INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
ISSN 1841-9836, 12(5), 645-660, October 2017.

Phasing of Periodic Tasks Distributed over Real-time Fieldbus

S.-H. Lee, H.-W. Jin, K. Kim, S. Lee

Sang-Hun Lee
Hyundai Mobis
17-2, 240 Mabuk-ro, Giheung-gu, Yongin-si,
Gyeounggi-do 16891, Korea
sanghun@mobis.co.kr

Hyun-Wook Jin*
Department of Computer Science and Engineering
Konkuk University
120 Neungdong-ro, Gwangjin-gu, Seoul 05029, Korea
*Corresponding author: jinh@konkuk.ac.kr

Kanghee Kim
Department of Smart Systems Software
Soongsil University
369 Sangdo-ro, Dongjak-gu, Seoul 06978, Korea
khkim@ssu.ac.kr

Sangil Lee
Agency for Defense Development
Songpa P.O.Box 132, Seoul 05661, Korea
happyjoy@add.re.kr

Abstract: In designing a distributed hard real-time system, it is important to reduce the end-to-end delay of each real-time message in order to ensure quick responses to external inputs and a high degree of synchronization among cooperating actuators. In order to provide a real-time guarantee for each message, the related literature has focused on the analysis of end-to-end delays based on worst-case task phasing. However, such analyses are too pessimistic because they do not assume a global clock. With the assumption that task phases can be managed by using a global clock provided by emerging real-time fieldbuses, such as EtherCAT, we can try to calculate the optimal task phasing that yields the *minimal* worst-case end-to-end delay. In this study, we propose a heuristic to manage the phase offsets in the distributed tasks to reduce the theoretical end-to-end delay bound. The proposed heuristic reduces the search time for a solution by identifying time intervals where actual communication occurs among inter-dependent tasks. Furthermore, to analyze the distribution of end-to-end delays in different phases, we implemented a simulation tool. The simulation results showed that the proposed heuristic can reduce worst-case end-to-end delay as well as jitter in end-to-end delays.
Keywords: task phasing, end-to-end delay, real-time, fieldbus, EtherCAT.

1 Introduction

Fieldbuses are industrial networks widely used in small-to-large-scale distributed control systems, such as vehicles and factory automation systems. While the fieldbuses define only the physical, the data link, and the application layers, they provide real-time features for deterministic communication latency.

Emerging real-time fieldbuses provide highly accurate clock synchronization as a key feature to support a global clock in distributed systems. For instance, the distributed clock of EtherCAT provides accurate clock synchronization with errors smaller than a few tens of nanoseconds [2] [3].

Such accurate clock synchronization provides a novel opportunity to adjust task phasing across distributed computing nodes in order to better synchronize dependent tasks (e.g., multi-axis motion control). However, obtaining optimal task phasing across distributed nodes has not been accorded sufficient attention in the literature, whereas worst-case task phasing in each node has been extensively studied [22] [19] [8]. This is due to coarse-grained clock synchronization on legacy networks; moreover, the performance analysis of distributed tasks by considering jitters in task execution time is not straightforward. Thus, researchers have focused on the analysis of worst-case end-to-end delays while assuming no global clock. However, once we assume a precise global clock for all nodes, we can investigate the problem of finding an optimal phase combination of tasks in order to reduce worst-case end-to-end delays. Though few efforts have been devoted in recent research to enhancing task phasing, these either assume no jitters for tasks [4] or apply phasing to a single node [11] [10].

In this paper, we describe a novel off-line algorithm that heuristically searches for near-optimal task phasing with respect to the worst-case end-to-end delay. One novelty of the proposed algorithm is that it takes into account the jitters in execution times of distributed tasks while trying to reduce search time to find near-optimal task phasing. An exhaustive search needs to consider all combinations of the variable execution times of tasks. To reduce search time, our algorithm deals with feasible time ranges of communication.

To analyze the distribution of end-to-end delays in different phase combinations, we implemented a tool that simulated variable execution times of tasks and the behavior of real-time fieldbus. In the analysis, we assumed a real-time fieldbus, such as EtherCAT, that comprised a master and multiple slave nodes. We modeled each node as a set of periodic tasks scheduled on a uniprocessor by a fixed-priority scheduling algorithm. The results of analysis showed that our algorithm can efficiently search for a phase combination that can significantly reduce both worst-case end-to-end delay and jitter.

The rest of this paper is organized as follows: In Sec. 2, we provide an overview of EtherCAT and discuss related work. In Sec. 3, we describe the system model that we assume in this study. We outline our algorithm to find a near-optimal task phase combination and describe our simulation tool for the analysis of end-to-end delays in Sec. 4 and Sec. 5, respectively. Sec. 6 contains details of a case that involves finding an optimal node phase combination for a given industrial task set. Finally, in Sec. 7, we present conclusions.

2 Background

2.1 EtherCAT

In this paper, we consider EtherCAT as the reference model for real-time fieldbuses. EtherCAT is an open specification that covers the physical, the data link and the application layers of the communication protocol stack. Owing to its low cost, compatibility with the TCP/IP protocol and efficiency, EtherCAT is becoming popular in many real-time applications such as factory automation systems and motion control systems.

An EtherCAT network consists of a single master node and multiple slave nodes allowing various network topologies such as line, daisy chain, and star. The master node generates EtherCAT telegrams, each of which is encapsulated in Ethernet frames. A telegram may include several datagrams for different slave nodes, where a datagram includes an address to specify the corresponding slave node. The master node transmits the resulting Ethernet frames through the standard Ethernet interface. Each slave node then reads from and writes to the datagram on the fly in a specially designed network controller called the EtherCAT slave controller. The EtherCAT slave controller can forward Ethernet frames to the next slave node at the hardware

level while performing such read-and-write operations. The propagation delay at the controller is less than $1\mu s$ [9]. Thus, EtherCAT provides high-speed message relay between adjacent nodes, and can guarantee deterministic communication delays.

EtherCAT provides a global clock synchronization feature called *Distributed Clock* that synchronizes the clocks of the master and slave nodes to the reference clock represented by a selected slave. The distributed clock controller can generate an interrupt called the *Sync* signal as the global reference clock [18]. The signal can operate in either cyclic or single-shot mode, where the period and shift time are specified in nanosecond resolution. Thus, we can adjust the release time (i.e., phase) of a task by means of the *Sync* signal.

