IEICE TRANSACTIONS

on Fundamentals of Electronics, Communications and Computer Sciences

DOI:10.1587/transfun.2024EAL2042

Publicized:2024/09/13

This advance publication article will be replaced by the finalized version after proofreading.

A PUBLICATION OF THE ENGINEERING SCIENCES SOCIETY

The Institute of Electronics, Information and Communication Engineers

Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

LETTER

Clock Drift Compensation for Master-Slave Clock Synchronization in EtherCAT Networks

Jiqian XU[†], **Lijin FANG**[†a)], **Qiankun ZHAO**[†], *Nonmembers*, **Yingcai WAN**[†], *Student Member*, **Yue GAO**[†], *and* **Huaizhen WANG**[††], *Nonmembers*

SUMMARY Clock synchronization represents an essential necessity for EtherCAT networks. This letter proposes a practical and efficient clock drift compensation scheme for the master-slave clock synchronization in EtherCAT networks, which decouples and then minimizes the adverse effect of master jitters for synchronization without degrading the clock synchronization performance between slave devices. Moreover, our method requires no modifications to the original EtherCAT protocol, nor does it introduce any additional bandwidth overhead. Comparative experimental results demonstrate the performance enhancement of the proposed approach over existing methods.

key words: *Clock synchronization, EtherCAT networks, pulse noise, master jitters, distributed clock*

1. Introduction

EtherCAT is an industrial Ethernet network system that boasts a powerful real-time capability and an open protocol [1]. It is extensively utilized in fields such as high-precision distributed measurement and motion control [2–4]. EtherCAT offers a hardware-based distributed clock (DC) synchronization mechanism in the dedicated EtherCAT slave controller (ESC) [5]. This mechanism ensures that all other DC-capable slaves (non-reference slaves) are synchronized with the first DC-capable slave (reference slave), resulting in clock deviations of less than 1 µs in small-to-medium systems [6]. In addition to the widely used 100 Mbit/s transmission data rate, EtherCAT Technology Group has launched EtherCAT G/G10 with 1/10 Gbit/s data rate in recent years, with no changes to the protocol. This demonstrates the scalability of EtherCAT networks, which is extremely valuable for application. However, the clock synchronization between the master and slaves requires additional effort since the master typically utilizes a standard Ethernet network interface card (NIC) for better compatibility [7]. The approaches for master-slave clock synchronization in EtherCAT networks can be categorized into two types: (a) synchronizing the reference slave to the master (S2M), and (b) synchronizing the master to the reference slave (M2S).

The EtherCAT master stack is typically implemented as a software package on large real-time operating systems (RTOS) such as PREEMPT_RT patched Linux, Xenomai, and RTAI [8, 9]. The acquisition and transmission of the EtherCAT master's clock for clock synchronization can be adversely affected by jitters in NIC driver, thread scheduling, and task processing. For the S2M method, the reference clock of EtherCAT network comes from the EtherCAT master [7]. Therefore, the jitters of EtherCAT master clock can influence clock synchronization of all nodes in EtherCAT network. Although the clock synchronization performance between the master and the reference slave (M-rS) is improved, the clock synchronization performance between the reference slave and the last non-reference slave (rS-lS) is degraded.

On the contrary, the M2S strategy obtains the reference clock of network from the reference slave which is more stable, ensuring reliable and accurate rS-lS clock synchronization. It should also be noted that the impact of jitters on M-rS clock synchronization remains for the M2S method. We still need to derive the clock drift (low frequency) of the master clock relative to the reference slave clock from the clock data coupling with jitters (high frequency). In [10], an M-rS clock drift compensation method is proposed. However, the performance could be further improved because the adverse effects of jitters are not fully decoupled and reduced by this method.

To fill the gaps of existing methods, this letter proposes a clock drift compensator which can theoretically eliminate the adverse effect of the thread scheduling and task processing jitters, as well as reducing that of NIC driver jitters on the clock drift calculation. The performance of rS-lS clock synchronization is also not compromised by our method.

2. Master-Slave Clock Synchronization Model

In this section, considering the jitters in the master and the clock drift of different devices, the master-slave clock synchronization model is set up under EtherCAT protocol. The master is allocated the local time through the system clock of the RTOS on which it runs. The slaves are allocated the local time using an internal clock. We assume that all slaves are capable of the DC mechanism and the slave closest to the master is designated as the reference slave.

