Game Theoretic Feedback Control for Reliability Enhancement of EtherCAT-Based Networked Systems

Liyang Li, Peijin Cong, Kun Cao, Junlong Zhou, Tongquan Wei, Member, IEEE, Mingsong Chen, Shiyan Hu, and Xiaobo Sharon Hu, Fellow, IEEE

Abstract—EtherCAT has become one of the leading real-time solutions for networked industrial systems, where a reliable communication infrastructure is needed due to highly error-prone environments. However, existing work on EtherCAT mainly focuses on clock synchronization and timeliness improvement. The reliability of EtherCAT-based networked systems has largely been ignored. In this paper, we present a proportional integral derivative (PID)-based feedback control scheme that aims at enhancing reliability of networked systems under timing and system resource constraints. Instead of retransmitting data upon error detection, we use forward error control technique based on inequality of arithmetic and geometric means to achieve the required system reliability at a low deadline miss rate of messages. We further optimize the forward error control technique and design a fast and fair error resilient mechanism by using a cooperative game. In addition to reliability enhancement, our PID-based error control scheme can also improve the stability of a system in terms of deadline miss rate in the presence of burst errors. Simulation results show that the proposed scheme can achieve reliability enhancement of up to 91% compared to benchmarking methods.

Index Terms—Embedded systems, EtherCAT, feedback control scheme, game theory, real-time, reliability.

I. INTRODUCTION

A CYBER physical system (CPS) of increasing importance in the era of industry 4.0 is composed of various physical and computing components that interact through embedded communication capabilities. The connectivity between physical entities and cyber components must ensure accurate and reliable data acquisition from the physical world and real-time information feedback from the cyber space. Networked machines are expected to work more efficiently and reliably under convergence of information and automation technology over the connectivity, which is enabled by the powerful technology of EtherCAT [1].

EtherCAT is an industrial Ethernet technology standardized by ISO [2]–[5]. It is one of the fastest real-time Ethernet networks superior to existing networks adopted in industry. Most existing industrial real-time networks are mainly designed to meet applications’ timing constraints, and are not suitable for transmitting large data [6]. For instance, controller area network (CAN) [7] is a popular real-time communication network designed to ensure the communication between micro-controlled and devices in applications without a host computer. CAN is widely used in various fields, such as robot systems, but supports only 1 Mb/s of bandwidth, which is not well-suited for systems that need to transmit large data in a short period. On the contrary, EtherCAT provides high data transmission efficiency at high speed. This is due to the fact that frames transmitted in EtherCAT networks are processed based on an “on the fly” mechanism that ensures the master and multiple slaves can exchange data in a very short time. EtherCAT frames are sent by the master to slaves cyclically. During each cycle time, every slave reads and/or writes its data from/into the EtherCAT frame and no buffering is required. Thanks to the unique way to transmit data, high speed in EtherCAT networks are achievable. For example, by using the full-duplex features of 100BASE-TX, the data rates of EtherCAT can reach more than 100 Mb/s [8]. Fig. 1 illustrates a CPS system, where multiple components are connected together by an EtherCAT cable for machine and plant control in various CPS applications.

Extensive research efforts have been made to investigate EtherCAT and its deployment in high performance industrial applications. Nguyen et al. [10] proposed the design and implementation of a closed-loop stepper motor drive control system using EtherCAT. Specifically, they presented the details on the embedded EtherCAT telegram and CiA402 motion profile, and implemented the open-loop control stepper motor based on EtherCAT. Yan et al. [11] built a micro-grid control system and used EtherCAT as a communication protocol to ensure the high communication speed for this system. The ring topology of EtherCAT is adopted to exert control over devices.
EtherCAT networks are typically deployed in harsh environments, where transmission links and processing nodes are very likely to suffer from errors. This necessitates a system design approach that takes into account reliability in addition to timeliness. Although EtherCAT has been investigated from various perspectives including its applications, synchronization schemes, and timeliness performance, the reliability of EtherCAT network has not been thoroughly investigated in the literature. The current reliability scheme of EtherCAT can be divided into backward and forward control mechanisms.

For backward control mechanism, unlike the scheme used in common wireless networks that sends the same frames continuously until the frame is correctly received, EtherCAT masters generally retransmits frames upon a failure detection or timeout. However, backward control mechanism leads to low channel utilization, and requires receivers to send acknowledgments to confirm whether data is received correctly, which increases network overheads and reduces the transmission speed. As to forward control mechanism, redundancy has been widely used to improve reliability. Maruyama and Yamada [17] presented a reliable communication architecture for EtherCAT masters by using the port redundancy. In the presented architecture, an EtherCAT master is equipped with two network interface controllers (i.e., ports). The EtherCAT master sends duplicated frames from both ports, and the frames are received at the other port. Then the master determines which frame can be used by taking a logical OR of data area of two frames. The presented approach enables highly accurate cyclic communications with high reliability. However, this technique only considers the time synchronization failure. In addition, extra hardware is required for EtherCAT masters and slaves, which incurs a significant amount of costs.

In this paper, we propose a feedback control-based scheme to enhance system reliability under the timing constraint and reliability requirement for messages as well as the resource constraint for network channels. The major contribution of this paper is summarized as follows.

1) We investigate reliability modeling of EtherCAT networks from aspects of transmission links and processing nodes, and propose a proportional integral derivative (PID)-based feedback control loop that aims at improving system reliability under the constraint of message deadline miss rate and channel utilization.

2) We improve the proposed PID-based error control scheme with respect to convergence speed and fairness by using a cooperative game and Nash bargaining solution. System reliability, message deadline miss rate, and channel utilization are also improved.

3) Extensive simulations show that the proposed control scheme can enhance system reliability by up to 91% and increase channel utilization by up to 69% when compared to benchmarking methods.

