Time-predictable End-system Design for Real-Time Communication

Kyriakakis, Eleftherios

Publication date:
2021

Document Version
Publisher's PDF, also known as Version of record

Time-predictable End-system Design for Real-Time Communication

Eleftherios Kyriakakis

Kongens Lyngby 2021
PhD

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, Building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3351
compute@compute.dtu.dk
www.compute.dtu.dk
PhD-2021
ISSN: 0000-0000

Abstract (English)

Over the years, computing systems have not only increased their processing power but have also increased the total number of interconnected nodes. Nowadays, most computing systems found in industrial, automotive and aerospace application domains are distributed cyber-physical systems comprised of a plethora of networked platforms. Thus it is becoming a challenge to guarantee both time-predictable task execution and bounded end-to-end communication latency. To this end, different techniques are employed on the network and the processor. On the network, different protocols have been developed to achieve deterministic and temporally isolated communication. While on the processor, time-predictable execution can be guaranteed by employing different scheduling policies combined with static worst-case execution time analysis.

This thesis explores the software and hardware solutions that extend an embedded system with mechanisms to provide precise and fault-tolerant clock synchronization, minimal end-to-end communication latency and synchronous task execution. These solutions potentially increase the system's time-predictability and create an overall reliable time-triggered computing platform for safety-critical system research.

Firstly, time-synchronization is established by exploring the IEEE 1588 Precise Time Protocol and developing a hardware unit. The design is experimentally verified and achieves nanosecond clock synchronization. The safety and security properties of the IEEE 1588 protocol are analyzed, and a fault-tolerant prototype design is proposed. The design enables reliable synchronization in time-sensitive networking communication systems. The design is evaluated in simulation using synthetic benchmarks that demonstrate the design's capability to tolerate multiple network failures and denial-of-service attacks.

Next, the time-triggered communication protocol TTEthernet is explored, and a time-analyzable network stack is presented. The design allows an embedded platform to communicate and synchronize its time over TTEthernet networks. Based on the developed network stack, synchronization of real-time tasks with an underlying time-triggered communication layer is explored. An open-source framework is presented for scheduling and executing synchronous distributed tasks. The framework is evaluated experimentally using a multi-rate synthetic application. The evaluation demonstrates minimal end-to-end communication latency as well as distributed task synchronization with minimal jitter. Finally, the evaluation is extended with an avionic benchmark application. The benchmark presents a longitudinal flight controller case study that is successfully distributed and scheduled using the proposed time-triggered framework. The benchmark is implemented in an experimental TTEthernet network with three nodes and successfully executes a flight scenario.

Resumé (Dansk)

Med tiden er computersystemer ikke kun blevet kraftigere, men også det samlede antal forbundne systemer er vokset. I dag er de fleste computersystemer indenfor industrien, autobranchen og rumfarten, distribuerede cyberfysiske systemer, bestående af en overflod af netværksenheder. Det er derfor en udfordring at afgrænse kommunikationsforsinkelsen, samt garantere tidsforudsigelig udførelse af opgaver. Til dette formål anvendes forskellige teknikker til netværk og processorer. Til netværk er der udviklet forskellige protokoller for at opnå deterministisk og tidsmæssigt isoleret kommunikation, mens på processorerne kan tidsforudsigelig udførelse garanteres ved at anvende forskellige planlægningspolitikker kombineret med statisk analyse til at finde den længste udførelsestid.

Denne afhandling udforsker software- og hardwareløsninger der udvider et integreret system med mekanismer til at give præcis og fejletolerant klokkesynkronisering, minimal kommunikationsforsinkelse og synkron opgaveudførelse. Disse løsninger øger potentielt systemets tidsforudsigelighed og skaber en tidspålidelig computerplatform til forskning af tidskritiske systemer.

Først udforskes klokkesynkronisering ved undersøgelse af IEEE 1588 Precise Time Protocol og udvikling af en hardwareenhed. Enhedens design er eksperimentelt verificeret og opnår synkronisering med en fejlmargin indenfor nanosekunder. Sikkerhedsegenskaberne i IEEE 1588-protokollen analyseres også, og der foreslås en fejltolerant udvidelse. Udvidelsen muliggør pålidelig synkronisering i tidsfølsomme netværk. Designet evalueres i simulering og demonstrer tolerans for flere netværksfejl og denial-of-service angreb.

Dernæst udforskes den tidsafhængige kommunikationsprotokol TTEthernet, og en tidsanalyserbar netværkstak præsenteres. Designet gør det muligt for en

integreret platform at kommunikere og synkronisere sin tid over TTEthernet-netværk. Baseret på den udviklede netværksstak udforskes synkronisering af realtidsopgaver med et underliggende tidsafhængigt kommunikationslag. Et open-source bibliotek præsenteres til planlægning og udførelse af synkrone, distribuerede opgaver. Biblioteket evalueres eksperimentelt ved hjælp af en syntetisk applikation med variabel kommunikation. Evalueringen demonstrerer minimal kommunikationsforsinkelse samt distribueret opgavesynkronisering med minimal tidsvarians. Endelig udvides evalueringen med en flysimulator test. Testen består af en flykontroller der distribueres og synkroniseres med hjælp af det præsenterede bibliotek. Testen implementeres i et eksperimentelt TTEthernet-netværk med tre enheder og udfører succesfuldt et flyvescenarie.

Preface

The work presented in this thesis was conducted at DTU Compute in fulfillment of the requirements of the PhD program. 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.

The work was supervised by Associate Professor Martin Schoeberl and Professor Jens Sparsø. The thesis explores time-predictable communication and task execution in distributed real-time systems.

The thesis is based on five published papers and is organized in seven chapters. Starting with an introductive background chapter, followed by one chapter per paper and finally closing with a conclusion chapter.

Kongens Lyngby, 01-May-2021

Eleftherios Kyriakakis

Acknowledgements

Firstly, I would like to thank my supervisors Martin Schoeberl and Jens Sparsø for their advice throughout my PhD years and all the interesting discussions during our Friday meetings. Thank you for this opportunity.

Secondly, I would like to thank all my colleagues at the Embedded Systems Engineering section at the Technical University of Denmark. Particularly my PhD colleagues Luca Pezzarossa, Tòrur Biskopstø, Koen Tange and Mohammadreza Barzegaran for their support and fruitful discussions during the late afternoons that helped lighten the frustration of the various coding bugs. Moreover, I wish good luck to all the FORA PhD students with their future research and upcoming defence. I am also very thankful for all the friends I have made over the last few years.

Special thanks go to my close friends and flatmates Pavlina Senikoglou and Christos Gkiokas for their perfectly cooked meals and for their patience in listening to my complaints during times of discouragement.

Finally, I would like to thank my girlfriend Myrto Asimakopoulou and my parents, Aikaterini Gkioka and Giorgos Kyriakakis, for their love, encouragement and sharing this journey with me despite the difficulties of the distance and the encumbrance of COVID-19 travel restrictions.

To the fresh and ambitious PhD students I dedicate the following verse:

> This is ten percent luck, Twenty percent skill
> Fifteen percent concentrated power of will
> Five percent pleasure, Fifty percent pain
>
> *Mike Shinoda - Fort Minor*

Contents

List of Acronyms

CM	compression master
COTS	commercial off-the-shelf
CPS	cyber-physical systems
ET	event-triggered
IO	input-output
IP	Internet protocol
LCM	least-common multiplier
MAC	medium access control
NIC	network interface controller
NoC	network-on-chip
NTP	network time protocol
OS	operating system
PCF	protocol control frame
PTP	precise time protocol
RC	rate-constrained
RTC	real-time clock
SC	synchronization client
SM	synchronization master
SMT	satisfiability modulo theories
SPM	scratchpad memory
TDM	time-division multiplexing
TSN	time-sensitive networking
TT	time-triggered
TTP	time-triggered protocol
UDP	user datagram protocol
WCEL	worst-case end-to-end latency
WCET	worst-case execution time

List of Publications

Journal Publications

[J1] Eleftherios Kyriakakis, Maja Lund, Luca Pezzarossa, Jens Sparsø, and Martin Schoeberl. "A time-predictable open-source TTEthernet end-system". In: *Journal of Systems Architecture* (2020), p. 101744.

Conference and Workshop Publications

[C1] Eleftherios Kyriakakis, Jens Sparsø, and Martin 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.

[C2] Eleftherios Kyriakakis, Jens Sparsø, Peter Puschner, and Martin Schoeberl. "Synchronizing Real-Time Tasks in Time-Aware Networks". In: *Proceedings of the 24th IEEE International Symposium on Real-Time Computing (ISORC)*. IEEE. 2021.

[C3] Eleftherios Kyriakakis, Koen Tange, Niklas Reusch, Eder Ollora Zaballa, Xenofon Fafoutis, Martin Schoeberl, and Nicola Dragoni. "Fault-tolerant Clock Synchronization using PreciseTime Protocol Multi-Domain Aggregation". In: *Proceedings of the 24th IEEE International Symposium on Real-Time Computing (ISORC)*. IEEE. 2021.

[C4] Eleftherios Kyriakakis, Jens Sparsø, and Martin Schoeberl. "Evaluating a Time-Triggered Runtime System by Distributing a Flight Controller". In: *Proceedsings of the 26th International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE. (Submitted).

Other Works

[O1] Eleftherios Kyriakakis, Jens Sparsø, and Martin Schoeberl. "Implementing time-triggered communication over a standard ethernet switch". In: *Proceedings of the Workshop on Fog Computing and the IoT*. 2019, pp. 21–25.

[O2] Eleftherios Kyriakakis, Jens Sparsø, and Martin Schoeberl. "InterNoC: Unified Deterministic Communication For Distributed NoC-based Many-Core". In: *Proceedings of the 13th Junior Researcher Workshop on Real-Time Computing*. Nov. 2019. URL: https://www.jopdesign.com/doc/internoc-jrwrtc.pdf.

[O3] Eleftherios Kyriakakis, Jens Sparsø, Peter Puschner, and Martin Schoeberl. "Synchronizing Real-Time Tasks in Time-Aware Networks: Work-in-Progress". In: *Proceedsings of the International Conference on Embedded Software (EMSOFT)*. IEEE. 2020, pp. 15–17.

Chapter 1

Introduction

Over the years, computing systems have evolved from information processing devices to microcontrollers embedded in several control systems both in industry and everyday appliances. A prominent example of this trend is the automotive industry, with manufacturers like BMW and Audi motors having triplicated the number of electronic control units in their vehicles since 1995. These systems are called embedded systems and consist of purpose-specific hardware and software components integrated into a mechanical or electrical system, e.g., cameras, refrigerators, vehicle cruise control systems and industrial robotics.

Nowadays, embedded systems are distributed over networks and interface with multiple input-output (IO) devices such as sensors that collect the relevant information about the physical world needed to control various actuation systems. A cyber-physical system a collection of distributed embedded and control systems that often find application in safety-critical domains such as aerospace and automotive. This integration adds the functional requirement of timeliness, and *such embedded systems* are defined as real-time systems. It is thus beneficial for embedded system designers to develop the necessary mechanisms that enable time-predictable application deployment. The scope of this thesis is to explore the design of hardware and software components necessary to provide temporal guarantees in the computation and communication of distributed cyber-physical systems.

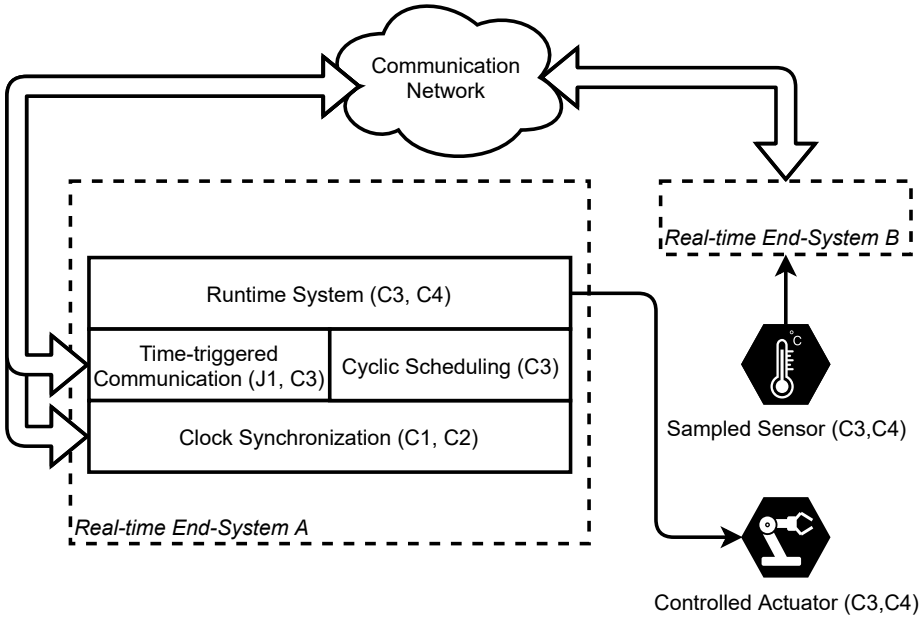


Figure 1.1: Research contributions, [C1], [C3], [J1], [C2], [C4] , mapped to the components of an example cyber-physical network with two real-time systems.

A typical cyber-physical network between two end-systems A & B is illustrated in Figure 1.1, which represents a typical command/control application. *End-system A* controls an actuator, e.g., a motor, based on the input received from *end-system B* conntected to a sensing device, e.g., a temperature sensor.

The example presents the hardware/software layers that comprise a distributed real-time end-system. The distributed system is based on a network through which end-systems can exchange data. Chapter 4 investigates a time-triggered network protocol and develops the necessary network stack. Time-triggered communication of connected end-systems requires a mechanism to synchronize their time relative to each other. Chapter 2 develops the necessary hardware and software components to achieve precise clock synchronization and Chapter 3 analyzes the reliability of the clock synchronization protocol and presents a fault-tolerant design. Actuators and sensing devices are controlled by the implementation of a runtime system that executes a task schedule. Chapter 5 proposes the synchronization of executed tasks with the underlying communication layer to minimize end-to-end communication latency and the proposed framework is evaluated using an avionic benchmark in Chapter 6.

This chapter aims to introduce the reader to the fundamental terminology of real-time systems and provide an overview of the background literature necessary to understand this work's contributions. Mainly, it summarizes the concepts of hard real-time systems, time-predictable hardware, task execution, real-time communication and time synchronization.

1.1 Hard Real-time Systems

Cyber-physical systems in safety-critical application domains such as aerospace, industry and automotive are composed of systems that require bounded temporal behaviour to guarantee safe and correct functionality [3]. For example, a correctly computed result for an imminent crash in an autonomous vehicle that is delivered with an un-predicted delay to the airbag deployment system can have catastrophic results. These systems are defined as real-time systems, and their functional correctness depends both on their functional aspects and on their temporal behaviour. Depending on the importance of the temporal requirements, real-time systems can be categorized into three classes:

1. **Soft real-time**: define systems with lose timing requirements that can tolerate late results, i.e., a streaming media application.
2. **Firm real-time**: are systems that depend on strict timing guarantees but can tolerate later or missed results due to the nature of the controlled process, i.e., motor controllers and some actuation devices.
3. **Hard real-time**: is the system class with the strictest timing requirements that cannot tolerate any missed deadlines as it will result in significant financial loss, casualty, and even fatality, i.e., a vehicle airbag system, aircraft landing gears etc.

As the system input varies depending on the operating conditions and controlled process, it can lead to different computation and communication times; thus, the system engineer has to employ system architectures and technologies to guarantee that the worst-case execution time (WCET) of any computation tasks and the worst-case end-to-end latency (WCEL) of any data communication is analyzed and bounded to avoid late or missing results. Thus a *system* is defined as time-predictable when the worst-case timing properties can be derived through analysis [44]. This thesis does not examine all real-time systems classes but instead it focuses on distributed hard real-time systems. The following sections summarize the most relevant technologies and techniques used to achieve time-predictable end-to-end operation.

1.2 Time-predictable Hardware

The architecture of embedded systems can significantly affect the timeliness of safety-critical applications. Different standard hardware units are known to be built for optimizing the average case, but that leads to architectures that are hardly WCET-analyzable. It is essential to investigate hardware architectures that enable the overall system timing analysis and optimize the worst-case. The following considerations have been proposed over the years here:

Avoid timing anomalies Real-time system architectures should avoid timing anomalies during different executions of the same program over the same data. A hardware architecture is free of timing anomalies when there is some consistently worse hardware state that produces an upper bound on the execution time of a program using the same data [61]. Timing anomalies can appear in different hardware components such as caches and processor pipelines, e.g., most-recently used cache replacement policy or out-of-order execution [23]. In contrast, it is proposed that a platform employs in-order processor pipeline execution with timing anomalous free cache replacement policies such as the least-recently used policy.

Scratchpad memories instead of caches Access on caches has been known to be hardly time-predictable; in research, two techniques have usually been implemented to estimate a safe WCET bound. Either during WCET analysis, the designer must assume that every *load* instruction is a *cache miss* or employ the use of a scratchpad memory (SPM) to provide constant access time and increase performance [12, 45]. SPM usage is not limited only for data access but recent research has also explored this concept for caching program instructions. Results show that SPM caches can offer more optimistic WCET bounds than method caches [84]. The drawback is that most commercial off-the-shelf (COTS) platforms are not equipped with SPMs with only a few exceptions, such as the recently introduced in the ARM Cortex-R5 using a tightly coupled memory system [156]. Additionally, managing memory allocation and data structures in SPMs must be manually maintained. Finally, on-chip scratchpad memories cannot be used for bulk data since their size is physically limited by the physical integrated chip area.

Time-division multiplexing Traditional multicore embedded systems handle memory access using fixed priority or round-robin arbitration through a shared bus. Recent research has shown that access time can be optimized for better worst-case analysis using a time-division multiplexing (TDM) access scheme that allocates pre-defined time-slots to processors during which they can access a device such as a memory [82]. Alternatively,

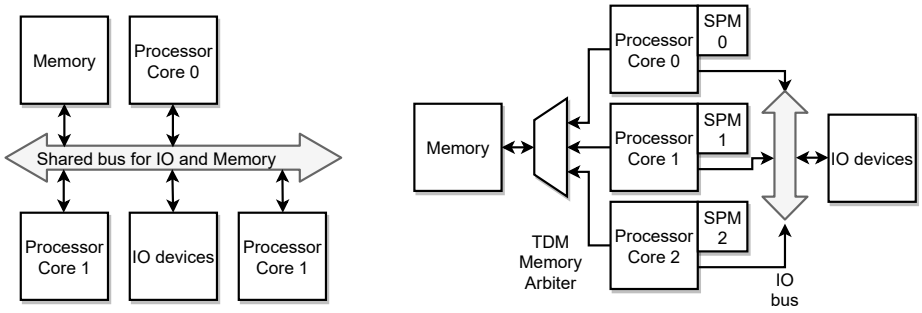


Figure 1.2: Multicore shared bus architecture on the left and TDM-based multicore architecture with private scratchpad memories on the right.

synchronizing access to IO devices and memory can be achieved using hardware locks [155], meaning that when a processor needs to access a shared resource, it first tries to acquire a lock.

Embedded platforms that optimize their architecture towards timing predictability by employing a number of the discussed components have been the focus of recent research in hard real-time systems [31, 71]. This thesis extends an in-house time-predictable platform of T-CREST [94], that implements the discussed optimizations. Figure 1.2 presents a comparison between a traditional multi-core architecture and the TDM-based architecture implemented in T-CREST.

1.3 Task Execution

Hard real-time applications are composed of several executing units, named tasks which together comprise a *task set*. An executing task can be a thread, or in the absence of an operating system (OS), a simple function call. An application can be composed of a fixed number of tasks or a dynamic *task set*. Most real-time systems perform a specific control process and are usually composed of a fixed number of tasks. This thesis only investigates solutions for these type of systems. Any real-time task has a deadline and must finish its execution before that, and some tasks also have a specified rate at which they must execute, i.e., in control loops or sensor sampling. Additionally, tasks can have precedence constraints relative to other tasks creating task chains between producer and consumer tasks. Overall, tasks can be categorized into three classes:

1. **Periodic**: define tasks that must be executed at a specific rate, and thus, are activated once every period based on a timer.
2. **Aperiodic**: are tasks that have no period constraints and are activated by an event such as an interrupt.
3. **Sporadic**: are tasks that can be activated at any point in time but cannot be re-activated until a certain period has elapsed.

Hard real-time systems require bounded timing guarantees on the execution of the tasks and the delivery time of data between producer and consumer tasks. It is, therefore, necessary to analyze a given *task set* and derive and execute a suitable schedule with which all the tasks can finish their execution within the deadline and adhere to their period and precedence constraints. Real-time systems model their functionality as periodic and sporadic tasks. Aperiodic tasks are not used in hard real-time systems, as they can interrupt the execution of the schedule at un-predicted points in time. This schedule interrupt can unwillingly elongate the execution of any ongoing task, missing its deadline and invalidating the execution of the schedule.

On a single-core environment, the execution of a *task set* needs to be coordinated by employing a *scheduler* mechanism. A *scheduler* is a piece of software that dictates the execution order and activation time of a task based on the *task set* constraints and using a specific algorithm. The schedule generation can be static (offline), meaning that the *scheduler* generates a table of task activation times that are executed at runtime using a function typically called *dispatcher*. Alternatively, a *scheduler* can be dynamic (online), meaning that the order of execution is determined at runtime and may change after each iteration. There exist different scheduling policies for determining the order of execution, such as fixed priority, rate-monotonic, earliest-deadline first, and cyclic executive [165]. Each policy offers different benefits and drawbacks. This thesis focuses on time-predictability and real-time communication. Thus, it explores and employs a cyclic executive policy that is hypothesized to combine seamlessly with the underlying communication and the TDM arbitration scheme of the T-CREST platform.

Figure 1.3 presents the execution time model of an example task τ_i that calculates a result and writes it back to memory. The task needs to read the memory for stored variables and sample new data from the connected IO devices to complete its execution.

Cyclic executives map a fixed *task set* of size n into a statically planned collection of periodic function calls. The minimum period T_i of a task τ_i defines the minimum cycle time, while all the minor cycles together form the major cycle

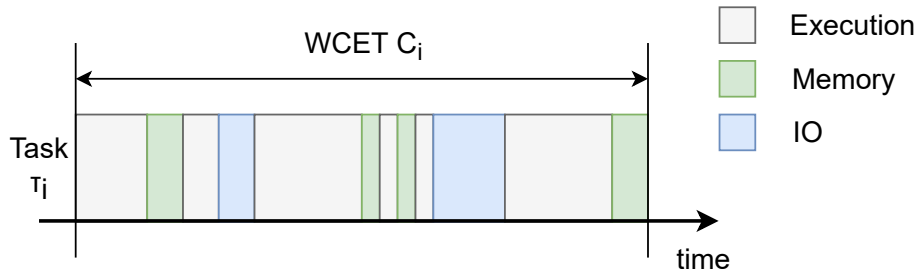


Figure 1.3: Execution time model of a task that accesses both memory and IO devices.

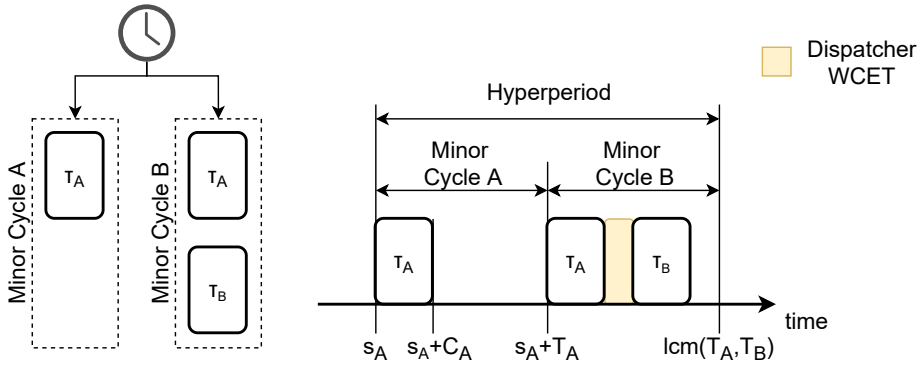


Figure 1.4: Multi-rate precedence constraints on the left and cyclic schedule on the right. Where s_A is the activation of τ_A and C_A its WCET.

time or *hyperperiod* that is computed as the least-common multiplier (LCM) of all the task periods: $lcm(T_i); \forall i \in n$. Figure 1.4 illustrates the task set order periodic constraints and cyclic execution in time.

Scheduling tasks using the cyclic executive policy offers fully deterministic behaviour and timing. Communication between tasks can be performed using shared data structures as it temporally guarantees mutual exclusion. Some of the drawbacks are that the task periods have to be harmonic, meaning that all tasks should be multiples of the minor cycle time. It is not very flexible as it does not allow scheduling of aperiodic or sporadic tasks. Additionally, long tasks need to split up to improve schedulability.

Figure 1.5 presents the localization part of an avionics flight management system that is responsible for receiving data from a set of sensors to compute the aircraft's probable position [77]. The case study presents an exciting applica-

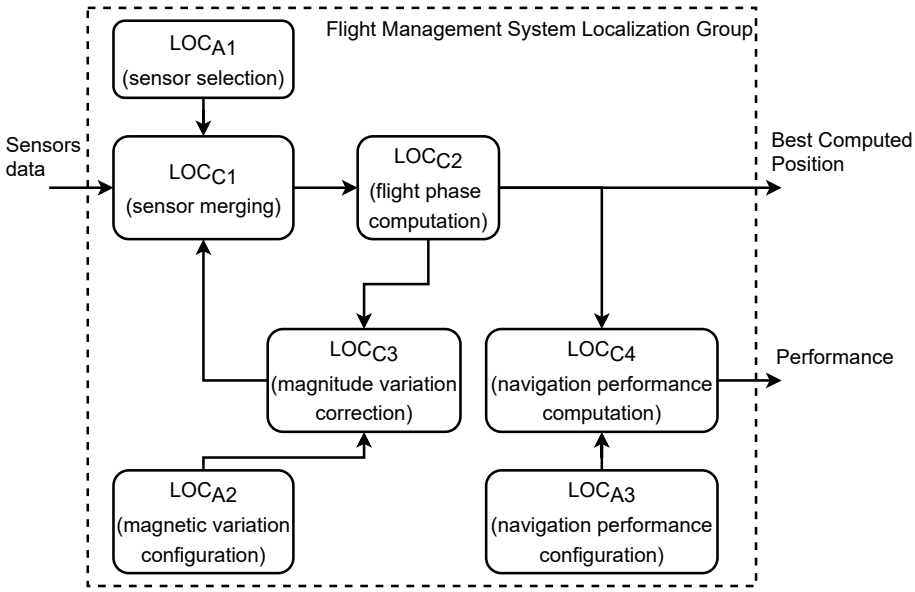


Figure 1.5: Example a flight management system localization task set that estimates the aircraft's position based on the input sensor data.

tion example that is composed of a task chain with periodic and aperiodic tasks. The aperiodic tasks are not scheduled together with periodic tasks as they are generated by the aircraft's pilot input. Instead aperiodic tasks are executed parallely in dedicated processors and the produced results are consumed by the periodic tasks asynchronously, when the tasks are activated. The benefit of this approach allows to successfully integrate aperiodic task (non real-time) with a real-time cyclic schedule without interference that can potentially lead to missed deadlines.

1.4 Real-time Communication

Although real-time communication has been investigated in various protocols such as [13, 26], in modern industrial and automotive networks, Ethernet is the preferred choice over fieldbuses [98]. This trend is mainly driven by the increasing need for efficient systems interoperability and support for mixed-criticality traffic, and Ethernet offers high-speed performance, flexibility and open specifications. Ethernet enables consistent integration on all levels of an application in

Table 1.1: Real-time Ethernet capabilities classification [33].

Class	Achievable cycle time	Implementation
C1	≥100 ms	Above the transport layer
C2	1 ms – 10 ms	Above the MAC layer
C3	250 us – 1 ms	Built-in at the MAC layer

these cases, allowing integration of an industrial system from the cyber-physical systems (CPS) level up to the data analytics level.

The operation of real-time Ethernet communication protocols is based on providing support for three levels of network traffic criticality that define the urgency of delivery-time: best-effort, time-triggered (TT), event-triggered (ET) and rate-constrained (RC). The ET, TT, and RC classes are analogous to the task classifications: aperiodic, periodic and sporadic. Moreover, to guarantee collision-free communication and without losses, the sender must never outpace the receiver, therefore the processing speed of the receiver determines the maximum communication rate.

Best-effort traffic does not require any real-time guarantees, and the only requirement is that the communication should be achieved at some point-in-time. This work does not investigate further its end-to-end delivery time optimization, and it can be assumed that this type of traffic can be scheduled as TT or RC using a scheme called porosity [66].

Scheduling real-time traffic can be achieved at any OSI layer based on the required guarantees, i.e., scheduling on top of the Internet protocol (IP) or the medium access control (MAC) layer. Additionally, handling the traffic can be done in software or processed by dedicated network interface controller (NIC). The capabilities of a real-time Ethernet system can be classified based on the schedule cycle time into three classes presented in Table 1.1.

Overall, communication in industrial, automotive and aerospace networks is provided by various proprietary protocols [117] such as PROFINET, EtherCAT, TTEthernet, and more recently, the IEEE 802.1 time-sensitive networking (TSN) task group [104]. These protocols implement a subset of the discussed traffic models and offer different capabilities. Table 1.2 summarizes the most popular hard real-time Ethernet solutions

Event-triggered ET traffic usually corresponds to aperiodic or sporadic traffic that initiates at arbitrary points in time based on an event, i.e., in an industrial setting, this could be an alarm or sensor interrupt that is

Table 1.2: Comparison of Real-time Ethernet Protocols [76, 159].

Protocol	Capabilities class	Traffic scheduling	Hardware requirements	IP support
PROFINET	C3	TT	Custom NIC	✗
EtherCAT	C3	TT	Custom NIC and software-based	✗
AFDX	C2	RC	Switches	✓
TTEthernet	C2/C3	TT & RC	Switches & Custom NIC	✓
TSN	C1/C2/C3	TT & RC & Priority	Switches	✓

delivered to a control system through the network. Explicit flow control mechanisms should be designed in place to protect a receiver from information overflow. Additionally, to guarantee the delivery of messages, this traffic flow needs a positive acknowledgement response. This scheme significantly increases the maximum transmission time as jitter can be induced based on the sporadic network load, and possible retransmissions [19].

Rate-constrained RC systems try to address the issues of ET systems by enforcing a bandwidth budget to every communication flow. Thus the rate of the sender is bounded by the communication system policy. Temporal guarantees can be given as long as the minimum assigned bandwidth for each channel is not exceeded. Guaranteed bandwidth in RC systems requires cooperative senders or a traffic shaping mechanism. Traffic shaping or policing is applied to communication flows to enforce that no channel exceeds the assigned bandwidth. Depending on the mechanism used, this is enforced by either delaying or by dropping non-conforming traffic. By delaying or dropping certain packets, the communication system can improve latency and guarantee bandwidth usage. These mechanisms have been studied and applied in avionic and industrial Ethernet protocols such as AFDX [17] and TSN [113].

