A Real-Time Operating System for Cyber-Physical Systems Based on Physical Time and Logical Time

Keiko Amadera[1], Ayumu Ichimura[1,2], Takanori Yokoyama[1](✉), and Myungryun Yoo[1]

[1] Tokyo City University, 1-28-1, Tamazutsumi, Setagaya-ku, Tokyo 158-8557, Japan
{tyoko,myoo}@tcu.ac.jp

[2] Presently with RISO KAGAKU CORPORATION, Tokyo, Japan

Abstract. The paper presents a real-time operating system (RTOS) of a time-triggered distributed computing environment based on physical time and logical time for cyber-physical systems. In the environment, input and output tasks are activated synchronized with physical time and computation tasks are activated by the reception of timestamped messages and managed based on logical time. The control performance is affected by the jitters of input and output tasks but not affected by the jitters of computation tasks, so the jitter of the computation task activation is tolerated. However, the response time of low priority computation tasks may be increased in fixed-priority scheduling, which is used by most RTOSs. The paper presents a RTOS with mixed scheduling, in which fixed scheduling is used for input and output tasks to minimize the jitters and earliest deadline first (EDF) scheduling based on logical deadlines is used for computation tasks to minimize the response time. The logical deadline is not affected by the task activation time and higher priority is assigned to a computation task with an earlier logical deadline even if its activation is delayed, so the response time is improved. We have evaluated the performance of the RTOS and have confirmed that the performance is acceptable for practical embedded control systems.

Keywords: Real-time operating system · Cyber-physical systems · Embedded systems · Time-triggered architecture

1 Introduction

Cyber-physical systems (CPS) are real-time systems that affect physical processes [1]. Embedded control systems are hard real-time CPS, in which the delay and jitter may lead to performance degradation [2]. The time-triggered architecture (TTA) is suitable for building a hard real-time distributed system with minimal jitter [3]. TTA utilizes a time-triggered network such as TTP (Time-Triggered Protocol) [4] and FlexRay [5], which supports clock synchronization. Tasks on each node are managed according to the synchronized system time. However, the static scheduling of network communication compromises the flexibility.

Researches on distributed control systems utilizing the synchronized system time but not requiring static scheduling of network communication have been done. Henzinger et al. have presented an abstract programmer's model based on TTA called Giotto [6]. Giotto provides platform-independent development environment. Benveniste et al. have presented Loosely Time-Triggered Architecture (LTTA), which does not require clock synchronization [7,8]. Lee et al. have presented event-triggered distributed systems with time synchronization [9] and a programming model called PTIDES to provide a coherent temporal semantics based on the synchronized time [10].

Recently, distributed control systems with wireless communication such as vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications are increasing. The communication time of such systems varies and the message transfer order may be disturbed when the routing is dynamically changed. However, the disturbance of the message transfer order is not considered in Giotto and LTTA. PTIDES permits out of order processing events by utilizing timestamped events. PTIDES, however, requires that the clocks of all nodes must be synchronized. A distributed computing environment that tolerates the variation of communication time and the disturbance of the message transfer order is required.

A distributed embedded control system generally consists of tasks that perform sensor input operations, actuator output operations and computations. The jitter of input and output operations must be minimized in CPS. However, the jitter of computation is tolerated unless it affects the input and output operations. So, we have presented a time-triggered distributed computing environment in which input and output tasks are activated synchronized with physical time and computation tasks are activated by events of receiving timestamped messages and managed based on logical time [11]. The jitter of computation task activation caused by the variation of communication time is tolerated in the environment. The previous version of the environment uses OSEK OS [12] or AUTOSAR OS [13], the fixed priority of which may increase the response time of computation tasks with low priorities.

We present a real-time operating system (RTOS) to minimize the response time of computation tasks in this paper. The RTOS utilizes mixed scheduling, in which fixed scheduling is used for input/output tasks to minimize task activation jitter and earliest deadline first (EDF) scheduling [14] based on logical deadlines is used for computation tasks to minimize the response time. The logical deadline of a computation task is not affected by the activation time and a higher priority is assigned to a computation task with an earlier logical deadline even if its activation is delayed, so the response time is improved.

The rest of the paper is organized as follows. Section 2 describes the time-triggered distributed computing environment based on physical time and logical time. Section 3 describes the RTOS with the mixed scheduling. We evaluate the RTOS in Sect. 4. Section 5 concludes the paper.

