Evaluating a Time-Triggered Runtime System by Distributing a Flight Controller

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

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

2021 IEEE 26th International Conference on Emerging Technologies and Factory Automation (ETFA) | 978-1-7281-2989-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/ETFA45728.2021.9613679

Abstract—**With the recent advancements in the Industrial Internet of Things and Industry 4.0, cyber-physical systems have become increasingly inter-connected. It is becoming a challenge to maintain the same quality-of-control and time-predictability of computation and communication required by safety-critical hard real-time systems as previously achieved through non-distributed architectures.**

This paper examines the problem of implementing and distributing a closed-loop command-control system over an Ethernet network with guaranteed timing bounds. To achieve bounded communication and computation time, we use an open-source software framework running on the T-CREST platform combined with a TTEthernet network star topology. We evaluate its quality-of-control performance in our experimental setup and compare the results against single-core and multi-core implementations. The proposed distributed time-triggered runtime system executes with jitter below $10\mu s$ and can perform a stable flight scenario as verified by the benchmark implementation.

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

I. Introduction

Safety-critical application domains like avionics, automotive, and Industrial Internet of Things can benefit from the design of computing systems that provide bounded timing in both computation and communication as this facilitates certification due to the guaranteed end-to-end temporal behavior [1]. However, as real-time systems become more interconnected, deploying multi-rate time-critical task chains on distributed systems and maintaining the same time-predictability becomes a challenge [2].

One approach to guarantee timing closure of distributed safety-critical applications is to employ the synchronous dataflow [3] model on time-predictable hardware architectures [4]. Following the synchronous data flow model allows an application to define the number of produced and consumed data samples at each given point-in-time during the design phase. This model enables the static analysis of all the timing properties of a design as well as memory, processor and network load since both communication and computation resource usage can be statically scheduled at design time.

Time-predictable platforms can enable static worst-case execution time (WCET) analysis for computation by providing WCET-optimized hardware design, such as in-order processor pipeline and support for scratchpad memories. Additionally, they aim to minimize end-to-end communication jitter and latency by providing a synchronized interface to the underlying communication layer [5], [6]. Different scheduling mechanisms must be implemented for both the communication and computation layer of an application to support the synchronous data flow model. Various policies can be used to generate static schedules that enable static execution of time-critical tasks on the processor, such as cyclic execution, earliest deadline first, and fixed priority [7]. Regarding communication, several industrial protocols have been developed to enable bounded network latency and temporal isolation of communication flows, such as TSN, TTEthernet, and PROFINET [8], [9].

The purpose of this work is two-fold: first, to experimentally demonstrate the distribution of open-source avionic control/-command case study and, secondly to evaluate the open-source time-triggered framework proposed by [10] as well as the underlying communication layer. We achieve this by implementing a realistic case study of a flight controller on a time-predictable processor Patmos [11], and distributing the application using a time-triggered communication protocol TTEthernet [12]. The main contributions are:

- A case study of a flight management system in a distributed implementation.
- An experimental evaluation of an open-source time-triggered runtime system.
- A comparative analysis between the quality of control of the proposed time-triggered implementation against single-core and multi-core implementations.

The rest of this paper is organized into eight sections: Section II presents the use-case application of a flight management system. Section III presents a background on the technologies and tools employed by this work. Section IV presents the design and implementation of the distributed flight management system using the proposed runtime system. Section V evaluates the proposed design using the developed application. Section VI discusses related work on time-triggered communication and relevant use-cases. Section VIII

TABLE I: Flight controller closed-loop variables

Entity	Variable name	Description
Reference Inputs	h_c	commanded altitude
	V_{a_c}	commanded true airspeed
Aircraft dynamics	V_z	vertical speed
	V_a	true airspeed
	h	altitude
	a_z	vertical acceleration
	q	pitch rate
Filtered measurements	V_{z_f}	vertical speed
	V_{a_f}	true airspeed
	h_f	altitude
	a_{z_f}	vertical acceleration
	q_f	pitch rate
Control outputs	V_{z_c}	vertical speed command
	δ_{e_c}	elevator deflection command
	δ_{th_c}	throttle change command
Aircraft Inputs	δ_{e_c}	elevator deflection
	T	engine thrust

summarizes the main conclusions derived from this work.

II. Use-Case: ROSACE Longitudinal Flight Controller

This section describes the case study of the Research Open-Source Avionics and Control Engineering (ROSACE) longitudinal flight controller and its control constraints along with its hard real-time specification.

