INT J COMPUT COMMUN, ISSN 1841-9836
8(4):608-621, August, 2013.

Toward a Holistic Delay Analysis of EtherCAT Synchronized Control Processes

Minyoung Sung, Ikhwan Kim, and Taehyoun Kim*
Dept. of Mechanical and Information Engineering,
University of Seoul, Seoul 130-743, Korea
{mysung, ihkim, thkim}@uos.ac.kr
*Corresponding author

Abstract: This paper analyzes the end-to-end delay of EtherCAT-based control processes that use the events of message frames and global clock for synchronized operation. With the end-to-end delay defined as the time interval between the start of a process cycle and the actual input or output, we develop a holistic delay model for control processes with EtherCAT, by taking into account the time for in-controller processing, message delivery, and slave-local handling. Based on the measurements from a real EtherCAT control system, we discuss the average and deviation of the process delay as we vary the number of slaves and process cycle time. The experiment results show that the output delays are mainly increased by the average controller delay, whereas the input delays are more affected by the deviation rather than the average of the controller delay. Our in-depth analysis on the controller reveals that the DMA (Direct Memory Access) overhead chiefly enlarges the controller delay for increasing number of slaves, while task release jitter is the main cause of the increased delay for longer cycle time. The presented delay model and evaluation results can be essentially used for the design of EtherCAT-based automation that requires highly synchronized operations, such as for coordinated motion and high-precision data sensing.
Keywords: real-time Ethernet, end-to-end delay, synchronized processes, automation system.

1 Introduction

Presently, automated control systems are experiencing a steady but fundamental change, i.e., the use of industrial Ethernet as a replacement for conventional fieldbuses [1, 2]. EtherCAT, in particular, is one of the industrial Ethernet standards (IEC 61784 and 61158-2), which is gaining increasing acceptance in precision automation [3–7]. It offers numerous attractive features such as short cycle time as low as dozens of microseconds [6], globally synchronized clock with deviation of the sub-microsecond range [8], and compatibility with TCP/IP. Because of these benefits, EtherCAT is currently being applied in various automation fields including factory automation, robotics, and production machinery [9–11].

An automation system typically requires highly synchronized operations of its components for coordinated actuation and sensing. For example, an industrial robot should be able to actuate its constituent motors in a synchronized manner so that the consequential motion accurately follows the planned trajectory. High-precision distributed measurement is another example, where synchronous operation is important for synchronized sensing and data freshness [12].

With an Ethernet-based control system, the design of such synchronized operations relies on a thorough analysis of the networked process delay, which includes the time for in-controller processing, message delivery, and local handling by each slave, i.e., the controlled device. Thus far, however, there have been few studies that have analyzed the end-to-end delay of EtherCAT automation systems. Although some recent works have addressed the performance of EtherCAT-based automation, they remain incomplete in that they either lack any consideration of the actual automation workload [13] or only deal with the network-level performance [5–7].

This paper evaluates the performance of EtherCAT-based synchronized processes in terms of the end-to-end delay. In EtherCAT, synchronized control schemes can be realized using events from the EtherCAT frames and synchronized clock. These two types of events relate directly to the precision of modern networked control processes: at each slave, the event of a frame reception begins a new cycle of computation, while the event from the synchronized clock can be utilized to latch input data and/or actuate the device in a globally synchronized manner. We formulate the end-to-end delay of the synchronized control schemes and investigate their performance characteristics in a comprehensive way. Aiming for providing an in-depth and practical insight, the performance evaluation has been conducted using a prototype EtherCAT controller that was constructed from open source software. Based on the measurement results using the controller, we discuss the delay performance as we vary the number of slaves and process cycle time.

The contributions of our work are two-folded. First, we present a delay analysis of the EtherCAT synchronized processes. Using a real automation testbed that operates a number of commercial motor drives, we evaluate the performance of EtherCAT control systems in terms of the average and deviation of the end-to-end process delay. To the best of our knowledge, our study is the first to analyze the end-to-end delay of EtherCAT control processes in a holistic way, considering not only the control network but also the actual in-controller automation workload. The experiment results confirm the importance of our in-depth analysis of the controller delay: the average end-to-end delays of output processes are mainly increased by the average controller delay while the input processes are directly affected by the deviation rather than the average of the controller delay. Second, we show the feasibility of using open source solutions to build up an automation controller. On a bare PC, we set up Xenomai-patched Linux [14] and IgH EtherCAT stack [15]. We used Beremiz [16] to generate automation workload, and we extended its communication interfaces so that it supports EtherCAT network. The experimental controller could successfully operate tens of drives in position or velocity mode with a cycle time of 0.5 ms. The largest actuation deviation among the drives was analyzed to be around 30 μs and 0.1 μs, respectively, in the frame-driven and clock-driven synchronization schemes.

The remainder of this paper is organized as follows. In Section 2, we review the background for EtherCAT-based synchronized processes. In Section 3, we present our end-to-end delay analysis and in Section 4 we describe the evaluation results. Section 5 concludes the paper.

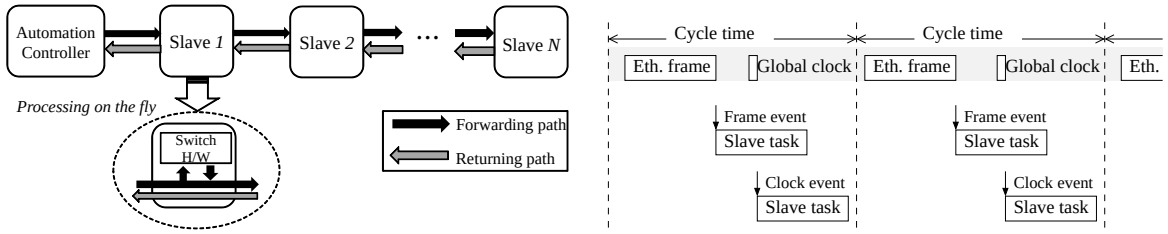
2 Background

2.1 EtherCAT and Synchronized Processes

Among the emerging real-time Ethernet profiles [1], we chose EtherCAT for our analysis because it has desirable features for realizing highly synchronized operations.