2 Distributed Computing Environment

2.1 Logical Time-Triggered Processing

Figure 1 shows the structures of an example distributed embedded control system, which consists of four nodes and four periodic tasks; *Input Task*, *Computation Task A*, *Computation Task B* and *Output Task* are distributed to each node.

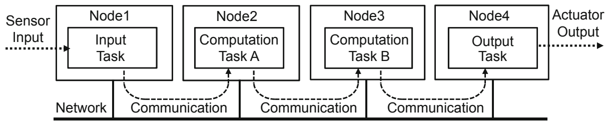

Fig. 1. Example distributed embedded control system

Figure 2 shows an example time chart of a time-triggered distributed system with a time-triggered network, the structure of which is the same as shown by Fig. 1. Each task is periodically activated in the period *10*. Each numbered rectangle shows a job of a task, which receives a message from its predecessor task and/or sends a message to its successor task. For example, a job of *Computation Task A* receives the message from *Input Task* and sends a message to *Computation Task B*. We call the difference between the time to activate a task and the time to activate its successor task *the inter-task delay time*. In this example, all the inter-task delay times are *10*.

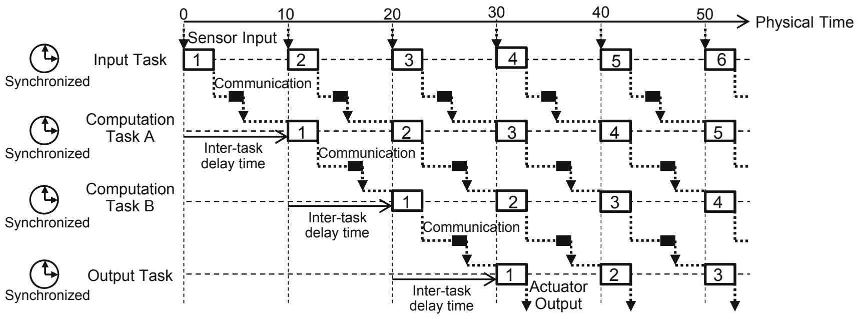

Fig. 2. Time chart of time-triggered architecture

Figure 3 shows an example time chart of a distributed system built with the time-triggered distributed computing environment based on physical time and logical time, the structure of which is same as shown by Fig. 1. This system utilizes a network with varying communication time delays. *Input Task* and *Output Task* are synchronously activated according to *physical time*. On the other hand, *Computation Task A* and *Computation Task B* are asynchronously activated by message reception events. The jitters of computation task activation are caused by varying communication time delays.

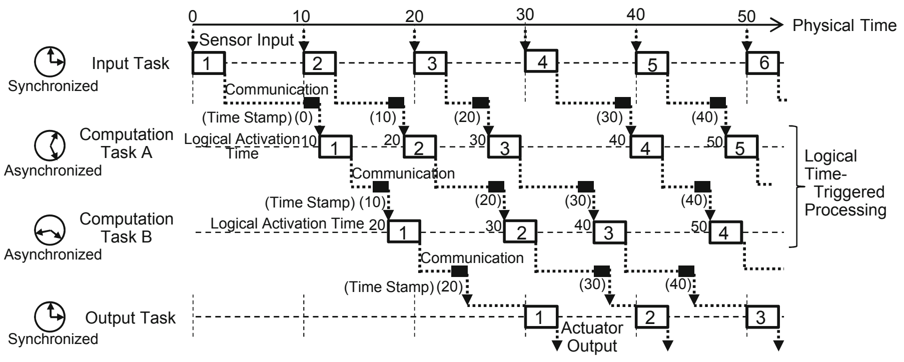

Fig. 3. Time chart of logical time-triggered processing

We introduce *logical time*, which is used to manage computation tasks. We call the activation time represented by logical time *the logical activation time*, which is given as same as the physical activation time in the corresponding TTA shown by Fig. 2. We also call the processing in which tasks are activated based on logical time *logical time-triggered processing*.

2.2 Timestamped Message

Timestamped messages are used to be tolerant with varying communication time delays. When a job of a task sends a message, the logical activation time of the job is attached to the message as a timestamp. The timestamps of messages are shown in parentheses at the receiver tasks in Fig. 3.

