Transaction-based model for real-time distributed control systems

Gh. Sebestyen[1], A. Hangan[1]

[1]Technical University of Cluj-Napoca, Gheorghe.Sebestyen@cs.utcluj.ro, Anca.Hangan@cs.utcluj.ro

Abstract-**The paper presents a transaction-based task allocation model suitable for real-time distributed control applications. In accordance with the proposed model the autonomous functionalities of a control application are implemented as chains of tasks and messages, called transactions. This model allows the evaluation of worst case response time for transactions and subsequently determines the real-time feasibility of the system. The task allocation strategy assures fault-tolerance and adaptable real-time behavior. As communication infrastructure a general purpose switched Ethernet network was considered. The transaction-based model was validated through simulations made with a Timed Petri-Net tool.**

I. INTRODUCTION

As real-time control applications become more and more distributed new task allocation models are needed, models that inherently offer facilities for real-time feasibility evaluation. In the past, hard real-time systems with critical time constraints required dedicated equipments, software components and communication infrastructures. In order to satisfy deadline restrictions, a significantly higher rate of computation power and communication throughput was used compared to the actual needs of the system. For instance in avionics, regulations impose one application per processor even if today's multi-core processors may process more tasks in parallel. In the MARS [1] or Spring [2] projects dedicated architectures were proposed for real-time purposes.

Nowadays there is a significant trend toward the use of common (general purpose) computer architectures (e.g. PCs, PDAs), programming platforms (Windows, .Net) and communication infrastructures (e.g. Ethernet, Internet) as support for real-time control applications. But an important drawback in such attempts is the lack of deterministic and predictable behavior of general purpose ITC technologies. Therefore a number of research efforts were made [3][4][5] in order to impose a more deterministic system behavior.

In the case of communication infrastructures the first approach was to develop a new class of network protocols dedicated for control applications. As a result a wide range of industrial network standards were promoted by the main manufacturers of automation equipments. Some standards such as CAN, ModBus, Profibus, ASi, P-Net are still in use in industry and offer the necessary support for real-time and safety critical control applications.

But as the speed of Ethernet networks grew from 10Mbs to 100Mbs and even to 1Gbs and the switched network topology solved the problem of packet collision, there is a significant pressure from the industrial user's community towards the use of Ethernet networks also for control applications. The arguments for such a trend are: the need for interoperability with other information systems of a company (e.g. ERP, SAP, B2B applications, etc.) and the very low cost of interfaces. Until recently Ethernet was avoided in case of hard real-time systems because of its inherent non-deterministic (collision-based) MAC mechanism.

Research effort is being made to provide Ethernet solutions for real-time communication. While an approach is to avoid collisions through implementing a medium access control mechanisms such as TDMA [1] or token-passing [6] over Ethernet, others use switches to separate collision domains [5] and traffic shapers to avoid bursts [3][4].

In this paper we show that with the use of switched network topologies collisions may be avoided and a worst-case packet transmission time can be computed.

The other aspect taken into consideration in this paper is the distribution of supervisory and control tasks along the "intelligent" nodes of the system. Usually a given control functionality requires a chain of actions or tasks (e.g. data acquisition, processing, storage/logging, visualization, etc.) placed in different equipments and interlinked through a set of specific messages. Job shop [7] is a classical model that can be used to describe end-to-end tasks in a distributed system with heterogeneous processors. Each task in a job shop is modeled as a chain of jobs that execute on different processors and for each task there is a visit sequence of processors. Message transmission over the network is considered as a job which executes on a communication processor.

In order to evaluate the real-time feasibility for a given set of control functionalities a new distributed transaction-based model is proposed. This model allows computation of the end-to-end worst case response/reaction time for each control chain and verification that deadlines are not missed.

The paper is organized as follows. Section II describes the transaction-based computational model. This model is the starting point of the end-to-end response time evaluation presented in section III. Section IV presents experiments that have been made in order to validate the proposed evaluation method. Section V concludes the paper.

