IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 21, 2021, accepted February 14, 2021, date of publication February 22, 2021, date of current version March 3, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3060856*

# Design and Implementation of an Ethernet-Based Linear Motor Drive for Industrial Transport Systems

**SEUNG-YONG LEE[1] AND MINYOUNG SUNG[2], (Member, IEEE)**
[1]Korea Electronics Technology Institute, Seongnam-si 13509, South Korea
[2]Department of Mechanical and Information Engineering, University of Seoul, Seoul 02504, South Korea

Corresponding author: Minyoung Sung (mysung@uos.ac.kr)

This work was supported by the 2019 Research Fund of the University of Seoul.

**ABSTRACT** Ethernet-based motor drives are hard real-time control systems used to operate servomotors through the industrial Ethernet. Recently, Ethernet-based drives have drawn attention as a solution for industrial transport systems where numerous linear motor drives move magnetic shuttles individually or collectively to accurately position production parts. This paper presents the design and implementation of an Ethernet-based motor drive that enables delay analysis for synchronized motor actuation and sensing to build a scalable and precise industrial transport system. Our software design constructs the drive function using periodic tasks run by a rate-monotonic real-time scheduler and performs worst-case response analysis to determine the end-to-end delay required for the control host to actuate or sense the motor in the drive. Based on the calculated drive delays, clock-based I/O using Ethernet-provided global time realizes synchronized motor operation across multiple drives. In the Ethernet-networked control system, different phases of the host cycle with respect to the drive cycle can result in different actuation and sensing delays. To reduce the delay, we propose a phase-shifted loop method and present a heuristic to find the best phase that minimizes the normalized drive delays. Experimental results obtained using a prototype EtherCAT drive show that the phase-shifted loop significantly reduces the difference between the commanded and feedback currents while properly managing tracking errors. Performance evaluations are performed to investigate the impact of different Ethernet technologies on delays. Elaborated delay models are developed for EtherCAT and Ethernet Powerlink, and a comparative study of delay performance is conducted for various parameters, such as the number of drives, the message size, the network topology, and the bandwidth.

**INDEX TERMS** Linear motor drives, industrial Ethernet, actuation and sensing delay, phase-shifted loop, industrial transport systems.

## I. INTRODUCTION

Ethernet-based motor drives are hard real-time embedded control systems used to actuate and monitor one or several servomotors through industrial Ethernet [1]–[7]. Owing to the deterministic communication delay and clock synchronization of industrial Ethernet [4], [8]–[12], they are widely used in robotics and manufacturing [6], [13]–[15]. Among the numerous industrial Ethernet solutions, EtherCAT and Ethernet Powerlink (EPL) are gaining ground in motion control systems. By requiring modification at the Ethernet data-link layer, they ensure deterministic communication delays. In EtherCAT, a special host integrates multiple messages into a single Ethernet frame and circulates the frames in a daisy-chained network. Frames are relayed by a hardware switch at each device, so that the delivery time of the message is almost deterministic. In EPL, deterministic communication is realized by sharing the network using a time-division multiple access method and, as in EtherCAT, all communication is controlled by a management host. Precise clock synchronization is another important feature of industrial Ethernet and enables highly synchronized operations between distributed devices. The synchronized clock of EtherCAT, known as Distributed Clock (DC), is very accurate and in many cases

The associate editor coordinating the review of this manuscript and approving it for publication was Najah Abuali.

has a deviation of only tens of nanoseconds. EPL uses the IEEE 1588 Precision Time Protocol (PTP) for clock synchronization and can easily provide accuracy in the sub-microsecond range.

With the advent of smart factories, Ethernet-based drives are drawing attention as a viable solution for industrial transport systems where multiple linear motor drives move magnetic shuttles individually or collectively to accurately place production parts [16]–[18]. The industrial transport system consists of a motion control host and several Ethernet-connected motor drives. The control host maintains a centralized position loop for each shuttle and periodically sends speed or torque commands to the drive to achieve coordinated shuttle motion. Given the target velocity or torque input, each drive computes a local loop and applies the appropriate current to the associated coil to produce the desired torque. With industrial Ethernet, deterministic communication allows the calculation of the time required by the host to deliver a message to each drive, and clock synchronization allows the drives to operate simultaneously.

However, it is challenging to design a motor drive for such linear-motor transport systems [3], [19]–[23]. First, in order to achieve position repeatability in the micrometer range and the shuttle speed in meters per second, the delay required for the control host to operate the motor drive must be minimized. Second, multiple drives must correctly and simultaneously control one or more shuttles while avoiding collisions between shuttles. For example, in an automotive glass transportation system, multiple shuttles may need to cooperate to move a single production part. This means that all motor actuations in different drives must be synchronized with the host command and must be performed when all drives complete the computation. Finally, the scalability of the transport system and the diversity of industrial Ethernet increase the complexity of the drive design. Tens or even hundreds of drives might be required for an industrial transport system, where the arrival time of the host command on each drive affects the delay in drive operation and depends on the Ethernet technology used, as well as various parameters, such as the number of drives, the message size, and topological location of the drive. Therefore, building a scalable Ethernet-based transport system that supports high-speed precision motion requires a systematic approach to drive software design. It must minimize the drive delay while ensuring synchronized motor actuation and sensing of all drives. However, there have been few such studies addressing how to analyze drive delays in Ethernet-based motion systems and fully covering how to design software for synchronized motion with minimized delay.

In this paper, we address the above design problem by developing an Ethernet-based linear motor drive with a sophisticated software design and Ethernet delay model, which can be used as a building block for building a scalable and highly responsive industrial transport system. Aimed at a software architecture that enables delay analysis, our implementation constructs the drive function with several periodic tasks run by a rate-monotonic fixed priority scheduler and realizes synchronized motor actuation and sensing via clock-based I/O. Based on the multitasked drive model, our design performs worst-case response analysis to determine the delay required for the control host to actuate or sense the motor in the drive, and attempts to minimize the delay using our phase-shifted loop method. In Ethernet-based transport systems, the task that handles real-time messages in the drive is activated when an Ethernet frame arrives, and the proposed drive model reflects this using a task offset. The offset of the real-time message task is calculated from the message delivery time provided by the Ethernet delay model, which can be developed separately, depending on the Ethernet mechanism of interest.

The drive delay covered in this study is expressed in terms of actuation delay and sensing delay. Actuation delay refers to the time interval from the start of a host cycle to motor actuation in the drive; sensing delay is the interval from the motor status read of the drive to the start of the next cycle. Given the Ethernet message delivery time as input, the drive-local delay required for motor actuation and sensing can be obtained through worst-case response analysis using the drive task set. In a clock-synchronized networked control system, different phases of the host cycle with respect to the drive cycle can result in different actuation and sensing delays. The phase-shifted loop method aims to reduce the delay based on this. Our heuristic tries to find the phase that minimizes the drive delay while ensuring synchronized motor actuation and sensing across all drives. The algorithm iteratively computes the delay for each drive to find the best host phase and, finally, returns the determined phase along with the task offsets for the clock-based input and output that produce the smallest actuation and sensing delays.

The proposed phase search heuristic requires an Ethernet delay model to determine the time it takes to deliver a message from the host to the drive, and it is important to know that the Ethernet technology and the related parameters greatly affect the message delivery time and thus the drive delay. In this paper, we present accurate delay models for EtherCAT and EPL by considering the actual frame delivery mechanism. The delay in EtherCAT can be determined relatively easily using the times taken for the transmission of summation frames and the hardware-based frame relay at the drive. In contrast, EPL has a complex model designed to take into account the poll-based mechanism for exclusive network access in different topologies (line and star). It is noteworthy that the model includes the latest EPL extension, the poll-response chaining (PRC) mechanism, which has been shown in our experiments to significantly improve the drive delay compared with the standard poll mode.

An EtherCAT motor drive that operates using a cyclic torque command has been implemented to verify the transport capability and to show improved performance. The experimental results demonstrate that the phase-shifted loop method reduces the difference between the commanded and feedback currents, while properly managing tracking errors.

In addition, with a comparative analysis of the drive delays in EtherCAT and EPL, we discuss the impact of Ethernet mechanisms on performance, and investigate the minimum achievable delay, depending on various factors, such as network topologies, message sizes, and network bandwidths.

Note that the scope of this paper covers only how to design the drive software, not the motion control host, and how to analyze the end-to-end delay relative to the host cycle for motor actuation and sensing. To this end, we design multitasked software that implements the drive function, and develop a model that analyzes the response time of the drive task and the message delay of industrial Ethernet. The challenge here is how to minimize the drive delay while ensuring that all motor actuation and sensing on different drives are synchronized with the control host. In this problem, shifting the phase of the host cycle relative to the drive cycle can reduce or increase the drive delay, and it is important to find the best phase that provides the smallest possible delay. We solve this problem with a phase search heuristic based on Ethernet and task delay models. It is worth noting that our phase search heuristic and delay analysis are general enough to cover other linear-motor transport systems using different industrial Ethernet technologies and different sets of drive tasks.

This paper is organized as follows. In Section II, we describe the background of industrial Ethernet and clock synchronization. In Section III, we explain the structure of the motor drive software and task design details. Section IV presents the experimental results with both measurement and analytical data, and Section V concludes the paper.

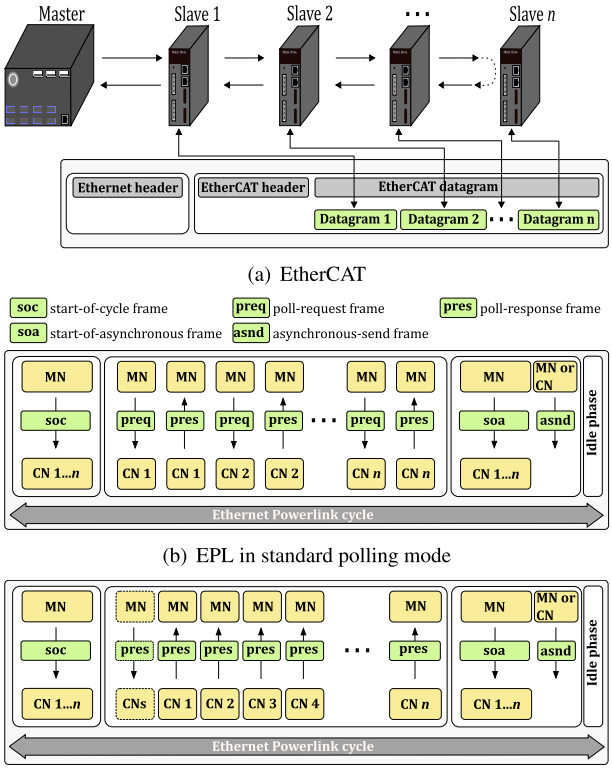



**FIGURE 1. Industrial Ethernet: (a) EtherCAT, (b) EPL in standard polling mode, and (c) EPL in poll-response chaining mode. EtherCAT has a master-slave control structure in a ring topology. It integrates multiple messages into a single EtherCAT frame and circulates the frame in the network. In EPL, a managing node controls all communication and the network is shared among controlled nodes using time-division multiple access. Poll-response chaining is an EPL extension to reduce protocol overhead.**

## II. BACKGROUND

### A. INDUSTRIAL ETHERNET AND CLOCK SYNCHRONIZATION

Industrial Ethernet, as standardized by IEC 61784 and 61158-2, has many attractive features for networked control systems, including high transmission speed, and compatibility with TCP/IP [1], [4], [5], [7], [24], [25]. Among numerous industrial Ethernet solutions, EtherCAT and Ethernet Powerlink are gaining increasing ground in precision applications such as motion control systems. By requiring modification at the Ethernet data-link layer, they ensure deterministic communication delays and support very short control cycles in the microsecond range.