We call the inter-task delay time represented by logical time *the logical delay time*. The logical delay time is a constant and is defined to be equal to the corresponding inter-task delay time in TTA shown by Fig. 2. The logical delay time is statically designed considering the task execution time and the message communication time. The relation between the logical activation time, the timestamp of the received message and the logical delay time is indicated by the following formula (1).

$$Logical\,Activation\,Time = Timestamp + Logical\,Delay\,Time \tag{1}$$

When a message to a computation task is received, the sum of its timestamp and the logical delay time is checked to be equal to the logical activation time of the next job of the computation task. If so, the next job is activated. Otherwise, the activation is postponed because it means that the order of message transfer is disturbed. When a message with the timestamp equal to the next logical activation time is received, the postponed jobs are activated. Thus, the environment can utilize a network in which the order of message transfer may be disturbed.

2.3 Task Scheduling

The priorities of input/output tasks should be higher than the priorities of computation tasks. Fixed priority scheduling such as Rate Monotonic (RM) scheduling [14] is suitable for input/output tasks because the jitter of the activation of a task with a high priority is small. However, fixed priority scheduling is not suitable for computation tasks. If the activation of a computation task with a low priority is delayed by the communication delay, the response time of the computation task may be increased.

Figure 4 shows the structures of an example distributed embedded control system, which consists of three nodes. *Computation Task A* and *Computation Task B* reside on *Node2*. *Computation Task A* is activated when receiving a message from *Input Task A* on *Node1* and *Computation Task B* is activated when receiving a message from *Input Task B* on *Node3*.

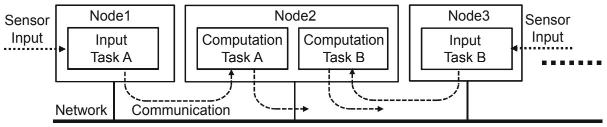

Fig. 4. Another example distributed embedded control system

Figure 5 shows an example time chart of the system when the computation tasks on *Node2* are scheduled by fixed priority scheduling. The priority of *Computation Task A* is higher than the priority of *Computation Task B*. We call the absolute deadline represented in logical time *the logical deadline*. The logical deadline of each job is calculated by adding the relative deadline of the task to the logical activation time. In this example, the relative deadline is *10*, which is same as the period of the task. The response time of *Computation Task B* is much larger than the response time of *Computation Task A* because the jobs of *Computation Task B* are preempted by the jobs of *Computation Task A*. For example, the first job of *Computation Task B* is activated at physical time *11* and completed at physical time *24* because it is preempted by the second job of *Computation Task A*.

We adopt EDF scheduling based on logical deadlines for computation tasks. Earlier the logical deadline of a job is, higher the priority of the job is. Thus, a higher priority is assigned to a task with an earlier logical deadline even if its activation is delayed.

Figure 6 shows the time chart of a system, in which the computation tasks on *Node2* are scheduled by EDF scheduling. The response time of *Computation Task B* is improved. For example, the first job of *Computation Task B* is not preempted by the second job of *Computation Task A* and completed at physical time *19* because the priority of the first job of *Computation Task B* with logical deadline *20* is higher than the second job of *Computation Task A* with logical deadline *30*.

