# Synchronizing Real-Time Tasks in Time-Triggered Networks

Eleftherios Kyriakakis*, Jens Sparsø*, Peter Puschner†, Martin Schoeberl*
*DTU Compute, Technical University of Denmark, Kgs. Lyngby, Denmark
†Inst. of Computer Engineering, TU Wien, Vienna, Austria
E-mail: elky@dtu.dk, jspa@dtu.dk, peter@vmars.tuwien.ac.at, masca@dtu.dk

2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC) | 978-1-6654-0414-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/ISORC52013.2021.00013

***Abstract*—In order to guarantee end-to-end latency and minimal jitter in distributed real-time systems, it is necessary to provide tight synchronization between computation and communication. This requires time-predictable execution of tasks across all processing nodes, and the use of a network protocol that can provide a global time base and bounded communication latency. TTEthernet is one such industrial communication protocol.**

**This paper investigates the synchronization of the task execution schedule with the underlying communication schedule, and we propose an open-source software framework for time-triggered end-systems. We present the implementation of a static cyclic task schedule, on a time-predictable platform that is integrated within a TTEthernet network and synchronized with the communication schedule. We evaluate the presented framework by developing a simple one-sensor, one-actuator industrial control example, distributed over three nodes that communicate over a single TTEthernet switch. The presented real-time system can exchange messages with minimal jitter as the distributed tasks are synchronized over the TTEthernet network with about 1.6 us precision. Due to the tight time synchronization, the system can operate stably with zero missed frames, using a single receiver and a single transmitter buffer.**

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

## I. Introduction

Modern safety-critical systems are often composed of distributed cyber-physical systems where applications tasks execute in different sub-systems. In such systems, both the communication and the task execution time become part of the critical end-to-end latency of the application, as transmitted frames often contain computation results that an actuator should consume at a precise moment in time, in-order and without missed data [1].

To achieve a high level of determinism, the synchronization of the task execution with the underlying communication layer would benefit the application. A typical real-time communication paradigm in industrial and safety-critical systems is the time-triggered protocol [2] and its Ethernet-based extension TTEthernet [3]. 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 that allows for sub-microsecond precision. TTEthernet is standardized under the aerospace standard AS6802 [4] and is used in the underdevelopment NASA Orion spacecraft [5].

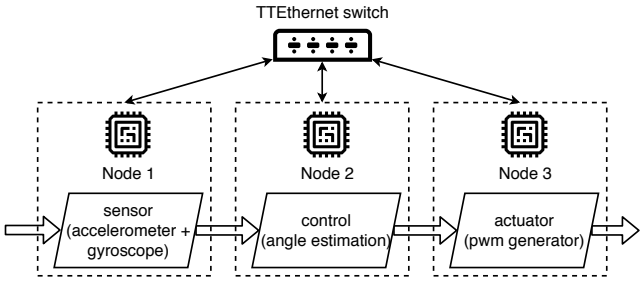


Fig. 1: Classic industrial control example with tasks distributed over TTEthernet network.

In this work, we follow the time-triggered communication paradigm and use TTEthernet as the underlying communication layer. Using a time-triggered communication paradigm does not only increase the application's level of determinism, but it also allows for precise end-to-end latency calculation, reduced jitter and relaxed end-system buffer requirements. System properties such as buffer usage can be statically estimated and decreased as the exact transmission/reception points in time are known during the development phase [6].

This paper investigates the problem of synchronizing the execution of real-time tasks with a time-triggered communication layer. This paradigm's key points are providing a stable time synchronization mechanism for the tasks and estimating the worst-case execution time (WCET) of the complete software stack. Static deterministic WCET analysis allows smaller pessimism in the WCET estimate than probabilistic or measurement-based methods of WCET [7]. Subsequently, this leads to improved resource utilization and end-to-end latency. The presented implementation uses the time-predictable T-CREST platform [8] and its WCET-optimized toolchain, which allows us to guarantee all timing properties, even in a distributed system setup. The paper extends the work-in-progress paper [9] and presents in-detail the design of a WCET analyzable open-source framework that achieves high precision task synchronization with short end-to-end communication latency, minimal jitter and buffer usage. We evaluate the proposed system by distributing a synthetic control application example of one-sensor and one-actuator, over three nodes, as shown in Figure 1.

The main contributions of this work are:

2375-5261/21/$31.00 ©2021 IEEE
DOI 10.1109/ISORC52013.2021.00013

- The integration of a TTEthernet communication system and open-source time-predictable computing nodes to an overall highly time-predictable distributed real-time system for applications that have tight deadlines and require microsecond end-to-end timing jitter.
- An open-source task scheduler that utilizes information about task dependencies and the TTEthernet communication schedule to generate cyclic executives for the time-predictable nodes that synchronize to the TTEthernet communication schedule.
- An experimental assessment of the proposed framework that demonstrates our approach's successful deployment using a time-predictable distributed sample application implemented within a standard TTEthernet network.

The rest of this paper is organized in 8 sections: Section II discusses the related work on time-triggered communication. Section III presents the system model of the task and network schedule. Section IV presents the proposed software framework and its implementation. Section V presents the synthetic control application example. Section VI evaluates the proposed design using the developed application. Section VII discusses the future improvements and plans. Section VIII concludes the paper.

## II. Related Work

This section reviews recent research related to distributed time-triggered networks and the challenges of synchronizing the task execution with an underlying communication schedule.

