Embedded Information Systems

1. Introduction (cont.)
2. Scheduling

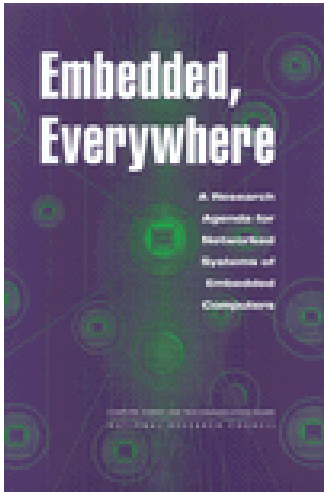
September 22, 2020

Reminder:

Embedded systems: a possible definition

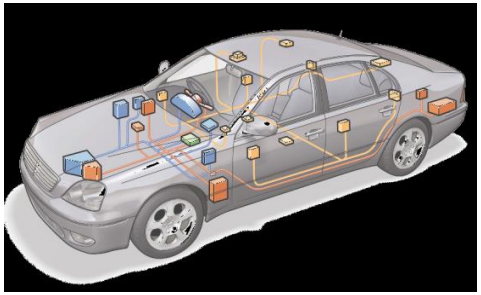
Embedded systems are **computer systems** which

- communicate intensively with their receiving physical/chemical/biological environment,
- operate autonomously,
- are highly reliable, and
- mostly “invisible”.
- Its elements have typically limited resources (memory, bandwidth, …),
- but at system level the resources prove to be ample



A Research Agenda for Networked Systems of Embedded Computers National Academy of Sciences (2001)

Fly-by-wire



Drive-by-wire

BMW 745i:
53 pcs. 8-bit,
11 pcs. 32-bit,
7 pcs. 16-bit processors,
2 000 000 line of code,
Windows CE OS,
Multiple network.

2% of the processors are used in IT and PC applications, 98% are embedded applications: vehicles, consumer electronics, mobile phones, etc.

Reminder:

The main actor is the embedded software

On one hand standardized hardware and software components (COTS) are applied, but the individual capabilities are provided by the software.

The components of the real systems interact more and more by computer mechanisms.

Within the premium category cars there are several thousand wires, and 70-100+ ECUs.

The embedded software is a universal system builder

Consequences:

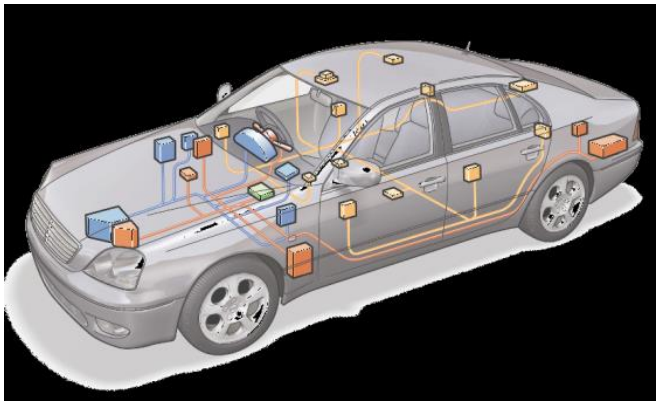
- On one hand the software absorbs its environment, while on the other it becomes part of the given application.
- The software meets both functional and physical requirements.

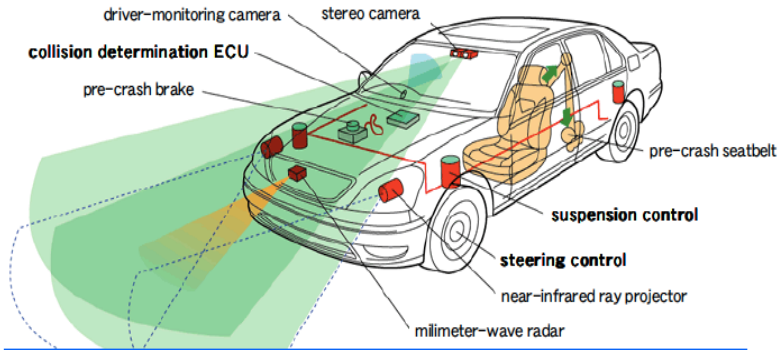
„... Software is Hard and Hardware is Soft ...”

Good news: using software many things are possible…
Bad news: using software many things are possible…

Reminder:

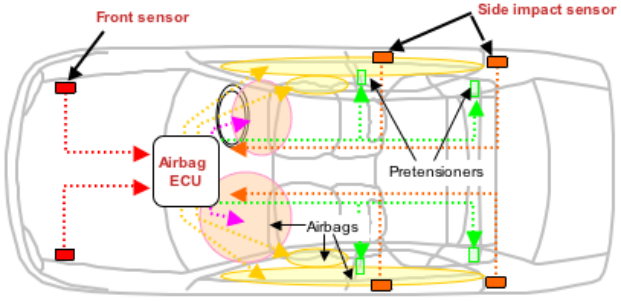
Cooperation of embedded components: systems of systems





Pre Collision Technology

Wiring harness is the 3rd most expensive car component after the engine and the body.
Wiring harness is the 3rd heaviest component after the body and the engine.
Its average weight is **100 kg**, its length **~5km**.
Half of the cost of manufacturing the wiring harness is wage.