Time-triggered This thesis does not examine ET and RC traffic criticalities. In contrast, it opts for periodic TT sampling of sporadic inputs to temporally encapsulate arbitrary events. The drawback of this approach is that the average latency is increased, but the maximum execution time is bounded, which is favourable in the design of hard real-time systems. TT traffic corresponds to the highest criticality level, where the delivery of

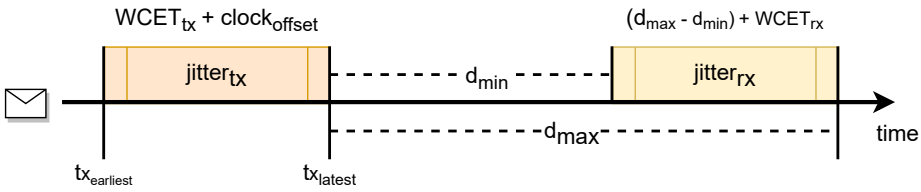


Figure 1.6: Example end-to-end transmission timing.

Ethernet frames must be guaranteed within specific time windows. This traffic class is addressed using the time-triggered protocol (TTP) paradigm [8, 47], which models the Ethernet frame communication as periodic tasks that are transmitted and received at predefined time-slots. The functionality of this concept requires a global schedule among senders and receivers that is dictated according to a network-wide notion of time [34]. Figure 1.6 illustrates the timing of a data structure transferred from one sender to one or more receivers. In the TT communication model, $tx_{earliest}$ and tx_{latest} are the boundaries of the sender's transmission window and introduce a transmission jitter $jitter_{tx}$. This $jitter_{tx}$ is dependent on the WCET of the processor plus the synchronization error of the device relative to the network time. The arrival jitter $jitter_{rx}$ is bounded by the minimum d_{min} and maximum d_{max} network delays experienced by the transmitted data combined with the receiving task's WCET. Time-triggered communication has been explored and implemneted in a variety of safety-critical domains inlcuding automotive [14, 64] and aerospace [90, 110] applications.

1.5 Time Synchronization

Embedded real-time systems keep track of time using dedicated hardware counters called real-time clock (RTC). Each increment of the counter value represents a time step based on the system's clock frequency, typically provided by an oscillator crystal. Two distributed systems will always experience a time offset relative to each other due to oscillator imperfections, temperature differences, different system frequencies and different boot-up times. Thus the RTC of a *system A* might appear to be ticking faster or slower relative to the RTC of a *system B*. The drift from the reference clock is a function of the length of the synchronization cycle and the drift rate of the clocks *A* and *B*. To mitigate this time drift, we must implement a synchronization protocol as shown in Figure 1.7.

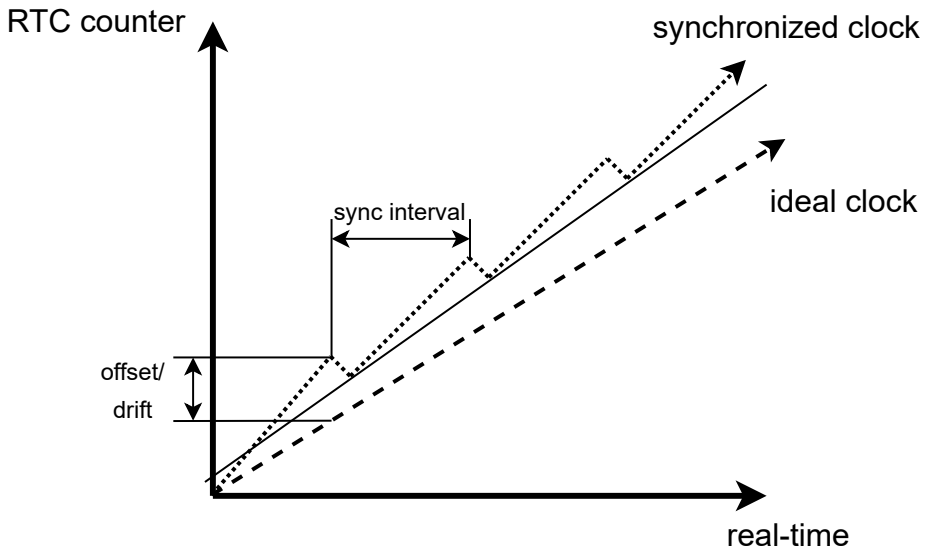


Figure 1.7: Progress of a faster clock that is synchronized relative to an ideal reference clock.

Real-time systems require clock synchronization for numerous reasons as it enables the coordination of distributed tasks by providing a global timebase on which systems can communicate [59], record the chronological order of sensor inputs and messages [53, 49] and schedule the isochronous execution of control loops in cyber-physical systems [38]. Synchronization of local clocks $C(t)$ can be performed in two main ways:

- **External source** meaning that an authoritative reference source $S(t)$ limits the skew to an offset bound $D > 0$ as $|C(t) - S(t) < D| \forall t$.
- **Distributed consensus** allows synchronization of each local clock within a distributed system to an offset bound $D > 0$ so that $|C_i(t) - C_j(t)| \forall i, j, t$.

Moreover different techniques can be applied to enable the measurement of the time offset between remote clocks such as timestamping, message passing and round-trip time. Different algorithms have been investigated in literature over the years and are summarized in the following paragraphs.

Cristian's algorithm Cristian's algorithm [4] presents the simplest form of authoritative synchronization. Assuming an externally synchronized time

server with a GPS or an atomic clock, the clients send requests for the current time and subtract the measured round-trip from the server. The drawback of this algorithm is that it assumes that the response is delivered through the same network as the request and the delay d of the server is not known; thus, its accuracy is bounded by $\pm 1/2T_{round} - d_{min}$. Moreover, it has a single point of failure, and thus, it is not suitable for safety-critical systems.

Berkley algorithm An alternative to Cristian's algorithm is the Berkley algorithm [5]. It does not depend on an external source of synchronization. Instead, it uses a master/slave network hierarchy. The master server periodically polls the slave nodes for their clock readings to estimate the local clock offset based on the round trip estimation. The master averages the values obtained and informs back the slave nodes with the amount of time to adjust their local clocks. If the master fails, a new master is elected.

Network time protocol The network time protocol (NTP) [25] operation is based on a layered client-server architecture that uses user datagram protocol (UDP) message passing with timestamps. The protocol can operate in two main modes, *multicast* and *procedure-call*. In *multicast* mode, the server is responsible for periodically sending the time info to all nodes in the network that update their local clocks, assuming a small transmission delay. In *procedure-call* mode, the NTP operates similar to Christian's algorithm, the server accepts requests from the clients and responds with the current time. In contrast to Berkley's and Christian's algorithms, NTP deploys a four-step message passing with timestamping to estimate the transmission delay and accurately derive the offset of the local clock from the master clock. Filtering algorithms are usually implemented to reduce network-induced jitter, and under particular conditions (i.e., in local area networks and low latency), NTP can achieve sub-millisecond accuracy. However, this is not sufficient for real-time systems, which often require microsecond precision.

Three clock synchronization protocols have been developed recently aiming to improve the performance of NTP for industrial systems and have been widely adopted by safety-critical real-time applications. The capabilities and requirements of these protocols are summarized in Table 1.3.

IEEE 1588 Precise Time Protocol The IEEE 1588 standard introduced the precise time protocol (PTP) as an alternative mechanism to NTP that allows for sub-microsecond clock synchronization on local area networks. PTP is a standard Ethernet protocol that uses a periodic exchange of messages based on master-slave network topology. It operates either on OSI

Table 1.3: Comparison of time synchronization protocols.

Protocols	Link layer	Hardware support	Accuracy	Open standard
TTEthernet (AS6802)	IEEE 802.3	required	microsecond	✗
IEEE 1588	IEEE 802.3	optional	nanosecond	✓
White Rabbit	SyncE	required	picosecond	✓

Layer 2 using Ethernet frames (IEEE 802.3) or Layer 3 using UDP. The protocol allows the precise calculation of each slave's clock offset relative to its master by considering the propagation delay of the message [24]. The recent extension of the protocol with the standard IEEE 802.1ASrev added support for fault-tolerance and cascading topologies. Cascading topologies are achieved with the use of the transparent clock component. Transparent clocks are usually implemneted in bridges/switches and are responsible for compensating the switch traverse time of a PTP message by injecting, in the message, the time from the reception of the frame until its relay.

TTEthernet synchronization protocols TTethernet's synchronization protocol [73] is a distributed clock protocol with multiple masters based on a network-wide time convergence algorithm. The main components of the synchronization protocol are the synchronization master (SM) that is typically implemented in end-systems, the compression master (CM) that is implemented in switches and the synchronization client (SC), which is implemented either in end-systems or switches. To establish a synchronized timebase, the synchronization protocols exchange protocol control frame (PCF)s between end-systems and switches. The PCFs are encapsulated as standard Ethernet frame format. There are two steps involved in the two steps in TTEthernet's clock synchronization algorithm: Firstly, the SMs send PCFs to the CMs. From the arrival points in time of the received PCFs, the CMs extract the current state of the SMs local clocks and estimate an average timebase. The CMs then execute a first convergence function, the so-called compression function. The result of the convergence function is delivered to the SMs and SCs in the form of a new PCF *compressed* PCF. In the second step, the SMs and SCs collect the compressed PCFs from the CMs and execute a second convergence function from which the synchronized time is derived.

White Rabbit Finally, it is worth mentioning the contribution of the synchronization application called White rabbit [54]. White Rabbit is developed

as part of the European Organization for Nuclear Research (CERN) and has demonstrated that sub-nanosecond accuracy over Ethernet is possible even on large scale network composed of thousands of end-systems that spans an area of tens of kilometres, claiming to be the most accurate PTP implementation of the world. The White Rabbit application combines PTP with the Synchronous Ethernet (SyncE) standard over a custom fibre-optic network. On a hardware level, every compatible slave node on the network uses a phase-locked loop to syntonize the phase of its local clock to the master clock transmitted through SyncE. On an application level, it uses the PTP to synchronize the time offset of its local clock to master network time [70], and can achieve sub-nanosecond precision in the range of 135.25 ps with a standard deviation of ≈ 6 ps [69].

1.6 Motivation

Designing a hard real-time system requires, among other things, end-to-end timing analysis, communication and task scheduling, precise time-synchronization and quality-of-control. While significant research efforts have been made to optimize these components, individually, few unified frameworks for time-triggered distributed systems have been proposed. Moreover, with most research relying on COTS platforms and proprietary solutions, even less focus has been given to the architecture design and implementation evaluation of real-time distributed systems experimentally.

Notably, research for time-triggered communication has been based on simulation and proprietary NICs to provide the necessary mechanisms for time-aware frame transmission and reception. More research is needed in providing open-source solutions for time-predictable isochronous communication paths that can guarantee bounded end-to-end communication delays that also consider the WCET of real-time applications. This could include providing network stacks and frameworks for TTEthernet and TSN and optimizing existing network stacks for IP for deterministic execution times. Time-predictable network stacks can enable the development of combined software/hardware time-predictable end-system platforms that can guarantee bounded communication delay. The benefits of isochronous software interfaces for communication have been described in [163]. While early works have presented methods to provide WCET-analyzable network stacks [143]. First efforts to provide an open design for low-latency and minimal jitter with TSN communication have been presented in [68, 78].

Finally, as previously discussed, network-time synchronization is the basis for time-triggered communication as it coordinates the data flow of distributed cyber-physical systems. Network faults or malicious actions can cause loss of precision, leading to data loss and possibly endanger lives even for a few milliseconds of local clock drift. Not all time-aware communication protocols provide fault-tolerant synchronization. It has been shown that reliability issues can lead to catastrophic failures in cyber-physical systems. The IEEE 1588 PTP used in the TSN communication protocol is vulnerable to a range of safety issues, denial-of-service attacks, frame spoofing and even common network failures that can significantly influence the achieved clock synchronization [139]. The author identifies that more research is needed to actively mitigate similar synchronization faults by developing new hardware/software designs integrated with existing communication protocols.

1.7 Thesis Outline

This thesis develops and proposes a collection of practical designs that enable time-predictable communication for hard real-time end-systems and is composed of five papers that are divided into five chapters.

Time-synchronization is explored first in Chapter 2, as it is the fundamental requirement for time-aware distributed computing. A hardware unit and the software driver are developed and presented that enable a computing system to correct its local clock relative to a global network time using the IEEE 1588 PTP. The design is experimentally evaluated on FPGA and compared against existing commercial solutions. Next, Chapter 3 analyzes the effects of network failures on the clock synchronization quality as well as the safety and security aspects of PTP. A fault-tolerant synchronization scheme is designed that can tolerate network failures and denial-of-service attacks. The design is integrated and evaluated in a discrete network simulator using synthetic benchmarks. The design can maintain bounded clock synchronization precision in various scenarios that emulate network failures and malicious attacks.

Using the developed work as a basis, Chapter 4 investigates the time-triggered paradigm by developing an open-source TTEthernet network stack that allows the in-house computing platform to integrate with industrial Ethernet networks. A complete worst-case analysis of the network stack is presented, and the developed platform is evaluated by integrating it into an existing TTEthernet network.

Chapter 5 explores the concept of synchronizing task execution with the underlying communication to achieve an overall time-triggered open-source architecture. An offline static scheduler is developed that leverages satisfiability modulo theories (SMT) to generate schedules for both computation and communication. Moreover, a cyclic dispatcher is developed and integrated with the TTEthernet network stack. The open-source framework is evaluated experimentally using a distributed synthetic benchmark of a multi-periodic control system, comprised of one sensor node, one control node and one actuator node. A worst-case execution time analysis of the framework is presented while the distributed executed tasks are synchronized with microsecond jitter.

Chapter 6 presents an improved experimental evaluation of the proposed framework in Chapter 5, by implementing an avionic benchmark. The benchmark presents a multi-rate longitudinal flight controller case-study that is implemented and scheduled using the proposed framework. The flight controller tasks are distributed over three nodes that communicate over a TTEthernet network switch. The experimental setup is demonstrated to perform a stable flight scenario validated using a set of quality-of-control objectives.

Finally, Chapter 7 concludes this work by summarizing the contributions of this thesis and presenting an outlook of possible future research extensions.

Chapter 2

Hardware Assisted Clock Synchronization with the IEEE 1588-2008 Precision Time Protocol

By Eleftherios Kyriakakis, Jens Sparsø, and Martin Schoeberl [C1]

Abstract

Emerging technologies such as Fog Computing and Industrial Internet-of-Things have identified the IEEE 802.1Q amendment for Time-Sensitive Networking (TSN) as the standard for time-predictable networking. TSN is based on the IEEE 1588-2008 Precision Time Protocol (PTP) to provide a global notion of time over the local area network. Commonly, off-the-shelf systems implement the PTP in software where it has been shown to achieve microsecond accuracy. In the context of Fog Computing, it is hypothesized that future industrial systems will be equipped with FPGAs. Leveraging their inherent flexibility, the required PTP mechanisms can be implemented with minimal hardware usage and can achieve comparable synchronization results without the need for a PTP-capable transceiver. This paper investigates the practical challenges of implementing the

PTP and proposes a hardware architecture that combines hardware-based time-stamping with a rate adjustable clock design. The proposed architecture is integrated with the Patmos processor and evaluated on an experimental setup composed of two FPGA boards communicating through a commercial-off-the-shelf switch. The proposed implementation achieves sub-microsecond clock synchronization with a worst-case offset of 138 ns.

2.1 Introduction

The IEEE 802.1 TSN task group [104] is in the process of standardizing Ethernet into a time-sensitive, deterministic, communication technology, by defining a range of sub-standards based on IEEE 802 networks. TSN is gaining popularity, in automotive and industrial automation networks over commonly used Fieldbus protocols, such as PROFINET and EtherCAT [76]. This is due to its support for: mixed-criticality traffic, high bandwidth and well-defined set of open standards [98, 111] that allow for interoperability between network devices. The deterministic communication of such systems is based on time-scheduled traffic with bound end-to-end latencies [102] and thus it requires a global notion of time to accurately synchronize network operations.

To ensure a global time reference among network devices, TSN employs the IEEE 1588-2008 Precision Time Protocol (PTP) [37] PTP enables accurate clock synchronization over master-slave network hierarchies, where the master is usually equipped with a high precision source of time (i.e., GPS, atomic clock). The worst-case precision of this clock synchronization correlates to the available scheduling accuracy of all network-based operations. There are two key mechanisms that influence the achieved clock precision of PTP, the method of time-stamping and the clock adjustment implementation [24]. These mechanisms are either implemented in software or via dedicated IEEE 1588-2008 compatible Ethernet PHY [86].

In the context of Industry 4.0, it is hypothesized that most upcoming industrial embedded systems, such as Fog Nodes, will be equipped with Field-Programmable Gate Array (FPGA) devices [137]. Based on the current price range of IEEE 1588-2008 compatible PHY transceivers, which is two times more expensive than standard PHYs and the decreasing prices of FPGAs, we argue that by taking advantage of the inherent flexibility of FPGAs, the PTP mechanisms can be implemented in the existing hardware without the need of costly PTP-capable PHY transceivers. These mechanisms can still provide comparable accuracy to

PTP-capable PHY transceivers and at the same time minimize the cost of real-time systems by 30% while provide design flexibility and hardware reusability.

This paper explores the challenges of clock synchronization with PTP and proposes a hardware architecture that combines, a hardware IEEE 1588-2008 clock adjustment unit together with a PTP message recognition and timestamping unit. The proposed hardware is integrated within the FPGA-based platform T-CREST [94] and evaluated using the worst-case execution time (WCET) optimized processor Patmos [145]. The design is evaluated on a simple network composed of two T-CREST nodes, acting as a master-slave PTP pair, and connected through a single commercial-off-the-shelf switch. The achieved clock synchronization between the two nodes is evaluated regarding two metrics, accuracy as average mean and jitter as standard deviation. The results are compared against a WCET analyzable software-based implementation of PTP. The contributions of this paper are:

- A hardware design that allows for network nodes to synchronize with sub-microsecond accuracy using standard PHYs.
- A WCET analysis of the PTP software-stack implementation.
- An evaluation of the achieved synchronization and the identified parameters affecting its accuracy and jitter.

The paper is organized in 6 sections: Section 2.2 provides the reader with a background on the fundamental concepts involved in the IEEE 1588-2008 PTP as well as a short introduction on the T-CREST platform and the WCET optimized processor Patmos. Section 2.3 reviews the challenges of implementing PTP and the different approaches that have been used. Section 2.4 presents the proposed hardware architecture and describes its integration with the T-CREST platform. Section 2.5 describes the experimental setup and presents the evaluation of the collected data from the implementations. Finally, Section 2.6 concludes the paper.

2.2 Background

Different protocols have been developed over the years to achieve time synchronization between networked devices. The Network Time Protocol (NTP) [25] is one the most widely used protocols due to its compatibility with the Internet protocol as well as simple operation. It uses a polling mechanism where devices request the current time from a server to update their own notion of time. This

protocol does not consider the network propagation delays along the path of the time server response and thus it can lead to significant offset and jitter. Filtering algorithms are usually implemented to reduce network-induced jitter and under special conditions (i.e., in local area networks), NTP can achieve sub-millisecond accuracies. However, this is not sufficient for real-time systems which often require microsecond precision.

2.2.1 IEEE 1588-2008 PTP

The IEEE 1588-2008 standard introduced PTP as an alternative mechanism to NTP to allow for sub-microsecond clock synchronization on local area networks. PTP is an Ethernet-based protocol that uses a periodic exchange of messages based on a master-slave network topology. This allows the precise calculation of each slave's clock offset, relative to its master, by considering the propagation delay of the message [24]. PTP is a distributed protocol that requires each ethernet port of an IEEE 1588-2008 compatible network device to execute the same stack of operations and implement the following fundamental blocks:

- IEEE 1588-2008 software stack
- IEEE 1588-2008 clock
- Clock adjustment
- Timestamp capturing
- Frame/Packet recognizer

These blocks can be implemented either in software or hardware and the most common cases are reviewed in Section 2.3. Each IEEE 1588-2008 network port can be either in a PTP_MASTER state or a PTP_SLAVE state. The port's state can be explicitly defined or implicitly by the best master clock selection algorithm. This paper focuses on the clock synchronization between two network devices, and thus the operation of the best master clock algorithm is out-of the scope of this paper. It is assumed that all ports are explicitly defined as PTP_MASTER or PTP_SLAVE.

The calculation of the slave clock offset involves two metrics, *offset* and *delay* which are estimated using four timestamps t_1, t_2, t_3, and t_4 that are generated based on the messages, SYNC, FOLLOW_UP, DELAY_REQ and DELAY_REPLY respectively. There are two synchronization mechanisms supported by the PTP protocol, the one-message model and the two-message model.

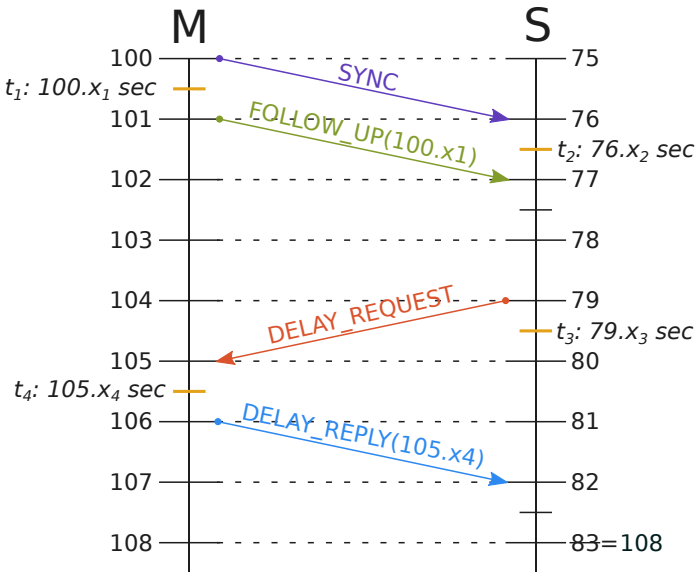


Figure 2.1: Simplified overview of the IEEE 1588-2008 PTP message flow. The example assumes a link delay of 1 second and variable delays x_i associated with each timestamp t_i

The one-message model can be used when high precision is not an application requirement and thus the message FOLLOW_UP containing the precise time of SYNC transmission (t_1) is not sent.

This paper implements the two-message model whose operation is illustrated in Figure 2.1 and described below. The master port **M** is responsible for periodically broadcasting SYNC messages and storing the transmission time in the timestamp t_1. Each SYNC message is followed by a FOLLOW_UP message containing the timestamp t_1. The slave port **S** keeps a receipt timestamp t_2 of the SYNC message and together with the timestamp t_1 contained in the FOLLOW_UP message it can estimate its *offset* from the master clock, but this does not take into consideration the transit time of the received messages. To calculate the propagation *delay* involved in the transmission, the slave port **S** sends a DELAY_REQ message to the master port **M** and keeps a transmit timestamp t_3. The master port **M** that receives the message replies with a DELAY_REPLY message containing the exact time it received the request t_4. The PTP slave can now accurately calculate its offset from the master taking into consideration also the transmission delay involved in their communication path. The use of the gathered timestamps by the slave in the calculations is shown in Equation 2.1 as described in [24].

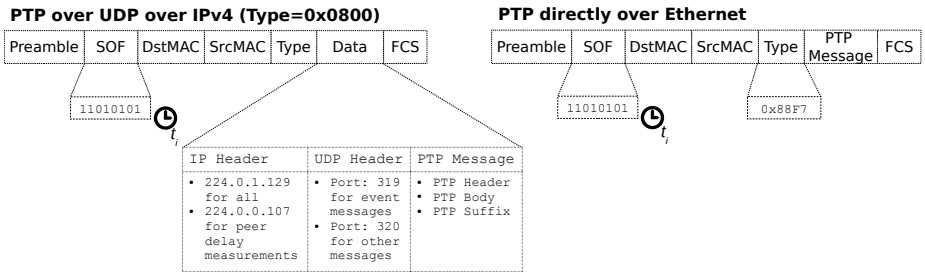


Figure 2.2: PTP Ethernet frame format depending on Ethernet Type. Moment of timestamping t_i is illustrated at SOF byte.

$$\mathit{offset} = t_2 - t_1 - \mathit{delay} \tag{2.1}$$

where:

$$\begin{aligned} \mathit{delay} &= \frac{(dA + dB)}{2} \\ dA &= t_2 - t_1 \\ dB &= t_4 - t_3 \end{aligned}$$

The example PTP synchronization, presented in Figure 2.1, assumes a propagation delay of 1 second and variable delays x_i associated with each timestamp t_i. These delays correspond to the time interval between the actual transmission of a PTP message and the moment the timestamp is captured. Considering the timestamp capture delay variations in Equation 2.1 we calculate:

$$\begin{aligned} \mathit{offset} &= (t_2 + x_2) - (t_1 + x_1) \\ &\quad - \frac{1}{2}\left((t_2 + x_2) - (t_1 + x_1) + (t_4 + x_4) - (t_3 + x_3)\right) \Rightarrow \\ \mathit{offset} &= \frac{1}{2}\left((t_2 - t_1 + t_3 - t_4) + (x_2 - x_1 + x_3 - x_4)\right) \end{aligned}$$

Thus we can derive that the approximate jitter is:

$$\mathit{jitter} = \frac{1}{2}\left(x_2 - x_1 + x_3 - x_4\right) \tag{2.3}$$

As illustrated in Equation 2.3 the need for time-predictable timestamp capturing (timestamping) is a crucial step in calculating the precise clock offset and every component that handles a PTP message, until the timestamp is registered, increases the synchronization error by a small amount. The IEEE 1588-2008 standard defines that timestamping should occur precisely when the last bit of the start-of-frame (SOF) byte of an Ethernet frame is received as illustrated in

Figure 2.2. However, depending on the timestamping technique used this cannot always be achieved. In general, there are three common ways of implementing timestamping as discussed in literature [24]:

1. Software-based timestamping, is handled in software at the reception/-transmission of an Ethernet frame from the MAC layer. The application has to interface with the MAC controller, pack/unpack the frame and check for valid PTP message, or at the transmission of a PTP frame.
2. MAC-based timestamping, is handled by dedicated hardware on the MAC layer, most often implemented by an FPGA or a micro-controller. The hardware unit is responsible for parsing and timestamping the received frame from the PHY.
3. PHY-based timestamping, is handled by a dedicated PHY device that incorporates both an IEEE 1588-2008 Clock and a PTP frame recognition & timestamping unit.

2.2.2 Experimental Platform

The implementation and evaluation described in this paper is built around the open-source platform T-CREST [94].

T-CREST is an FPGA-based multi-core platform that has been developed for on-going research in real-time applications and is based around the WCET-optimized processor Patmos [145]. Patmos is a time-predictable, dual-issue, RISC processor that has been designed with focus on WCET analysis. It uses special WCET-optimized instruction and data caches along with private scratch-pad memories for instructions and data. It is supported by an LLVM-based [18] compiler, also optimized for WCET and by the WCET analysis tool *platin* [89].

The tool *platin* performs static analysis to compute the WCET of a certain code segment by using the information generated and preserved during compilation to determine a control flow graph. Together with low-level timing information of the processor architecture it can calculate a safe WCET of the analyzed code segment.

This work is integrated with the T-CREST platform as part of an on-going effort to provide support for time-triggered communication over TSN networks. Patmos is used to provide a time-predictable execution of the PTP software stack and together with the WCET analysis tool *platin* it is used to identify software-based causes of jitter in the PTP clock synchronization.

2.3 Related Work

This section reviews the challenges and common approaches of implementing PTP, including timestamping and clock correction methods, as well as presents a state-of-art clock synchronization implementation.

Regarding the implementation of a PTP timestamping mechanism, different solutions have been presented that each tries to address different system challenges.

Software-based timestamping is very simple to implement in existing systems as it does not require any additional hardware. However, it can introduce significant jitter since the application runs in user space and its performance cannot be guaranteed. Both the processor load and the delay with handling interrupts or checking flags impact the precise moment of timestamping and lead to error offset and jitter. This is investigated in [30], where it was shown that under special conditions (i.e., low network traffic, high bit-rate connection) this implementation can achieve sub-millisecond clock synchronization.

MAC-based timestamping can be found implemented in modern commercial micro-processors, such as the STM32F105xx or the STM32F107xx [127] where the clock synchronization has been characterized by [67]. To the best of our knowledge, the implementation challenges and performance of this method have not been thoroughly investigated for IEEE 1588-2008 PTP. Related works such as [78] have investigated the concept of packet reception and timestamping using the AS6802 [22, 59] synchronization algorithm but the implementation or the precision of the achieved clock synchronization are not discussed. Designs for hardware-based PTP timestamping have been presented in [58, 57], but they have not been implemented in a real system nor evaluated in terms of their clock synchronization precision.

Finally, PHY-based timestamping has been implemented in commercial packages such as the Texas Instruments PHYTER [86]. This off-the-shelf component has been used in different works [53, 49] where it is presented that it can ensure nanosecond accuracy clock synchronization.

Regarding the clock correction mechanism, the IEEE 1588-2008 standard does not define an algorithm or procedure for adjusting the slave clock despite this having a significant influence on the overall precision of the system. Different approaches have been implemented to adjust the clock including:

- Clock rate adjustment by pulse addition and swallowing. An algorithm has been proposed by [48] and the procedure is also described in [24].

- Error register maintenance instead of clock correction. This technique is discussed in literature [24]. The error register is updated with the current offset from the master clock while the slave clock is never modified.

Finally, PTP has also been investigated in simulation to identify common sources of jitter and their effects on the synchronization as well as to estimate the best achievable clock synchronization in multi-hop large-scale networks [122]. The results show that the precision reduces as the number of hops increases and guaranteed precise synchronization proves challenging as the network scales.

At this point it is worth mentioning that research at the European Organization for Nuclear Research (CERN) has demonstrated the versatility of PTP and has shown that sub-nanosecond accuracy is possible. The developed application called White Rabbit [54] made use of PTP and Synchronous Ethernet standards on a custom network built on fiber optic links. The application achieved sub-nanosecond precision in the range of 135.25 ps with a standard deviation of approximately 6 ps.

This paper differs from related work as it aims to realize and evaluate an FPGA implementable architecture, able to provide nanosecond precision between master-slave node pairs using the IEEE 1588-2008 PTP on standard LAN networks without the use of dedicated PTP-capable Ethernet PHY hardware.

2.4 Design And Implementation

This section presents the hardware architecture that integrates with T-CREST, the proposed hardware-assist logic, for timestamping and clock adjustment, and finally the PTP software stack that is used to control and evaluate the design.