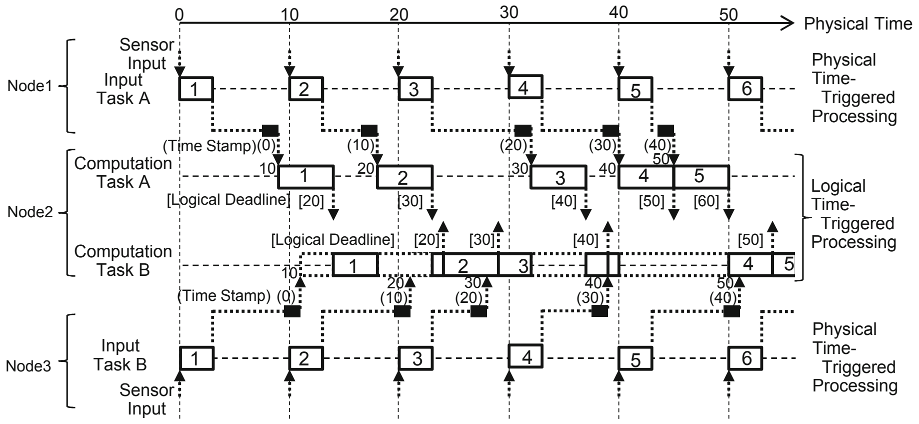

Fig. 5. Time chart of fixed priority scheduling

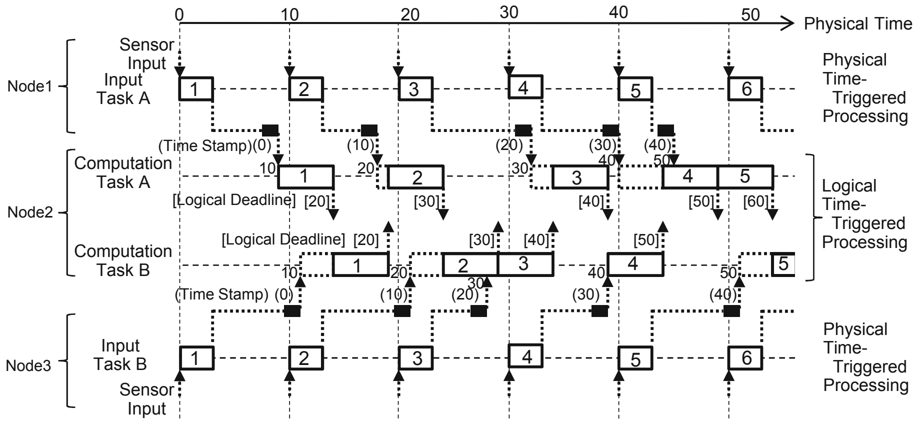

Fig. 6. Time chart of EDF scheduling

The response time of computation tasks is minimized because the logical deadline of the job of a computation task is not affected by the activation time and higher priority is assigned to a job with an earlier logical deadline even if its activation is delayed.

We call this scheduling the mixed scheduling, the mechanism of which is described in Sect. 3.1.

2.4 Software Structure of Distributed Computing Environment

Figure 7 shows the software structure of the distributed computing environment. The distributed computing middleware [11] runs on the RTOS presented in this paper.

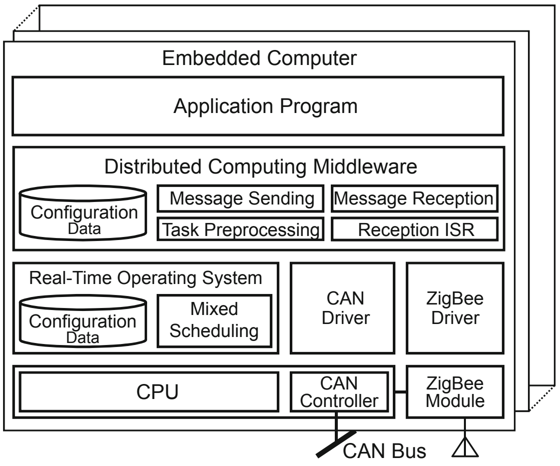

Fig. 7. Software structure

We develop the RTOS by extending an OSEK-compliant RTOS called TOPPERS/ATK1, which has been developed by TOPPERS Project [15]. We use a time synchronization mechanism based on GNSS (Global Navigation Satellite Systems) [16] for nodes in which input or output tasks reside. The system time synchronization is not needed if input tasks and output tasks reside on the same node.

The middleware supports timestamped message communication and consists of modules that performs *message sending*, *message reception*, *task preprocessing* and *reception ISR* (Interrupt Service Routine). The middleware also supports message communication through CAN network [17] and ZigBee wireless network [18].

3 Real-Time Operating System

3.1 Mixed Scheduling

We extend the scheduler of TOPPERS/ATK1 to support mixed scheduling. The scheduler deal with three kinds of tasks: input/output tasks, computation tasks and non real-time tasks. Input/output tasks and non real-time tasks are scheduled by fixed scheduling and computation tasks are scheduled by EDF scheduling based on their logical deadlines. The priorities of input/output tasks are higher than the priorities of computation tasks. The priorities of non real-time tasks are lower than the priorities of computation tasks.