EtherCAT enables high-speed real-time communication among networked devices by the use of summation frames and hardware-based frame relays [4], [7], [25]–[28]. As shown in Fig. 1 (a), EtherCAT has a master-slave control structure in a ring topology at the physical level, where the master controls traffic and initiates all transmissions. Each slave, when receiving a message, processes it and forwards it to the next connected slave. The basic message unit, called a datagram in EtherCAT, contains a command that reads and/or writes data at the addressed memory in slave devices. When a message is relayed by a destined slave, the output or input data is, respectively, written on or read from the buffer memory in the slave. By integrating multiple messages into a single Ethernet frame and circulating frames in the network, EtherCAT achieves very high bandwidth utilization. Moreover, since the frames are relayed by a hardware switch at each device, the message delivery time is almost deterministic and, hence, it is possible to design a highly synchronized distributed system. Owing to the desirable features, EtherCAT is currently being applied in various control applications, including factory automation, robotic surgery, and production machinery [23], [29]–[31].

EPL is an industrial Ethernet solution commonly used in automation systems, ranging from simple I/O to complex motion control applications [2], [6], [32]–[36]. EPL messages are encapsulated in the standard Ethernet frames, and different message types are defined, including start-of-cycle, poll-request, poll-response, start-of-asynchronous, and asynchronous-send. To enable deterministic communication, the network is shared using a time-division multiple access method and, as in EtherCAT, a special host called managing node (MN) controls all communication. Fig. 1 (b) illustrates a cycle in EPL. For each cycle, MN first multicasts a start-of-cycle message to all other devices, known as controlled

nodes (CNs) in EPL. This message synchronizes all CNs and signals the start of a new isochronous phase. During the isochronous phase, cyclic control messages are exchanged: MN unicasts the poll-request message to each CN, and the CN that receives the request immediately multicasts the poll-response message to MN. Note that the request and response messages carry the output and input data of MN, respectively. An asynchronous phase then follows: MN multicasts a start-of-asynchronous message and the MN or CN designated by the message sends the asynchronous-send message that contains a single asynchronous data.

The poll-response chaining (PRC) mechanism is a latest EPL extension to improve performance. It aims to reduce protocol overhead, especially in line topology when nodes exchange small amounts of data. As shown in Fig. 1 (c), instead of poll-request messages, a single poll-response message is used by MN to carry all outputs to CNs, and is multicast immediately after the start-of-cycle message. In PRC mode, the transmission of poll-response frames by each CN is automated using preconfigured timers, such that consecutive frames are separated from each other with minimum intervals to avoid collisions.

The duration of the isochronous phase depends on many factors, such as network topology, cyclic data size, and the use of a PRC mechanism. Before starting a request poll, MN has to wait for a predetermined time period to ensure that all CNs receive and process the precedent frames. The topology of an EPL network is basically either a line or star, and the mandatory wait period differs depending on the topology and/or CN position because the wait time is determined by the propagation delay and round-trip time for the CN. The line topology is preferable for factory transport systems because the star topology using a hub switch has cabling difficulties in the long-range connection condition. However, the line topology has longer wait times compared with the star topology due to the inherently increased hop count. The exact calculations of wait times will be detailed in Section IV, where we discuss the performance impact of Ethernet technology.

Precise clock synchronization is a feature that is becoming increasingly important in modern distributed control systems [8], [9]. The globally synchronized clock, often referred to as global time, enables highly synchronized operations among distributed devices. The networked motor drives in an industrial robot, for example, can utilize the events from the synchronized clock to actuate their associated motors synchronously, such that the rendered motion accurately follows the desired trajectory. High-precision measurement is another example, where distributed sensing devices can synchronously latch input data based on the global time [37].

One of the most attractive features of EtherCAT is the availability of a precisely synchronized clock, i.e., the DC [12], [27], [28], [38]. The globally synchronized clock is very accurate and, in many cases, it has a deviation of only tens of nanoseconds. Basically, the DC-enabled slave that is closest to the master acts as the timing reference for the entire network. During the initialization phase, the master calculates the offsets of slave-local clocks from the reference clock, and delivers them to slaves. Using the offset, each slave can then determine the global clock based on its local clock. After initialization, the master periodically broadcasts the value of the reference clock, with which slaves compensate the drift of local oscillators.

In contrast, EPL has a simple mechanism to support only isochronous operations [10], [39]. The start-of-cycle frame is used as the basis for the common timing of all CNs. In addition, the frame can optionally contain the network time at the MN and, as a result, each CN can adjust its local clock when receiving the frame. In industry, several Ethernet solutions rely on the IEEE 1588 PTP to support clock synchronization [40]; examples are Ethernet/IP and Profinet IO. A PTP implementation can easily provide accuracy in the sub-microsecond range [10]. In practice, EPL uses PTP for synchronization, and it is planned to be included in future standards [9].

### B. RELATED WORKS

In recent years, numerous studies have examined the performance of EtherCAT and EPL. Cena *et al.* [34] have conducted a theoretical and simulation-based analysis for EPL in different network configurations. An exhaustive analysis of different performance indicators defined by the IEC 61784-2 standard has been carried out by Vitturi *et al.* [41] for a coordinated motion control application scenario. Knezic *et al.* [2], [32] provided a performance analysis of the EPL PRC mechanism in linear and star topologies and have shown that PRC allows significant performance improvement in comparison to the standard polling mode.

Owing to the deterministic communication delay and clock synchronization of industrial Ethernet, there have been efforts to use Ethernet for motion control systems. Benzi *et al.* [23] present an overall architecture for Ethernet-based electrical drives. They describe communication solutions for single-drive and multi-drive systems, and discuss a layered architecture that encompasses process levels for management as well as real-time control. Several works propose Ethernet-based motion control systems, for example, in applications of lift control [33], computer numerical control (CNC) [6], and robotics [42]. These studies, however, do not address how to analyze the drive delays and, hence, enhance performance. Kim *et al.* [3] have proposed a delay analysis applicable to an EtherCAT-based motion control system. Using stochastic analysis, they analyzed the end-to-end delay from message release to motor actuation. Our work is distinguished from theirs in the sense that while they focus on period synthesis to minimize the periods of tasks for reduced actuation delays, we present the software design to realize a deterministic and synchronized motor drive in terms of both actuation and sensing, and try to fully cover the design and implementation of drive tasks.

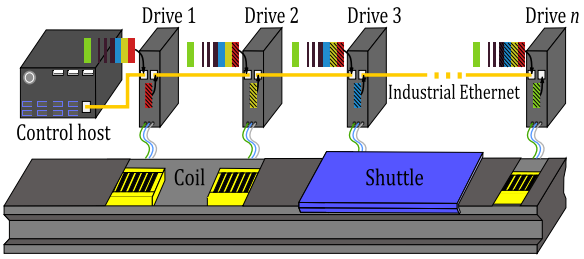



**FIGURE 2.** An industrial transport system consisting of a motion control host and several linear motor drives connected via EtherCAT. The host periodically sends a velocity or torque command to each drive, and each drive applies current to the associated coil to produce the desired shuttle motion. The drive additionally monitors shuttle status and sends the shuttle position to the host.

## III. ETHERNET-BASED LINEAR MOTOR DRIVE

### A. LINEAR-MOTOR TRANSPORT SYSTEM

An industrial transport system consists of a motion control host and several Ethernet-connected linear motor drives (Fig. 2). The control host maintains a centralized position loop for each shuttle and periodically sends a velocity or torque command to each motor drive to achieve coordinated shuttle motion. Given the target velocity or torque input, each drive operates a local loop and applies appropriate current to the associated stator or coil to produce the desired torque. Moreover, the drives use connected sensors to monitor shuttle status, such as actual shuttle position, and send the collected information to the host.

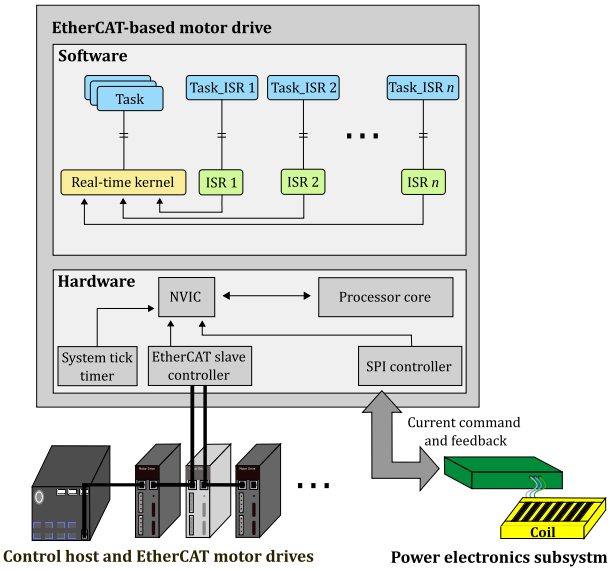



**FIGURE 3.** Organization of an EtherCAT-based motor drive, which primarily consists of a processor core, a system timer, a Nested Vector Interrupt Controller, an EtherCAT controller, and an SPI controller that handles communication with power electronics.

Our target system is an EtherCAT-based motor drive consisting of an embedded computing system and a power electronics subsystem, as shown in Fig. 3. The hardware organization of the computing system mainly consists of a processor core, a system timer, a Nested Vector Interrupt Controller (NVIC), an EtherCAT controller, and an SPI controller that handles communication with power electronics. The power electronics part is responsible for converting the current commanded by the processor into an analog signal and determining the shuttle position from encoder signals. For the EtherCAT slave controller, the motor drive uses a Microchip Technology LAN9252 [43], which covers the physical and data link layers of the communication stack, while the software running on the processor handles the upper layers. The NVIC provides a prioritized interrupt mechanism to improve processor performance and reduce interrupt latency. The priority consists of a group priority for preemption between different interrupts, and a sub-priority that determines the order of execution between interrupts with the same group priority. However, the NVIC does not support flexible access to shared resources and can cause priority inversion [44]. In a motor driving system, multiple interrupt service routines (ISRs) must use shared resources, such as encoders, sensors, and data buffers communicated with the control host.