2.4.1 Hardware Architecture

The proposed hardware architecture, presented in Figure 2.3, is integrated with the T-CREST platform as a single IP core that interfaces with the Patmos processor. Its functionality is to snoop on the media independent interface RX/TX channels between the PHY and the MAC controller for PTP messages

and provide times stamps for their arrival and departure accordingly. The unit is composed of three functional entities:

1. The two *RX/TX timestamp units* (TSUs) that parse and timestamp a received or transmitted PTP frame, based on the SOF byte (see Figure 2.2). The units also include an interrupt/flag signal that is raised when a PTP frame has been parsed successfully and a valid timestamp is available for reading.

2. The *IEEE 1588-2008 Clock* is composed of two counters representing seconds and nanoseconds that operate under a pre-scaled frequency. This unit is augmented with the proposed clock adjustment mechanism. In addition, it includes a configurable timer interrupt which can be used to schedule time-triggered network operations.

3. The *PTP software stack*, executing on the Patmos processor, is responsible for the PTP message exchange as well as the offset calculation. The software stack is also responsible for managing the time-stamping, either by reading the *RX/TX timestamp units* or by reading the *IEEE 1588-2008 Clock* as well as for controlling the clock adjustment mechanism.

2.4.2 RX/TX Timestamp Unit

The *RX/TX timestamp unit* (TSU) is presented in Figure 2.4. The hardware unit uses a finite state machine (FSM) to parse Ethernet frames as they are communicated between the PHY and the MAC controller. Incoming nibbles are de-serialized into bytes and consecutively stored into a double-word (64-bit) buffer. The FSM is initialized in the SFD (start-of-frame-detect) state where it waits and checks the double-word buffer until the frame's preamble and SOF sequence (`0x55555555555555D5`) is detected, at which moment it registers the current *IEEE 1588-2008 Clock* time and transitions to the next state to start parsing the incoming frame. Stepping through the FSM is done by keeping a record of how many bytes have been received at each state and resetting the counter when transitioning to a new state. States DSTMAC and SRCMAC read the Ethernet frame's destination and source MAC address respectively. State ETHTYPE is responsible for recognizing the received Ethernet frame's type and it provides support for the two PTP frame formats (see Figure 2.2). After the reception of the Ethernet type the state transitions to the IP state and consecutively to the UDP state where it parses the respective protocol headers. If the FSM detects that an IP header does not contain a valid UDP packet or that the UDP header does not contain a valid PTP message it proceeds

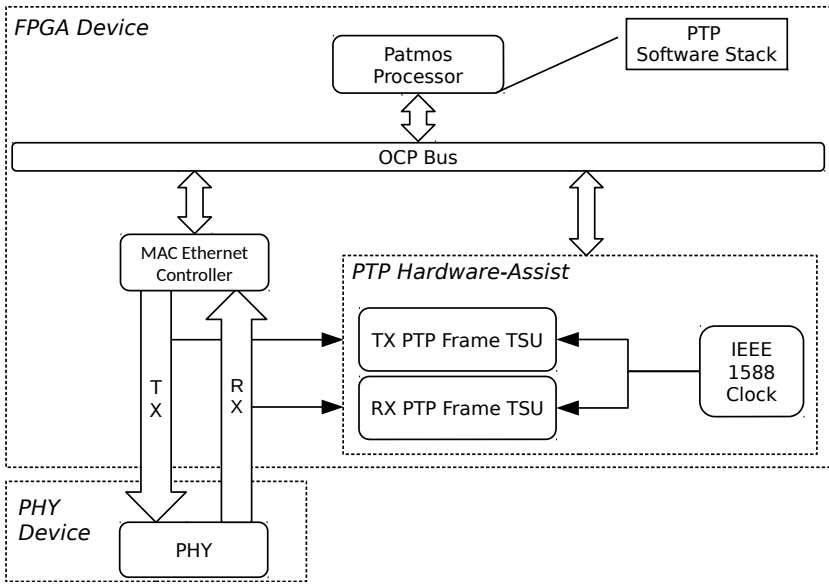


Figure 2.3: Implementation of PTP Hardware-Assist unit inside a T-CREST node. PHY TX/RX signals are split between the MAC controller and the hardware-assist PTP unit.

to transition to state FCS (frame-check-sequence) where it remains until the remaining bytes have been received. When the FSM reaches the PTPHEAD state, it registers the stored IEEE 1588-2008 time along with the PTP message type and a valid bit indicating that a PTP timestamp is available for reading.

2.4.3 Clock Adjustment

The proposed *IEEE 1588-2008 clock & adjustment* mechanism is presented in Figure 2.5, and is composed of three parts:

1. The *clock counter* which consists of a 48-bit nanosecond counter and a 32-bit seconds counter and complies with the time format specified by the IEEE 1588-2008 standard.
2. The *abrupt update register* which can be used to instantaneously update the clock to a specific time value
3. The *offset correction register* which is used to gradually correct the offset by adjusting the clock rate.

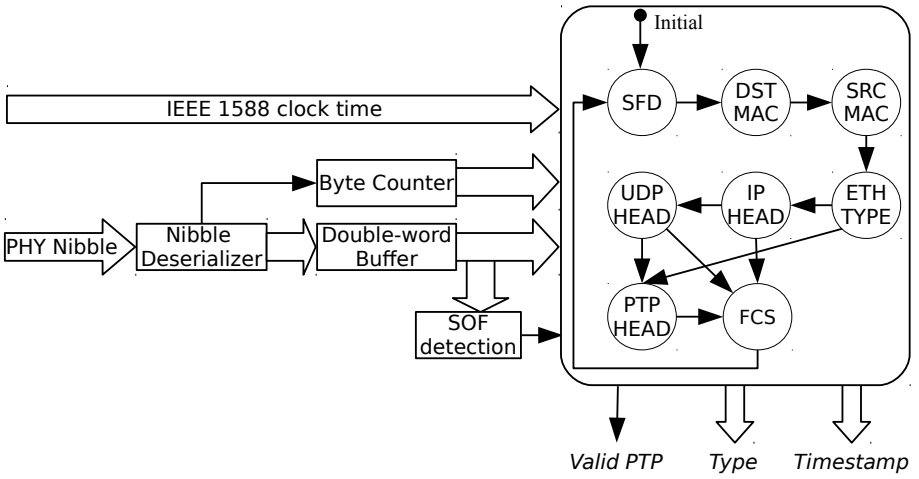


Figure 2.4: Implementation of the proposed RX/TX PTP timestamp unit (TSU).

For large offsets (i.e., when a new node is connected to an already synchronized network or epoch changes), the time can be updated through the *abrupt update register*. For small values, the offset can be written directly in the *offset correction register*. The register indexes a look-up table (LUT) based on a set of configurable thresholds. The threshold values for this implementation are chosen empirically and are divided into the following categories for both positive and negative rates:

- ± 1 ms to ± 1 us
- ± 1 us to ± 100 ns
- ± 100 ns to ± 50 ns
- ± 50 us to ± 1 ns

The LUT controls the amount added/subtracted to or from the base time-step of the *clock counter* nanosecond counter. The operation increases or decreases the *offset correction register* according to the indexed LUT value and stops when its value reaches zero. This mechanism is based on the pulse addition and deletion technique discussed in Section 2.3 and works by gradually correcting the clock offset by increasing or decreasing the rate (time-step) of the *clock counter*. The LUT is indexed by a set of configurable thresholds for the value of *offset correction register*. The LUT rate values are chosen empirically, according

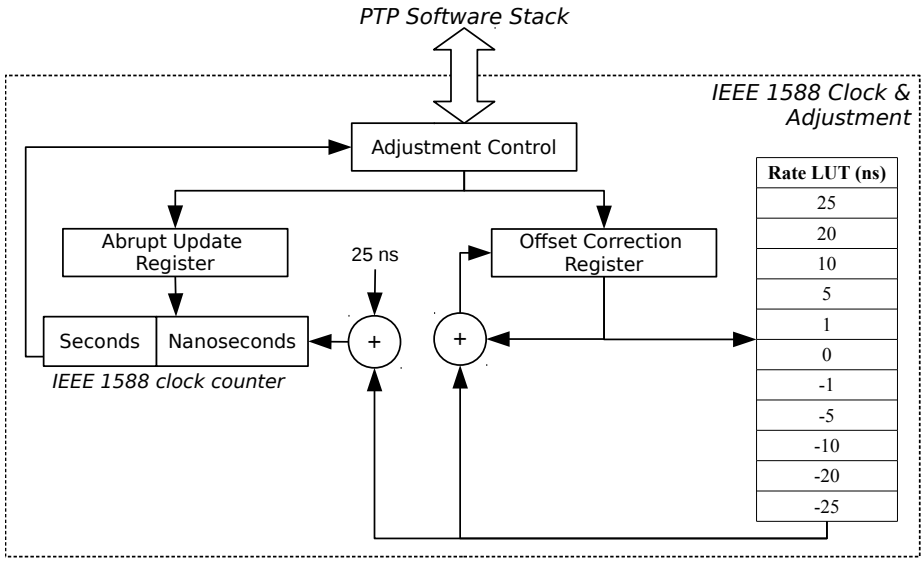


Figure 2.5: Implementation of the proposed IEEE 1588-2008 clock adjustment mechanism. Time-step of 25 ns is related to a clock frequency of 40 MHz and can be tuned accordingly.

to the resolution of the *clock counter* and allow to double the rate or completely stop the counter. Further fine tuning depending on the system's requirements can be applied.

Choosing between correcting the clock offset using the *abrupt update register* or the *offset correction register* is managed in software and can be configured by the `PTP_NS_OFFSET_THRESHOLD` parameter in code.

2.4.4 PTP Software Stack

The PTP software stack runs on the Patmos processor and involves the execution of a simplified PTP protocol were the master/slave mode is explicitly defined. Although open-source software projects that implement the full IEEE 1588-2008 standard are available [93], they are not developed with WCET in mind and are hardly time-predictable, thus they could not be used in our evaluation.

The PTP software stack is responsible for the following tasks:

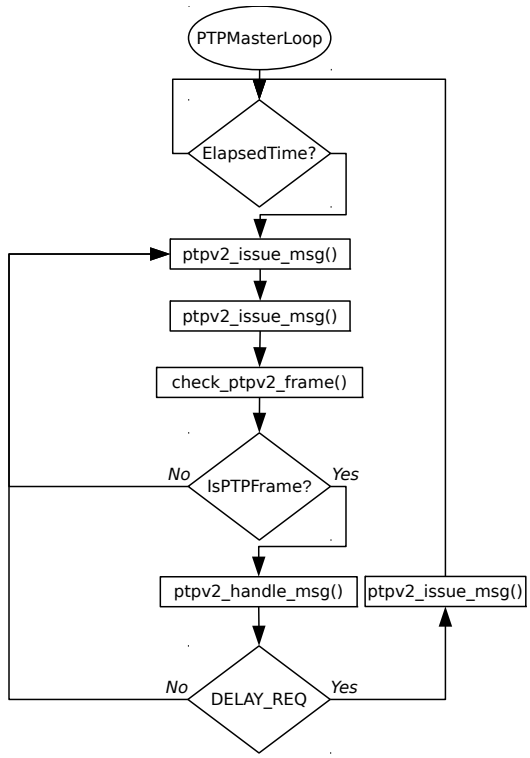
- Initializing Patmos in master or slave port mode.
- Performing the clock synchronization, depending on the port mode.
- Reporting the clock offset at each synchronization interval.

The software is implemented in a way that both the PTP_MASTER and the PTP_SLAVE share the same codebase with the following functions:

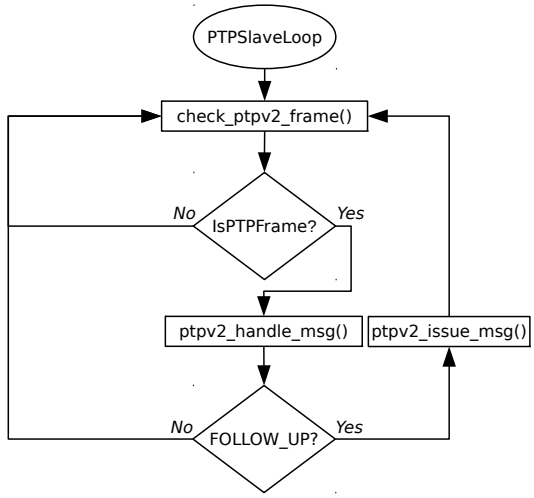
1. `ptpv2_issue_message()` involves creating, sending and timestamping the transmission of a PTP message. Checking the completion of the transmission and registering the timestamp is done by function `read_tx_timestamp()`.
2. `check_ptpv2_frame()`, involves reading the MAC receive buffer, checking the ethernet type field of the frame and responding accordingly. When a PTP frame is detected the function `ptpv2_handle_message()` is called.
3. `ptpv2_handle_message()` involves the unpacking, timestamping and the possible clock offset calculation/correction depending on the received PTP message type. Registering the time of reception timestamp is done by calling the function `read_tx_timestamp()`. Correcting the clock offset is done by first calling the functions `ptp_calc_one_way_delay()` and `ptp_calc_offset()`, for seconds and nanoseconds respectively, and finally calling the function `ptp_correct_offset()` for adjusting the clock.

Since both the PTP_MASTER and the PTP_SLAVE are explicitly defined, their operation was implemented as two simple cyclic procedures presented in Figures 2.6a & 2.6b respectively. The PTP_MASTER is responsible for issuing SYNC and FOLLOW_UP messages at a fixed rate as well as checking for any received PTP messages and replying to DELAY_REQ messages. The PTP_SLAVE is responsible for checking for any received PTP messages and, if a FOLLOW_UP message is received, replying with a DELAY_REQ message.

As presented in [30], multi-tasking can have a negative impact on the clock synchronization precision of software-based PTP. Taking this into account, to allow us to compare the performance of the proposed hardware-assist mechanisms with the best-case software execution, the application is implemented on a single task environment on the Patmos processor.



(a) PTP_MASTER loop



(b) PTP_SLAVE loop

Figure 2.6: Program flow of PTP master and slave

Table 2.1: Hardware-assist Architecture Resource Utilization

Entity	Combinational LUTs	Logic Registers
PTP Hardware-Assist	1485 (82)	1182 (142)
MIITimestampUnit	454 (376)	402 (327)
DeserializePHYbyte	13 (13)	11 (11)
DeserializePHYBuffer	65 (65)	64 (64)
RTC	431 (431)	234 (234)

2.5 Evaluation

This section presents the experimental setup over which the proposed hardware architecture was evaluated as well as its hardware resources. Finally, the collected results from the WCET analysis and the clock synchronization are discussed.

2.5.1 Experimental Setup

The presented hardware architecture was synthesized on two FPGA Terasic DE2-115 boards and explicitly configured as a PTP master/slave pair. The clock synchronization was evaluated on a simple experimental setup composed of the two FPGA boards communicating over a single off-the-shelf switch via a 100 Mbps Ethernet. Each FPGA board used a PLL to generate the internal logic clocks. The Patmos processor was operating at frequency of 80 MHz. The *IEEE 1588-2008 clock* was operating at a frequency of 40 MHz and had a resolution of 25 ns. The PLL input was provided by a commercial off-the-shelf oscillator operating at a nominal frequency of 50 MHz with an accuracy of 50 ppm.

2.5.2 Hardware Resources

The hardware was synthesized for an Altera Cyclone IV FPGA [103]. Table 2.1 presents the hardware utilization of the proposed PTP Hardware-Assist IP. The values outside the parentheses indicate the aggregate resources used by the entity, while the value inside the parentheses indicate the utilization of the specific entity alone.

Table 2.2: WCET Analysis of PTP Software Stack

Function	WCET	
	Clock Cycles	Time (at 80 MHz)
`ptpv2_issue_msg()`	2560141	32 ms
`readTXTimestamp()`	5	62.5 ns
`check_ptpv2_frame()`	684	8.55 us
`ptpv2_handle_msg()`	3893	48.6 us
`readRXTimestamp()`	5	62.5 ns
`ptp_correct_offset()`	66	850 ns
`ptp_calc_offset()`	4	50 ns
`ptp_calc_one_way_delay()`	7	87.5 ns

The hardware cost of the proposed hardware-assist unit is minimal. The utilization of PTP Hardware-Assist is only 1.7 % of the total available resources of the Cyclone IV FPGA device (114480 Logic Elements) and when compared to the the small-sized processor Patmos, it is 11 % of its total size (13503 LUTs and 8325 Logic Registers).

2.5.3 WCET Analysis

To reveal possible sources of jitter, as well as to estimate the processor load involved in the execution of the PTP software-stack, a formal WCET analysis was performed using the tool *platin* [89] and the results are presented in Table 2.2.

The WCET revealed that the worst-case delay between the arrival of a PTP message and the software capturing the timestamp to be 8.6 us. This includes the software packing/unpacking the frame plus registering the time of arrival/departure that amounts to $685 + 5 = 689$ WCET clock cycles. As shown in Equation 2.3 this can lead to significant error in the calculated clock offset and consecutively lead to jitter. If hardware-based timestamping is used, the worst-case delay of 689 clock cycles does not introduce any jitter, since the timestamp has already been captured in hardware and thus the software only needs to read the stored value.

Furthermore, the WCET analysis of functions `ptpv2_issue_msg()` and `ptpv2_handle_msg()` showed that there is a significant overhead in the processor to execute the PTP protocol.

We propose as future work a complete in-hardware implementation of the PTP synchronization. Building up from the presented results, the proposed hardware-assist architecture can be extended with the addition of a PTP message generator controller. This is hypothesized to both minimize jitter but also significantly reduce the processor load especially in multi-tasking environments where the precision of PTP can be reduced significantly [30]. This will also allow for greater scalability on large scale networks, were devices require more than one ethernet ports to synchronize using PTP.

2.5.4 Clock Synchronization

To evaluate the clock synchronization, the PTP_SLAVE was configured to report, over serial a port, the calculated clock offset at each PTP synchronization interval (after all four timestamps were gathered). To best evaluate the performance of the proposed implementation, four sets of results were collected by testing different combinations of implementation, namely:

- software-based timestamping
- hardware-based timestamping
- abrupt clock updates
- clock rate control adjustment

As a base of comparison to the evaluation of the results and to determine the static drift between the master and slave clocks on the two FPGA boards, measurements were gathered using PTP but without implementing any corrections or adjustments to the slave clock. Figure 2.7 presents the PTP slave's clock offset as calculated after an initial correction and no further adjustments. The relative drift was estimated at an average of 34 us/sec, which corresponds to 34 ppm and illustrates the need for accurate synchronization.

First, the effects of the proposed hardware-based timestamping against software-based timestamping were compared in terms of jitter as standard deviation. Figure 2.8 presents the results from 18000 collected samples from two different measurements with a *SYNC* period of 0.5 ms. The calculation used only abrupt updates for correcting the clock offset, to clearly reveal the influence of the timestamping mechanism. The hardware-based timestamp mechanism managed to reduce jitter to a standard deviation of 49.8 ns, while software-based timestamping could only achieve a standard deviation of 95.3 ns.

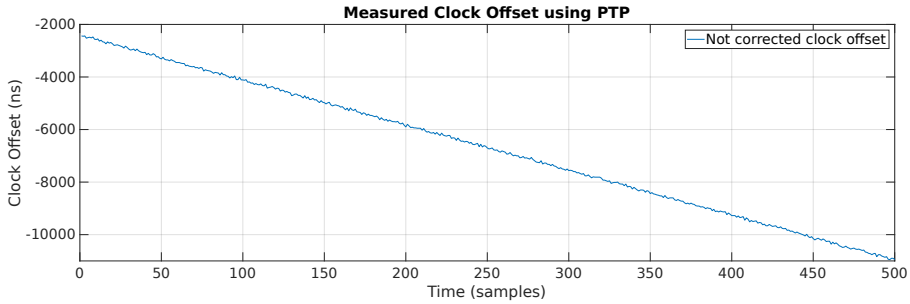


Figure 2.7: Measured clock offset between the two FPGA boards (using hardware-based time-stamping).

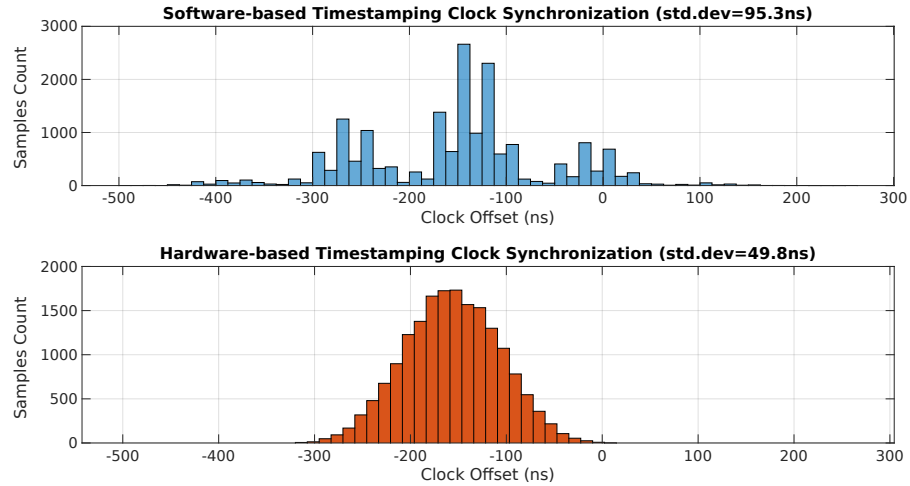


Figure 2.8: PTP-Slave clock offset timestamping method comparison between software-based (top) and hardware-based (bottom).

Secondly, the effects of the proposed rate control mechanism were investigated in terms of accuracy (avg. mean) of the calculated offset. The measurements presented in Figure 2.8 show that there is an avg. mean offset of 154 ns. This offset is introduced by the time it takes to read the clock, add the delay and write-back the new value into the clock. Figure 2.9 presents and compares the improved avg. mean using the proposed rate control mechanism against the achieved accuracy using only abrupt updates. The data were collected from two different measurements that both used a 0.5 ms *SYNC* message period and hardware-based timestamping. The achieved achieved accuracy of rate-control was within 17 ns with a std. deviation of 48.7 ns.

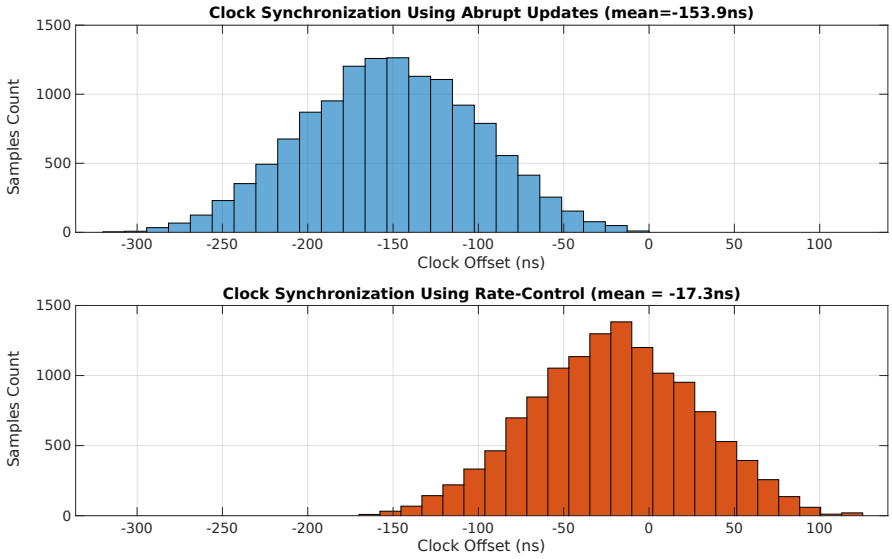


Figure 2.9: PTP-Slave clock offset adjustment method comparison between abrupt-updates (top) and rate-control (bottom).

The results show that the timestamping method influences the jitter of the calculated offset, while the clock adjustment method was mainly responsible for the accuracy of the achieved synchronization but also revealed a slight improvement in jitter.

The proposed PTP Hardware-Assist implementation offers sub-microsecond clock synchronization of an avg. mean of -17.3 ns and a jitter of approx. 48.7 ns with a worst-case clock offset of 138 ns. The results improve the worst-case synchronization offset of 500 ns that was achieved when using only software-based timestamping and abrupt updates. The presented results also achieve better worst-case offset when compared to the related architecture of the STM32F107xx microprocessor, which as characterized by [67] it achieves a worst-case offset of 260 ns. Moreover, the achieved performance is comparable to the clock synchronization of a commercial PTP-capable Ethernet PHY transceiver that is presented in [53, 49], and allows for an avg. mean of 10 ns and a jitter of approximately 50 ns.

We propose as future work to extend the experimental setup and evaluate the presented architecture over a large-scale TSN network composed of multiple T-CREST nodes. It is hypothesized that traffic load and multiple-hops will have a negative effect on the clock synchronization precision between network nodes due

to an increased transit time of PTP messages, as shown in the analysis of [122] [30]. In addition we plan to increase the resolution of the *clock counter*. Finally, the WCET analysis highlighted the need for hardware-based timestamping as well as revealed how time intensive the execution of PTP is. Based on the results we plan to investigate a complete in-hardware solution for performing the PTP synchronization, including transmission of PTP frames, that will effectively reduce the processor load of network devices as well as provide transparent to the user clock synchronization.

2.5.5 Source Access

The presented hardware-architecture is integrated with the open-source project T-CREST which is hosted at [128]. The PTP Hardware-Assist design can be found at *https://github.com/t-crest/patmos/tree/master/hardware/src/main/scala/ptp1588assist*. The PTP software is part of the ethernet lib of Patmos and is available at *https://github.com/t-crest/patmos/tree/master/c/ethlib*. To monitor the live offset from a connected PTP_SLAVE T-CREST node, a visualization script was developed available at *https://github.com/t-crest/patmos/blob/master/c/ethlib/other/plotPTPOffset.py*

2.6 Conclusion

This paper investigated the IEEE 1588-2008 Precise Time Protocol, which provides a global time reference for IEEE 802.1 TSN networks. We explored the design of a hardware-assisted implementation of PTP using a standard Ethernet PHY transceiver and finally implemented a hardware architecture that can achieve sub-microsecond precision.

The proposed architecture was successfully integrated with the T-CREST and synthesized on FPGA with minimal hardware overhead. The PTP software stack was implemented on the time-predictable Patmos processor which allowed for a full WCET analysis of the application.

The clock synchronization was evaluated on an experimental setup, composed of two FPGA boards implementing the proposed architecture and communicating through a commercial off-the-shelf switch at 100 Mbps. The evaluation was performed using two metrics, jitter as standard deviation and accuracy as average mean, and data were collected over a variety of timestamping and clock adjustment combinations.

The results showed that the proposed architecture greatly improved the precision of software-based timestamping and it achieved comparable results with commercial off-the-shelf PTP-capable Ethernet PHY transceivers, showing that FPGA-based PTP clock synchronization is feasible.

Acknowledgment

This is work was part of the Fog Computing for Robotics and Industrial Automation (FORA) European Training Network (ETN) funded by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 764785.

Chapter 3

Fault-tolerant Clock Synchronization using Precise Time Protocol Multi-Domain Aggregation

By Eleftherios Kyriakakis, Koen Tange, Niklas Reusch, Eder Ollora Zaballa, Xenofon Fafoutis, Martin Schoeberl, and Nicola Dragoni [C3]

Abstract

Distributed real-time systems often rely on time-triggered communication and task execution to guarantee end-to-end latency and time-predictable computation. Such systems require a reliable synchronized network time to be shared among end-systems. The IEEE 1588 Precision Time Protocol (PTP) enables such clock synchronization throughout an Ethernet-based network. While security was not addressed in previous versions of the IEEE 1588 standard, in its most recent iteration (IEEE 1588-2019), several security mechanisms and recommendations were included describing different measures that can be taken to improve system security and safety. One proposal to improve security and reliability is to add redundancy to the network

through modifications in the topology. However, this recommendation omits implementation details and leaves the question open of how it affects synchronization quality.

This work investigates the quality impact and security properties of redundant PTP deployment and proposes an observation window-based multi-domain, PTP end-system, design to increase fault-tolerance and security. We implement the proposed design inside a discrete-event network simulator and evaluate its clock synchronization quality using two test-case network topologies with simulated faults.

3.1 Introduction

Modern Cyber-Physical Systems (CPS) are becoming increasingly connected to the Internet through the advancements of Fog Computing and Industrial Internet of Things. Thus nowadays, security becomes an essential factor in the design of such systems, in addition to the traditional safety and reliability requirements [125, 74].

Time-triggered communication is often used in distributed CPS that require strict guarantees on the timing of messages. Such systems need a high precision global notion of time to be shared among the nodes in the network to achieve synchronous scheduled communication and computation [121, 162]. Time-Sensitive Networking (TSN) [104] is a newly developed standard that aims to enable deterministic real-time communication for mixed-criticality traffic while preserving the high-bandwidth capabilities of Ethernet. It is developed by the TSN Task Group as an extension to the 802.1 Ethernet standard and consists of many sub-standards for different components. TSN uses a profile (802.1 AS-Rev [99]) of the IEEE 1588 Precision Time Protocol (PTP) standard [39] to enable accurate clock synchronization. Although PTP has been in use for decades, recent research indicates that this protocol's security and safety aspects have been overlooked, leaving it vulnerable to time synchronization attacks [151]. An attack on an automated factory's network time would disrupt the communication and computation schedule leading to missed deadlines and messages. This could have catastrophic consequences both in the production line and operating machinery, as well as possibly endanger human lives or the environment.

To address some of these issues, the IEEE Precise Networked Clock Synchronization Working Group has included various security measures in the updated IEEE 1588-2019 standard [160]. This updated standard proposes several measures involving redundancy to mitigate security and safety issues due to unavailable links. To support the proposed redundancy, the standard recommends using a

voting algorithm to derive a converged clock offset from the multiple domains. However, no further information is given, leaving the algorithm's choice and its implementation to the user.

While distributed consensus and voting algorithms are extensively studied [100], to our knowledge, no such work exists in the context of highly time-sensitive PTP networks. We explore the concept of fault-tolerant clock synchronization within TSN and propose a multi-domain synchronization scheme that uses redundant paths combined with frame aggregation and a time-based observation window to achieve secure and fault-tolerant operation. We evaluate the proposed approach by simulating three test-case network topologies in a discrete-event network simulation tool OMNeT++ [42]. The achieved clock synchronization is compared against standard PTP end-systems and evaluated regarding two metrics, accuracy as average mean and jitter as the standard deviation of the clock offset. The proposed multi-domain design is able to preserve microsecond precision despite the existence of network failures. The contributions of this paper are:

- A fault-tolerant PTP end-system design that supports multiple synchronization domains.
- A timed observation window mechanism that aims to increase security by filtering received frames.
- A comparative analysis of clock synchronization quality in different test-case scenarios with faults.

The remainder of this paper is structured in 6 sections: Section 3.2 presents the fundamental concepts of PTP and fault-tolerant synchronization and introduces the problem statement. Section 3.3 discusses the related work in PTP security and fault tolerance. Section 3.4 presents the proposed multi-domain end-system architecture and discusses the required network topology. Section 3.5 evaluates the proposed multi-domain design and compares its performance against the standard PTP mechanisms by simulating different test-cases with synthetic scenarios. Section 3.6 provides a discussion on the safety and security implications of the proposed multi-domain aggregation mechanism. Section 3.7 presents the planned future extensions of this work. Section 3.8 summarizes the presented work and concludes the paper.