The case study developed and analyzed by Pagetti et al. [13] describes a multi-rate longitudinal flight controller operating on a medium-sized aircraft that is in the en-route phase at a starting altitude of 10000 m. The study investigates a flight management system for the scenario of a 1000 m step climb command. During the climb, the autopilot flight management system maintains a constant ascend rate V_z while preserving a constant airspeed V_a and achieving a stable flight at the commanded altitude h. Similar flight-level changes are often performed in real life for fuel economy or maintain altitude separation from ongoing air traffic routes.

The flight controller in use has been designed in SIMULINK [14] as a closed-loop system and is divided into two logical parts: (a) the system that simulates the aircraft, elevator, and engine dynamics and (b) the controller that includes the control loops (`altitude_hold`, `Vz_control`, `Va_control`) and a collection of filters, which aim to model the sensor data acquisition. Table I lists the variables involved in the operation of the flight controller.

The case study provides a set of four validation objectives (P1, P2, P3, P4) for the step response of the V_a and V_z loops to the input command for the climbing scenario. P1 is the amount of time required for the controlled variable to settle within 5% of the steady-state value. P2 examines the overshoot as the maximum value attained minus the steady-state value. P3 is the time duration it takes for the value to rise from 10% to 90% of the steady-state value. Lastly, P4 is the steady-state error, which is the difference between the input and the output as $t \to 0$. These validation objectives are used in Section V to drive the evaluation of the presented real-time system implementation and compare it against other non-distributed implementations.

III. Background

This section provides an overview of the relevant technologies and tools that this work integrates.

A. Time-triggered Communication

For the network communication, we integrate the nodes in a TTEthernet network. TTEthernet is based on a cyclic communication schedule (called *TTE network schedule*) that specifies periodic communication flows called virtual links (VL) to transmit time-triggered frames within a scheduled *transmission window*. Subsequently, the reception of any frames is only accepted within the network devices' scheduled *reception window*. Any frames that arrive outside the *reception window* are dropped and not forwarded by devices. The cyclic transmission pattern of the *TTE network schedule* repeats in hyper-periods called *cluster cycles*.

Network time synchronization is required for all hosts to synchronize their communication to the respective *transmission/reception windows*. TTEthernet achieves this by exchanging unique Ethernet 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 their connected switch. Typically, each switch assumes the role of a *compression master* that uses this information to calculate the global network time offset using a compression function, as described and analyzed in [15]. The switch broadcasts a new PCF to all connected devices at the beginning of each *integration cycle* that can be used to align their local time with the network time.

B. Offline Scheduler

The task and network schedules are synthesized using the open-source framework described by [10]. The authors present a static scheduler for synthesizing time-triggered schedules using constraint programming. They present a custom Python application developed to generate a cyclic executive schedule synchronized to the *TTE network schedule* using the Z3 satisfiability modulo theory (SMT) Prover/Solver [16].

C. Runtime System

This work aims to evaluate and deploy the open-source runtime system presented in [10]. Commercial off-the-shelf TTEthernet applications rely on transmission, reception and synchronization mechanisms implemented on proprietary hardware. The implemented runtime system performs this operations completely in software.

The runtime system is responsible for controlling the execution policy of the tasks and the communication with the underlying TTEthernet network. The system deploys a cyclic dispatcher to execute tasks based on their release time. Each task is defined as a simple C structure that maintains a period, an array of release times, the current release index and the total number of releases. The dispatcher searches through an array of tasks for an upcoming release time using the TTEthernet synchronized time as a parameter. After executing a task, its period is increased by the hyper-period of the schedule and the current release index points to the next release time. The proposed runtime system does not support task preemption as this facilitates the WCET analysis of the presented platform. Instead best-effort tasks such as the `LOGGING` task can be scheduled with relaxed jitter constraints and are executed in a time-triggered fashion.

D. Hardware Platform

The presented system is implemented on the open-source research platform T-CREST [17]. The platform features a time-predictable processor, Patmos [11], that uses WCET-optimized caches along with private scratchpad memories. Additionally, we use its toolchain, particularly the static WCET analysis tool *platin* [18], to derive the execution time constraints for our scheduler. The platform is also equipped with an Ethernet controller that features a hardware-assisted timestamping unit [19]. The timestamping unit measures the arrival time and transmit time of PTP and PCF Ethernet frames by monitoring the MII interface between the PHY and the Ethernet controller.

IV. System Design and Implementation

This section describes the system model involved in distributing a flight management system in a TTEthernet network using a synchronized cyclic task execution runtime environment.