The rest of this paper is organized as follows. Section II introduces EtherCAT system architecture and models. Section III formalizes the problem studied in this paper and...
provides an overview of the proposed scheme. Section IV describes in details the proposed feedback control scheme. Section V improves of the channel allocation mechanism based on a cooperative game theory. Section VI presents the experimental results, and Section VII concludes this paper.

II. SYSTEM ARCHITECTURE AND MODELS

The focus of this paper is on reliability enhancement of an EtherCAT system in the presence of transient faults. Below, we present the various models used in this paper.

A. System Architecture

EtherCAT is one of the real-time Ethernet communication technologies and is included as a part of the ISO standards [5]. It enables a multitude of network topologies, including line, tree, ring, star, or any combination. In this paper, we adopt the ring topology as depicted in Fig. 2. The system is composed of one master and \( N \) slaves connected by the standard Ethernet cable. The master cyclically sends a standard Ethernet frame containing several subtelegrams or messages (see Fig. 3) to slaves. The frame transmits through all slaves. As the frame passes through slaves on the fly, every slave is responsible for reading or/and writing the frame. Specifically, each slave distinguishes subtelegrams addressed to itself by address parameter in the header, then takes an action specified by command parameter (read and/or write data) in the header without buffering a frame. For those subtelegrams that are not addressed to a slave, the slave only need to forward them. After the last slave in the topology transmits the frame back to the master, the next cycle starts again. We refer to the master and the slaves as computing nodes in the topology.

In fact, the scheduling for message transmission through a topology is similar to the scheduling for task execution in a CPU. That is, both processes determine the transmission/execution sequence of message/tasks that compete for shared resources. The difference between the two processes is that task execution in a CPU can be preemptive, while message transmitting through a network topology cannot be interrupted once it starts. Table I gives a brief comparison between message transmission and task execution. The two processes are compared in terms of the period, transmission/execution time, deadline, network/CPU utilization, and preempt. In this paper, we extend task scheduling methods for a CPU node to message processing/transmission in an EtherCAT network.

B. Message Model

The EtherCAT protocol is optimized for processing data. The payload of an EtherCAT frame is encapsulated in the standard IEEE 802.3 Ethernet frame and is typically composed of several subtelegrams (or messages) [24]. Fig. 3 illustrates the fields of a standard IEEE 802.3 Ethernet frame of Ethertype 0x88a4. As shown in the figure, each Ethernet frame contains 10 bytes of Ethernet header, 2 bytes of EtherCAT header, an EtherCAT data field, and 4 bytes of Ethernet tail field. The data field of EtherCAT frame may consists of multiple EtherCAT messages. Each EtherCAT message consists of 10 bytes of header, a messages data field which is up to 1486 bytes and 2 bytes of working counter. The working counter is a mechanism for EtherCAT master to monitor slaves’ behavior cyclically and synchronously. It is incremented by the slaves every time they read and/or write data into a telegram successfully. EtherCAT master can monitor the slaves in the topology by checking the working counter value contained in the periodic frames.

We consider a message set \( \Gamma \), which consists of \( M \) independent messages and is denoted by \( \Gamma: \{ \tau_1, \tau_2, \ldots, \tau_M \} \). A message in \( \Gamma \) corresponds to a subtelegram in the EtherCAT frame, and we use messages and subtelegrams interchangeably in the following sections. Real-time message \( \tau_i \) is associated with \( \{ T_i, D_i, L_i, R_G \} \), where \( T_i \) is the period of \( \tau_i \), \( D_i \) represents the deadline of the \( \tau_i \), \( L_i \) denotes the length of \( \tau_i \), and \( R_G \) is the reliability target of \( \tau_i \). The reliability requirement of each message may be different, so different reliability target can be set according to the different reliability requirement, determined by the number of different message’s backups.

C. Reliability Model

A forward error control technique [26] is adopted in this paper to provide fault-tolerance. Unlike the automatic repeat...
request (ARQ) technique that resends messages when a fault occurs [25], the forward error control technique sends and executes original messages and their backups at the same time [26]. Since messages in an EtherCAT system are likely to suffer transient faults at nodes and over links, we first discuss the soft error model for nodes, and then introduce the bit error model for links.

D. Soft Error Model for Nodes

The master of EtherCAT transmits a frame that passes through all the slave in topology. When the frame is transmitted forward, each slave recognizes the relevant commands and executes them accordingly while the frames are forwarded to the next device [28], [29]. Since there are multiple EtherCAT frames composed of several messages for different slaves, these relevant commands are usually executed for many times. For instance, Delgado et al. [30] presented a real-time motion control system using EtherCAT protocol. More specifically, they conducted an established trajectory planning algorithm presented in [31] to generate a large number of velocity commands and send them to slaves. The slaves recognize the relevant commands and executes them just like CPU execute tasks. Thus, soft errors may occur when messages are processed in slaves. Soft errors mainly result from transient faults. Poisson distribution is widely used to model the occurrences of transient faults in computing nodes [27]. Let \( \lambda_j \) be the average fault occurrence rate at computing node \( j \) for \( 0 \leq j \leq N \), then it is given by

\[
\lambda_j = \gamma_j \cdot e^{-\alpha_j f_j}
\]

where \( \gamma_j \) and \( \alpha_j \) are node dependent constants, and \( f_j \) is the operating frequency of node \( j \).