2.2 Related work

Sung et al. [18] showed the potential of the global clock provided by EtherCAT for synchronized control processes, but they considered only communication tasks while assuming no preemption by higher-priority tasks. A few researchers recently suggested heuristic approaches that iteratively adjusted task phases for isochronous control on EtherCAT [11] [10]. Although they could improve actuation jitters, the suggested approaches could not be applied at design phase for a system yet to be constructed because those are based on an on-line algorithm. In addition, they targeted only task phasing on the master node by assuming a simple task set in the slave nodes. Craciunas et al. [4] tried to decide the optimal offset of tasks in terms of utility, but did not consider jitters in execution times of tasks. In this study, we propose a off-line algorithm that searches for a near-optimal task phase combination on both master and slave nodes while considering jitters in execution times of tasks.

Traditionally, task phasing was addressed in order to obtain worst-case response times of tasks. Under fixed-priority scheduling, tasks with varying inter-release times were analyzed at a worst-case time instant, called a critical instant [13], since the release time of each task was assumed not to be managed. In a distributed real-time system, such an assumption was extended to every node because the unpredictability of the release time of a task was exacerbated by the jitters introduced by varying response times of the preceding task [19] [8]. In addition, it was assumed that the communicating tasks on different nodes were synchronized by messages. There were also studies to analyze the best-case response time [1] [16], but these studies did not target distributed systems.

Several studies were conducted on the analysis and simulation of real-time industrial networks [17] [6] [7] [22]. These studies, however, did not address task phasing to obtain minimum worst-case end-to-end delays in the design of distributed real-time systems. It is important to reduce analysis time to find an optimal solution. For example, Garcia-Valls et al. [21] [20] proposed selectively checking for dynamic systems in real-time reconfiguration. In this study, we try to reduce analysis time by considering only feasible communication time intervals.

3 System model

We consider a distributed real-time system that interconnects a master node and $N-1$ slave nodes with a real-time fieldbus, which is common in many industrial applications. For example, in a motion control system, the master node generates position commands to describe motion trajectory and slave nodes serve as motor drives to respond to these commands. In the following, we explain the task and the network models, and define end-to-end delay.

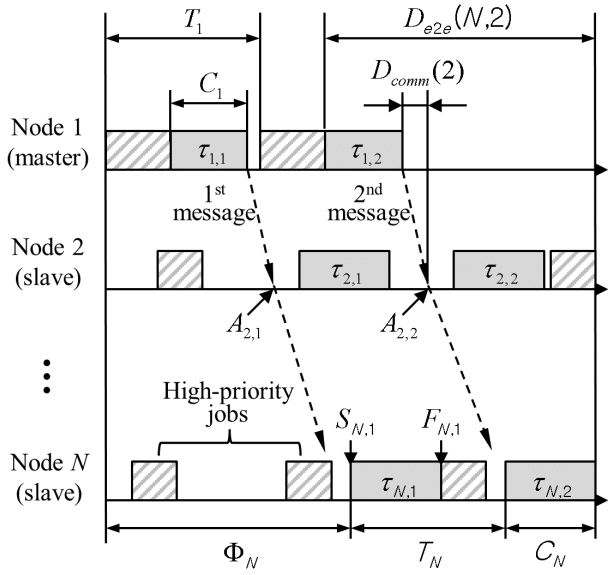

Figure 1: System model

3.1 Task model

We consider a set of periodic tasks for both master and slave nodes and these may have different sets of tasks. We denote a communication task by $\tau_n = (C_n^{min}, C_n^{max}, T_n, P_n, \Phi_n)$, where C_n^{min} and C_n^{max} are its minimum and maximum execution times, T_n is the period, P_n is its priority, and Φ_n is its phase with respect to a certain reference time. The jitter in the execution time represented by $C_n^{max} - C_n^{min}$ is caused not only by jitters in system software, but also by different code branches in tasks. We assume that the relative deadline of each task is equal to its period. Each node has only one communicating task. The rationale for this is that many industrial network protocols do not support multiplexing and de-multiplexing between tasks [5] [14]. Moreover, as the name implies, only the master node is assumed to initiate message transmission. Each slave node may only piggyback some data on message passing through the node on the fly, but cannot initiate message transmission. Thus, there exists a chain of communicating tasks across nodes (i.e., $\tau_1 \rightarrow \tau_2 \rightarrow \cdots \rightarrow \tau_n$) for each communication period as shown in Fig. 1.

Each communication task gives rise to an infinite sequence of jobs. $\tau_{n,i}$ denotes the i^{th} job of τ_n. The start time of $\tau_{n,i}$ is denoted by $S_{n,i}$ and the finish time by $F_{n,i}$. A sender job on the master node (i.e., $\tau_{1,i}$) sends a message to the network device at the finish time, $F_{1,i}$, while receiver jobs on the slave nodes (i.e., $\tau_{n,i}$, $1 < n \leq N$) look up the message from the message queue at the start time, $S_{n,i}$. We do not suppose that the receiver jobs are synchronously released by a message; rather, we assume that the jobs are released by a periodic task scheduler as many real systems do. If there is no message arrived, then the receiver job may finish without further processing. The periods of the sender and receiver tasks are identical because both operate on the same message initiated by the master node. The message arrival time of the i^{th} message on node n is represented by $A_{n,i}$. For task scheduling, we assume a fixed-priority scheduling algorithm, such as Rate Monotonic [13]. Since each task is assigned a static priority, the execution of a task assigned a lower priority can be preempted by one assigned a higher priority, which were modeled as the release jitter in several previous studies. Due to such preemption and jitters in execution times of tasks, there can be many potential time points at which $\tau_{n,i}$ starts. Thus, we define the start time of $\tau_{n,i}$ as a set of time points, i.e., $S_{n,i} = \{s_{n,i}^{min}, \cdots, s_{n,i}^{max}\}$. Similarly, the

finish time of $\tau_{n,i}$ is defined as a set $F_{n,i} = \{f_{n,i}^{min}, \cdots, f_{n,i}^{max}\}$. The one-to-many correspondence $\mathcal{R} : S_{n,i} \rightarrow F_{n,i}$ maps a start time point of $\tau_{n,i}$ to a set of potential finish time points. That is, $\mathcal{R}(s_{n,i}) = \{f_{n,i}^{1}, \cdots, f_{n,i}^{m}\} \subseteq F_{n,i}$. and $\mathcal{R}(s_{n,i}^{min}) \cup \cdots \cup \mathcal{R}(s_{n,i}^{max}) = F_{n,i}$. we also define the arrival time $A_{n,i} = \{a_{n,i}^{min}, \cdots, a_{n,i}^{max}\}$.

