Synchronizing Real-Time Tasks in Time-Aware Networks: Work-in-Progress

Eleftherios Kyriakakis
DTU Compute
Technical University of Denmark
Kgs. Lyngby, Denmark
Email: elky@dtu.dk

Jens Sparsø
DTU Compute
Technical University of Denmark
Kgs. Lyngby, Denmark
Email: jspa@dtu.dk

Peter Puschner
Inst. of Computer Engineering
TU Wien
Vienna, Austria
Email: peter@vmars.tuwien.ac.at

Martin Schoeberl
DTU Compute
Technical University of Denmark
Kgs. Lyngby, Denmark
Email: masca@dtu.dk

***Abstract*—Distributed safety-critical systems require both time-predictable task execution and communication. On the processor, the execution of the tasks is dictated by a scheduling policy, while on the network, different industrial communication protocols can be deployed to guarantee bounded message latency. In this paper, we investigate the synchronization of the task execution with the underlying communication schedule, and we propose an open-source software framework. We implement a cyclic executive task scheduling policy on a time-predictable platform and synchronize the task execution with the underlying TTEthernet communication schedule. We evaluate our framework by developing a simple one-sensor, one-actuator industrial control example, distributed over three nodes. The presented real-time system can exchange messages with minimal jitter, and the distributed tasks synchronize to a precision of $\approx 1.6\mu$s.**

***Index Terms*—Time-triggered communication, network synchronized task execution, clock synchronization, WCET analysis, cyclic executive.**

I. Introduction

Communication and task execution in distributed cyber-physical systems are both parts of the critical end-to-end latency of an application, as computation results contained in transmitted frames often contain data that need to be consumed by an actuator at a precise moment in time, in-order and without missed data [1].

This work follows the time-triggered communication paradigm and investigates the synchronization of the distributed real-time task executions with the underlying communication schedule using TTEthernet [2]. TTEthernet deploys a cyclic communication schedule called *TTE network schedule* that is built offline and defines the exact transmission and reception points in time. At runtime, end-systems use a fault-tolerant, network-wide, time-synchronization protocol [3] that allows for sub-microsecond precision.

Synchronized distributed task execution not only increases the application's level of determinism, but it also allows for precise end-to-end latency calculation, reduced jitter and end-system buffer requirements [4]. System properties such as buffer usage can be statically estimated and decreased as the exact transmission/reception points in times are known during the development phase.

In this work, we develop and present a complete implementation of a worst-case execution time (WCET) analyzable open-source framework that achieves high precision task synchronization with short end-to-end communication latency, minimal jitter and buffer usage.

II. Task Model

We define each task τ_i by the tuple $(T_i, C_i, D_i, O_i, J_i)$, where T_i is the period, C_i is the WCET, D_i is the deadline, O_i is a relative offset, and J_i is the allowed maximum jitter of each task. We only consider tasks with harmonic periods. The hyper-period of the schedule is the least-common multiplier of the periods of the considered tasks: $lcm\{T_1, T_2...T_n\}$. Let $S_{i,n}$ be the release time of task i at its n-th instance within a hyper period, then we constraint $S_{i,n}$ to T_i, D_i and O_i as shown in Eq. 1a, 1c & 1b.

$$S_{i,n} - S_{i,n-1} \leq T_i \pm J_i \qquad J_i \leq T_i \quad (1a)$$

$$S_{i,n} \geq n \times T_i + O_i \qquad O_i \leq T_i \quad (1b)$$

$$S_{i,n} + C_i \leq n \times T_i + D_i + J_i \qquad D_i, J_i \leq T_i \quad (1c)$$

Subsequently, we test the activation time of each task $S_{i,n}$ to never coincide within the execution instance of an on-going task $S_{j,k}$ within a hyper-period of the schedule using Eq. 2. We define ϕ as a constant offset between release times, which allows us to account for the WCET of the runtime system and any other possible delays: $\phi = WCET_{runtime}$.

$$(S_{i,n} \geq S_{j,k} + C_j + \phi) \ \vee \ (S_{i,n} + C_i + \phi \leq S_{j,k}) \quad (2)$$

Where n, k are instances of the tasks i, j respectively in the range of a hyper-period, $\forall i \neq j$. Setting D_i and O_i to predefined points in time (PIT), such as the *transmission* or the *reception window* of the *TTE network schedule* allows to setting precedence constraints.

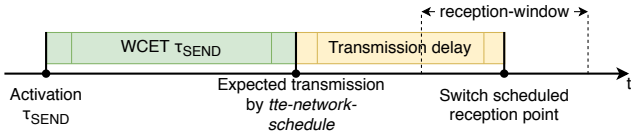

Fig. 1: Task release point in time relationship to scheduled *transmission-window*.

III. Design and Implementation

Off-the-shelf TTEthernet applications use transmission and reception mechanisms implemented by proprietary network-interface cards and drivers. In contrast, we design a bare-metal cyclic executive runtime system[1] that synchronizes to the underlying time-triggered communication schedule. We implement the design on the time-predictable processor Patmos [5].