A. Task and Network Model

In this work, we use the offline scheduler described in Section III, and thus we model each task τ_i as a 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 also model the communication of VL flows as periodic tasks that can be constrained by the transmission time of the VL together with the WCET of the end-system software responsible for transmitting or receiving the frame. The communication tasks are then integrated with the cyclic task set of each node to derive valid transmission and reception points-in-time for the network schedule. The start of each sending task is offset a bit earlier than its scheduled point-of-transmission to account for the copy of the data into the Ethernet controller's buffer. Finally, we model the PCF reception and clock synchronization as a periodic task since it is handled in software.

B. Communication

We distribute ROSACE over three nodes, as shown in Figure 1. Node 1 is responsible for simulating the aircraft dynamics, engine, and elevators and provides raw data of the aircraft state. Node 2 is responsible for filtering the aircraft state. Node 3 generates the control commands for the aircraft. Node 4 is a TTEthernet switch. The communication takes

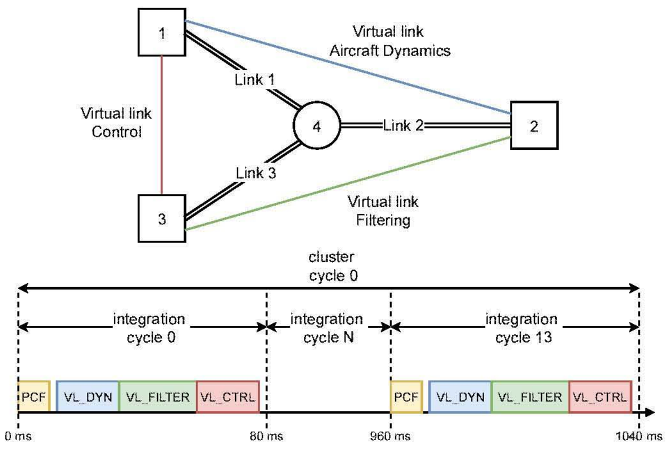

Fig. 1: Distributed ROSACE topology and example TTEthernet VL communication schedule.

place over three VLs between each of the three TTEthernet nodes, and a total of 21 tasks are distributed over the nodes, which include communication and computation.

We implement dedicated periodic transmission and reception tasks for each scheduled VL in software. Listing 1 presents the communication message structures for the respective three nodes encapsulated in IP/UDP frames at transmission time. The payload size of the exchanged UDP packets is 33 bytes, 25 bytes and 25 bytes for `VL_DYN`, `VL_FILTER` and `VL_CTRL`, respectively. The floating-point variables correspond directly to the case study variables described in Table I. The variable `step` represents the current simulation time derived from the *Aircraft Node 1* and it propagates through the rest of the distributed nodes to indicate when to stop the benchmark simulation.

Additionally, we introduce the variables `enable_filtering`, `enable_control`, and `is_controlling` that we use as flags to facilitate the programming and boot-up of the nodes in a sequential order starting from *Aircraft Node 1*. The flags indicate to the receiving nodes, *Filter Node 2* and *Control Node 3* that the previous node is programmed and should process the received data.

As discussed in Section III, all transmission tasks are associated with a reception task acceptance window. Thus, we implement the reception tasks as non-blocking functions that poll the Ethernet controller for the scheduled acceptance window time. We configure the Ethernet controller to use dual-buffering in a ping-pong management scheme. When we read from the active buffer, we clear the other buffer and swap the active buffer pointer to the other buffer so that it is ready to receive a new frame.

C. Static Scheduling

The ROSACE benchmark is composed of 12 periodic tasks (see Table II) that are scheduled over three distributed nodes. Each of the aircraft, filtering and control nodes is assigned in-order three, five and four tasks respectively.

Listing 1: Implemented message structs containing the closed-loop variables (see Table I) for each communication flow (VL_DYN, VL_FILTER, and VL_CTRL).

```
typedef struct {
  uint32_t step;
  uint8_t enable_filter;
  float engine_T;
  float elevator_delta_e;
  float Va;
  float Vz;
  float q;
  float az;
  float h;
} aircraft_state_message;

typedef struct{
  uint32_t step;
  uint8_t enable_control;
  float h_meas;
  float q_meas;
  float az_meas;
  float vz_meas;
  float va_meas;
} filter_state_message;

typedef struct{
  uint32_t step;
  uint8_t is_controlling;
  float h_c;
  float Va_c;
  float Vz_c;
  float delta_e_c;
  float delta_th_c;
} control_state_message;
```

Originally, the case study is coded using the PRELUDE [20] formal language to generate a set of dependent periodic tasks. PRELUDE can add real-time primitives to the synchronous data-flow model, by modelling tasks as nodes, where consuming nodes have a proportional rate constraint relative to a respective producing node. More precisely, the authors [13] describe the following proportional execution rates for the flight controller tasks. The filter tasks (`h_filter`, `Va_filter`, `Vz_filter`, `az_filter`, `q_filter`) should execute at half the rate of the environment simulation (`aircraft_dynamics`, `engine`, `elevator`), and the control tasks (`altitude_hold`, `Vz_control`, and `Va_control`) should execute at half the rate of the filter tasks.