As depicted in Fig. 1, the EtherCAT master immediately collects the timestamp t_{ms}^{i} by the function *clock_gettime()* in Linux userspace before the master sends the i-th cyclic EtherCAT frame to slaves. Therefore, the

[†]The authors are with the Faculty of Robot Science and Engineering, Northeastern University, Shenyang, China.

[††]The author is with the Institute of Shandong New Generation Information Industry Technology, Inspur Group, Jinan, China.

a) E-mail: ljfang@mail.neu.edu.cn (Corresponding author)

Copyright © 200x The Institute of Electronics, Information and Communication Engineers

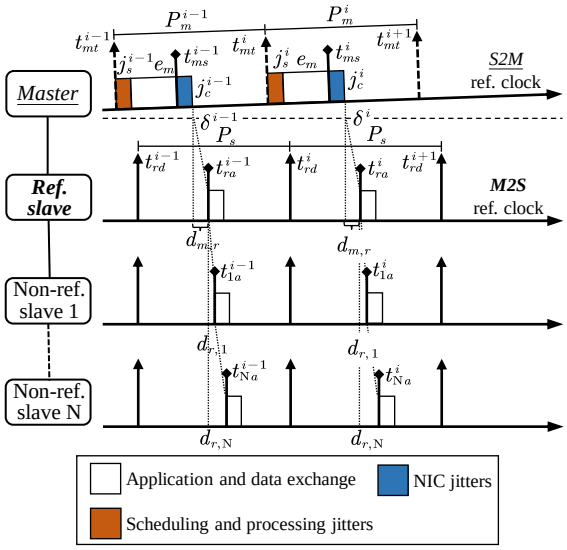


Fig. 1 Diagram of the master-slave clock synchronization model.

following relation is satisfied

$$t^i_{ms} = t^i_{mt} + j^i_s + e_m \tag{1}$$

where t^i_{mt} represents the theoretical timestamp when the master thread starts executing at the i-th cycle; j^i_s denotes the thread scheduling and processing jitters at the i-th cycle, which is a bounded random variable for RTOS; e_m is the execution time of the master application and data exchange. Similarly, the timestamp t^i_{ra} (stored in System Time register (0x0910:0x0917)) at which the i-th cyclic EtherCAT frame arrives at the reference slave is

$$t^i_{ra} = t^i_{ms} + j^i_c + \delta^i + d_{m,r} - f_{m,r} \tag{2}$$

$$\delta^i = \sum_{b=1}^{b=i} \Delta^b \tag{3}$$

where j^i_c denotes the NIC driver jitters at the i-th cycle, which is also a bounded random variable for RTOS; δ^i represents the total M-rS clock drift from the first cycle to the i-th cycle and Δ^b is the M-rS clock drift during the b-th cycle, which both are influenced by the quality of the crystal oscillator of each device; $d_{m,r}$ and $f_{m,r}$ (not shown in Fig. 1) refer to the propagation delay and time offset between the master and the reference slave, respectively. Similarly, we define $d_{r,n}$ and $f_{r,n}$ as the propagation delay and time offset between the reference slave and each non-reference slave, respectively, where the subscription $n = 1, ..., \mathrm{N}$ and N represents the number of non-reference slaves. Hence, the time relationship between the reference slave and the n-th non-reference slave is given by

$$t^i_{ra} = t^i_{na} + f_{r,n} - d_{r,n} \tag{4}$$

where t^i_{na} denotes the timestamp of the i-th cyclic EtherCAT frame arrives at the n-th non-reference slave.

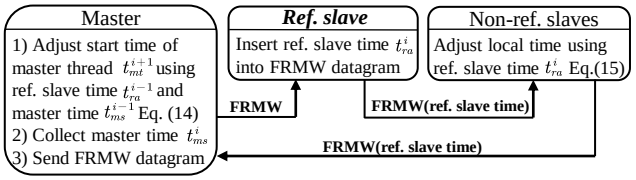


Fig. 2 The proposed M2S-based clock drift compensation procedure.

3. Proposed Master-Slave Clock Synchronization Method

The proposed method includes the measurement of propagation delay during the master initialization phase, and the clock drift compensation depicted in Fig. 2 during the mater operation phase.

