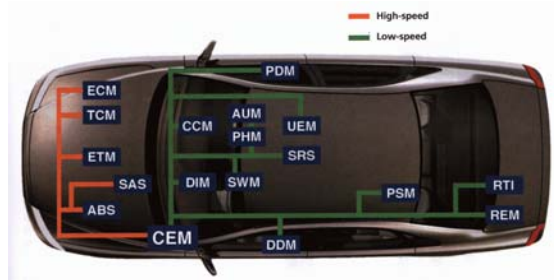
Real-Time Networks and Distributed Systems

- Topics
  - Distributed Real-Time Systems
    - Bus-based multi-processor systems
  - Real Time Networks
    - RT busses e.g. CAN, TTP, TTCAN
  - Analysis of Distributed RT Systems
    - Message Transmission Analysis
    - Response Time Analysis

A Distributed Real-Time System



Electronics in automobiles

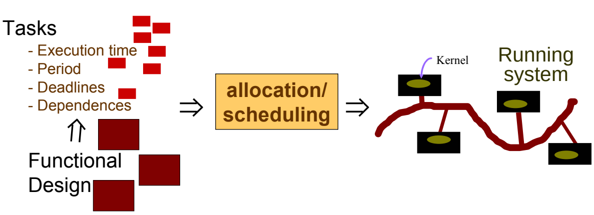
- Trend towards more and more electronics
  - 15% to 30% of component cost of a car is electronics
- Trend towards more complex systems
  - Many functions (both comfort and vehicle control)
  - Eg. Volvo S80 contains 18 major units connected via two in-vehicle networks

Why Distributed Systems ?

- **Physically distributed applications -** (close to physical equipment, e.g. engine control)
- **Capacity** (better price/performance than single CPU and much less wiring, 1200m in 1997!)
- **Modularity** (components developed in isolation)
- **Scalability** (just add another node)
- **Debugging** (easier?)
- **Fault tolerance** (errors only propagate within subpart of system)

Challenge: build complex distributed systems and maintain high reliability *at low cost*!

Dsign of Distributed RTS



3 aspects:
- off-line allocation
- run-time scheduling
- á priori schedulability analysis

The core for the development of real time systems

RT-Networking: Basic Problem

Bounded latency: A → B

- **Competing traffic**
- **Guarantees**
  - Hard-RT: Absolute G.
  - Soft-RT: Probabilistic G.
- **Other issues**
  - Reliability
    - F. detection & recovery
  - Resource Utilisation

RT-Networking: Solutions

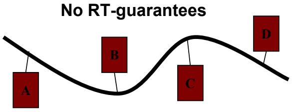
CSMA/CD

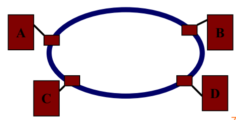
(Carrier Sense Multiple Access / Collision Detection)

- **Ethernet**
  - **Collisions**
  - **back-off**
- **Stochastic behaviour**
- **No RT-guarantees**

Token-ring

- **Physical or logical ring**
  - **Circulating token**
  - **No collisions**
- **RT-guarantees possible**





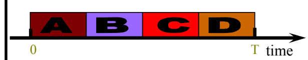
RT-Networking: Solutions

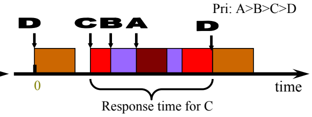
Time triggered

- **TDMA**
  - *(Time Division Multiple Access, e.g. GSM)*
- **Pre-Scheduled**
  - **Predictable**
  - **Testable**
- **Static**
- **e.g. TTP**

Event triggered

- **CSMA/CR**
  - **Priority driven**
  - **Dynamic Scheduling**
- **Flexible**
  - **e.g. CAN**



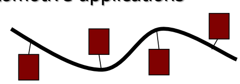


More examples

- Time triggered:
  - **SAFEbus - airplanes, eg. Boeing**
  - **TTA - cars, eg. Audi, Volkswagen**
  - **FlexRay - cars, eg. BMW, DaimlerChrysler**
- Event triggered
  - **CAN - cars, eg. Volvo, Saab, VW, Ford, GM**
  - **Byteflight - cars, BMW**
  - **LIN – a cheaper and simple bus protocol**
- Mixtures
  - **Time-triggered CAN**
  - **TTA extended with events**