The benefits of synchronizing task execution with time-triggered network communication have been described by [6] where the authors compare an asynchronous, non-blocking communication interface with the synchronous, time-aware communication of the tasks of the computing nodes. They explain that the following two mechanisms must be implemented for time-aware systems : (b) a mechanism for adjusting the local clock and providing the synchronized global time to all computing nodes and (a) a real-time task scheduler. In this paper, we implement, describe and WCET-analyze these proposed mechanisms in detail. Moreover, we evaluate them over an experimental control application example.

The feasibility of task synchronization with the network schedule has been previously presented by [10]. However, the runtime system *TTE-RTS* used is proprietary, and thus the implementation and the WCET analysis were not presented. In contrast, we propose an open-source runtime system for synchronizing and communicating with a TTEthernet network.

The challenges of generating schedules with synchronized tasks and communication have been investigated by [11]. The authors discuss the simultaneous co-generation of static network and task schedules for distributed systems. Their task set consists of preemptive time-triggered tasks, prioritized by earliest deadline first and scheduled using satisfiability modulo theory (SMT). The authors proceed to optimize various properties of the system, such as end-to-end latency and buffer utilization using mixed integer programming solvers. Our work also uses SMT solvers but, in contrast to the previous work, our work uses a more straightforward cyclic executive scheduling policy. We focus on presenting the software framework's implementation details for synchronizing an application task schedule with the TTEthernet network schedule.

Different works have investigated further optimization of communication schedules. In [12], the authors pack multiple application messages in different time-triggered frames of selected order and length. In [13], the authors investigate the implementation of a basic AI fuzzy particle swarm algorithm for optimizing scheduling in high load TTEthernet networks. Such optimization methods are complementary to our work. Such optimization methods are complementary to our work.

The implementation of TTEthernet end-systems has been previously presented in the context of automotive real-time communication use-case for AUTOSAR in [14]. The implemented end-system acts as a synchronization client and the results show an observed jitter of 32 $\mu$s end-to-end latency jitter. Additionally, the authors provide metrics for the CPU utilization and the memory overhead of the end-system and discuss the non-determinism of the transmit and receive functions. In contrast, the system presented in this work uses a fully WCET analyzable software stack, and our system can synchronize the individual distributed tasks among the network with microseconds precision. The tight task synchronization bounds the evaluated end-to-end latency jitter

In [15], the authors investigate and develop a measurement technique for performance analysis of TTEthernet using commercial off-the-shelf tools. However, the related work software stack used for synchronizing and communicating with the TTEthernet network is based on proprietary drivers provided by TTTech. Thus no details on the implementation and its functionality are presented.

Finally, cyclic executive scheduling is a well-known engineering concept [16] that has also been implemented in multicore real-time systems [17], but to the knowledge of the authors, no work has investigated this concept in distributed systems. In contrast, we focus on implementing a cyclic executive synchronized with the underlying cyclic network communication.

## III. System Model

This section describes the model involved in synchronizing a distributed cyclic task execution with a time-triggered communication schedule, which is composed of two aspects.

### A. Network Model

TTEthernet follows the time-triggered protocol paradigm by extending it to IEEE 802.3 Ethernet networks to guarantee bandwidth and end-to-end latency.

Each network device (i.e., switches and end-systems) defines a critical traffic domain using a critical traffic (CT) marker for the frames and specifies communication flows called virtual links (VL). TTEthernet uses a cyclic communication schedule (called *TTE network schedule*) of the defined VLs, to transmit time-triggered frames within a scheduled *transmission window*. Subsequently, the reception of any CT frames is only accepted

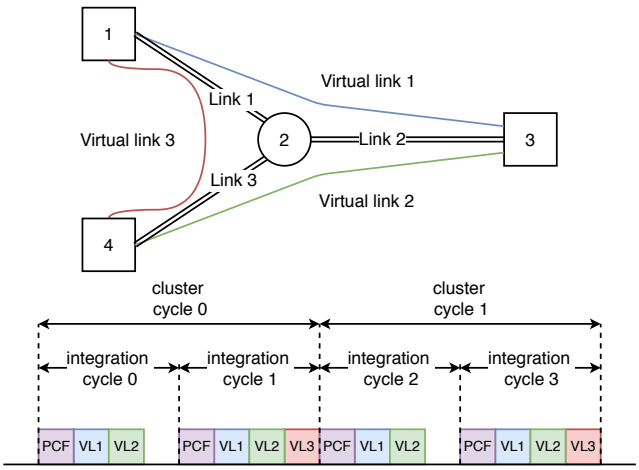


Fig. 2: Example three end-system TTEthernet network with three VLs. Communication *cluster cycle* comprising of two *integration cycles*.

within the network devices' scheduled *reception window*. Any frames that arrive outside the *reception window* are not forwarded. This way, collision-free and temporally isolated communication can be guaranteed. The cyclic transmission pattern in a *TTE network schedule* is repeated in hyper-periods called *cluster cycles*, as illustrated in Figure 2.

TTEthernet employs a global fault-tolerant clock synchronization algorithm [4] to align the *transmission window* and the *reception window* of the network's distributed end-systems. The synchronization protocol works periodically on iterations called *integration cycles* and a *cluster cycle* defining several *integration cycles*. At the start of each integration cycle, the TTEthernet end-systems exchange synchronization frames called protocol control frames (PCF). A PCF contains the accumulated time information of the transmission from a sender to a receiver. *Synchronization masters* transmit PCFs at fixed points in time to the connected switch. A TTEthernet switch with the role of *compression master* uses this information to calculate the global network time using a compression function, as described and analyzed in [18]. The switch transmits a compressed PCF at the beginning of each *integration cycle* that can be used by synchronization masters/clients to align their time with the network time, as shown in [19]. The number of *integration cycles* per *cluster cycle* controls the network's clock synchronization precision, and thus by configuring these parameters, the network can be configured to the desired clock precision as shown in [20].