EtherCAT computing nodes process frames on the fly. Specifically, the incoming frame of a node is divided into multiple fragments of equal length, each of which is processed by the node in a unit time. A key characteristic of the EtherCAT on the fly processing is that the processing time of a fragment is equal to its forwarding time, thus, there is no need to buffer the frame. Let \( \Delta f \) denote the fragment length of a frame that a node can process at a time, and \( E_j \) denote the processing time of a fragment length message at node \( j \). \( E_j \) is calculated as \( E_j = \Delta f / f_j \). The probability that no faults occur at node \( j \) during the processing of message \( \tau_i \), denoted by \( P_{ij} \), is hence expressed as

\[
P_{ij} = \left( e^{-\lambda_j E_j} \right)^{\frac{1}{\Delta f}}
\]

where \( L_i \) is the length of the message \( \tau_i \). Since each message passes through all the \( N+1 \) nodes (including the master and slaves) in the EtherCAT topology, the probability that message \( \tau_i \) is processed and forwarded successfully at all nodes, denoted by \( P_{i, \text{nodes}} \), is calculated as

\[
P_{i, \text{nodes}} = \prod_{j=0}^{N} P_{ij} = e^{-\frac{L_i}{\Delta f} \sum_{j=0}^{N} E_j \lambda_j}.
\]

E. Bit Error Model for Links

In digital transmission, bit errors are induced by noise, interference, distortion, or bit synchronization errors over links. Let \( t_i \) be the transmission time of message \( \tau_i \) through all links of the topology. Then the probability that message \( \tau_i \) is successfully transmitted over links, which is denoted by \( P_{i, \text{links}} \), can be modeled as [32]

\[
P_{i, \text{links}} = e^{-\theta t_i}
\]

where \( \theta \) is the constant bit error rate. Let \( P_i \) be the probability that message \( \tau_i \) is successfully processed and transmitted in a given EtherCAT system when no messages are replicated for tolerance. \( P_i \) is obtained by

\[
P_i = P_{i, \text{nodes}} \cdot P_{i, \text{links}} = e^{-\theta t_i} e^{-\frac{L_i}{\Delta f} \sum_{j=0}^{N} E_j \lambda_j}.
\]

The reliability of a message is defined as the probability that the message issued by the master is successfully processed, and routed back to the master in the presence of errors. Assume that \( k_i \) backups are used for message \( \tau_i \) to achieve the required reliability. The reliability, denoted by \( R_i(k_i) \), is expressed as

\[
R_i(k_i) = 1 - (1 - P_i)^{k_i+1}.
\]

The reliability of the system of \( M \) messages, defined as the product of the reliability of individual messages and denoted by \( R_{\text{sys}} \), is thus given by

\[
R_{\text{sys}} = \prod_{i=1}^{M} R_i(k_i).
\]

III. PROBLEM DEFINITION AND OVERVIEW OF THE PROPOSED SCHEME

Our goal is to design a fault-tolerance message scheduling scheme in order to enhance the overall reliability of the EtherCAT system (i.e., \( R_{\text{sys}} \)). We first formulate in this section the problem to be tackled, followed by an overview of the proposed control scheme. We assume a scenario that messages transmitted in an EtherCAT system are periodic and independent, and the characteristics of the messages are known a priori. The forward error control technique is used in the EtherCAT system to achieve fault tolerance.

A. Problem Definition

Given an EtherCAT system of a ring topology that contains \( N+1 \) nodes (one master and \( N \) slaves), and a set of \( M \) messages, find the number of backups for each message such that the system reliability, \( R_{\text{sys}} \), is maximized under the timing and message reliability constraint. That is

\[
\text{Maximize : } R_{\text{sys}}
\]

Subject to : MissRate \( \leq \varepsilon \)

\[
R_i \geq RG_i
\]

\[
\text{NET} \leq 1
\]

where MissRate is the deadline miss rate of messages during one sampling period of the proposed controller, \( \varepsilon \) is a positive constant that indicates the threshold for deadline miss.
controller, MB controller, and EDF scheduler. In this section, we present in details the working mechanism of backups of messages in the ACCEPTED queue. Finally, the controller cannot accommodate all of the trolling message flow into the ACCEPTED queue. If the MA controller decides whether it can be accepted into the ACCEPTED queue, the MA and MB controller sequentially to accommodate the actually achieved throughput [33]. The main controller periodically samples the current deadline miss rate, the PID controller only considers messages that have entered the EtherCAT system. Messages rejected from entering the system are not taken into account when sampling. The PID controller then computes the control variable ΔNET in terms of requested channel utilization using the control equation given by [34]

\[
\Delta NET = - C_P \cdot \text{err}(t) - C_I \cdot \sum_{iW} \text{err}(t) - C_D \cdot \frac{\text{err}(t) - \text{err}(t-DW)}{DW}
\]  

(8)

where err(t) is the difference between the threshold for system deadline miss rate and the current system deadline miss rate, that is, \( \text{err}(t) = \varepsilon - \text{MissRate} \). The \( C_P \), \( C_I \), and \( C_D \) are coefficients of the PID controller.

Algorithm 1: PID Control Algorithm

Input: Threshold \( \varepsilon \) for deadline miss rate.
Output: Total channel utilization to be accommodated, \( \Delta NET \).

```
1 do
2 PID controller samples messages to derive MissRate;
3 Calculate ΔNET using Equation (8);
4 return ΔNET;
5 while (MissRate > \varepsilon);
6 return ΔNET;
```

A. PID Controller

PID controller is a control loop feedback mechanism that improves robustness of a control process against external disturbances. The operation of our PID controller is outlined in Algorithm 1. Taking as input the threshold \( \varepsilon \) for deadline miss rate, the PID controller periodically samples messages to derive the process variable MissRate. Note that the PID controller only considers messages that have entered the EtherCAT system. Messages rejected from entering the system are not taken into account when sampling. The PID controller then computes the control variable ΔNET in terms of requested channel utilization using the control equation given by [34]

\[
\Delta NET = - C_P \cdot \text{err}(t) - C_I \cdot \sum_{iW} \text{err}(t) - C_D \cdot \frac{\text{err}(t) - \text{err}(t-DW)}{DW}
\]  

(8)

where err(t) is the difference between the threshold for system deadline miss rate and the current system deadline miss rate, that is, \( \text{err}(t) = \varepsilon - \text{MissRate} \). The \( C_P \), \( C_I \), and \( C_D \) are coefficients of the PID controller. IW is the time window for the last IW time units over which the errors are summed. Similarly, DW is the time window for the last DW time units over which the derivative error is calculated as \( (\text{err}(t) - \text{err}(t-DW))/DW \).