In contrast, we do not use PRELUDE but instead deploy a distributed synchronized cyclic executive. We derive a task set for each node constrained by the relative rates of the tasks, the WCET and the network transmission points-in-time. Figure 2 illustrates an example distribution of the tasks in a time-triggered network.

To generate the combined task and network schedule, we follow a process similar to the one proposed by [21]. First, we perform a static WCET analysis on the implemented ROSACE tasks using the tool *platin* [18] and present the results in Table II. The lack of a floating-point unit in the used hardware platform significantly increases the tasks' WCET. In [13], the authors report a WCET of 200 μs for the aircraft_dynamics while the presented implementation is estimated at 17.9 ms. Consecutively, we derive valid periods for the tasks that aim

TABLE II: WCET of ROSACE tasks/functions on the T-CREST platform.

Function	WCET (clock cycles)
`engine`	13326
`elevator`	33675
`aircraft_dynamics`	1435878
`h_filter`	14923
`q_filter`	15119
`az_filter`	14902
`Vz_filter`	15119
`Va_filter`	14923
`h_c`	1046
`Vz_control`	35420
`altitude_hold`	14084
`Va_control`	40352

to maintain the relative proportions to each other as originally defined by [13]:

- Aircraft dynamics: 20 ms (originally 5 ms)
- Filtering: 40 ms (originally 10 ms)
- Control loops: 80 ms (originally 20 ms)

Empirically, we select the TTEthernet integration period at 80 ms based on the slowest transmission rate. This allows aligning the start of each node's schedule hyper period with the reception of a PCF and a synchronization task.

To generate a valid schedule, we derive the maximum transmission time from the TTTech development suite [22] and consider this duration additionally to the WCET of each transmitting task. Moreover, we configure the acceptance window time of each receiving task to the maximum clock drift and include this time in the WCET of the respective tasks. Finally, we allocate an inter-task time gap ϕ that is bounded by the overhead of the runtime dispatcher, the WCET of reading the system clock, and the measured clock synchronization offset, as shown in Equation 1. We provide the task definition to the custom SMT scheduler and generate the header files for each nodes that contain the tasks' activation times.

$$\phi = WCET_{dispatcher} + WCET_{read\_clock} + Offset_{clock} \quad (1)$$

D. 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 presented benchmark application is hosted at *https://github.com/t-crest/patmos/tree/master/c/apps/rosace*

V. Evaluation

This section presents the experimental setup that the case study is deployed and evaluates the performance of the proposed open-source runtime system.

A. System Setup

The presented ROSACE benchmark is distributed over three nodes that execute the proposed runtime system using the softcore processor Patmos. The hardware platforms are