Controller Area Network (CAN)

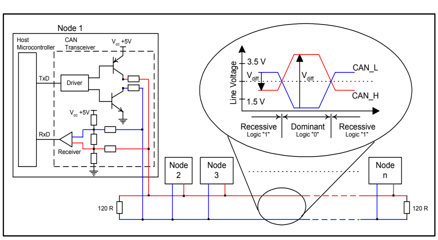
- Initiated in the late 70's to connect a number of processors over a cheaper shared serial bus
- From Bosch (mid 80ies) for automotive applications
- De facto standard for invehicle comm.
  (100 million CAN nodes sold 2000)
- Controllers available
  (from Phillips, Intel, NEC, Siemens, etc.)

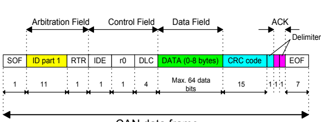


twisted-pair,
optic fibre or coax

- Shared broadcast bus (one sender many receivers) (CSMA/CR)
- Highly robust (error mechanisms to overcome disturbance on the bus)
- Medium speed:
  - **Max: 1Mbit/sec; typically used from 35 Kbit/sec up to 500Kbit/sec**

Controller Area Network (CAN)





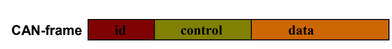
Global Time

Controller Area Network (CAN)

Frame layout:

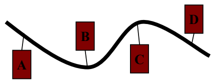
SOF, Start Of Frame	Identifier	RTR, Remote Trans-mission Request	Control	Data	CRC, Cyclic Redun-dancy Check	CRC DEL, CRC Delimiter	ACK, Acknow-ledge	ACK DEL, Acknow-ledge Delimiter	EOF, End Of Frame	IFS, Inter Frame Space
1 bit	11 bitar	1 bit	6 bitar	0–8 bytes	15 bitar	1 bit	1 bit	1 bit	7 bitar	3–, min 3 bitar

- Small sized frames (messages)
  - **0 to 8 bytes**
  - **Very different from mainstream computing messaging**
- Relatively high overhead
  - **A frame size of more than 100 bits to send just 64 bits**

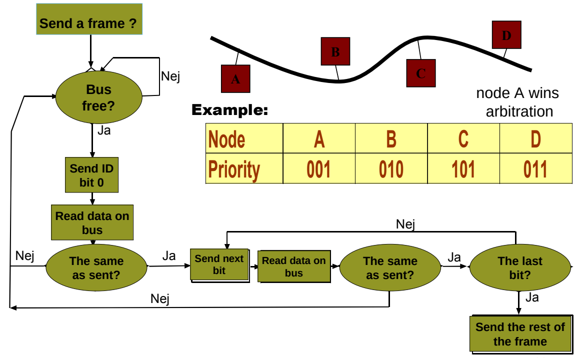


The CAN Arbitration Mechanism

- Shared broadcast bus
- Bus behaves like a large AND-gate
  - if all nodes sends 1 the bus becomes 1, otherwise 0.
- A frame is tagged by an *identifier*
  - **indicates contents of frame**
  - **also used for arbitration as "priority"**
- Bit-wise arbitration
  - **Each message has unique priority ⇒ node with message with lowest id wins arbitration**
  - **Lowest id = highest priority!**
- The CAN bus is a prio. scheduled resource



The CAN Arbitration Mechanism



Example:

Node	A	B	C	D
Priority	001	010	101	011

Details on CAN

- Each message has a priority: a unique static number, used as the identifier of the message
- Arbitration mechanism to ensure: the highest priority message is the one transmitted
- Limits on speed and length (physical/electrical properties):
  - **to send 1Mbit/sec, wire/bus must be no longer than 50m**
  - **To send 0.5Mbit/sec, the bus must be no longer than 100m**
  - **The bus can have an arbitrary number of nodes**
  - **Each station has a queue for messages ordered by priorities**