3.2 Background

3.2.1 Fault-Tolerant Clock Synchronization

Precise and fault-tolerant time synchronization is an operational requirement of distributed safety-critical real-time systems, such as those found in aerospace and automotive industry. Redundancy is the key to tolerate Byzantine faults in these systems, as any master clock can exhibit arbitrary behaviour and provide false readings of its local clock to connected systems. Consequently, slave clocks can misinterpret this information either because accurate convergence algorithms have not been implemented or simply because the in-place redundancy is not sufficient. It can lead to drift in the relative clock offset of the network-wide time base.

This effect has been described in research [6, 7], where the authors have analyzed the need for $3f + 1$ nodes available in a distributed system that can tolerate f faults and have provided static bounds for different convergence algorithms. The most predominant algorithm of these is the Fault-Tolerant Average (FTA), which was first introduced in [1] and is incorporated in the fault-tolerant clock synchronization of TTEthernet [73] that is now part of the aerospace standard AS6802 [59]. In this work, we try to incorporate FTA principles in PTP and evaluate its performance in TSN networks as a means to provide fault-tolerant multi-domain clock synchronization.

3.2.2 IEEE 1588-2019 Precise Time Protocol

PTP is a hierarchical clock synchronization protocol based on a periodic exchange of Ethernet frames that estimates the clock offset between end-system ports configured as slaves and masters [24]. Typically, a PTP stack is assigned to each PTP port and is responsible for executing the protocol. A mechanism, called a clock servo, is responsible for correcting the device clock using a proportional-integral filter [32]. The PTP stack on a slave port calculates the time difference from a master by collecting four timestamps using four respective frames:

1. SYNC, from master to slave
2. FOLLOW_UP, from master to slave
3. DELAY_REQ, from slave to master

4. DELAY_REPLY, from master to slave

Moreover, precise time-stamping of the received/sent frames is a crucial part of the protocol, as it directly influences the precision of the estimated clock offset. Select hardware units can be used for this purpose [138] . In the rest of this work, we assume that such time-stamping units are available in all end-systems.

PTP allows for multiple masters to exist, but only one master's synchronization frames are used to calibrate an end system's internal clock at each given synchronization cycle. This selection is made using the best master clock algorithm (BMCA). The BMCA works by comparing an arbitary value, which represents the remote clock quality, connected network end-systems advertised that in dedicated periodic frames called ANNOUNCE frames. From this information it derives the best clock and then it compares that to the quality of its local clock to determine its role as a master or a slave.

The IEEE-1588-2019 standard adds several security features to PTP [160]. Most notably, it adds support for multiple types of authenticated encryption, addressing many of the security concerns that were present in its predecessor, IEEE-1588-2008 [39]. However, none of the introduced features protects against delay attacks, nor do they consider faulty master nodes (e.g., compromised by a malicious party). A malicious master node might try to influence the system time by announcing high accuracy during the BMCA, subsequently moving the time window once it has been elected. Delay attacks assume that an attacker can control a link, and might delay messages for an indefinite amount of time. To mitigate the impact of such attacks, the IEEE-1588-2019 standard only includes two recommendations: to deploy redundant master clocks; and to deploy redundant network topologies. The first recommendation works without any alterations to the protocol. As the PTP protocol is a distributed algorithm, it will eventually select one of the redundant master clocks if the primary one fails; however, this can introduce significant time overhead that leads to jitter. The second recommendation stands out: a PTP system distils a logical minimum spanning tree topology with the elected master clock as a root, and all slaves (i.e., consumers of the synchronization signal) as leaves. A minimum spanning tree does not allow multiple paths between any two nodes to exist by its very definition. The solution to this is to run multiple PTP domains in parallel, ensuring that they choose different physical network paths for their tree topology. A PTP domain is a numerical identifier included in every protocol message. It allows multiple PTP systems to operate on one network without interfering with other PTP systems.

A multi-domain setup combines neatly with the first recommendation of using multiple master clocks. With multiple parallel domains, every slave system

needs to execute a deterministic voting algorithm to arrive at the same approximate time. However, the IEEE-1588-2019 standard does not recommend any voting algorithms. Additionally, it is left unclear what the performance impact and effectiveness of these measures will be. Therefore, we attempt to fill this knowledge gap by analyzing two voting algorithms' performance in a simulated PTP system. Further, we explore the impact of link failures on timing accuracy during the execution of a PTP system both with and without redundancy in place.

3.3 Related Work

In the broad spectrum of network attacks related to PTP, disrupting the synchronization is the primary goal. Lack of message authentication is one of the main attack vectors to break master-slave synchronization. The authors of [151] analyze the security risks associated with PTP by building a testbed that shows synchronization disruption between PTP devices. The tests conducted include master spoof attacks (spoofing ANNOUNCE and SYNC packets), ANNOUNCE DoS attacks (spamming target slave) and master clock takeover attacks. Similarly, Lisova [139] presents a threat model that shows an attack classification that lists several PTP clock synchronization attacks (e.g. replay and delay attacks, flooding/DoS) that target availability among other factors. Lisova proposes a distributed monitoring strategy to detect if an attacker is affecting clock synchronization. While both studies [151] [139] point out existing threats to availability, the current work provides a fault-tolerant design to guarantee availability.

To tackle some of the attacks mentioned earlier, IPSec and MACSec have already been analyzed for time synchronization [55] to provide authentication, encryption, and confidentiality. However, neither provide any availability guarantees or fault tolerance against a compromised endpoint or delay attacks. We consider IPSec and MACsec complementary to the fault tolerance algorithms and mechanism discussed in this work.

In 2014, Mizrahi published an informational Request For Comments (RFC) with the requirements to secure time protocols in packet-switched networks [79]. The document presents a threat model and threat analysis that lists several attack types such as packet manipulation, spoofing or replay attacks. It focuses on listing minimum security requirements such as authentication, authorization, confidentiality. While these requirements could create a security basis for next versions of time synchronization protocols, they do not guarantee availability. Additionally, the document briefly references a few mechanisms to protect

against delay attacks or attacks that degrade clock accuracy, such as using of multiple paths [63]. This RFC also proposes that outliers in received time values should be considered erroneous and be ignored. The current study aims to fill the gap of fault tolerance, resilience and availability that the RFC does not cover. Specifically, it presents an implementation and evaluates its resilience to faults.

Mizrahi presents the concept of *slave diversity* [63] to obtain high clock accuracy and reduce time error using multiple paths. Similarly, Shipiner et al. present a multi-path approach [72] that evaluates path diversity. While both studies demonstrate the applicability of multiple path time synchronization, there are significant differences with this work. First, Mizrahi [63] does not tackle the master redundancy and availability features into fault tolerance and Shipiner et al. [72] do not provide simulation and performance results. In contrast, this work, uses different PTP domains with multiple masters to guarantee availability and evaluates the fault-tolerance in simulation.

3.4 Multi-domain Node and Algorithm Design

Our proposed approach consists of multiple design elements and considerations spread over multiple layers. Firstly, we introduce a redundant variant of a typical PTP node. A node's ability to interact with singular (i.e., non-redundant) nodes is preserved, leaving room for hybrid PTP systems. Secondly, we discuss network topology requirements that should be taken into account when designing redundant PTP systems. Finally, we describe the implemented convergence algorithms.

The design proposed in this section aims to mitigate link failures and protect against Byzantine actors on the network, but it does not guarantee the communicated messages' integrity or authenticity. It is intended to complement existing resilience and security features proposed by the IEEE-1588-2019 standard, which provide these properties.

3.4.1 Node Architecture

A redundant PTP node has to support running PTP on n domains at once. To this end, we design a node architecture that maintains n parallel PTP stacks and aggregates their computed offsets. As is usual for Byzantine fault-tolerant systems, to protect against f faults, n should be picked as $n = 3f+1$. Figure 3.1

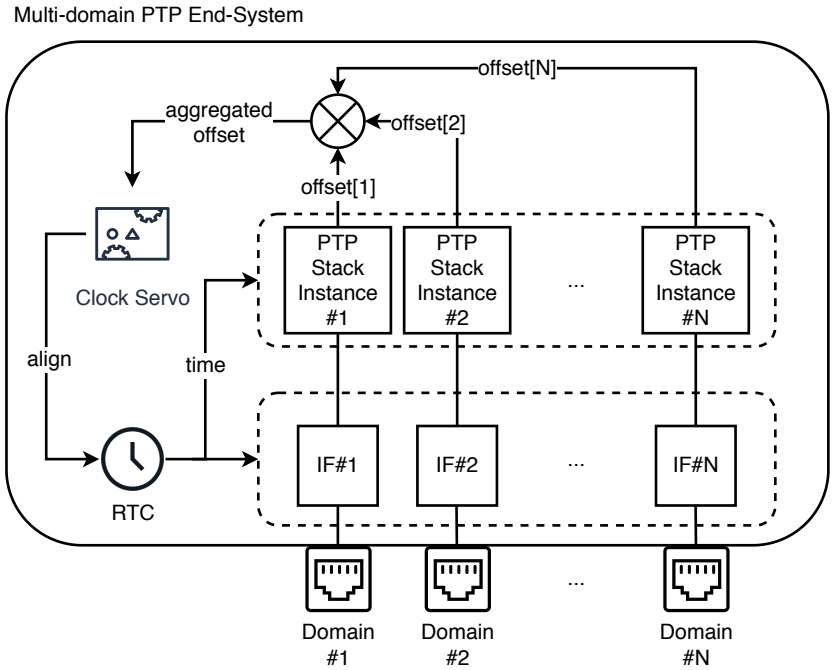


Figure 3.1: Extended PTP end-system architecture to support multi-domain aggregation. Each domain uses a separate network interface and PTP stack. The calculated offsets are fed into an aggregation function, which corrects the clock.

presents the design of the proposed node. Each PTP stack is assigned an individual network interface port and executes isolated from the others. Using only one network interface is possible, but it would turn this into a bottleneck and the weakest link for each node. If the node is a slave, each stack periodically receives PTP messages that have to be aggregated somehow. Each stack distils an offset from the incoming messages. The calculated offset is combined with the latest PTP frame ingress timestamps as a tuple and fed into a convergence algorithm. If the node is a master node, it simply has to transmit PTP messages on every domain.

The convergence algorithm aggregates the most recently received offsets for each domain within an observation window, and produces a single aggregated offset correction for the real-time clock (RTC). This convergence algorithm is transparent to the PTP stacks, the clock servo, and any applications depending on the synchronized time of the RTC.

3.4.2 Network Topology

To effectively mitigate link/node failures and malicious PTP actor nodes, network paths for each domain should be entirely disjoint. Therefore, one can specify the main goal for the network topology is to introduce redundancy where possible. The observation window should be tuned according to the maximum expected latency of all the redundant domain paths. Thus, to minimize the observation window span , a design using redundant network paths should strive to preserve a symmetric topology with the same number of hops between slaves and master nodes. Further optimization on the asymmetry of links has been investigated by [11, 40]. Note that while a fully symmetric topology describes an ideal situation, it is not an explicit requirement. A symmetric topology allows for balanced network delays with equal worst-case end-to-end latency (WCEL), and thus it is hypothesized to lead to better convergence algorithm performance. In the remainder of this work, we thus assume a fully symmetric topology to explore the ideal case.

3.4.3 Convergence Algorithms

The convergence algorithm is run on each PTP slave node individually and takes as inputs a collection of latest observed offsets from each domain PTP stack. We implement two different convergence algorithms for evaluation. The first offset aggregation algorithm is a simple averaging function (AVG) over the available

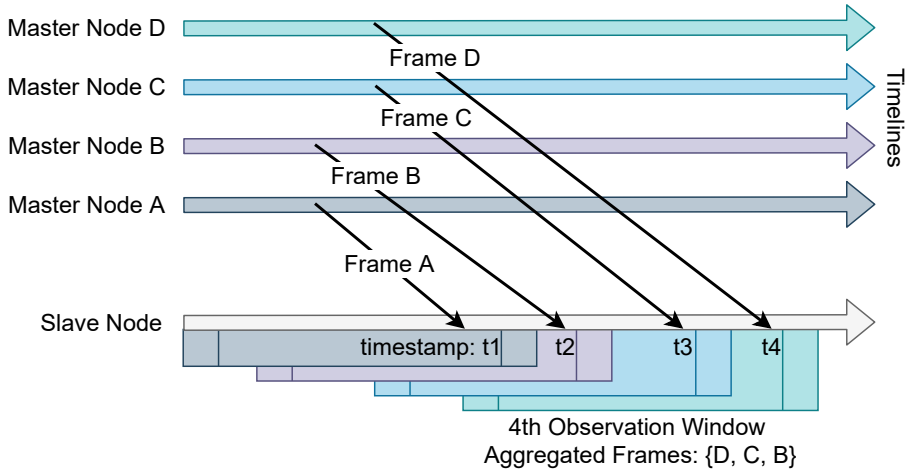


Figure 3.2: Observation windows are generated by new SYNC/FOLLOW_UP frames. Received frames that are within the time window are used in the aggregated offset calculation.

offsets. The second algorithm implements a Byzantine Fault Tolerant approach (FTA) for clock synchronization.

3.4.3.1 Observation Window Filtering

Figure 3.2 illustrates how individual PTP frames from the different PTP stacks are converged by initiating separate observation windows. Each received SYNC, or FOLLOW_UP frame initiates a new observation window based on the ingress timestamp over which the convergence algorithm operates. Only frames within an observation window time are taken into account to calculate the converged clock offset for that specific point in time. The duration of the observation window, controls the accepted time difference threshold of the received master frame timestamps from the last received PTP frame timestamp. This parameter should be tuned proportionally to the WCEL that the PTP master frames can experience, i.e. the longest path delay between a redundant master and the receiving slave.

The windowed decision algorithm is listed in Algorithm 1. This algorithm takes a new (incoming) offset o from its local clock as input, together with its ingress time i and PTP domain d. The algorithm's output is an approximate offset and ingress time, which can be used by the clock servo to correct the RTC. First, it stores the tuple (o, i) in a table structure using the domain as an index, ensuring

Algorithm 1 Windowed Decision Algorithm

1: **procedure** WINDOWEDDECISION(o, i, d) ▷ Executes a windowed decision algorithm using the latest received timestamps
State: S ▷ A table $d \rightarrow (o_d, i_d)$ mapping all domains $d \in D$ to (offset, ingress) tuples
2: $S[d] \leftarrow (o, i)$
3: $S' \leftarrow \{x \rightarrow (o_x, i_x) \in S \text{ where } |i - i_x| \leq WINDOW\}$
4: $i_a \leftarrow \begin{cases} 0 \text{ if } |S'| = 0 \\ \frac{\Sigma_{x \in S'} i_x}{|S'|} \text{ otherwise} \end{cases}$
5: $o_a \leftarrow \text{FTA}(S')$ or $\text{AVG}(S')$
6: **return** (o_a, i_a)
7: **end procedure**

that only one offset per domain is considered. After this, the table S is filtered to S', excluding offsets that were not received within a given delta `WINDOW` from the new ingress timestamp. Then, the ingress of all offsets in S' are averaged to i_a, and an approximate offset o_a is calculated using either the FTA or the AVG algorithm.

3.4.3.2 Averaging Algorithm (AVG)

The AVG consists of a simple averaging function that extracts all offsets from the given map and returns the average of these, or 0 if there are no offsets.

3.4.3.3 Fault Tolerant averaging Algorithm (FTA)

The FTA [1] is an algorithm that provides bounded clock synchronization even in the presence of faulty and possibly malicious master clocks (see also Section 3.2). Algorithm 2 describes the implemented FTA algorithm.

In the general case where k faults should be tolerated, this algorithm drops the earliest and last k offsets and averages the remaining offsets. First, usable offsets are extracted from the given map structure S', and special assignments are made for the $2k$ most extreme offsets. Then, it distinguishes 3 cases: firstly, if there is only one offset, we return that offset; secondly, if there are only two offsets, their average is returned, and finally, if there are three or more offsets, it drops the extremes and returns the average of the remaining offsets.

Algorithm 2 Fault Tolerant Algorithm

1: **procedure** FTA(S) ▷ Executes a fault-tolerant convergence algorithm over a set of offsets
2: **if** $|S| = 0$ **then**
3: **return** 0
4: **end if**
5: $O = \{o_x | x \in S\}$
6: $o_{min} \leftarrow k$ earliest offsets in O
7: $o_{max} \leftarrow k$ latest offsets in O
8: **if** $|O| = 1$ **then**
9: **return** o_{min}
10: **else if** $|O| = 2$ **then**
11: **return** $\frac{o_{min}+o_{max}}{2}$
12: **else if** $|O| \geq 3$ **then**
13: $O' \leftarrow O \setminus \{o_{min}, o_{max}\}$
14: **return** $\frac{\Sigma_{x \in O'} x}{|O'|}$
15: **end if**
16: **end procedure**

The first two cases will usually only trigger if there are remote failures, and the system does not receive enough offsets. In this case, the failures are regarded as faulty nodes, thereby exceeding the number of tolerated faults, and the most we can do is a best-effort execution of the algorithm. The third case covers the standard execution of the algorithm. By dropping the $2k$ outer offsets, adversaries are forced to operate within a limited time offset range. By taking the average of the remaining offsets, adversaries would have to control more master nodes than our model tolerates to have a considerable effect on the aggregated offset. For a formal proof, we refer the interested reader to [1, 6].

3.5 Evaluation

To demonstrate the fault-tolerance of the proposed redundant PTP scheme and evaluate the synchronization quality, we generate two test-case network topologies [1]. These topologies are simulated witin the OMNeT++-4.6 [42] discrete-event network simulator using our extended version [2] of a PTP simulation library named LibPTP [116]. LibPTP [115] is a complete simulation framework for OMNeT++ that allows the simulation of standard PTP devices. To the RTC

[1] https://github.com/dtu-ese/ptp_multidomain
[2] https://github.com/dtu-ese/libPTP

oscillator noise and yield more realistic clock drift results, we utilize a Power-law noise library (LibPLN [114]) as described in the LibPTP documentation [116]. All experiments are done on a 64-bit i7-7700HQ CPU system running at 2.8 GHz with 32GB RAM.

3.5.1 Simulation parameters

The presented experiments are based on the following assumptions. Firstly, we assume that every node has multiple network interfaces, one for each domain, which is in line with the standard's recommendations, where it is advised that each domain operates over a separate network interface. Secondly, to optimize the simulation time and isolate the PTP evaluation, we assume that the network is used exclusively by PTP, so no other network traffic is simulated in the experiments. Empirically, we assume that every link has a bit-rate of 1 Gbps and is 1 meter long. Finally, every PTP stack uses the recommended gPTP profile for TSN [96] as shown in Table 3.1 and a peer-to-peer (P2P) delay mechanism.

Table 3.1: PTP port profile options. Values correspond to the interval of the respective messages in seconds and are represented as powers of two.

Parameter	Value
logAnnounceInterval	1
announceReceiptTimeout	3
logSyncInterval	-3
logMinDelayReqInterval	-3
logMinDelayReqInterval	-3

3.5.2 Test-case 1: Single PTP master on four redundant domains

This experiment aims to evaluate the stability of the proposed multi-domain aggregation scheme using the custom design of Figure 3.1 for both master and slave nodes. We generate a synthetic topology with three nodes and four switches as shown in Figure 3.3. A single multi-domain PTP master is connected to four redundant transparent clock nodes over four different domains. We integrate three different types of PTP slaves in the network: (A) a standard PTP

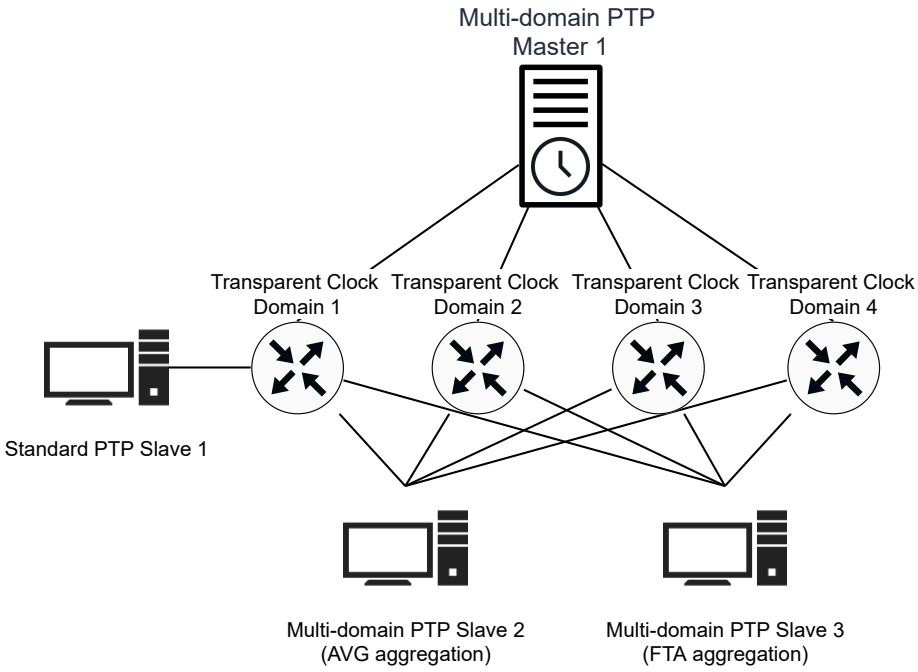


Figure 3.3: First test-case network topology of single multi-domain PTP master on four isolated redundant domain paths. The domains are isolated using four different switches.

slave connected only to the first transparent clock switch (domain), (B) a multi-domain PTP slave that uses the AVG algorithm and is connected to all domains and (C) a multi-domain PTP slave that uses the proposed FTA algorithm and connects to all domains.

We evaluate the synchronization quality in terms of average mean clock offset and standard deviation using a synthetic scenario. We simulate a simple scenario of consecutive link failures where at 60 seconds the first link between Master 1 and Transparent Clock 1 is disconnected. The rest of the links between Master 1 and the transparent clocks are disconnected/fail similarly in intervals of 30 seconds. We simulate the scenario for a total run-time of 180 seconds.

Figure 3.4 compares the measured mean clock offset and jitter of the two clock servo aggregation methods (AVG and FTA). Although the mean of the AVG and FTA aggregation methods are similar when no failures occur, we measure significantly less jitter using FTA throughout the experiment's run-time, resulting in more predictable clock synchronization. This is likely due to the nature of

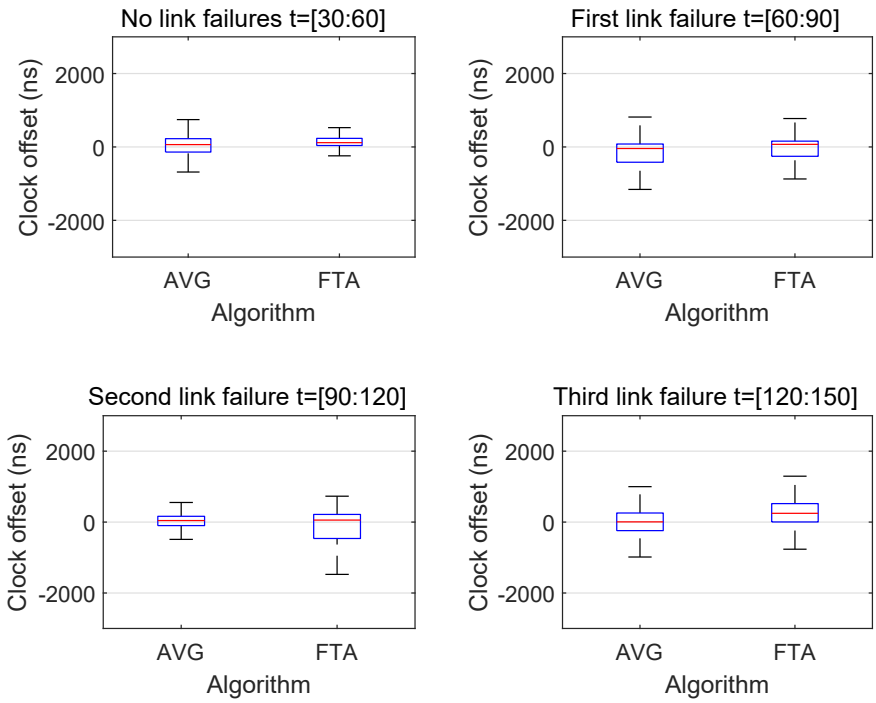


Figure 3.4: Comparison of the mean clock offset and std. deviation measurements through the link failures of the experimental test-case 1 (see Section 3.5.2) with topology from Figure 3.3.

the FTA: outliers are discarded , ensuring that the system will take the average of the most consistent master clocks. If there are some master clocks that drift at different rates, or if these clock oscillators are very noisy, then it is likely that they are often discarded for the aggregated timestamp.

3.5.3 Test-case 2: Four PTP masters on four redundant domains

For the second test-case, we generate and simulate two network topologies comparing the standard BMCA against the proposed multi-domain scheme. The first topology (see Figure 3.5a) has four PTP master capable standard nodes and a standard node that is configured as a PTP slave. All nodes operate over the same domain and are connected through a transparent clock switch in a star topology. The second topology (see Figure 3.5b) has four standard PTP masters

connected and two redundant PTP slave nodes. The PTP masters operate over four different domains and are respectively connected to four different transparent clock switches. For simplicity, we assume that individual PTP master node clocks are synchronized to each other in order for the observation window to use all available domains. This requirement is further discussed in Section 3.6. PTP slave nodes 1 and 2 use respectively, the multi-domain aggregation methods described in Section 3.4. We evaluate the performance of the synchronization by simulating two synthetic scenarios.

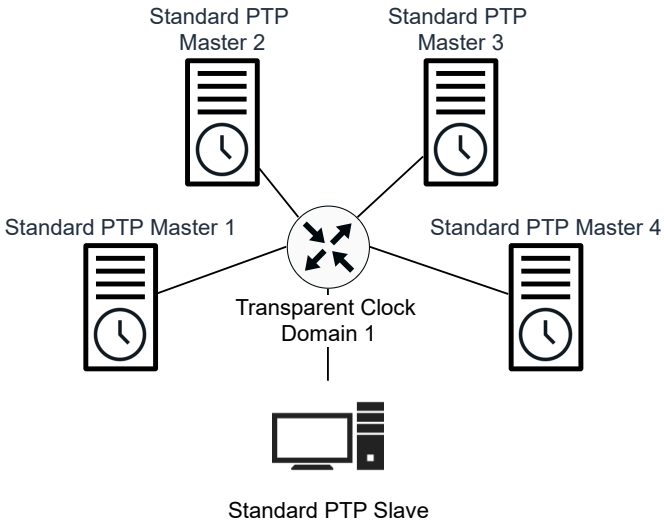
3.5.3.1 Link/node failure scenario

In the first scenario, each of the PTP masters fails in sequence every 30 seconds after the first minute of stable operation. This scenario covers a variety of real-life failures such as device failures, cable failures or denial-of-service attacks. We simulate the experiment for a total run-time of 180 seconds.

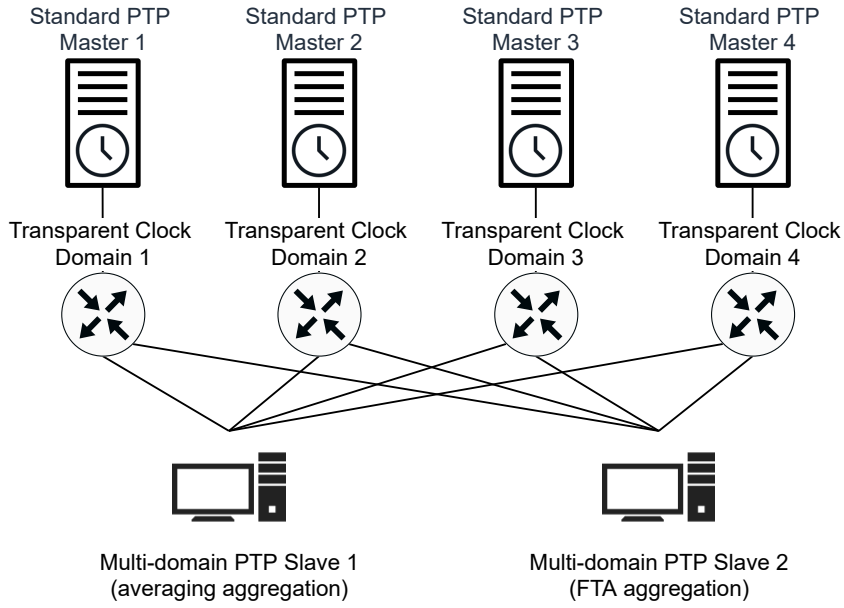
Figure 3.6 presents the mean time difference of the three PTP slave nodes and compares the upper/lower bounds of the three PTP slave nodes. We observe that in contrast to Test-case 1, the standard PTP slave node can stay synchronized to the master through the consecutive link failures as it can now select a new master, from each operating domain, after each link failure. However, BMCA suffers from significant synchronization drift of more than 2 μs. The FTA and the AVG aggregation manage to achieve better clock synchronization accuracy with tighter bounds than the standard BCMA during the first two link failures. As more links fail this difference between the methodologies is normalized because fewer nodes are available to aggregate.

3.5.3.2 Malicious PTP master scenario

In this scenario, we investigate the effects of a malicious PTP master clock that tries to offset the synchronized network time. The malicious end-system is connected to the network at a specific point in time and advertises that it has a higher clock quality that the existing master clocks. We emulate this scenario by simulating the instantaneous connection of a new PTP master with higher quality clock attributes after one minute of run-time at the first switch. The malicious master has its local clock offset by 100 μs than the existing masters. Due the implemented observation window's properties, a malicious master must be carefully implemented so that its local clock offset is within the observation window's bounds.



(a) Connect one standard PTP slave to four PTP masters operating on the same domain. Clock selection based on BMCA.



(b) Two multi-domain PTP slaves connected to four PTP masters operating on separate domains. Clock offset calculation uses multi-domain aggregation.

Figure 3.5: Second test-case parallel network topologies evaluation.