3.2 Network model

Emerging real-time fieldbuses, such as EtherCAT, guarantee deterministic communication delays between any two nodes through two design choices. The first involves connecting any two adjacent nodes with a dedicated link. That is, there are no interfering nodes on the link and, thus, no packet collisions. The second design choice is such that each slave node n conducts cut-through switching to relay packets between node $n-1$ and $n+1$, instead of, store-and-forward switching. This switching scheme when implemented at the hardware level eliminates the likelihood that the internal software operations of each slave contribute to end-to-end communication delay. As a result, EtherCAT guarantees deterministic communication delays from the master to any slave n, which is analyzed by Prytz et al. [15] as follows:

$$D_{comm}(n) = D_t(s) + (n-1) \times D_f + D_r(s), \tag{1}$$

where $D_{comm}(n)$ is the communication delay from the master's memory to slave n's memory as shown in Fig. 1, $D_t(s)$ is the total transmission time of a message of size s on the master network device, and D_f is the total forwarding time of a message on the slave side, which is a constant of 1 μs, irrespective of the message size. $D_r(s)$ is the total reception time of a message of size s on the slave network controller. We do not have to consider any delay in piggybacking data on the message at the slave node because this operation is conducted on the fly while the message is being forwarded by the network controller.

3.3 Problem statement

In this paper, we define end-to-end delay of the i^{th} message as the time from the beginning of the sender job at the master to the completion of the receiver job that consumes the i^{th} message at slave. Suppose $\Phi_1 + D_{comm}(n) \leq \Phi_n$, the i^{th} message is consumed by either $\tau_{n,i}$ or $\tau_{n,i+1}$ at slave n because periods of sender and receiver tasks are the same as mentioned in Sec. 3.1.

We define $D_{e2e}^{0}(n,i)$ to be the set of end-to-end delay values observed when $\tau_{n,i}$ consumes the message on slave, i.e.,
for all $f_{1,i}$ and $s_{n,i}$ that satisfy

$$f_{1,i} + D_{comm}(n) \leq s_{n,i}, \tag{2}$$

$$D_{e2e}^{0}(n,i) = \{f_{n,i} - s_{1,i} \mid f_{n,i} \in \mathcal{R}(s_{n,i}),\ s_{1,i} \in \mathcal{R}^{-1}(f_{1,i})\}, \tag{3}$$

where $\mathcal{R}^{-1}$ is the inverse correspondence of $\mathcal{R}$ defined in Sec. 3.1 (i.e., $\mathcal{R}^{-1} : F_{n,i} \rightarrow S_{n,i}$).

We also define $D_{e2e}^{+1}(n,i)$ to be the set of end-to-end delays when $\tau_{n,i+1}$ receives the message as the message arrives after the beginning of $\tau_{n,i}$, i.e.,
for all $f_{1,i}$ and $s_{n,i}$ that satisfy

$$f_{1,i} + D_{comm}(n) > s_{n,i}, \tag{4}$$

$$D_{e2e}^{+1}(n,i) = \{f_{n,i+1} - s_{1,i} \mid f_{n,i+1} \in F_{n,i+1},\ s_{1,i} \in \mathcal{R}^{-1}(f_{1,i})\}. \tag{5}$$

Thus, we define the end-to-end delay $D_{e2e}(n,i)$ of the i^{th} message as follows:

$$D_{e2e}(n,i) = D^0_{e2e}(n,i) \cup D^{+1}_{e2e}(n,i) = \{d^{min}_{n,i}, \cdots, d^{max}_{n,i}\}. \tag{6}$$

In this study, we want to analyze the worst-case end-to-end delay $\max(d^{max}_{N,1}, \cdots, d^{max}_{N,H/T_1})$, where $d^{max}_{n,i} = \max(D_{e2e}(n,i))$, and H is the hyperperiod defined as the least common multiple of the periods of all tasks. Then, we aim to focus on finding an optimal combination of task phases leading to the minimal worst-case end-to-end delay:

$$\begin{aligned} &\underset{(\Phi_1,\Phi_2,\cdots\Phi_N)}{\text{Minimize}} \ \max(d^{max}_{N,1}, \cdots, d^{max}_{N,H/T_1}) \\ &\text{subject to } U_n \leq U_{upper} \text{ for } 1 \leq n \leq N, \end{aligned} \tag{7}$$

where U_n and U_{upper} represent the processor utilization of a given task set on node n and upper-bound of the utilization which guarantees schedulability, respectively. This is challenging given that $D_{e2e}(n,i)$ has very large number of elements due to variable execution times of tasks and preemption by higher-priority tasks. In the next section, we describe the proposed algorithm that searches for a near-optimal task phase combination without calculating all elements of $D_{e2e}(n,i)$.

4 Algorithm for optimal task phasing

In this section, we propose a heuristic algorithm for optimal task phasing with respect to minimal worst-case end-to-end delay. The algorithm can efficiently consider the jitter in execution times of tasks. If we exhaust all possible combinations of the variable execution times of tasks, it requires a long time to analyze the worst-case end-to-end delay for a given task phase combination. To avoid this, prior to the actual analysis of the worst-case end-to-end delay of a phase combination, we first determine the time intervals where communication can occur and efficiently find the worst-case end-to-end delay by using this information.

4.1 Overview

Fig. 2 shows the steps of our algorithm. In order to find the optimal phases of tasks, we heuristically calculate the worst-case end-to-end delays for different phase combinations. A phase combination is denoted by $Phase_i = \{_1 + \Delta_{1,i}, _2 + \Delta_{2,i}\}$, where $_n$ is an initial sequence of task phases on node n, and $\Delta_{n,i}$ is the i^{th} sequence of variations to be added to $_n$. If we analyze end-to-end delays for all possible phase combinations, we need to examine $H^{\Sigma t}$ cases, where H is the hyperperiod, and Σt is the number of all tasks in the system. To reduce the number of search cases, in the first step, we assume that the slave nodes have the same set of tasks, which is common in many industrial control systems [12] [11]. We do this in order to be able to consider only the master and the first slave nodes, while simply calculating the phase combinations for the other slave nodes n $(2 < n \leq N)$ by adding $(n-2) \times D_f + D_r(s)$ to the best phase combination for the first slave node. Moreover, we only consider phases that satisfy $\Phi_1 + D_{comm}(2) \leq \Phi_2$, and do not specifically consider the phases for tasks with lower priority than the communication task. We also deal with phases with identical relative phase combinations, such as $\{_1, _2\}$ and $\{_1 + \Delta\prime, _2 + \Delta\prime\}$, as the same phase combinations; thus, we can omit the latter. The granularity of the task phase is also configurable to reduce analysis time, which is discussed in Sec. 6.3.