### B. Task Model

In TTEthernet end-systems, applications use transmission and reception mechanisms implemented by proprietary network-interface hardware cards and drivers. In this work, we investigate and implement these mechanisms, in software, on an open-source end-system platform. Although we do not implement a real-time operating system, we develop a cyclic executive runtime system that executes tasks according to their scheduled release time and period. The system takes into consideration the clock offset calculated by the TTEthernet clock synchronization protocol to search for the next scheduled task to activate. The presented runtime system does not support task preemption as this facilitates the WCET analysis of the presented platform. Section IV discusses the runtime system implementation in detail.

In our task model, each task $\tau_i$ is defined 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 of the task, $O_i$ is a relative offset to the release time and $J_i$ is the maximum allowed jitter. We only consider tasks with harmonic periods, i.e., all periods of the tasks are an integer multiple of a shorter period. This allows scheduling tasks for zero allowed jitter $J_i = 0$ and is not a limitation of the proposed system rather than a design decision. The schedule's hyper-period 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. First, we constrain the release time to the period, deadline and relative offset as shown in Eq. 1c & 1b. Setting these constraints to predefined points in time, such as the *transmission* or the *reception window* of the *TTE network schedule* allows to set precedence constraints on the task execution and order the task release times accordingly.

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

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

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

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

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

Where $n, k$ are instances of the tasks $i, j$ respectively in the range of a hyper-period, $\forall i \neq j$.

## IV. Design and Implementation

This section presents the fundamental components of the proposed open-source framework, the hardware platform, the schedule generation and the design of the runtime system. It presents the methodology for generating the synchronized cyclic executive task schedule and the mechanisms involved in synchronizing the task execution with the underlying communication schedule.

### A. Hardware Platform

The presented system is implemented on the open-source research platform T-CREST with a few modifications. The platform features a time-predictable processor, Patmos [21]. Patmos is a dual-issue RISC processor that uses WCET-optimized

caches along with private scratchpad memories. A complete toolchain supports it with an LLVM-based compiler [22] and a static deterministic WCET analysis tool *platin* [23]. The platform also features a hardware-assisted timestamping unit [24] that measures the the arrival time and transmit time of Ethernet frames. The hardware timestamp unit, built to identify PTP frames, is modified to parse and identify the PCF frame format of the AS6802 [4] standard presented in Figure 3 and timestamp it at the start-of-frame (SOF) byte.

### B. Offline Scheduling

The synthesis of the task schedule based on the communication schedule resembles the process proposed by [10]. Figure 4 presents the general design flow of the proposed framework's fundamental blocks (in blue). It illustrates how the individual node task sets are defined using information from the application requirements, the network schedule and the WCET bounds of the task implementation.

First, the task set is defined, and the periods $T_i$ are constrained to the application's control requirements.

Second, the code of the application is developed. The WCET bounds of the implemented tasks and the runtime system's significant functions are calculated using the WCET analysis tool platin [23]. The WCET bounds are used as input to execution times $C_i$ in the task set definition.

Third, a network description is created according to the application's requirements using the TTTech development suite for configuring TTEthernet systems [25]. In this step, properties such as the synchronization domain, virtual links, maximum frame size and communication periods are defined. The network configuration is then generated using the *TTE-Plan* tool, which contains the *TTE network schedule*. The network configuration is then compiled to the individual configuration files for TTTech built end-systems and switches using the *TTE-Build* tool.

Finally, the cyclic *task schedule* is generated based on the task definition, the WCET analysis bounds and uses the transmission and reception points defined in the *TTE network schedule* as precedence constraints to the equations presented in Section III. More specifically, the WCET is used as input to each task's execution time $C_i$ and the transmission/reception time slots, defined by the *TTE network schedule*, are used as input to the activation times $S_{i,n}$ of each related task. To synchronize any computation tasks relative to the transmission/reception tasks, the receive and transmit time slots defined in the *TTE network schedule* are mapped to the offset $O_i$ and the deadline $D_i$ of the computation task. The total utilization of the defined task set is tested before scheduling according to $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$. A custom SMT Python script is developed to generate a cyclic executive schedule synchronized to the *TTE network schedule* using the Z3 Theorem Prover (SMT solver) [26].

### C. Transmission and Reception

A transmission task encapsulates the data message in a TTEthernet compatible link-layer protocol frame format that specifies the correct CT marker and VL. The frame is then transmitted using a non-blocking call to the Ethernet controller that returns a success boolean. The frame must arrive within the receive acceptance window of the connected switch, and in the correct VL; otherwise, the frame is dropped. The start of the sending task is set to a bit earlier than the start of the frame transmission, offset by its WCET, as illustrated in Figure 5.

The reception of CT frames on each VL is handled by respective tasks scheduled periodically at each receive point in time defined by the *TTE network schedule*. Each reception task is listening for a predefined *reception-window* time duration. During this time window, the receive function polls the Ethernet controller for any received frames that match the expected frame Ethernet type. The polling is active for a predefined duration of time. The reception window time duration is specified during the generation of the *TTE network schedule*.