Details on CAN

- When the bus is busy, the stations wait (listening all time)
- As soon as the bus is idle, all stations who want to send enter the arbitration phase (run the arbitration algorithm)
  - **Transmit the highest priority message, from the most significant bit to the least significant one**
  - **0 is the highest priority!!**
    - **0: dominant bit (in fact, sending 0 by "high voltage")**
    - **1: recessieve bit**
  - **It behaves like an AND-gate**
  - **Send and monitor:**
    - **Send a 1, but monitor a 0: a collision**
    - **the protocol says: nodes sending 0's win, the others back off (monitor and send)**
    - **This means: the highest priority message wins, to be transmitted**
    - **E.g. 100, 101, 111 on three stations, 100 will be sent**

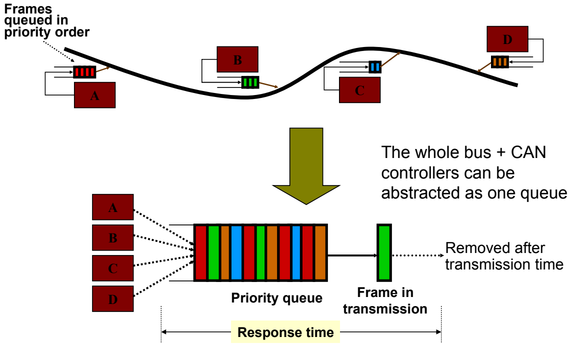
Details on CAN

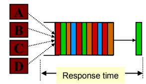
- After the priority transmitted (the arbitration is finished), the rest of the message is transmitted
- A message contains: 0-8 bytes for data and 47bits OH
  - **Priority/identity: 11bits**
  - **Data field: 0-8 bytes long**
  - **CRC field (checksum, parity bits etc: checking the message has not been corrupted, and other "housekeeping" bits)**
  - **Out of the 47, 34 bits are bitstuffed**
- 000000 and 111111 are reserved as "marker" to signal all stations on the bus
  - **So "bitstuffing" is needed: whenever 00000 or 11111 appears in a bitstream, an extra bit of the opposite sign should be added**
  - **E.g. 1111 1000 0111 1000 0111 1 should be**
    1111 1000 0011 1110 0000 1111 10

More details on CAN

- The total number before bitstuffing: 8n+47
- After bitstuffing: $8n+47+\lfloor(34+8n-1)/4\rfloor$
  - **Max: 64+47+24=135 bits**
  - **E.g. 1Mbit/sec, 1 bit needs 1 micro seconds**
  - **The max transmission time for one message= 135 micro sec**

The CAN-bus Abstraction





Transmission Delay Calculation for CAN

Set of messages = $\mathcal{M}$ (queued on different nodes)

$M_j = < T_j, C_j >$ $(M_j \in \mathcal{M})$

T_j = period (time between queuing)

C_j = transmission time

B_j = blocking time (waiting for low priority message, bus non-preemptive)

Worst-case waiting/queuing time (before transmission):

$q_i = B_i + \sum_{j \in hp(i)} \lceil q_i / T_j \rceil C_j$

hp(i) = frames with priority higher than P_i

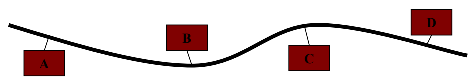
Worst-case Response time (delay before delivered):

$R_i = C_i + q_i$

Transmission delay analysis

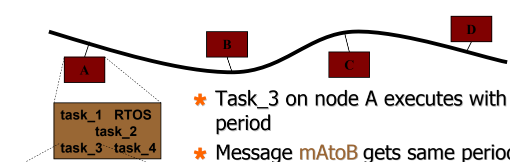
- C_i = (number of bits) X (time to transmit 1 bit)
- B_i = MAX $_{\forall k \in lp(i)}$ (Ci) <= time to transmit135 bits
- **Worst case: B_i = C_i = 135 micro sec**
  (for 1MB/sec CAN)

End of Story?