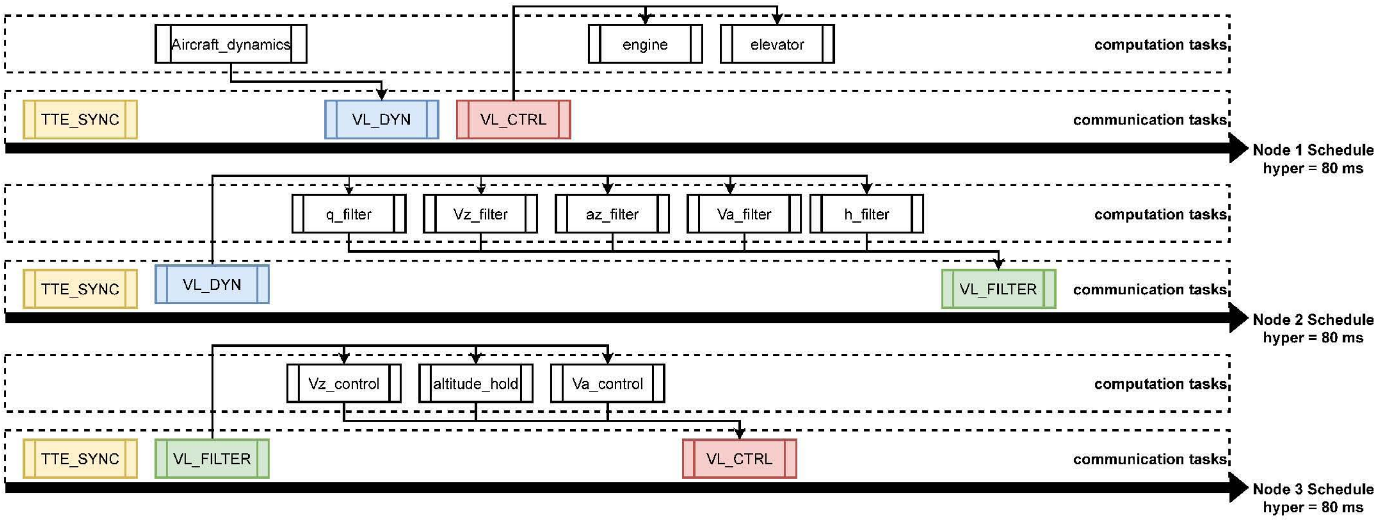

Fig. 2: Example ROSACE task execution flow over three nodes with software-based TTEthernet communication. Individual node timing is not relative to each other.

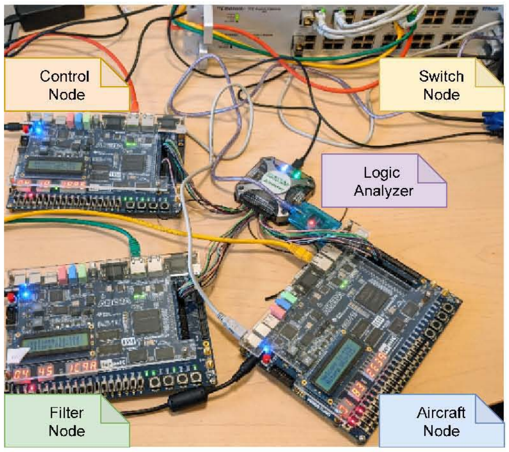

Fig. 3: Distributed ROSACE setup over three nodes that are integrated in a TTEthernet network.

synthesized on three Terasic DE2-115 FPGA boards [23] that operate at a frequency of 80 MHz. The nodes are integrated, as synchronization clients, in a TTEthernet network star topology that is composed of a single TTE Chronos 18/6 Rugged switch acting as a compression master and two Linux desktops that act as the synchronization masters. Figure 3 shows the experimental setup composed of the three distributed ROSACE nodes interconnected through a TTEthernet network switch. A logic analyzer is used to monitor the clock synchronization precision and the task execution.

To enable our comparative analysis, we additionally execute and measure the original ROSACE benchmark code [1] in simulation time on a 64-bit i7-7700HQ CPU system running at 2.8 GHz with 32 GB RAM.

[1]https://svn.onera.fr/schedmcore/branches/ROSACE_CaseStudy/c_posix_implementation/

B. Runtime System and Task Scheduling

We perform a complete system analysis by collecting statistics from the distributed schedules of the three nodes during the benchmark execution time of 600 seconds. Table III presents the gathered results of the performance of the proposed runtime system. The dispatcher manages to execute jobs with jitter below 10 μs compared to 32 μs of the software system presented by [24]. The existing dispatcher jitter, is hypothesized to be caused by reading the clock and searching through the schedule table. Moreover, the computed schedule is a candidate for further optimization by estimating tighter bounds for the acceptance windows and network transmission time. However, this is outside the scope of this work.

C. Clock Synchronization

We evaluate the distributed system's clock synchronization relative to the TTEthernet network switch by generating I/O pulses from the synchronization tasks on each node and the hardware timestamp units when a valid PCF is received. By comparing the time difference between the I/O pulse generated by the arrival of a PCF frame and the I/O pulse generated by the execution time of the synchronization task, we can visualize and derive the overall synchronization offset of the node. We measure the offset using a logic analyzer and present the results in Figure 4. The nodes are synchronized to the network schedule (reception of PCFs) to a measured precision of $\approx$ 100 μs. While the observed synchronization of the distributed task schedules relative to each other is $\leq$ 50 μs.