Listing 1: Time-triggered task type definition.

```
typedef struct
{
  unsigned long long period;
  unsigned long long *releases;
  unsigned long act_inst;
  unsigned long nr_releases;
  unsigned long long last_time;
  unsigned long long delta_sum;
  unsigned long exec_count;
  generic_task_fp task_fp;
} SimpleTTETask;
```

### D. Runtime System

In the proposed runtime system, tasks are defined as a simple C structure shown in Listing 1. Each task has the following functional properties: a period, an array of release times, the current release time index, the number of releases and a function pointer. Additionally, each task keeps track of the following properties for quality control: the last time it was executed, the sum of delta times (the difference between the current and the last time the task executed) and the number of times it was executed. The task set is defined globally as an array of `SimpleTTETask` type variables, as presented in the example shown in Listing 2. The release times of each task are initialized according to the generated offline schedule.

The cyclic execution dispatcher of the task set is defined as a loop function illustrated in Listing 3. This function is called after the program has initialized fully, i.e., configured the Ethernet controller, initialized the task set according to the scheduled release times and allocated the communication message buffers. The function takes as argument a pointer to the initialized task set.

The dispatcher searches through the task set for an upcoming release time using the TTEthernet synchronized time. When a task is found, the system proceeds to execute activated task's function call and subsequently update its release time by adding the schedule hyper-period. The rest of the fields are updated accordingly. It is worth noting that before executing the executive loop, the task set is ordered according to the initial release time values. This eliminates unnecessary search

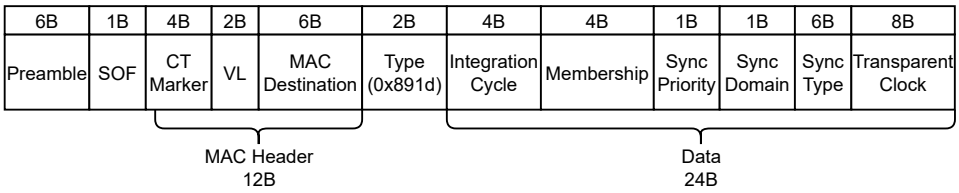


Fig. 3: TTEthernet (AS6802) PCF Format.

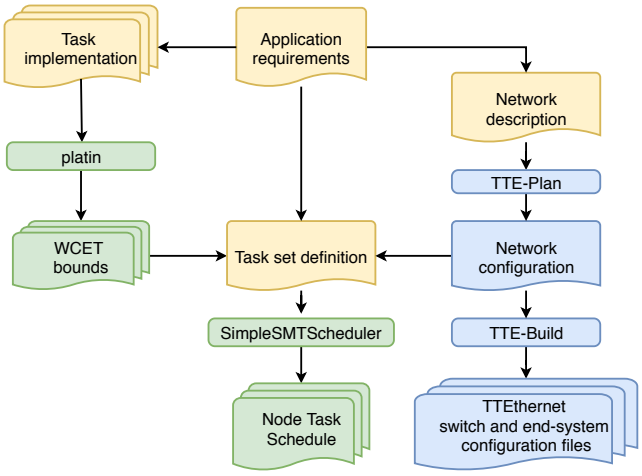


Fig. 4: Design flow for synchronized communication and task schedule generation. Yellow represents the application specific definitions, green represents the components/tools of the proposed framework and blue represents the tools provided by TTTech.

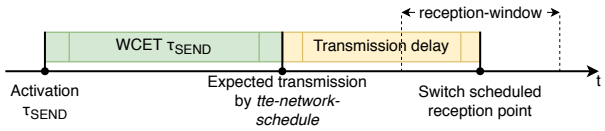


Fig. 5: Task activation point in time relationship to the scheduled *transmission-window*.

queries since after a task has been activated, the loop can safely break and take a new time reading to begin a new search.

### *E. Clock and Task Synchronization*

The presented system acts as a synchronization client to the TTEthernet network. It synchronizes with the network time by scheduling a periodic task responsible for handling incoming PCFs. The task is scheduled to execute at specific points in time according to the *TTE network schedule* and is responsible for calculating the clock offset according to the *permanence function* [4]. According to the *permanence function*, the clock offset is calculated as the difference between the scheduled receive point in time and the actual reception timestamp of the incoming PCF, plus the difference between a maximum transparent clock value and the transparent clock information found in the PCF [19]. The calculated clock

Listing 2: Example task set definition of actuator node.

```
static SimpleTTETask sched[NR_TASKS] = {
  {
    .period = 10000000,
    .releases = {0},
    .act_inst = 0,
    .nr_releases = 2,
    .last_time = 0,
    .delta_sum = 0,
    .exec_count = 0,
    .task_fp = (generic_task_fp)task_sync
  },
  {
    .period = 5000000,
    .releases = {1200000, 6200000},
    .act_inst = 0,
    .nr_releases = 1,
    .last_time = 0,
    .delta_sum = 0,
    .exec_count = 0,
    .task_fp = (generic_task_fp)task_recv
  },
  {
    .period = 5000000,
    .releases = {1471107, 6471107},
    .act_inst = 0,
    .nr_releases = 2,
    .last_time = 0,
    .delta_sum = 0,
    .exec_count = 0,
    .task_fp = (generic_task_fp)task_ctrl
  },
  {
    .period = 5000000,
    .releases = {3571700, 8571700},
    .act_inst = 0,
    .nr_releases = 2,
    .last_time = 0,
    .delta_sum = 0,
    .exec_count = 0,
    .task_fp = (generic_task_fp)task_send
  }
};
```

offset is not used directly to modify the hardware clock. Instead, the value is stored and used by calling the function `get_tte_aligned_time()`, which accepts a time value and returns the synchronized time after applying a proportional-integral filter similar to [27], [28].