Thus, our design uses a real-time operating system with mutex that supports the priority inheritance protocol [44]. The processor core runs the lightweight real-time kernel and multiple periodic tasks, which are detailed later in this section. Each ISR is associated with a dedicated task and activates the task on each invocation. In the drive, this is implemented utilizing the semaphore primitives, sem_release and sem_acquire, provided by the real-time operating system we use. The real-time kernel provides hardware independence and portability benefits, but the problem is that the start time of critical task instances can be affected by the ISR for lower priority task instances. To cope with this problem and ensure very periodic servo activation, the operating system has been extended to support *no-preempt switching*. Interrupts that release a no-preempt task are set to the highest priority in the system, and the switch interrupt is assigned the second highest priority. When an ISR wakes up the no-preempt task, interrupts remain disabled even after the context switch is complete, preventing other ISRs from preempting during the transition. After applying this, the activation jitter of the servo task was reduced from 1.09 $\mu$s to 6.9 $n$s.

### B. DRIVE TASK DESIGN FOR DETERMINISTIC MOTOR OPERATION AND MINIMIZED DELAY

Aimed at a software architecture that enables delay analysis, the drive function has been constructed with several periodic tasks running on a real-time operating system. In our drive software design, we assumed that all tasks arrive periodically and are scheduled by a rate-monotonic fixed priority scheduling algorithm [45]. Tasks with shorter periods are assigned higher priorities, and tasks with the same priority are scheduled in a round-robin fashion. The task model is a set of periodic tasks $\tau_j$, denoted by the set of parameters $\tau_j(T_j, O_j, C_j, B_j, J_j)$; $T_j$ is the period of task $\tau_j$, which is fixed; $O_j$ is the fixed offset of $\tau_j$, which is the release time of the first instance of $\tau_j$ ($0 \leq O_j < T_j$); $C_j$ is the worst case execution time of $\tau_j$; $B_j$ is the blocking factor, representing the worst-case time by which $\tau_j$ can be delayed to acquire the

semaphores already locked by lower priority tasks; and $J_j$ is a bounded delay between task arrival and release.

With the task model described above, we have five periodic tasks in each drive, namely, $\tau_j = \{\tau_{srv}, \tau_{sin}, \tau_{sou}, \tau_{eth}, \tau_{msc}\}$. The servo task $\tau_{srv}$ executes an integrated control loop for the torque and velocity commands. Because this task has a stringent requirement of release jitter and usually has the shortest period in the drive, it is assigned the highest priority and set to no-preempt. Tasks $\tau_{sou}$ and $\tau_{sin}$ are responsible for synchronous actuation and sensing, respectively, where $\tau_{sou}$ outputs the data produced by $\tau_{srv}$ and ensures that the intended current flows through the coil. Similarly, $\tau_{sin}$ reads the shuttle position and prepares for transfer to the host. In addition, $\tau_{eth}$ runs the protocol stack for real-time Ethernet messaging. Activated by the arrival of an Ethernet frame, this task extracts the host command and stores it in a memory area shared with $\tau_{srv}$. Tasks $\tau_{eth}$ and $\tau_{srv}$ access the shared memory mutually exclusively by using a semaphore. Additionally, $\tau_{eth}$ is responsible for delivering the shuttle position updated by $\tau_{sin}$, and $\tau_{msc}$ is the lowest priority task performing other jobs, such as processing of non-real-time messages for drive configuration and monitoring [46]–[48]. Hereafter, $\tau_j^i(T_j^i, O_j^i, C_j^i, B_j^i, J_j^i)$ refers to the parameters of task $j$ in drive $i$, where $1 \le i \le n$ and $j \in \{srv, sin, sou, eth, msc\}$. For notational convenience, $\tau_j(T_j, O_j, C_j, B_j, J_j)$ represents the parameters of task $j$ in any of the drives. Our design is general enough that drives can have different parameter values but, for simplicity, we assume identical values unless otherwise stated.

In this paper, we address the drive operation delays in terms of actuation delay and sensing delay. The first metric is defined as follows.

*Definition 1:* The actuation delay, denoted by $\delta_{act}$, is defined as the time between the start of a control cycle on the host, and motor actuation on the drive.

With industrial Ethernet, we can determine $O_{eth}^i$ as the message delay for drive $i$, which is the time required for the host to deliver a message to the drive. Notably, $O_{eth}^i$ may vary depending on the Ethernet technology used and the topological location of the drive. Additionally, the message size and number of drives may affect $O_{eth}^i$. Based on the task and message delay models, we analyze the worst-case response time for $\tau_{eth}^i$ and obtain $f_{srv}^i$, the completion time of $\tau_{srv}^i$ arriving after completion of $\tau_{eth}^i$.

Once the $\tau_{srv}$'s in all drives are completed, synchronized actuation can be achieved via synchronous output. Our design executes the $\tau_{sou}$'s of all drives simultaneously using global time. This is equivalent to setting $O_{sou}$ to the same predetermined value, with $\delta_{act}$ minimized when $O_{sou} = \max_{1 \le i \le n} f_{srv}^i$.

The second delay metric for an Ethernet-based motor drive is aimed at measuring the freshness of drive data observed by the control host, which is defined as follows.

*Definition 2:* The sensing delay, $\delta_{sen}$, is defined as the time between the motor status read by the drive, and the start of the next host cycle.

Similar to motor actuation, status sensing in the drive is performed synchronously by the $\tau_{sin}$ tasks, which are released with the same offset. The larger the offset, the smaller the sensing delay. Because each $\tau_{sin}$ must be completed before the Ethernet frame arrives, $\delta_{sen}$ is minimized when $O_{sin}$ is maximized while satisfying $f_{sin}^i \le O_{eth}^i, \forall i, 1 \le i \le n$, where $f_{sin}^i$ denotes the completion time of $\tau_{sin}$.

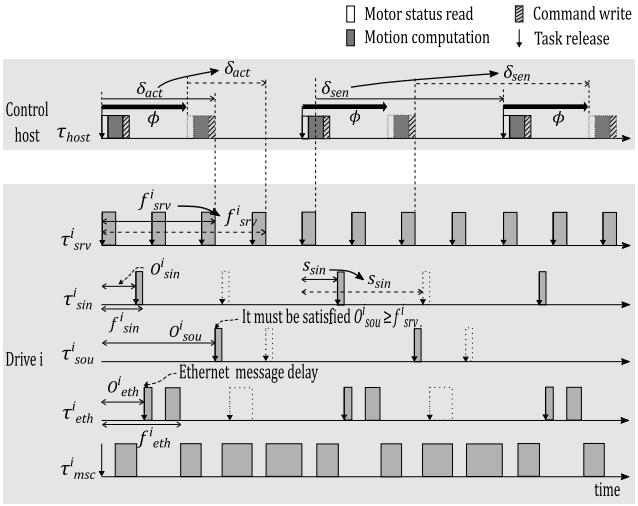



**FIGURE 4.** Actuation ($\delta_{act}$) and sensing ($\delta_{sen}$) delays of drive $i$. Shifting the relative phase of the host cycle with respect to the drive cycle may result in different $\delta_{act}$ and $\delta_{sen}$. The figure shows a case of decreased $\delta_{act}$ with a phase shift of $\phi$.

In a clock-synchronized system, shifting the relative phase of the host cycle with respect to the drive cycle can result in different $\delta_{act}$ and $\delta_{sen}$ minimums. Fig. 4 shows the case where $\delta_{act}$ decreases when the phase shifts by $\phi$. Let $T_{host}$ denote the host period. With the normalized delay $\delta$ defined as $\delta = \alpha\delta_{act} + (1-\alpha)\delta_{sen}, 0 \le \alpha \le 1$, our design attempts to find $\phi (0 \le \phi < T_{host})$ that minimizes $\delta$ for a given $\alpha$.

To calculate the worst-case response time, we adopted the analysis from a previous study [49], [50]. For each individual task with fixed offset and release jitter, we compute the worst-case response time within the hyperperiod, where the hyperperiod of $\tau_j$ is calculated as the least common multiple of all higher-priority tasks, including $\tau_j$. All task instances of any higher-priority task $\tau_k$ can be divided into three sets at the time $\tau_j$ is released. The worst-case response time is then determined by calculating the amount of interference for the three sets for every instance of $\tau_k$ within the hyperperiod [49].

Table 1 summarizes the heuristic for finding the best phase. The input is a periodic task set ($\tau_{host}$, $\tau_{srv}^i$, $\tau_{sin}^i$, $\tau_{sou}^i$, $\tau_{eth}^i$), where $\tau_{host}$ is the task running on the control host. The heuristic tries to find the host phase $\phi^*$ that minimizes the normalized delay by repeatedly computing $\delta_{act}$ and $\delta_{sen}$ while increasing $\phi$, and finally returns ($\delta^*$, $\phi^*$, $O_{sou}^*$, $O_{sin}^*$), where $\delta^*$ is the minimum of the normalized delay; and $O_{sou}^*$ and $O_{sin}^*$, respectively, are the offsets producing the smallest actuation and sensing delays with $\phi^*$. FindWCRT($\tau_1, \ldots, \tau_j$) calculates the worst-case response time of $\tau_j$ by applying the

**TABLE 1. Phase search heuristic.**

**FindPhaseAndOffsets**

**input:** $(\tau_{host}, \tau^i_{srv}, \tau^i_{sin}, \tau^i_{sou}, \tau^i_{eth})$ **where** $\tau^i_j = (T^i_j, O^i_j, C^i_j, B^i_j, J^i_j)$ **and** $i = 1, \cdots, n$