First, EtherCAT supports high-speed deterministic communication. Figure 1(a) shows a typical configuration of EtherCAT network, which is in the form of a line topology. Along the forwarding path, every slave in an EtherCAT network relays message frames between the input and output ports *on-the-fly* using switch hardware. When the message frame is relayed by each slave, the output or input process data in the frame is written to, or updated from, the corresponding part of the buffer in the slave. Once an EtherCAT frame arrives at the end of the network, it returns to the controller. EtherCAT allows only peer-to-peer connections between any two consecutive slaves, thereby eliminating the possibility of indeterminism arising from multi-party simultaneous access to the medium. The design of a synchronized process can be greatly benefited from the almost deterministic message delivery time.

Second, EtherCAT provides efficient clock synchronization, which is known as the Distributed Clock (DC) mechanism [8]. Basically, the DC-enabled slave closest to the EtherCAT controller



(a) EtherCAT system in a line topology

(b) Cyclic execution of slave tasks using frame events vs. clock events

Figure 1: EtherCAT-based control system.

acts as the timing reference for the entire EtherCAT network. In the setup phase, the controller collects local clock values from each DC-enabled slave. The controller calculates the offsets between the reference and local clock values, and then it writes the offsets to the slaves, with which they can compute the global time based on their local clocks. After this is done, the controller periodically distributes the value of the reference clock to all the slaves for the compensation of clock skew that is caused by local clock drift. The DC mechanism is rather simple, but it can accurately synchronize the clocks of distributed slaves with a skew of usually less than 1 μs.

EtherCAT provides two types of synchronized events to trigger a local task on each slave, i.e., the frame and clock events (see Fig. 1(b)). The cycle time in this paper is defined as the period with which the controller repeats the automation process. For each process cycle, a slave task can be designed to start its execution immediately after the arrival of an EtherCAT frame that is cyclically generated by the controller. However, such frame-driven cycle may have a relatively large deviation, e.g., a few to dozens of microseconds, due to the variable processing delays on the controller and the intermediate slave devices. Alternatively, a slave task can be invoked by the autonomous interrupt that is generated synchronous to the global clock, in which case the deviation can be reduced by up to a few nanoseconds.

2.2 IEC 61131-3 and Open Source Automation

Currently, the control logic of many industrial automation processes is usually implemented using PLC (Programmable Logic Controller). Owing to the proliferation of PLCs in various industries, it is becoming very important for programmers and field engineers with different domain backgrounds and skills to easily handle PLC-based systems. However, the lack of a consistent approach to PLC programming makes it difficult to integrate devices from different vendors to build a large and complex automation system. In order to solve this problem, the International Electrotechnical Commission (IEC) introduced IEC 61131-3, a standard for programming industrial PLC systems [17].

As IEC 61131-3 is gaining worldwide acceptance by the industry, there have been efforts in the open source community to provide an IEC 61131-3 compliant Integrated Development Environment (IDE). Beremiz [16] is a representative IDE, which is used and distributed freely under the GNU license. Beremiz has three major components: PLCOpen editor, MatIEC backend compiler, and plugin extensions. The PLCOpen editor lets users write PLC programs using the languages defined in the IEC 61131-3 standard. The MatIEC compiler translates the textual form of automation programs into corresponding ANSI C codes.

The most attractive feature of Beremiz is the *plugin* extension. The plugin structure allows adding new functions to the IDE simply with the implementation of the corresponding class definitions. Because of its extensibility and open source policy, Beremiz has been utilized in

many studies of automation systems [18–20]. For the same reason, our prototype controller employed Beremiz, which was extended to include the EtherCAT plugin.

2.3 Related Work

There have been a few studies on the evaluation of EtherCAT networks. Early works [5, 6] formulated the end-to-end delay of the EtherCAT network and analyzed the achievable minimum cycle time according to the slave number and packet sizes. On the basis of a theoretical performance model, Seno *et al.* presented the simulation model and the performance results in terms of two important performance indicators, i.e., minimum cycle time and jitter [7, 21]. Robert *et al.* also presented an in-depth analysis of minimum cycle time for Ethernet-based real-time protocols [22]. Although the previous studies provided the analysis models for analyzing important performance indicators, they did not address the device-level delay factors, only dealing with the network transmission times.

Recent studies have started to address the performance of automation controller or slave devices for networked control systems. Cereia *et al.* evaluated the performance of a Linux-based EtherCAT controller in terms of the cycle accuracy of periodic control tasks [13]. Kim *et al.* presented a statistical delay analysis of EtherCAT motor drives [4]. These studies, however, have limitations because they either lack any consideration of the actual workload [13] or ignore the controller latency [4]. Precise clock synchronization among devices is also crucial for coordinated actuation and sensing. A recent study by Cena *et al.* evaluated the performance of the DC mechanism using extensive measurements [8, 23].

3 EtherCAT Control Processes and Delay Analysis

In this section, we describe the EtherCAT control schemes for synchronized processes [24], and formulate their end-to-end delays. Aiming for an in-depth analysis, we explain the design of an open source EtherCAT controller, and present the delay model for the in-controller processing.

3.1 End-to-end Delay Model of Synchronized Processes

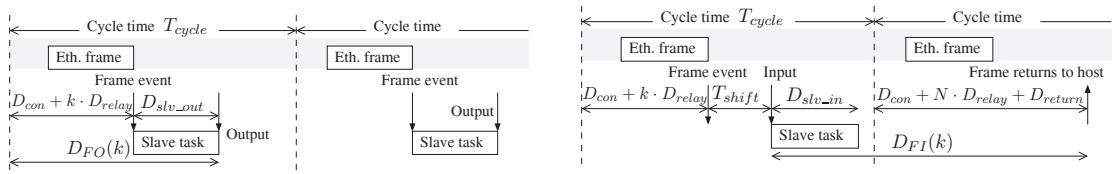
Although EtherCAT allows other topologies such as star and tree, we assume the line topology. It is because the line topology or daisy chain is the most preferred in industries, and our analysis can be easily extended to other types of topologies if a proper model for the frame delivery is given.