The synchronization task accepts a parameter pointer to the defined task set and is responsible for updating the release times based on the calculated clock offset. This is necessary because after executing a synchronization task, the next dispatch loop will use the newly aligned time to query any upcoming task activations. If the task release times are not updated accordingly, it can cause an activation point to be considered in the past or

Listing 3: Code excerpt from runtime cyclic execution loop.

```
void executive_loop(SimpleTTETask* sched)
{
  uint64_t start_time = get_rtc_nanos();
  while(1){
    uint64_t sched_time = get_tte_aligned_time(
          get_rtc_nanos() - start_time);
    for (int i = 0; i < NR_TASKS; i++) {
      if (sched_time >= sched[i].activate)
      {
        sched[i].task_fp(/*arguments*/)
        sched[task].releases[sched[task].act_inst] +=
          HYPERPERIOD;
        sched[task].act_inst =
          (sched[task].act_inst + 1) %
          sched[task].nr_releases;
        sched[i].delta_sum += sched_time -
          get_tte_aligned_time(
            sched[i].last_time);
        sched[i].last_time = sched_time;
        sched[i].exec_count += 1;
        break;
      }
    }
  }
}
```

the present and thus violate the task period. This miss-alignment is demonstrated in Figure 6.

## V. Example Application

To evaluate the presented framework, we define and implement a simple control application that reads the input from a motion processing unit to determine its attitude/orientation and control servo motor's rotation. The application comprises one sensor and one actuator distributed over three nodes: a *sensor node*, a *control node* and an *actuator node*. Similar multi-periodic control systems can be found in various safety-critical applications including flight controllers [29].

The *sensor node* interfaces with a motion processing unit sensor MPU-9250 [30], which features an inertia measurement unit (IMU), a gyroscope and magnetometer. The sensor is sampled by reading alternating measurements from either the IMU or the gyroscope at a sampling frequency of 100 Hz. The values are transmitted 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 sent to the *actuator node*. The *control node* calculates the motion processing unit sensor's angle on the X-axis by fusing the accelerometer's and the gyroscope's measurements of the using a complimentary filter (see Equation 3a) [31]. The angle is then converted to a valid duty-cycle range according to Equation 3b.

$$\theta_x = 0.93 * (\theta_x + gyro_x * dt) + 0.07 * atan2\,(accel_y, accel_z) \tag{3a}$$

$$duty\_cycle = \frac{\theta_x * (0.1 - 0.015)}{180} + 0.015 \tag{3b}$$

The *actuator node* drives a servo motor using pulse-width modulation (PWM) signal. 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 sets 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. Task set

The following task set is derived based on the presented application's description and requirements. The *sensor node* executes three tasks:

1) $\tau_{SSYNC}$ synchronizes with the TTEthernet network time
2) $\tau_{SENSE(a/g)}$ collects alternating measurements from either the IMU sensor (a) or the connected gyroscope sensor (g)
3) $\tau_{SSEND}$ transmits the sensor values to the control node

The *control node* executes four tasks:

1) $\tau_{CSYNC}$ synchronizes with the TTEthernet network time
2) $\tau_{CRECV}$ receives the read sensor measurement
3) $\tau_{CTRL}$ calculates the angle of the MPU-9250 sensor and computes a valid duty-cycle value
4) $\tau_{CSEND}$ transmits the computed duty-cycle result

The *actuator node* executes four tasks:

1) $\tau_{ASYNC}$ synchronizes with the TTEthernet network time
2) $\tau_{ARECV}$ receives the control instructions computed from the control node
3) $\tau_{PWM}$ produces the pulse-width modulation to move the interfaced actuator

### B. Source Access

All the components of the presented framework are open-source. The SMT scheduler for the task generation is hosted at *https://github.com/egk696/SimpleSMTScheduler* The implemented runtime system is integrated with the T-CREST platform and the developed application is hosted at *https://github.com/t-crest/patmos/tree/master/c/apps/ttecps*

## VI. Evaluation

### A. System Setup

The cyclic executive's proposed synchronization with the communication schedule is evaluated and tested experimentally using a synthetic control application example of one-sensor, one controller, and one-actuator distributed over three nodes. The nodes are integrated, as synchronization clients (SC), in an existing TTEthernet network star topology and assume the different roles described in Section V by executing the proposed runtime system. The hardware platform is synthesized on three FPGA Terasic DE2-115 boards [32] and operates at a frequency of 80 MHz. The network consists of a single industrial TTE Chronos 18/6 Rugged Switch acting as a compression master (CM) and two Linux desktops equipped with TTEthernet capable PCI Ethernet cards acting as synchronization masters (SM). Figure 7 presents the network setup of the evaluated control application example. A similar network setup has been described in [33].