II. THE TRANSACTION-BASED MODEL

Distributed control systems include sequences of tasks such as data acquisition, processing, storage, visualization, event and alarm handling, data logging etc. Usually, control system's nodes do not implement the same functions and the processors may be heterogeneous (e.g. some processors are specialized in data acquisition, other in data processing). In our approach a task may be executed on a subset of nodes, in accordance with node's capabilities and resources (e.g. existing I/O interfaces, sufficient computation power). As a consequence, some tasks can be executed in parallel, while others may have to compete for the same resources.

In the proposed model the autonomous functionalities of a distributed control application are implemented through a set of transactions. A transaction *TR* is defined as an ordered list of tasks (*t*) and messages (*m*). As an example, in the case of a temperature monitoring functionality, the transaction may be a sequence of three tasks and messages between them. The first task is data acquisition (t_1), the second is data processing (t_2) and the last is data visualization (t_3). Between tasks there are two messages: m_1 contains measured (raw) data and m_2 contains data sent for visualization. The temperature monitoring transaction is described as $TR = (t_1, m_1, t_2, m_2, t_3)$.

The model includes two types of transactions:

- time-triggered periodic transactions, released with a predefined frequency
- event-triggered sporadic transactions, caused by alarms or operator requests

A periodic transaction TR_i is defined by a set of parameters: T_i – release period, D_i – deadline, r_i – worst case response time. The components of a transaction (tasks and messages) inherit its period and deadline. The worst case response time of the transaction is equal to the response time of the last task in the transaction. A sporadic transaction has similar parameters with the difference that the "period" is the smallest interval of time in which it may occur. This time is usually determined by physical factors specific for the controlled process (e.g. the same mechanical failure does not occur twice in less than 0.1 seconds). It is considered that the execution time C_i of a task t_i and the transmission time C_j of message m_j are determined a-priori.

The tasks belonging to a transaction will be distributed in the control system taking into consideration the capabilities and resources of nodes. For example, in the case of the temperature monitoring functionality, the data acquisition task can be executed on a given node only if there is a temperature sensor attached to that node. To assure fault-tolerance and dynamic reconfigurability features it is considered that subsets of nodes have similar capabilities and a given task may be allocated to any node in a subset. It is assumed that a node contains the executable code for all tasks it can execute. When the task is allocated to a node, a new instance of that task is created and activated. More instances of the same task may be active in a node, hence competing for processor time. Tasks belonging to the same transaction communicate and synchronize through messages sent over the network infrastructure.

In the initialization phase the transactions are gradually loaded on the distributed infrastructure, starting with the most critical ones. In this process tasks are allocated to nodes in accordance with the following restrictions and criteria:

- the node has the necessary capabilities to execute the task
- the time restrictions (deadlines) for the current transaction and the previously loaded ones are fulfilled (e.g. the worst case response time for every transaction is less than its deadline)
- adding a new task to a node does not overload the node
- if possible (the other conditions are fulfilled), a uniform task allocation heuristics is applied