Figure 2(a) shows the frame-driven output (FO) scheme that uses the frame event only. We define the end-to-end delay for an output process as the time taken from the start of a process cycle to the corresponding output. Let $D_{FO}(k)$ denote the end-to-end delay on slave k ($1 \leq k \leq N$) in the FO scheme where k means that the slave is k-th nearest to the controller. Table 1 summarizes the notations used in this paper. Assuming homogeneous slaves, we can express $D_{FO}(k)$ as

$$D_{FO}(k) = D_{con} + k \cdot D_{relay} + D_{slv\_out}, \tag{1}$$

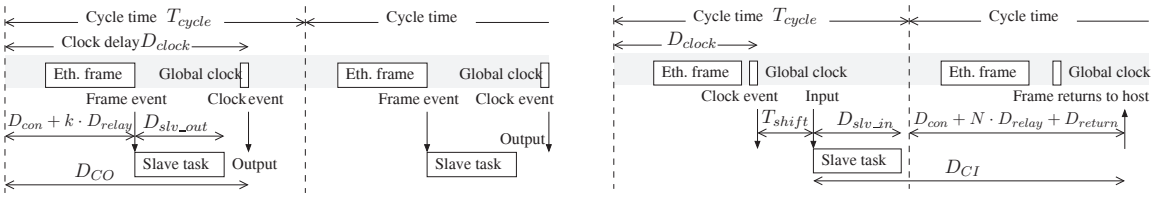
where D_{con}, D_{relay}, and D_{slv_out} refer to the respective delays for in-controller processing, per-slave message relay, and slave-local output handling. The D_{slv_out} is defined as the time taken to compute the output signal, which is usually a constant value given by the slave device vendor. Note that D_{relay} includes the propagation time on the link between consecutive nodes (controller or slave) as well as the link-level handling time to relay frames.

In the frame-driven input (FI) scheme shown in Fig. 2(b), the frame event triggers the slave task that latches an input data and copies it to the memory in the EtherCAT switch hardware.



(a) Output process (b) Input process

Figure 2: Frame-driven synchronized processes.



(a) Output process (b) Input process

Figure 3: Clock-driven synchronized processes.

The slave-local input delay, D_{slv_in} specifies the time required for the latch and copy operation. The input data is conveyed ultimately to the automation controller by the subsequent EtherCAT frame. The slave may allow the controller to adjust the *shift-time*, T_{shift}, by which it delays the input latch to improve the freshness of the sensed data. In the input process, we define the end-to-end delay as the time interval between the time when the input is latched and the time when the frame that contains the input data returns to the controller. The end-to-end delay in the FI configuration is then computed as

$$D_{FI}(k) \quad = \quad T_{cycle} - (D_{con} + k \cdot D_{relay} + T_{shift}) + D_{con} + N \cdot D_{relay} + D_{return}, \tag{2}$$

where T_{cycle} and D_{return} refer to the process cycle time and frame return delay, respectively. In order to ensure that the input data is ready by the arrival of the next frame, it is required that

$$\forall k, D_{con}^{max} + k \cdot D_{relay}^{max} + T_{shift} + D_{slv\_in}^{max} \leq T_{cycle} + D_{con}^{min} + k \cdot D_{relay}^{min}. \tag{3}$$

Therefore, T_{shift} should be determined such that

$$T_{shift} \quad \leq \quad T_{cycle} - (D_{con}^{max} - D_{con}^{min}) - N \cdot (D_{relay}^{max} - D_{relay}^{min}) - D_{slv\_in}^{max}. \tag{4}$$

Note that, hereinafter, X^{avg}, X^{min}, and X^{max} represent the average, minimum, and maximum value of variable X, respectively (e.g., D_{con}^{max} means the maximum of D_{con}).

In order to achieve a higher degree of synchronization, the clock events should be used. As shown in Fig. 3(a), the clock-driven output (CO) scheme uses the clock event to synchronously actuate the output after the computation has been completed. If we denote the delay of the global clock event by D_{clock}, the end-to-end delay in the CO scheme becomes

$$D_{CO}(k) = D_{CO} = D_{clock}, \tag{5}$$

where D_{clock} requires that

$$D_{clock}^{min} \geq D_{con}^{max} + N \cdot D_{relay}^{max} + D_{slv\_out}^{max}. \tag{6}$$

Table 1: Notations.

Notation	Description
N	Number of slaves.
$D_{FO}(k)$	End-to-end delay at slave k ($1 \leq k \leq N$) in the frame-driven output scheme.
$D_{FI}(k)$	End-to-end delay at slave k in the frame-driven input scheme.
D_{CO}	End-to-end delay in the clock-driven output scheme.
D_{CI}	End-to-end delay in the clock-driven input scheme.
D_{con}	In-controller processing time. The delay from the start of a process cycle to the release of EtherCAT frames.
D_{relay}	Per-node message relay time. It includes the time for frame relay by a slave and inter-node frame propagation.
D_{slv_out}	Slave output delay. The time taken to compute the output signal.
D_{slv_in}	Slave input delay. The time required for latching the input data and making it ready to be transferred.
D_{return}	Frame return delay. The time taken for a frame to return from the N-th slave to the controller.
D_{clock}	Global clock delay, the time interval from the start of a process cycle to the generation of the clock event.
D_{con_sw}	In-controller processing time to packetize the control data.
D_{con_dma}	DMA delay to copy data from the EtherCAT driver to the FIFO in the network interface.
T_{cycle}	Process cycle time - constant.
T_{shift}	Shift time by which the slave delays the input latch in order to improve the data freshness - constant.
J_{con_task}	Release jitter of control task.
X^{avg},X^{min}, X^{max}	Average, minimum, and maximum value of variable X: e.g., D_{con}^{max} means the maximum of D_{con}.

The clock-driven input (CI) configuration in Fig. 3(b) uses the clock event to synchronize the input latch. The end-to-end delay is expressed as

$$D_{CI} = T_{cycle} - (D_{clock} + T_{shift}) + D_{con} + N \cdot D_{relay} + D_{return}, \tag{7}$$

where D_{clock} and T_{shift} should be set such that

$$D_{clock}^{max} + T_{shift} + D_{slv\_in}^{max} \leq T_{cycle} + D_{con}^{min} + D_{relay}^{min}. \tag{8}$$

Note that the end-to-end delays in the clock-driven processes are independent of the slave position in the network.