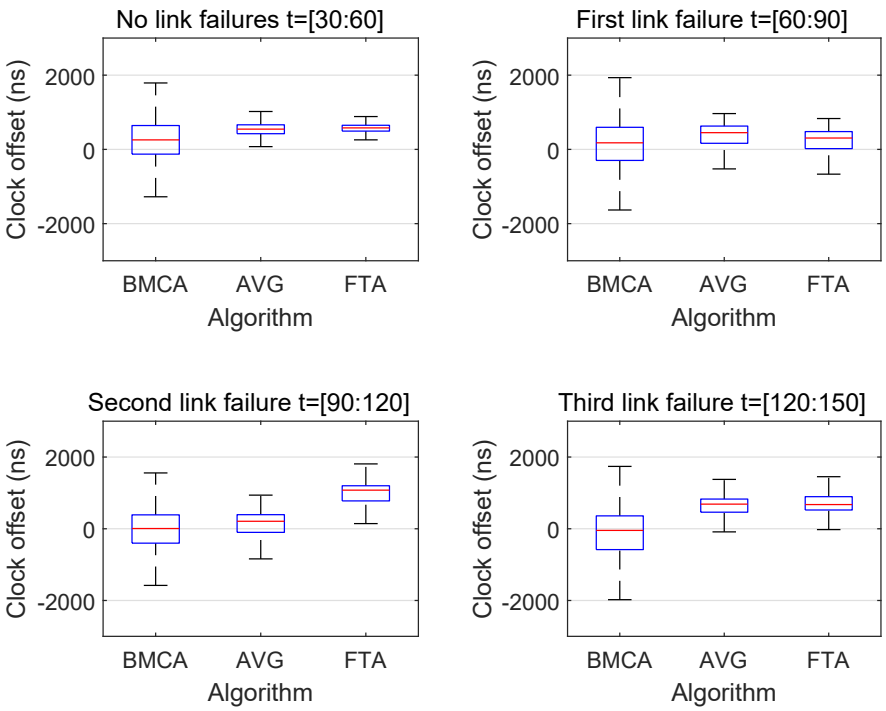


Figure 3.6: Comparison of the mean clock offset and std. deviation measurements through the link failures of the experimental test-case 2 (Section 3.5.3.1) with topology from Figure 3.5.

We measure this attack's effects on the clock synchronization precision of the topology's three PTP slaves relative to node Master 1. Figure 3.7 presents the measured results of the time-difference for the three PTP slaves. The top plot corresponds to the measurements taken from the Standard BMCA slave shown in Figure 3.5a. In comparison, the bottom plot presents the measurements from the multi-domain slaves shown in Figure 3.5b. We run the experiment for 120 seconds of simulation time.

In the standard PTP topology 3.5a, the newly connected malicious master is quickly elected as the best clock by the BMCA. We note a significant initial drift of the PTP slave relative to Master 1 after which the network is synchronized to the time of the malicious master clock. In the redundant PTP topology 3.5b, the connection of the malicious master cannot influence the independent masters as they operate in different domains. The simple approach of averaging the aggregated multi-domain master clocks is not sufficient as it is easily disturbed by the malicious clock's offset. In this scenario, the FTA proves to be the most resilient as the malicious master's relative clock offset is discarded according to Algorithm 2.

3.6 Discussion

In the evaluated test-cases, we experimentally showed that a multi-domain approach could guarantee synchronized network time availability despite network failures and malicious actions.

The platform designer has to guarantee that the PTP stack processes are isolated and cannot affect each other if the security of one PTP stack is compromised. This can be achieved using specialized hardware or sandboxing techniques such as virtualization. Considering the capabilities of modern industrial computing systems [137], the software cost for running the redundant PTP stacks in-parallel is minimal, especially if the proposed design is implemented completely in software. Preliminary results show that the CPU overhead generated by the PTP stack is less than 1% of the available computing resources. Nevertheless, the system designer should consider the additional cost for the redundant network topology based on the safety requirements of the application, as there is a significant cost increase in the number of links and switches.

The results showed that the FTA convergence algorithm could mitigate against link or node failures, as well as a compromised master node broadcasting incorrect timestamps. This work illustrates the importance of a fault-tolerant method of converging the calculated offset from the multiple PTP domains. It

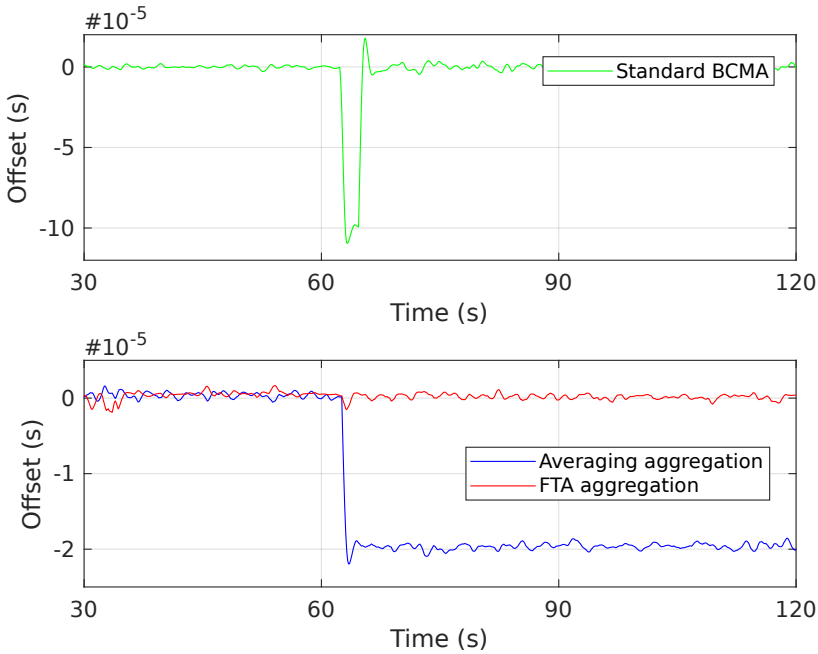


Figure 3.7: Measured PTP-Slave clock offset relative to Master 1 in the test-case scenario of a new malicious PTP master node connection at t=60s (Section 3.5.3.2).

is worth noting that although the averaging aggregation performed as well as the FTA method, it was easily influenced by a malicious node and failed to provide secure synchronization. While our design does not enforce authentication and integrity of PTP messages by itself, the FTA algorithm leaves very little room for tampered messages, as it discards everything outside of a margin known to have a majority of correct offsets. What this approach does inherently provide is protection against various forms of DoS, timing, and delay attacks where the number of affected links/nodes is less than k. As already noted in Section 3.4, this can be combined with the security measures proposed in IEEE-1588 (2019) to further harden the security by providing authenticity, confidentiality and integrity of messages. Thus, the combined application of the measures proposed in this work and the standardized security measures results in a secure PTP system that in addition to the standardized measures is difficult to disrupt with DoS and timing attacks.

Finally, although the proposed multi-domain PTP end-system scheme was tested with both master and slave roles, its functionality is based on the assumption that the redundant master clocks of each separate domains are synchronized to each other. This assumption is easily achievable using the proposed multi-domain PTP end-system design (see Figure 3.1), however standard PTP master clocks on separate devices require an external fault-tolerant mechanism of clock synchronization. One possible solution to this would be to use dual roles for master nodes, were on specific domains they would act as slaves to each other and other domains as masters in an interleaved scheme. It is hypothesized that the standard PTP boundary clock component can support this dual role functionality, but its implementation in a multi-domain network topology requires further investigation.

3.7 Future Work

As future work, we plan to explore the implementation and characterization of boundary clocks as a mechanism to enable standard PTP master clock synchronization in redundant domains. Additionally, we plan to extend the evaluated scenarios and investigate different types of attacks on PTP, such as frame spoofing. This will allow us to characterize further the proposed multi-domain design performance and identify its tuning parameters.

Moreover, one can think of a scenario where only a limited subset of all nodes are connected to multiple domains. This raises questions such as how many multi-domain nodes are necessary to meet a certain required timing accuracy? For this, we plan to explore the integration of the proposed design in boundary

clocks that are connected to multiple domains, each maintaining slave clocks connected to only one of these domains.

3.8 Conclusion

The presented work investigated the requirements for fault-tolerance in TSN clock synchronization and proposed a PTP end-system design that supports multi-domain aggregation. The proposed design implements isolated PTP stacks that use an FTA-based aggregation mechanism to correct the clock servo. This is combined with a time-based observation window for additional security. The multi-domain PTP end-system was evaluated and compared against standard PTP nodes in two scenarios with emulated link failures and possible malicious PTP masters. Overall, this work illustrated empirically the necessity for fault-tolerance in PTP and multi-domain aggregation design that manages to overcome network faults.

Acknowledgment

This is work was part of the Fog Computing for Robotics and Industrial Automation (FORA) European Training Network (ETN) funded by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 764785.

Chapter 4

A Time-predictable Open-Source TTEthernet End-System

By Eleftherios Kyriakakis, Maja Lund, Luca Pezzarossa, Jens Sparsø, and Martin Schoeberl [J1]

Abstract

Cyber-physical systems deployed in areas like automotive, avionics, or industrial control are often distributed systems. The operation of such systems requires coordinated execution of the individual tasks with bounded communication network latency to guarantee quality-of-control. Both the time for computing and communication needs to be bounded and statically analyzable.

To provide deterministic communication between end-systems, real-time networks can use a variety of industrial Ethernet standards typically based on time-division scheduling and enforced by real-time enabled network switches. For the computation, end-systems need time-predictable processors where the worst-case execution time of the application tasks can be analyzed statically.

This paper presents a time-predictable end-system with support for deterministic communication using the open-source processor Pat-

mos. The proposed architecture is deployed in a TTEthernet network, and the protocol software stack is implemented, and the worst-case execution time is statically analyzed. The developed end-system is evaluated in an experimental network setup composed of six TTEthernet nodes that exchange periodic frames over a TTEthernet switch.

4.1 Introduction

Advancements in the field of safety-critical systems for industrial control and avionics/automotive automation have brought a recent focus on distributed real-time system communication [98]. This trend is emphasized by the upcoming paradigm of Industry 4.0 and the recent efforts of the time-sensitive networking (TSN) group [104] to develop a set of deterministic Ethernet standards that meets the requirement for system's interoperability and real-time communications.

The correct execution of real-time applications depends both on the functional correctness of the result as well as the time it takes to produce the result. A typical example that illustrates this criticality of functional and temporal correctness is a vehicle collision avoidance system. In this situation, the processor must correctly detect a possible object, but it is equally important that communication time, from the camera to the processor, and processing time, of the image on the processor, are deterministically bounded to consider the system as correctly functioning.

To achieve deterministic communication with bounded latency, real-time systems often employ a time-triggered (TT) communication schemes such as the well-known time-triggered protocol [8], which is based on a cooperative schedule and a network-wide notion of time [34]. This approach can be implemented on Ethernet communications, such as TTEthenet [22] and TSN [133], to provide the guaranteed networking services (latency, bandwidth, and jitter) required by distributed real-time applications such as avionics, automotive, and industrial control systems.

In this work, we focus on safety-critical systems and thus investigate the TTEthernet protocol. TTEthernet uses a fault-tolerant synchronized communication cycle with strict guarantees for transmission latency [22]. In addition, the protocol provides support for rate-constrained traffic and best-effort traffic classes. TTEthernet has been standardized under the aerospace standard SAE AS6802 [59] and has been implemented as a communication bus replacement in both automotive and aerospace real-time applications [62, 90].

This paper presents a time-predictable TTEthernet end-system implemented on the open-source processor Patmos [145], allowing for static WCET analysis of the networking code. This work enables the Patmos processor to communicate through a deterministic network protocol, allowing the possibility for end-to-end bounded latency communication that includes the software stack. We do so by extending the existing Ethernet controller driver and implement a TTEthernet software stack. We test the performance of the controller driver by evaluating the achieved clock synchronization and correct exchange of TT frames and perform a static WCET analysis of the software stack.

The main contributions of this work are:

- A TTEthernet node that combines time-predictable execution of tasks with time-triggered communication through TTEthernet
- A WCET analyzable Ethernet software stack, which allows to statically guarantee that all deadlines and end-to-end timing requirements are met
- Performing a comparative analysis of the effects of number of integration cycles against the achieved clock synchronization
- Implementing a PI controller that improves the achieved clock synchronization precision

To the best of our knowledge, this is the first WCET analyzable TTEthernet node that combines time-predictable communication over Ethernet with time-predictable execution of tasks. The presented design is available in open source.[1] An initial version of this work has been presented in [153].

This paper is organized into six sections: Section 4.2 presents related work on TTEthernet. Section 4.3 provides a background on the TTEthernet internals. Section 4.4 describes the design and implementation of a time-predictable TTEthernet node. Section 4.5 evaluates our design with measurements and static WCET analysis performed on a system consisting of a switch and six nodes. Section 4.6 concludes the paper.

4.2 Related Work

Traditional real-time communication was based on bus protocols such as CAN and PROFIBUS that can send small prioritized frames with bounded latency.

[1] see https://github.com/t-crest/patmos

CAN was later extended with TT capabilities. This TTCAN bus restricts nodes to only transmit frames in specific time slots, thus increasing the determinism of the communication [14]. The main limitations of both CAN and TTCAN are a maximum bandwidth of 1 Mbit/s, and limited cable length, which depends on the bandwidth of the bus [9]. To overcome these obstacles, FlexRay was introduced to replace CAN discussed in [27].

As the demand for higher bandwidth increases, the industry has started looking towards Ethernet-based real-time protocols. Several different standards and protocols, such as EtherCAT, Ethernet Powerlink, and TTEthernet, were developed, some of which have been compared in [76].

Another emerging Ethernet protocol for real-time systems is TSN [133]. TSN emerges from the audio-video bridging (AVB) protocol with the addition of a time-scheduled mechanism for real-time communication through the use of a gate control list on the transmission paths. Worst-case analysis of TSN networks is presented in [148]. The schedulability of the gate control list has been investigated by various works such as [135, 154] that showed that rate-constrained traffic can co-exist with time-triggered but introduces small jitter. In [102], the authors achieve zero jitter determinism of TT frames by enforcing time-based isolation of the traffic flows but reducing the solution space for TSN networks. The timing synchronization mechanism of TSN is based on the well known IEEE 1588 Precise Time Protocol which has been characterized by[107] and experimentally verified to achieve sub-microsecond precision by various works such as [54, 136, 138].

Both protocols, TSN and TTEthernet, aim to provide support for TT communication. They have been directly compared in [147], and the two protocols focus on different real-time system requirements and provide different levels of criticality. TSN offers greater flexibility and bandwidth fairness over TTEthernet but is only suitable for soft-real time traffic due to its lack of fault-tolerant clock synchronization and low granularity scheduling mechanism [121].

Research on TTEthernet communication has been focused on the following perspectives: (a) timing analysis of the communication links [129, 119], (b) schedule synthesis for frame flows [65, 66, 132], and (c) investigating the clock synchronization [73, 29]. In contrast, this work investigates the implementation characteristics of a TTEthernet compatible end-system that supports WCET analysis of the software on the end-system and its integration within a TTEthernet network.

Latency and jitter through a single TTEthernet switch have been measured in [51] using off-the-shelf components combined with a proprietary TTEthernet Linux driver. A performance validation setup was presented for TTEthernet

networks, and the relation between end-to-end latency, jitter, and frame size was investigated. A comparison between a commercial off-the-shelf switch and a TTEthernet switch was presented as a function of link utilization and end-to-end latency. This emphasized the benefits of TTEthernet over standard Ethernet switches. The measured jitter for the system was dependent on frame size, and the authors observed a jitter of 10 μs for frames smaller than 128 bytes and 30 μs for larger frames. Furthermore, a model of analyzing the worst-case latency of TTEthernet in the existence of rate-constrained traffic load is presented in [119]. The model shows that it is possible to provide safe bounds for rate-constrained traffic, and it is evaluated over a simulated network topology of an Airbus A380 aircraft.

Scheduling of TTEthernet communication has been investigated in [65]. It proposes a scheduling approach for TT traffic that allows the calculation of the transmission and reception time instants by each connected real-time application. The synthesis for static scheduling for mixed-criticality systems has been investigated in [66]. The concept of schedule porosity was introduced, allowing un-synchronized (best-effort or rate-constrained) traffic to be mixed with time-triggered traffic without suffering from starvation. Moreover, in [83], the authors further optimize TTEthernet schedules for mixed-criticality applications by presenting a schedule that allocates more bandwidth to best-effort traffic while still preserving determinism of TT traffic.

Clock synchronization is an essential part of TTEthernet as it guarantees the synchronized communication of the network end-systems according to the global schedule. Depending on the clock synchronization accuracy requirements of an application, the minimum number of integration cycles per cycle period can be calculated [126]. In [118], the authors investigate in a simulated environment a least-squares algorithm that manages the compensation of the error. In both cases, accurate measurements of the achieved synchronization accuracy, i.e., standard deviation and avg/max/min values, are not discussed, and the methodology is implemented on a simulated environment of a TTEthernet clock. In our setup, we use the TTEthernet clock synchronization mechanism but improve the clock error by adding a PI controller.

In our review of related work, we identified that most papers focus on analyzing the communication components of TTEthernet. We found just a single paper that described the implementation and analysis of a TTEthernet end-system. A software-based TTEthernet end-system has previously been developed for AUTOSAR [87], which is a standardized software architecture for control units in cars. The implemented AUTOSAR system acts as a synchronization client and uses existing hardware capabilities of Ethernet controllers to timestamp incoming clock synchronization frames, and the authors observed a jitter of approximately 32 μs. Regarding the processing time of the protocol the authors

provide CPU utilization and memory overhead metrics. Precise end-to-end latency of the system is unclear due to a non-deterministic dispatch and receive function. In contrast to our work, the authors do not provide WCET analysis of these functions, and although they discuss the importance of these delays in the calculation of the end-to-end latency, they do not provide measurements or static timing analysis. We provide static WCET analysis of all software components of our TTEthernet stack.

To the best of our knowledge, our paper is the first to present a WCET analyzable TTEthernet end-system that combines time-predictable communication over Ethernet with time-predictable execution of tasks. The presented design is available in open source.

4.3 TTEthernet Background

4.3.1 Overview

The deterministic communication capabilities offered by TTEthernet are based on special switches that handle TT traffic according to a global schedule, as well as end-system equipped with TTEthernet capable controllers for transmission and clock synchronization. TTEthernet technology is proprietary. However, an initial version of the switch architecture is presented in [28]. The design of the first hardware TTEthernet controller is presented in [35].

TTEthernet supports best-effort (BE) traffic and two types of critical traffic (CT): rate-constrained [17, 129] and time-triggered (TT). TT traffic takes priority over rate-constrained traffic, which takes priority over best-effort traffic. This paper focuses on TT traffic.

CT is sent and received using the concept of a virtual link (VL). A VL is a relation between a sender and one or more receivers and is identified by a unique ID. Switches know the VL definitions, and nodes know on which VL they are allowed to send. CT is formatted as standard Ethernet frames, but it differs from best-effort traffic by having the destination MAC address field used for the CT marker and for the VL on which the frame belongs, as shown in Figure 4.1. Depending on the VL, switches can forward the CT frame to the right port.

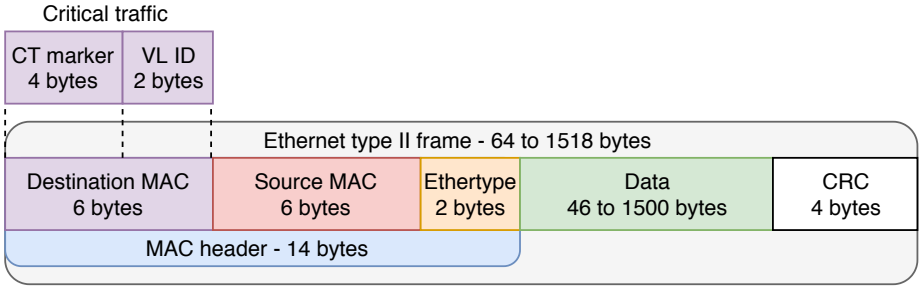


Figure 4.1: Ethernet type II frame. Critical traffic identifies a destination using a CT marker and VL ID instead of a MAC address.

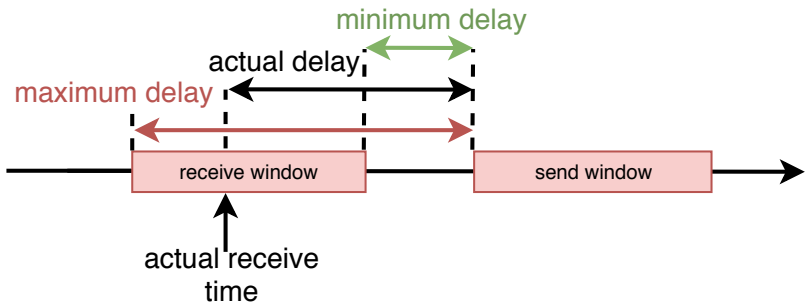


Figure 4.2: Switch delay in relation to the receive and send windows.

4.3.2 Time-Triggered Traffic

TT traffic is transmitted at pre-defined time slots. Thus, a VL definition includes a receive window where the switch accepts frames from the sender, and send windows where the switch passes frames to the receivers. The switch ignores all frames received outside of the defined window in order to guarantee bounded end-to-end latency and minimal jitter for other frames. The latency depends on the delay between the receive and the send windows in the switch. This latency is called switch delay. Figure 4.2 shows the possible minimum and maximum delays of an outgoing frame in relation to the receive and send windows.

The latency also depends on the transmission time (frame size over bandwidth) and the propagation delay (cable distance over propagation speed). For a system with short wires, the propagation delay is in the range of nanoseconds. The expected minimum and maximum latency for a VL in a given TTEthernet system can be calculated using Equations 4.1 and 4.2. 64 and 1518 are the

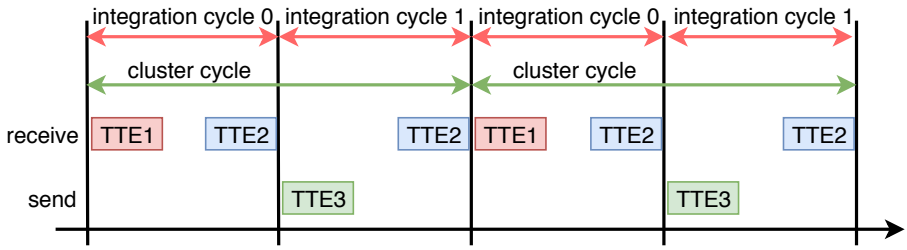


Figure 4.3: Example of integration and cluster cycles.

minimum and maximum possible Ethernet frame sizes, SD_{min} and SD_{max} are the minimum and maximum switch delays, d the network cable length, and s the propagation speed.

$$L_{min} = SD_{min}(s) + \frac{64\ bytes \cdot 8\frac{bit}{byte}}{bandwidth(\frac{bit}{s})} + \frac{d_{cable}(m)}{s_{cable}(\frac{m}{s})} \tag{4.1}$$

$$L_{max} = SD_{max}(s) + \frac{1518\ bytes \cdot 8\frac{bit}{byte}}{bandwidth(\frac{bit}{s})} + \frac{d_{cable}(m)}{s_{cable}(\frac{m}{s})} \tag{4.2}$$

4.3.3 Clock Synchronization

All nodes and switches need a global notion of time to send frames at the right moment in time. Clock synchronization is carried out periodically every integration cycle (typically in the range of 1 to 10 milliseconds). The schedule for TT traffic is also periodic and repeats every cluster cycle, which is an integer multiple of the integration cycle. Figure 4.3 shows an example of TT traffic in a system with an integration period of 10 ms and two integration cycles per cluster cycle. The schedule defines TT traffic by its period and the offset from the start of the cluster cycle. For example, TTE3 in Figure 4.3 has a period of 20 ms and an offset of 10 ms.

Clock synchronization is achieved through the exchange of protocol control frames (PCF). There are three types of PCFs in TTEthenet: integration frame, cold-start frame, and cold-start acknowledge frame. Integration frames are used in the periodic synchronization, while the last two types of PCF are used exclusively during start-up. PCFs are used for synchronization only when they become permanent. This happens at the point in time when the receiver knows

that all related frames that have been sent to it prior to the send time of this frame have arrived or will never arrive [52]. The permanence point in time ($Permanence_{PIT}$) is calculated by the TTEthernet protocol as the worst-case delay (D_{max}) minus the dynamic delay (D_{actual}) that a synchronization frame experiences plus the reception timestamp as shown in Equation 4.3. The dynamic delay (D_{actual}) is provided by the frames transparent clock value. This mechanism allows for a receiver to re-establish the send order of frames, and it used for remote clock reading during a synchronization operation. Assuming the transparent clock depicts the transmission time (D_{actual}) and based on the statically scheduled receive point in time ($ScheduledRX_{PIT}$) the clock difference ($ClockDiff$) is calculated as in Equation 4.4.

$$Permanence_{PIT} = RX_{PIT} + (D_{max} - D_{actual}) \tag{4.3}$$

$$ClockDiff = ScheduledRX_{PIT} - Permanence_{PIT} \tag{4.4}$$

Switches and nodes are involved in the exchange of PCFs for synchronization in three different roles: synchronization masters, synchronization clients, and compression masters. Typically, the switches act as compression masters, and the nodes are either synchronization masters or clients. Each node keeps track of when they believe the integration cycle has started, which is when synchronization masters send out integration frames. Compression masters use the permanence times of these frames to decide on the correct clock and send integration frames to all synchronization masters and clients. A returning integration frame is expected to be permanent $2 \cdot max\_delay + comp\_delay$ after the beginning of the integration cycle, where *comp_delay* is the time it takes a compression master to evaluate the frames. An acceptance window around this expected permanence point defines whether or not the node should accept the PCF as correct. The acceptance window has a width of twice the expected precision of the system, defined as the maximum difference between two correct local clocks. If the PCF is accepted, the difference between the expected and actual permanence time is used to correct the clock. Correction is typically delayed until it is sure that the corrected clock will not fall back within the acceptance window, as shown in Figure 4.4. If more than one compression master is present, the synchronization masters and clients receive multiple PCF in the acceptance window. In this case, the clock correction uses a fault-tolerant average of the differences between expected and actual permanence time.

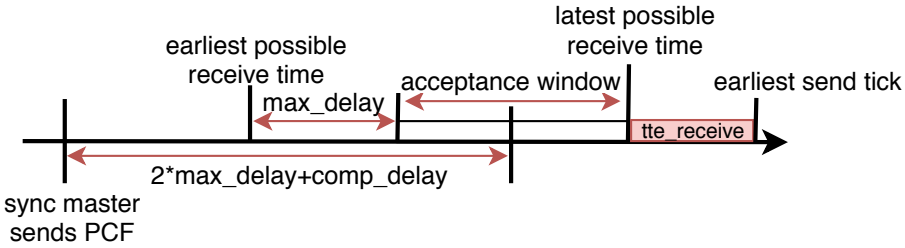


Figure 4.4: Overview of clock synchronization, adapted from [146].

4.4 Design and Implementation of the TTEthernet Node

In this section, we present the design and implementation of our TTEthernet node. First, we describe the hardware platform. Then, we explain the functionality of the developed software stack. Finally, we present the theoretical limits of the implementation.

We provide a time-predictable end node for a TTEthernet system, including hardware design and WCET analysis of the network software. We focus on time-predictable program execution and traffic transmission. Generating the static schedule for the time-triggered traffic and allocation of TT frames is out of the scope of this paper. We rely on available solutions, e.g., the scheduling tool that is part of the TTEthernet toolset.

4.4.1 Hardware

The proposed TTEthernet node is based on the Patmos [145], a time-predictable processor used in the T-CREST platform [94, 144], and on an open-source Ethernet controller. The controller is based on the EthMac block from OpenCores [15], which was previously ported for Patmos [91].

Figure 4.5 shows the hardware architecture of the node. The Patmos processor, as well as the RX/TX buffer, uses a variant of the OCP interface, while the EthMac block uses the Wishbone interface. A dedicated multiplexing bridge component manages the conversion between the two protocols. It allows Patmos to access the configuration registers in the EthMac controller and the RX/TX buffer as memory-mapped IO devices. The EthMac controller connects to the PHY chip through the media-independent interface (MII).

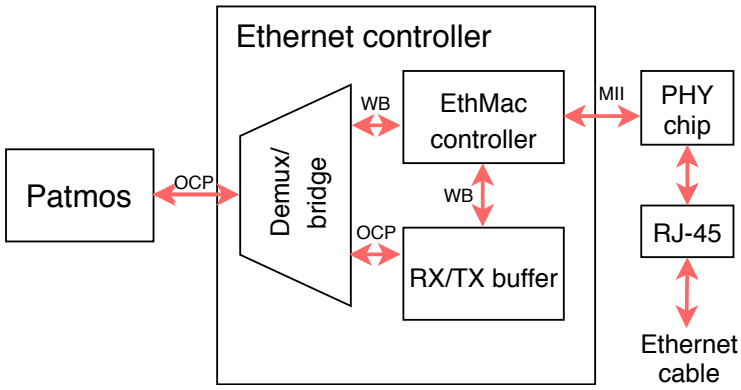


Figure 4.5: Overview of the Patmos Ethernet controller, adapted from [91]. It is connected to Patmos through OCP signals, and to a physical PHY chip through MII.

Receiving and transmitting frames from the EthMac block is based on buffer descriptors. These are data structures stored in the EthMac controller and containing the address to an associated buffer in the RX/TX buffer component, as well as the length and status of the buffer. The EthMac controller can receive a new frame only when there is at least one available receive buffer. Otherwise, the EthMac controller discards the frame. After receiving a frame, the controller writes the receive status into the associated buffer descriptor, and the controller may generate an interrupt (if enabled). The buffer will stay unavailable until the driver software marks the buffer descriptor empty again.

To send and receive TT traffic, no changes are required to the hardware architecture of the existing Ethernet controller [91]. However, the EthMac core was configured in promiscuous mode to avoid any filtering of frames on MAC addresses. Additionally, it was configured as full-duplex to avoid sending and receiving blocking each other. The functionality of the proposed node entirely lies in software, in the C library `tte.c`.

We implemented two different versions of the proposed solution: (1) where the program discovers received frames through polling, and (2) where the Ethernet controller triggers an interrupt whenever a frame is received. Using interrupts for time stamping of an arriving Ethernet frame is not the best solution since the start of the execution of the interrupt routine introduces jitter due to cache hits and misses. This receive jitter is critical in our implementation as it degrades the timestamp precision and results in lower clock synchronization quality. The jitter was measured at -26 μs, with the resulting clock precision varying be-

tween -10 μs and 16 μs. Further results regarding the evaluation of the clock synchronization are discussed in Section 4.5.2.

The polling solution solves this problem by using a periodic task that is scheduled to be released just before the next synchronization frame arrives. The release time needs to include enough time to contain the worst-case preemption delay and the possible jitter of the PCF itself. In this case, the processor is ready to listen to the Ethernet port in a tight loop in order to get a better timestamp in software. Therefore, in the polling solution, the actual polling runs only for a short time, without wasting processor time. As future work, we plan to change the Ethernet controller to include hardware support for time-stamping [138].

4.4.2 Software

Our node only acts as a synchronization client and only connects to a single switch. The developed software stack offers three main functionalities: initialization, receiving, and sending.

Figure 4.6 shows the intended flow of programs using the developed system. At first, the program initializes the controller with static information regarding the system, information on VLs, and the schedule. After initialization, the periodic task starts. It contains a call to the application code, which the programmer needs to organize as a cyclic executive, and then polling the Ethernet controller when a new frame is expected to arrive.

