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

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!!

Design of Distributed RTS

- Tasks: Execution time, Period, Deadlines, Dependencies
- Functional Design
- allocation/scheduling

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

RT-Networking: Basic Problem

Bounded latency: $A \rightarrow B$

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

Time triggered
- TDMA
- Pre-Scheduled
- Testable
- Static
  - e.g. TTP

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

More examples
- Time triggered:
  - SAFEbus - airplanes, e.g. Boeing
  - TTA - cars, e.g. Audi, Volkswagen
  - FlexRay - cars, e.g. BMW, Daimler/Chrysler
- Event triggered:
  - CAN - cars, e.g. 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 Philips, Intel, NEC, Siemens, etc.
- 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)
- Frame layout:
  - 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

Controller Area Network (CAN)
- Nodes
- Transceiver
- Frame layout:
  - CAN-frame
  - 11 bits
  - 36 bits
  - 6-8 bytes

Controller Area Network (CAN)
- Controller Area Network (CAN)
- Global Time
- Frame layout:
  - CAN-frame
  - 11 bits
  - 36 bits
  - 6-8 bytes
The CAN Arbitration Mechanism

- Shared broadcast bus
- Bus behaves like a large AND-gate
  - if all nodes sends 0 the bus becomes 0, otherwise 1.
- 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
- The CAN bus is a pri. scheduled resource

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

More details on CAN

- The total number before bitstuffing: 8n+47
- After bitstuffing: 8n+47+(34+8n-1)/4
  - 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

The whole bus + CAN controllers can be abstracted as one queue

Set of messages \( M \) (queued on different nodes)
\[ M_j = (T_j, C_j) \]  \( (M_j \in M) \)
\( T_j \) = period (non-transmission time)
\( C_j \) = transmission time
\( B_j \) = blocking time (waiting for low priority message, bus non-preemptive)

Worst-case waiting/queuing time (before transmission):
\[ q_i = B_i + \sum_{j \in hp(i)} \left[ q_i/T_j \right] C_j \]

Worst-case Response time (delay before delivered):
\[ R_i = C_i + q_i \]

Transmission Delay Calculation for CAN

Transmission delay analysis

\[ C_i = \text{(number of bits)} \times \text{(time to transmit 1 bit)} \]
\[ B_i = \max \{ \forall k \in lp(i) \} \text{(time to transmit 135 bits)} \]

Worst case: \( B_i = C_i = 135 \text{ micro sec} \) (for 1MB/sec CAN)

Queuing causes jitter

Task_3 on node A executes with certain period
Message mAtoB gets same period as task_3
Shortest time before send:
BCET = \( C_i \) min for task_3
Longest time before send: task_3's worst case response time = \( R_3 \)
\[ R_3 \times C_i \text{ min} \] = jitter for message mAtoB

Adding Jitter to the Analysis

New equation for worst-case Transmission Delay:
\[ R_i = C_i + J_i + q_i \]
\[ q_i = B_i + \sum_{j \in hp(i)} \left[ (q_i + J_j)/T_j \right] C_j \]
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 $E(t) = \max \text{time required for error signalling and recovery in any time interval of length } t$

  New equation for worst-case transmission delay:
  $$ R_i = C_i + q_i + \sum_{j \in \text{hp}(i)} \left( \frac{(q_i + J_j)}{T_j} \right) C_j $$

Analysis of Distr. Systems

- System wide (end-to-end) timing requirements
  - includes sensors, CPUs, controllers, buses, 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 $send(l)$:
  $$ J_{m(i)} = R_{send(l)} - C_{send(l)} $$
- Task $dest(l)$ is activated by a message $m(i)$:
  $$ J_{dest(l)} = R_{m(i)} - C_{m(i)} $$
**Distributed Systems**

Example:

Node A:

- \( R_{\text{send}}(i) = C_{\text{send}}(i) \sum_{j \in h_p(\text{send}(i))} \left[ (R_{\text{send}}(j) + J_{\text{send}}(j)) / T_{\text{send}}(j) \right] C_{\text{send}}(j) \)

- \( J_{\text{send}}(i) = R_{\text{send}}(i) - C_{\text{send}}(i) \)

- \( W_{\text{send}}(i) = C_{\text{send}}(i) + B_{\text{send}}(i) + \sum_{j \in h_p(\text{send}(i))} \left[ (W_{\text{send}}(j) + J_{\text{send}}(j)) / T_{\text{send}}(j) \right] C_{\text{send}}(j) \)

Node B:

- \( R_{\text{dest}}(i) = C_{\text{dest}}(i) \sum_{j \in h_p(\text{dest}(i))} \left[ (W_{\text{dest}}(j) + J_{\text{dest}}(j)) / T_{\text{dest}}(j) \right] C_{\text{dest}}(j) \)

- \( J_{\text{dest}}(i) = R_{\text{dest}}(i) - C_{\text{dest}}(i) \)

- \( W_{\text{dest}}(i) = C_{\text{dest}}(i) + B_{\text{dest}}(i) + \sum_{j \in h_p(\text{dest}(i))} \left[ (W_{\text{dest}}(j) + J_{\text{dest}}(j)) / T_{\text{dest}}(j) \right] C_{\text{dest}}(j) \)

**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 have identical message tables

**TTP - Clock Synchronization**

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

\[ \delta \frac{dC}{dt} > 1 \]

\[ \delta \frac{dC}{dt} < 1 \]

\[ \frac{dC_n}{dt} = 1 \]

\[ \frac{dC}{dt} < 1 \]
TTCAN: an example of TTP

TTP - CAN: a comparison

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

CAN
- Event triggered
- No message sending if not necessary
- 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.