More importantly, we need to deal with the variable execution times of tasks. If we consider every case of variable execution times of all tasks, we need to examine $\Big(\big((C^{max}_1 - C^{min}_1) \times H/T_1 \times (C^{max}_{m1} - C^{min}_{m1}) \times H/T_{m1} \times \cdots \times (C^{max}_{mj} - C^{min}_{mj}) \times H/T_{mj}\big) \times \big((C^{max}_2 - C^{min}_2) \times H/T_2 \times (C^{max}_{s1} - C^{min}_{s1}) \times$

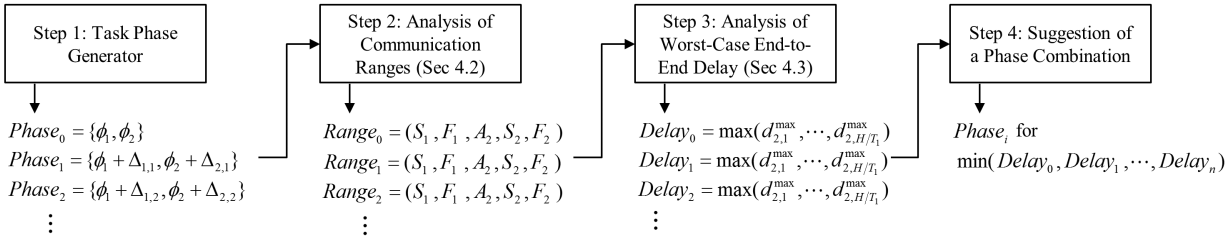

Figure 2: Steps to find optimal phase combination

$H/T_{s1} \times \cdots \times (C_{sk}^{max} - C_{sk}^{min}) \times H/T_{sk})\Big)$ cases for each phase combination, where j and k are the number of non-communicating tasks that have a higher priority than τ_1 and τ_2 on the master and the slave nodes, respectively. We cannot simply reduce the number of these cases by assuming a larger resolution of execution times, because we can omit the case that generates the worst-case end-to-end delay with a large resolution. In order to reduce analysis time while considering variable execution times, the second step of our algorithm analyzes the communication range, $Range_i$, for each $Phase_i$ by only using the minimum and maximum execution times of tasks. A communication range represents a time interval where actual communication occurs between τ_1 and τ_2. We describe the second step in finding communication ranges in Sec. 4.2.

In the third step, we efficiently calculate the maximum end-to-end delay $d_{2,i}^{max}$ for every message for a given phase. In Sec. 4.3, we detail the third step to analyze the maximum end-to-end delay for a given communication range. Once we determine the maximum end-to-end delay for each phase combination, we choose the phase combination that provides the minimum worst-case end-to-end delay in the last step.

4.2 Analysis of communication ranges

As we have mentioned in Sec. 4.1, in order to analyze the worst-case end-to-end delay for a given phase combination, we deal with the ranges of communication time points instead of considering all possible combinations of the execution times of tasks. We find the communication ranges by using the minimum and maximum execution times of tasks. First, we calculate the start and finish time ranges for all jobs of τ_1 on the master node, which are represented by $S_1 = \{S'_{1,1}, \cdots, S'_{1,H/T_1}\}$ and $F_1 = \{F'_{1,1}, \cdots, F'_{1,H/T_1}\}$, respectively, where $S'_{n,i} = \{s_{n,i}^{min}, s_{n,i}^{max}\}$ and $F'_{n,i} = \{f_{n,i}^{min}, f_{n,i}^{max}\}$. It should be noted that $S'_{n,i}$ and $F'_{n,i}$ have only minimum and maximum values, whereas $S_{n,i}$ and $F_{n,i}$ defined in Sec. 3.1 include all possible time points. The time range information F_1 is then translated into the range information of message arrival time on the first slave node, which is denoted by $A_2 = \{A'_{2,1}, \cdots, A'_{2,H/T_1}\}$, where $A'_{2,i} = \{a_{2,i}^{min}, a_{2,i}^{max}\}$. The start and finish time ranges on the first slave node represented by $S_2 = \{S'_{2,1}, \cdots, S'_{2,H/T_2}\}$ and $F_2 = \{F'_{2,1}, \cdots, F'_{2,H/T_2}\}$ are calculated as well.

For example, Fig. 3 shows the communication range for the first message. For each node in the figure, the upper row represents time flow with the maximum execution times of tasks while the lower one shows time flow with the minimum execution times. In the master node of this example, τ_1 is preempted by a higher-priority task as shown in the upper row, which results in a larger $s_{1,1}^{max}$ than $s_{1,1}^{min}$. The difference between $f_{1,1}^{min}$ and $f_{1,1}^{max}$ becomes even greater due to the jitter in execution time (i.e., $C_1^{max} - C_1^{min}$). We can calculate $a_{2,1}^{min}$ and $a_{2,1}^{max}$ by adding $D_{comm}(2)$ to $f_{1,1}^{min}$ and $f_{1,1}^{max}$, respectively. In the slave node, two non-communicating tasks have a higher priority than τ_2. Although the figure shows that $\tau_{2,1}$ is not preempted by these higher-priority tasks in the case of minimum execution time (i.e., the lower row), $\tau_{2,1}$ is delayed by a higher-

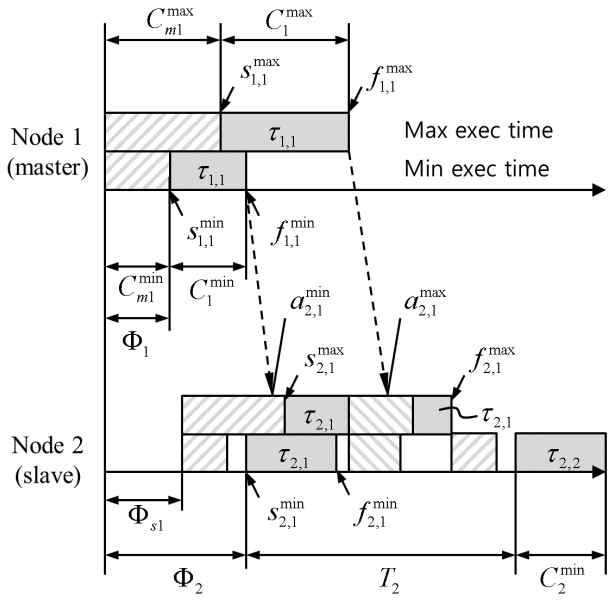

Figure 3: Analysis of communication ranges

priority task at the start point and preempted by the other one during execution in the case of maximum execution time. Consequently, for the given task set and phase combination, the range of message arrival, $[a_{2,1}^{min}, a_{2,1}^{max}]$, and that of the start time of $\tau_{2,1}$, $[s_{2,1}^{min}, s_{2,1}^{max}]$, overlap. By providing such information for the third step, we can efficiently analyze the worst-case end-to-end delay $d_{2,1}^{max}$.