**output:** $(\delta^*, \phi^*, O^*_{sou}, O^*_{sin})$

**begin procedure**

1. $\phi = 0, \delta^* = 2T_{host}$;
2. **while (true)**
3. // update $\tau_{eth}$ offsets regarding $\phi$
4. **for** $i = 1$ **to** $n$ **do**
5. $O^i_{eth}$ = CalcEthOffset($i$, $n$) + $\phi$;
6. **end for**
7. // calculate the minimum possible actuation delay
8. **for** $i = 1$ **to** $n$ **do**
9. $f^i_{eth} = O^i_{eth}$ + FindWCRT($\tau^i_{srv}$, $\tau^i_{sin}$, $\tau^i_{eth}$);
10. $f^i_{srv} = \left\lceil \frac{f^i_{eth}}{T_{srv}} \right\rceil \times T_{srv} + C_{srv}$;
11. **end for**
12. $\delta_{act} = max_{1 \le i \le n} f^i_{srv} - \phi$;
13. // calculate the minimum sensing delay
14. $O_{sin} = 0, s_{sin} = 0$;
15. **while (true)**
16. $f_{sin} = O_{sin}$ + FindWCRT($\tau_{srv}$, $\tau_{sin}$);
17. **if** $(f_{sin} > \min_{1 \le i \le n} O^i_{eth})$
18. **break**;
19. **end if**
20. $s_{sin} = O_{sin}, O_{sin} = O_{sin} + \Delta t$;
21. **end while**
22. $\delta_{sen} = T_{host} + \phi - (\left\lfloor \frac{s_{sin}}{T_{srv}} \right\rfloor \times T_{srv} + C_{srv})$;
23. // update outputs if $\phi$ improves $\delta^*$, the smallest delay thus far
24. $\delta = \alpha\delta_{act} + (1 - \alpha)\delta_{sen}$;
25. **if** $(\delta < \delta^*)$
26. $\delta^* = \delta$;
27. $\phi^* = \phi, O^*_{sou} = \delta_{act} + \phi, O^*_{sin} = s_{sin}$;
28. **end if**
29. $\phi = \phi + \Delta t$;
30. **end while**

**end procedure**

abovementioned analysis to a given set of tasks $(\tau_1, \ldots, \tau_j)$, which are listed in decreasing order of priority. The **while** loop in lines 2–30 is the main loop for finding $\phi^*$, which is obtained by repeatedly evaluating $\delta$ for various $\phi$ from zero to $T_{host}$. This loop first updates the message arrival times $O^i_{eth}$'s based on $\phi$ and the topological drive position (lines 4–6). CalcEthOffset($i$, $n$) calculates $O^i_{eth}$ and is described in detail subsequently with real Ethernet technologies. With the updated $O^i_{eth}$'s, the loop then computes the completion time of $\tau^i_{srv}$ for each drive $i$ (lines 8–11) to obtain the minimum possible $\delta_{act}$ (line 12). Next, we find the minimum sensing delay. The **while** loop on lines 15–21 calculates the completion time of $\tau_{sin}$ (line 16) and compares it with the earliest frame arrival time of the drives (line 17). It then attempts to determine the maximum allowable $O_{sin}$ by repeating the loop while increasing $O_{sin}$ and exiting if $\tau_{sin}$ is not completed before the frame arrives. The minimum $\delta_{sen}$ is obtained according to line 22. Next, with the computed $\delta_{act}$ and $\delta_{sen}$, the normalized delay $\delta$ is calculated and compared with the smallest $\delta^*$ thus far. If the current $\phi$ improves $\delta^*$, it sets $\delta^*$ to $\delta$ and updates $\phi^*$, $O^*_{sou}$, and $O^*_{sin}$ (lines 25–28).

### C. ETHERNET MESSAGE DELAYS

The Ethernet message delay can be analyzed by considering the actual message delivery mechanism. This section presents

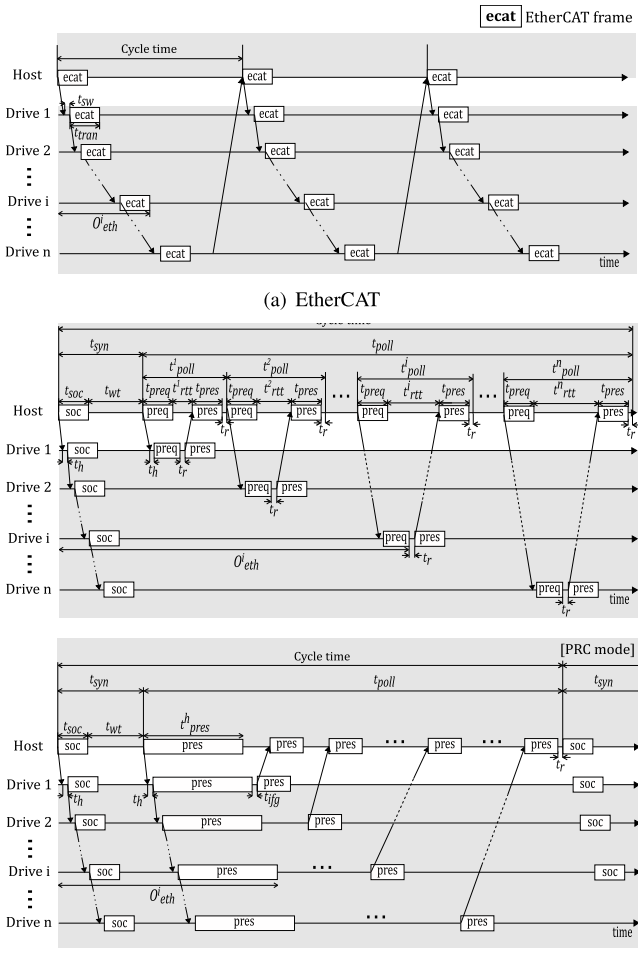



**FIGURE 5. Message delay $O^i_{eth}$ in (a) EtherCAT and (b) EPL. In EtherCAT, $t_{sw}$ is the switching delay for the frame relay in each drive, and $t_{tran}$ is the frame transmission time. In EPL, an isochronous phase consists mainly of $t_{sync}$ and $t_{poll}$, which represent the synchronization and poll periods, respectively.**

the message delay models in EtherCAT and EPL, respectively, and describes the calculation of $O^i_{eth}$ for each drive $i$, $1 \le i \le n$.

In EtherCAT, the master or the control host integrates all messages for a cycle into one or more Ethernet frames and circulates the frames in the line-topology network. The frames traverse every drive and eventually return to the host after reaching the end of the network. Fig. 5(a) illustrates $O^i_{eth}$, the message delay for drive $i$, where $t_{sw}$ is the switching delay for the frame relay in each drive, and $t_{tran}$ is the time required for frame transmission. Thus, $O^i_{eth}$ in EtherCAT can be calculated as

$$O^i_{eth} = i \times t_{sw} + t_{tran}. \quad (1)$$

Owing to hardware-based switching, $t_{sw}$ is very small, approximately 0.5 $\mu$s for the controller used in the experiment [43], but if the number of drives is very large, the accumulated $t_{sw}$ becomes significant for drives near the network end.

The $O^i_{eth}$ in an EPL isochronous phase is shown in Fig. 5(b) when using standard polls and PRC, respectively, in a line topology. In the figure, $t_{soc}$, $t_{preq}$ and $t_{pres}$ denote the time required to transmit frames for start-of-cycle, poll-request and poll-response, respectively. The duration of an isochronous phase is the sum of $t_{sync}$ and $t_{poll}$, the time required for the synchronization and poll periods, respectively. The term $t_{sync}$ is calculated as

$$t_{sync} = t_{soc} + t_{wt}.$$

The mandatory wait time $t_{wt}$ is to ensure that all CNs, i.e., all drives, receive and process the start-of-cycle message. In standard polling mode, $t_{poll}$ is written as $t_{poll} = \sum_{i=1}^{n} t^i_{poll}$, where $t^i_{poll}$ represents the time taken for the request and response polls with drive $i$, expressed as $t^i_{poll} = t_{preq} + t_{pres} + t^i_{rtt} + t_r$, $1 \le i \le n$. The term $t^i_{rtt}$ is the round-trip time of drive $i$, and $t_r$ is the node response time. Thus, $t_{poll}$ can be written as

$$\begin{aligned} t_{poll} &= \sum_{i=1}^{n} t^i_{poll} = \sum_{i=1}^{n} \left( t_{preq} + t_{pres} + t^i_{rtt} + t_r \right) \\ &= n \left( t_{preq} + t_{pres} + t_r \right) + \sum_{i=1}^{n} t^i_{rtt}. \end{aligned} \tag{2}$$

Note that $t^i_{rtt}$ includes the propagation and hub repetition delays and varies greatly depending on the topology. Without loss of accuracy, we ignore the propagation delay (approximately 5 ns/m) because it is very small compared to the hub delay (approximately 500 ns). Let $t_h$ denote the hub delay. Based on the argument of the round-trip delay [2], $t^i_{rtt}$ can then be written as $t^i_{rtt} = (2i-1)t_h + t_r$ for line topology, and as $t^i_{rtt} = 2t_h + t_r$ for star topology. Let $t^0_{poll} = 0$. In standard polling mode, $O^i_{eth}$ of EPL is then obtained as

$$O^i_{eth} = t_{sync} + \sum_{k=0}^{i-1} t^k_{poll} + t_{preq} + t^i_h, \tag{3}$$

where $t^i_h = (i-1) \times t_h$ for line topology, and $t^i_h = t_h$ for star topology.

In PRC mode, $t^h_{pres}$ represents the time required to transmit a poll-response by the MN or the motion host, which contains all outputs to drives. Then $t_{poll}$ can be expressed as

$$t_{poll} = t^h_{pres} + n \times (t_{pres} + t_{ifg}) + t^n_{rtt} + t_r. \tag{4}$$

Note that consecutive poll-response frames are generated with the minimum interval required to avoid collisions, i.e., an Ethernet inter-frame gap, of which the value is denoted by $t_{ifg}$. It holds that $t^h_{pres} \le n \times t_{preq}$ and that $t^n_{rtt} \le \sum_{i=1}^{n} t^i_{rtt}$. Thus, from Eqs. (2) and (4), the gain of reduced $t_{poll}$ in PRC mode is expected to be best in situations where the message size is relatively small and line topology is employed. The message delay $O^i_{eth}$ in PRC mode is given by

$$O^i_{eth} = t_{sync} + t^h_{pres} + t^i_h. \tag{5}$$

**TABLE 2.** Calculation of $O^i_{eth}$ with drive position ($i$) and number of drives ($n$).

**CalcEthOffset**
**input:** $(i, n)$ **/* for EtherCAT */**
**output:** $O^i_{eth}$
// $S_{hoe}$ : Ethernet header size (14 byte)
// $S_{ecat}$ : Maximum frame size (1498 byte)
// $S_{hoc}$ : EtherCAT header size (2 byte)
// $S_{hod}$ : EtherCAT datagram header and working counter size (12 byte)
// $S_{msg}$ : Size of request and response messages. (24 or 100 byte)
// $B$ : Network bandwidth, ether 100 Mbps or 1 Gbps
**begin procedure**
1. $t_{sw} = 0.5\mu$s;
2. $t_{tran} = (\lceil n \times (S_{msg} + S_{hod})/S_{ecat} \rceil \times (S_{hoe} + S_{hoc}) + n \times (S_{msg} + S_{hod}))/B$;
3. $O^i_{eth} = i \times t_{sw} + t_{tran}$;
**end procedure**