It is necessary to ensure that the receive time of integration frames is recorded as precisely as possible to enable correct clock synchronization. The received frame is then passed through the `tte_receive` function, which will synchronize the local clock in case of an integration frame, or otherwise return a value indicating the frame type. The rest of the body depends on the purpose of the program itself.

Outgoing TT frames can be scheduled anywhere in the program body and will be sent according to the system schedule through timer interrupts. To avoid fluctuations in the clock synchronization, the system schedule, and the WCET of the program body should follow the limits described in Section 4.4.3.

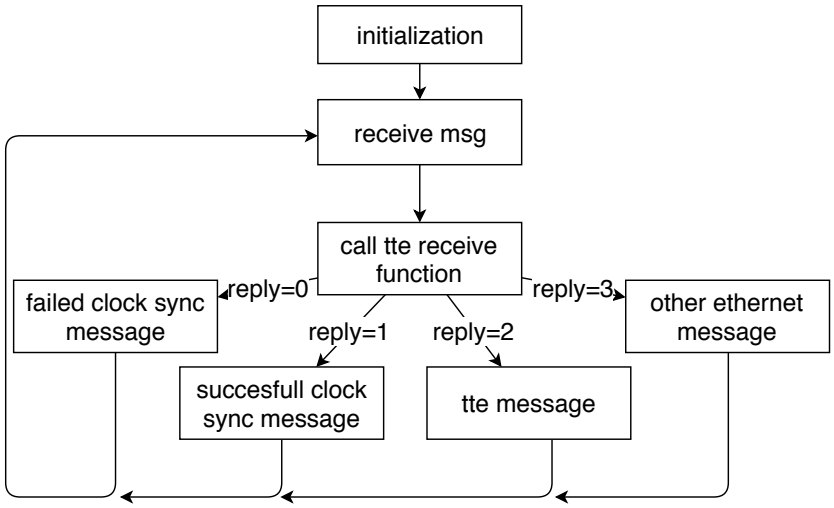


Figure 4.6: The intended flow of user programs. The controller is first initialized with all constants of the system. The regular operation is performed by a loop where the program continually waits until a frame is received, calls the `tte_receive` function, and then reacts to the reply.

4.4.2.1 Initialization

The system needs to be initialized before a program loop can start. The initialization data includes: the integration cycle, the cluster cycle, how many VLs the node can send on, the maximum transmission delay, the compression delay of the switch, and the system precision. Furthermore, the permitted send time for each VL needs to be known, so each VL gets initialized with an ID, an offset, and a period. The TTEthernet switch ensures that only relevant and correctly timed TT frames are passed on to the network. As soon as the TTEthernet receive function receives the first PCF, it starts the timer for the sending function.

During initialization, RX buffers are also set up (by configuring buffer descriptors). Multiple buffers are needed to ensure that frames are not lost while the latest frame is still in use. The precise number of buffers depends on the system and schedule.

4.4.2.2 Receiving and Clock Synchronization

Frame reception is performed periodically based on the scheduled receive point in time. At each reception cycle, a function starts continuously polling the interrupt source register for a specified timeout duration, until the bit signifying that a frame has been received is set. This is done with a function called `tte_wait_for_frame`, which is also responsible for recording the receive time by reading the current cycle count. After a frame has been received and the receive time has been stored, we mark the buffer as empty and clear the interrupt source (implemented in the function `tte_clear_free_rx_buffer`). Afterwards, the `tte_receive` function (described below) is called.

The `tte_receive` function initially checks the type of the frame. If it is a PCF type, the integration frame is used to synchronize the local clock to the master clock. If the received frame is not a PCF type, the function returns the received frame to the application.

For clock synchronization, the permanence point is calculated by adding the maximum delay and subtracting the transparent clock. For keeping track of when the controller expects synchronization masters to have sent the PCF, a variable called `start_time` is used. On receiving the very first PCF, this is set to $permanence\_time - (2 \cdot max\_delay + comp\_delay)$. `start_time` is used to calculate the scheduled receive point, which is used to calculate the acceptance window. If the permanence point is outside the acceptance window, the `start_time` is reset to zero, and the function returns zero. In this way, the user program can immediately see that an error has occurred, and the controller returns to regular operation when it receives a correctly timed PCF once again.

If the permanence point is within the acceptance window, the difference between permanence point and scheduled receive point is added to the `start_time`, synchronizing the local clock to the master clock. The controller does not need to wait until after the acceptance window to synchronize, because the implementation only assumes one switch in the network, and thus only one PCF per integration cycle. Therefore, it is irrelevant whether or not the local clock goes back within the acceptance window.

4.4.2.3 Sending

Frames must be sent according to a predefined schedule, which requires some queuing mechanism, as the program should not be expected to calculate the exact send times itself. The `ethlib` send function expects outgoing frames to

be stored in the RX/TX buffer and requires the specific address and size of the frame. One send queue is created per VL during initialization and is allowed to hold the maximum amount of frames that the VL can send in one cluster cycle, calculated as $\frac{clustercycle}{VLperiod}$. The send queues hold addresses and sizes of frames scheduled for sending. Each queue operates in a FIFO manner, keeping track of the head and tail through two variables.

The programmer has the responsibility to create frames in the RX/TX buffer according to its intended use by mean of the function `tte_prepare_header` to create the header of a TTEthernet frame. Frames are scheduled through the function `tte_schedule_send`, which takes the address and size of the frame and which VL it should be scheduled. The function then checks the queue of the VL and, if not full, schedules the frame for sending. The programmer shall not overwrite the buffer before the software stack sends the frame.

4.4.2.4 Generating the send schedule

An end-system should know the TTEthernet schedule running on the switch. By knowing the maximum frame size of a VL, the period, and the offset, it is the possible send times for each VL can be computed by repeatedly adding the period to the offset. A small algorithm is then used to combine these into a single schedule, generated on the fly at the startup time of the end-system. This is explained through the example presented in Figure 4.7. To simplify the scheduling of the next timer interrupt, each entry in the schedule represents the time until the next interrupt. We represent time in tenths of ms.

The first event for each VL is at its offset; thus, the very first event in the schedule can be found by finding the smallest offset. The starting offset is stored in a global variable `startTick` and the VL it belongs to in the first place in the schedule. For all VLs, a temporary variable named `current` is set to the offset of the VL. The VL that had the starting offset adds its period to this value, which is the $VL.current$ value when i is 0 in Figure 4.7. We calculate the schedule time when $i = 0$ as the difference between the minimum current value (here 26) and the last current value (here 10). We store the VL with the smallest current value in the next place in the schedule (when $i = 1$), and we increment its current value by its period. We repeat these steps until the smallest current value is larger than the cluster-cycle (8 ms in this example).

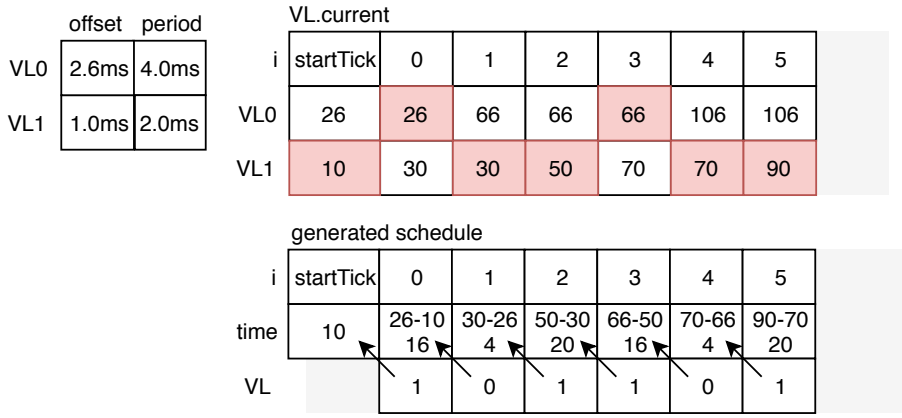


Figure 4.7: Example of schedule generation. The two VLs with the offset and period shown in the top left result in the schedule in the bottom right when the cluster cycle is 8 ms. The top right shows the next send time of each VL at different steps in the algorithm.

4.4.3 Theoretical Limits of the Implementation

Because of the single-threaded nature of the implementation, the controller is characterized by certain limits, which we describe in the following three subsections.

4.4.3.1 Earliest outgoing TT frame

At the start of every cluster cycle, the transmission of frames is scheduled by the PCF handle function right after the clock has been corrected. Since the start of the cycle is defined as the point were the PCF frame is sent by the synchronization masters, scheduling a VL to send at 0 ms would cause the function to schedule a timer-tick in the past.

We do not know the exact receive times of PCFs at compile-time, but we can assume that a PCF is permanent within the acceptance window, the latest possible receive time would be the same as the latest possible permanence time. Since the acceptance window is twice as wide as the precision, the latest receive time can be calculated with Equation 4.5.

$$rec_{latest} = 2 \cdot max\_delay + comp\_delay + precision \tag{4.5}$$

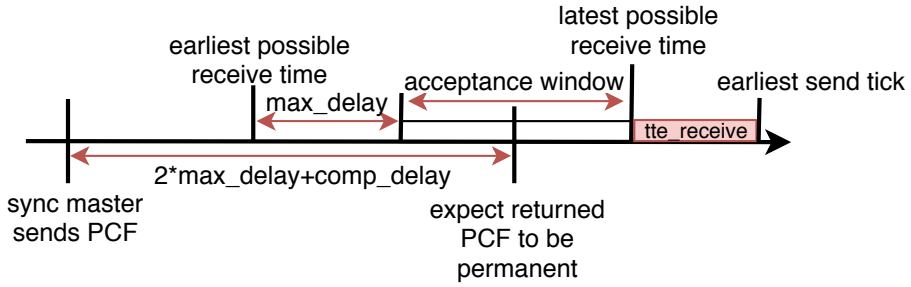


Figure 4.8: The earliest possible receive time is max_delay before the start of the acceptance window. The latest possible receive time is at the end of the acceptance window. The first TT frame should be scheduled no earlier than the WCET of `tte_receive` after the latest possible receive time.

To ensure that the first outgoing TT frame is never scheduled too soon, it should be scheduled no earlier than the latest possible receive time plus the WCET of the `tte_receive` function. Figure 4.8 illustrates this timing relationship.

4.4.3.2 Maximum execution time after a TT frame

If the program has a long execution time, the reception of PCF might be delayed, negatively impacting the clock synchronization. Part of this could arise from the code executed after receiving a TT frame. The maximum allowed execution time after a TT frame depends on the TT frame scheduled with the smallest gap to the next integration frame. In our switch implementation, the *send_window* defined in the switch dictates the latest possible receive time in the node.

The earliest possible receive time of a PCF (assuming it is on schedule) would be if the actual transmission time were 0, and the frame was permanent as early as possible in the acceptance window. This is equivalent to *max_delay* before the acceptance window, as seen in Figure 4.8. All in all, the maximum execution time of the code executed on receiving a TT frame can be calculated with Equation 4.6, as illustrated in Figure 4.9.

$$\begin{aligned} max_{tt} = & start\_time - ttrec_{latest} + max\_delay \\ & + comp\_delay - precision \end{aligned} \tag{4.6}$$

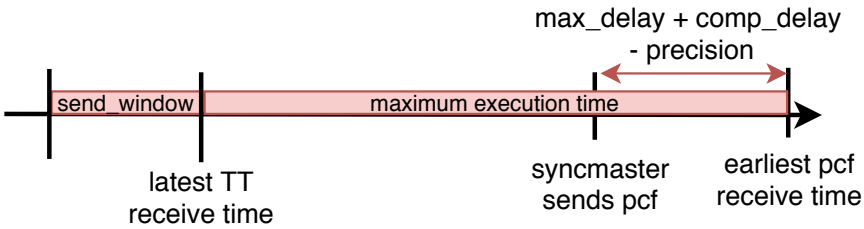


Figure 4.9: The code executed after receiving a TT frame should take no longer than from the latest TT receive time until the earliest receive time of the next PCF.

4.4.3.3 Maximum execution time during the integration cycle

Even if code executed upon receiving TT frames follow the limits described above, the reception of a PCF could still be delayed if the combined execution times of everything executed during an integration cycle exceed the integration period. This limit can be expressed with Equation 4.7, where inc_{tt} is the number of received TT frames.

$$period > WCET_{int} + inc_{tt} \cdot WCET_{tt} + send\_ticks \cdot WCET_{send} \tag{4.7}$$

4.4.4 Source Access

The TTEthernet controller and the relevant software are in open source and are available at https://github.com/t-crest/patmos. The software can be found at https://github.com/t-crest/patmos/tree/master/c/apps/tte-node.

4.5 Evaluation

4.5.1 System Setup

For implementation and testing, we used the star network configuration shown in Figure 4.10. It consists of a TTEthernet Chronos switch from TTTech Inc., four Linux end nodes (AS, AC, VS, and VC), a Microsoft Windows node used for configuration and monitoring, and our TTEthernet node. Three of the Linux end nodes (AS, AC, and VC) act as synchronization masters. The fourth Linux node (VS), as well as our TTEthernet node, act as synchronization clients.

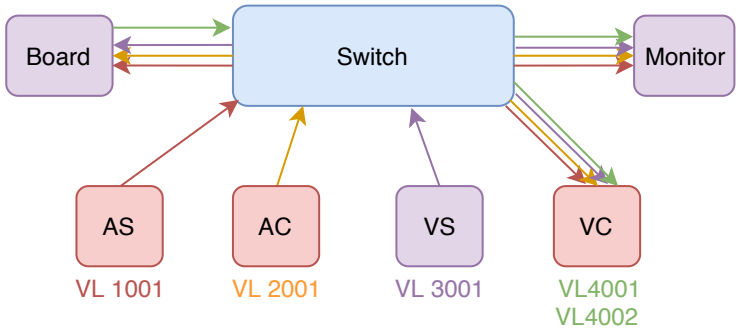


Figure 4.10: Illustration of components and VLs in the system. Red nodes are synchronization masters; purple nodes are synchronization clients. This is also the physical setup used for tests presented in Section 4.5.

The TTTech Chronos switch has 24 ports: six supporting Gigabit Ethernet and 18 supporting fast Ethernet (10/100 Mbit/s). We use 100 Mbits/s. The four Linix nodes are Dell precision T1700 PCs running Ubuntu. They are all equipped with a TTEthernet PCIe network card. The network card has two small form-factor pluggable ports that support connections with 100/1000 Mbit/s. The PCs execute a TTEthernet driver and API, as well as various test programs. The Windows PC does not contain any TTEthernet specific hardware. It runs the TTEthernet tools from TTTech Inc. that is used for configuring the TTEthernet system. The Windows PC is used to monitor the traffic on all VLs using Wireshark. The final node is our TTEthernet node, which is implemented on an Altera DE2-115 FPGA board using two Ethernet controllers: one for sending and one for receiving. All implemented hardware and software is open-source.

The TTEthernet tools are a TTTech development suite for configuring TTEthernet systems [166]. The tools include a GUI editor based on Eclipse [166]. Figure 4.11 shows the typical process for generating configuration files for all devices in a TTEthernet system. The first step is to create a network description file. It contains information about: senders, receivers, physical links, virtual links, and synchronization domain. The TTE-Plan tool uses this file to generate a network configuration file, including the schedule for the entire network. For a specific schedule, we can also manually edit the network configuration file. From the network description and the network configuration files the tool TTE-Build generates the device-specific binary images that are loaded into the switch and the end nodes at initialization time.

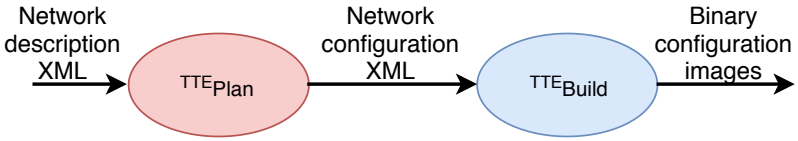


Figure 4.11: Typical process using TTEthernet tools.

We have experimented with different schedules, all implementing the same set of VLs, as shown in Figure 4.10. We tested the network at different integration periods and cluster cycles, but most experiments have used an integration period of 10 ms and a cluster cycle of 20 ms. The maximum possible transmission delay of any frame in the system was calculated with the TTEthernet tools and configured as 135600 ns. The compression delay of 10500 ns was used for collecting the results in Section 4.5. We calculated the precision with the provided TTEthernet tools, and configured it as 10375 ns.

4.5.2 Clock Synchronization

The clock error is calculated as the difference between the scheduled receive point in time, and the actual permanence point in time at each received integration frame as specified by the TTEthernet standard SAE AS6802 [59]. The clock error was measured in two different setups using different integration periods (synchronization cycles), a 10 ms period, and a 100 ms period. Figure 4.12 presents a comparison of the measured clock error for the two integration periods. For an integration period of 100 ms, the clock error ranges between 2875 ns and 3225 ns with a mean of 3055 ns, while for an integration period of 10 ms, the clock error ranges between 2812 ns and 3112 ns with a mean of 2948 ns.

To reduce the systematic error, we implemented a proportional/integral (PI) controller. The controller was manually tuned by first increasing the proportional part until there was a steady oscillation and then increasing the integral part until the systematic error was removed, this procedure led to the coefficient values $K_i = 0.3$ and $K_p = 0.7$. The results of the PI controller implementation are presented in Figure 4.13 and compared between the two different integration periods. When the control loop stabilizes, the clock error is just a few clock cycles with a mean of 126 ns for the integration period of 100 ms and a mean of 16.23 ns for the integration period of 10 ms. This is the best that can be achieved by taking timestamps in software in a tight loop. Similar methods for compensating the clock error have been investigated in [43]. The authors presented in simulation the use of a Least Squares Algorithm that managed to achieve 2000 ns offset. By applying a simple PI controller, not only we re-

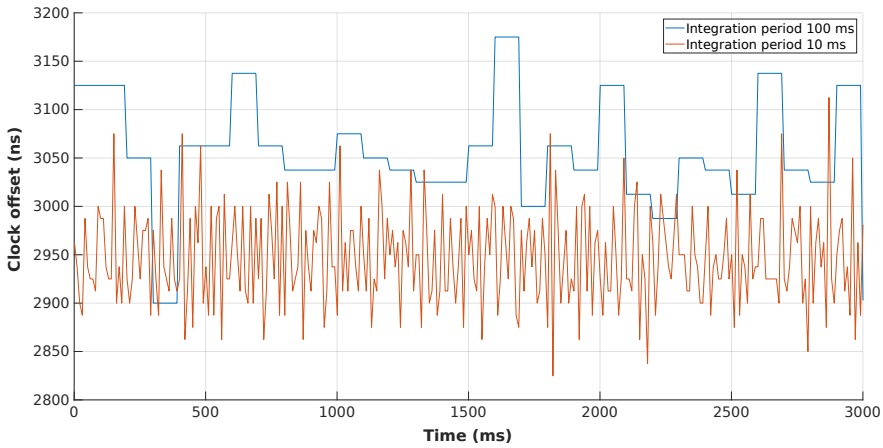


Figure 4.12: Clock error comparison between two different integration periods.

duce the complexity, but we also measured a significant increase in the clock synchronization accuracy.

4.5.3 Latency and Jitter

To precisely measure latency and jitter of TT frames from the implemented controller, we used a physical setup similar to the one described in [51] and shown in Figure 4.10.

The test uses the two available Ethernet ports on the Altera board and sets up a schedule with a VL from one port to the other. Both ports are considered to be their own device by the schedule and are both synchronization clients. The second controller is accessed like the first, but uses a different address in local memory. This was not supported by the original ethlib, and was accommodated by duplicating several ethlib IO functions. A single VL with a period of 10 ms and an offset of 8.2 ms was used, which simplifies the test program since one frame can be sent and received each integration cycle.

The test program follows the form described in Figure 4.6, with the first controller receiving frames in the overall loop. After a successful synchronization frame, a TT frame is scheduled, and the program waits until the second controller receives the frame before proceeding. This enables the program to collect both the schedule and receive points of frames as the current clock cycle count. A slightly modified version of the sending interrupt function was used to mea-

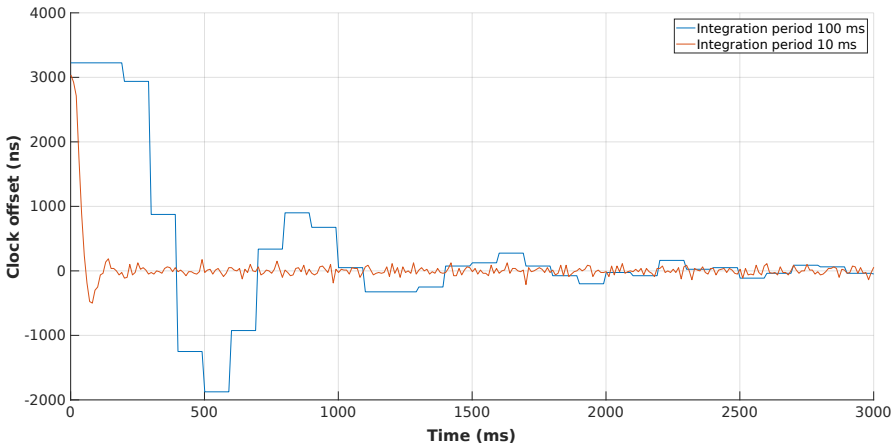


Figure 4.13: Clock error with two different integration periods using a PI controller.

sure the send point, making it possible to calculate latency and jitter. Both receive and send window in the switch where 263 μs wide.

Figure 4.14 shows latency measured as the average difference in send and receive time over 2000 measurements for 3 different frame sizes with various minimum switch delays. Lmin and Lmax have been calculated using Equations 4.1 and 4.2, disregarding propagation delay, and plotted alongside the values. All measured values are inside the expected minimum and maximum values.

The expected transmission times for frames of the measured sizes are 5.12 μs, 32 μs, and 121.12 μs respectively, which means that the actual switch delay for these experiments must be approximately 200 μs higher than the minimum, judging by the trend-lines. This indicates that the switch receives the frames approximately 63 μs into the receive window in these tests. The jitter, measured as the smallest latency subtracted from the highest, did not vary significantly as a function of switch delay but stayed between 4.5 us and 4.6 us throughout all experiments.

4.5.4 Worst-Case Execution Time

To enable the WCET analysis of the software, all loops need to be bounded. Therefore, we needed to perform some small modifications to our code. The function which converts the transparent clock from the format used in PCF to

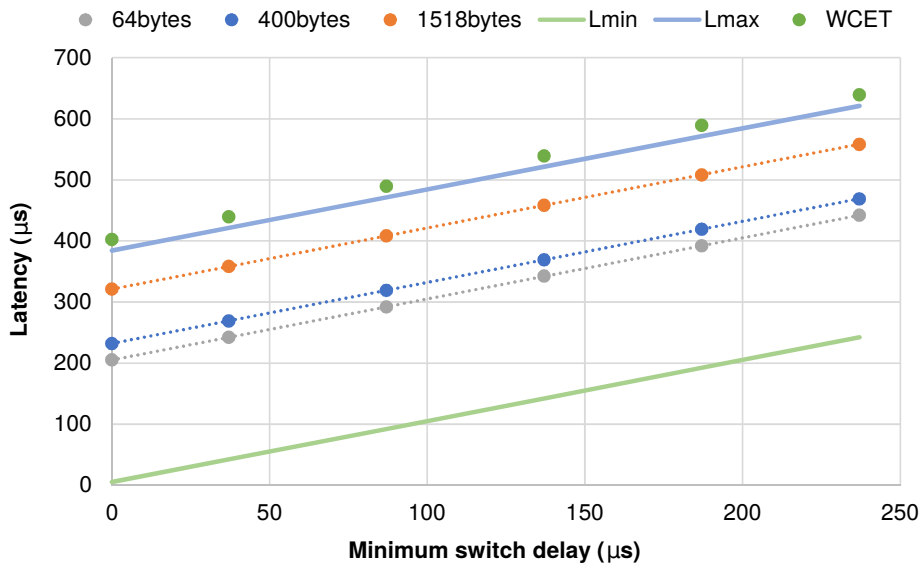


Figure 4.14: Latency for various frame sizes as a function of minimum switch delay. The correlation is solid, and well within the expected minimum and maximum values.

clock cycles initially performed division on an unsigned long. According to the analysis tool, the division function contains an unbounded loop. We replaced the division by an almost equivalent series of multiplication and bit-shifting to make the function analyzable. Additionally, we added pragmas containing loop bounds to all loops in the code, to aid the analysis tool.

We performed a WCET analysis on significant parts of the controller using the platin WCET tool [89]. For the analysis, the board configuration was assumed to be the default for DE2-115.

We run the WCET analysis on a program resembling the demo program used with the regular implementation in Section 4.5.2. To verify that the program and schedule satisfy the limits presented in Section 4.4.3, parts of the program have been moved into separate functions. Additionally, to analyze the timer interrupt function, it had to be explicitly called.

The tool requires that analyzed functions have the attribute `noinline` to prevent inlining. This makes the analysis slightly more pessimistic than necessary. The results can be seen in Table 4.1, where all functions are part of the implemented

Table 4.1: Worst case execution time of TTEthernet software stack functions.

Function	WCET (in clock cycles)
tte_clear_free_rx_buffer	10
tte_receive	3018 (3424)
tte_receive_log	3154 (3561)
handle_integration_frame	1454 (1860)
hande_integration_frame_log	1590 (1997)
tte_prepare_test_data	63149
tte_schedule_send	244
tte_send_data	306
tte_clock_tick	1721
tte_code_int	392419
tte_code_tt	40156

`tte.c` library, except for the final two, which are part of the tested demo program. The parentheses indicate WCET with PI implementation.

`tte_clear_free_rx_buffer` and `tte_receive` are mentioned in Section 4.4.2.2. `tte_receive_log` is the TTEthernet receive function with logging enabled. `handle_integration_frame` and `handle_integration_frame_log` are called by the `tte_receive` function if the frame is an integration frame. `tte_prepare_test_data` creates a TT frame where the data part repeats a specified byte until the frame has a certain length. `tte_schedule_send` is described in section 4.4.2.3. `tte_clock_tick` and `tte_clock_tick_log` are the timer interrupt functions with and without logging, and call `tte_send_data` when sending a frame. `tte_code_int` is executed after each successfully received integration frame, and `tte_code_tt` is executed after each received TT frame. The addition of logging adds about 150 clock cycles to the WCET. It is worth noting that the PI implementation adds 400 clock cycles while using fixed-point calculations.

4.5.5 Verifying Theoretical Limits of the Demo Program

With this example program and schedule, it is possible to verify that it satisfies the theoretical limits. The earliest outgoing TT frame in this example has an offset of 0.8 ms. Equation 4.8 presents the calculation of the earliest allowed transmission of a TT frame. It accounts for the equations presented in Section 4.4.3.1, the system constants, and the WCET of the `tte_receive` function (the log version) as $WCET_{tte\_rx}$. Since 332.3 μs is approximately 0.33 ms, the

example follows this limit.

$$\begin{aligned} tt_{out} &= 2 \cdot max\_delay + comp\_delay + precision + WCET_{tte\_rx} \\ &= 2 \cdot 135.6\mu s + 10.5\mu s + 10.4\mu s + 3216 cycles \cdot \frac{12.5 \frac{ns}{cycle}}{1000 \frac{ns}{\mu s}} \\ &= 332.3\mu s \end{aligned} \tag{4.8}$$

The TT frame, which arrives closest to a PCF in this example, arrives between 18.6 ms and 18.863 ms. Using this information, Equation 4.6 and the system constants, the maximum allowed execution time after TT frames is calculated with Equation 4.9. The WCET of `tte_code_tt` in this example can be seen in Table 4.1. Since it is less than the 101,816 calculated cycles, the example program follows this limit.

$$\begin{aligned} max_{tt} &= sched\_send - ttrec_{latest} + max\_delay \\ &\quad + comp\_delay - precision \\ &= 20,000\mu s - 18,863\mu s + 135.6\mu s + 10.5\mu s - 10.4\mu s \\ &= 1272.7\mu s \end{aligned} \tag{4.9}$$

$$\begin{aligned} max_{cc} &= 1272.7\mu s \cdot \frac{1000 \frac{ns}{\mu s}}{12.5 \frac{ns}{cycle}} \\ &= 101,816 cycles \end{aligned} \tag{4.10}$$

The example schedule has a maximum of 3 incoming TT frames in a single integration cycle. One of the VL can send a maximum of 3 outgoing TT frames, and the other a maximum of 5. An integration period of 10 ms is assumed, which is equivalent to 800,000 clock cycles. This information, Equation 4.7 and the WCET in Table 4.1 are used to verifying the final limit in Equation 4.11 (in clock cycles (cc)).

$$\begin{aligned} int\_period &> WCET_{int} + inc_{tt} \cdot WCET_{tt} + send\_ticks \cdot WCET_{send} \\ 800,000cc &> 392,419cc + 3 \cdot 40,156cc + 8 \cdot 1824cc \\ 800,000cc &> 527,479cc \end{aligned} \tag{4.11}$$

4.5.6 Future Work

The presented time-predictable TTEthernet controller is a good basis for future work. We plan to re-implement the whole TCP/IP stack in a time-predictable

version. We will avoid the blocking calls to read and write, as the usual implementation of sockets. We will use non-blocking functions that can be called from periodic tasks.

Furthermore, we are working on coordinating the scheduling of tasks with the scheduling of TTEthernet frames. With a tight coupling of time-triggered execution and time-triggered communication, the end-to-end latency can be reduced.

Furthermore, we plan to add support of TSN to our node. Then we can directly compare TTEthernet with TSN.

4.6 Conclusion

This paper presented a time-predictable TTEthernet end-system, built on top of the time-predictable Patmos processor. To the best of our knowledge, this solution is the first TTEthernet end-system that can be analyzed for the worst-case execution time.

We evaluated the TTEthernet node in a test environment with one TTEthernet switch and six TTEthernet nodes that exchanged frames in various periods. The presented end-system can synchronize to the network clock with nanosecond precision by using a PI controller that significantly improved the synchronization error measured in previous work.

We performed a WCET analysis of the main functions of the network code. This analysis allowed to statically estimate the end-to-end latency of transmitted time-triggered frames and verify the expected maximum latency. Overall, this paper provides a solution for deterministic communication with TTEthernet and WCET analyzable tasks and network code on the Patmos platform. To the best of our knowledge, this is the first open-source TTEthernet node with a WCET analyzable network stack.

Acknowledgement

This research has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA—Fog Computing for Robotics and Industrial Automation

Chapter 5

Synchronizing Real-Time Tasks in Time-Triggered Networks

By Eleftherios Kyriakakis, Jens Sparsø, Peter Puschner, and Martin Schoeberl [C2]

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.

5.1 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 [130].

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 [8] and its Ethernet-based extension TTEthernet [46]. 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 [59] and is used in the underdevelopment NASA Orion spacecraft [141].

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 [163].