4.3 Analysis of worst-case end-to-end delay

By using the feasible time ranges of communication provided by the second step, we analyze the worst-case end-to-end delay for a given task phase. As we have mentioned in Sec. 3.3, to analyze the worst-case end-to-end delay we need to calculate $D_{e2e}(2,i)$ for $1 \leq i \leq H/T_1$ and find $\max(d_{2,1}^{max}, \cdots, d_{2,H/T_1}^{max})$. Since $D_{e2e}(2,i) = D_{e2e}^{0}(2,i) \cup D_{e2e}^{+1}(2,i)$ by Eq. 6, we describe how to calculate $\max(D_{e2e}^{0}(2,i))$ and $\max(D_{e2e}^{+1}(2,i))$ in this section, respectively.

As defined by Eq. 2 and Eq. 3 in Sec. 3.3, $D_{e2e}^{0}(2,i)$ is a set of end-to-end delays calculated supposing that the i^{th} receiver job $\tau_{2,i}$ consumes the i^{th} message. However, since we have only range information represented by minimum and maximum values, we modify Eq. 2 as follows:

$$f_{1,i}^{min} + D_{comm}(2) = a_{2,i}^{min} \leq s_{2,i}^{max}, \tag{8}$$

which is satisfied if there is a possibility that the i^{th} receiver job receives the i^{th} message. In this case, we can calculate the maximum end-to-end delay of the i^{th} message as follows:

$$\begin{aligned} \max\left(D_{e2e}^{0}(2,i)\right) &= f_{2,i}^{max} - \min\left(\mathcal{R}^{-1}\left(a_{2,i}^{min} - D_{comm}(2)\right)\right) \\ &= f_{2,i}^{max} - \min\left(\mathcal{R}^{-1}\left(f_{1,i}^{min}\right)\right) \\ &= f_{2,i}^{max} - s_{1,i}^{min}, \end{aligned} \tag{9}$$

which is derived from Eq. 3. In order to calculate the end-to-end delay, we need to know the start time of the sender job (i.e., $s_{1,i}$) and the finish time of the receiver job (i.e., $f_{2,i}$). It is obvious that the end-to-end delay becomes maximum if the sender sends the message in earliest

time (i.e., starts at $s_{1,i}^{min}$) while the receiver is delayed for as long as possible (i.e., finishes at $f_{2,i}^{max}$). In Eq. 9, the start time of the sender job is reversely calculated from the message arrival time $a_{2,i}^{min}$.

On the other hand, if there is a chance that the message arrives after the beginning of the i^{th} receiver job, the message happens to be handled by the next (i.e., $(i+1)^{th}$) job. This occurs when the following equation derived from Eq. 4 is satisfied:

$$f_{1,i}^{max} + D_{comm}(2) = a_{2,i}^{max} > s_{2,i}^{min}. \tag{10}$$

Then the equation for finding the maximum end-to-end delay of the i^{th} message can be derived from Eq. 5 as follows:

$$\max\left(D_{e2e}^{+1}(2,i)\right) = f_{2,i+1}^{max} - \min\left(\mathcal{R}^{-1}\left(a_{2,i}^{*} - D_{comm}(2)\right)\right). \tag{11}$$

Here $a_{2,i}^{*}$ is the earliest arrival time of the message processed by $\tau_{2,i+1}$. That is,

$$a_{n,i}^{*} = \max(s_{n,i}^{min} + \epsilon,\, a_{n,i}^{min}), \tag{12}$$

where ϵ represents a very small value. If the message is arrived at $s_{2,i}^{min} + \epsilon$ while the receiver job starts at $s_{2,i}^{min}$, the message is processed by the next receiver job. However, if $s_{n,i}^{min} + \epsilon < a_{n,i}^{min}$, then $a_{n,i}^{min}$ is the earliest time point. It should be noted that $\min(\mathcal{R}^{-1}(a_{2,i}^{*} - D_{comm}(2)))$ in Eq. 11 can be neither $s_{1,i}^{min}$ nor $s_{1,i}^{max}$. However, we have only minimum and maximum values that present range information to reduce analysis time. Thus, it is difficult to find the exact $\min(\mathcal{R}^{-1}(f_{1,i}))$ for arbitrary $f_{1,i}$ (i.e., $a_{2,i}^{*} - D_{comm}(2)$), unless we analyze every preemption point of the sender job by considering variable execution times. Although we could calculate a converged value by using a recurrence equation for each message, to reduce the time needed for analysis, we pessimistically assume that the worst-case preemption always occurs and estimate $\min(\mathcal{R}^{-1}(f_{n,i}))$ as follows:

$$\min(\mathcal{R}^{-1}(f_{n,i})) = \begin{cases} f_{n,i} - C_{n,i}^{max} - Pr^{max}, & \text{if } f_{n,i} - C_{n,i}^{max} - Pr_{n}^{max} > s_{n,i}^{min}, \\ s_{n,i}^{min}, & \text{otherwise,} \end{cases} \tag{13}$$

where Pr^{max} is the worst-case preemption time of $\tau_{n,i}$. As we calculate $s_{n,i}^{max}$ and $f_{n,i}^{max}$ assuming the maximum execution time for both communicating task and non-communicating higher-priority tasks, the time range $[s_{n,i}^{max},\, f_{n,i}^{max}]$ includes the worst-case preemption time. Hence, Pr^{max} can be expressed as:

$$Pr^{max} = f_{n,i}^{max} - s_{n,i}^{max} - C_{n,i}^{max}. \tag{14}$$

By using Eq. 14, we restate Eq. 13 as follows:

$$\min(\mathcal{R}^{-1}(f_{n,i})) = \begin{cases} f_{n,i} - f_{n,i}^{max} + s_{n,i}^{max}, & \text{if } f_{n,i} - f_{n,i}^{max} + s_{n,i}^{max} > s_{n,i}^{min}, \\ s_{n,i}^{min}, & \text{otherwise,} \end{cases} \tag{15}$$