3.2 Controller Delay Analysis

For our study, we constructed an EtherCAT controller using open source software. Our controller design uses Xenomai-patched Linux, IgH EtherCAT protocol stack, and Beremiz as the key software components. This facilitates a highly synchronized control process, whereby the Xenomai kernel significantly reduces the deviation of controller delay, while EtherCAT enhances the predictability of the message and clock events through its almost deterministic message relaying and precisely synchronized clock.

In Beremiz, we implemented the EtherCAT plugin and integrated the EtherCAT stack via the plugin interface. The plugin is mainly composed of two types of classes, each representing the profiles of the controller and slave devices. We also implemented a C wrapper API to the EtherCAT stack. The class definitions together with the EtherCAT API are used by the IEC compiler for the generation of runtime codes. When a new program has been configured to use EtherCAT, the plugin support module imports information on the slave profile and the EtherCAT API from the plugin definition.

The constructed build procedure facilitates the development of EtherCAT automation programs. During the build procedure, illustrated in Fig. 4, PLC codes are translated into C codes

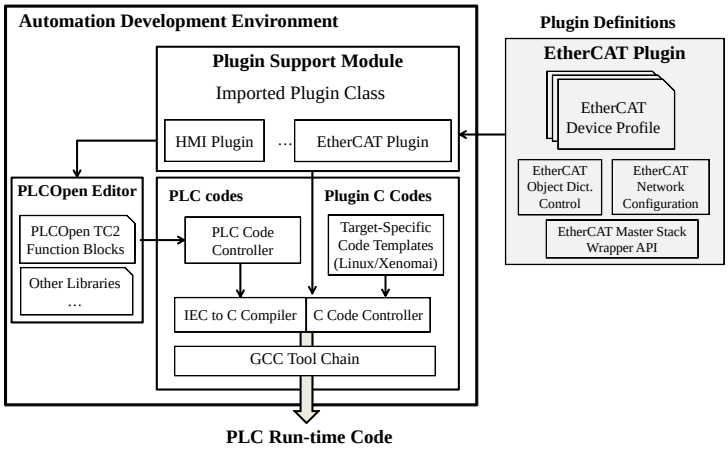


Figure 4: Build procedure for EtherCAT automation programs.

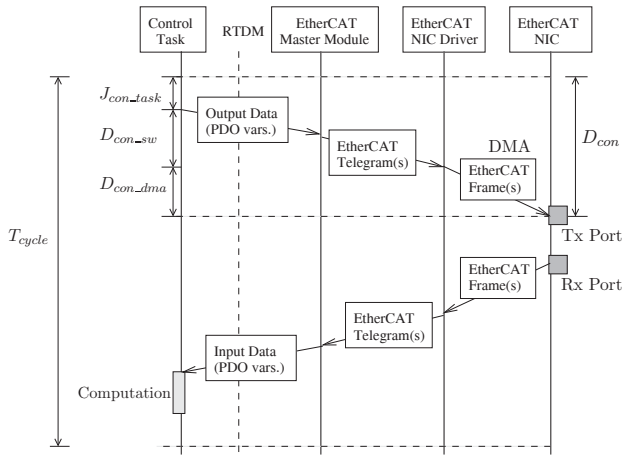


Figure 5: Execution sequence of the control task.

by the IEC compiler, and after being compiled and linked with target-specific stub modules, it is finally converted into an executable runtime. The stubs have interfaces to the time and task management functions of the target operating system, i.e., Xenomai Linux in our case. This build procedure may seem to be complicated, but most of the build steps are performed automatically and thus transparently to users.

In order to model the in-controller delay, we investigate the behavior of the automation runtime. Figure 5 shows the sequence diagram of the control task that executes the automation process. An automation program is typically realized as a periodic control task that repeats reading and writing variables in the slaves. The input and output variables of the task are mapped from the PDOs (Process Data Objects) that are defined in the slave profiles. For each process cycle, the task executes a sequence of transmission of output data, reception of input data, and computation. At the beginning of the cycle, the control task writes output data into the EtherCAT kernel module via the Xenomai interface for real-time I/O, called RTDM (Real-Time Driver Model). In turn, the EtherCAT module packetizes the control data into EtherCAT telegrams and copies them to the buffer in the EtherCAT NIC (Network Interface Card) driver. Finally, the NIC driver encapsulates the telegrams into Ethernet frames and copies them to the transmission FIFO in the NIC.

From the execution flow for frame transmission, we can define the in-controller processing delay D_{con} as the sum of J_{con_task}, D_{con_sw}, and D_{con_dma}. The J_{con_task}, D_{con_sw}, and D_{con_dma}, respectively, refer to the release jitter of the control task, the time taken for copying the data from the user space to the device driver buffer, and the time for sending out EtherCAT frames through the NIC via DMA (Direct Memory Access). Note that D_{con_sw} includes the packetization and queuing overheads as well.

4 Performance Results

For the experiments, we set up an EtherCAT automation system with our open source controller and collected the time taken for the components of the end-to-end delay. Based on the measurements, we discuss the characteristics of the EtherCAT process delays.

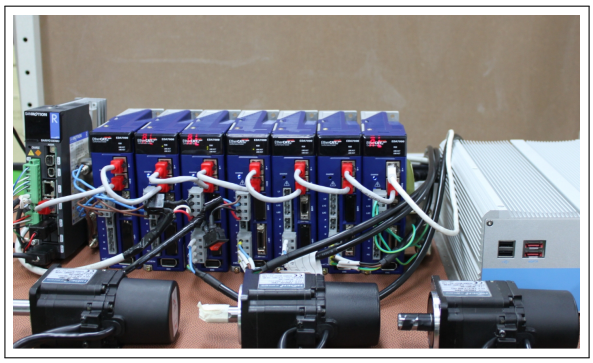


Figure 6: Experimental automation system.

Item	Description
Controller	
CPU	Intel Core 2 Duo E4500 running at 2.2 GHz (1 Core disabled)
Memory	3 GB DDR2 SDRAM
OS	Linux 2.6.37 with Xenomai 2.6.0
Network	Realtek RTL 8139D 100 Mbps Ethernet
Auto. S/W	Beremiz 1.0.3 with IgH EtherCAT Master 1.5.0
Slaves	
Product	Sanyo Denki AC Servo Drive
PDO	64 bytes for each slave (RxPDO 34 bytes, TxPDO 30 bytes)

Table 2: Specification of automation system.