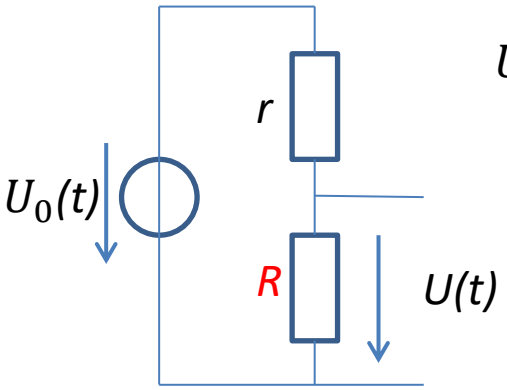


Air-bag system

Several types of automotive networks:
CAN, LIN, Flexray, MOST, TTCAN, TT-Ethernet, …

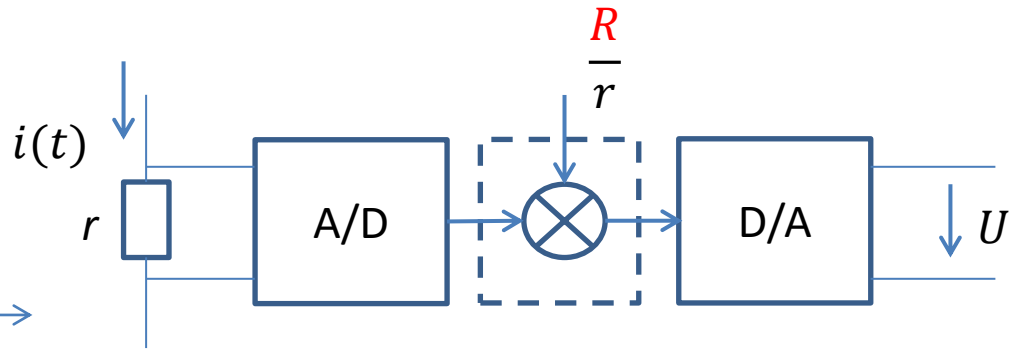
Modelling CPS systems

Example: Programmable voltage divider.

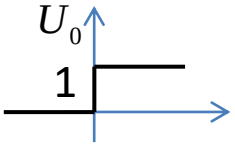


$$U(t) = U_0(t)\frac{R}{r+R} \qquad U(t) = i(t)R \qquad i(t) = \frac{U_0(t)}{r+R}$$

R can be tuned! Let's substitute R with the following circuit!



Consequence:



$$U(t=0) = 0 \qquad i(t=0) = \frac{U_0}{r}$$

μP, DSP, "cloud", …

$$U(t) = Ri(t-\Delta t)$$

$$U(t=\Delta t) = R\frac{U_0}{r} \qquad i(t=\Delta t) = \left(U_0 - R\frac{U_0}{r}\right)\frac{1}{r} = \left(1-\frac{R}{r}\right)\frac{U_0}{r}$$

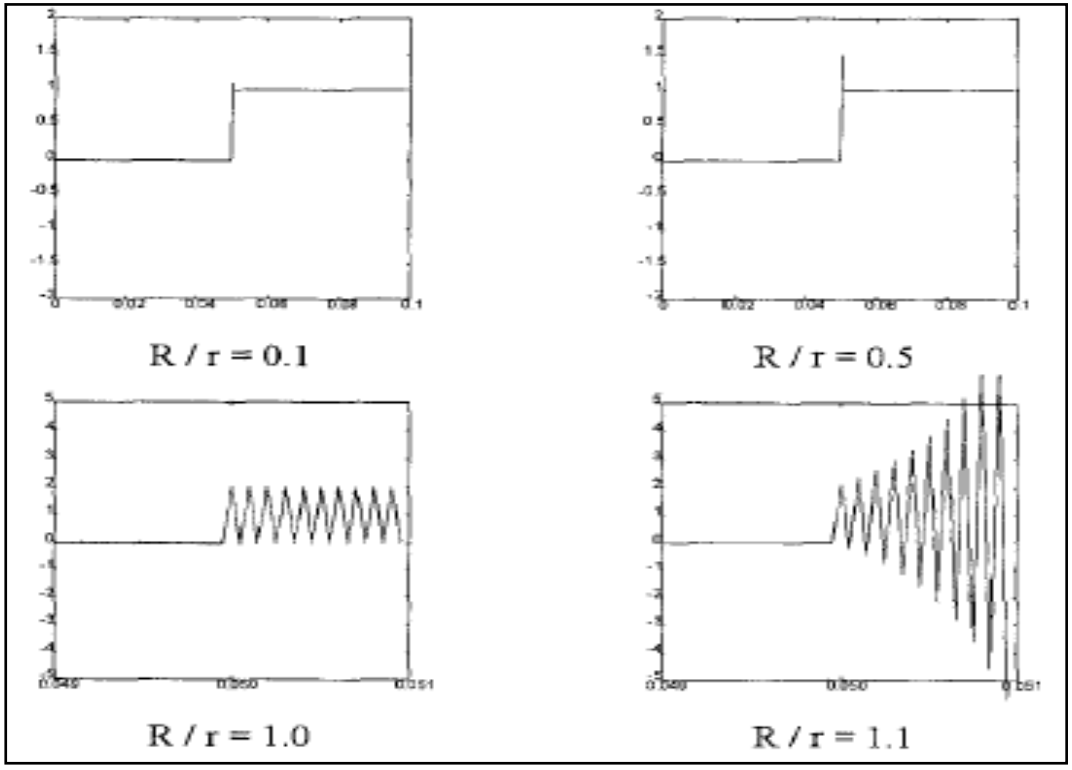
$$U(t=2\Delta t) = R\left(1-\frac{R}{r}\right)\frac{U_0}{r} \qquad i(t=2\Delta t) = \left[U_0 - R\left(1-\frac{R}{r}\right)\frac{U_0}{r}\right]\frac{1}{r} = \left(1-\frac{R}{r}+\left(\frac{R}{r}\right)^2\right)\frac{U_0}{r}$$

