

Real-Time (Paradigms) (59)

Impact of TTRT on available bandwidth (Throughput):

Token algorithms incur the following overheads constantly per cycle, regardless of the data volume transmitted:

Medium propagation delay: It takes a certain time for a message to propagate from one node to the next.

Token transmission time: Sending out the token takes some time. Since the token is usually much smaller than a frame that contains information, this overhead is typically very small.

Token capture delay: There is usually some time lag between when a node captures the token and when it begins transmitting.

Network interface latency: At each network interface, the input is retransmitted to the output (except for packets that are removed from the ring). The network interface latency is the time between when a bit is received by the network interface and when it is retransmitted.

Let the overall overhead of one cycle be O --> the useful time for message transmission per cycle is $TTRT - O$.

The utilization of the medium is upper-bounded by

$$\Psi = \frac{TTRT - O}{TTRT}$$

--> the throughput is reduced to ΨB (B is the bandwidth bits/time).

→ Trade-off:

A smaller TTRT leads to a smaller upper bound of the token delay, but also to a smaller throughput ΨB .

Real-Time (Paradigms) (60)

Analysis of the Timed-Token Protocol

Theorem 2:

The total duration of L consecutive cycles of the token is upper bounded by $(L+1)TTRT$, for $L = 1, 2, \dots$

Corollary 1: Over any interval of duration I , node n_i will be able to transmit at least:

$$\left\lfloor \frac{I}{TTRT} - 1 \right\rfloor f_i TTRT B \quad \text{bits (} B \text{ is the bandwidth bits/time)}$$

Supporting periodic messages

Corollary 2: If a node n_i wants to send with a period of P_i : $TTRT \leq \frac{P_i}{2}$ (1)

Corollary 3: (Computation of the synchronous quota (fraction of time reserved for synchronous messages c_i))

If node n_i has to send c_i bits per period P_i , then
$$\left\lfloor \frac{P_i}{TTRT} - 1 \right\rfloor f_i (TTRT - O) B \geq c_i$$

This equation can be solved for f_i , the upper bound for c_i .

Both equations are necessary and sufficient conditions for node n_i to be able to transmit c_i bits of real-time data every P_i seconds.

Real-Time (Paradigms) (61)

Initialization of the Timed-Token Protocol

- In the 1. cycle, each node n_i sends out its desired TTRT (desired $P_i/2$ if periodic tasks)
- The smallest requested TTRT is chosen
- The node having requested the smallest TTRT generates the token. If two nodes request the same TTRT, the tie is broken by the node-id
- The token site computes and distributes the fraction f_i
- During the 2. cycle of the token only synchronous packets are permitted
- Now the protocol is in the steady state

Real-Time (Paradigms) (61a)