4.1 Measurement of Delay Components

Table 2 summarizes the specification of our experiment system. The controller was connected to a group of commercial EtherCAT servo drives [25], each of which was configured to use an identical PDO set for communication (See Fig. 6). This set was 64 bytes in size and included all the necessary PDOs to command a drive in cyclic synchronous position or velocity mode [26].

Table 3: Measurement results for the controller delay.

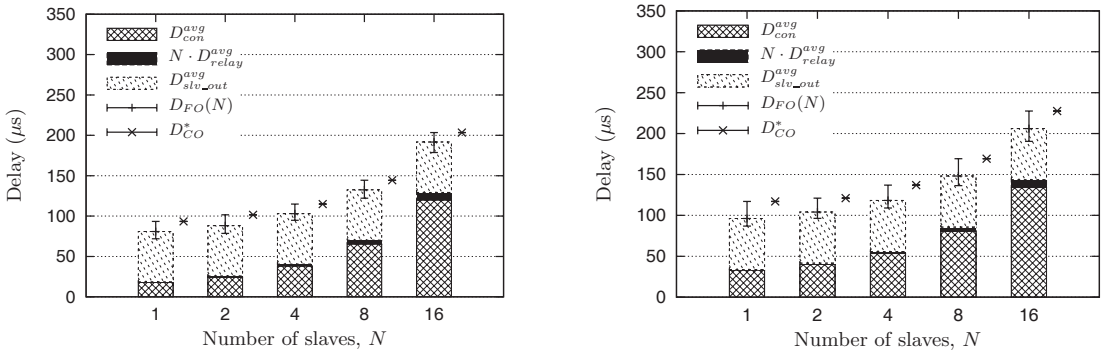
T_{cycle}	4 ms					2 ms					1 ms					0.5 ms				
N	1	2	4	8	16	1	2	4	8	16	1	2	4	8	16	1	2	4	8	16
D_{con} (μs)																				
avg	33.1	40.4	53.3	80.9	134.1	24.2	32.0	45.3	72.3	127.0	20.1	27.1	40.2	67.5	121.9	17.8	24.4	38.0	65.3	119.8
st.d	0.7	0.7	0.8	1.0	1.3	0.6	0.6	0.7	0.9	1.0	0.8	1.1	1.1	1.1	1.2	0.6	0.9	0.9	0.9	0.8
min	23.9	32.8	44.2	69.3	118.7	15.2	15.5	36.0	59.0	114.5	11.0	12.8	30.5	56.5	109.5	8.8	15.0	30.0	55.0	107.1
max	53.8	57.3	71.9	101.9	155.3	41.5	50.0	62.3	89.6	145.3	32.8	43.7	55.2	82.1	135.6	30.3	37.8	49.9	77.0	131.2
Low 95%	33.6	40.8	53.8	81.5	134.7	24.4	32.2	45.5	72.7	127.5	20.9	28.3	41.3	68.7	123.2	18.9	26.1	39.6	66.9	120.7
Low 99%	36.6	43.5	56.8	85.1	140.7	26.3	34.1	47.3	76.0	132.8	21.3	28.9	42.0	70.2	125.0	19.1	26.2	39.7	67.2	121.8
J_{con_task} (μs)																				
avg	16.4	16.5	16.1	16.4	14.9	7.5	8.1	8.0	7.7	8.0	3.9	4.0	3.8	3.8	3.8	2.0	1.9	2.0	2.0	2.0
st.d	0.5	0.6	0.6	0.8	0.9	0.4	0.5	0.5	0.6	0.7	0.3	0.4	0.4	0.5	0.6	0.3	0.3	0.3	0.5	0.6
min	6.6	6.7	6.7	2.7	-1.0	-3.2	-8.9	-3.2	-6.5	-6.0	-5.8	-9.9	-6.6	-10.1	-9.3	-8.7	-8.7	-8.2	-9.6	-10.5
max	26.1	27.7	28.4	28.8	31.0	17.8	22.2	21.1	19.5	21.5	13.0	17.5	15.1	17.3	16.5	12.5	12.6	13.2	13.0	13.1
D_{con_sw} (μs)																				
avg	3.2	3.3	3.3	3.5	4.2	3.1	3.3	3.3	3.5	4.0	3.0	3.2	3.3	3.4	3.9	3.0	3.1	3.2	3.3	3.6
st.d	0.3	0.3	0.4	0.5	0.8	0.3	0.3	0.3	0.5	0.8	0.2	0.2	0.3	0.4	0.4	0.2	0.2	0.2	0.3	0.3
min	3.0	3.2	3.2	3.3	3.7	3.0	3.1	3.1	3.2	3.7	2.9	3.0	3.1	3.2	3.6	2.8	2.9	3.0	3.1	3.5
max	14.2	11.5	12.0	18.0	17.4	11.3	11.2	13.5	16.6	17.0	10.0	12.8	13.0	14.8	16.2	12.6	12.7	11.6	12.9	13.3
D_{con_dma} (μs)																				
avg	13.5	20.6	33.9	61.0	115.0	13.5	20.6	34.0	61.0	115.0	13.1	20.0	33.1	60.3	114.2	12.9	19.5	32.9	60.0	114.1
st.d	0.1	0.1	0.1	0.3	0.3	0.1	0.1	0.1	0.3	0.4	0.5	0.8	0.8	0.7	0.8	0.4	0.7	0.7	0.6	0.5
min	13.2	14.9	30.1	57.2	110.8	10.9	17.2	30.3	56.8	110.4	7.0	16.3	28.0	56.0	109.6	9.3	15.7	29.2	55.9	109.6
max	18.1	25.6	37.5	66.5	121.4	17.3	24.5	37.6	67.7	120.7	20.2	27.3	40.4	68.0	122.6	12.6	25.8	39.6	67.8	121.4

The in-controller processing time, D_{con} was measured as the difference of TSC (Time Stamp Counter) values logged at the time instants when the control task was activated and when the EtherCAT frames had been transmitted. The time instant of frame transmission could be determined from the TX_OK interrupt generated by the NIC [27]. It should be noted that D_{con} includes the cycle time deviation, which corresponds to the difference of the measured time interval between two consecutive task activations from the intended cycle time. Table 3 summarizes the results. It shows the measured delays as we varied the cycle time, T_{cycle} and the number of slaves, N. The measurement was performed for 30 minutes for each T_{cycle}, of which collected data size amounts to 3,600,000 samples with T_{cycle}=0.5 ms, for instance.