Modelling CPS systems

$$U(t = n\Delta t) = R\left(1 - \frac{R}{r} + \left(\frac{R}{r}\right)^2 \mp \cdots \mp \left(\frac{R}{r}\right)^{n-1}\right)\frac{U_0}{r} \to U_0\frac{R}{r+R}$$

$$i(t = n\Delta t) = \left(1 - \frac{R}{r} + \left(\frac{R}{r}\right)^2 \mp \cdots \pm \left(\frac{R}{r}\right)^{n}\right)\frac{U_0}{r} \to \frac{U_0}{r+R}$$

If $\frac{R}{r} < 1$



The delay can cause
overshoots,
damping oscillations,
constant magnitude oscillations,
growing magnitude oscillations
depending on the
parameter values.

The direct utilisation of continuous models is not always feasible!

Reminder:

Examples of specific time relations in embedded systems:

- **relativistic effect:** the time conditions of the communication through different channels may change the order of the event at the receiving node.

The figure illustrates, that in the case of client Q the message about event E2 precedes the message about event E1, which occurred earlier.

Such a situation might cause problems, if the decisions made at client Q depend on the order of the messages.

If the events E1 and E2 are not independent, after the arrival of the message about E2 to server Q, it might be reasonable to propose to wait for all those messages which were sent possibly at the same time instant or earlier as the message about E2.

This waiting time is called ***action delay***, which is the worst-case value of the possible message forwarding time for the case described above.

The necessary action delay can be calculated if the minimum and the maximum of the message forwarding time is known, i.e. for the message forwarding time the following is valid:

$$d_{min} \leq d \leq d_{max}$$

Reminder:

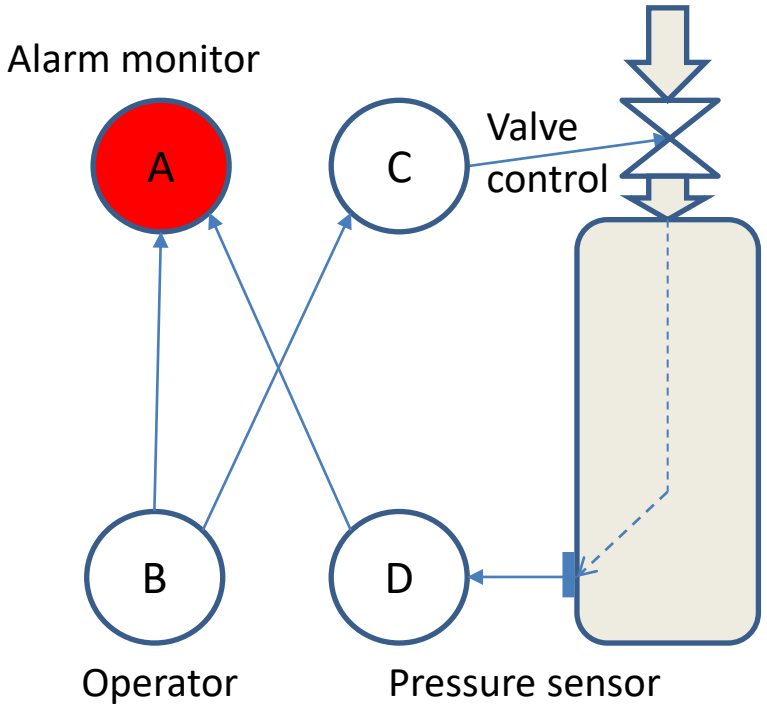
Quantities and variables in real-time systems

Example: We are monitoring the pressure within a container with a distributed system.

Node ***A***: alarm monitor,
Node ***B*** : operator,
Node ***C*** : valve control,
Node ***D*** : pressure sensor.
Possible messages:

$\mathbf{M}_{DA}$: indicates a drastic change of pressure,
$\mathbf{M}_{BC}$: operator command to change the valve,
$\mathbf{M}_{BA}$: It was an intentional change, no alarm.

Note: There is a hidden communication channel between the valve and the pressure sensor due to the operation of the physical system.