Therefore, by Eq. 9 and Eq. 11, we can find the worst-case end-to-end delay of i^{th} message as follows:

$$\begin{aligned} d_{2,i}^{max} &= \max\left(D_{e2e}(2,i)\right) \\ &= \max\left(\left\{\max\left(D_{e2e}^{0}(2,i)\right), \max\left(D_{e2e}^{+1}(2,i)\right)\right\}\right). \end{aligned} \tag{16}$$

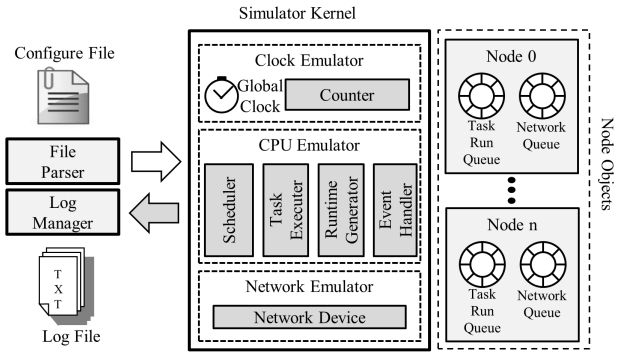

Figure 4: Simulation tool

5 Simulation tool

In real systems, end-to-end delays vary due to jitters in the execution times of tasks. Thus, it is not feasible to evaluate the distribution and the average of end-to-end delays by using a simple mathematical equation. Instead, we implemented a simulation tool consisting of four components: configuration manager, node objects, simulation kernel, and log manager. Fig. 4 shows the overall design of the proposed simulator. The analysis tool is based on discrete-event simulation to simulate task scheduling, message transmission/forwarding, DMA, and I/O event handling.

Configuration parameters such as the number of nodes, task sets, network bandwidth, message size, and event handling mode are defined in the configuration file in XML format. Thus, a user of the tool can easily change the attributes of the system to be analyzed. Furthermore, newly defined attributes can be described simply by extending the XML schema. The configuration module reads the configuration file at the initialization phase and sets the environment variables of the simulation kernel with values extracted from the configuration file. The configuration module also creates node objects as much as described in the configuration file and sets attributes of these objects, such as event handling mode.

The node objects contains a snapshot of a run-time image of each node, such as the state information of the run queue and the message queue. The run queue stores task control blocks, which saves a given task's mission as well as the amount of time for which it executes for the given period. The mission is classified into three categories: send, receive, and other. The message queue stores network messages, which are yet to be sent to the network or yet to be consumed by a receiver task.

The simulation kernel performs the analysis loops. The simulation kernel consists of a clock emulator, a CPU emulator, and a network emulator. The clock emulator simulates a synchronized global clock and increases its counter for every simulation loop.

The CPU emulator run task(s) on each node for every time unit. The scheduler selects a task to run based on the fixed-priority scheduling algorithm. Our current implementation only supports the RM scheduling algorithm but our design is general enough to add other scheduling algorithms. The task executer emulates the behavior of the selected task based on its mission information. If the mission is *send*, the task executer posts a send request to the network device at the finish time of the job. On the contrary, in the *receive* case, the task executer consumes a message from the message queue at the start time of the job. The task executer simply consumes the time for tasks dedicated to *other* missions. The runtime generator generates the execution time of a job according to C_n^{min} and C_n^{max} for every task. In the current implementation, we

Table 1: Task set on slave nodes

Task Name	T_i and C_i	Description
MotorAct	250μs, 25∼35μs	Controls a motor (highest priority)
RtMsg	250μs, 10∼15μs	Receives real-time messages from the network, and shares it with the MotorAct task
NrtMsg	250μs, 7∼10μs	Handles non-real-time messages (lower priority than RtMsg)
HealthMon	500μs, 6∼9μs	Performs health monitoring (lowest priority)

Table 2: Simulation parameters

Parameters	Value	Parameters	Value
Network bandwidth	100 Mbps	Network forwarding delay (D_f)	1 μs
Event handling mode	Interrupt	Interrupt handling time	5 μs
Packet size	50 bytes	DMA overhead for a 50byte packet	1 μs

support uniform distribution and normal distribution for the type of execution time distribution but other distribution type can also be added.

The network emulator simulates operations of the network device (e.g., message transmission, DMA, and raising an interrupt). Event handling can be carried out in either polling or interrupt mode. If the network device is configured as a polling-based device, the event is handled when a receiver task is released. Thus, in this mode, the network emulator silently inserts a received message into the receive queue without any notification when it arrives. When the network device is configured as an interrupt-based device, the network emulator sends an interrupt signal to the CPU emulator so that the event handler is invoked as soon as a message arrives preempting a running task. The network emulator also decides the message transmission speed based on the bandwidth information, and maintains the remaining bytes of a message being sent on each object node.

The log manager stores the results of analysis in output files. The final output represents the total running time of the tool, the end-to-end delay for each message, and the average end-to-end delay. The log manager also saves intermediate information generated during analysis.

6 Industrial case study

In this section, we detail an industrial case to show that our algorithm can efficiently propose enhanced task phasing across distributed nodes for a given task set.

6.1 Analysis environments

Table 1 shows the task set we ran on the slave nodes. We used the task set defined in Kim et al. [12] for the motor drive. Each slave node scheduled these tasks using the fixed-priority scheduling algorithm. We assume that the execution times of tasks followed a uniform distribution, which was not necessary for our analysis. Table 2 shows the configuration parameters for the simulation tool. We borrowed such network parameter values as queue size, bandwidth, and forwarding delay from EtherCAT specifications.

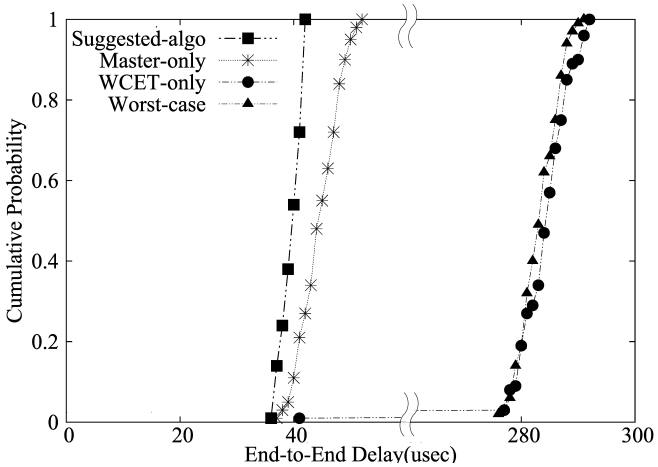

Figure 5: Distribution of end-to-end delays of different phasing schemes

6.2 Comparisons with other phasing schemes

In this section, we compare end-to-end delays of four phase combinations. One of these phase combinations was generated by our algorithm in Sec. 4 (*Suggested-algo*). Another phase combination was limited to the master node by assuming the critical instant on the slave (*Master-only*). The third was generated based on the worst-case execution times (WCET) of tasks without considering the jitters in execution time (*WCET-only*). The other phase combination represented worst-case phasing with respect to end-to-end delay (*Worst-case*). By utilizing the simulator described in Sec. 5, we measured the end-to-end delays, and present the cumulative probability of these and the minimum, average, and maximum end-to-end delays.