A. Communication

The transmission and reception, of critical traffic frames, is handled by periodic designated tasks, scheduled at each transmit/receive PIT according to the *TTE network schedule*. Transmitted frames must arrive within the receive acceptance window of the connected switch, and in the correct virtual link; otherwise, the frame is dropped. Similarly, scheduled reception tasks are listening for a predefined *reception-window* time duration, as illustrated in Figure 1.

B. Synchronization

The runtime system synchronizes to the network time by scheduling a periodic task that handles incoming protocol control frames. The task uses the *permanence function* [2] to calculate the relative clock offset that is then accessible through a function call that returns the synchronized time after applying a PI filter. The synchronized time is used by the cyclic executive scheduler to query the release time of each task. It is worth noting that the synchronization task is also responsible for updating the upcoming release times of each task.

C. Offline Scheduling

The synthesis of the task schedule is based on the communication schedule and resembles the process described by [6]. We develop a custom SMT offline scheduler[2] based on the task model described in Section II. We generate a cyclic *task schedule* according to the task definitions derived by the control requirements of the application (i.e. the sensor sampling rate, the max. duty-cycle and period of the motor), the WCET bounds and the transmission/reception PIT defined by the *TTE network schedule*. [3] We use the transmission/reception PIT defined by the *TTE network schedule* as inputs to the release times $S_{i,0}$ of each related task. Computation tasks are synchronized to any related communication tasks by mapping the transmit/receive PIT of the *TTE network schedule* to their O_i and D_i constraints during scheduling.

[1] *https://github.com/t-crest/patmos/tree/master/c/apps/ttecps*ff
[2] *https://github.com/egk696/SimpleSMTScheduler*ff
[3] The network schedule is generated using the TTTech development suite.

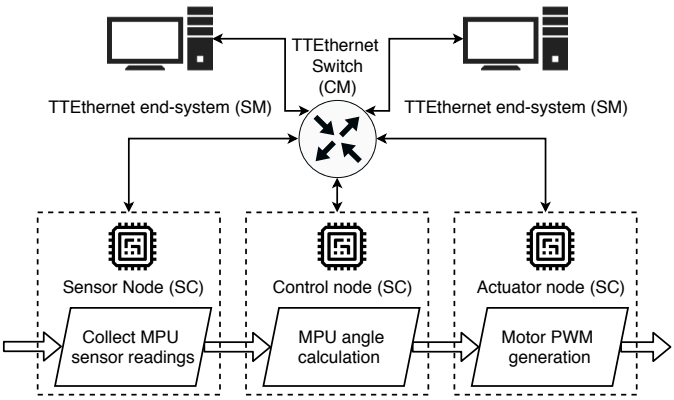

Fig. 2: Experimental setup of the control application with tasks distributed over a TTEthernet network.

IV. Evaluation

To evaluate the presented framework, we develop a simple distributed control application that reads the input from a motion processing unit (MPU), determines its orientation and controls the rotation of a motor using pulse-width modulation (PWM) signal. The application is composed of three distributed nodes: a *sensor node*, a *control node*, and an *actuator node*.

The *sensor node* interfaces with a motion processing unit sensor MPU-9250 The sensor is sampled by reading alternating measurements from either the IMU or the gyroscope at a sampling frequency of 200 Hz. The node transmits the measurements to the *control node*. The sampling rates of the sensors are empirically chosen.

The *control node* is responsible for converting the received values from the *sensor node* to a duty-cycle that is sent to the *actuator node*. It calculates the angle of the MPU sensor on the X-axis by fusing the measurements of the accelerometer and the gyroscope using a complementary filter [7] and converting the attitude estimation to a valid duty-cycle range.

The *actuator node* receives new duty-cycles and drives a servo motor using pulse-width modulation (PWM) signal generated in software. The PWM signal must adhere to the following characteristics; a duty cycle in the range of 1.5%–10% and a period of 20 ms. This requirement does not only set a constraint on the task execution but also on the end-to-end latency, as the new command for the servo motor should arrive before its next period.

A. System Setup

The application nodes are implemented in three FPGA Terasic DE2-115 boards operating at 80 MHz and integrated as synchronization clients (SC) in an existing TTEthernet network topology, as shown in Figure 2. The network consists of an industrial TTE Chronos Switch acting as compression master (CM) and two Linux desktops equipped with TTEthernet Ethernet cards that act as synchronization masters (SM).

TABLE I: WCET of runtime system functions of the individual nodes in clock cycles.

Function	Node		
	Sensor	Control	Actuator
`sort_ttetasks`	6607	24123	13908
`executive_loop`	12369	14200	11346
`get_tte_aligned_time`		129	

TABLE II: Generated task schedule of the evaluated system. Asterisks indicate a constraint by the *TTE network schedule*.