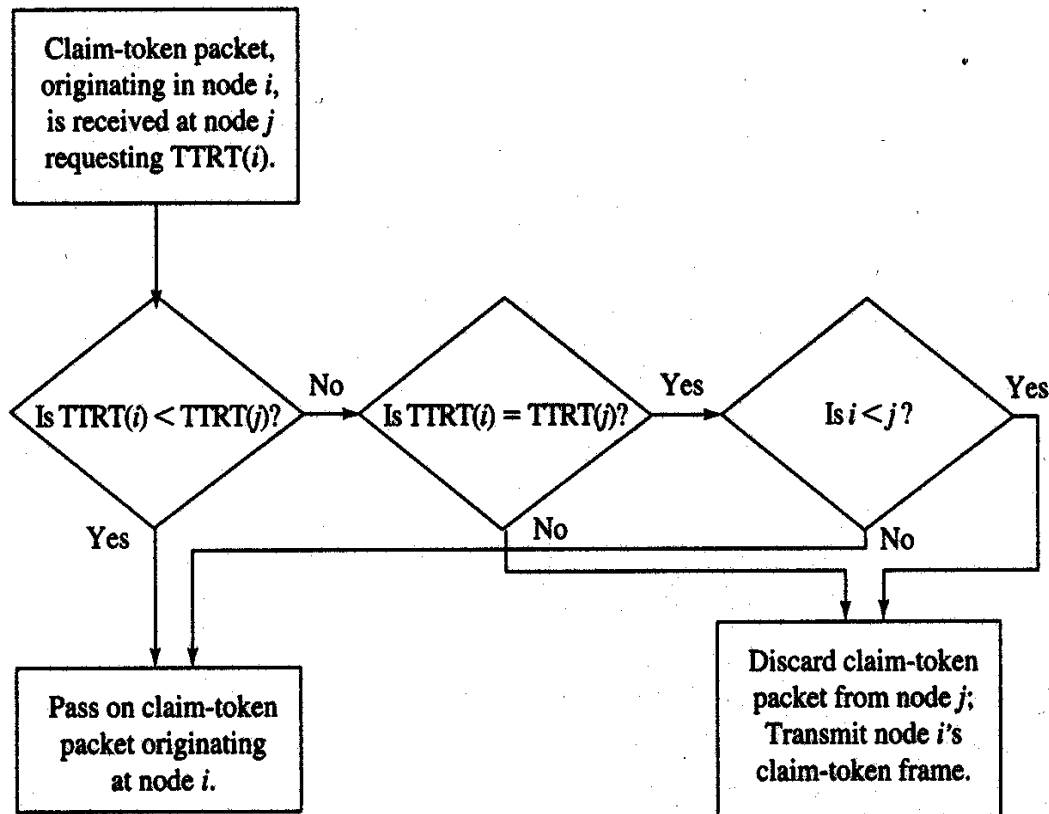
Fault-Tolerance

The Timed-Token protocol is vulnerable if a token gets lost.

Token loss is easily detected by each node n_i when $C(m, i)$ exceeds $2TTRT$.

In case of token-loss, new initialization starts by sending claim-token packets

Handling the claim-token packet



Real-Time (Paradigms) (62)

Static TDMA (Time Domain Multiplexed Access)

Medium access:

Each node is allowed to send messages only during a predetermined time span, called its TDMA slot. The allocation of these slots is determined at system's design time. During run-time the nodes maintain a global clock and each nodes exactly knows which messages can be expected in the next slot.

Pros:

- Hard real-time capable by construction
- No control messages (e.g. tokens) or bus arbitration required
- A-priori knowledge can be used for fault detection

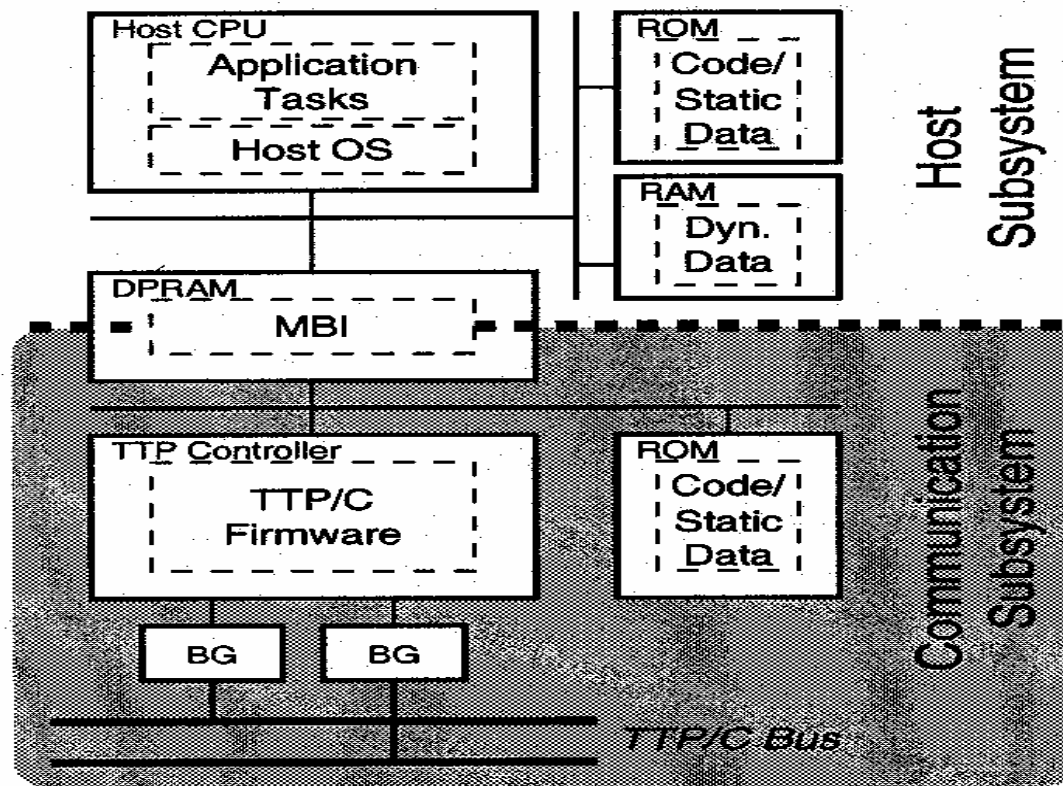
Cons:

- No flexibility, everything must be known a-priori (static scheduling required)
- Can be inefficient (e.g. all sporadic messages must be mapped to periodic slots)
- raises problems (waste or lack of bandwidth) when number of users varies
- inherently inefficient for most general-purpose computer systems, since data traffic is extremely bursty (1000:1 ratios)
- Only real-time traffic considered

Implementation of static TDMA : TTP/C (Time-Triggered Protocol)

Real-Time (Paradigms) (64)

Structure of a TTA node computer:



Real-Time (Paradigms) (64a)

Fault-Handling Strategy

TTP assumes that a node either sends

- a correct message at the correct (specified) point in time
- no message (crash or omission failure)
- To tolerate message losses, each message is send twice in one FTU slot (one produced by each node) and TTP uses 2 independent wires for transmission ---> 4 physical copies of each message are transmitted
- a detectably incorrect message at the correct point in time
- a correct message at an incorrect point in time
 - Babbling Idiot Avoidance
means the spontaneous transmission of senseless messages at arbitrary points in time by Bus Guardians:
If a node tries to send a message outside its TDMA slot, a separate controller (the “Bus Guardian”, one per network wire) terminates the operation and initiates self-check and recovery of the node

Real-Time (Paradigms) (65)

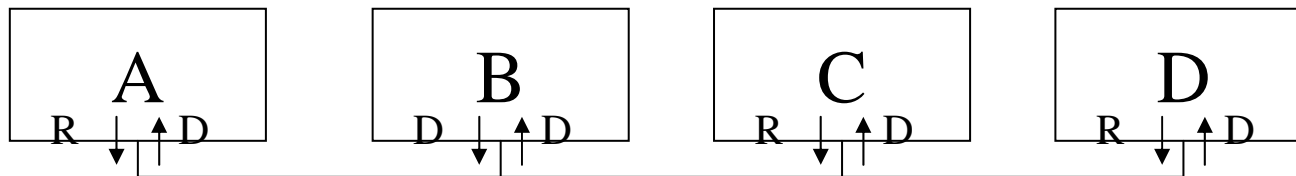
Polled Bus Protocol

Medium access:

The bus maintains a *busy*-line. This line is used for for priority-based bus arbitration and during transmission for enforcing mutual exclusion on the broadcast medium.

Busy-line: executes wired-OR

- the line has two states *dominant* and *recessive*
 - if one node assigns *dominant* to the busy-line, all node perceive *dominant*
 - the time on the bus is divided into equally long bit-times (slots)
 - one bit-time is long enough to propagate the signal to all participants
- > Bit-time is a function of the bus-length



Mutual exclusion:

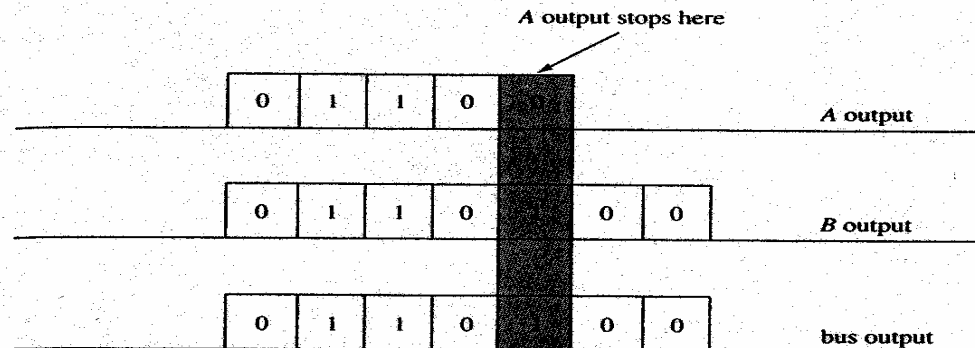
- During a message transmission the sending node keeps the busy signal *dominant*
- As long as a node perceives the busy signal *dominant*, it doesn't start a new transmission.

Real-Time (Paradigms) (66)

Bus-arbitration:

- A node starts sending if it detects no *dominant* signal on the busy-line for one bit-time.
- Then it starts sending a unique bit-string (1 = *dominant*, 0 = *recessive*) of predefined length (the binary coding (poll number) of the messages' priority) on the busy-line, one bit per bit-time
- At the same time it listens to the received value. If it receives a different value than the one it sends, it aborts arbitration.
- If all bit-values have been transmitted successfully, it keeps the busy-line *dominant* and sends its message.
- Type CSMA/CD+CR (“Collision Detection and Collision Resolution”)

What happens if two nodes A and B are starting arbitration simultaneously?



the message priority must be unique!--->

- Global assignment of priorities for each message and nodes a-priori or dynamically by a priority server
- Locally: Priorities are tuples (Prio, Node-id), only the Node-id is statically assigned and unique

Real-Time (Paradigms) (67)

Other solutions:

- deadline-driven scheme
- combined deadline-driven and priority scheme

Analysis of the Polled Bus Protocol

This Protocol implements a priority-based, non-preemptive resource access

Claim: A message with the highest priority is at most delayed for one message having maximal length.