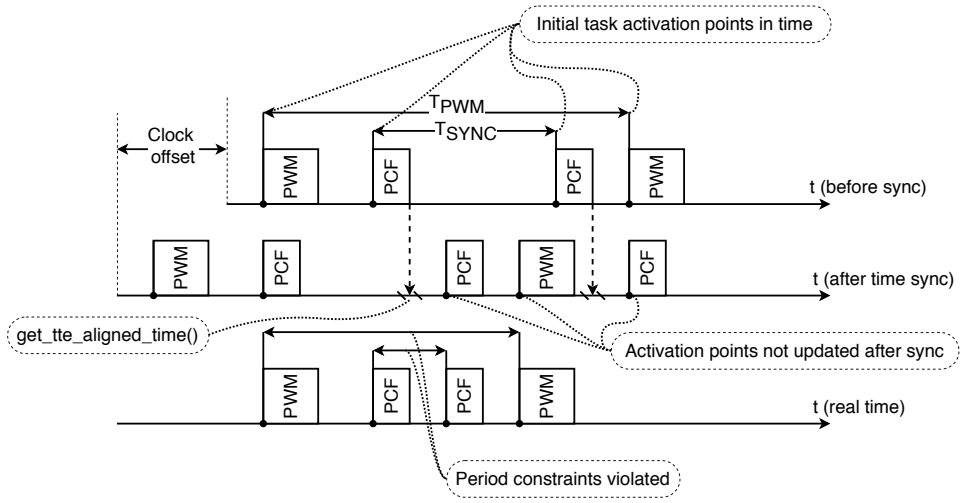


Fig. 6: Erroneous task execution example, due to a miss-alignment between the time-base of the dispatcher schedule time and the upcoming task release times.

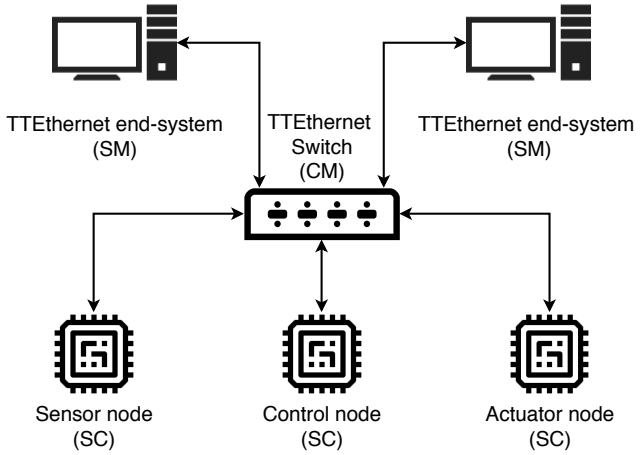


Fig. 7: Experimental network setup of the evaluated control application example.

### B. WCET Analysis and Schedule Generation

It is necessary to perform a static WCET analysis on the runtime system's significant functions and the task set to reveal possible jitter sources and accurately schedule and synchronize the task execution. We analyze the developed software using the tool *platin* and present the results for implementing and evaluating the experimental setup.

Table I presents the significant functions of the proposed runtime system in clock cycles. It is worth noting that the sorting function, `sort_ttetasks()`, depends on the number of tasks scheduled; thus, the WCET varies depending on the task set. The static WCET bound of the runtime dispatcher `executive_loop()` is used to set the $\Phi_i = 44.737$ $\mu$s constant, which is used in the scheduling constraints (see Section III).

Table II presents the combined generated task set using the

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

| Function | Node | | |
|---|---|---|---|
| | Sensor | Control | Actuator |
| `sort_ttetasks` | 6607 | 24123 | 13908 |
| `get_tte_aligned_time` | | 129 | |
| `executive_loop` | | 3579 | |

presented SMT scheduler of the three nodes' distributed tasks. As discussed in Section IV, the receive and transmit points are generated using the *TTEtools* and provided to the scheduler as constraints for the respective tasks initial activation point $S_{i,\emptyset}$ (indicated by an asterisk). The activation times of the transmission functions are offset by the WCET bounds of the respective tasks. The *reception-* and *transmission-window* of the *TTE network schedule* is configured at 20 $\mu$s, and this is added to the WCET of the related tasks: $\tau_{SSYNC}$, $\tau_{CSYNC}$, $\tau_{ASYNC}$, $\tau_{CRECV}$ and $\tau_{ARECV}$. It is worth noting that the WCET of the synchronization tasks varies in each node as it depends on the number of task release times that it has to update. According to the specification of the generated PWM for the servo motor, the total execution time of task $\tau_{PWM}$ can vary. In the presented analysis, it is considered that task $\tau_{PWM}$ generates the maximum duty-cycle duration of 2 ms (0.1%), which is added in the WCET of the task $\tau_{PWM}$.

The processor utilization of the three task sets for the *sensor node*, the *control node* and the *actuator node* is 4.5%, 16.7% and 16.6% respectively. The developed SMT scheduler is executed on an Intel Core i7-7700HQ CPU (2.80 GHz) and requires 17.47 ms, 20.04 ms and 10.07 ms to find a solution for each of the three nodes task sets: *sensor node*, *control node* and *actuator node* respectively.

TABLE II: Generated task set of the three end-systems. The asterisks indicate a constrain by the *TTE network schedule*.

| Node | Task | Period ($\mu$s) | WCET ($\mu$s) | $S_{i,\emptyset}$ ($\mu$s) |
|---|---|---|---|---|
| sensor | $\tau_{SSYNC}$ | 10000 | 90.550 | 0 |
| | $\tau_{SENSE(a/g)}$ | 5000 | 153.412 | 67.220 |
| | $\tau_{SSEND}$ | 5000 | 23.200 | (*) 771.700 |
| control | $\tau_{CSYNC}$ | 10000 | 90.550 | 0 |
| | $\tau_{CRECV}$ | 5000 | 285.450 | (*) 1200.000 |
| | $\tau_{CTRL}$ | 5000 | 485.05 | 1471.107 |
| | $\tau_{CSEND}$ | 5000 | 23.200 | (*) 3571.700 |
| actuator | $\tau_{ASYNC}$ | 10000 | 90.550 | 0 |
| | $\tau_{ARECV}$ | 5000 | 285.450 | (*) 4000.000 |
| | $\tau_{PWM}$ | 20000 | 2005.462 | 4271.107 |

### C. Communication and Clock Synchronization

To evaluate the correctness of the presented runtime system as well as to emphasize the precision of the synchronization, each node's Ethernet controller is configured with a single transmit and a receive buffer. This way any packets that are not captured in-time within the *reception-window* are overwritten and dropped. The system was tested for a timespan of 24-hours and with zero missed frames recorded.