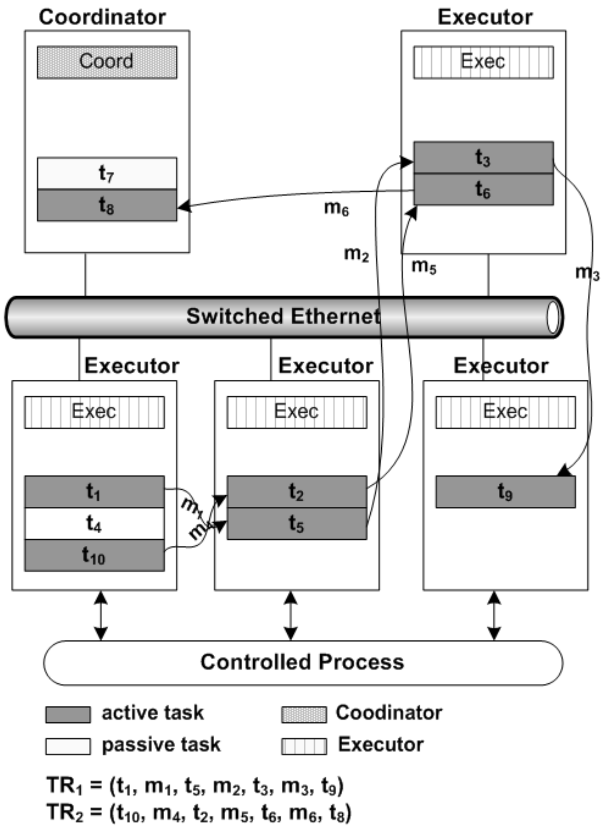

Figure 1. The distributed transaction-based model

Fig. 1 shows a set of transactions distributed over the existing nodes of the distributed system.

In order to implement the task allocation strategy one Coordinator and a set of Executor tasks are used. The Coordinator keeps track of the active nodes, their capabilities and actual loads. Based on a specially designed protocol the Coordinator periodically tests the actual state of the nodes and in case of hardware failures tries to reallocate the active transactions. If the reallocation scheme does not satisfy the real-time conditions, the Coordinator will discharge one by one the less critical transactions, until the real-time feasibility test succeeds. The Executors controls the execution of tasks inside of a node and respond to requests coming from the Coordinator. It also schedules the active tasks based on their period, priority and precedence relations. The Rate monotonic algorithm was chosen as scheduling policy at node level, because it is optimal for uniprocessor scheduling, assuming that task priorities do not change over time.

III. Evaluation of the End-to-End Response Time

As part of the feasibility test, the worst case end-to-end response time for every transaction has to be computed. If for each transaction this time is less than its deadline, then the system (set of transactions) is considered feasible from the real-time point of view. As mentioned in the previous section, the worst case response time for a transaction is equal to the worst case response time of the last task in the chain. The next two subsections present the methodology used for computing the worst case response time for tasks and for messages. The link between a task and its subsequent message is made through the "jitter time" j_i of the message [], which is measured from the release time of the transaction until the moment the message is actually released for transmission. Jitter time is equal to the worst case execution time of the emitting task. In a similar way the jitter time of a task is equal to the worst case transmission time of its precedent message. The first task in the transaction has jitter time equal to zero.

As described in [8], computing the worst case execution and transmission time for tasks and messages is an iterative process. In each iterative step jitter time is set to the worst case transmission/execution time computed in the previous step. The process ends when there is no change in two consecutive steps. The following equations describe the iterative method.

$$\begin{cases} R_{1\,(m+1)} = \Re_{RM}(J_{1(m)}) \\ R_{2\,(m+1)} = \Re_{RM}(J_{2(m)}) \\ \ldots\ldots \\ R_{n\,(m+1)} = \Re_{RM}(J_{n(m)}) \\ R_{net(m+1)} = \Re_{SW}(J_{net(m)}) \end{cases} \qquad (1)$$

where:

- $R_{i(m)}$ – is the vector of response times for all the tasks executed by node i computed in step m of the iteration
- $R_{net(m)}$ – is the vector of transmission times for all the messages exchanged through the network, computed in step m
- $J_{i(m)}$ – is the jitter computed for the tasks contained in node i
- $J_{net(m)}$ – is the jitter computed for the messages transmitted over the network
- $\Re_{RM}(J_{i(m)})$ – is the matrix formula for computing the tasks response times, using a rate monotonic (RM) scheduling algorithm
- $\Re_{SW}(J_{net(m)})$ - is the matrix formula for computing the messages transmission times in the case of a switched Ethernet (SW) network infrastructure

The last two formulas will be detailed in the next subsections.