Figure 8 shows the mixed scheduling mechanism of the RTOS. There are ready queues with priority levels, in which tasks with state *ready* are queued according to their priority levels. One of the ready queues is used for computation tasks scheduled by EDF scheduling. The ready queues with higher priority levels are used for input/output tasks and the ready queues with lower priority levels are used for non real-time tasks. The number of ready queues and the priority level of the computation task ready queue can be statically defined by a developer. In the case of Fig. 8, there are sixteen priority levels: the priority levels from *8* to *15* for input/output tasks, the priority level *7* for computation tasks, and the priority levels from *0* to *6* for non real-time tasks.

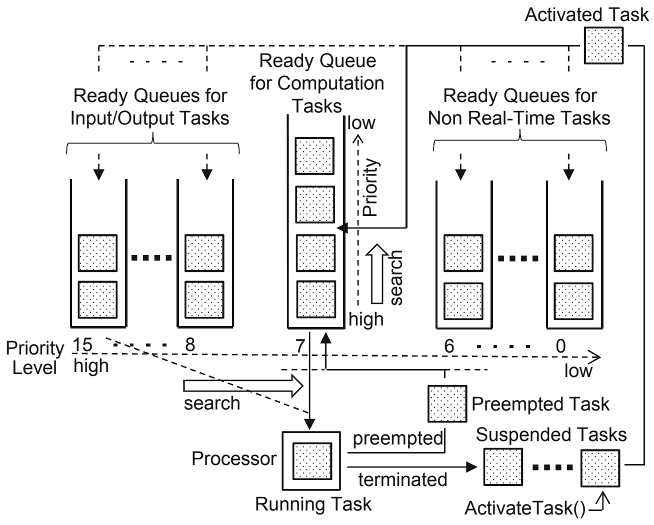

Fig. 8. Mixed scheduling mechanism

When a task is activated, the scheduler shifts the state of the task to *ready* and enqueues the task into a ready queue according to its priority level. If the priority level of the activated task is higher than the priority level of the running task, a preemption occurs and the preempted task is enqueued into a ready queue. When a running task is terminated, the scheduler selects the first task in the highest-priority no-empty ready queue to be executed by the processor.

Computation tasks with state *ready* are queued in a single ready queue according to their logical deadlines, which mean priorities. When a computation task is activated, the scheduler calculates the logical deadline by adding the relative deadline to the logical activation time and inserts the computation task into the computation task ready queue comparing its logical deadline and other queued tasks' deadlines. The task with the earliest logical deadline other than the running computation task is to be at the head of the ready queue. If the logical deadline of the activated computation task is earlier than the logical deadline of the running computation task, a preemption occurs and the preempted computation task is enqueued into the computation task ready queue. When a running computation task is terminated, the scheduler selects the computation task at the head of the ready queue to be executed by the processor.

3.2 Task Activation Mechanism

In an OSEK OS or AUTOSAR OS, periodic tasks are activated by an alarm associated with the system counter, which counts the system time. Thus, the system counter represents physical time. The upper part of Fig. 9 shows the system counter and alarms for input/output tasks. The system counter is updated by the tick interrupt, which is periodically issued by a hardware timer. If a non real-time task is a periodic task, it is also activated by an alarm associated with the system counter.

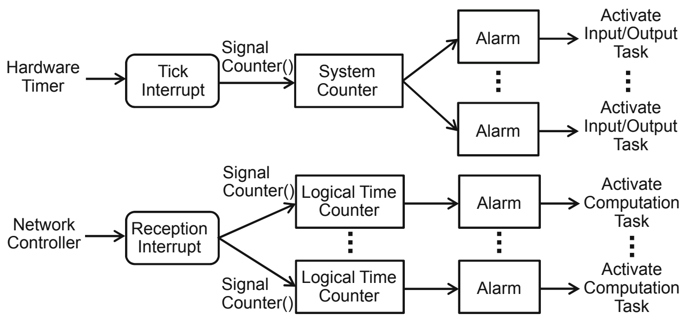

Fig. 9. Counters and alarms

A counter other than the system counter can be defined in an OSEK OS or AUTOSAR OS. A user-defined counter is used to represent a logical time and an alarm to activate a computation task is associated with the counter. The lower part of Fig. 9 shows logical time counters (user-defined counters for logical times) and alarms to activate computation tasks.