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 [85]. Subsequently,

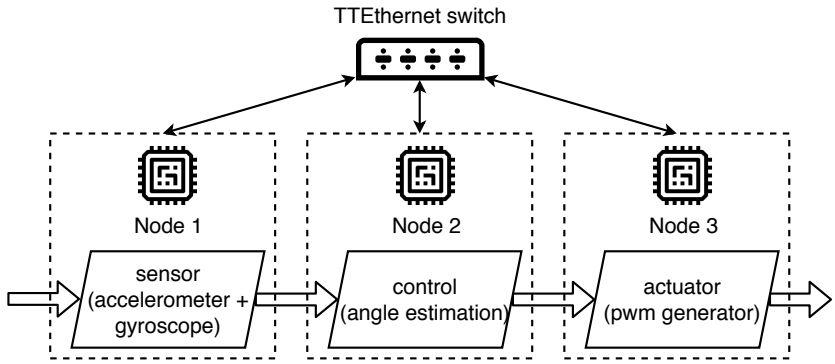


Figure 5.1: Classic industrial control example with tasks distributed over TTEthernet network.

this leads to improved resource utilization and end-to-end latency. The presented implementation uses the time-predictable T-CREST platform [94] 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 [162] 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 5.1.

The main contributions of this work are:

- 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 5.2 discusses the related work on time-triggered communication. Section 5.3 presents the system model

of the task and network schedule. Section 5.4 presents the proposed software framework and its implementation. Section 5.5 presents the synthetic control application example. Section 5.6 evaluates the proposed design using the developed application. Section 5.7 discusses the future improvements and plans. Section 5.8 concludes the paper.

5.2 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 [163] 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 [75]. 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 [101]. 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 [97], the authors pack multiple application messages in different time-triggered frames of selected order and length. In [150], 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 [87]. The implemented end-system acts as a synchronization client and the results show an observed jitter of 32 μs 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 [51], 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 [10] that has also been implemented in multicore real-time systems [50], 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.

5.3 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.

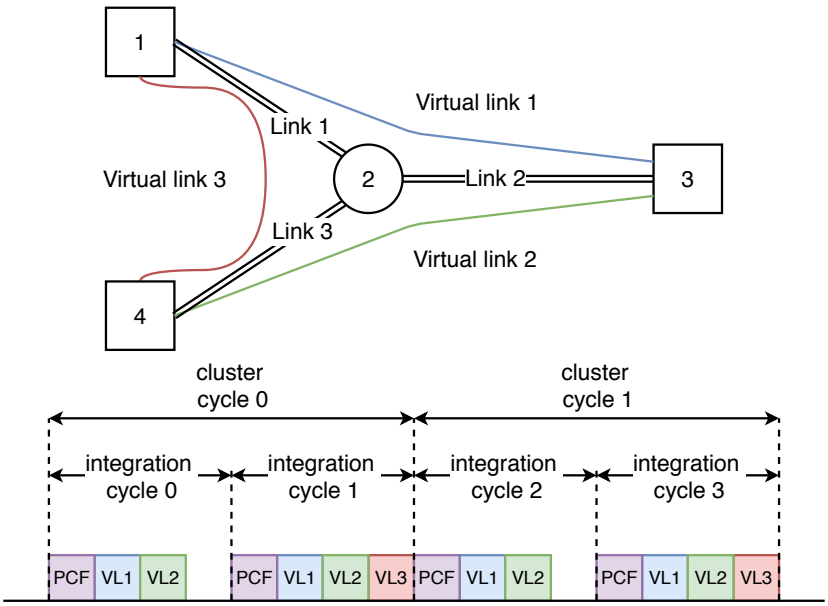


Figure 5.2: Example three end-system TTEthernet network with three VLs. Communication *cluster cycle* comprising of two *integration cycles*.

5.3.1 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 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 5.2.

TTEthernet employs a global fault-tolerant clock synchronization algorithm [59] 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 *integra-*

tion 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 [73]. 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 [146]. 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 [126].

5.3.2 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 5.4 discusses the runtime system implementation in detail.

In our task model, each task τ_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. 5.1c & 5.1b. 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.

$$S_{i,n} - S_{i,n-1} \leq T_i \pm J_i \qquad J_i \leq T_i \tag{5.1a}$$

$$S_{i,n} \geq n \times T_i + O_i \qquad O_i \leq T_i \tag{5.1b}$$

$$S_{i,n} + C_i \leq n \times T_i + D_i + J_i \qquad D_i, J_i \leq T_i \tag{5.1c}$$

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. 5.2. This is considered within a hyper-period of the schedule. Additionally, we define ϕ as a constant offset between release time, which allows us to account for the WCET of the runtime system or any other possible delays: $\phi = WCET_{runtime}$.

$$(S_{i,n} \geq S_{j,k} + C_j + \phi) \ \vee \ (S_{i,n} + C_i + \phi \leq S_{j,k}) \tag{5.2}$$

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

5.4 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.

5.4.1 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 [145]. 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 [18] and a static deterministic WCET analysis tool *platin* [89]. The platform also features a hardware-assisted time-stamping unit [138] 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 [59] standard presented in Figure 5.3 and timestamp it at the start-of-frame (SOF) byte.

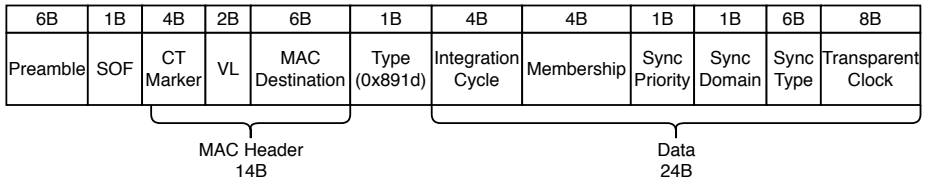


Figure 5.3: TTEthernet (AS6802) PCF Format.

5.4.2 Offline Scheduling

The synthesis of the task schedule based on the communication schedule resembles the process proposed by [75]. Figure 5.4 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 [89]. 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 [166]. 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 5.3. 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*

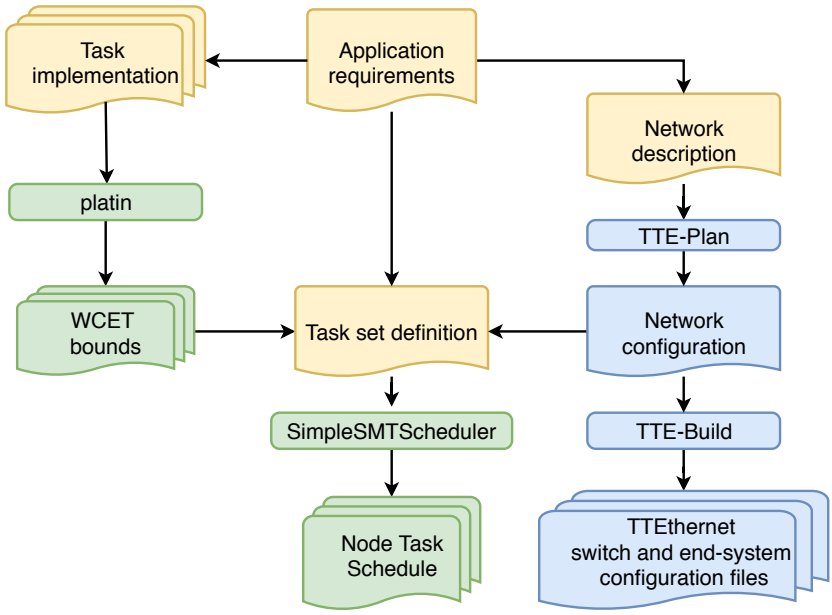


Figure 5.4: Design flow for synchronized communication and task schedule generation. Yellow represents the application specific definitions, green represents the components/tools of the proposed framework and blue represents the tools provided by TTTech.

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) [36].

5.4.3 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 5.5.

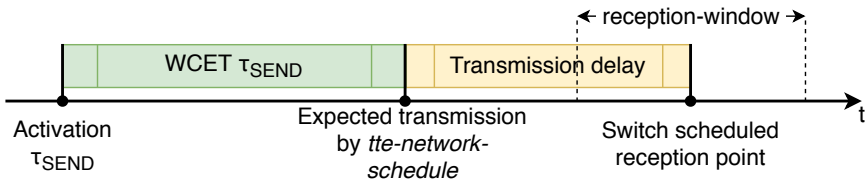


Figure 5.5: Task activation point in time relationship to the scheduled *transmission-window*.

Listing 5.1: 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;
```

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*.

5.4.4 Runtime System

In the proposed runtime system, tasks are defined as a simple C structure shown in Listing 5.1. 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 5.2. The release times of each task are initialized according to the generated offline schedule.

Listing 5.2: 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
  }
};
```

Listing 5.3: 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 cyclic execution dispatcher of the task set is defined as a loop function illustrated in Listing 5.3. 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 queries since after a task has been activated, the loop can safely break and take a new time reading to begin a new search.

5.4.5 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

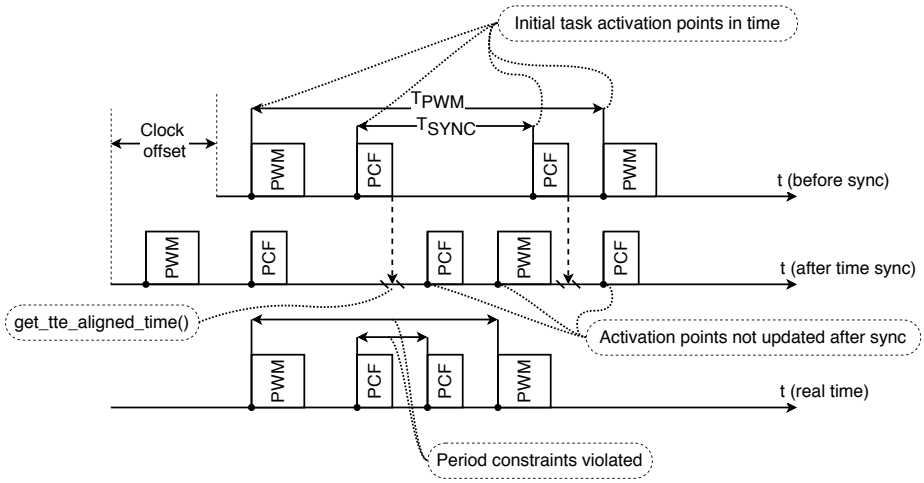


Figure 5.6: Erroneous task execution example, due to a miss-alignment between the time-base of the dispatcher schedule time and the upcoming task release times.

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* [59]. 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 [146]. The calculated clock 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 [161, 32].

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 the present and thus violate the task period. This miss-alignment is demonstrated in Figure 5.6.

5.5 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 [56].

The *sensor node* interfaces with a motion processing unit sensor MPU-9250 [109], 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 5.3a) [88]. The angle is then converted to a valid duty-cycle range according to Equation 5.3b.

$$\theta_x = 0.93 * (\theta_x + gyro_x * dt) + 0.07 * atan2\,(accel_y, accel_z) \tag{5.3a}$$

$$duty\_cycle = \frac{\theta_x * (0.1 - 0.015)}{180} + 0.015 \tag{5.3b}$$

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.

5.5.1 Task set

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

1. τ_{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. τ_{SSEND} transmits the sensor values to the control node

The *control node* executes four tasks:

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

The *actuator node* executes four tasks:

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

5.5.2 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*

5.6 Evaluation

5.6.1 System Setup

The cyclic executive's proposed synchronization with the communication schedule is evaluated and tested experimentally using a synthetic control application

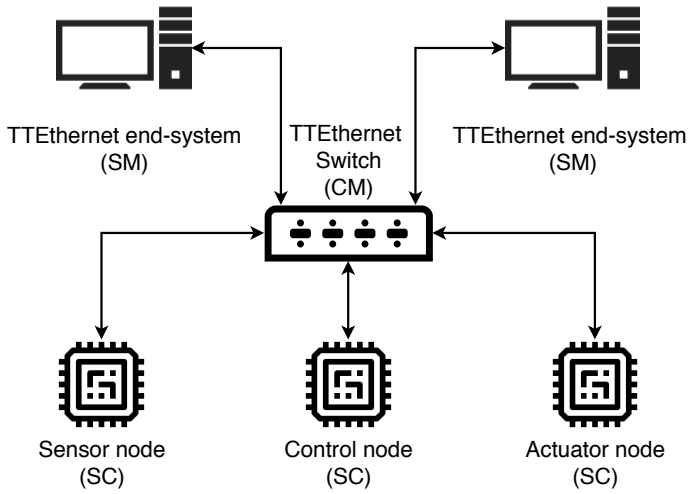


Figure 5.7: Experimental network setup of the evaluated control application example.

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 5.5 by executing the proposed runtime system. The hardware platform is synthesized on three FPGA Terasic DE2-115 boards [112] 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 5.7 presents the network setup of the evaluated control application example. A similar network setup has been described in [123].

5.6.2 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 5.1: 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	

Table 5.1 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 $\Phi_i = 44.737$ μs constant, which is used in the scheduling constraints (see Section 5.3).

Table 5.2 presents the combined generated task set using the presented SMT scheduler of the three nodes' distributed tasks. As discussed in Section 5.4, 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 μs, and this is added to the WCET of the related tasks: τ_{SSYNC}, τ_{CSYNC}, τ_{ASYNC}, τ_{CRECV} and τ_{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 τ_{PWM} can vary. In the presented analysis, it is considered that task τ_{PWM} generates the maximum duty-cycle duration of 2 ms (0.1%), which is added in the WCET of the task τ_{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 5.2: Generated task set of the three end-systems. The asterisks indicate a constrain by the *TTE network schedule*.

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

5.6.3 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 (τ_{SSYNC}, τ_{CSYNC}, τ_{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 ≈1.6 μs while the individual synchronization accuracy of each node relative to the TTEthernet switch was measured at ≈136 μs.

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 τ_{PWM} on the *actuation node* shown in Equation 5.4.

$$L_{e2e} = S_{\tau_{PWM},\emptyset} - (S_{\tau_{SENSE},\emptyset} + WCET_{\tau_{SENSE(a/g)}}) \tag{5.4}$$

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 ≈ 4.034 ms. The task τ_{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 τ_{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 [141], and thus an increase of ≈ 2.5 ms could be intolerable.

5.7 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 [21, 106, 16]. 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 [152].

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 [142, 134].

5.8 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 ≈ 1.6 μs and the end-to-end latency was bounded at ≈ 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.

Chapter 6

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

By Eleftherios Kyriakakis, Jens Sparsø, and Martin Schoeberl [C4]

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 a microsecond jitter and can perform a stable flight scenario as verified by the benchmark implementation.

6.1 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 [140]. 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 [120].

One approach to guarantee timing closure of distributed safety-critical applications is to employ the synchronous data-flow [2] model on time-predictable hardware architectures [31]. 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 aim by providing a synchronized interface to the underlying communication layer [163, 124]. 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 [165]. 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 [76, 149].

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 [164] 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 [145], and distributing the application using a time-triggered communication protocol TTEthernet [41]. 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 6.2 presents the use-case application of a flight management system. Section 6.3 presents a background on the technologies and tools employed by this work. Section 6.4 presents the design and implementation of the distributed flight management system using the proposed runtime system. Section 6.5 evaluates the proposed design using the developed application. Section 6.6 discusses related work on time-triggered communication and relevant use-cases. Section 6.8 summarizes the main conclusions derived from this work.

6.2 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. [80] 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 has 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.

Table 6.1: Flight controller closed-loop variables

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

The flight controller in use has been designed in SIMULINK [158] 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 6.1 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 stead-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 6.5 to drive the evaluation of the presented real-time system implementation and compare it against other non-distributed implementations.

6.3 Background

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

6.3.1 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 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 [73]. 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.

6.3.2 Offline Scheduler

The task and network schedules are synthesized using the open-source framework described by [164]. 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 [36].

6.3.3 Runtime System

This work aims to evaluate and deploy the open-source runtime system presented in [164]. 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.

6.3.4 Hardware Platform

The presented system is implemented on the open-source research platform T-CREST [94]. The platform features a time-predictable processor, Patmos [145], that uses WCET-optimized caches along with private scratchpad memories. Additionally, we use its toolchain, particularly the static WCET analysis tool *platin* [89], 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 [138]. 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.

6.4 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.

6.4.1 Task and Network Model

In this work, we use the offline scheduler described in Section 6.3, 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.

6.4.2 Communication

We distribute ROSACE over three nodes, as shown in Figure 6.1. 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 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 6.1 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 6.1. 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

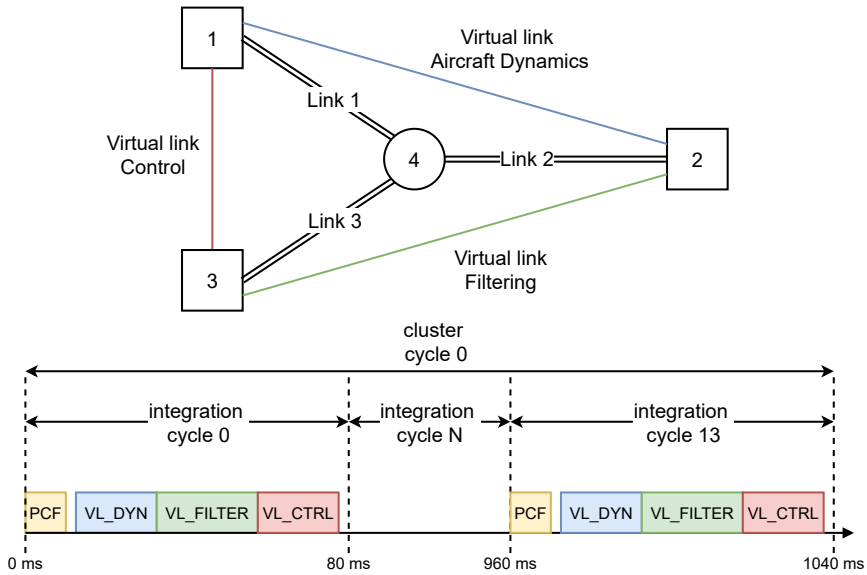


Figure 6.1: Distributed ROSACE topology and example TTEthernet VL communication schedule.

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 6.3, 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.

6.4.3 Static Scheduling

The ROSACE benchmark is composed of 12 periodic tasks (see Table 6.2) 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.

Originally, the case study is coded using the PRELUDE [56] formal language to generate a set of dependent periodic tasks. PRELUDE can add real-time