A. Evaluation of a task's worst-case response time

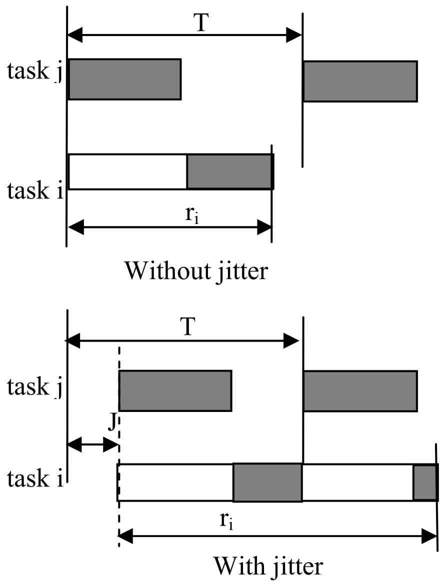

Figure 2. The effect produced by the arrival jitter of task *j* on the response time of task *i*

Given a set of periodic and independent tasks and a rate monotonic scheduling strategy the worst case response time of a task is given by the following formula [9]:

$$r_i = C_i + \sum_{j \in p_i} \left(\left\lceil \frac{r_i}{T_j} \right\rceil * C_j \right) \qquad (2)$$

where: r_i – worst case response time of task i
C_i – execution time for task i
T_i – period of task i
p_i – includes all tasks with higher priority than task i

In the above expression the term $\lceil ri/Tj \rceil$ gives the number of arrivals of task j during the response time of task i. The upper rounding operator $\lceil\ \rceil$ makes the above equation difficult to compute. An iterative method may be used to evaluate the worst response time:

$$r_i(k+1) = C_i + \sum_{j \in p_i} \left(\left\lceil \frac{r_i(k)}{T_j} \right\rceil * C_j \right) \qquad (3)$$

where: $r_i(k)$ – worst case response time of task i in iteration step k

In the first iteration, the worst case response time $r_i(0)$ is considered equal to the execution time C_i. It can be demonstrated [8] that this iterative method generates a solution in a limited number of steps, if the processor's load is smaller or equal with 100%.

A given schedule is feasible if the worst case response time of any task is smaller or equal with its deadline.

$$r_i \le D_i \qquad (4)$$

where: $i = \overline{1,n}$ and n = number of tasks

The above method may be used even in the case of sporadic tasks, if their minimum arrival interval is known. The worst-case response time considers that all the sporadic tasks arrive with a period equal to the minimum arrival interval.

If a task depends on the arrival of a message or another task's execution then the task's arrival is delayed. This delay may vary from 0 to the maximum response time of the previous item (message or task). This assumption is based on the fact that the task may be released any time during the response time of the previous item. This delay, called arrival jitter affects the worst-case response time of all tasks, which have smaller priority than the delayed task. Because of this jitter the minimum time interval between two consecutive arrivals of a periodic task may be smaller then its period. In this case the worst case response time is computed with:

$$r_i = J_i + w_i \tag{5}$$

$$w_i = C_i + \sum_{j \in p_i} \left(\left\lceil \frac{J_i + w_i}{T_j} \right\rceil * C_j \right) \tag{6}$$

Fig. 2 shows the effect of jitter time of task *j* on task *i* that has less priority. It can be observed that the execution time of task *i* is greater in the second case when the jitter time is considered.

B. Evaluation of message transmission time

In case of deterministic network protocols, such as TDMA (Time Division Multiple Access), or token passing protocols a number of methods for the evaluation of worst case transmission time were reported [1][6]. For a switched Ethernet network the transmission time may be evaluated easier if the packet losses at switch level are avoided. This can be achieved if the output buffers have enough storage capacity. This assumption is reasonable in case of high performance Ethernet switches. The only condition for the designer is to evaluate the maximum buffer capacity required for a given set of data flows.