The PID controller returns the computed ΔNET to the main controller, which in turn sends ΔNET to the MA and MB controller for allocation. When ΔNET > 0, the channel utilization should be increased, hence more messages and/or message backups are admitted into the system to allocate the ΔNET. On the contrary, when ΔNET < 0, the channel utilization should be decreased, hence some messages and/or message backups will be dismissed from the system to distribute the ΔNET. The procedure repeats until MissRate ≤ \( \varepsilon \).

B. Message Access Controller

The MA controller is responsible for controlling the admission of original messages into the EtherCAT system. When a new message \( \tau_i \) is submitted to the WAITING queue, the MA controller decides whether it can be accepted into the system. Messages in the WAITING queue are sorted according to the EDF scheduling policy. Let ΔNET\( ^a \) be the portion of the channel utilization ΔNET that can be allocated by the MA controller. As shown in Algorithm 2, the MA controller takes ΔNET as input and returns ΔNET\( ^a \) to the main controller. Given ΔNET > 0, the MA controller admits message \( \tau_i \) if the condition \( \Delta NET^a = \Delta NET_{(e+1)} > 0 \) holds (lines 3–18). e denotes the minimum number of backups to

IV. FEEDBACK CONTROL SCHEME FOR RELIABILITY ENHANCEMENT

In this section, we present in details the working mechanism of the proposed controller that integrates a PID controller, MA controller, MB controller, and EDF scheduler.
Algorithm 2: MA Control Algorithm

Input: \(\Delta NET; M\), the number of messages in ACCEPTED queue; \(W\), the number of messages in WAITING queue.

Output: The portion of \(\Delta NET\) that can be accommodated by the MA controller.

// initialization
1 \(\Delta NET_{\text{W}} = 0;\)
2 if \(\Delta NET > 0\) then
3 \hspace{1em} Sort messages in WAITING queue according to the EDF policy;
4 \hspace{1em} for \(i = 1; i \leq W; i++\) do
5 \hspace{2em} if \(\Delta NET_{\text{W}} \geq \Delta NET\) then
6 \hspace{3em} break;
7 \hspace{1em} end
8 \hspace{1em} Calculate minimum number of backups that can meet \(\tau_i\)’s reliability target (\(c\));
9 \hspace{1em} if \(\Delta NET_{\text{W}} - \Delta NET_{\text{W}+(e+1)} \geq 0\) then
10 \hspace{2em} // \(\Delta NET_{\text{W}}\) is given by Equation (9)
11 \hspace{2em} \(\Delta NET_{\text{W}} = \Delta NET_{\text{W}} + \Delta NET_{\text{W}+(e+1)};\)
12 \hspace{1em} Dequeue head message from WAITING queue;
13 \hspace{1em} Enqueue the message to ACCEPTED queue;
14 \hspace{1em} Update the number of message \(\tau_i\)’s backup;
15 \hspace{1em} \(M + +;\)
16 \hspace{1em} end
17 \hspace{1em} end
18 else
19 \hspace{1em} Messages remain in WAITING queue;
20 end
21 return \(\Delta NET_{\text{W}}\), the portion of \(\Delta NET\) that can be accommodated by the MA controller; the number of messages’ backup;

Determine the number \(i\) of messages for \(R_i < R_{\text{avg}}\) in the queue; Sort the \(m\) messages in the queue in the ascending order of reliability, \(R_i\);

while \(\Delta NET > 0\) do
2 \hspace{1em} Increment the number of message \(\tau_i\)’s backup by 1;
3 \hspace{1em} if \(i = m + 1\) then
4 \hspace{2em} \(i = 1;\)
5 \hspace{2em} Recalculate \(R_{\text{avg}}\) and update \(m;\)
6 \hspace{1em} end
7 \hspace{1em} end
8 else
9 \hspace{1em} Derive mean \(R_{\text{avg}}\) of message reliabilities in ACCEPTED queue;
10 \hspace{1em} Derive the number \(m\) of messages for \(R_i > R_{\text{avg}}\) in the queue;
11 \hspace{1em} Sort the \(m\) messages in the queue in ascending order of reliability, \(R_i;\)
12 while \(\Delta NET < 0\) do
13 \hspace{1em} Decrement the number of message \(\tau_i\)’s backup by 1;
14 \hspace{1em} // \(\Delta NET_{\text{b}} = \Delta NET_{\text{b}} - \Delta NET_{\text{b}+1};\)
15 \hspace{1em} if \(i = 0\) then
16 \hspace{2em} \(i = m;\)
17 \hspace{2em} Recalculate \(R_{\text{avg}}\) and update \(m;\)
18 \hspace{1em} end
19 return the number of messages’ backup;

The MB controller is designed based on the above principle to enhance the system reliability. It first calculates the average message reliability in the ACCEPTED queue and selects messages with reliability below/above the average. It then iteratively increases/decreases the number of backups of the selected messages to improve system reliability \(R_{\text{sys}}\). As shown in Algorithm 3, the MB controller takes \(\Delta NET_{\text{b}}\) as input. \(\Delta NET_{\text{b}}\) is the portion of the channel utilization that can be allocated by the MB controller, which is calculated by the main controller.