D. Quality of Control

The achieved quality of control of the presented design is evaluated by measuring the step response of the aircraft during the scenario of a 1000 m climb that starts at 50 seconds and executes for 600 seconds. The aircraft has an initial altitude of $h = 10000m$ and requires a total of $\approx$ 400 seconds to

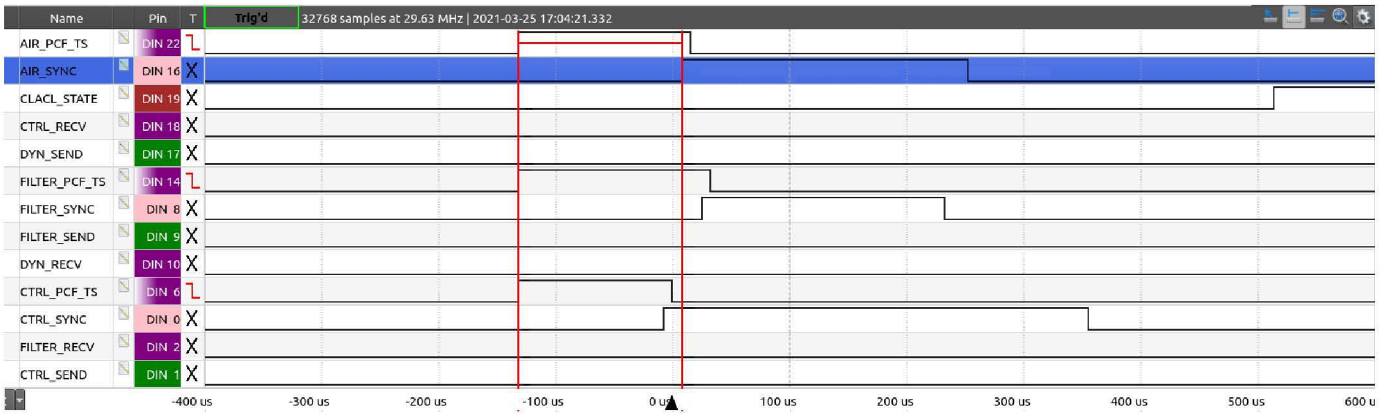

Fig. 4: Measured clock synchronization accuracy of the synchronization task I/O pulse relative to the arrival time of the TTEthernet PCF I/O pulse.

TABLE III: Runtime system task execution measurements from the three distributed nodes.

Node	**Task**	**Avg. Δt (μs)**	**Avg. Jitter (μs)**	**Max. ET (μs)**
aircraft	SYNC	79996.464	3.536	44189.587
	ENGINE	19997.416	2.584	118.325
	ELEVATOR	19997.478	2.522	292.263
	AIRCRAFT_DYN	19997.074	2.926	13749.913
	VL_DYN_SEND	19997.404	2.596	110.600
	LOGGING	19997.484	2.516	2656.088
	VL_CTRL_RECV	79993.408	6.592	287.463
filter	SYNC	79991.664	8.336	14304.613
	VL_DYN_RECV	19996.990	3.010	286.100
	Q_FILTER	39996.480	3.520	133.000
	VZ_FILTER	39996.480	3.520	133.000
	AZ_FILTER	39995.624	4.376	133.000
	VA_FILTER	39995.656	4.344	134.025
	H_FILTER	39995.676	4.324	134.025
	VL_FILTER_SEND	39995.700	4.300	103.750
control	SYNC	79991.600	8.400	3272.250
	VL_FILTER_RECV	39996.108	3.892	286.100
	VZ_CONTROL	79991.600	8.400	278.850
	ALTI_HOLD	79991.600	8.400	129.238
	VA_CONTROL	79991.600	8.400	103.488
	VL_CTRL_SEND	79991.600	8.400	317.538

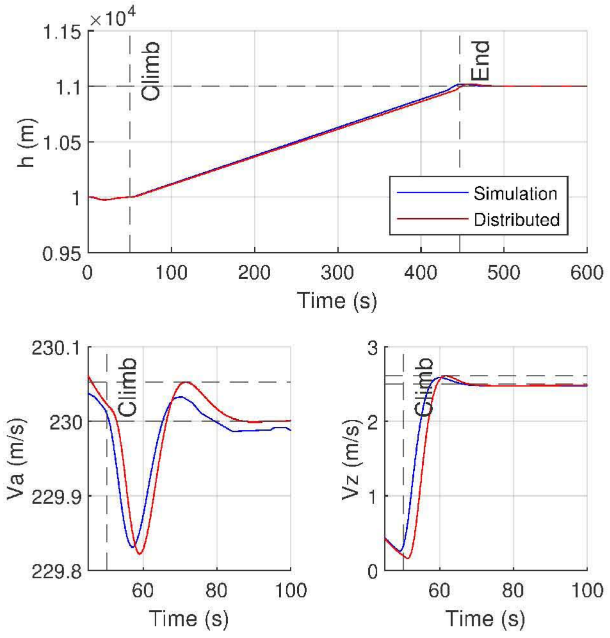

Fig. 5: Aircraft step response comparison between simulation and distributed real-time system implementation. Commanded 1000 m climb starts at 50 seconds.

reach the designated altitude with a reference vertical speed $Vz = 2.5m/s$ and a reference airspeed $Va = 230m/s$. Figure 5 presents the results of the distributed ROSACE implementation, gathered by the `LOGGING` task, during the commanded flight scenario. The results are compared against the simulated execution on the Linux machine. The figure is split into three sub-plots that describe the aircraft's ascend as follows:

- The top plot presents the altitude curve during the entire runtime of the scenario.
- The bottom-left plot presents a close-up view of the aircraft's true airspeed step response during a 20 seconds time-widnow around the commanded climb.
- And the bottom-right plot presents a view of the aircraft's vertical speed response during the commanded step climb.