3.1 Measurement of Propagation Delay and Time Offset

Compared with the rS-lS clock synchronization, the M2S method requires relatively low M-rS clock synchronization accuracy. Therefore, instead of measuring the time offset $f_{m,r}$ and propagation delay $d_{m,r}$ of M-rS separately and precisely, we derive the difference $h_{m,r}$ between $f_{m,r}$ and $d_{m,r}$ by

$$h_{m,r} = t_{ms} - t_{ra}. \tag{5}$$

Substituting (2) into (5), we can get

$$h_{m,r} = f_{m,r} - d_{m,r} - j_c - \delta. \tag{6}$$

To minimize the adverse effect of NIC jitters j_c on this measurement, the value of $h_{m,r}$ is averaged by sending 10,000 EtherCAT frames with interval of 1000 µs. In addition, due to the short measurement process after master initialization (<15 s), the total clock drift δ could be considered equal to zero during this process, i.e., $\delta = 0$. Hence, the following relationship can be obtained by

$$\bar{h}_{m,r} = \mathrm{AVG}\left(h_{m,r}\right) = f_{m,r} - d_{m,r} \tag{7}$$

where $\bar{h}_{m,r}$ is the averaged $h_{m,r}$ implemented for M-rS clock drift compensation.

For the rS-lS clock synchronization, by sending a broadcast write (BWR) datagram to Receive Time Port 0 of each slave, the master activates the propagation delay $d_{m,r}$ and time offset $f_{m,r}$ measurement [7]. Once the slave receives this frame, it latches the local time of the first preamble bit of this frame at all ports. Depending on the identified network topology, the master then records the Receive Time Port register of each slave and calculates these compensation values. For the linear network topology, $d_{r,n}$ and $f_{r,n}$ are calculated by

$$d_{r,n} = \frac{\left(t^r_{p1} - t^r_{p0}\right) - \left(t^n_{p1} - t^n_{p0}\right)}{2} \tag{8}$$

$$f_{r,n} = t^r_{p0} - t^n_{p0} + d_{r,n} \tag{9}$$

where t^r_{p0} and t^r_{p1} denote the Receive Time Port 0 and 1 register values of the reference slave, respectively; t^n_{p0} and t^n_{p1} represent the Receive Time Port 0 and 1 register values of the n-th non-reference slave, respectively. Finally, the master writes the corresponding calculation results back to the System Time Offset register (0x0920:0x0927) and System Time Delay register (0x0928:0x092B) of each non-reference slave, respectively. These values are implemented for rS-lS clock drift compensation.

Therefore, all clocks are considered to have completed coarse clock synchronization after the master initialization phase.

3.2 Clock Drift Compensation

3.2.1 M-rS Compensation

As depicted in Fig. 1, using t^{i-1}_{ra} and t^{i-1}_{ms}, we can calculate the M-rS clock drift ϕ^i at the i-th cycle

$$\phi^i = t^{i-1}_{ra} + \bar{h}_{m,r} - \left(t^{i-1}_{ms} + \hat{\delta}^{i-1}\right) \tag{10}$$

where $\hat{\delta}^{i-1}$ is the estimation of δ^{i-1} and $\hat{\delta}^0 = 0$. The master records the timestamp t_{ra} by sending configured address physical read multiple write (FRMW) datagram to read the System Time register of the reference slave in each cycle.

It is assumed that the clock synchronization has been successfully achieved before the last sampling cycle, i.e., $\delta^{i-1} = \hat{\delta}^{i-1}$. Hence, bringing Eqs. (2) and (7) into Eq. (10), the following relationship could be satisfied

$$\phi^i = \Delta^{i-1} + j^{i-1}_c \tag{11}$$

This suggests the advantage of our method of using t_{ms} and t_{ra} to compute the M-rS clock drift is that it is not affected by the thread scheduling and processing jitters j_s. Therefore, our method should be more efficient and accurate for clock drift estimation by a filter.

The NIC driver jitters j^{i-1}_c in ϕ^i can be viewed as a type of pulse noise [11]. The median filter (MED) is implemented for reducing this type of noise

$$\bar{\phi}^i = \mathrm{MED}\left(\phi^i, w\right) \tag{12}$$

where w is the sliding window width. Taking into account factors such as computational load and filtering performance, it is concluded through repeated experiments that the sliding window width w is designed to be 11 in this study. Then, the estimation of the total clock drift $\hat{\delta}^i$ at i-th cycle is obtained by

$$\hat{\delta}^i = \hat{\delta}^{i-1} + k\bar{\phi}^i \tag{13}$$

where k is the adjustment gain.