For the proposed method the following assumptions are made:

- there are no packet losses due to buffer overflows
- the packets are transmitted to the destination channel with a FIFO (first in first out) policy a
- message sent between two control tasks fits into an Ethernet packet; usually control data has small dimensions

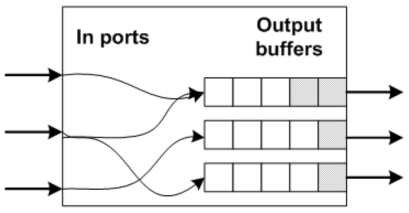

Figure 3. Inside view of a switch

- concerning the way in which messages are forwarded from the input channel to the output channel, a switch may use a store-and-forward or cut-through method
- the deadline of a transaction and implicitly of a packet is smaller than its period; under this assumption two packets of the same type do not compete for the service of the same switch

Fig. 3 gives an inside view of a switch, where packets coming from different sources are directed to the buffers associated to the output channels. The delay of a packet inside of a switch has two components:

- t_c - the switching time (a constant parameter of a switch)
- t_b- the time the packet spends in the output buffer

$$t_{SW} = t_c + t_b \tag{7}$$

In case of multiple switches a packet is delayed at each "hop". But the overall delay caused by switches is not a sum of the possible delays in every switch because some delays may be overlapped in time. For instance if there are 3 packets that follow the same rout the last arrived packet must wait only at the first switch because at the next switches it will arrive "just in time" after the other packets were transmitted. The overall transmission time of a packet from an emitting task to the destination task may be computed as:

$$t_{tr} = \sum_i t_{w_i} + t_{delay} + n_{hop} * t_t \tag{8}$$

where:

t_{wi} – the packet transmission time over cable segment *i*

t_{delay} – the overall delay for a packet caused by buffering in switches (for a given data-flow configuration)

t_t – the transmission time of the packet

n_{hop} – in case of store-and-forward it is equal with the number of hops plus one; if the cut-through technique is used it is equal to 1

The transmission time t_w over the wire is determined by the length of the segment (L_i) and the transmission speed (v) of the electric signal.

$$t_w = L_i * v \tag{9}$$

The transmission time of a packet is computed using the length of a packet (in bits) divided with the network's frequency f_{net} (in bits per second):

$$t_t = \frac{L_{packet}}{f_{net}} \tag{10}$$

The only variable part in (8) is the delay caused by buffering. The time a packet spends in a buffer depends on the other packets that may arrive in the same time to the same switch output. Because there are no priorities between packets (common Ethernet switches do not recognize priorities) the worst case delay time for a given packet should be computed in a scenario in which all the competing packets arrive in the same time to the same output. It is obvious that in case of multiple switches the delay of a packet is influenced by all the other packets that intersect its route. What is not so obvious but it can be demonstrated is the fact that all the packets that intersect the studied packet may cause a delay only once. So in the worst case the packet has to wait in different buffers for all the packets it may intersect in its route, but only once. In this case:

$$t_{delay_i} = \sum_i t_{t_i} \tag{11}$$

The worst case transmission time of a packet t_{WCi} measured from the start of a transaction is the sum between the jitter caused by the emitting task and the transmission time.

$$t_{WCi} = J_i + t_{tr_i} \tag{12}$$

Fig. 4 shows a scenario with 3 switches, 6 nodes and 4 intersecting messages. For instance message m_{DC} intersect the route of message m_{EB} only on one segment, and the route of message m_{AC} in two segments. The overall worst case delay of message m_{DC} caused by buffering is the sum of the transmission times of messages m_{AC}, m_{EB} and m_{FC}.

IV. Experimental Results

The proposed transaction allocation framework was validated through simulations using a Timed-Petri-Net (TPN) tool. Task execution times and message transmission times were simulated using transitions with firing time. Task concurrency at node level and message concurrency at network level were simulated through places and tokens. The TPN tool allowed measurement of worst case response time and average time of transactions. The analytically determined worst case times were consistent with the simulation values.

In many scenarios the average response time was much less than the worst case time. This means that in normal conditions the system's reaction time is much under the predicted one. It was observed that under similar workloads the difference between the worst case and the average time is greater if the number of tasks and messages is grater. Therefore it is more advantageous to group some of the tasks in larger ones (if possible) in order to reduce the worst case response time. For instance all the transactions which have the same period and share the same route may be merged into a single transaction.