False alarm may occur, if
through $B \rightarrow \mathrm{C} \rightarrow D \rightarrow A$ the information runs faster, than through $B \rightarrow A$.
To avoid this all actions of the alarm monitor should be delayed.
(Certain actions can not be withdrawn: a catapult, a shooting, etc.)

Note: the technological system itself implements a communication channel!

Reminder:

Quantities and variables in real-time systems

Action delay: we must wait until the **permanence** of the message.
Calculation of the delay:(1) If the global time is available:
$t_{permanent} = t_{sent} + d_{max} + 2g$, where d_{max} is the worst-case value of the message delay, and g stands for the resolution of the clock. (2) If the global time is not available:
$t_{permanent} = t_{sent} + 2d_{max} - d_{min} + g_l$, where d_{min} is minimum message delay, and g_l stands for the resolution of the local clock.

In the second case the delay is larger, because the sending time is not known, while in the first case it can be calculated from the time-stamp sent with the message.

The difference between the two cases is $d_{max} - d_{min}$ that can be large.
In the case e.g. of a token controlled bus if the token round takes 10 *ms*, while message forwarding time is always 1 *ms*, then d_{max} = 11*ms* and d_{min} = 1 *ms*, since if the token just left we must wait for 10 *ms*.

Comments:

- To understand the calculation of the action delay, imagine that you are an external observer, who knows the time of every event, and is familiar what is known and what is not at the different nodes.
- A RT image can be utilised only after reaching **permanence**. If this time exceeds the **time accuracy** of the image, only the state estimation can help.

Reminder:

Quantities and variables in real-time systems

Example: Some engine parameters are given in the Table below together with their magnitude accuracy and the corresponding time intervals.

RT image	max. change	accuracy	accuracy in time
Piston/cylinder position	6000 rpm	0.1°	3µsec
Gas pedal position	100%/sec	1%	10 msec
engine load	50%/sec	1%	20 msec
Oil and water temperature	10%/min	1%	6 sec

Among the accuracy intervals of the RT images the difference is more than 6 magnitude.

In the case of the piston position such an accuracy can be provided only with state estimation (prediction) within the program.

New material starts here:

The magnitude error caused by the time difference between the observation and the utilisation in the case of a variable $v(t)$:

$$error(t) \cong \frac{dv(t)}{dt}[C(t_{utilisation}) - C(t_{observation})]$$

If the RT image is accurate in time, the worst-case error is:

$$error = \underbrace{max}_{\forall t}\left|\frac{dv(t)}{dt}\right| d_{accuracy}$$

In case of balanced design this value should be in the range of the magnitude measurement error.

To provide accurate calculations based on the RT images, we must meet the following condition:

$$[C(t_{utilisation}) - C(t_{observation})] \leq d_{accuracy}$$

Quantities and variables in real-time systems

Example for validity in time:

September 14, 1993. Warsaw Airport: A Lufthansa Airbus A320 could not stop on the runway: 2 dead, 54 injured.

The accident was caused by a design error of the control logic.

For nine seconds the plane relied only on one wheel, and because the braking mechanisms were allowed to operate only if the wheels on both sides rely on the ground, the plane could not decelerate properly.

Practically the conclusion that „*the plane is in the air therefore the braking mechanisms can not be activated*" became invalid as one of the wheels landed.

A periodically updated RT image is called ***parametric***, or **phase-insensitive**, if

$$d_{accuracy} > \left(d_{update} + WCET_{message\ forwarding}\right).$$

The parametric RT image at the receiver node can be utilised without further investigations, because the updated value arrives within the accuracy interval.

A periodically updated RT image is called ***phase-sensitive***, if

$$WCET_{message\ forwarding} < d_{accuracy} < \left(d_{updating} + WCET_{message\ forwarding}\right).$$

In this case it is not sure that the update arrives within the accuracy interval: we must check the time conditions and possibly wait for the update.

Quantities and variables in real-time systems

Example: Imagine that in the previous example forwarding the gas pedal position required **4 *msec***. If the periodic updating time is less than **6 *msec***, the RT image is parametric, while if it is e.g. **8 *msec***, then it is phase-sensitive.

Phase-sensitivity can be avoided by applying appropriate sampling frequency or by the application of state estimation.

Idempotency: If – to improve fault tolerance - the very same message arrives several times to a particular node, then this set of messages is called idempotent, if the effect is the same as in the case of a single one.

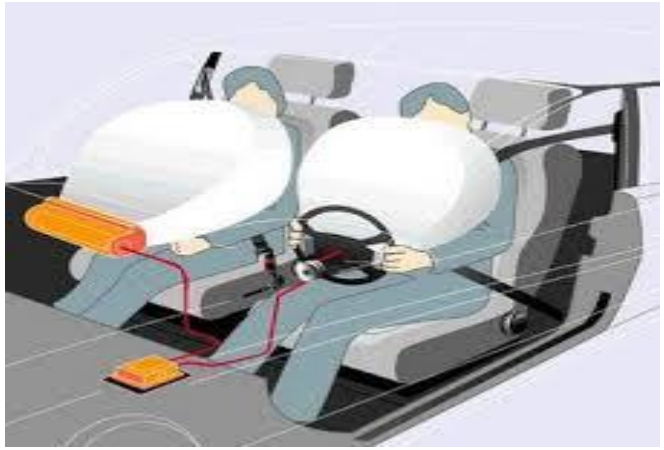
This concept is important, because if the message is only a change, then its multiple application will result in multiple corrections, while the intention was only a single one.