The clock synchronization precision of the presented distributed system, is evaluated by generating an I/O pulse during the synchronization tasks' execution ($\tau_{SSYNC}$, $\tau_{CSYNC}$, $\tau_{ASYNC}$) and measuring the relative time offset of the pulses using a digital logic analyzer. Both the clock synchronization relative to the TTEthernet switch and the synchronized schedule execution between the nodes were evaluated. The maximum measured relative time offset between the three nodes' task I/O pulses was $\approx$1.6 $\mu$s while the individual synchronization accuracy of each node relative to the TTEthernet switch was measured at $\approx$136 $\mu$s.

Finally, we consider the end-to-end latency of the presented distributed system as the time difference between when the system measures the physical world on the *sensor node* to when the new duty-cycle is consumed by the $\tau_{PWM}$ on the *actuation node* shown in Equation 4.

$$L_{e2e} = S_{\tau_{PWM},\emptyset} - (S_{\tau_{SENSE},\emptyset} + WCET_{\tau_{SENSE(a/g)}}) \quad (4)$$

The end-to-end latency is statically calculated at 4.05 ms. We verify the calculated bound experimentally by comparing a time-difference of the timestamps between the sensor readout and the value's extraction by the PWM generation task. We measure this time-difference at $\approx 4.034$ ms. The task $\tau_{PWM}$ consumes the new duty-cycle well within the required deadline for the PWM period of the servo motor.

The presented evaluation emphasized that tight synchronization of transmission/reception tasks with the communication schedule is essential to software-based TTEthernet end-system's operation. Moreover, although the synchronization of computation tasks to the communication is not functionally required, it is beneficial to the overall end-to-end latency of a real-time distributed system. Using the evaluated control application as an example, we calculate and measure that if the sensor reading task $\tau_{SENSE(a/g)}$ were not synchronized with the transmission task $\tau_{SSEND}$, the worst-case end-to-end latency would be increased by half the period the next scheduled transmission slot. In some hard real-time distributed systems, the end-to-end latency requirements are in the range of a few milliseconds [5], and thus an increase of $\approx 2.5$ ms could be intolerable.

## VII. Future Work

To relax the restrictions of a cyclic executive on a single core, we plan to extend the presented framework to a multicore version that will allow us to dedicate a single core to handle the TTEthernet traffic. Inter-core communication can be handled using time-division multiplexing (TDM) network-on-chip such as [34], [35], [36]. TDM-based network-on-chip use cyclic schedules similar to TTEthernet but with different resolutions. An interesting research challenge arises regarding the synchronization of these schedules that theoretically can improve the end-to-end latency and jitter of messages transmitted via both communication channels [37].

The presented framework and runtime system is not dependent on a specific communication protocol. Thus we plan to investigate its implementation within time-sensitive networks (TSN), which have shown promising results in supporting mixed-criticality industrial applications together with time-triggered communication [38], [39].

## VIII. Conclusion

This paper investigated the concept of synchronizing the task execution in a real-time distributed system with the time-triggered communication schedule in a time-aware network and presented an open-source and WCET analyzable software framework.

First, the problem was explored by describing the system model comprising the network and the task model. Subsequently, an open-source SMT scheduler was developed that utilizes information regarding task dependencies and the communication schedule to generate a cyclic executive for a time-predictable node. An open-source runtime system was developed and integrated with a time-predictable open-source research platform, and the overall design process was presented in detail.

The developed framework was evaluated by developing and successfully deploying a synthetic distributed control application of one sensor, one controller, and one actuator over a TTEthernet network with three nodes. The task schedule synchronization with the communication schedule was emphasized by configuring the nodes to use only one receive, and one transmit buffer. A full static WCET analysis was performed on the tasks as well as the significant parts of the runtime system. The individual cyclic executives of the nodes were synchronized relative to each to a measured precision of $\approx 1.6$ $\mu$s and the end-to-end latency was bounded at $\approx 4.05$ ms.

Overall, we demonstrated the feasibility of precise task synchronization with time-triggered communication using a COTS open-source TTEthernet framework and presented a synthetic distributed control application example.

*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] H. Kopetz and G. Grunsteidl, "Ttp-a time-triggered protocol for fault-tolerant real-time systems," in *FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing*. IEEE, 1993, pp. 524–533.

[3] W. Steiner and G. Bauer, "Ttethernet: Time-triggered services for ethernet networks," in *Proceedings of 28th IEEE Digital Avionics Conference (DASC)*, 2009.

[4] TTTech, *AS6802: Time-Triggered Ethernet*, SAE International Std., 2011.

[5] M. Paulitsch, E. Schmidt, C. Scherrer, and H. Kantz, "Industrial applications," in *Time-Triggered Communication*. CRC Press, 2018, ch. 14, pp. 315–333.

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

[7] J. Abella, C. Hernandez, E. Quiñones, F. J. Cazorla, P. R. Conmy, M. Azkarate-Askasua, J. Perez, E. Mezzetti, and T. Vardanega, "Wcet analysis methods: Pitfalls and challenges on their trustworthiness," in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*. IEEE, 2015, pp. 1–10.