The presented design performs a stable flight-level climb similar to the simulated implementation with a slightly higher overshoot and response delay due to the reduced sampling frequency of the implementation.

To validate the performance characteristics of the presented distributed real-time system, we focus on the time-domain performance specifications defined by the ROSACE case-study. In [13], the authors use the same validation rules to verify the correctness of the SIMULINK model performance. Table IV presents the flight validation results of the presented distributed system evaluation that executes in real-time and compares them against a Linux implementation that executes in simulation time. The results evaluate the performance of

TABLE IV: ROSACE requirements validation and results comparison

Property	Objective	Linux Results	Distributed Results
P1 5% settling time	Vz $\leq$ 10 s	6.430 s	8.360 s
	Va $\leq$ 20 s	5.560 s	0.020 s
P2 Overshoot	Vz $\leq$ 10 %	3.443%	4.360%
	Va $\leq$ 10 %	0.014%	0.023%
P3 Rise time	Vz $\leq$ 6 s	0.170 s	5.460 s
	Va $\leq$ 12 s	0 s	0 s
P4 Steady-state error	Vz $\leq$ 5 %	0.973%	0.960%
	Va $\leq$ 5 %	0.004%	4.374e-04%

the flight regarding the vertical speed Vz and true airspeed Va quality-of-control against a set of objectives discribed in Section II. The system performs well within the specification bounds and with results very close to the simulated performance of the Linux implementation.

VI. Related Work

This section reviews recent related research in the domain of distributed time-triggered system evaluation.

Synchronizing task execution with the underlying communication schedule has been advertised over an asynchronous approach by [5] and has been explored in detail by [25], where the authors present an SMT-based approach for synthesizing combined schedules for communication and task execution. The authors focus on optimizing the approach's schedulability and using an earliest-deadline first scheduling policy evaluated on a proprietary runtime system using a synthetic task set. In contrast, our work focuses on designing and evaluating a realistic closed-loop control application experimentally using a complete open-source framework.

There is much significant research being carried out in the field of optimizing time-triggered systems regarding communication and scheduling, some examples being [26], [27], [28]. To the authors' knowledge, few of these have illustrated the design process of a realistic case study in time-triggered embedded systems that evaluates the performance of a runtime system and the underlying network, particularly in the case of TTEthernet.

An analytical view on time-triggered architectures for avionic embedded systems is presented in [29]. The authors present a time-triggered constraint-based calculus framework for formal analysis of integrated modular avionics systems. They present a formal analysis of an avionic landing-gear system connected through a TTEthernet network that can specify the schedulability and the end to end delay of functional chains properties of such a system.

A TTEthernet-based flight management system is investigated and modeled in [30]. The authors present a methodology to model the individual components of a time-triggered embedded system and evaluate the model in simulation using the Ptolemy II actors framework [31]. In contrast, our work is experimentally driven and presents the evaluation of a runtime system that is implemented on an experimental distributed system setup.

In [13], the authors presented the ROSACE case study of a multi-rate longitudinal flight controller. The authors presented a complete design approach from modelling the controller in SIMULINK to implementing the application in a multi/many-core executable using PRELUDE. The tasks were mapped to three tiles on a many-core TILERA TILEMPOWERGX-36 platform [32] based on their periods. In this work, we examine and extend this approach to a distributed time-triggered implementation. We evaluate an open-source framework to derive and schedule a cyclic task set that we distribute in a TTEthernet network over three nodes.

Finally, in [24], the authors present a software-based time-triggered system for automotive applications. It deploys a buffering scheme for standard Ethernet controllers to support time-triggered communication and evaluate the performance and time-predictability of the end-system with synthetic traffic flows. In contrast to our work, the authors do not evaluate a specific benchmark control application, and thus they do not evaluate task execution together with communication.

VII. Future Work

Distributing the ROSACE benchmark over a TTEthernet network using an open-source framework for time-triggered communication allows evaluating the performance of different communication protocols and scheduling policies using a realistic closed-loop control application.