Node	Task	Period (μs)	WCET (μs)	$S_{i,\emptyset}$ (μs)	Utilization
sensor	τ_{SSYNC}	10000	67.175	0	4.30%
	$\tau_{SENSE(a/g)}$	5000	153.412	229.025	
	τ_{SSEND}	5000	28.300	(*) 771.700	
control	τ_{CSYNC}	10000	75.063	0	16.43%
	τ_{CRECV}	5000	271.062	(*) 1200.000	
	τ_{CTRL}	5000	485.05	1632.912	
	τ_{CSEND}	5000	28.300	(*) 3571.700	
actuator	τ_{ASYNC}	10000	67.175	0	16.44%
	τ_{ARECV}	5000	271.062	(*) 4000.000	
	τ_{PWM}	20000	2070.937	5426.432	

B. Generated schedule

We perform a static WCET analysis on the significant functions of the runtime system and the tasks of the example application using the tool *platin* [8]. Table I presents the WCET bounds of the significant functions of the proposed runtime system in clock cycles. Both the sorting function `sort_ttetasks()` and the runtime dispatcher `executive_loop()` depend on the number of tasks scheduled; thus, the WCET varies depending on the task set. We use the static WCET bound of the runtime dispatcher to set the scheduling constraint ϕ for each node's schedule. This bound includes the required 129 clock cycles WCET to read the hardware clock and apply the PI filter.

Table II presents the generated initial release times $S_{i,\emptyset}$ of the evaluated application's task set using the presented offline scheduler. It is worth noting that the WCET of the τ_{SYNC} function varies depending on the number of upcoming releases per task that it has to synchronize. We configure the communication schedule to use max frame-size of 64 bytes.

C. Task synchronization and communication

To evaluate the task synchronization precision, we use an I/O pulse generated by the synchronization tasks on each node and measure the relative time offset. Preliminary results show that the tasks are synchronized to a precision of $\approx$1.6μs. The system's end-to-end latency can be defined as the time difference between the *sensor node* readout and the consumption of a new duty-cycle (τ_{PWM}) by the *actuator node*: $L_{e2e} = S_{PWM,\emptyset} - S_{SENSE\emptyset}$. We calculate the end-to-end latency as 5.4 ms, which is well within the deadline constrain of the motor PWM.

D. Interoperability

Although the presented design was implemented on the Patmos processor, the framework itself does not depend on the underlying hardware architecture. It can be implemented on any PRET-like platform [9], that can guarantee statically bounded WCET and provides an interface to the underlying network. Moreover, the presented methodology and framework do not depend on TTEthernet, and with few modifications to the synchronization task, the runtime system could integrate with other industrial Ethernet protocols such as time-sensitive networks (TSN) [10].

V. Conclusion and Future Work

The presented work investigated the concept of synchronizing the task execution with the time-triggered communication schedule and presented an open-source WCET analyzable software framework. The design was evaluated using a simple distributed control application that achieved low end-to-end latency and precise task synchronization.

In the future we plan to extend our framework and evaluation in the following ways: (a) perform a comparative analysis of the proposed network synchronized task execution versus asynchronous, (b) design a multicore extension of the framework and (c) perform an extended scalability evaluation by creating and distributing synthetic workloads (i.e. using ROSACE [11]).

Acknowledgment

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 764785.

References

[1] R. Zurawski, *Industrial communication technology handbook, second edition*. CRC Press, 2017.

[2] W. Steiner, G. Bauer, B. Hall, and M. Paulitsch, "Time-triggered ethernet," in *Time-Triggered Communication*. CRC Press, 2018, ch. 8, pp. 209–248.

[3] W. Steiner and B. Dutertre, "Automated formal verification of the ttethernet synchronization quality," in *NASA Formal Methods Symposium*. Springer, 2011, pp. 375–390.

[4] P. Puschner and R. Kirner, "Asynchronous vs. synchronous interfacing to time-triggered communication systems," *Journal of Systems Architecture*, vol. 103, p. 101690, 2020.

[5] M. Schoeberl, W. Puffitsch, S. Hepp, B. Huber, and D. Prokesch, "Patmos: A time-predictable microprocessor," *Real-Time Systems*, vol. 54(2), pp. 389–423, Apr 2018.

[6] S. S. Craciunas, R. S. Oliver, and V. Ecker, "Optimal static scheduling of real-time tasks on distributed time-triggered networked systems," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. IEEE, 2014, pp. 1–8.

[7] P. Gui, L. Tang, and S. Mukhopadhyay, "Mems based imu for tilting measurement: Comparison of complementary and kalman filter based data fusion," in *2015 IEEE 10th conference on Industrial Electronics and Applications (ICIEA)*. IEEE, 2015, pp. 2004–2009.

[8] S. Hepp, B. Huber, J. Knoop, D. Prokesch, and P. P. Puschner, "The platin tool kit - the T-CREST approach for compiler and WCET integration," in *Proceedings 18th Kolloquium Programmiersprachen und Grundlagen der Programmierung, KPS 2015, Pörtschach, Austria, October 5-7, 2015*, 2015.

[9] S. A. Edwards and E. A. Lee, "The case for the precision timed (PRET) machine," in *DAC '07: Proceedings of the 44th annual conference on Design automation*. New York, NY, USA: ACM, 2007, pp. 264–265.

[10] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner, "Enabling fog computing for industrial automation through time-sensitive networking (tsn)," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 55–61, 2018.

[11] C. Pagetti, J. Forget, H. Falk, D. Oehlert, and A. Luppold, "Automated generation of time-predictable executables on multicore," in *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, 2018, pp. 104–113.