Algorithm 3 works as follows. For the case of \(\Delta NET_{\text{b}} > 0\), the algorithm calculates the average reliability of messages in the ACCEPTED queue (denoted by \(R_{\text{avg}}\), picks the \(m\) messages for \(R_i < R_{\text{avg}}\) and \(1 \leq i \leq m\), and sorts the \(m\) messages in the queue in the ascending order of reliability (lines 2–5). When all of \(\Delta NET_{\text{b}}\) is allocated by MB controller (i.e., \(\Delta NET_{\text{b}} > 0\)), the algorithm increments the number of message \(\tau_i\)’s backup by 1, updates \(\Delta NET_{\text{b}}\) to \(\Delta NET_{\text{b}} + \Delta NET_{\text{b}+1}\) and increments \(i\) by 1. If all the \(m\) messages have been updated by increasing a backup and \(\Delta NET_{\text{b}}\) is not used up yet, the algorithm resets \(i = 1\), recalculates the average reliability, updates the value of \(m\) and repeats the accommodation process (lines 7–14). Assume that \(\tau_i\) is the message selected during the accommodation process. Let \(\text{NET}_{\text{b}}\) be the incurred channel utilization due to the admission of message \(\tau_i\) and its \(c\).
copies, then \( \text{NET}_{ic} \) is given by
\[
\text{NET}_{ic} = \frac{c \left( \sum_{j=0}^{N} E_j + t_i \right)}{T_i}
\]
(9)
where \( t_i \) is the total time needed to transmit a message over all the links of the ring topology, \( T_i \) is the period of message \( \tau_i \), \( E_j \) is the processing time of unit length message at node \( j \), and \( N \) is the number of nodes in the system. \( \text{NET}_{iA} \) in line 9 can be easily derived using (9).

In the case of \( \Delta \text{NET}^a < 0 \), the algorithm works the same as in the case of \( \Delta \text{NET} > 0 \) except that backups of messages satisfying \( R_i > R_{\text{avg}} \) for \( 1 \leq i \leq m \) are iteratively dismissed from the system (lines 16–29).

The control scheme of the MB controller above simply considers the average of message reliability (\( R_{\text{avg}} \)). However, messages may have different reliability targets. It is expected that the MB controller can allocate different channel resources for messages according to their respective reliability targets. Further, incrementally increasing the backup of messages during each iteration takes a long time for the EtherCAT system to converge. Note that the time complexity of Algorithm 3 is at least \( O(M) \). Thus, we propose an optimization strategy based on the cooperative game theory and Nash bargaining solution for the MB controller. The time complexity of the game-based optimization strategy is \( O(M) \), and is introduced in Section V.

### D. EDF Scheduler and Main Controller
EDF is a dynamic scheduling algorithm that dictates the arrangement of messages in a priority queue [42]. When the EDF scheduler is called, the EDF scheduler first finds the task with the earliest deadline and then executes the task [43]. In our feedback control scheme, once a message is admitted into the system by the MA controller and the number of its backups is dispatched to the EtherCAT master for processing. Thus, the EDF scheduler is scheduled every PID controller’s sampling period, and this scheduling frequency has no relation to the period of messages. The EDF scheduler dynamically arranges the execution order of messages in the ACCEPTED queue. The message with the earliest deadline is selected by the EDF scheduler and is dispatched to the EtherCAT master for processing.

The main control algorithm integrates the PID, MA, and MB controllers to form a closed loop that effectively improves the robustness of the control process against external disturbances. It is called periodically for the enhancement of system reliability, the period of which is determined by the minimum sampling interval.

Algorithm 4 describes the operation of the main control algorithm. It takes as input the message set \( \Gamma \), the total channel utilization returned by the PID controller for allocation (\( \Delta \text{NET}^a \)), the portion of channel utilization allocated by the MA controller (\( \Delta \text{NET}^b \)), and the updated numbers of all the messages’ backups. If this is the first time the algorithm is called, it first determines the number of backups for each message \( \tau_i \) based on the reliability target (\( R_{G_i} \)) and (6) (lines 2–6). It then calls the PID controller to calculate the deadline miss rate \( \text{MissRate} \) and \( \Delta \text{NET} \) (line 7), calls the MA algorithm to derive \( \Delta \text{NET}^b \) (line 8) and calculates \( \Delta \text{NET}^b \) (lines 9). Afterward the MB algorithm is called to allocate \( \Delta \text{NET}^b \) if \( \Delta \text{NET}^b \neq 0 \) and update messages’ backups (lines 10–12). In the end, the EDF scheduler is called (line 13) to dispatch messages to the master for processing.

### V. GAME THEORY-BASED REFINEMENT OF MESSAGE BACKUP CONTROL
Due to the slow convergence and unfairness of the MB control mechanism described in Section IV, we propose a game theoretic approach to refining the channel allocation process for further reliability enhancement. In this section, we first introduce the concepts of cooperative game and Nash bargaining, then model the channel allocation game among multiple messages, and finally refine our MB control mechanism based on a game theory.

#### A. Cooperative Game and Nash Bargaining
A cooperative game consists of \( M \) players, a performance function \( f \), and an initial agreement point \( RG \). The \( M \) players are represented by a 3-tuple of nonempty, closed, and convex set \( \{ \kappa, \mathfrak{S}, \mathfrak{R} \} \), where \( \kappa \) is the set of strategies, \( \mathfrak{S} \) denotes the states of the assigned resource, and \( \mathfrak{R} \) gives the states of the \( M \) players. The performance function \( f \) maps \( \kappa \) to \( \mathfrak{R} \). The vector \( RG = (RG_1, RG_2, \ldots, RG_M) \) is defined as the initial agreement point, where \( RG_i \) indicates the minimum value of performance function \( f \). \( RG_i \) is the minimal performance required for the player \( i \) to enter the game without any cooperation. The above cooperative game is in general resolved by Nash bargaining, and the generated solutions to the cooperative game are called Nash bargaining solutions.

Nash bargaining solution (NBS solution) [38] is defined as follows. A mapping \( f : (\kappa)(RG) \rightarrow \mathfrak{R} \) is an NBS solution if \( f(\kappa)(RG) \in \mathfrak{R} \), where \( \kappa \) is the set of strategies, i.e., the set of possible bargaining agreements that \( M \) players may reach. \( RG \) is the set of initial agreement point. \( \mathfrak{R} \) represents the set of
of players’ current states and $f(\kappa | RG)$ is Pareto optimal and satisfies the fairness axioms [39].