- Unfortunately not!
  - **Non-periodic queuing times causes jitter**
  - **No global time reference**
  - **Transmission errors (recovery + retransmission)**

Queuing causes jitter




```
task_3() {
  while(1) {
    read_sensor();
    if(...)
      // do some work
    else
      // do some other work
    send_CAN(mAtoB, prio);
    // some more work
    sleep_until_next_period();
}
```

- Task_3 on node A executes with certain period
- Message mAtoB gets same period as task_3
- Shortest time before send: BCET = $C_{3\ min}$ for task_3
- Longest time before send: task_3's worst case response time = R_3
- $R_3 - C_{3\ min}$ = jitter for message mAtoB

Adding Jitter to the Analysis

New equation for worst-case Transmission Delay:

$$R_i = C_i + J_i + q_i$$

$$q_i = B_i + \sum_{j \in hp(i)} \lceil (q_i + J_j) / T_j \rceil C_j$$

Error handling

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

Transmission Errors

- Max number of errors must be bounded
- Fault hypothesis ⇒
  - **Error function E(t) = max time required for error signalling and recovery in any time interval of length t**

New equation for worst-case transmission delay:

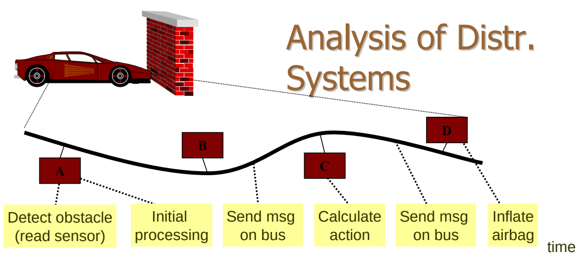
$$R_i = C_i + q_i$$

$$q_i = B_i + E(q_i) + \sum_{j \in hp(i)} \lceil (q_i + J_j)/T_j \rceil C_j$$

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**
- ERROR-counter **EC**
  - **EC:= EC+1 when an error is signalled**
  - **EC:= EC-1 when a frame is correctly received**
  - **EC > K ⇒ the node shuts-off itself (is fail-silent)**

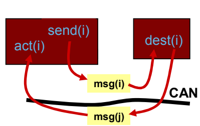
Analysis of Distr. Systems



- System wide (end-to-end) timing requirements
  - **control closed over the entire system**
  - **includes sensors, CPUs, controllers, busses, actuators, OS, …**
- *Holistic analysis* can be applied!

Holistic Scheduling Problem

- When tasks on a node can both send and receive messages we have a *holistic scheduling problem*
- The equations giving the worst case time for tasks depends on 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*

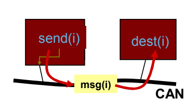
Distributed Systems

- Tasks on CPUs are exchanging msgs over CAN
- Tasks are queuing messages
  - **Completion times will vary =>**
  - **Jitter (variations in release times) will be inherited**
- Message m(i), queued by a task *send(i):*

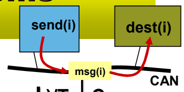
  $\Rightarrow J_{m(i)} = R_{send(i)} - C_{send(i)}$

- Task *dest(i)* is activated by a message m(i):

  $\Rightarrow J_{dest(i)} = R_{m(i)} - C_{m(i)}$



Distributed Systems



* Example:

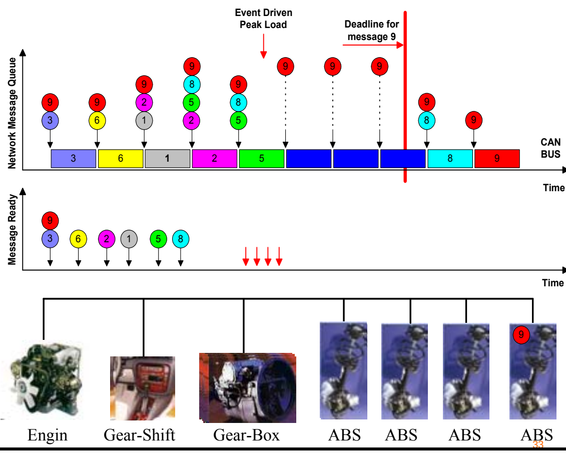
Node A:
- $R_{send(i)} = C_{send(i)} + \sum_{j \in hp(send(i))} \lceil (R_{send(i)} + J_j)/T_j \rceil C_j$
- $J_{m(i)} = R_{send(i)} - C_{send(i)}$