3.3 Logical Time Updating

TOPPERS/ATK1 provides the API *SignalCounter()* to update a counter. When a message for a computation task is received, the reception ISR (Interrupt Service Routine) of the middleware is issued. The reception ISR calls *SignalCounter()* if the timestamp of the received message meets the following formula (2).

$$Timestamp = Next\,Logical\,Activation\,Time - Logical\,Delay\,Time \tag{2}$$

If the order of message transfer is disturbed and the timestamp does not meet the formula (2), the ISR does not call *SignalCounter()* and the task activation is postponed. When a message with the timestamp corresponding to the next logical activation time of the task is received, the ISR repeatedly calls *Signal-Counter()* and the postponed jobs are sequentially executed together.

Figure 10 shows a time chart that illustrates the behavior of computation task activation. The period of the message communication is *10* and the logical

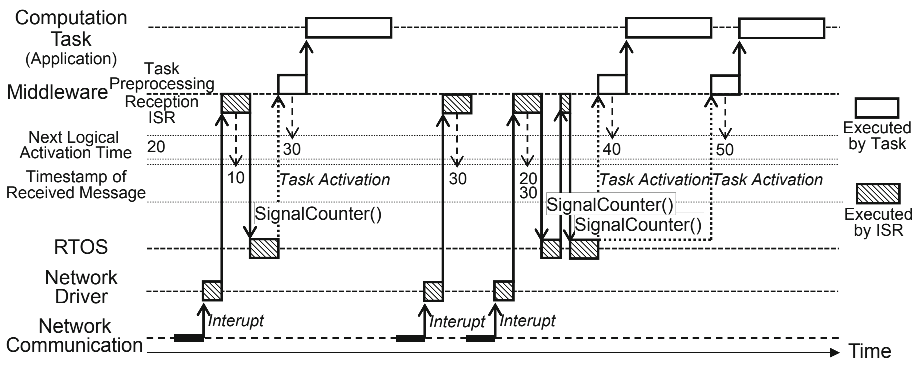

Fig. 10. Time chart of computation task activation

delay time is *10* in this example. The initial value of the next logical activation time is *20*.

When the first message with timestamp *10* is received, the reception ISR of the middleware is issued. The reception ISR stores the timestamp and the data of the received message into the buffer and calls *SignalCounter()* to update the logical time counter because the timestamp of the received message meets the formula (2). Then, the computation task is activated by the alarm of the RTOS. When the computation task is activated, the task preprocessing is executed before executing the application. The task preprocessing updates the next logical activation time.

In this example, the timestamp of the second message is *30* and the timestamp of the third message is *20* because the message transfer order is disturbed. The reception ISR does not call *SignalCounter()* when the second message is received and calls *SignalCounter()* twice when the third message is received. Thus, the distributed computing environment is tolerated with the disturbance of the message transfer order.

The logical time counter is updated by the period of the message communication at a time. If the period of the task is n times of the message communication period, the task is activated once in n times of message reception.

4 Implementation and Evaluation

We use an evaluation board in which a microprocessor called H8S/2638F is installed. The H8S/2638F has on-chip memories: the 256 kBytes ROM and the 16 kBytes RAM. The clock rate of the microprocessor is 20 MHz. The H8S/2638F also has two on-chip CAN controllers. An XBEE wireless module that supports ZigBee protocol is connected to the H8S/2638F through UART serial communication.

We have measured the CPU execution time of the task management system calls of the developed RTOS to evaluate the overhead of the mixed scheduling.

We have measured the CPU execution time of *ActivateTask()*, *TerminateTask()*, *ChainTask()* and *Schedule()* in the case of fixed scheduling and in the case of EDF scheduling. We have separately measured the execution time of *Schedule()* in the case without dispatching and in the case with dispatching.

Table 1 shows the CPU execution time of the system calls. The execution time of task activation in EDF scheduling is 27% larger than in fixed priority scheduling because of ready queue search. The table also shows the CPU execution time of original TOPPERS/ATK1 that supports just fixed scheduling for comparison. We think that the overhead is practically acceptable because the overhead is less than 9% in fixed scheduling.