**CalcEthOffset**
**input:** $(i, n)$ **/* for EPL */**
**output:** $O^i_{eth}$
// $S_{preq}$, $S_{pres}$ : Poll request and response message size (equals to $S_{msg}/2$ each)
// $S_{hoe}$ : Ethernet header size (14 byte)
// $S_{soc}$ : EPL start-of-cycle frame size (46 byte)
// $S_{hop}$ : Header size for poll request or response (3 byte)
// $B$ : Network bandwidth, ether 100 Mbps or 1 Gbps
**begin procedure**
1. $t_r = 1\mu$s;
2. $t_h = 0.5\mu$s;
3. $t_{soc} = S_{soc}/B$;
4. $t_{preq} = (S_{hoe} + S_{hop} + S_{preq})/B$;
5. $t_{pres} = (S_{hoe} + S_{hop} + S_{pres})/B$;
6. $t^h_{pres} = (S_{hoe} + n \times (S_{hop} + S_{preq}))/B$;
7. **if** network topology is LINE
8. $t_{wt} = (n-1) \times t_h$;
9. $t^i_h = (t-1) \times t_h$;
10. $\sum_{i=1}^{i} t^i_{rtt} = \sum_{i=1}^{i} \{(2i-1)t_h + t_r\}$;
11. **else** /* STAR topology */
12. $t_{wt} = t_h$;
13. $t^i_h = t_h$;
14. $\sum_{i=1}^{i} t^i_{rtt} = \sum_{i=1}^{i} \{2t_h + t_r\}$;
15. **end if**
16. $t_{sync} = t_{soc} + t_{wt}$;
17. **if** polling mode is STD
18. $\sum_{k=0}^{i-1} t^k_{poll} = (i-1) \times (t_{preq} + t_{pres} + t_r) + \sum_{i=1}^{i} t^i_{rtt}$;
19. $O^i_{eth} = t_{sync} + \sum_{k=0}^{i-1} t^k_{poll} + t_{preq} + t^i_h$;
20. **else** /* PRC mode */
21. $O^i_{eth} = t_{sync} + t^h_{pres} + t^i_h$;
22. **end if**
**end procedure**

Putting the above descriptions together, CalcEthOffset($i$, $n$) in Table 2 shows the procedure to calculate $O^i_{eth}$ in EtherCAT and EPL, respectively. While EtherCAT assumes a line topology (LINE), EPL additionally includes a star topology (STAR) and takes into account the polling mode used, i.e., standard mode (STD) or poll-response chaining mode (PRC). For EPL, assuming that "autoresponse" is used, we estimate $t_r$ as $t_r = 1\ \mu$s.[1]

## IV. PERFORMANCE EVALUATION

This section validates the drive software design and evaluates the performance. We implemented an EtherCAT motor drive operating with periodic torque commands. Using the prototype drive, we verify the effectiveness of the task phase shift and examine the position tracking errors. In addition,

[1]The node response time $t_r$ can be reduced using a hardware-assisted frame processing mechanism, called autoresponse, available on the network interface. A preconfigured mask automatically triggers the transmission of a prepared frame when it receives a frame that meets the filter conditions.

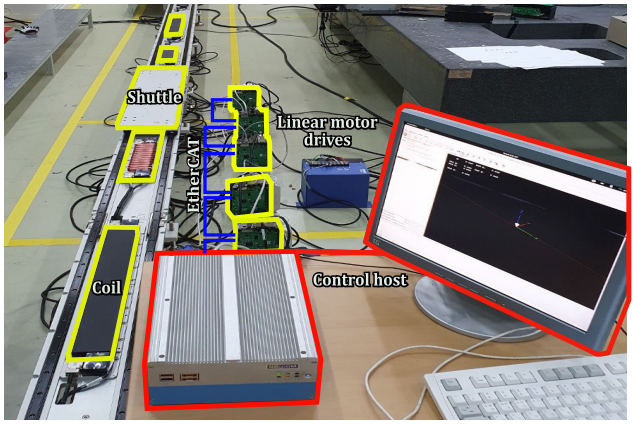



**FIGURE 6.** An experimental testbed built up with a PC-based control host and a group of linear motor drives interconnected by EtherCAT in line topology. A motion controller is set up using RTAI-patched Linux and IgH EtherCAT master stack. LinuxCNC is used to generate motion trajectories. The experimental system achieves desired shuttle movements by controlling motor drives through EtherCAT-based communication of real-time process data.

we discuss the impact of Ethernet technologies and investigate drive delays for different network topologies, message sizes, and network bandwidths.

### A. EXPERIMENTAL SETUP

For the evaluation of the proposed software design, we built up an experimental testbed with a PC-based control host and several linear motor drives interconnected by EtherCAT in line topology. Fig. 6 shows the testbed. A motion controller was set up using RTAI-patched Linux [51], [52] and an IgH EtherCAT master stack [27], [29], [53], [54]. As the host control software, we use LinuxCNC [6], [14], [55], [56] to generate motion trajectories. Because LinuxCNC has a component-based structure, it is possible to integrate new drives into an existing motion control system by developing the periodic input and output interfaces [57]. This, in our case, corresponded to the implementation of an EtherCAT messaging component that complies with the CiA 402 standard drive profile [58]. The experimental system achieves desired shuttle movements by controlling motor drives through EtherCAT-based communication of real-time process data. Table 3 summarizes the specifications of the hardware and software used in the experiment.

Based on the prototype drive hardware, we measured the values of task parameters, such as execution times, blocking factors, and release jitters. The measurement results of $\tau_j(T_j, O_j, C_j, B_j, J_j)$ are listed in Table 4. In addition to pure task execution time, $C_j$ includes the time overhead for context switching and semaphore operations with ISRs associated with $\tau_j$. Also note that $\tau_{srv}$ is very sensitive to activation jitter, so it was set up as a no-preempt task, and the ISR that releases $\tau_{srv}$ was assigned the highest priority. For measurements, the drive software was instrumented to output a signal to the debug port at the event of interest, and the time between events was observed and recorded using an oscilloscope.

**TABLE 3.** Details of motion control system.

| Item | Description |
|---|---|
| Motor drive | |
| Processor | ARM Cortex M4-based STM32F407 MCU |
| RAM/ ROM | 192 KB SRAM/ 1MB flash memory |
| Network I/F | Microchip LAN9252 EtherCAT slave controller |
| Power electronics | Includes encoder, three 12-bit ADCs, two 12bit DACs, and Hall sensor interface. |
| Operating system | Real-time kernel with priority-based scheduler and priority inheritance protocol. |
| Real-time messages | Two message sets complying with the CiA 402 drive profile, each consisting of 6 and 25 data objects with total sizes of 24 and 100 bytes, respectively. |
| Task parameters (in $\mu$s) | $\tau_j(T_j, O_j, C_j, B_j, J_j)$ = { $\tau_{srv}$(250, 0, 29.24, 2, 0.28), $\tau_{sin}$(1000, 540, 24.67, 17, 2.94), $\tau_{sou}$(1000, 780, 25.64, 17, 2.89), $\tau_{eth}$(1000, 666, 6.68, 22, 1.69), $\tau_{msc}$(2000, 0, 1000, 0, 4) } |
| Control host | |
| CPU | Intel Core i5 6500 operating at 3.2 GHz |
| Memory | 4GB DDR4 DRAM |
| Operating system | Linux Debian 3.4-9 with RTAI patch |
| Motion control | LinuxCNC 2.7.11 & IgH EtherCAT master stack. Control cycle set to 1 ms. |
| Network | Intel EXPI9301CTBLK 1GB/s Ethernet adapter |

**TABLE 4.** Measured values of task parameters.

| $\tau_j$ | Parameter values ($\mu$s) | | | | |
|---|---|---|---|---|---|
| | $T_j$ | $O_j$ | $C_j[S_{msg}$= 24 bytes, 100 bytes] | $B_j$ | $J_j$ |
| $\tau_{srv}$ | 250 | - | [29.24, 30.26] | 2 | 0.28 |
| $\tau_{sin}$ | $T_{host}$ | - | [24.67, 92.50] | 17 | 2.94 |
| $\tau_{sou}$ | $T_{host}$ | - | [25.64, 99.33] | 17 | 2.89 |
| $\tau_{eth}$ | $T_{host}$ | - | [6.68, 7.01] | 22 | 1.69 |
| $\tau_{msc}$ | $2T_{host}$ | - | [273.31, 269.83] | 0 | 2.96 |

Measurements were collected for 10 min for each $\tau_j$, and a cold reset was applied to the drive prior to conducting each experimental trial. The experiment was repeated for a message size $S_{msg}$ of 24 and 100 bytes. With the host period $T_{host}$ given, the offsets, $O_{eth}$, $O_{sin}$, and $O_{sou}$ can then be computed by the heuristics in Table 1 and Table 2. Notice that $O_{eth}$ may vary depending on the drive number; our heuristic takes this value into account to determine $O_{sin}$ and $O_{sou}$.

### B. PHASE-SHIFTED LOOP

In experiments involving the drive, we first evaluated the performance of task phase shift and measured the current errors when the host phase is shifted according to the proposed algorithm. The control host was set up to generate a 1–10 Hz sine wave current with an amplitude of 1000 mA and send a command every 1 ms using EtherCAT. Then, with the host command as input, the drive outputs the commanded current to its associated coil. Additionally, the drive reports the current feedback to the host. We measured the target and actual currents on the host and calculated the absolute value of the difference as the error. Measurement data were collected for 10000 cycles after 5 min of warm up.

It has been observed that the phase-shifted loop significantly reduces the error. Fig. 7 shows a plot of the measured values for a 2-Hz sine current. When no phase shift was applied ($\phi = 0$), the mean error was 33.64 mA, the standard deviation was 3.11 mA, and the maximum error was 103 mA. By contrast, when the host phase was shifted by 660 $\mu$s ($\phi = 660$), which is the best phase analyzed with $\alpha = 0.6$, the mean and standard deviation decreased to 26.43 and 2.01 mA, respectively. The maximum error was 99 mA.

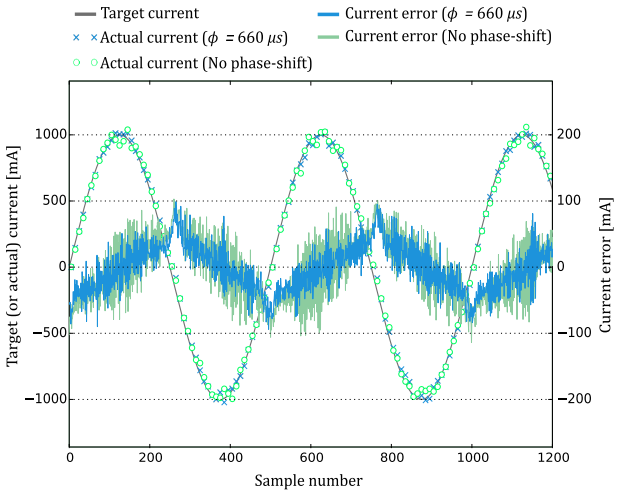



**FIGURE 7.** Target and actual currents measured on the control host. The host produces a sine current and sends a target current every 1 ms. The drive actuates the commanded current and reports the current feedback to the host. The error, defined as the difference between the target and actual currents, is shown to decrease at $\phi = 660$ $\mu$s, the best phase obtained from the heuristic.