Industry 4.0 is enabling fog computing for industrial automation through the use of time-sensitive networking (TSN). We identify the challenges involved in the control optimization performance [33] and plan to explore the design extensions of the proposed framework needed to integrate into a TSN network and deploy the presented distributed ROSACE benchmark. This will allow us to perform a comparative analysis of the design and performance between different underlying network protocols and scheduling policies.

VIII. Conclusion

This work explored the design of distributing a closed-loop control application on a TTEthernet network using an open-source time-triggered runtime system. Using the proposed open-source framework, we were able to successfully distribute and schedule the benchmark tasks on three nodes as well as schedule the underlying communication.

We presented the ROSACE longitudinal flight controller and the validation objectives of the benchmark. Consecutively, we described the design process of distributing the flight controller on a time-triggered distributed system and the implementation details needed to achieve a functional and time-predictable design.

Finally, we deployed and evaluated the runtime system in an experimental TTEthernet network and measured its performance. The benchmark demonstrated the correct functionality of the proposed framework, and the presented design was able to pass the validation goals with tight synchronization and minimal task jitter within 10μs.

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] T. Mitra, J. Teich, and L. Thiele, "Time-critical systems design: A survey," *IEEE Design & Test*, vol. 35, no. 2, pp. 8–26, 2018.

[2] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *Journal of Systems Architecture*, vol. 80, pp. 104–113, 2017.

[3] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sept 1987.

[4] 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.

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

[6] A. Mehmed, S. Punnekkat, and W. Steiner, "Deterministic ethernet: Addressing the challenges of asynchronous sensing in sensor fusion systems," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2017, pp. 22–28.

[7] A. Minaeva and Z. Hanzálek, "Survey on periodic scheduling for time-triggered hard real-time systems," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–32, 2021.

[8] P. Danielis, J. Skodzik, V. Altmann, E. B. Schweissguth, F. Golatowski, D. Timmermann, and J. Schacht, "Survey on real-time communication via ethernet in industrial automation environments," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. IEEE, 2014, pp. 1–8.

[9] L. L. Bello and W. Steiner, "A perspective on ieee time-sensitive networking for industrial communication and automation systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1094–1120, 2019.

[10] E. Kyriakakis, J. Sparsø, P. Puschner, and M. Schoeberl, "Synchronizing real-time tasks in time-triggered networks," in *24th International Symposium On Real-Time Distributed Computing (ISORC)*. IEEE, 2021.

[11] 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.

[12] W. Steiner, G. Bauer, and D. Jameux, "Ethernet for space applications: Ttethernet," in *International SpaceWire Conference 2008, Nara, Japan*, 2008.

[13] C. Pagetti, D. Saussié, R. Gratia, E. Noulard, and P. Siron, "The rosace case study: From simulink specification to multi/many-core execution," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2014, pp. 309–318.

[14] S. Documentation, "Simulation and model-based design," 2020. [Online]. Available: https://www.mathworks.com/products/simulink.html

[15] 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.

[16] 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.

[19] 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.

[17] 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.

[18] 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.

[20] 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.

[21] 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.

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

[23] *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

[24] 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.

[25] 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.

[26] 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.

[27] P. Pop, P. Eles, and Z. Peng, "Schedulability-driven communication synthesis for time triggered embedded systems," *Real-Time Systems*, vol. 26, no. 3, pp. 297–325, 2004.

[28] F. Pozo, G. Rodriguez-Navas, H. Hansson, and W. Steiner, "Smt-based synthesis of ttethernet schedules: A performance study," in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*. IEEE, 2015, pp. 1–4.

[29] S. Hamadou, J. Mullins, and A. Gherbi, "A real-time concurrent constraint calculus for analyzing avionic systems embedded in the ima connected through ttethernet," in *Theoretical Information Reuse and Integration*. Springer, 2016, pp. 85–111.

[30] G. Brau and C. Pagetti, "Ttethernet-based architecture simulation with ptolemy ii," in *6th Junior Researcher Workshop on Real-Time Computing (JRWRTC 2012)*, 2012, pp. p29–32.

[31] C. Ptolemaeus, Ed., *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.

[32] Y. Liu, H. Sasaki, S. Kato, and M. Edahiro, "A scalability analysis of many cores and on-chip mesh networks on the tile-gx platform," in *2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*. IEEE, 2016, pp. 46–52.

[33] M. Barzegaran, A. Cervin, and P. Pop, "Performance optimization of control applications on fog computing platforms using scheduling and isolation," *IEEE Access*, vol. 8, pp. 104 085–104 098, 2020.