- Proof:**
- a) If the busy-line is *recessive*, the message with the highest priority can be sent immediately as it will win any arbitration phase.
 - b) If the busy-line is *dominant*, the sending node has to wait for the remainder of the ongoing transmission (i.e. at most one max. message length) until the busy-line becomes *recessive* again, then see a)

---> This algorithm is acceptable in case of a small bit-time of the bus

Is there any guarantee that can be given for messages with lower priorities?

Each message can be delayed by

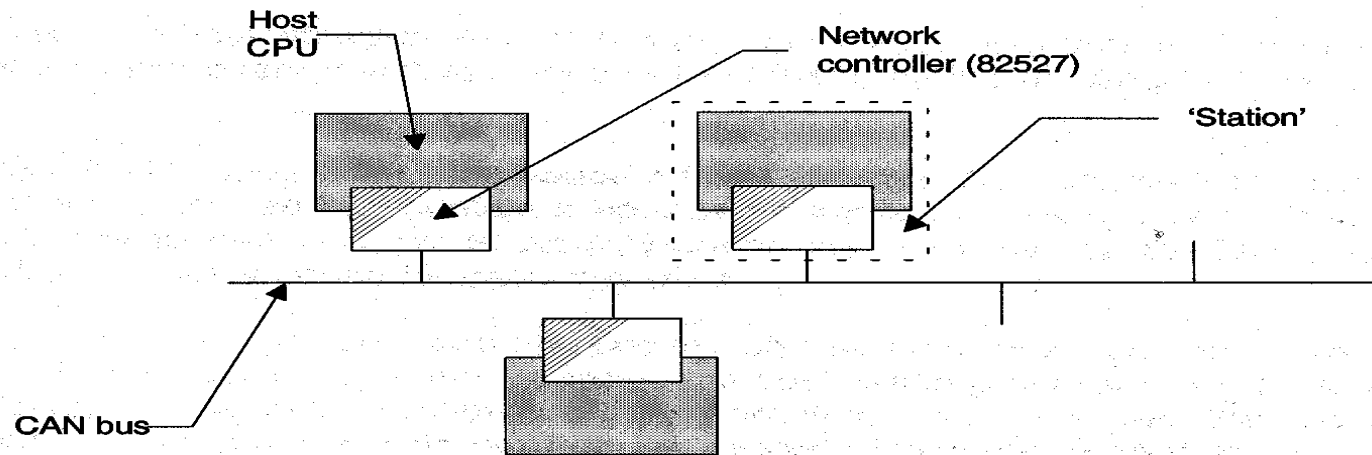
- a) a message already on the wire.
- b) by any pending message with a higher priority.

Real-Time (Paradigms) (68)

Implementation of the Polled Bus Protocol on a CAN-Bus

CAN (Control Area Network) is a fieldbus designed by BOSCH. It has the character of a broadcast communication medium where a number of processors are connected to the bus via an controller interface.

CAN Architecture



Features of the CAN-Bus

- Priority (message-identifier) length 11 bit or 29 bit
- Data-length max. 8 bytes
- Bit-time $1\mu\text{s} - 0.1 \text{ ms} \Rightarrow$ Bandwidth 1MB/s - 10kB/s
- Bus-Length 30 m - 1 km

Aim is to bound the worst-case response time (latency) of a given real-time message type.

Real-Time (Paradigms) (69)

Theorem: Given that for each message m the maximal transmission time C_m and the minimal Period T_m is known, the worst-case response time R_m of a message m (defined as the longest time between the start of a task queuing m and the latest time m arrives at the destination stations) is bounded by:

$$R_m = C_m + w_m$$

where

$$w_m = B_m + \sum_{\forall j \in hp(m)} \left\lceil \frac{w_j}{T_j} + 1 \right\rceil C_j \quad \text{with} \quad B_m = \max_{\forall k \in lp(m)} (C_k)$$

$hp(m)$ is the set of messages with a higher priority than m

$lp(m)$ is the set of messages with a lower priority than m

w_m is the waiting time (for messages with higher priority) that m remains queued before it is transmitted on the medium

B_m is the time that m is blocked by a lower priority message on the medium that started before m became queued

---> For each message a worst case delay can be computed!