The distribution of end-to-end delays of four phase combinations are compared in Fig. 5. It can be seen that the end-to-end delays of *Suggested-algo* were distributed among the lowest values, whereas those of *Worst-case* yielded the highest values. Although *WCET-only* adjusted the phases of tasks, it was largely unable to enhance end-to-end delay in comparison with *Worst-case*, because *WCET-only* did not consider variable execution times of tasks. It is also evident that *Master-only* significantly reduced end-to-end delays, but still generated higher values than *Suggested-algo* due to the absence of phasing on the slave node.

Fig. 6 compares the minimum, average and maximum end-to-end delays of the four phase combinations. Compared with *Master-only* and *WCET-only*, *Suggested-algo* yielded end-to-end delays that were shorter by 19% and 86%, respectively. It also reduced jitter by 67% and 98%, respectively. In the case of *WCET-only*, the minimum end-to-end delay was significantly lower than its average and maximum values. This was because *WCET-only* shows the lowest end-to-end delay when tasks consume their WCET, although its probability is very low, as shown in Fig. 5.

6.3 Impact of phase granularity

As mentioned in Sec. 4.1, we can change the granularity of the task phase to reduce the number of phase combinations to be considered. A higher granularity can reduce search time but harms optimality. To analyze the impact of granularity on search time and optimality, we ran our algorithm by varying granularity and analyzed the worst-case end-to-end delays of the suggested phase combinations.

Fig. 7 shows that the number of search cases can be significantly reduced with a higher phasing granularity. However, Fig. 8 shows that the worst-case end-to-end delays increased as

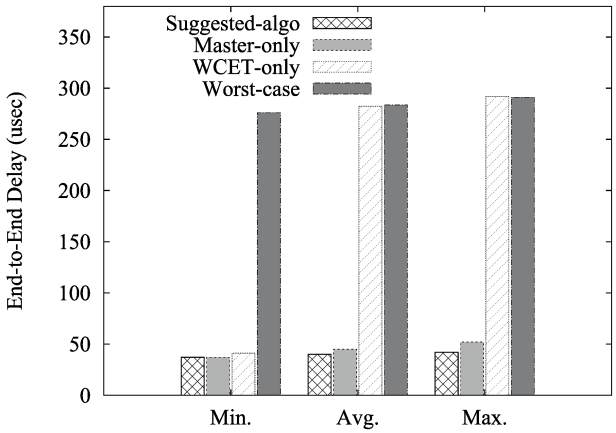

Figure 6: Minimum, average, and maximum end-to-end delays of different phasing schemes

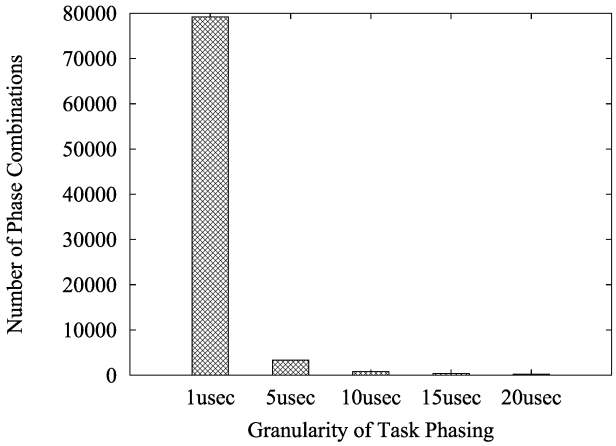

Figure 7: Number of phase combinations with different phase granularities

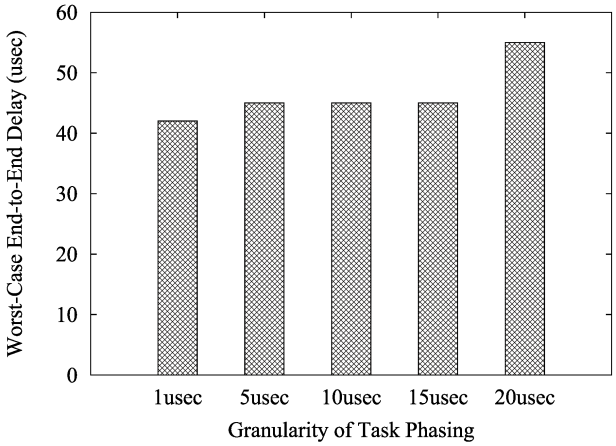

Figure 8: Worst-case end-to-end delay with different phase granularities

granularity increased, which means that the chances of finding an optimal phase combination also decreased with a higher granularity.

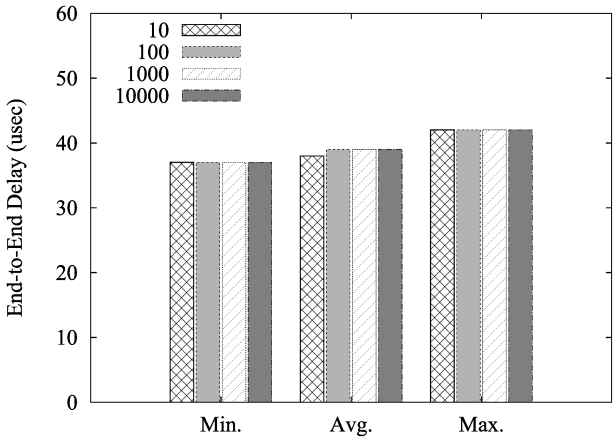

Figure 9: Impact of the number of simulated periods on the analysis accuracy

6.4 Accuracy of delay distribution

To analyze the distribution of end-to-end delays, our simulator varied the execution times of tasks for every period during the simulation. Therefore, the simulator could provide an accurate distribution when a sufficient number of periods had been simulated. Thus, the larger the number of periods simulated, the more accurate the distribution of end-to-end delays. In order to analyze the impact of the number of simulated periods on the accuracy of performance measurement, we ran our simulator by varying the number of simulated communication periods from 10 to 10,000.