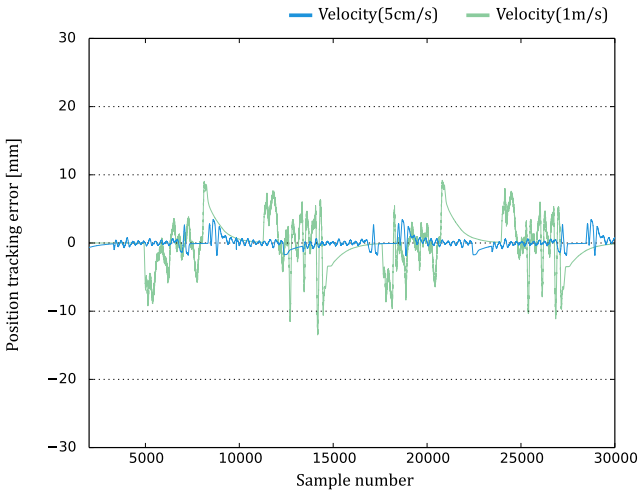
**TABLE 5.** Measurement results of current errors.

| | $\phi$ ($\mu$s) | Current errors (mA) | | |
|---|---|---|---|---|
| | | Average | Maximum | Std. dev. |
| No phase-shift | 0 | 33.64 | 103 | 3.11 |
| Phase-shifted loop | 660 | 26.43 | 99 | 2.01 |

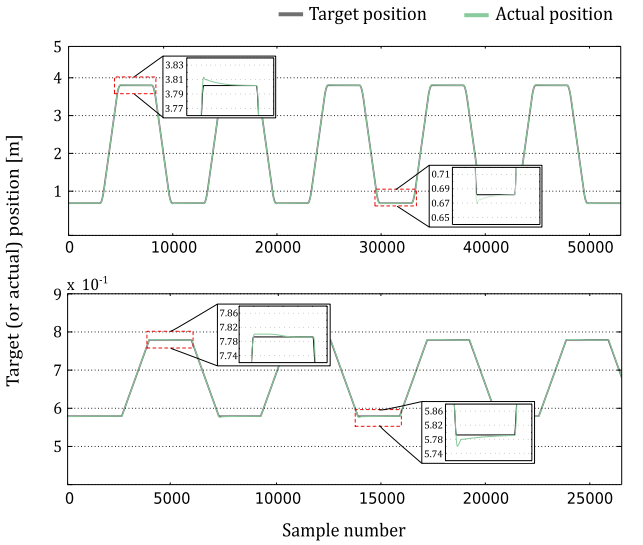
The results are summarized in Table 5. Note that in the experiment, all host and drive tasks were scheduled using EtherCAT global time, so that the intended offsets for $\tau^i_{sou}$ and $\tau^i_{sin}$ could be realized by configuring the DC interrupts of drive $i$ to occur with the desired phases. In Fig. 7, significant improvements can be seen in samples of around 150, 600 and 1100. The experimental results indicate that the phase shift reduced the mean error by up to 21.4%.

To verify and demonstrate the transport capability of the drive, we examined positioning errors while moving the shuttle. Position tracking error, defined as the difference between the actual position and the target position, was measured for two movement cases: low-speed movement within a drive, and high-speed movement between drives. For intra-drive movement, the shuttle was made to reciprocate repeatedly in a 220-mm section within the range controlled by a drive. Tracking errors were measured for average shuttle speeds of 5, 10, and 15 cm/s. For the case of inter-drive movement, the shuttle repeated high-speed reciprocating motions in a straight 3120-mm section. Here, the shuttle was driven by individual or cooperative propulsion of coils, each controlled by a separate motor drive. A total of five drives were used in the experiment. Tracking errors were measured for shuttle speeds of 1, 1.5, and 2 m/s. The control host was set up to calculate the required trajectory and send a torque command to the drive every 1 ms.

The measurement results of position tracking errors are shown in Table 6. For inter-drive shuttle movement, the




(a) Position tracking error



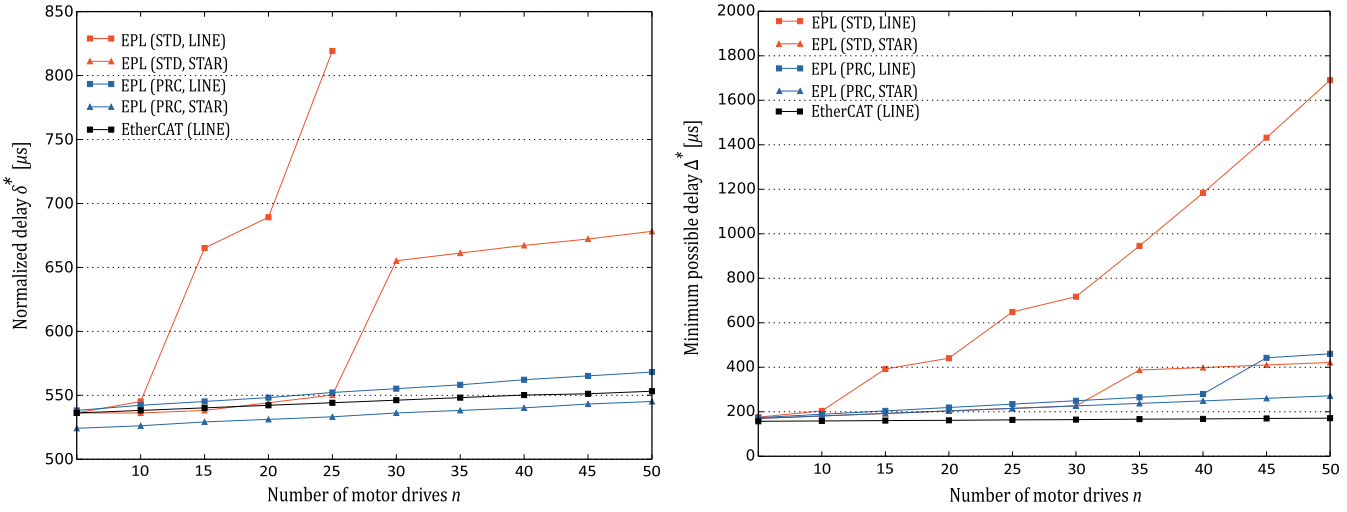

(b) Target and actual position profile

**FIGURE 8.** Tracking errors and position profiles. (a) Position tracking errors for intra-drive movement (5 cm/s) and inter-drive movement (1 m/s), and (b) target and actual position profiles at shuttle speeds of 2 m/s (upper figure) and 15 cm/s (lower figure), respectively. It can be observed that tracking errors are properly managed and that the target and actual positions closely match.

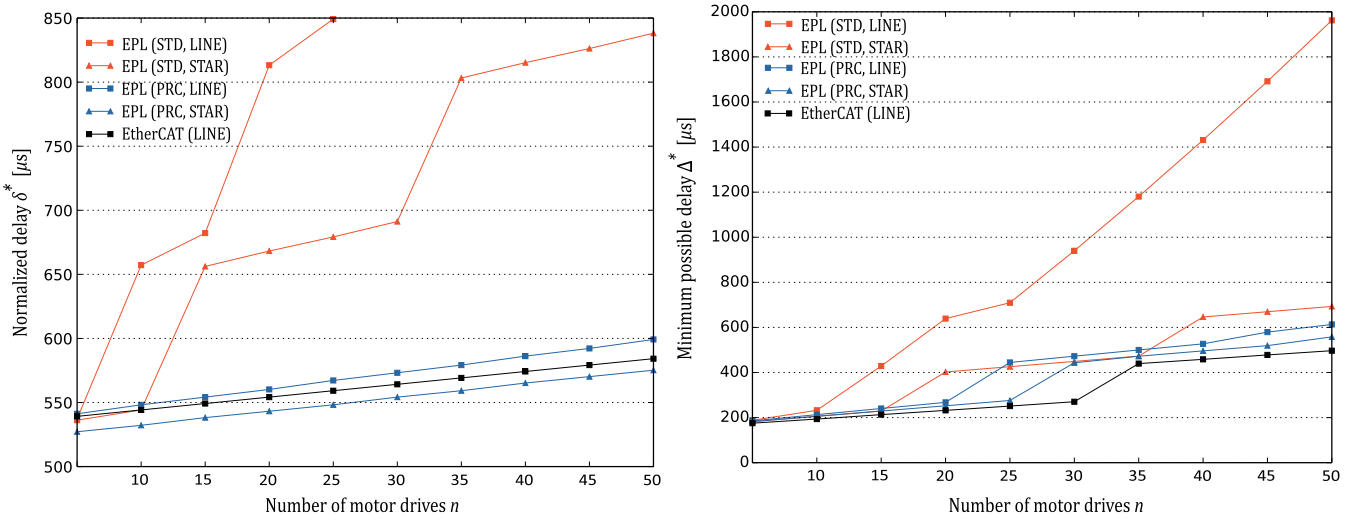
**TABLE 6.** Measurement results of position tracking errors.

| | Velocity | Position tracking errors (mm) | | |
|---|---|---|---|---|
| | | Average | Maximum | Std. dev. |
| Intra-drive movement | 15 cm/s | 0.427 | 4.181 | 0.545 |
| | 10 cm/s | 0.369 | 4.005 | 0.598 |
| | 5 cm/s | 0.380 | 3.462 | 0.549 |
| Inter-drive movement | 2.0 m/s | 5.169 | 39.223 | 6.744 |
| | 1.5 m/s | 4.222 | 33.644 | 3.815 |
| | 1.0 m/s | 2.125 | 13.859 | 2.238 |