V. Conclusions

The paper presents a solution for the task allocation problem in a real-time distributed control system. The proposed framework is based on the evaluation of worst-case end-to-end response times of task sequences and messages.

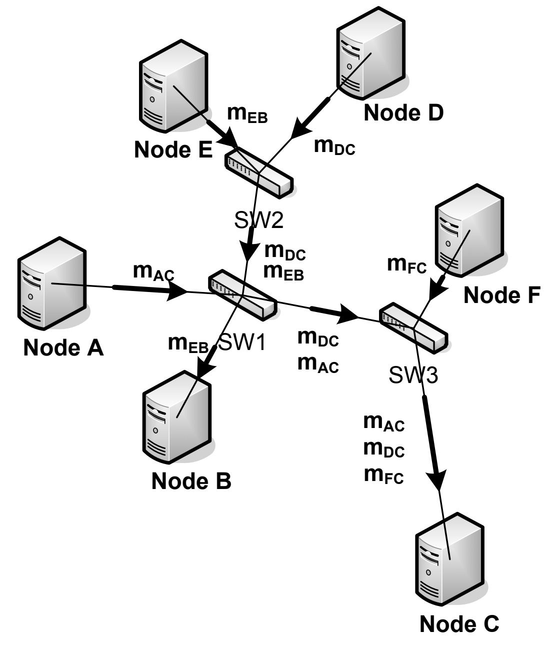

Figure 4. Messages sent over switched Ethernet

The generated allocation solution guarantees the fulfillment of real-time requirements and it takes into account the existing resources and ordering restrictions. The allocation heuristic tries to assure a uniform load of control tasks on the network nodes. In case of a node failure the distributed system is automatically reconfigured, by reallocating the tasks present in the defective node.

The paper presents a new approach in the evaluation of real-time behavior of switched Ethernet networks. The worst case transmission time computation is a mandatory step in the design and implementation of distributed control systems over Ethernet infrastructures.

The proposed solution was validated through simulations, using a TPN software tool. The simulated scenarios revealed interesting aspects concerning the influence of task granularity on the computed response time. These observations may be used for better system design.

References

[1] H. Kopetz, A. Damm, C. Koza, and M.Mullozzani, “Distributed Fault Tolerant Real-Time Systems: The MARS Approach” IEEE Micro, 9(1):25–40, 1989.

[2] M. Humphrey, G. Wallace, J.A. Stankovic, "Kernel-level threads for dynamic, hard real-time environments," rtss, p. 38, 16th IEEE Real-Time Systems Symposium (RTSS '95), 1995

[3] A. Mifdaoui, F. Frances, C. Fraboul, "Full Duplex Switched Ethernet for Next Generation "1553B"-Based Applications," rtas, pp. 45-56, 13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07), 2007

[4] J. Loeser, H. Haertig, "Low-Latency Hard Real-Time Communication over Switched Ethernet," ecrts, pp. 13-22, 16th Euromicro Conference on Real-Time Systems (ECRTS'04), 2004

[5] B.-Y. Choi, S. Song, N. Birch, J. Huang, "Probabilistic approach to switched Ethernet for real-time control applications," RTCSA , p. 384, Seventh International Conference on Real-Time Computing Systems and Applications (RTCSA'00), 2000

[6] C. Venkatramani, T. Chiueh, “Supporting real-time traffic on Ethernet”, Real-Time Systems Symposium, 1994, Proceedings Volume, pp. 282 – 286

[7] J. W.S. Liu, Real-Time Systems, Prentice Hall, 2000

[8] K. Tindell, J. Clark, “Holistic schedulability analysis of distributed hard real-time systems”. Microprocessors and Microprogramming, Elsevier. no.40, 1994

[9] Gh. Sebestyen, K. Pusztai, "Dynamic task allocation in real-time distributed control systems", 13-th International Conference on Control Systems and Computer Science CSCS2001, Bucuresti, 2001