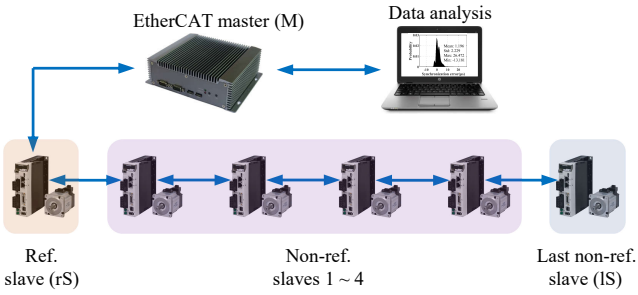


Fig. 3 The experimental environment.

Finally, we calculate the $(i+1)$-th cycle start time of master thread based on $\hat{\delta}^{i-1}$ as follows

$$t^{i+1}_{mt} = t^1_{mt} + iP_s - \hat{\delta}^i \tag{14}$$

where t^1_{mt} is configured by the master stack; P_s is the timing period of the slave DC interrupt.

3.2.2 rS-lS Compensation

The master periodically sends FRMW datagram to obtain the reference slave time t_{ra} from System Time register and writes it to the System Time register of each non-reference slave to trigger clock synchronization (DC mechanism) with the reference slave [6]. Combining Eqs. (4), (8), and (9), the clock drift $\theta^i_{r,n}$ between the reference slave and the n-th non-reference slave is given by

$$\theta^i_{r,n} = t^i_{na} + f_{r,n} - d_{r,n} - t^i_{ra}. \tag{15}$$

Depending on the sign of $\theta^i_{r,n}$, the Time Control Loop will adjust the local time of the n-th non-reference slave to follow that of the reference slave. The entire rS-lS clock synchronization process is implemented in the ESC by hardware circuits and therefore does not require user intervention.

4. Experiments

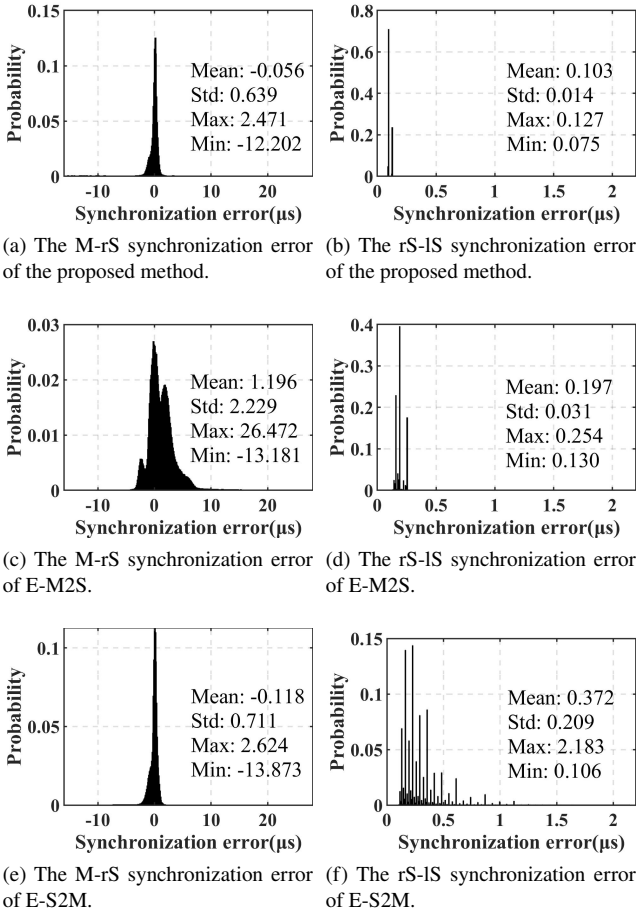
4.1 Experimental Setup

Using the open source IgH EtherCAT master stack [9], comparative experiments are conducted to assess the clock synchronization performance of the proposed method. As shown in Fig. 3, the EtherCAT master is an industrial PC equipped with a Realtek RTL8169 NIC and an Intel Core i3-4170 CPU. The RTOS is Ubuntu 18.04 LTS distribution patched with PREEMPT_RT (4.4.139-rt192). EtherCAT slaves consist of six Panasonic A6 servo drives and linear network topology is utilized, which is a typical application in the field of industrial robots. The slave closest to the master is designated as the reference slave and the rest are non-reference slaves. The cable length between the reference slave and the last non-reference slave is about 1.5 meters. 204 bytes of PDO (process data objects) data are exchanged between the master and slaves in each cycle.