Example: setting a valve position to 45° (state message) ↔ changing the valve position by 5° (event message).

Hard RT systems versus soft RT systems

Hard real-time system (**HRT**): which must produce the result at the correct instant, because if we do not meet the time limitation, it might result in **catastrophic consequences**. (See e.g. the **electronic control of vehicles**).

Soft real-time system (**SRT**), online system: the result has value also if we do not meet the time limitation, only the **quality of the service** will **degrade** (See e.g. **transaction processing systems**).



Characterisation of HRT and SRT systems:

characteristic	hard real-time	soft real-time
response time	hard required	soft desired
peak-load performance	predictable	degraded
control of pace	environment	computer
safety	often critical	non-critical
size of data files	small/medium	large
redundancy type	active	checkpoint-recovery
data integrity	short-term	long term
error detection	autonomous	user assisted

***Response time*:** **HRT** systems: often in the order of **ms** or less, preclude direct human intervention during normal operation and in critical situations.

A **HRT** system must be **highly autonomous** to maintain safe operation of the process.

The response time requirements of **SRT** and on-line systems are often **in the order of seconds**.

If a deadline is missed in a **SRT** system, **no catastrophe** can result.

***Peak-load performance*:** In a **HRT** system, the peak-load scenario must be **well-defined.**

It must be guaranteed by design that the system meets the specified deadlines **in all situations**.

In a **SRT** system the **average performance** is important.

A degraded operation in a rarely occurring peak load case is tolerated for economic reasons.

***Control of pace*: A HRT system must remain synchronous with the state of the environment.** In all operational scenarios! A **SRT** can exercise some control over the environment in case it cannot process the offered load. If the computer cannot keep up with the demands of the operators, it just extends the response time and forces the operator to **slow down.**

***Safety*:**The safety criticality in **HRT** applications has many consequences for the system designer. Error detection must be **autonomous** so that the system can initiate appropriate **recovery actions** within the time intervals dictated by the application.

***Size of data files*: HRT** systems have **small** data files, that are composed of the **temporally accurate** images of the RT-entities. The key concern in **HRT** systems is on the short-term temporal accuracy of the RT database.

In contrast in on-line transaction processing systems, the maintenance of the **long-term integrity** of large data files is the key issue.

***Redundancy type*:** After an error has been detected in a **SRT** system, the computation is rolled back to a previously established checkpoint to initiate a **recovery** action.

In **HRT** systems, **roll-back/recovery** is limited utility for the following reasons:

(1) It is difficult to guarantee the deadline after the occurrence of an error, since the roll-back/recovery action can take an unpredictable amount of time,

(2) An irrevocable action which has been effected on the environment, cannot be undone,

(3) The temporal accuracy of the checkpoint data is invalidated by the time difference between the checkpoint time and the instant now.

Error detection: **HRT**: **autonomous**,

SRT: supported by the user/operator

Hardware redundancy is needed!

Event triggered (ET) and time triggered (TT) systems

The **event triggered** systems execute the program associated with the event immediately after the arrival of the request. With this approach, we can get good response times, but if the number of (almost) simultaneous events increase, the throughput/capacity of the system might be insufficient, therefore, to meet the deadlines will be impossible.

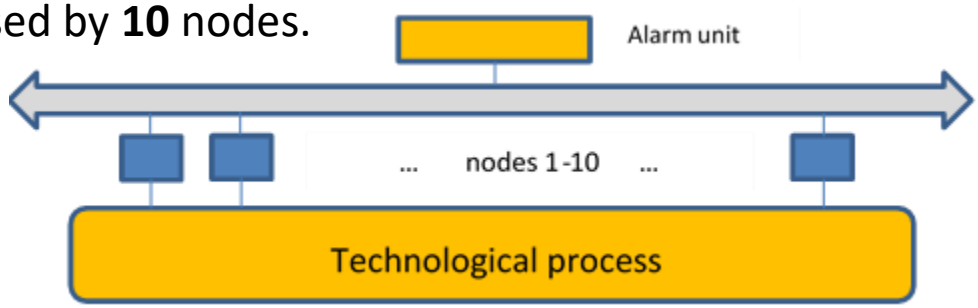
Within the **time-triggered** systems a separate timeslot is assigned to every task in design time, Thus, if the response times are a priori known, the program execution can be guaranteed.

Example: A technological process is supervised by **10** nodes.

Each node monitors **40** binary signals (alarm signals, e.g. limit crossings, etc.).

The communication is solved via a **bus**.

A system-level **alarm unit** is also connected.



The speed of the bus is **100 kbit/s**.