In the modeling of our channel allocation, $\kappa = (\Delta k_1, \Delta k_2, \ldots, \Delta k_M)$ denotes the set of possible bargaining agreements that $M$ messages may reach. $\mathcal{R}$ represents the set of $M$ messages’ current reliability. The performance function $f$ maps allocation strategies (i.e., $\kappa$) to messages’ current reliability (i.e., $\mathcal{R}$). The initial agreement point $RG$ is the set of minimum guarantee (i.e., the reliability target of messages) that system must satisfy. We assume that message $\tau_i$ ($1 \leq i \leq M$) involved in the cooperative game can achieve its initial performance requirement $(RG_i)$ without any cooperation. Thus, we have $\mathcal{R} = [R_i | R_i \geq RG_i]$. Under these definitions and the assumption, we can derive an NBS solution with its strategy $\Delta k_i \in \kappa$, which is obtained by solving the following optimization problem [37]:

Problem : $\max \prod_{i=1}^{M} (f(\Delta k_i | RG_i) - RG_i)$ \hspace{1cm} (10)

In the NBS solution above, multiple players (typically more than two) enter the cooperative game with their corresponding initial performance requirements in $RG$ satisfied. The messages (players) cooperate in the game to achieve a win-win solution, which enhances the performance given in (10) and leads to a relative fairness among all messages. Using the logarithm of the objective function, an equivalent problem can be derived as

Problem’ : $\max \sum_{i=1}^{M} \ln f(\Delta k_i | RG_i) - RG_i)$ \hspace{1cm} (11)

where Problem’ is a convex optimization problem and has a unique solution [40], [41]. The unique solution to the problem is the NBS solution.

B. Channel Allocation Refinement for MB Controller

We consider a cooperative game in which $M$ messages are competing for the shared available channel resource ($\Delta NET$). In the context of our channel allocation, we define the performance function $f$ that maps the change in the number of message $\tau_i$’s backups (i.e., $\Delta k_i$) to the reliability of message $\tau_i$ (i.e., $R_i$). The performance function is formulated as

\[ f(\Delta k_i | RG_i) = \left(1 - \left(1 - e^{-\theta_i} \sum_{j=0}^{N} E_j^{\lambda_j} \right) k_i^{\Delta k_i} + 1 \right) \] \hspace{1cm} (12)

where $k_i$ denotes the original number of message $\tau_i$’s backups and $\Delta k_i$ represents the change in the number of $\tau_i$’s backups. $t_i$, $\sum_{j=0}^{N} E_j$, and $T_i$ denotes the time that $\tau_i$ transmits over links in EtherCAT topology, the time $\tau_i$ processed in $N + 1$ computing nodes ($N$ slaves plus 1 master), and $\tau_i$’s period, respectively. Suppose that each message $\tau_i$ has an initial reliability requirement $RG_i$, with $RG_i$ we can derive the minimal number of $\tau_i$’s backups that need to be guaranteed without any cooperation. We also assume that the $M$ messages can achieve the same or better performance (i.e., $R_i \geq RG_i$).

Our goal is to enhance system reliability and improve the reliability of individual messages under the messages’ reliability requirements. The problem can be described as follows. Given the shared available channel resource ($\Delta NET^b$) and reliability requirements ($RG$), $M$ messages cooperate in the game to obtain a win-win solution described by $(\Delta k_1, \Delta k_2, \ldots, \Delta k_M)$. Therefore, this optimization problem can be formulated as

Maximize $\prod_{i=1}^{M} \left(1 - \left(1 - e^{-\theta_i} \sum_{j=0}^{N} E_j^{\lambda_j} \right) k_i^{\Delta k_i} + 1 \right) - RG_i$ \hspace{1cm} (13)

Subject to $\sum_{i=1}^{M} \left( t_i + \sum_{j=0}^{N} E_j^{\lambda_j} \Delta k_i / T_i \right) \leq \Delta NET^b$ \hspace{1cm} (14)

where (14) indicates all of the available channel can be allocated to enhance reliability.

In the above formulation, $\max \prod_{i=1}^{M} f(\Delta k_i | RG_i) - RG_i$ is selected as the objective rather than $\max \sum_{i=1}^{M} f(\Delta k_i | RG_i) - RG_i$. This is because the former formulation not only demonstrates the capability of maximizing system reliability, but also shows the expectation of the $M$ messages for maximizing their respective reliability. According to the analysis given in the end of Section V, the objective in (13) is equivalent to

$\max \sum_{i=1}^{M} \ln (1 - \left(1 - e^{-\theta_i} \sum_{j=0}^{N} E_j^{\lambda_j} \right) k_i^{\Delta k_i} + 1 - RG_i)$ \hspace{1cm} (15)

Equation (13) is an optimization problem that attempts to maximize system reliability under the constraint of channel resources [i.e., (14)]. Since Lagrange multiplier is powerful for solving this type of problem with low computation complexity, we adopt it to obtain the best solution to our problem. The Lagrangian of this problem is expressed as

$\tau(\Delta k_i, \alpha) = -\sum_{i=1}^{M} \ln \left(1 - \left(1 - e^{-\theta_i} \sum_{j=0}^{N} E_j^{\lambda_j} \right) k_i^{\Delta k_i} + 1 - RG_i \right) - \alpha \left( \sum_{i=1}^{M} \left( t_i + \sum_{j=0}^{N} E_j^{\lambda_j} \Delta k_i / T_i \right) - \Delta NET^b \right)$ \hspace{1cm} (16)

where $\alpha \in \mathbb{R}$, and it is the Lagrange multiplier associated with the constraints given in (14).