The D_{relay} was measured using a Tektronix DPO3012 oscilloscope with a time resolution of 0.01 μs. We connected the probes to the frame interrupt pins of the EtherCAT switch hardware in two adjacent slaves, and we measured the time difference of their trigger events for the same EtherCAT frame. The measured D_{relay} turned out to be highly deterministic, having negligible deviation: The average of D_{relay} was 0.59 μs, and the maximum and minimum were 0.61 μs and 0.57 μs, respectively. We observed that T_{cycle} and N did not make any difference on D_{relay}. Since we can assume that the backward frame transmission exhibits similar timing behavior as the forward case, without loss of accuracy, we use D_{relay} for D_{return} as well. The D_{return} is hence calculated as $D_{return} = N \cdot D_{relay}$. The jitter of the global clock was measured by reading the *system time difference* register that is available in the EtherCAT switch hardware. This register maintains the difference between the local clock and the reference clock in a nanosecond resolution. The average, minimum, and maximum of the jitter were measured as -1 ns, -12 ns, and 7 ns, respectively. For D_{slv_in} and D_{slv_out}, we used the constant values provided by the manufacturer, which were 415 μs and 62.5 μs, respectively [25].

4.2 End-to-end Process Delay

Using the developed delay models together with the measurement results in the previous sections, we evaluate the end-to-end delays. Figure 7 shows the end-to-end output delays at slave N that experiences the largest delays. The graphs show the average, minimum, and maximum of the output delays along with their major delay components. It can be seen that the average



(a) T_{cycle}: 0.5 ms (b) T_{cycle}: 4 ms

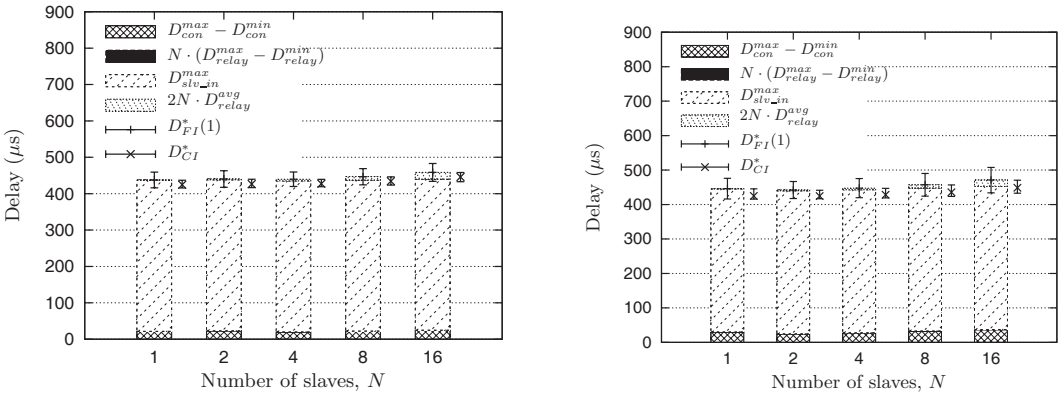
Figure 7: End-to-end output delays at slave N.

output delays increase linearly with the number of slaves. It is mainly due to the highly increased D_{con}. In contrast, the messaging delay, i.e., $N \cdot D_{relay}$ has a much lower effect on the end-to-end delay, although it grows slightly as N increases. As shown in Table 3, for a large N, the in-controller DMA time dominates D_{con} and its linearly increasing feature directly affected the overall end-to-end delays. In the figure, D_{CO}^{*} represents the minimum feasible delay in the clock-driven output scheme under the requirement of Eq. (6). We can observe that D_{CO}^{*} is affected mainly by the maximum of the delay components, i.e., D_{con}, $N \cdot D_{relay}$, and D_{slv_out}, out of which D_{con} again is the main cause of the increase in D_{CO}^{*} for growing N.

It can be seen that we have longer output delays for a larger cycle time. This is because D_{con} tends to increase as T_{cycle} gets larger. We also see that D_{con} has a larger span with a longer cycle. With D_{FO}, this resulted in a noticeably larger deviation while, with D_{CO}^{*}, it was the main reason of an increased average. This is shown clearly in Fig. 7(b) when compared with Fig. 7(a). It should be noted that D_{CO}^{*} has a negligible deviation, being less than 0.1 μs throughout the results, whereas the deviation of D_{FO} is far larger, reaching up to 30 μs and 50 μs with the cycle time of 0.5 ms and 4.0 ms, respectively. This reconfirms the strength of the clock-driven scheme, which in our platform enabled a highly synchronized actuation among servo drives, having only a few nanoseconds of standard deviation.

Based on the results, it could be estimated that the maximum N that is possible with the experimental system for a cycle time of 0.5 ms would be around 32. It is because $D_{FO}(N)$ is calculated to be 330 μs for N of 32, which leaves little time for the slave to prepare for the next cycle. This is confirmed by our experimental experience, where we failed to maintain the system in a stable state with 32 slaves. On the other hand, the experiment results with the prototype controller indicate that the open source software can be a viable automation solution. The controller could successfully operate tens of drives in position or velocity mode with a cycle time of 0.5 ms.