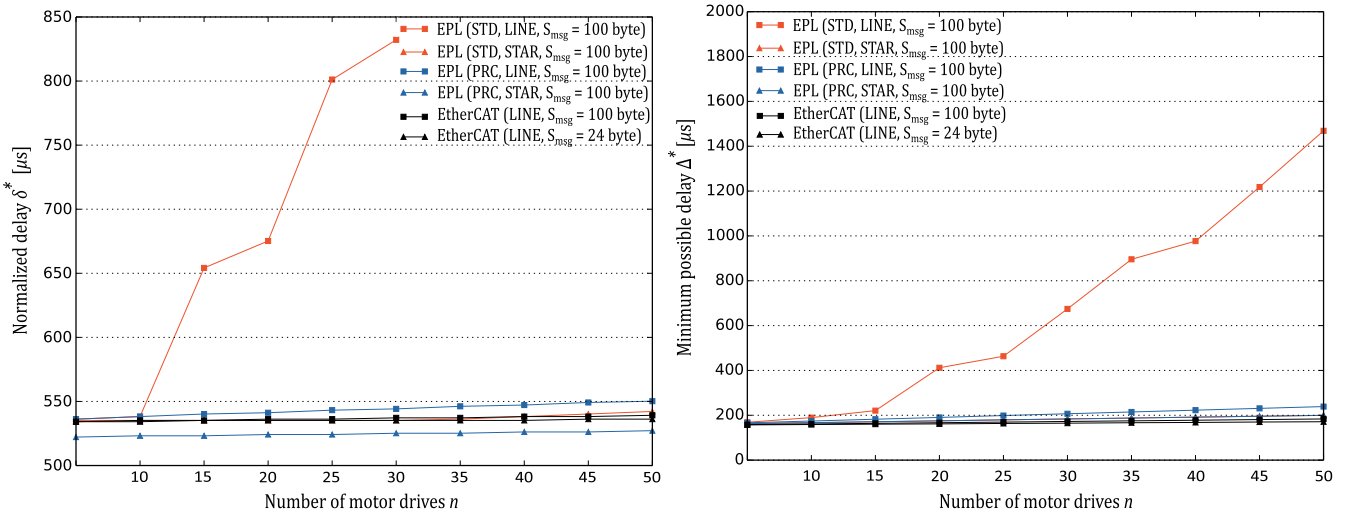
average tracking error was measured in the millimeter range, and an error of up to 39.223 mm was observed at a velocity of 2 m/s. In contrast, for intra-drive movement, it has been found that the average tracking error remains below 1 mm for all speed profiles. The maximum and standard deviation




(a) $S_{msg}$ = 24 bytes, $B$ = 100 Mbps




(b) $S_{msg}$ = 100 bytes, $B$ = 100 Mbps




(c) $S_{msg}$ = 24 bytes or 100 bytes, $B$ = 1 Gbps

**FIGURE 9.** Normalized delay $\delta^*$ ($T_{host}$ = 1 ms) and minimum possible delay $\Delta^*$ according to number of drives. EtherCAT produces a relatively small delay throughout the experiment, whereas in standard polling mode, EPL delay increases significantly as the number of drives or message size increases in a 100 Mbps network. For EPL, we can see that star topology and/or poll-response chaining significantly improve delay performance.

of the tracking error showed similar values regardless of the shuttle speed. Fig. 8 shows the tracking errors and position profiles. It can be observed that tracking errors are properly managed. As can be seen in Fig. 8(a), the tracking error is kept low when the shuttle moves at low speeds. For a shuttle speed of 5 cm/s, the tracking error remains small throughout the movement, and the peak is 3.462 mm. The maximum error increases to 13.859 mm in the 1 m/s high-velocity profile. From the results in Fig. 8(b), we can see that the target and actual position profiles closely match in both movement cases.

### C. ACTUATION AND SENSING DELAY

To investigate the effect of different Ethernet mechanisms on performance, we analyzed and compared the drive delays in EtherCAT and EPL. Fig. 9 shows the experimental results and plots the normalized delay $\delta^*$ and the minimum possible delay $\Delta^*$ according to the number of drives $n$. The task parameter values in Table 4 were used for the experiment. For $\delta^*$, we used the host period $T_{host}$ of 1 ms. For $\Delta^*$, we first determined the smallest host period with the given number of drives and Ethernet technology, and then ran the heuristic with the period to get the minimum possible delay $\Delta^*$.

From $\delta^*$ in Fig. 9, we can see that EtherCAT produces a relatively small delay throughout the experiment, whereas in standard polling mode, EPL delay increases significantly as the number of drives or message size increases in a line-topology network. On the other hand, EPL in PRC mode using a star topology always showed the lowest delay in all cases, demonstrating the notable performance benefits of the latest EPL extension. We see that this is because the hub delay in the star topology is kept small and is not affected by the number of drives. Fig. 9 (a) shows that in standard polling mode using a 100 Mbps line-topology network, the maximum $n$ allowed is 25 and the corresponding $\delta^*$ is 849.19 $\mu$s, as opposed to 548.19 $\mu$s for PRC mode in star topology.

The delay $\Delta^*$ in Fig. 9 illustrates the effect of the number of drives on the minimum achievable delay with EtherCAT and EPL. It was found that EtherCAT can control more than 100 drives with $T_{host} = 1$ ms at 100 Mbps and has the lowest $\Delta^*$ in all cases. However, in the line-topology standard polling EPL network, $\Delta^*$ also increased rapidly with increasing $n$. This is because each EtherCAT slave reads or writes datagram and then passes it to the next slave immediately, whereas EPL messages are exchanged sequentially between the MN and CNs. It is also notable that at a 1-Gbps bandwidth, the minimum feasible cycle for $n \leq 50$ is less than 200 $\mu$s in all cases except for the standard polling EPL in the line topology.

In EPL with the standard polling mode, we can see that the network topology has a significant impact on the drive delay. Both $\delta^*$ and $\Delta^*$ of EPL always had lower values in the star topology than in the line topology. However, the line topology is preferable for factory transport systems. The star topology is relatively more expensive to install when constructing the network and has difficulties in network management and reconfiguration. In particular, for an industrial transport system requiring tens or hundreds of drives, cabling in star topology may not be a feasible solution. Thus, given the number of drives and the cycle time requirement, the topology should be carefully determined taking into account the trade-off between the delay performance and management costs.

It should be noted that the PRC mechanism, the new EPL standard feature, significantly improves the delay performance. As can be seen from $\delta^*$ in Fig. 9, PRC reduces the EPL delay for all message sizes and topologies. In particular, it is important to observe that PRC improves EPL to have comparable performance in line topology, given that the line is the best topology for a factory transport system. It was found that in star topology, EPL using PRC even outperforms EtherCAT. This contrasts with previous findings that EtherCAT has the highest throughput and the shortest round-trip delay among industrial Ethernet standards. However, EtherCAT has smaller feasible host cycles than EPL, and thus still has smaller minimum delays, which is shown by $\Delta^*$ in the figure.

## V. CONCLUSION

This paper has proposed a systematic approach to designing Ethernet-based drives for synchronized motor operation with minimized delay. For the target system, we construct the drive function with several periodic tasks run by a rate-monotonic real-time scheduler and realize synchronized actuation and monitoring via global time-based I/O. In order to minimize the delay required for the control host to actuate or sense the motor in the drive, we proposed the phase-shifted loop method and presented a heuristic for finding the best phase. We experimentally evaluated the performance of the phase-shifted loop and conducted a comparative study of delay performance with EtherCAT and EPL for various parameters, such as the number of drives, the message size, the network topology, and the bandwidth. The contribution and major results of the paper can be summarized as follows.

- The paper proposed the design of an Ethernet-based linear motor drive that enables end-to-end delay analysis for synchronized motor actuation and sensing in industrial transport systems.
- The paper proposed a phase-shifted loop method and presented a heuristic to find the best phase that minimizes drive delay while ensuring synchronized motor operation across all drives. The performance was verified experimentally, and the results show that the phase shift could reduce the average current error by up to 21.4%.
- We investigated the impact of Ethernet technology and compared the minimum delays of EtherCAT and EPL. Notably, the new EPL extension, the PRC mechanism, has been shown to significantly reduce EPL delays and provide comparable performance in line topology, while EtherCAT has smaller feasible host cycles than EPL, and thus still has smaller minimum delays.

In our future research, we will study how to further improve the drive delay by considering different software design and applying other Ethernet mechanisms to motor drives.

## REFERENCES


[1] S. Wang, J. Ouyang, D. Li, and C. Liu, "An integrated industrial Ethernet solution for the implementation of smart factory," *IEEE Access*, vol. 5, pp. 25455–25462, 2017.

[2] M. Knezic, B. Dokic, and Z. Ivanovic, "Theoretical and experimental evaluation of Ethernet powerlink PollResponse chaining mechanism," *IEEE Trans. Ind. Informat.*, vol. 13, no. 2, pp. 923–933, Apr. 2017.

[3] K. Kim, M. Sung, and H.-W. Jin, "Design and implementation of a delay-guaranteed motor drive for precision motion control," *IEEE Trans. Ind. Informat.*, vol. 8, no. 2, pp. 351–365, May 2012.

[4] L. Seno, S. Vitturi, and C. Zunino, "Real time Ethernet networks evaluation using performance indicators," in *Proc. IEEE Conf. Emerg. Technol. Factory Autom.*, Sep. 2009, pp. 1–8.

[5] B. M. Wilamowski and J. D. Irwin, *Industrial Communication Systems*. Boca Raton, FL, USA: CRC Press, 2016.

[6] K. Erwinski, M. Paprocki, L. M. Grzesiak, K. Karwowski, and A. Wawrzak, "Application of Ethernet powerlink for communication in a linux RTAI open CNC system," *IEEE Trans. Ind. Electron.*, vol. 60, no. 2, pp. 628–636, Feb. 2013.

[7] D. Jansen and H. Buttner, "Real-time Ethernet: The EtherCAT solution," *Comput. Control Eng.*, vol. 15, no. 1, pp. 16–21, Feb. 2004.

[8] A. Mahmood, R. Exel, H. Trsek, and T. Sauter, "Clock synchronization over IEEE 802.11—A survey of methodologies and protocols," *IEEE Trans. Ind. Informat.*, vol. 13, no. 2, pp. 907–922, Apr. 2017.

[9] G. Cena, I. C. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "Synchronize your watches: Part I: General-purpose solutions for distributed real-time control," *IEEE Ind. Electron. Mag.*, vol. 7, no. 1, pp. 18–29, Mar. 2013.

[10] G. Cena, I. Cibrario Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "Synchronize your watches: Part II: Special-purpose solutions for distributed real-time control," *IEEE Ind. Electron. Mag.*, vol. 7, no. 2, pp. 27–39, Jun. 2013.

[11] G. Cena, I. C. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "On the accuracy of the distributed clock mechanism in EtherCAT," in *Proc. IEEE Int. Workshop Factory Commun. Syst. Proc.*, May 2010, pp. 43–52.

[12] G. Cena, I. C. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "Evaluation of EtherCAT distributed clock performance," *IEEE Trans. Ind. Informat.*, vol. 8, no. 1, pp. 20–29, Feb. 2012.

[13] P. Cronin, F. S. Hosseini, and C. Yang, "A low overhead solution to resilient assembly lines built from legacy controllers," *IEEE Embedded Syst. Lett.*, vol. 10, no. 3, pp. 103–106, Sep. 2018.

[14] E. Wings, M. Müller, and M. Rochler, "Integration of real-time Ethernet in LinuxCNC," *Int. J. Adv. Manuf. Technol.*, vol. 78, nos. 9–12, pp. 1837–1846, Jun. 2015.