CAN:
- $R_{m(i)} = w_{m(i)} + J_{m(i)}$
- $w_{m(i)} = C_{m(i)} + B_{m(i)} + \sum_{j \in hp(m(i))} \lceil (w_{m(i)} + J_{m(j)}) / T_{m(j)} \rceil C_{m(j)}$

Node B:
- $J_{dest(i)} = R_{m(i)} - C_{m(i)}$
- $R_{dest(i)} = w_{dest(i)} + J_{dest(i)}$
- $w_{dest(i)} = C_{dest(i)} + \sum_{j \in hp(dest(i))} \lceil (w_{dest(i)} + J_j)/T_j \rceil C_j$

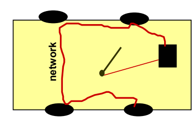
$C_{m(i)} = B_{m(i)}$ = 135 micro sec

Problem with CAN: some of the message may never get a chance for transmission

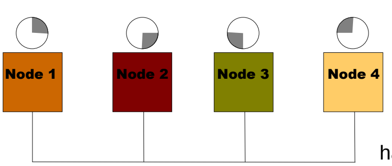


Other Solutions:
e.g.TTP - the Time Triggered Protocol

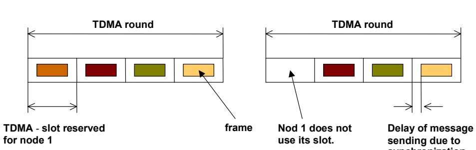
* Intended for X-by-wire applications
  Example: Break-by-wire in car
* A lot of features built in into the bus protocol
  (which must be added on top of the CAN bus)
* Conceptually similar to
  static cyclic scheduling



TTP - Time Triggered (TDMA)

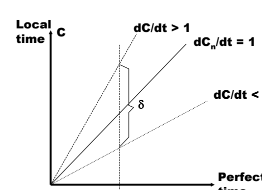


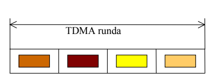
All nodes has identical message tables



TTP - Clock Synchronization

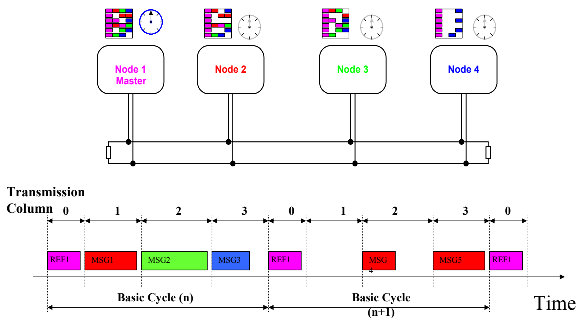
* All nodes must have the same time
* Clocks synchronized within bound δ
* Adjust by speeding up or down
* Messages used for sync.
  - Expected arrival (EA)
  - Real arrival (RA)
  - All clocks set to average





EA	0	25	50	75
RA	0	23	52	79
Diff	0	2	-2	-4

TTCAN: an example of TTP



TTP - CAN: a comparison

TTP

- **Time triggered**
- **Overallocation of aperiod messages**
- **No jitter**
- **Ultra-reliable systems**
- **Includes distributed syst functionality**
  - **Clock-synchronization**
  - **Fault-handling**
  - **Membership protocol**
- **Capacity 10 Mb/sec**

CAN

- **Event triggered**
- **No message sending if not neccessary**
- **Jitter due to varying system loads**
- **Priority driven**
- **RT-Network**
- **Some functionality added on top**
- **Capacity: 1Mb/sec**

Trends for RT networks in Automotives

- Today CAN dominates
- Time-triggered seems to be the future for X-by-wire: TTP e.g. FlexRay, TTCAN
- Future cars will include many different and parallel buses:
  - **CAN for comfort**
  - **TT for X-by-wire**
  - **MOST for multimedia**
  - **etc.**