Table 1 Statistics of clock synchronization errors of different methods.

Indicators (μs)	Proposed		E-M2S		E-S2M	
	M-rS	rS-lS	M-rS	rS-lS	M-rS	rS-lS
MMean	-0.091	0.098	1.236	0.135	-0.082	0.502
MStd	0.729	0.017	2.609	0.033	0.867	0.369
MMax	2.153	0.132	28.002	0.229	3.274	2.913
MMin	-11.858	0.061	-18.315	0.085	-12.479	0.156

The existing M2S method in [10] (E-M2S) and S2M method in [7] (E-S2M) are utilized as benchmarks to produce performance assessment. All experiments use the Linux command *stress-ng* to keep the CPUs in a fully loaded state so that the results are more in line with real-world usage scenarios. The parameters of our method are designed as $w = 11$ and $k = 0.01$. The PI controller parameters of E-M2S are $k_P = 0.125$ and $k_I = 0.0005$. The timing period of DC interrupt P_s is 1000 μs. Following the Section 3.1, $d_{r,n}$, $f_{r,n}$, and $\bar{h}_{m,r}$ are estimated during the master initialization phase. The M-rS clock synchronization errors of the proposed method and E-M2S are computed by Eq. (10), and the M-rS clock synchronization errors of E-S2M are calculated by $t^i_{ra} - t^i_{ms}$. The rS-lS clock synchronization errors (absolute values) of all methods are collected by periodically sending Broadcast Read (BRD) datagram to read the System Time Difference register (0x092C:0x092F) of slaves. We repeat the experiment five times for each method. For each experimental round, 1,800,000 samples (30 minutes) are recorded after the mater enters the operation phase for 2 minutes.



(a) The M-rS synchronization error of the proposed method.
(b) The rS-lS synchronization error of the proposed method.
(c) The M-rS synchronization error of E-M2S.
(d) The rS-lS synchronization error of E-M2S.
(e) The M-rS synchronization error of E-S2M.
(f) The rS-lS synchronization error of E-S2M.

Fig. 4 Distribution of clock synchronization errors of different methods.

4.2 Results

Fig. 4 illustrates the distribution of clock synchronization errors of different methods, where M-rS synchronization errors of the proposed method, E-M2S, and E-S2M are depicted in Figures. 4(a), 4(c) and 4(e), respectively; rS-lS synchronization errors of the proposed method, E-M2S, and E-S2M are displayed in Figures. 4(b), 4(d) and 4(f), respectively. Moreover, those synchronization error data are also statistically analyzed by four indicators including mean (Mean), standard deviation (Std), maximum (Max) and minimum (Min). Table 1 shows the five experimental means of the Mean, Std, Max, and Min indicators defined as MMean, MStd, MMax, and MMin, respectively.

Comparing the M-rS and rS-lS errors shown in Fig. 4 and Table 1, it is observed that the Std, Max and Min of M-rS errors are much larger than those of the rS-lS errors. This is because the master stack runs on a large RTOS (PREEMPT_RT, Xenomai, etc.) and uses a standard Ethernet NIC, resulting in greater jitters in transmitting EtherCAT frame, while the slave generally runs on a small embedded RTOS (FreeRTOS, RT-Thread, etc.) and uses dedicated ESC which offers a hardware-based DC mechanism.