The monitoring nodes should send an alarm message to the alarm unit within **100 ms.**

Event triggered operation: ET/CAN protocol is applied. The shortest message is **one byte.**

According to the protocol the message will contain: **44 bits** overhead, **1-byte** data, **4-bit** length inter-message gap. The total size **is 56 bits.**

100 kbit/s means, that within **100 ms 10 000 bits** can get through.If the messages are of **56 bits**, then 10 000/56 ~ **180** messages can arrive to the alarm unit **within the specified time.**

Since **180 < 400**, therefore it is not possible to send all the possible changes, the communication channel for **~180** simultaneous messages will **completely saturate.**

Time triggered operation: TT/CAN protocol is applied. The nodes periodically send all the signalling bits to the alarm unit.

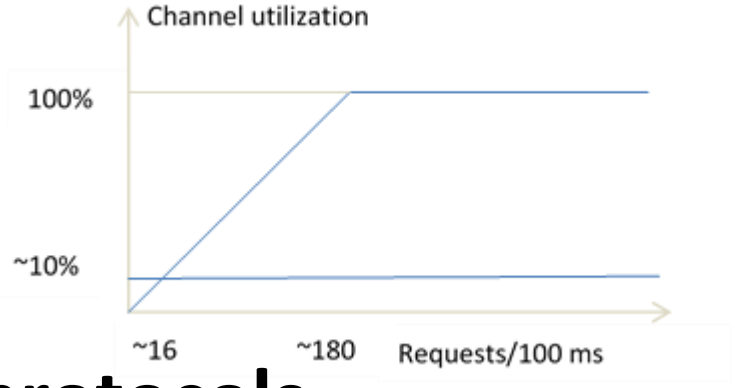
This can be performed for every **40** binary signals using a single message.

According to the protocol the message will contain: **44 bits** overhead, **5-byte** data, followed by a **4-bit** length inter-message gap. The total size is **88 bits**.

100 kbit/s means, that within **100 ms 10 000 bits** can get through. If the messages are of **88 bits**, then 10 000/88 ~ **110** messages can arrive to the alarm unit **within the specified time**.

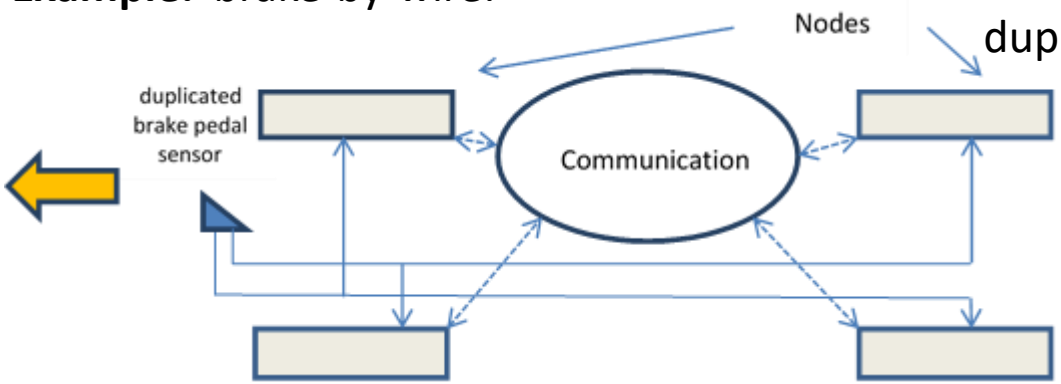
Since **110 > 10**,

thus all the signalling bits will arrive to the alarm unit, and what is more, at a load level of **~ 10%.**



The importance of agreement protocols

Example: brake-by-wire:



In this example, for safety reasons, duplicated brake pedal sensors are applied.

The brakes of each wheel have separate control nodes.

The nodes inform each other about their knowledge of the actual sensor value, and calculate the braking force.

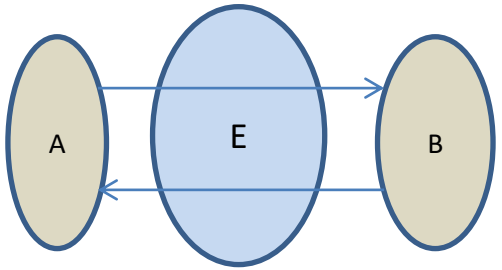
If a node is violated and fails, the corresponding wheel will run free automatically, and braking force will not be provided.

The other three nodes, after observing this situation, recalculate the braking forces, and will brake safely.

In distributed systems there are several situations where run-time agreements are needed: *time synchronisation, consistency of distributed states, distributed mutual exclusion, distributed transactions, distributed completion, distributed election, etc.*

A further problem is that even **in case of errors**, **an agreement** would be **needed.**
This is not always possible.

Example: Two armies' problem:
The allied armies, **A** and **B** together have more soldiers than the enemy (**E**), but separately each has less.