Figure 8 shows the end-to-end input delays on the first slave that has inevitably the largest delay among the slaves. The D_{FI}^{*}, the minimum possible input delay in the frame-driven scheme is obtained when we maximize T_{shift} using Eq. (4). Thus, from Eq. (2) we can determine the



(a) T_{cycle}: 0.5 ms (b) T_{cycle}: 4 ms

Figure 8: End-to-end input delays at slave 1.

respective average, minimum, and maximum of $D_{FI}^{*}(1)$ as

$$\begin{aligned}
D_{FI}^{*\,avg}(1) &= (D_{con}^{max} - D_{con}^{min}) + N \cdot (D_{relay}^{max} - D_{relay}^{min}) + D_{slv_in}^{max} + (N-1) \cdot D_{relay}^{avg} + D_{return}^{avg} \\
&\simeq (D_{con}^{max} - D_{con}^{min}) + D_{slv_in}^{max} + 2N \cdot D_{relay}^{avg}, \\
D_{FI}^{*\,min}(1) &= (N-1) \cdot D_{relay}^{max} + D_{slv_in}^{max} + D_{return}^{min} \\
&\simeq D_{slv_in}^{max} + 2N \cdot D_{relay}^{avg}, \\
D_{FI}^{*\,max}(1) &= 2 \cdot (D_{con}^{max} - D_{con}^{min}) + 2N \cdot D_{relay}^{max} - (N+1) \cdot D_{relay}^{min} + D_{slv_in}^{max} + D_{return}^{max} \\
&\simeq 2 \cdot (D_{con}^{max} - D_{con}^{min}) + D_{slv_in}^{max} + 2N \cdot D_{relay}^{avg}.
\end{aligned} \tag{9}$$

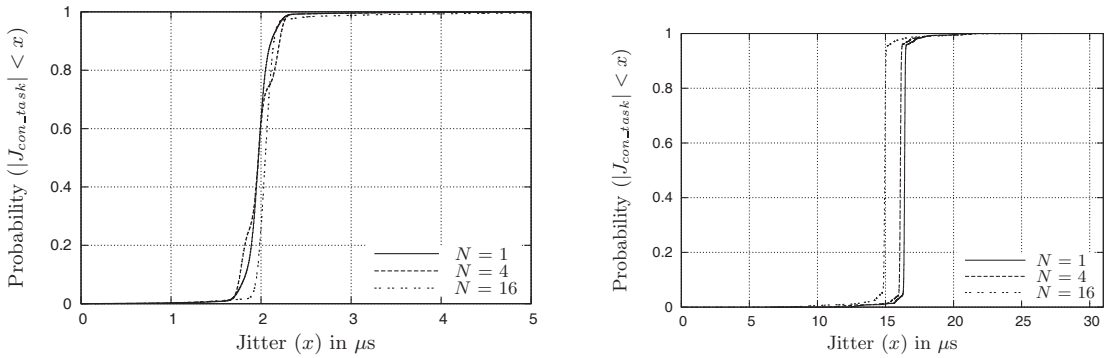
In the equations, we used the approximation of $D_{return} = N \cdot D_{relay}$ and $D_{relay}^{avg} = D_{relay}^{min} = D_{relay}^{max}$. The D_{CI}^{*} can be derived as follows. The jitter of the global clock is negligible, and hence we can write $D_{clock} + T_{shift} = D_{clock}^{max} + T_{shift}$, the maximum of which is given by Eq. (8). Thus, based on Eq. (7), D_{CI}^{*} becomes

$$\begin{aligned}
D_{CI}^{*} &= D_{con} + N \cdot D_{relay} + D_{return} - D_{con}^{min} - D_{relay}^{min} + D_{slv_in}^{max} \\
&\simeq D_{con} - D_{con}^{min} + D_{slv_in}^{max} + 2N \cdot D_{relay}^{avg}.
\end{aligned} \tag{10}$$

We see from Fig. 8 that the frame-driven end-to-end input delay is affected by the deviation of D_{con} but not by the average. This contrasts with the output cases where the mean of D_{con} has the biggest impact on the delays. The backward message relay in EtherCAT is highly deterministic and the slave-local input processing time is given as a constant; hence, it can be stated that it is essential to minimize the frame jitter caused by the controller in order to reduce D_{FI}.

It can be seen that D_{CI}^{*} outperforms D_{FI}^{*} in minimizing both the average and deviation of the delays, whereas in the output processes, the clock-driven scheme produced a larger average delay than the frame-driven. The use of clock events makes the delay dependent only on the variance of the frame that carries the input data, thereby disengaging it from the effect of the preceding frame. The frame-driven input, on the other hand, is affected by the variance of the frame twice (see Eq. (2)). In our testbed system, the D_{CI}^{*} had a deviation of -16.0 $\sim$ 37.9 μs while $D_{FI}^{*}(1)$ had -37.2 $\sim$ 73.9 μs.

Observe that the input delays are relatively less affected by the number of slaves when compared with the output delays. This is mainly due to the characteristics of D_{con}, whose deviation does not change much with varying N.



(a) T_{cycle}: 0.5 ms (b) T_{cycle}: 4 ms

Figure 9: Release jitter of control task (J_{con_task}).

As shown in the evaluation results for the end-to-end delays, the performance of the EtherCAT-based synchronized processes depends heavily on the delays by the controller. The measurement results in Table 3 show that the controller delay D_{con} tends to grow as we increase T_{cycle}. We see that this mainly comes from the increased J_{con_task}: the J_{con_task} gets larger in line with the increased T_{cycle}, whereas other in-controller delay components, D_{con_sw} and D_{con_dma} are relatively unaffected by T_{cycle}. In Fig. 9, we plot the distributions of $|J_{con\_task}|$, i.e., the absolute value of J_{con_task}. It is shown that 99% of $|J_{con\_task}|$ lies within 3.26 μs and 18.16 μs for T_{cycle} of 0.5 ms and 4 ms, respectively. It is considered that the cache pollution due to other background tasks and kernel activities, such as for debug and monitoring, is enlarging the release jitter of the real-time task.

On the other hand, the observed maximum D_{con} was quite large, ranging from 109.5% to 171.5% of the average depending on the cycle time. As can be seen in Table 3, however, the low 95% and 99% range values are very close to the average, being far lower than the maximum. It can be said that the frame release jitter of our system is statistically kept under a tolerable range although it remains to be improved in case of relatively large control cycle.

5 Conclusion and Future Work

In this paper, we have addressed the performance analysis of EtherCAT control systems in term of the end-to-end delay. On the basis of two types of cyclic events, i.e., the frame and clock events, we explain the control schemes that enable synchronized input and output in EtherCAT, and we formulate their end-to-end delays with the time for in-controller processing, message delivery, and slave-local handling. Using the developed delay model, we discuss the characteristics of EtherCAT synchronized processes for varying number of slaves and process cycle time. For an in-depth and practical evaluation, the analyses have been conducted using a real EtherCAT controller that was constructed from open source software.