Observing the M-rS synchronization errors of each method shown in Fig. 4 and Table 1, we can conclude that the M-rS error of the proposed method is the smallest, slightly better than that of E-S2M, while the M-rS error of E-M2S is the largest. This is because, in the E-M2S method, the different jitter sources in the master stack are not fully analyzed and dealt with accordingly. Fig. 4 and Table 1 also indicate that although the M-rS of E-S2M is roughly equivalent to that of the proposed method, the rS-lS synchronization performance of E-S2M is sharply inferior to the M2S-based method. For the rS-lS synchronization errors, the MStd of E-S2M are approximately twenty-two times that of the proposed method. In E-S2M method, the clock of the reference slave will not be stable due to the large transmission jitters in the clock synchronization frame sent by the master, which in turn affects the clock synchronization between slaves. That is, the M-rS errors will be passed to the rS-lS errors. Additionally, it is observed that the synchronization performance of the proposed method for rS-lS is slightly improved compared to E-M2S. Therefore, the proposed clock synchronization method has the better comprehensive performance over other methods.

5. Conclusion

A novel clock drift compensation method of master-slave clock synchronization is proposed for the distributed synchronization applications in EtherCAT networks. Considering the jitters in the master and the clock drift of different devices, the master-slave clock synchronization model is set up under EtherCAT protocol. Under the M2S clock synchronization framework, for the calculation of M-rS clock drift, the proposed method can theoretically eliminate the adverse effects of thread scheduling and task processing jitters, while the NIC jitters are efficiently suppressed by the median filter with low sliding window width. Comparative experiments are conducted to evaluate the clock synchronization performance of the proposed method. Experimental results show that the proposed method has the better comprehensive performance for master-slave clock synchronization than existing methods. What's more, our method requires no modifications to the original EtherCAT protocol and introduces no communication overhead.

In future work, we will conduct in-depth testing on the long-term stability and compatibility of the proposed method on different EtherCAT networks such as TSN (Time Sensitive Networking) capable EtherCAT networks.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 62273081, in part by the Science and Technology Small and Medium Enterprises Innovation Ability Enhancement Project of Shandong Province under Grant 2023TSGC0226, and in part by the Key R&D Plan of Shandong Province (Competitive Innovation Platform) under Grant 2023CXPT094.

References

[1] X. Wu and L. Xie, "Performance evaluation of industrial ethernet protocols for networked control application," Control Engineering Practice, vol.84, pp.208–217, 2019.

[2] J. Ahn, S. Park, J. Sim, and J. Park, "Dual-channel EtherCAT control system for 33-dof humanoid robot TOCABI," IEEE Access, vol.11, pp.44278–44286, 2023.

[3] J. Xu, H. Wang, Q. Zhao, Y. Gao, Y. Wan, and L. Fang, "A robotic manipulator using dual-motor joints: Prototype design and anti-backlash control," IEEE Robotics and Automation Letters, vol.8, no.12, pp.8327–8334, 2023.

[4] C.C. Tsai, F.C. Tai, C.A. Lin, and C.C. Chan, "EtherCAT-based impedance control of a 6-dof industrial robotic manipulator," 2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), Hong Kong, China, pp.80–85, IEEE, 2019.

[5] S.M. Park, H. Kim, H.W. Kim, C.N. Cho, and J.Y. Choi, "Synchronization improvement of distributed clocks in EtherCAT networks," IEEE Communications Letters, vol.21, no.6, pp.1277–1280, 2017.

[6] G. Cena, I.C. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "Evaluation of EtherCAT distributed clock performance," IEEE Transactions on Industrial Informatics, vol.8, no.1, pp.20–29, 2012.

[7] S.M. Park, H.W. Kim, H.J. Kim, and J.Y. Choi, "Accuracy improvement of master–slave synchronization in EtherCAT networks," IEEE Access, vol.8, pp.58620–58628, 2020.

[8] H.C. Yi and J.Y. Choi, "Cycle time improvement of EtherCAT networks with embedded linux-based master," IEICE Transactions on Information and Systems, vol.E102.D, no.1, pp.195–197, Jan. 2019.

[9] M. Cereia, I.C. Bertolotti, and S. Scanzio, "Performance of a real-time EtherCAT master under linux," IEEE Transactions on Industrial Informatics, vol.7, no.4, pp.679–687, 2011.

[10] N. Zhou and D. Li, "Cyber–physical codesign of field-level reconfigurations in networked motion controllers," IEEE/ASME Transactions on Mechatronics, vol.26, no.4, pp.2092–2103, 2021.

[11] K. Ye, Y. Yan, and H. Wu, "Time synchronization algorithm for networked control systems based on stochastic search," IEEE Transactions on Industrial Informatics, vol.18, no.1, pp.26–34, 2022.