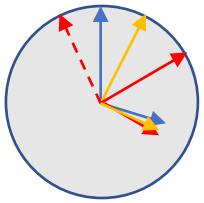


A and **B** have decided the attack, but **an agreement** upon the time **is** still **needed.**
The agreement needs communication, e.g. a messenger (**M**) should be send,
but the messenger can be captured by enemy **E**, i.e. the communication is not error-free.
If the general of army **A** sends a messenger to the general of army **B** with the message:
„***Let's attack tomorrow afternoon at four o`clock***",
an acknowledgement is needed, since the communication channel is not error-free.
(And what is more, it is also possible, that the general of army B sends a message to A with a different timing proposal.) **The problem is obvious:**
If **M** does not return to **A**, what is the conclusion? If **M** is captured on the way to **B**, **A** will be in danger, if acts alone.
If **M** is captured on the way back to **A**, then **B** might depart with a given probability, but **A** will not, because **acknowledgement** was not returned.

What we can do is to increase the probability of successful agreement.

Impossibility Result: It can be proven by **formal methods**, that to reach to an **agreement** of two or more **distributed** units in **limited time**, and through an **asynchronous medium**, which is **lossy**, cannot be **guaranteed**.

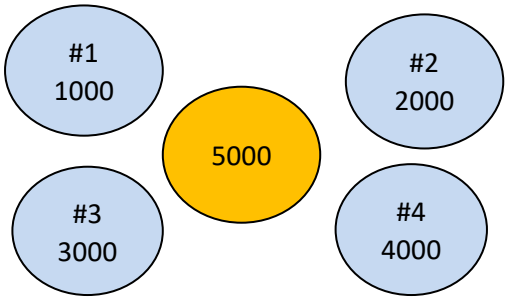
Agreement in case of Byzanthine errors: **Example**: Synchronisation of clocks:



The position of clock A is 4h00m, and that of B is 4h05m.
Clock C does not operate properly, because communicating with A sends 3h.55m, and to B 4h10m. This type of error is called **Byzanthine error**.
In such a situation the agreement **is not possible**, because both clocks A and B realize that their value is the arithmetic mean of the two other clocks, thus **there is no reason** to change!

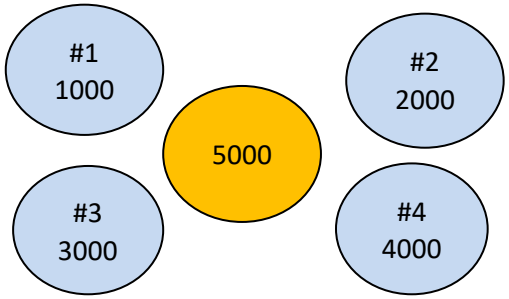
To filter out a node with Byzanthine error is possible only if at least **3*k*+1** nodes participate in the synchronization, where ***k*** is the number of nodes having Byzanthine error.
In our case one further correctly operating clock-node (D) is required.

Example: Problem of the Byzanthine generals:



According to the figure, the generals of the "blue" armies **try to agree** the total number of soldiers available for **a joint action**.
In the meantime, it turns out that **one of the generals** sends **wrong information.** (There is a **bug** in the software.)
The enemy has 5000 soldiers.

The generals of the allied armies inform each other about the number of the available soldiers.
Suppose that they can communicate **without any error!**



At the different nodes the following data are available:

#1: (1K, 2K, xK, 4K),

#2: (1K, 2K, yK, 4K),

#3: (1K, 2K, 3K, 4K),

#4: (1K, 2K, zK, 4K),

where **x, y, z** are values which differ from the true values and from each other, because the general of node #3 sends **wrong data** (software bug).

For nodes #1, #2 and #4 **this error is not known**, only the given data are available.
To check the values, all the nodes send their information vector to the other nodes.

The node with Byzanthine error will send **wrong values**.

Finally, the information available at the correctly operating nodes (in unit of 1000 soldiers):

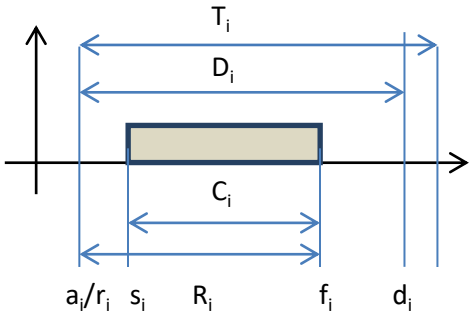
#1: $\begin{bmatrix} 1 & 2 & y & 4 \\ a & b & c & d \\ 1 & 2 & z & 4 \end{bmatrix}$ #2: $\begin{bmatrix} 1 & 2 & x & 4 \\ e & f & g & h \\ 1 & 2 & z & 4 \end{bmatrix}$ #4: $\begin{bmatrix} 1 & 2 & x & 4 \\ 1 & 2 & z & 4 \\ i & j & k & l \end{bmatrix}$

The generals of the three properly operating nodes will get the same information from two nodes, but from the third, from general #3, the information is different.

Their conclusion is $[1 \quad 2 \quad unknown \quad 4]$, i.e. minimum **7000** soldiers will participate, and the general of node #3 is the source of **wrong information**.

2. Scheduling

Problem: The processors should execute **various tasks** with **different timing** requirements. The timing conditions can be interpreted using the following figure:



Here $\mathbf{a_i}$ or $\mathbf{r_i}$ is the **arrival/release/request time**,
$\mathbf{s_i}$ is the **start time** of execution, its **finishing time** is $\mathbf{f_i}$,
$\mathbf{d_i}$ stands for **deadline**, T_i is the **period time**,
$\mathbf{D_i=d_i-a_i}$ is the **deadline** relative to the **request time**,
$\mathbf{C_i}$ stands for **computation time,** and $\mathbf{R_i=f_i-a_i}$ is the **response time**.