[8] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, S. Hepp, B. Huber, A. Jordan, E. Kasapaki, J. Knoop, Y. Li, D. Prokesch, W. Puffitsch, P. Puschner, A. Rocha, C. Silva, J. Sparsø, and A. Tocchi, "T-CREST: Time-predictable multi-core architecture for embedded systems," *Journal of Systems Architecture*, vol. 61, no. 9, pp. 449–471, 2015.

[9] E. Kyriakakis, J. Sparsø, P. Puschner, and M. Schoeberl, "Synchronizing real-time tasks in time-aware networks: Work-in-progress," in *2020 International Conference on Embedded Software (EMSOFT)*. IEEE, 2020, pp. 15–17.

[10] 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.

[11] S. S. Craciunas and R. S. Oliver, "Combined task-and network-level scheduling for distributed time-triggered systems," *Real-Time Systems*, vol. 52, no. 2, pp. 161–200, 2016.

[12] D. Tămaş-Selicean, P. Pop, and W. Steiner, "Design optimization of ttethernet-based distributed real-time systems," *Real-Time Systems*, vol. 51, no. 1, pp. 1–35, 2015.

[13] H. Chen, L. Wang, P. Shen, and J. Di, "Static schedule generation for time-triggered ethernet based on fuzzy particle swarm optimization," *Chinese Journal of Electronics*, vol. 28, no. 6, pp. 1250–1258, 2019.

[14] T. Fruhwirth, W. Steiner, and B. Stangl, "TTEthernet sw-based end system for AUTOSAR," *2015 10th Ieee International Symposium on Industrial Embedded Systems, Sies 2015 - Proceedings*, pp. 21–28, 2015.

[15] F. Bartols, T. Steinbach, F. Korf, and T. C. Schmidt, "Performance analysis of time-triggered ether-networks using off-the-shelf-components," in *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*. IEEE, 2011, pp. 49–56.

[16] A. Burns and A. J. Wellings, *Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX*. Pearson Education, 2001.

[17] A. P. Ravn and M. Schoeberl, "Cyclic executive for safety-critical Java on chip-multiprocessors," in *Proceedings of the 8th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES 2010)*. New York, NY, USA: ACM, 2010, pp. 63–69.

[18] W. Steiner and B. Dutertre, "The ttethernet synchronisation protocols and their formal verification," *International Journal of Critical Computer-Based Systems 17*, vol. 4, no. 3, pp. 280–300, 2013.

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

[20] M. Sandić, I. Velikić, and A. Jakovljević, "Calculation of number of integration cycles for systems synchronized using the as6802 standard," in *2017 Zooming Innovation in Consumer Electronics International Conference (ZINC)*. IEEE, 2017, pp. 54–55.

[21] 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.

[22] C. Lattner and V. S. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *International Symposium on Code Generation and Optimization (CGO'04)*. IEEE Computer Society, 2004, pp. 75–88.

[23] 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.

[24] E. Kyriakakis, J. Sparsø, and M. Schoeberl, "Hardware assisted clock synchronization with the ieee 1588-2008 precision time protocol," in *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, 2018, pp. 51–60.

[25] TTTech, "TTETools – TTEthernet Development Tools v4.4." [Online]. Available: https://www.tttech.com/products/aerospace/development-test-vv/development-tools/tte-plan/

[26] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[27] E. Kyriakakis, M. Lund, L. Pezzarossa, J. Sparsø, and M. Schoeberl, "A time-predictable open-source ttethernet end-system," *Journal of Systems Architecture*, p. 101744, 2020.

[28] G. Giorgi and C. Narduzzi, "Modeling and simulation analysis of PTP clock servo," in *2007 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, 2007.

[29] C. Pagetti, J. Forget, F. Boniol, M. Cordovilla, and D. Lesens, "Multi-task implementation of multi-periodic synchronous programs," *Discrete event dynamic systems*, vol. 21, no. 3, pp. 307–338, 2011.

[30] *MPU-9250 Product Specification*, InvenSense, June 2016. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf

[31] 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.

[32] *Terasic DE2-115 User Manual*, Altera, March 2016. [Online]. Available: https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-1404062209-de2-115-user-manual.pdf

[33] P. Lambert, "D11.4 –final demonstrator implementation and evaluation," EMC2 Project Consortium,, Tech. Rep., 4 2017. [Online]. Available: https://www.artemis-emc2.eu/fileadmin/user_upload/Publications/Deliverables/EMC2_D11.4_WP11_Final_demonstrator_implementation_and_evaluation_v1.0.pdf

[34] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal network on chip: concepts, architectures, and implementations," *IEEE Design & Test of Computers*, vol. 22, no. 5, pp. 414–421, 2005.

[35] E. Kasapaki, M. Schoeberl, R. B. Sørensen, C. T. Müller, K. Goossens, and J. Sparsø, "Argo: A real-time network-on-chip architecture with an efficient GALS implementation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, pp. 479–492, 2016.

[36] E. Nilsson, "Design and implementation of a hot-potato switch in a network on chip," *Mémoire, Departement of Microelectronics and Information Technology, Royal Institute of Technology*, 2002.

[37] E. Kyriakakis, J. Sparsø, and M. Schoeberl, "Internoc: Unified deterministic communication for distributed noc-based many-core," in *th Junior Researcher Workshop on Real-Time Computing*, 2019.

[38] 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.

[39] V. Gavriluţ and P. Pop, "Scheduling in time sensitive networks (tsn) for mixed-criticality industrial applications," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*. IEEE, 2018, pp. 1–4.