The experiment results show that the controller has a crucial effect on the precision of the networked control processes. We observed that the average end-to-end delay of output processes is mainly increased by the average controller delay while the input processes are directly affected by the deviation of the controller delay. It is shown that, with a larger number of slaves, the in-controller DMA time primarily contributes to the increased average controller delay, whereas, for a longer cycle time, the task release jitter tends to get larger and increases the controller delay. It should be also noted that the use of a global clock event significantly reduces the deviation of the end-to-end delay for both input and output processes.

Aiming for a holistic delay analysis, we have dealt with the controller delay, mainly discussing its average and deviation. In our future research, we will extend our analysis by studying the probability model. Together with our already developed delay model for motor drives [4], it will enable a delay-guaranteed synchronized control system, which is planned to be used for our development of a sub-micron-level motion stage.

Acknowledgement

This research was supported in part by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (No. 2013R1A1A2004984) and in part by the National Research Foundation of Korea(NRF) grant funded by the Korea government (No. 2012015266).

Bibliography

[1] M. Felser, Real Time Ethernet: Standardization and implementations, *Proc. of Int. Symposium on Industrial Electronics*, 3766-3771, Jul. 2010.

[2] F. Benzi, G. S. Buja, and M. Felser, Communication architectures for electrical drives, *IEEE Trans. Industrial Informatics*, 1(1): 47-53, Feb. 2005.

[3] *EtherCAT Technology Group, EtherCAT: Ethernet for control automation technology.* [Online]. Available: http://www.ethercat.org.

[4] K. Kim, M. Sung, and H.-W. Jin, Design and Implementation of a Delay-Guaranteed Motor Drive for Precision Motion Control, *IEEE Trans. Industrial Informatics*, 8(2): 351-365, May 2012.

[5] J. Jasperneite, M. Schumacher, and K. Weber, Limits of increasing the performance of Industrial Ethernet protocols, *Proc. 12th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, 17-24, Sep. 2007.

[6] G. Prytz, A performance analysis of EtherCAT and PROFINET IRT, *Proc. 13th IEEE Int. Conf.on Emerging Technologies and Factory Automation*, 408-415, Sep. 2008.

[7] L. Seno, S. Vitturi, and C. Zunino, Real Time Ethernet networks evaluation using performance indicators, *Proc. 14th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, 1-8, Sept. 2009.

[8] G. Cena, I. C. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, Evaluation of EtherCAT Distributed Clock Performance, *IEEE Trans. Ind. Informatics*, 8(1): 20-29, v Feb. 2012.

[9] R. Ramesh, S. Jyothirmai, and K. Lavanya, Intelligent automation of design and manufacturing in machine tools using an open architecture motion controller, *Journal of Manufacturing Systems*, 32(1): 248-259, Jan. 2013.

[10] J. Schacht, J. Sachtleben, H. Jensen, U. Stutz, and M. Wiese, Piezo-valve controller for the gas inlet system of the fusion experiment Wendelstein 7-X, *Fusion Engineering and Design*, 87(12): 1961-1966, Dec. 2012.

[11] S. G. Robertz, R. Henriksson, K. Nilsson, A. Blomdell, and I. Tarasov, Using Real-Time Java for Industrial Robot Control, *Proc. of 5th Int. Workshop on Java Technologies for Real-Time and Embedded Systems (JRTES)*, 104-110, Sep. 2007.

[12] P. Ferrari, A. Flammini, D. Marioli, and A. Taroni, A Distributed Instrument for Performance Analysis of Real-Time Ethernet Networks, *IEEE Trans. Industrial Informatics*, 4(1): 16-25, vol. 4, Feb. 2008.

[13] M. Cereia, I. C. Bertolotti, and S. Scanzio, Performance of a Real-Time EtherCAT Master Under Linux, *IEEE Trans. Ind. Informatics*, 7(4): 679-687, Nov. 2011.

[14] Xenomai. [Online]. Available: http://www.xenomai.org.

[15] IgH EtherCAT Master for Linux. [Online]. Available: http://www.etherlab.org.

[16] E. Tisserant, L. Bessard, and M. de Sousa, An Open Source IEC 61131-3 Integrated Development Environment, *Proc. 5th IEEE Int. Conf. on Industrial Informatics*, 183-187, Jun. 2007.

[17] *IEC 61131-3: Programmable Controllers - Part 3: Programming Languages 2nd Ed.*, IEC Std., 2005.

[18] C. Gerber, S. Preusse, and H.-M. Hanisch, A Complete Framework for Controller Verification in Manufacturing, *Proc. 15th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, 1-9, Sep. 2010.

[19] O. Ljungkrantz, K. Akesson, M. Fabian, and Y. Chengyin, Formal Specification and Verification of Industrial Control Logic Components, *IEEE Trans. Automation Science and Engineering*, 7(3): 538-548, Jul. 2010.

[20] J. Farines, M. H. De Queiroz, V. G. da Rocha, A.M.M. Carpes, F. Vernadat, and X. Cregut, A model-driven engineering approach to formal verification of PLC programs, *Proc. 16th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, 1-8, Sep. 2011.

[21] L. Seno and C. Zunino, A simulation approach to a Real-Time Ethernet protocol: EtherCAT, *Proc. 13th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, 440-443, Sep. 2008.

[22] J. Robert, J.-P. Georges, E. Rondeau, and T. Divoux, Minimum Cycle Time Analysis of Ethernet-Based Real-Time Protocols, *INT J COMPUT COMMUN*, ISSN 1841-9836, 7(4): 743-757, Nov. 2012.

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

[24] *ETG.1020: EtherCAT Protocol Enhancements v.1.0.0*, EtherCAT Technology Group Std., Aug. 2011.

[25] *AC Servo Systems with EtherCAT Interface for Rotary/Linear Motor: Instruction Manual*, Sanyo Denki, Inc., Mar. 2011.

[26] *ETG.6010: Implementation Directive for CiA 402 Drive Profile v.1.0.0*, EtherCAT Technology Group Std., Feb. 2012.

[27] *Single-Chip Muilti-Function 10/100Mbps Ethernet Controller with Power Management Datasheet Rev. 1.2*, RealTek, Inc., 2005.