It is clear that the optimal solution is derived when the derivative of $\tau(\Delta k_i, \alpha)$ with respect to $\Delta k_i$ equals zero. In this case, the expression

$\nabla \tau(\Delta k_i, \alpha) = 0 \Leftrightarrow \nabla t_i + \alpha \nabla \tau_i = 0$ \hspace{1cm} (17)

and the Karush–Kuhn–Tucker conditions [40] holds. In (17),

$\nabla t_i = (1 - e^{-\theta_i} \sum_{j=0}^{N} E_j^{\lambda_j} k_i^{\Delta k_i} + 1 \left( \ln (1 - e^{-\theta_i} \sum_{j=0}^{N} E_j^{\lambda_j}) / (1 - \left(1 - e^{-\theta_i} \sum_{j=0}^{N} E_j^{\lambda_j} \right) k_i^{\Delta k_i} + 1) \right) / (1 - \left(1 - e^{-\theta_i} \sum_{j=0}^{N} E_j^{\lambda_j} \right) k_i^{\Delta k_i} + 1) \right)$

and the best solution to the optimization problem can be derived from (17), and it can be given by

$\Delta k_i = (\ln (1 - RG_i) \cdot \nu - \ln (1 - \nu) \cdot \omega) / \omega - k_i - 1$ \hspace{1cm} (18)

where $\nu$ denotes $(t_i + \sum_{j=0}^{N} E_j^{\lambda_j} \cdot \nu) / T_i$ and $\omega$ represents $\ln (1 - e^{-\theta_i} \sum_{j=0}^{N} E_j^{\lambda_j})$. 


Algorithm 5: Refined MB Control Algorithm

Input: The portion of accommodated channel utilization ($\Delta NET^b$).
Output: The number of messages backup.

for $i = 1$ to $M$ do
  Calculate the number of backups of message $t_i$ that need to be changed ($\Delta k_i$) using Equation (18);
  if $\Delta NET^b > 0$ then
    Increase the number of message $t_i$'s backup by $\Delta k_i$
  else
    Decrease the number of message $t_i$'s backup by $\Delta k_i$
  end
end
return the number of messages' backup;

As indicated in (18), we can improve the original MB controller algorithm (Algorithm 3) by using the method above, the refined MB control algorithm is shown as follows.

Algorithm 5 works as follows. For each message $t_i$ in the ACCEPTED queue, it calculates $\Delta k_i$, the change in the number of message $t_i$'s backups ($\Delta k_i$), by using (18) (line 3).

In the case of $\Delta NET^b > 0$, the algorithm increases the number of message $t_i$'s backup by $\Delta k_i$ (lines 4–6). In the case of $\Delta NET^b < 0$, it decreases the number of message $t_i$'s backup by $\Delta k_i$ (lines 7–9). The time complexity of Algorithm 5 is $O(M)$.

VI. SIMULATION-BASED EVALUATION

Extensive simulation-based experiments have been conducted to validate the effectiveness of the proposed scheme.

In this section, we first describe simulation settings in detail and then verify the effectiveness of the refined channel allocation mechanism proposed in Section V. To evaluate the performance of the proposed scheme, we compare the original feedback control scheme with two benchmarking methods in terms of deadline miss rate, channel utilization, and system reliability.

Finally, we compare the refined channel allocation mechanism with the original one in order to validate the effectiveness of the refined mechanism.

A. Simulation Settings

The simulations are conducted on a machine equipped with 2.4 GHz Intel i7 quad-core processor and 8 GB DDR4 memory, and running a Windows version of MATLAB_x64 and OMNeT++. OMNeT++ is an extensible, modular, and component-based C++ simulation library and framework, primarily for building network simulators [36]. We use OMNET++ to simulate the EtherCAT ring topology and MATLAB_x64 to simulate the message scheduling process of the proposed feedback scheme. Two different scales of EtherCAT ring topologies are considered in the simulation for a better comparison study. The first topology has 1 master and 10 slaves, while the second topology contains 1 master and 20 slaves. We use three message sets, each of which contains 5, 10, and 20 messages, respectively.

Similar to the work presented in [34], coefficients $C_P$, $C_I$, and $C_D$ of the PID controller are set to 0.5, 0.005, and 0.1, respectively. The time window $IW$ and $DW$ are set to 100 and 1 units of time, respectively. The PID controller samples the network once every 500 time units. The values of $RG$ reflect the difference in reliability targets of messages. We randomly generate reliability target $RG_i$ for message $t_i$ in the interval of (0, 1). The period $T_i$ (in time units), deadline $D_i$ (in time units), and length $L_i$ (in bytes) of message $t_i$ are randomly generated in the interval of (200, 500), (200, 800), and (12, 1498), respectively. The fragment length a node can process on the fly at a time ($\Delta f$) is set to 4 Bytes [20].

In order to prove the effectiveness of our proposed methods, we compare the proposed methods with three benchmarking methods in various aspects. The three benchmarking methods are referred to as no-backup (NBK), ARQ, and allocating channel equally (ACE), respectively. The first method, referred to as NBK, sends messages with no backups even if errors occur, that is, no error control technique is taken for reliability enhancement. The second method is ARQ, which also sends messages with no backup, however, it resends messages when a fault occurs. The third one is called ACE. This method assigns available channel utilization equally to each message.

For the sake of easy presentation, our proposed reliability enhancement methods are referred to as Proposed_Mean and Proposed_Game, respectively. Our Proposed_Mean method derives the initial number of each message’s backup according to our reliability model, then allocates channel utilization based on inequality of arithmetic and geometric means, and obtains the ultimate number of backups for each message through multiple iterations. Our Proposed_Game method refines the Proposed_Mean method in the way that channel utilization is allocated. It distributes channel utilization to each message based on the game theory and Nash bargaining solution, and calculates the final number of message’s backups by using Lagrange multiplier.

B. Proposed_Mean Versus Benchmarking NBK and ARQ

We compare the Proposed_Mean method with the NBK method and the ARQ method in terms of message deadline miss rate, channel utilization, and system reliability, respectively. Fig. 5 shows the deadline miss rate of the three methods. It can be seen from the figure that the MissRate of the Proposed_Mean method is higher than that of the NBK method in two different topologies. This is primarily due to the fact that NBK method does not use any backup for reliability enhancement. On the contrary, the MissRate of the Proposed_Mean method is lower than that of the ARQ method.