[15] K. Wang, C. Zhang, X. Xu, S. Ji, and L. Yang, "A CNC system based on real-time Ethernet and windows NT," *Int. J. Adv. Manuf. Technol.*, vol. 65, nos. 9–12, pp. 1383–1395, Apr. 2013.

[16] J. F. Gieras, Z. J. Piech, and B. Tomczuk, *Linear Synchronous Motors: Transportation and Automation Systems*. Boca Raton, FL, USA: CRC Press, 2016.

[17] S.-Y. Chen, H.-H. Chiang, T.-S. Liu, and C.-H. Chang, "Precision motion control of permanent magnet linear synchronous motors using adaptive fuzzy fractional-order sliding-mode control," *IEEE/ASME Trans. Mechatronics*, vol. 24, no. 2, pp. 741–752, Apr. 2019.

[18] K. Sato, M. Katori, and A. Shimokohbe, "Ultrahigh-acceleration moving-permanent-magnet linear synchronous motor with a long working range," *IEEE/ASME Trans. Mechatronics*, vol. 18, no. 1, pp. 307–315, Feb. 2013.

[19] W. Geelen, D. Antunes, J. P. M. Voeten, R. R. H. Schiffelers, and W. P. M. H. Heemels, "The impact of deadline misses on the control performance of high-end motion control systems," *IEEE Trans. Ind. Electron.*, vol. 63, no. 2, pp. 1218–1228, Nov. 2016.

[20] Y. Kim, I. Kim, I. Kang, T. Kim, and M. Sung, "Formal modeling and verification of motor drive software for networked motion control systems," *J. Universal Comput. Sci.*, vol. 20, no. 14, pp. 1903–1925, 2014.

[21] K. Kozlowski, M. Kowalski, M. Michalski, and P. Parulski, "Universal multiaxis control system for electric drives," *IEEE Trans. Ind. Electron.*, vol. 60, no. 2, pp. 691–698, Feb. 2013.

[22] G. Gu, L. Zhu, Z. Xiong, and H. Ding, "Design of a distributed multiaxis motion control system using the IEEE-1394 bus," *IEEE Trans. Ind. Electron.*, vol. 57, no. 12, pp. 4209–4218, Dec. 2010.

[23] F. Benzi, G. S. Buja, and M. Felser, "Communication architectures for electrical drives," *IEEE Trans. Ind. Informat.*, vol. 1, no. 1, pp. 47–53, Feb. 2005.

[24] M. Felser, "Real time Ethernet: Standardization and implementations," in *Proc. IEEE Int. Symp. Ind. Electron.*, Jul. 2010, pp. 3766–3771.

[25] J. Jasperneite, M. Schumacher, and K. Weber, "Limits of increasing the performance of industrial Ethernet protocols," in *Proc. IEEE Conf. Emerg. Technol. Factory Autom. (EFTA)*, Sep. 2007, pp. 17–24.

[26] T. Maruyama and T. Yamada, "Communication architecture of EtherCAT master for high-speed and IT-enabled real-time systems," in *Proc. IEEE 20th Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2015, pp. 1–8.

[27] EtherCAT Technology Group. *Ethernet for Control Automation Technology*. Accessed: Feb. 2021. [Online]. Available: http://www.ethercat.org

[28] BECKHOFF. (2017). *Sheet-ET1100, Hardware Data Sheet Version 2.0*. [Online]. Available: https://download.beckhoff.com/download/Document/io/ethercat-development-products/ethercat_et1100_datasheet_v2i0.pdf

[29] M. Cereia, I. C. Bertolotti, and S. Scanzio, "Performance of a real-time EtherCAT master under linux," *IEEE Trans. Ind. Informat.*, vol. 7, no. 4, pp. 679–687, Nov. 2011.

[30] R. Ramesh, S. Jyothirmai, and K. Lavanya, "Intelligent automation of design and manufacturing in machine tools using an open architecture motion controller," *J. Manuf. Syst.*, vol. 32, no. 1, pp. 248–259, Jan. 2013.

[31] M. Sung, I. Kim, and T. Kim, "Toward a holistic delay analysis of EtherCAT synchronized control processes," *Int. J. Comput. Commun. Control*, vol. 8, no. 4, pp. 608–621, 2013.

[32] M. Knezic, B. Dokic, and Z. Ivanovic, "Performance analysis of the Ethernet powerlink PollResponse chaining mechanism," in *Proc. IEEE World Conf. Factory Commun. Syst. (WFCS)*, May 2015, pp. 1–4.

[33] A. Soury, M. Charfi, D. Genon-Catalot, and J.-M. Thiriet, "Performance analysis of Ethernet powerlink protocol: Application to a new lift system generation," in *Proc. IEEE 20th Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2015, pp. 1–6.

[34] G. Cena, L. Seno, A. Valenzano, and S. Vitturi, "Performance analysis of Ethernet powerlink networks for distributed control and automation systems," *Comput. Standards Interface*, vol. 31, no. 3, pp. 566–572, Mar. 2009.

[35] L. Seno, S. Vitturi, and C. Zunino, "Analysis of Ethernet powerlink wireless extensions based on the IEEE 802.11 WLAN," *IEEE Trans. Ind. Informat.*, vol. 5, no. 2, pp. 86–98, May 2009.

[36] L. Seno and S. Vitturi, "A simulation study of Ethernet powerlink networks," in *Proc. IEEE Conf. Emerg. Technol. Factory Autom. (EFTA)*, Sep. 2007, pp. 740–743.

[37] P. Ferrari, A. Flammini, D. Marioli, and A. Taroni, "A distributed instrument for performance analysis of real-time Ethernet networks," *IEEE Trans. Ind. Informat.*, vol. 4, no. 1, pp. 16–25, Feb. 2008.

[38] D. Orfanus, R. Indergaard, G. Prytz, and T. Wien, "EtherCAT-based platform for distributed control in high-performance industrial applications," in *Proc. 18th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, 2013, pp. 1–8.

[39] *Ethernet Powerlink Communication Profile Specification, Version 1.4.0*, E. P. S. Group, New Delhi, India, 2018.

[40] J. C. Eidson, M. Fisher, and J. White, *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, Standard 1588, 2002.

[41] S. Vitturi, L. Peretti, L. Seno, M. Zigliotto, and C. Zunino, "Real-time Ethernet networks for motion control," *Comput. Standards Interface*, vol. 33, no. 5, pp. 465–476, 2011.

[42] S. G. Robertz, R. Henriksson, K. Nilsson, A. Blomdell, and I. Tarasov, "Using real-time java for industrial robot control," in *Proc. 5th Int. Workshop Java Technol. Real-Time Embedded Syst.*, 2007, pp. 104–110.

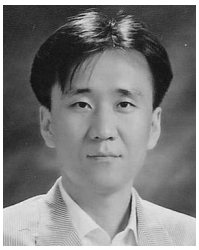
[43] Michrochip. (2015). *LAN9252-2/3-Port EtherCAT Slave Controller with Integrated Ethernet PHYs*. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/00001909A.pdf

[44] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Trans. Comput.*, vol. 39, no. 9, pp. 1175–1185, Dec. 1990.

[45] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[46] A. Girbea, C. Suciu, S. Nechifor, and F. Sisak, "Design and implementation of a service-oriented architecture for the optimization of industrial applications," *IEEE Trans. Ind. Informat.*, vol. 10, no. 1, pp. 185–196, Feb. 2014.
[47] D. Hastbacka, L. Barna, M. Karaila, Y. Liang, P. Tuominen, and S. Kuikka, "Device status information service architecture for condition monitoring using OPC UA," in *Proc. IEEE Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2014, pp. 1–7.
[48] T. Sauter and M. Lobashov, "How to access factory floor information using Internet technologies and gateways," *IEEE Trans. Ind. Informat.*, vol. 7, no. 4, pp. 699–712, Nov. 2011.
[49] O. Redell and M. Torngren, "Calculating exact worst case response times for static priority scheduled tasks with offsets and jitter," in *Proc. 8th IEEE Real-Time Embedded Technol. Appl. Symp.*, Sep. 2002, pp. 164–172.
[50] K. Tindell, *Adding time-offsets to schedulability Analysis*. Princeton, NJ, USA: Citeseer, 1994.
[51] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio, "Performance comparison of VxWorks, Linux, RTAI, and Xenomai in a hard real-time application," *IEEE Trans. Nucl. Sci.*, vol. 55, no. 1, pp. 435–438, Feb. 2008.
[52] *RTAI*. Accessed: Feb. 2021. [Online]. Available: http://www.rtai.org
[53] *IgH EtherCAT Master for Linux*. Accessed: Feb. 2021. [Online]. Available: http://www.etherlab.org
[54] M. Cereia and S. Scanzio, "A user space EtherCAT master architecture for hard real-time control systems," in *Proc. IEEE 17th Int. Conf. Emerg. Technol. Factory Autom.*, Sep. 2012, pp. 1–8.
[55] T. Hu, P. Li, C. Zhang, and R. Liu, "Design and application of a real-time industrial Ethernet protocol under linux using RTAI," *Int. J. Comput. Integr. Manuf.*, vol. 26, no. 5, pp. 429–439, May 2013.
[56] M. Minhat, V. Vyatkin, X. Xu, S. Wong, and Z. Al-Bayaa, "A novel open CNC architecture based on STEP-NC data model and IEC 61499 function blocks," *Robot. Comput.-Integr. Manuf.*, vol. 25, no. 3, pp. 560–569, Jun. 2009.
[57] D. Yu, Y. Hu, X. W. Xu, Y. Huang, and S. Du, "An open CNC system based on component technology," *IEEE Trans. Autom. Sci. Eng.*, vol. 6, no. 2, pp. 302–310, Apr. 2009.
[58] CiA. *CiA 402 Series: CANopen Device Profile for Drives and Motion Control*. Accessed: Feb. 2021. [Online]. Available: https://www.can-cia.org/can-knowledge/canopen/cia402/

**SEUNG-YONG LEE** received the B.S. and M.S. degrees in mechanical and information engineering from the University of Seoul, South Korea, in 2014 and 2016, respectively, where he is currently pursuing the Ph.D. degree. He is also working as a Senior Researcher with the Intelligent Robotics Research Center, Korea Electronics Technology Institute. His research interests include system software, automation systems, and real-time systems.



**MINYOUNG SUNG** (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer engineering from Seoul National University, South Korea, in 1995, 1997, and 2002, respectively. He worked with the Software Research Institute, Samsung Electronics, South Korea. He was also associated as a Visiting Scholar with the Department of EECS, University of Michigan, Ann Arbor, MI, USA, in 2005. He is currently a Professor with the Department of Mechanical and Information Engineering, University of Seoul, South Korea. His research interests include system software and automation systems.