1. Periodic scheduling: this is the simplest method: in design time fix time slots are assigned for the completion of the periodic requests, and this is repeated periodically.

The assignment is typically **clock-driven**; therefore these types of scheduling are called **time-triggered.** They have **different** versions, but what is **common**: the decisions concerning schedules are made **in design-time**, thus their run-time **overhead is low**.

A further feature is that **the parameters** of the **HRT** tasks **are known** and **fixed** in advance.

Example: To each task there is assigned a frame of **10 ms**. **4 functions** are implemented:

The first function operates with a periodicity of **50 Hz**, i.e. it receives **10 ms** in every **20 ms**.

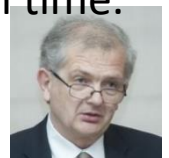
The second function operates with a periodicity of **25 Hz**, i.e. it receives **10 ms** in every **40 ms**.

The third function at a rate of **12.5 Hz**, i.e. in every **80 ms** receives **10 ms.**

The fourth function at a rate of **6.25 Hz**, i.e. in every **160 ms** receives **10 ms.**

Obviously, the assignment of the frames can be different; it can be done only in design time.

Such a scheduling can be rather unpleasant and rigid!



Comment: In the example above the first function utilises **one half** of the processor time, the second the **one fourth**, the third the **one eighth**, etc. It worth bringing back the result:

$$\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\ldots\rightarrow 1$$

i.e. the number of the functions can be increased **to the infinity**, if the required overall processor time is always **one half** of the previous.

This property was utilised by the designers of **the first real-time** 1/3 octave **spectrum analyser** (Brüel & Kjaer 2131) based on **digital filters** in **1977**!

This device performs **1/3 octave** analysis in the frequency range of **1.6 Hz** and **20 kHz**, in 42 bands. The hardware is based on **octave filters**, having at **3 dB** attenuation frequency ratio **1:2**.

This property is utilised in the following way: imagine an **octave filter** with a centre frequency of **16 kHz** and sampling frequency of f_s=**66.667 kHz**. If the signal is properly bandlimited, then the **octave filter** with a centre frequency of **8 kHz** can be successfully operated at a sampling frequency of $\boldsymbol{f_s/2}$, etc.

The evaluation of the highest frequency range takes **one half** of the time, all the other ranges share the **second half** of the time!

2. Time-shared/round-robin scheduling:

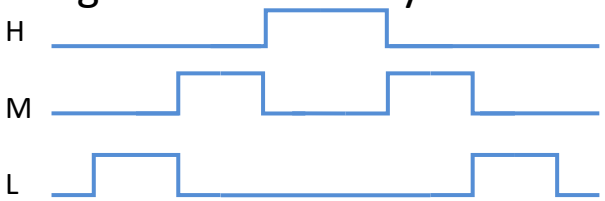
The tasks ready to run are placed into a **FIFO** (First-In First-Out), and the task **first on the list** will get the processor for **a fixed amount** of time. This time slot is typically few times 10 ms, and independent of the tasks. If the given task is not completed within the slot, it will be interrupted, and placed to the last position of the FIFO.

3. Priority based scheduling:

From the set of tasks ready to run the one having **the highest priority** will run.
Priority assignment can be performed either in design or in run-time.

For simplicity imagine that to every task a different priority level is assigned.

Illustration:



We have one **low priority** (L=low), one **medium priority** (M=medium) and one **high priority** (H=high) task.

This assignment happened in **design-time**. All the tasks start running immediately after the **request**, if their priority is **the highest** among the tasks **ready** to run.

The **response time** of the **lowest priority** task on the figure is: $R_L=C_L+C_M+C_H$

If the medium and/or the high priority task are released **periodically**, then depending on the time relations, it might happen, that these tasks will run **more than one time** during R_L.

In a more general case, for a task at priority level i, the **worst-case response time** can be calculated using the following formula:

$$R_i = C_i + I_i = C_i + \sum_{\forall k \in hp_i} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

where I_i is the so-called **interference.** The **interference time** is the total computation time of those higher priority tasks, which prevent task i to complete its actual run. $\forall k \in hp_i$ refers those tasks, which have **higher priority** than i (*hp*=higher priority). The $\lceil \quad \rceil$ sign is the operator of assigning the **upper integer**. $\lceil 1.02 \rceil = 2$, $\lceil 2.0 \rceil = 2$. Since in the above formula the unknown R_i on the left-hand side is present also in the argument of the **highly nonlinear** function on the right-hand side, it can be evaluated only via an **iterative** procedure:

$$R_i^{n+1} = C_i + I_i = C_i + \sum_{\forall k \in hp_i} \left\lceil \frac{R_i^n}{T_k} \right\rceil C_k$$

The iteration will **stop** at step n_0 where

$$R_i^{n_0+1} = R_i^{n_0}$$

The name of this method in the literature is **Deadline Monotonic Analysis** (DMA).