In terms of stability, the MissRate variance of ARQ method is 24, while the MissRate variance of NBK method and the Proposed_Mean method is 1.12 and 0.64, respectively. This is because the ARQ method does not send message backups until an error occurs, resulting in burst transmission of message backups, thus an increased deadline miss rate.

Fig. 6 shows the channel utilization of the three methods. Compared with the NBK and the ARQ method, the Proposed_Mean method consumes up to 52% more channel utilization in topology with 10 slaves and 63% in topology with 20 slaves. This is because the Proposed_Mean method sends backups together with messages while the NBK and the ARQ method do not. Results also show that the variance of NBK, ARQ and proposed method in channel utilization is...
2.04, 28.97, and 1.69, respectively. Therefore, ARQ method is not suitable for systems of high stability requirements.

Fig. 7 shows that the Proposed_Mean method can effectively enhance system reliability by up to 74% when compared to the NBK and the ARQ method in 10 slave-topology and 79% in 20 slave-topology. The figure also shows that the system reliability slightly gets lower when the number of slaves, i.e., the system complexity, increases.

C. Proposed_Game Method Versus Proposed_Mean Method

Before we compare the Proposed_Game method with the Proposed_Mean method, we first verify the effectiveness of proposed schemes. We set channel utilization to be allocated (i.e., $\Delta$NET) to 15%, 20%, 25%, and 30%, respectively. Then we compare the Proposed_Mean method and the Proposed_Game method with the ACE method in terms of system reliability.

As shown in Fig. 8, the proposed schemes outperform the ACE method in terms of system reliability improvement, and the Proposed_Game is more effective in allocating channel utilization to enhance the system reliability. The reason is that the Proposed_Game method can allocate Pareto optimal channel resources to messages, while the ACE method just equally allocates available channel utilization to messages, which makes limited channel resources unavailable to messages of greater impact on system reliability.

Then we compare the Proposed_Game method and Proposed_Mean method with respect to message deadline miss rate, channel utilization and system reliability. Fig. 9 shows the deadline miss rate of the Proposed_Mean method and the Proposed_Game method in two different topologies. It can be seen from the figure that compared to the Proposed_Mean, the Proposed_Game can reduce the message deadline miss rate by up to 11%. This is primarily due to the fact that the Proposed_Game method has better control over messages’ backups, thus, fewer messages miss their deadlines.

Fig. 10 shows the channel utilization of the Proposed_Mean method and the Proposed_Game method in two different topologies. As compared to the Proposed_Mean, the Proposed_Game method consumes up to 9.2% more channel utilization in topology with 10 slaves and 5.3% in topology with 20 slaves. This is because the Proposed_Game method has better control over messages’ backups and thus make better use of the available channel resources. Fig. 11 shows that when compared to the Proposed_Mean method, the Proposed_Game method can effectively enhance system reliability by up to 16% for 10 slave-topology and 9.1% for 20 slave-topology. As compared to the benchmarking methods, the Proposed_Game method can improve system reliability by up to 91% in topology with 10 slaves and 86% in topology with 20 slaves.

The reason is that the Proposed_Game method allocates Pareto
optimal channel resources to messages of greater impact on system reliability.

We also compare the execution time of the two proposed schemes. As shown in Fig. 12, the execution time of the Proposed_Mean method is up to 2.1 times of the Proposed_Game method for 10 slave-topology, and 1.8 times for 20 slave-topology. According to Section IV and V, the time complexity of Algorithm 5 is $O(M)$ and that of Algorithm 3 in the best case is $O(M)$. This is because the Proposed_Mean method needs to derive the number of messages’ backup in multiple iterations, while the Proposed_Game can quickly obtain a Pareto optimal result by using the cooperative game.

VII. CONCLUSION

In this paper, we aim to enhance EtherCAT system reliability while meeting the timing constraint of real-time messages and resource constraint of the network channel. Our Proposed_Mean scheme adopts a PID-based feedback control mechanism that improves system reliability by adjusting the number of messages and their backups admitted into the system using inequality of arithmetic and geometric means method. In addition, in order to allocate channel resource faster and fairer, we design the Proposed_Game method, which optimize the Proposed_Mean method by using cooperative game theory and Nash bargaining solution. Simulation results show that the Proposed_Mean and Proposed_Game method improves system reliability by up to 79% and 91%, respectively, when compared to two benchmarking schemes. In addition, the Proposed_Game takes only about half the time of Proposed_Mean to derive the ultimate number of message backups.

REFERENCES

Junlong Zhou is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology, East China Normal University, Shanghai, China. He was a Research Visitor with the University of Notre Dame, Notre Dame, IN, USA. His current research interests include real-time embedded systems, cyber physical systems, and cloud computing. He has published a dozen of papers in the above areas.

Mr. Zhou was a recipient of the Reviewer Award from the Journal of Circuits, Systems, and Computers in 2016. He is an Active Reviewer of several international journals, including the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, the Journal of Systems and Software, and the Journal of Circuits, Systems, and Computers.

Leonard John Golden, received the B.S. degree in electrical engineering from the University of Florida, Gainesville, FL, USA, in 2007. He is currently pursuing the Ph.D. degree in computer science from the University of Florida. His current research interests include cyber-physical systems, cloud computing, and formal verification of cyber-physical systems.

Mingsong Chen received the B.S. and M.E. degrees from the Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2003 and 2006, respectively, and the Ph.D. degree in computer engineering from the University of Florida, Gainesville, FL, USA, in 2010. He is currently a Full Professor with the Department of Embedded Software and Systems, East China Normal University, Shanghai, China. His current research interests include design automation of cyber-physical systems, formal verification of cloud computing, and Internet of Things. He has published more than 100 refereed papers in the above areas.