Fig. 9 shows the minimum, average and maximum end-to-end delays analyzed with different numbers of periods. We can see that the average end-to-end delay of the 10-periods case was slightly lower than in other cases, while the minimum and maximum end-to-end delays were the same for all cases. In general, if variations in tasks' execution time and the number of tasks are large, a small number of simulated periods may produce a less accurate distribution of end-to-end delays. However, we would not need an excessively large number of periods, as shown in our results.

7 Conclusions

In this study, we proposed a heuristic algorithm that searched for a near-optimal task phase combination on distributed nodes with respect to the minimal worst-case end-to-end delay for emerging real-time fieldbuses, such as EtherCAT. To reduce search time, the algorithm identifies time intervals where communication occurs and efficiently calculates the worst-case end-to-end delay for a given task phase combination without considering all possible execution times of tasks and their combinations. To analyze the distribution of end-to-end delays of different phasing approaches, we also implemented a simulation tool that considered variable execution times of tasks and the behavior of a real-time fieldbus.

We carried out an industrial case study to show that the proposed heuristic can efficiently suggest near-optimal task phasing across distributed nodes for minimum worst-case end-to-end delays. The simulation results clearly showed that our algorithm can reduce end-to-end delay and its jitter by 86% and 98%, respectively, compared with other task phasing approaches. In summary, we need to consider variable execution times of tasks for optimal phasing of distributed tasks, but also need to apply task phasing to both master and slave nodes to achieve minimum worst-case end-to-end delay.

Acknowledgment

This work was supported by the Dual Use Technology Program (UM13018RD1).

Bibliography

[1] Bril R.J., Steffens E.F.M., Verhaegh W.F.J. (2004); Best-case response times and jitter analysis of real-time tasks, *Journal of Scheduling*, 7(2), 133-147, 2004.

[2] Cena G., Bertolotti I.C., Scanzio S., Valenzano A., Zunino C. (2012); Evaluation of EtherCAT distributed clock performance, *IEEE Trans. Industrial Informatics*, 8(1), 20-29, 2012.

[3] Cena G., Bertolotti I.C., Scanzio S., Valenzano A., Zunino C. (2010); On the accuracy of the distributed clock mechanism in EtherCAT, In *Proc. the 8th IEEE Int. Workshop on Factory Communication Systems*, 43-52, 2010.

[4] Craciunas S.S., Oliver R.S., Ecker V. (2014); Optimal static scheduling of real-time tasks on distributed time-triggered networked systems, *Proc. IEEE Int. Conf. on Emerging Technologies and Factory Automation*, 1-8, 2014.

[5] Farsi M., Ratcliff K., Barbosa M. (1999); An introduction to CANopen, *Computing & Control Engineering Journal*, 10(4), 161-168, 1999.

[6] Garner G.M., Gelter A., Teener M.J. (2009); New simulation and test results for IEEE 802.1 as timing performance, *Int. Symp. on Precision Clock Synchronization for Measurement, Control and Communication*, 1-7, 2009.

[7] Harbour M.G., Gutiérrez J.J., Drake J.M., Martínez P.L., Palencia J.C. (2013); Modeling distributed real-time systems with MAST2, *Journal of Systems Architecture*, 59(6), 331-340, 2013.

[8] Henia R., Racu R., Ernst R. (2007); Improved Output Jitter Calculation for Compositional Performance Analysis of Distributed Systems, *Proc. IEEE Int. Parallel and Distributed Processing Symp.*, 1-8, 2007. doi: 10.1109/IPDPS.2007.370356

[9] International Electrotechnical Commission (2007); Industrial Communication Networks Fieldbus specifications - Part 3-12: Data-Link Layer Service Definition - Part 4-12: Data-link layer protocol specification-Type 12 elements, *IEC*, 61158-3/4-12 (Ed. 1.0), 2007.

[10] Kang H., Kim K., Jin H.-W. (2016); Real-Time Software Pipelining for Multidomain Motion Controllers, *IEEE Trans. Industrial Informatics*, 12(2), 705-715, 2016.

[11] Kim I.; Kim T. (2015); Guaranteeing isochronous control of networked motion control systems using phase offset adjustment, *Sensors*, 15(6), 13945-13965, 2015.

[12] Kim K., Sung M., Jin H.-W. (2012); Design and implementation of a delay-guaranteed motor drive for precision motion control, *IEEE Trans. Industrial Informatics*, 8(2), 351-365, 2012.

[13] Liu C., Layland J. (1973); Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of the ACM*, 20(1), 46-61, 1973.

[14] Potra S., Sebestyen G. (2006); EtherCAT protocol implementation issues on an embedded linux platform, *Proc. IEEE Int. Conf. on Automation, Quality and Testing, Robotics*, 420-425, 2006.

[15] Prytz G. (2008); A performance analysis of EtherCAT and PROFINET IRT, *Proc. IEEE Int. Conf. on Emerging Technologies and Factory Automation*, 408-415, 2008.

[16] Redell O., Sanfridson M. (2002); Exact Best-Case Response Time Analysis of Fixed Priority Scheduled Tasks, *Proc. Euromicro Conf. on Real-Time Systems*, 165-172, 2002.

[17] Steinbach T., Kenfack H.D., Korf F., Schmidt T.C. (2011); An extension of the OMNeT++ INET framework for simulating real-time Ethernet with high accuracy, *Proc. the 4th Int. ICST Conf. on Simulation Tools and Techniques*, 375-382, 2011.

[18] Sung M., Kim I., Kim T. (2013); Toward a Holistic Delay Analysis of EtherCAT Synchronized Control Processes, *International Journal of Computers Communications & Control*, 8(4), 608-621, 2013.

[19] Tindell K., Clark J. (1994); Holistic schedulability analysis for distributed hard real-time systems, *Microprocessing and Microprogramming*, 40(2), 117-134, 1994.

[20] Valls M.G., Alonso A., de la Puente J.A. (2012); A dual-band priority assignment algorithm for dynamic QoS resource management, *Future Generation Computer Systems*, 28(6), 902-912, 2012.

[21] Valls M.G., López I.R., Villar L.F. (2013); iLAND: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems, *IEEE Trans. Industrial Informatics*, 9(1), 228-236, 2013.

[22] Zeng H., Di Natale M., Giusto P., Sangiovanni-Vincentelli A. (2009); Stochastic analysis of CAN-based real-time automotive systems, *IEEE Trans. Industrial Informatics*, 5(4), 388-401, 2009.