Table 1. Execution time of task management system calls

System call		Execution time [μsec]		
		Developed RTOS		TOPPERS/ATK1
		Fixed scheduling	EDF scheduling	
ActivateTask		21.9	27.9	20.1
TerminateTask		8.3	8.3	7.8
ChainTask		18.3	21.6	17.5
Schedule	without dispatch	21.2	22.0	20.3
	with dispatch	27.3	28.1	26.4

5 Conclusion

We have presented a RTOS for time-triggered distributed computing environment based on physical time and logical time, which is suitable for cyber-physical systems utilizing networks with varying communication time. The RTOS schedules input/output tasks by fixed scheduling and schedules computation tasks by EDF scheduling based on their logical deadlines. The response time of computation tasks is minimized because a higher priority is assigned to a computation task with an earlier logical deadline even if its activation is delayed. We have evaluated the CPU execution time of the task management system calls of the RTOS and have confirmed the overhead is practically acceptable.

The distributed computing environment just supports periodic tasks. The future work is to extend the RTOS and the middleware to support not only periodic tasks but also aperiodic tasks. We are considering to adopt another scheduling algorithms such as a fixed priority server or a dynamic priority server to support aperiodic tasks. We are also extending the middleware to support other wireless communications such as IEEE802.11p.

Acknowledgment. The authors would like to thank the developers of TOPPERS/ATK1. This work was supported by JSPS KAKENHI Grant Number JP18K11225.

References

1. Lee, E.A.: Cyber physical systems: design challenges. In: Proceedings of 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing, pp. 363–369 (2008)
2. Cervin, A., Henriksson, D., Lincoln, B., Eker, J., Arzen, K.: How does control timing affect performance? Analysis and simulation of timing using jitterbug and TrueTime. IEEE Control. Syst. **23**(3), 16–30 (2003)
3. Kopetz, H.: Should responsive systems be event-triggered or time-triggered? IEICE Trans. Inf. Syst. **E76–D**(11), 1325–1332 (1993)
4. Kopetz, H., Grunsteidl, G.: TTP-A protocol for fault-tolerant real-time systems. IEEE Comput. **27**(1), 14–23 (1994)
5. Makowitz, R., Temple, C.: FlexRay - a communication network for automotive control systems. In: Proceedings of 2006 IEEE International Workshop on Factory Communication Systems, pp. 207–212 (2006)
6. Henzinger, T.A., Horowitz, B., Kirsch, C.M.: Giotto: a time-triggered language for embedded programming. In: Henzinger, T.A., Kirsch, C.M. (eds.) EMSOFT 2001. LNCS, vol. 2211, pp. 166–184. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45449-7_12
7. Benveniste, A., Caspi, P., Guernic, P.L., Marchand, H., Talpin, J.-P., Tripakis, S.: A protocol for loosely time-triggered architectures. In: Sangiovanni-Vincentelli, A., Sifakis, J. (eds.) EMSOFT 2002. LNCS, vol. 2491, pp. 252–265. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45828-X_19
8. Baudart, G., Benveniste, A., Bourke, T.: Loosely time-triggered architectures: improvements and comparisons. In: Proceedings of the 12th International Conference on Embedded Software (EMSOFT'15), pp. 85–94 (2015)
9. Lee, E.A., Matic, S.: On determinism in event-triggered distributed systems with time synchronization. In: Proceedings of 2007 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication, pp. 56–63 (2007)
10. Eidson, J.C., Lee, E.A., Matic, S., Seshia, S.A., Zou, J.: Distributed real-time software for cyber-physical systems. In: Proceedings of the IEEE, vol. 100, no. 1, pp. 45–59 (2012)
11. Ichimura, A., Yokoyama, T., Yoo, M.: A time-triggered distributed computing environment for cyber-physical systems based on physical time and logical time. In: Proceedings of 2018 IEEE Region 10 Conference (TENCON'18), pp. 1516–15210 (2018)
12. OSEK/VDX: Operating System, Version 2.2.3 (2005)
13. AUTOSAR: Specification of Operating System, Release 4.3.0 (2016)
14. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM **20**(1), 46–61 (1973)
15. TOPPERS Project. http://www.toppers.jp/. Accessed 9 May 2019
16. Yokoyama, T., Matsubara, A., Yoo, M.: A real-time operating system with GNSS-based tick synchronization. In: Proceedings of IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications, pp. 19–24 (2015)
17. Kiencke, U.: Controller area network - from concept to reality. In: Proceedings of 1st International CAN Conference, pp. 11–20 (1994)
18. Aliance, Z.: http://www.zigbee.org/. Accessed 9 May 2019