like a giant AND gate this gives arbitration.

Once arbitration has been achieved the winning node continues and transmits its packet.

CAN Arbitration Mechanism Example

Three nodes:

Node	A	B	C
ID	011	010	101

Suppose A, B and C all try and transmit at the same time.

A	B	C	What's on the Bus?
Transmit 0	Transmit 0	Transmit 1	0
Transmit 0	Transmit 0	Waiting	0
Transmit 1	Transmit 1		1
Transmit 1	Transmit 0		0

A wins.

The CAN-bus abstraction



The CAN bus is a simple priority based resource without preemption.

That is the highest priority message that is able to be sent is sent.

Once a message has been sent it can't be preempted.

Because the packet sizes are small the no preemption is not to bad.

Timing Analysis of the CAN Bus

Assume speed of 1Mbit/sec transmission rate the longest message

that can be sent is 8 bytes long and takes $C_i = 135\mu s$.

Assuming messages are periodic with deadlines equal to the period basic timing analysis is as follows:

- T_j period of the message;
- C_j transmission, first approximation assume $C_j = 135\mu s$ (otherwise use the equation in the notes for the message time);
- B_i blocking of lower priority message (no preemption); if all messages are assumed to take time $135\mu s$ then maximum blocking time $B_i = 135\mu s$, otherwise B_i is the largest message time of a all the lower priority tasks.

Message time C_i

There are lots of stuffing bits and various other things (CRC etc) but

the message time for a message with size s_i (in bytes) is:

$$C_i = \left(8s_i + 47 + \left\lfloor \frac{34 + 8s_i - 1}{4} \right\rfloor \right) \tau_{\text{bit}}$$

where τ_{bit} is the time to send one bit.

Jitter



- Jitter is a big problem with CAN Busses and has to be taken into account.
- The equation is nearly $R_i = J_i + w_i$ where

$$w_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j$$

(Remember Jitter calculations from earlier lecture).

But a higher priority message j can not preempt message i if j is queued after the first bit of i has been transmitted: because the arbitration has started and any higher priority message must wait until the bus becomes idle.

Timing Analysis - Continued

So we need to find the longest time taken for the first bit of task i to be transmitted without interference from higher priority messages. This is given by:

$$w_i = \tau_{\text{bit}} + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j$$

The worst-case response time of the messages includes the queueing jitter, the queueing time (the longest time before the first bit of the message is sent without interference), and the time taken to send the rest of the message (the whole message minus the first bit):

$$R_i = J_i + w_i + C + i - \tau_{\text{bit}}$$

ID filtering



- All CAN controllers will receive all frames
 - Even if the CPU is not interested in the frame
 - Much CPU performance can be wasted if messages are not filtered away (new messages might signal an interrupt)
 - ID filtering hardware throws away unnecessary frames.

CAN Error Handling



- Several types of errors:
 - Checksum error, acknowledge error, bit error, ...
- When an error is detected by a node it sends an error frame
 - starting with 6 dominant bits (000000) in a row
 - tells other nodes that an error occurred
 - other nodes then also send error frames
 - Arbitration restarts when the bus is idle.
- In effect, error frames are used to resync the protocol engine.

Transmission Errors



- Max number of errors must be bounded
- Fault hypothesis \Rightarrow
 - Error function $E(t) = \max$ time required for error signalling and recovery in any time interval of length t .
- The time equation then becomes:

$$w_i = \tau_{\text{bit}} + E(w_i) + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j$$

Transmission Errors



- CAN has a mechanism to protect against broken hardware: error counters.
- The CAN controller in a node counts failed frames and successful frames
 - When errors exceed a threshold the controller gets disconnected.

The Holistic Scheduling Problem



- When tasks on a node can both send and receive messages we have a holistic scheduling problem.
- The equations given the worst case time for tasks depends on the messages arriving at the node.
- We cannot apply the processor scheduling analysis before we get values from the bus scheduling analysis
- Similarly: We cannot apply the bus scheduling analysis before we get values from the processor scheduling analysis.
- Solution: Holistic analysis, write out lots of equations coupling together all the information and look for a fixpoint.

TTP - Time Triggered Protocol



- From TU-Vienna (Kopetz et al.) (see link on the web-page.
- Intended for X-by-wire applications (where X = Break, fly , ...).
- A lot of features built into the bus protocol.
- Conceptually similar to static cyclic scheduling.

TTP - Continued



- Every node has identical message tables, that is tables of which time slots are allocated to which messages.
- Assumes that all clocks are synchronised.
- Schedules precomputed
- Nodes only sends messages when it is allowed to.
- Of course it is more complicated.
- A clock synchronisation algorithm is needed.