Listing 6.1: Implemented message structs containing the closed-loop variables (see Table 6.1) 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;
```

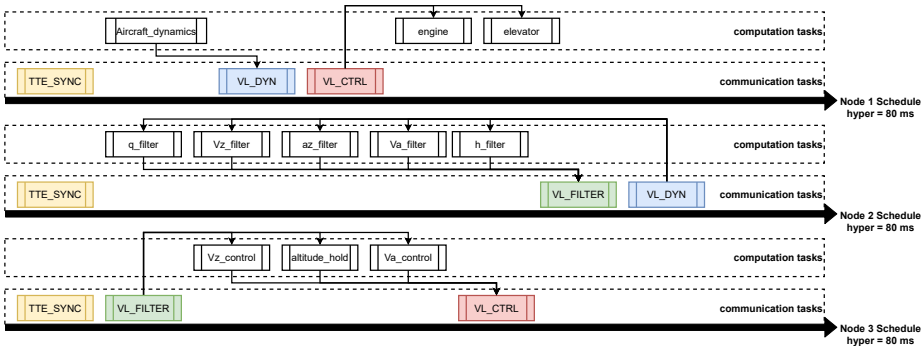


Figure 6.2: Example ROSACE tasks distribution on three nodes with software-based TTEthernet network communication.

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 [80] 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 6.2 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 [75]. First, we perform a static WCET analysis on the implemented ROSACE tasks using the tool *platin* [89] and present the results in Table 6.2.

The lack of a floating-point unit in the used hardware platform significantly increases the tasks' WCET. In [80], the authors report a WCET of 200μs for the aircraft_dynamics while the presented implementation is estimated at 17.9 ms. Consecutively, we derive valid periods for the tasks that aim to maintain the relative proportions to each other as defined in [80]:

- Aircraft dynamics: 20 ms

Table 6.2: 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

- Filtering: 40 ms
- Control loops: 80 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 [166] 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 6.1.

$$\phi = WCET_{dispatcher} + WCET_{read\_clock} + Offset_{clock} \tag{6.1}$$

6.4.4 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*.

6.5 Evaluation

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

6.5.1 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 synthesized on three Terasic DE2-115 FPGA boards [112] 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 6.3 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.

6.5.2 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 6.3 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 [87]. The existing dispatcher jitter, is hypothesized to be caused by reading the clock and

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

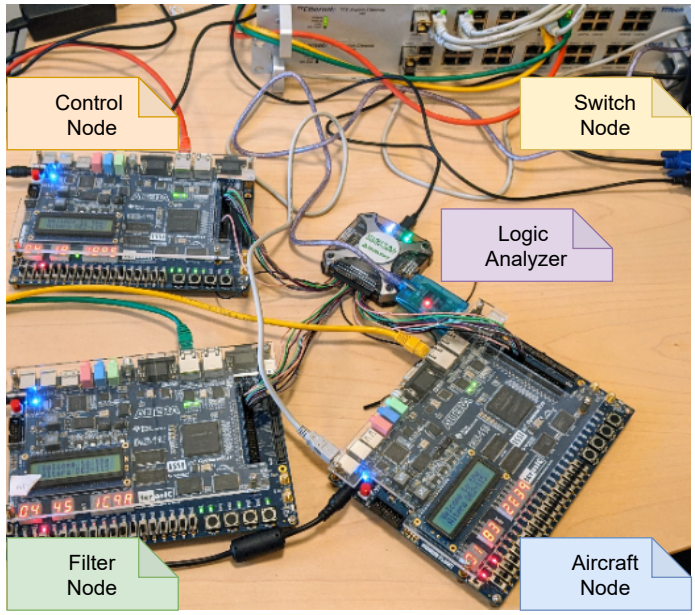


Figure 6.3: Distributed ROSACE setup over three nodes that are integrated in a TTEthernet network.

Table 6.3: 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

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.

6.5.3 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 derive the overall synchronization quality of the

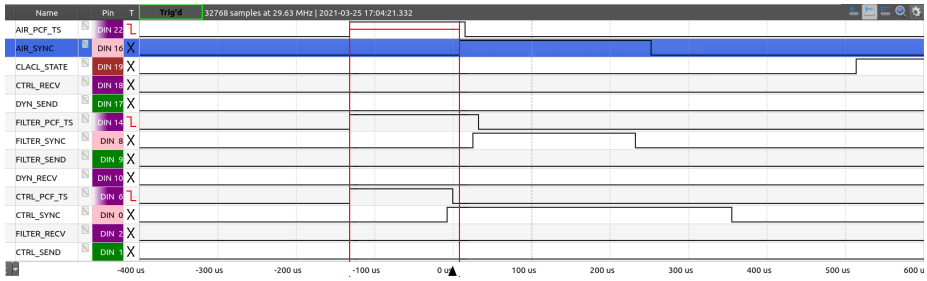


Figure 6.4: Measured clock synchronization accuracy of the synchronization task I/O pulse relative to the arrival time of the TTEthernet PCF I/O pulse.

node. We measure the offset using a logic analyzer and present the results in Figure 6.4. The nodes are synchronized to the network schedule to a measured precision of $\approx 100\mu$s. While the observed synchronization of the distributed task schedules relative to each other is $\leq 50\mu$s.

6.5.4 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 ≈ 400 seconds to reach the designated altitude with a reference vertical speed $Vz = 2.5m/s$ and a reference airspeed $Va = 230m/s$. Figure 6.5 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.

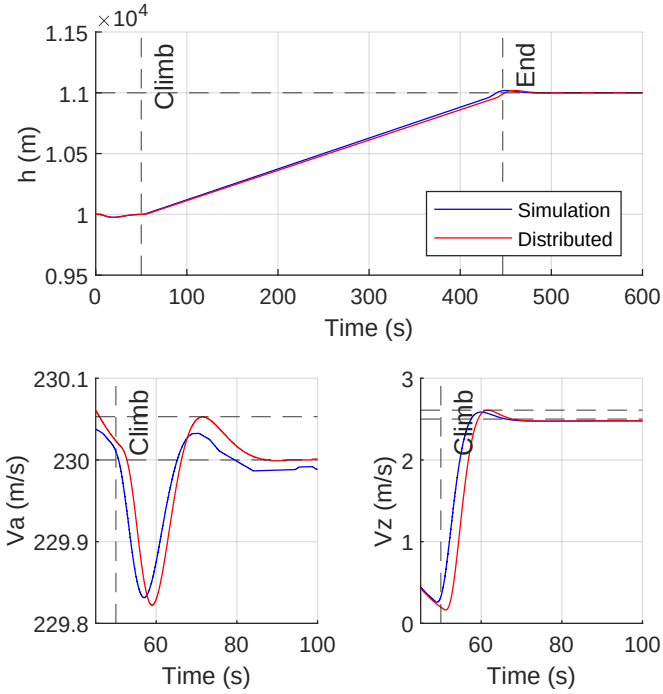


Figure 6.5: Aircraft step response comparison between simulation and distributed real-time system implementation. Commanded 1000 m climb starts at 50 seconds.

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 [80], the authors use the same validation rules to verify the correctness of the SIMULINK model performance. Table 6.4 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 the flight regarding the vertical speed Vz and true airspeed Va quality-of-control against a set of objectives discribed in Section 6.2. The system performs well within the specification bounds and with results very close to the simulated performance of the Linux implementation.

Table 6.4: 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%

6.6 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 [163] and has been explored in detail by [101], 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 [97, 20, 92]. 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 [105]. 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 [60]. 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 [81]. 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 [80], 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 [108] 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 [87], 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.

6.7 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 [157] 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.

6.8 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.

Chapter 7

Conclusion

This chapter summarizes the presented contributions of this work and how they can be integrated within an end-system to provide an overall time-predictable architecture suitable for real-time communication. Finally, it concludes this work by outlining future research based on the proposed hardware/software components.

7.1 Summary of Contributions

The presented thesis was composed of five research papers to enable time-predictable communication for networked end-systems by exploring the required components. The contributions of the presented research chapters can be thematically grouped into three areas:

1. Time synchronization
2. Open-source real-time communication
3. Synchronized task execution with communication

Time synchronization Time synchronization was first explored in Chapter 2 using the IEEE 1588 PTP. The developed software stack was able to achieve sub-microsecond precision. The design is enhanced by a proposed hardware-assisted timestamping and clock correction hardware component. The developed hardware component reduces the standard deviation (jitter) of the achieved clock synchronization as illustrated in Figures 2.8 & 2.9. This component can enable an end-system to accurately synchronize within TSN networks and enable time-aware communication as was illustrated in the implementation presented in [O1]. The investigation on time synchronization is extended in Chapter 3, which explores the current reliability and safety issues of PTP using a simulation setup of multiple network failures and malicious PTP masters. A fault-tolerant end-system design is presented, that combined with redundant network topology, can effectively guarantee precise clock synchronization.

Synchronized task execution with communication This work promotes the synchronous data flow model for hard real-time systems. Chapter 5 investigated the concept of end-to-end synchronization of computation tasks with the underlying time-triggered communication layer and proposed an open-source framework to allow scheduling of both computation and communication. A static scheduler is developed that uses constraint programming to derive valid cyclic schedules. A bare-metal runtime system enables the network-time synchronized execution of the generated schedule. This work illustrated that using a fully synchronous approach enables bounded communication and computation jitter and minimal memory usage as computation and memory resources can be statically scheduled.

Open-source real-time communication Chapter 4 first presented a prototype open-source TTEthernet node. The communication and synchronization mechanisms were implemented entirely in software and enable any bare-metal processor to be integrated within a TTEthernet network as a synchronization client. The experimental clock synchronization results (see Figure 4.12) highlighted the feasibility of the design as the TTEthernet node was able to synchronize with the network time with a precision of $\approx 3\mu$s. It is worth noting that the WCET analysis of the developed network stack illustrates the cost of this implementation. Transmission, reception and synchronization have shown to be time consuming software functions that should be taken into consideration by system designers of performance constrained embedded systems. Chapter 5 extends the developed TTEthernet node by presenting a complete time-triggered framework for scheduling computation and communication tasks together with an open-source runtime system. Chapter 6 demonstrates the successful operation of the proposed framework by distributing an avionic benchmark over an experimental TTEthernet network. The runtime system can successfully perform a flight scenario with microseconds jitter (see Table 6.3).

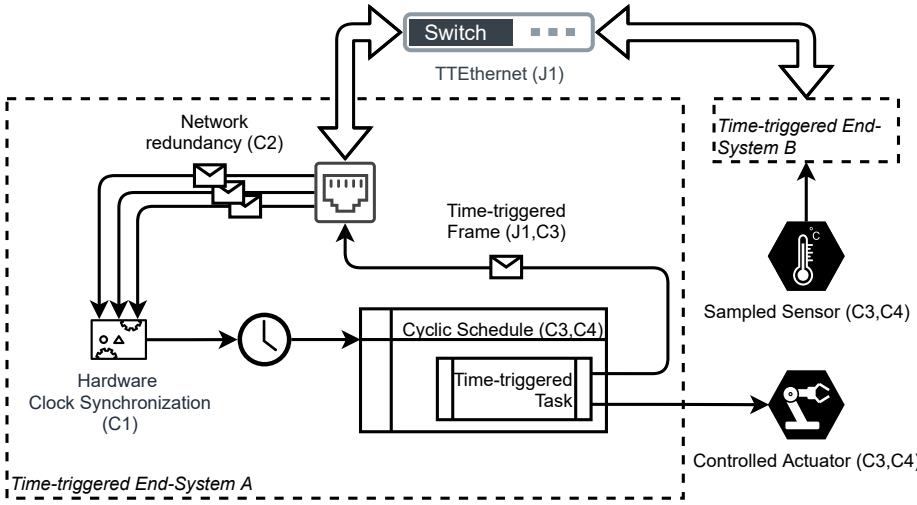


Figure 7.1: Composition of the fundamental components of an example time-triggered end-system integrated within a TTEthernet network. Papers indicated within parentheses ([C1], [C3], [J1], [C2], [C4]).

7.2 Composing a Time-Triggered End-System

The presented components of this work can be integrated with any WCET-analyzable platform to provide support for fault-tolerant time-synchronization and time-triggered communication and computation to industrial and safety-critical real-time applications. Figure 7.1 presents an overview of the proposed components mapped to an example time-triggered network architecture between two end-systems A & B (see Figure 1.1). The example illustrates how the proposed contributions, indicated in parentheses, can be combined to provide synchronous operation of a hard real-time cyber-physical system network. The network redundancy, together with the hardware clock synchronization unit, allow for fault-tolerant clock synchronization. The open-source network stack enables integration with industrial communication protocols (i.e. TTEthernet). Furtermore, the SMT scheduler, together with the time-predictable runtime system, allow for synchronized distributed task execution. Overall, the composed time-triggered end-system offers time-predictable sensor sampling and actuator control with synchronized communication and minimal jitter.

7.3 Future Research Outlook

As previously discussed, this work aims to design open-source hardware and software components that enable real-time communication and computation for distributed end-systems. Although the presented efforts have demonstrated the successful operation of an open-source time-triggered research platform, they have covered just part of the available communication protocols (e.g., TTEthernet, IEEE 1588), and have mainly focused on single-core processors. Some suggestions for future research topics are presented below:

Expanding communication support: The author suggests that part of future research extend the proposed runtime system to support different protocols such as PROFINET, EtherCAT and particularly TSN. The presented runtime system can be easily modified to support TSN provided that the existing TTEthernet synchronization function will be replaced with a PTP slave function similar to the one presented in Figure 2.6b. The author identifies that the primary research challenge is that TTEthernet schedules synchronization frames (PCF) together with data frames at the start of each integration cycle. In contrast, TSN does not specify a specific point-in-time that PTP frames are transmitted; thus, synchronization slave end-systems should always be ready to accept PTP frames. Thus reducing the determinism of computation and memory resources usage. It is proposed that scheduling of PTP frames together with data should be investigated to compose isochronous communication channels between end-systems and neighbouring TSN switches.

Extending the framework to multicore: Additionally, it is worth exploring time-triggered communication in multicore architectures. The author has made first efforts to modify the static scheduler to generate mapped schedule tables per core, and an initial coding example for the ROSACE benchmark has been developed [1] that executes on the T-CREST multicore platform using the Argo network-on-chip (NoC) message-passing library for inter-core communication [95]. The efforts presented in this thesis can serve as a base for future research to explore synchronization of task chains beyond multicore processors to mixed NoC and time-triggered communication topologies as has been already proposed in [O2]. In the proposed work, the author describes a communication scheme that hides the spatial locality of cores by employing a NoC driver layer that maps each core to an IP address. The driver-layer converts the on-chip core-to-core message-passing communication to off-chip Ethernet communication depending on the IP address. This operation is transparent to the user

[1] *https://github.com/t-crest/patmos/blob/master/c/apps/rosace/rosace_patmos_argo.c*

application level, which communicates using only IP packets. It is hypothesized that the combination of Internet Protocol core masking with an on-chip TDM-based NoC layer and a time-triggered off-chip communication channel can enable flexible distribution of applications on real-time distributed multicore systems such as modern avionics [131].

Bibliography

[1] Danny Dolev, Nancy A Lynch, Shlomit S Pinter, Eugene W Stark, and William E Weihl. "Reaching approximate agreement in the presence of faults". In: *Journal of the ACM (JACM)* 33.3 (1986), pp. 499–516.

[2] Edward A. Lee and David G. Messerschmitt. "Synchronous Data Flow". In: *Proceedings of the IEEE* 75.9 (Sept. 1987), pp. 1235–1245. ISSN: 0018-9219. DOI: 10.1109/PROC.1987.13876.

[3] John A. Stankovic. "Misconceptions about real-time computing: A serious problem for next-generation systems". In: *Computer* 21.10 (1988), pp. 10–19.

[4] Flaviu Cristian. "Probabilistic clock synchronization". In: *Distributed computing* 3.3 (1989), pp. 146–158.

[5] Riccardo Gusella and Stefano Zatti. "The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3 BSD". In: *IEEE transactions on Software Engineering* 15.7 (1989), pp. 847–853.

[6] Hermann Kopetz, Andreas Damm, Christian Koza, Marco Mulazzani, Wolfgang Schwabl, Christoph Senft, and Ralph Zainlinger. "Distributed fault-tolerant real-time systems: The Mars approach". In: *IEEE Micro* 9.1 (1989), pp. 25–40.

[7] Parameswaran Ramanathan, Kang G. Shin, and Ricky W. Butler. "Fault-tolerant clock synchronization in distributed systems". In: *Computer* 23.10 (1990), pp. 33–42.

[8] Hermann Kopetz and Günter 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.

[9] William Buchanan. “CAN bus”. eng. In: *Computer Busses* (2000), pp. 333–343. DOI: 10.1016/B978-034074076-7/50021-3,10.1016/B978-0-340-74076-7.X5000-7.

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

[11] Omer Gurewitz and Moshe Sidi. “Estimating one-way delays from cyclic-path delay measurements”. In: *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*. Vol. 2. IEEE. 2001, pp. 1038–1044.

[12] Rajeshwari Banakar, Stefan Steinke, Bo-Sik Lee, Mahesh Balakrishnan, and Peter Marwedel. “Scratchpad memory: A design alternative for cache on-chip memory in embedded systems”. In: *Proceedings of the Tenth International Symposium on Hardware/Software Codesign. CODES 2002 (IEEE Cat. No. 02TH8627)*. IEEE. 2002, pp. 73–78.

[13] Wilfried Elmenreich and Martin Delvai. “Time-triggered communication with UARTs”. In: *Factory Communication Systems, 2002. 4th IEEE International Workshop on*. IEEE. 2002, pp. 97–104.

[14] Gabriel Leen and Donal Heffernan. “TTCAN: A New Time-Triggered Controller Area Network”. In: *Microprocessors and Microsystems* 26.2 (2002), pp. 77–94.

[15] Igor Mohor. *Ethernet IP core design document*. eng. 2002. URL: http://opencores.org/project,ethmac (visited on 02/07/2018).

[16] Erland Nilsson. “Design and Implementation of a hot-potato Switch in a Network on Chip”. In: *Mémoire, Departement of Microelectronics and Information Technology, Royal Institute of Technology* (2002).

[17] Airlines Electronic Engineering Committee. *Aircarft Data Network, Part 7-avionics Full Duplex Switched Ethernet (AFDX) Network*. 2004.

[18] Chris Lattner and Vikram 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. ISBN: 0-7695-2102-9.

[19] Roman Obermaisser. *Event-triggered and time-triggered control paradigms*. Vol. 22. Springer Science & Business Media, 2004.

[20] Paul Pop, Petru Eles, and Zebo Peng. “Schedulability-driven communication synthesis for time triggered embedded systems”. In: *Real-Time Systems* 26.3 (2004), pp. 297–325.

[21] Kees Goossens, John Dielissen, and Andrei Radulescu. “Æthereal network on chip: concepts, architectures, and implementations”. In: *IEEE Design & Test of Computers* 22.5 (2005), pp. 414–421.

[22] Hermann Kopetz, Astrit Ademaj, Petr Grillinger, and Klaus Steinhammer. "The time-triggered Ethernet (TTE) design". In: *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*. IEEE. 2005, pp. 22–33.

[23] Ingomar Wenzel, Raimund Kirner, Peter Puschner, and Bernhard Rieder. "Principles of timing anomalies in superscalar processors". In: *Fifth International Conference on Quality Software (QSIC'05)*. IEEE. 2005, pp. 295–303.

[24] John C. Eidson. *Measurement, Control, and Communication Using IEEE 1588*. 1st. Springer Science & Business Media, 2006. ISBN: 9781846282508.

[25] David L. Mills. *Computer Network Time Synchronization: The Network Time Protocol*. 2006.

[26] Roman Obermaisser. "Reuse of CAN-based legacy applications in time-triggered architectures". In: *IEEE Transactions on Industrial Informatics* 2.4 (2006), pp. 255–268.

[27] F Sethna, FH Ali, and E Stipidis. "What lessons can controller area networks learn from FlexRay". In: *2006 IEEE Vehicle Power and Propulsion Conference*. IEEE. 2006, pp. 1–4.

[28] Klaus Steinhammer, Petr Grillinger, Astrit Ademaj, and Hermann Kopetz. "A time-triggered ethernet (TTE) switch". In: *DATE '06: Proceedings of the conference on Design, automation and test in Europe*. Munich, Germany: European Design and Automation Association, 2006, pp. 794–799. ISBN: 3-9810801-0-6.

[29] Astrit Ademaj and Hermann Kopetz. "Time-triggered Ethernet and IEEE 1588 clock synchronization". In: *2007 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. IEEE. 2007, pp. 41–43.

[30] L Benetazzo, C Narduzzi, and M Stellini. "Analysis of clock tracking performances for a software-only IEEE 1588 implementation". In: *Proceedings of the Instrumentation and Measurement Technology Conference Proceedings (IMTC)*. IEEE, 2007, pp. 1–6.

[31] Stephen A Edwards and Edward A Lee. "The case for the precision timed (PRET) machine". In: *Proceedings of the 44th annual Design Automation Conference*. 2007, pp. 264–265.

[32] 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.

[33] Peter Neumann. "Communication in industrial automation-What is going on?" In: *Control Engineering Practice* 15.11 (2007), pp. 1332–1347.

[34] Wilfried Steiner. "Advancements in Dependable Time-Triggered Communication". In: *Software Technologies for Embedded and Ubiquitous Systems*. Ed. by Roman Obermaisser, Yunmook Nah, Peter Puschner, and Franz J. Rammig. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 57–66. ISBN: 978-3-540-75664-4.

[35] Klaus Steinhammer and Astrit Ademaj. "Hardware Implementation of the Time-Triggered Ethernet Controller". In: *Embedded System Design: Topics, Techniques and Trends, IFIP TC10 Working Conference: International Embedded Systems Symposium (IESS), May 30 - June 1, 2007, Irvine, CA, USA*. Ed. by Achim Rettberg, Mauro Cesar Zanella, Rainer Dömer, Andreas Gerstlauer, and Franz-Josef Rammig. Vol. 231. IFIP Advances in Information and Communication Technology. Springer, 2007, pp. 325–338. ISBN: 978-0-387-72257-3.

[36] Leonardo De Moura and Nikolaj 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.

[37] Institute of Electrical and Electronics Engineers. *1588-2008 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. 2008.

[38] Fang He and Shouhua Zhao. "Research on synchronous control of nodes in distributed network system". In: *2008 IEEE International Conference on Automation and Logistics*. IEEE. 2008, pp. 2999–3004.

[39] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. eng. 2008. DOI: 10.1109/IEEESTD.2008.4579760.

[40] Sungwon Lee. "An enhanced IEEE 1588 time synchronization algorithm for asymmetric communication link using block burst transmission". In: *IEEE communications letters* 12.9 (2008), pp. 687–689.

[41] Wilfried Steiner, Günther Bauer, and David Jameux. "Ethernet for space applications: TTEthernet". In: *International SpaceWire Conference 2008, Nara, Japan*. 2008.

[42] András Varga and Rudolf Hornig. "An Overview of the OMNeT++ Simulation Environment". In: *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*. Simutools '08. Marseille, France: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008. ISBN: 9789639799202.

[43] D Macii, D Fontanelli, and D Petri. "A master-slave synchronization model for enhanced servo clock design". In: *2009 International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. IEEE. 2009, pp. 1–6.

[44] Martin Schoeberl. “Time-predictable computer architecture”. In: *EURASIP Journal on Embedded Systems* 2009 (2009), pp. 1–17.

[45] Martin Schoeberl, Wolfgang Puffitsch, and Benedikt Huber. “Towards time-predictable data caches for chip-multiprocessors”. In: *IFIP International Workshop on Software Technolgies for Embedded and Ubiquitous Systems*. Springer. 2009, pp. 180–191.

[46] Wilfried Steiner and Günther Bauer. “TTEthernet: Time-triggered services for Ethernet networks”. In: *Proceedings of 28th IEEE Digital Avionics Conference (DASC)*. 2009.

[47] Wilfried Steiner, Günther Bauer, Brendan Hall, Michael Paulitsch, and Srivatsan Varadarajan. “TTEthernet dataflow concept”. In: *2009 Eighth IEEE International Symposium on Network Computing and Applications*. IEEE. 2009, pp. 319–322.

[48] Weidong Ye. “IEEE1588 Clock servo algorithm”. In: *Proceedings of the 9th International Conference on Electronic Measurement & Instruments*. IEEE, Aug. 2009, p. 5274861.

[49] Jeff Preston, Dan Blankenship, Les Hoy, MF Ohmes, Andrey Gueorguiev, and Juergen Stein. “Novel timing method using IEEE 1588 and synchronous Ethernet for Compton telescope”. In: *Proceedings of the Nuclear Science Symposium Conference Record (NSS/MIC)*. IEEE, 2010, pp. 1404–1407.

[50] Anders P. Ravn and Martin 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)*. Prague, Czech Republic: ACM, 2010, pp. 63–69. ISBN: 978-1-4503-0122-0. DOI: 10.1145/1850771.1850779.

[51] Florian Bartols, Till Steinbach, Franz Korf, and Thomas 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.

[52] Hermann Kopetz. “Temporal Relations”. In: *Real-Time Systems*. Springer, 2011, pp. 111–133.

[53] Yan Lin, Li Hao, and Tian Dan. “Research and implementation in synchronized system of data acquisition based on IEEE 1588”. In: *Proceedings of the 10th International Conference on Electronic Measurement & Instruments (ICEMI)*. Vol. 2. IEEE, 2011, pp. 198–201.

[54] Maciej Lipiński, Tomasz Włostowski, Javier Serrano, and Pablo Alvarez. "White rabbit: A PTP application for robust sub-nanosecond synchronization". In: *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*. IEEE, 2011, pp. 25–30.

[55] T. Mizrahi. "Time synchronization security using IPsec and MACsec". In: *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. 2011, pp. 38–43. DOI: 10.1109/ISPCS.2011.6070153.

[56] Claire Pagetti, Julien Forget, Frédéric Boniol, Mikel Cordovilla, and David Lesens. "Multi-task implementation of multi-periodic synchronous programs". In: *Discrete event dynamic systems* 21.3 (2011), pp. 307–338.

[57] Jae Won Park, Jin Ha Hwang, Won Young Chung, Seung Woo Lee, and Yong Surk Lee. "Design time stamp hardware unit supporting IEEE 1588 standard". In: *SoC Design Conference (ISOCC), 2011 International*. IEEE, 2011, pp. 345–348.

[58] Mingzhu Qi, Xiaoli Wang, and Zhiqiang Yang. "Design and implementation of IEEE1588 time synchronization messages timestamping based on FPGA". In: *Electric Utility Deregulation and Restructuring and Power Technologies (DRPT), 2011 4th International Conference on*. IEEE, 2011, pp. 1566–1570.

[59] TTTech. *AS6802: Time-Triggered Ethernet*. 2011.

[60] Guillaume Brau and Claire Pagetti. "TTEthernet-based architecture simulation with Ptolemy II". In: *6th Junior Researcher Workshop on Real-Time Computing (JRWRTC 2012)*. 2012, p29–32.

[61] Franck Cassez, René Rydhof Hansen, and Mads Chr Olesen. "What is a timing anomaly?" In: *12th International Workshop on Worst-Case Execution Time Analysis*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2012.

[62] Rodney Cummings, Kai Richter, Rolf Ernst, Jonas Diemer, and Arkadeb Ghosal. "Exploring use of Ethernet for in-vehicle control applications: AFDX, TTEthernet, EtherCAT, and AVB". In: *SAE International Journal of Passenger Cars-Electronic and Electrical Systems* 5.2012-01-0196 (2012), pp. 72–88.

[63] T. Mizrahi. "Slave diversity: Using multiple paths to improve the accuracy of clock synchronization protocols". In: *2012 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication Proceedings*. 2012, pp. 1–6. DOI: 10.1109/ISPCS.2012.6336621.

[64] Till Steinbach, Hyung-Taek Lim, Franz Korf, Thomas C Schmidt, Daniel Herrscher, and Adam Wolisz. "Tomorrow's in-car interconnect? A competitive evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802)". In: *2012 IEEE Vehicular Technology Conference (VTC Fall)*. IEEE. 2012, pp. 1–5.

[65] Ekarin Suethanuwong. "Scheduling time-triggered traffic in TTEthernet systems". In: *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*. IEEE. 2012, pp. 1–4.

[66] Domitian Tamas-Selicean, Paul Pop, and Wilfried Steiner. "Synthesis of Communication Schedules for TTEthernet-Based Mixed-Criticality Systems". In: *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM. 2012, pp. 473–482.

[67] Guang You Yang, Yi Zheng, Zhi Yan Ma, and Xin Yu Hu. "The Implementation of IEEE 1588-2008 Precision Time Protocol on the STM32F107". In: *Key Engineering Materials* 522 (2012), pp. 868–873.

[68] Gonzalo Carvajal and Sebastian Fischmeister. "An open platform for mixed-criticality real-time Ethernet". In: *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2013, pp. 153–156.

[69] PPM Jansweijer, HZ Peek, and E De Wolf. "White Rabbit: Sub-nanosecond timing over Ethernet". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 725 (2013), pp. 187–190.

[70] Maciej Lipinski. "White Rabbit-Ethernet-based solution for sub-ns synchronization and deterministic, reliable data delivery". In: *IEEE Plenary Meeting Geneva*. Vol. 15. 2013.

[71] Peter Puschner, Daniel Prokesch, Benedikt Huber, Jens Knoop, Stefan Hepp, and Gernot Gebhard. "The T-CREST approach of compiler and WCET-analysis integration". In: *16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2013)*. IEEE. 2013, pp. 1–8.

[72] A. Shpiner, Y. Revah, and T. Mizrahi. "Multi-path Time Protocols". In: *2013 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS) Proceedings*. 2013, pp. 1–6. DOI: 10.1109/ISPCS.2013.6644754.

[73] Wilfried Steiner and Bruno Dutertre. "The TTEthernet synchronisation protocols and their formal verification". In: *International Journal of Critical Computer-Based Systems 17* 4.3 (2013), pp. 280–300.

[74] Ivan Studnia, Vincent Nicomette, Eric Alata, Yves Deswarte, Mohamed Kaaniche, and Youssef Laarouchi. "Survey on security threats and protection mechanisms in embedded automotive networks". In: *Proc. DSN* (2013).

[75] Silviu S Craciunas, Ramon Serna Oliver, and Valentin 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.

[76] Peter Danielis, Jan Skodzik, Vlado Altmann, Eike Bjoern Schweissguth, Frank Golatowski, Dirk Timmermann, and Joerg Schacht. "Survey on Real-Time Communication via Ethernet in industrial automation environments". eng. In: *19th Ieee International Conference on Emerging Technologies and Factory Automation, Etfa 2014* (2014). DOI: 10.1109/ETFA.2014.7005074.

[77] Guy Durrieu, Madeleine Faugère, Sylvain Girbal, Daniel Gracia Pérez, Claire Pagetti, and Wolfgang Puffitsch. "Predictable flight management system implementation on a multicore processor". In: *Embedded Real Time Software (ERTS'14)*. 2014.

[78] Friedrich Groß, Till Steinbach, Franz Korf, Thomas C Schmidt, and Bernd Schwarz. "A hardware/software co-design approach for Ethernet controllers to support time-triggered traffic in the upcoming IEEE TSN standards". In: *Proceedings of the Fourth International Conference on Consumer Electronics–Berlin (ICCE-Berlin)*. IEEE, 2014, pp. 9–13.

[79] Tal Mizrahi. *Security Requirements of Time Protocols in Packet Switched Networks*. RFC 7384. Oct. 2014. DOI: 10.17487/RFC7384. URL: https://rfc-editor.org/rfc/rfc7384.txt.

[80] Claire Pagetti, David Saussié, Romain Gratia, Eric Noulard, and Pierre 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.

[81] Claudius Ptolemaeus, ed. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.

[82] Martin Schoeberl, David Vh Chong, Wolfgang Puffitsch, and Jens Sparsø. "A time-predictable memory network-on-chip". In: *14th International Workshop on Worst-Case Execution Time Analysis*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2014.

[83] Domiţian Tămaş-Selicean and Paul Pop. "Optimization of TTEthernet networks to support best-effort traffic". In: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. IEEE. 2014, pp. 1–4.

[84] Jack Whitham and Martin Schoeberl. "WCET-based comparison of an instruction scratchpad and a method cache". In: *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*. IEEE. 2014, pp. 301–308.

[85] Jaume Abella, Carles Hernandez, Eduardo Quiñones, Francisco J Cazorla, Philippa Ryan Conmy, Mikel Azkarate-Askasua, Jon Perez, Enrico Mezzetti, and Tullio 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.

[86] *DP83640 Precision PHYTER-IEEE 1588 Precision Time Protocol Transceiver*. Texas Instruments. Apr. 2015. URL: www.ti.com/lit/ds/symlink/dp83630.pdf (visited on 07/03/2018).

[87] Thomas Fruhwirth, Wilfried Steiner, and Bernhard Stangl. "TTEthernet SW-based end system for AUTOSAR". eng. In: *2015 10th Ieee International Symposium on Industrial Embedded Systems, Sies 2015 - Proceedings* (2015), pp. 21–28. DOI: 10.1109/SIES.2015.7185037.

[88] Pengfei Gui, Liqiong Tang, and Subhas 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.

[89] Stefan Hepp, Benedikt Huber, Jens Knoop, Daniel Prokesch, and Peter 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.

[90] Andrew T Loveless. "On TTEthernet for integrated Fault-Tolerant spacecraft networks". In: *AIAA SPACE 2015 Conference and Exposition*. 2015, p. 4526.

[91] Luca Pezzarossa, Jakob Kenn Toft, Jesper Lønbæk, and Russell Barnes. *Implementation of an Ethernet-Based Communication Channel for the Patmos Processor*. eng. 2015.

[92] Francisco Pozo, Guillermo Rodriguez-Navas, Hans Hansson, and Wilfried 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.

[93] PTPd. *PTPd*. 2015. URL: https://github.com/ptpd/ptpd (visited on 06/02/2018).

[94] Martin Schoeberl et al. "T-CREST: Time-predictable Multi-Core Architecture for Embedded Systems". In: *Journal of Systems Architecture* 61.9 (2015), pp. 449–471. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2015.04.002.

[95] Rasmus Bo Sørensen, Wolfgang Puffitsch, Martin Schoeberl, and Jens Sparsø. "Message passing on a time-predictable multicore processor". In: *2015 IEEE 18th International Symposium on Real-Time Distributed Computing*. IEEE. 2015, pp. 51–59.

[96] Karthik Sridharan, KK Goossens, Nicola Concer, and HGH Bart Vermeulen. "Investigation of time-synchronization over Ethernet in-vehicle networks for automotive applications". MA thesis. Eindhoven: Eindhoven University of Technology, 2015.

[97] Domiţian Tămaş–Selicean, Paul Pop, and Wilfried Steiner. "Design optimization of TTEthernet-based distributed real-time systems". In: *Real-Time Systems* 51.1 (2015), pp. 1–35.

[98] TTTech. "Deterministic Ethernet & TSN: Automotive and Industrial IoT". In: *Industrial Ethernet Book* 89 (July 2015).

[99] *802.1AS-Rev - Timing and Synchronization for Time-Sensitive Applications*. `http://www.ieee802.org/1/pages/802.1AS-rev.html`. Accessed: 17.12.2020. 2016.

[100] Sergiy Bogomolov, Christian Herrera, and Wilfried Steiner. "Verification of Fault-Tolerant Clock Synchronization Algorithms." In: *ARCH@ CPSWeek*. 2016, pp. 36–41.

[101] Silviu S Craciunas and Ramon Serna Oliver. "Combined task-and network-level scheduling for distributed time-triggered systems". In: *Real-Time Systems* 52.2 (2016), pp. 161–200.

[102] Silviu S Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. "Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks". In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. ACM. 2016, pp. 183–192.

[103] *Cyclone IV FPGA Device Family Overview*. ALTERA. Mar. 2016. URL: https://www.altera.com/en_US/pdfs/literature/hb/cyclone-iv/cyiv-51001.pdf (visited on 07/03/2018).

[104] Institute of Electrical and Electronics Engineers. *Time-Sensitive Networking Task Group*. 2016. URL: http://ieee802.org/1/pages/tsn.html (visited on 06/22/2018).

[105] Sardaouna Hamadou, John Mullins, and Abdelouahed 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.

[106] Evangelia Kasapaki, Martin Schoeberl, Rasmus Bo Sørensen, Christian T. Müller, Kees Goossens, and Jens Sparsø. "Argo: A Real-Time Network-on-Chip Architecture with an Efficient GALS Implementation". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24 (2016), pp. 479–492. DOI: 10.1109/TVLSI.2015.2405614.

[107] Tamás Kovácsházy. "Towards a quantization based accuracy and precision characterization of packet-based time synchronization". In: *2016 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. IEEE. 2016, pp. 1–6.

[108] Ye Liu, Hiroshi Sasaki, Shinpei Kato, and Masato 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.

[109] *MPU-9250 Product Specification*. InvenSense. June 2016. URL: https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf (visited on 04/06/2020).

[110] Tiyam Robati, Abdelouahed Gherbi, and John Mullins. "A modeling and verification approach to the design of distributed IMA architectures using TTEthernet". In: *Procedia Computer Science* 83 (2016), pp. 229–236.

[111] Wilfried Steiner and Stefan Poledna. "Fog computing as enabler for the Industrial Internet of Things". In: *e & i Elektrotechnik und Informationstechnik* 133.7 (2016), pp. 310–314.

[112] *Terasic DE2-115 User Manual*. Altera. Mar. 2016. URL: https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-1404062209-de2-115-user-manual.pdf (visited on 03/25/2020).

[113] Daniel Thiele and Rolf Ernst. "Formal worst-case timing analysis of Ethernet TSN's burst-limiting shaper". In: *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2016, pp. 187–192.

[114] Wolfgang Wallner. *LibPLN: A Library for Efficient Powerlaw Noise Generation*. 2016.

[115] Wolfgang Wallner. *LibPTP: A Library for PTP Simulation*. 2016.

[116] Wolfgang Wallner. "Simulation of Time-synchronized Networks using IEEE 1588-2008". PhD thesis. Wien, 2016.

[117] Bogdan M Wilamowski and J David Irwin. *Industrial communication systems*. CRC Press, 2016.

[118] Yingjing Zhang, Feng He, Guangshan Lu, and Huagang Xiong. "Clock synchronization compensation of Time-Triggered Ethernet based on least squares algorithm". In: *2016 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*. IEEE. 2016, pp. 1–5.

[119] Xuan Zhou, Feng He, and Tong Wang. "Using network calculus on worst-case latency analysis for TTEthernet in preemption transmission mode". In: *2016 10th International Conference on Signal Processing and Communication Systems (ICSPCS)*. IEEE. 2016, pp. 1–8.

[120] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. "End-to-end timing analysis of cause-effect chains in automotive embedded systems". In: *Journal of Systems Architecture* 80 (2017), pp. 104–113.

[121] Silviu S Craciunas, Ramon Serna Oliver, and TC AG. "An overview of scheduling mechanisms for time-sensitive networks". In: *Proceedings of the Real-time summer school LÉcole dÉté Temps Réel (ETR)* (2017), pp. 1551–3203.

[122] Marina Gutiérrez, Wilfried Steiner, Radu Dobrin, and Sasikumar Punnekkat. "Synchronization quality of IEEE 802.1 AS in large-scale industrial automation networks". In: *Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2017, pp. 273–282.

[123] Peter Lambert. *D11.4-Final demonstrator implementation and evaluation*. Tech. rep. EMC2 Project Consortium, Apr. 2017. URL: https://www.artemis-emc2.eu/fileadmin/user_upload/Publications/Deliverables/EMC2_D11.4_WP11_Final_demonstrator_implementation_and_evaluation_v1.0.pdf (visited on 03/25/2020).

[124] Ayhan Mehmed, Sasikumar Punnekkat, and Wilfried 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.

[125] T. Pereira, L. Barreto, and A. Amaral. "Network and information security challenges within Industry 4.0 paradigm". In: *Procedia Manufacturing* 13 (2017).

[126] Miladin Sandić, Ivan Velikić, and Aleksandar 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.

[127] *STM32F107VC*. DocID15274 Rev 10. ST Microelectronics. Mar. 2017. URL: https://www.st.com/en/microcontrollers/stm32f107vc.html (visited on 07/03/2018).

[128] T-CREST. *Patmos Source*. 2017. URL: https://github.com/t-crest/patmos (visited on 06/02/2018).

[129] Luxi Zhao, Paul Pop, Qiao Li, Junyan Chen, and Huagang Xiong. "Timing analysis of rate-constrained traffic in TTEthernet using network calculus". In: *Real-Time Systems* 53.2 (2017), pp. 254–287.

[130] Richard Zurawski. *Industrial communication technology handbook, second edition*. eng. CRC Press, 2017, pp. 1–1756. ISBN: 9781482207323, 9781482207330. DOI: 10.1201/b17365.

[131] Laure Abdallah, Jérôme Ermont, Jean-Luc Scharbarg, and Christian Fraboul. "Reducing afdx jitter in a mixed noc/afdx architecture". In: *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*. IEEE. 2018, pp. 1–4.

[132] Sascha Einspieler, Benjamin Steinwender, and Wilfried Elmenreich. "Integrating time-triggered and event-triggered traffic in a hard real-time system". In: *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. IEEE. 2018, pp. 122–128.

[133] Janos Farkas, Lucia Lo Bello, and Craig Gunther. "Time-sensitive networking standards". In: *IEEE Communications Standards Magazine* 2.2 (2018), pp. 20–21.

[134] Voica Gavriluţ and Paul 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.

[135] Voica Gavriluţ, Luxi Zhao, Michael L Raagaard, and Paul Pop. "AVB-aware routing and scheduling of time-triggered traffic for TSN". In: *Ieee Access* 6 (2018), pp. 75229–75243.

[136] Shiying He, Liansheng Huang, Jun Shen, Ge Gao, Guanghong Wang, Xiaojiao Chen, and Lili Zhu. "Time Synchronization Network for EAST Poloidal Field Power Supply Control System Based on IEEE 1588". In: *IEEE Transactions on Plasma Science* 46.7 (2018), pp. 2680–2684.

[137] *Intel's Fog Reference Design Overview*. Intel. Apr. 2018. URL: https://www.intel.com/content/www/us/en/internet-of-things/fog-reference-design-overview.html (visited on 06/22/2018).

[138] Eleftherios Kyriakakis, Jens Sparsø, and Martin 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.

[139] Elena Lisova. "Monitoring for Securing Clock Synchronization". PhD thesis. Mälardalen University, 2018.

[140] Tulika Mitra, Jürgen Teich, and Lothar Thiele. "Time-critical systems design: A survey". In: *IEEE Design & Test* 35.2 (2018), pp. 8–26.

[141] M. Paulitsch, E Schmidt, C Scherrer, and H Kantz. "Industrial Applications". In: *Time-Triggered Communication*. CRC Press, 2018. Chap. 14, pp. 315–333.

[142] Paul Pop, Michael Lander Raagaard, Marina Gutierrez, and Wilfried Steiner. "Enabling fog computing for industrial automation through time-sensitive networking (TSN)". In: *IEEE Communications Standards Magazine* 2.2 (2018), pp. 55–61.

[143] Martin Schoeberl and Rasmus Ulslev Pedersen. "tpIP: A Time-Predictable TCP/IP Stack for Cyber-Physical Systems". In: *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE. 2018, pp. 75–82.

[144] Martin Schoeberl, Luca Pezzarossa, and Jens Sparsø. "A Multicore Processor for Time-Critical Applications". In: *IEEE Design Test* 35 (2018), pp. 38–47. ISSN: 2168-2356. DOI: 10.1109/MDAT.2018.2791809.

[145] Martin Schoeberl, Wolfgang Puffitsch, Stefan Hepp, Benedikt Huber, and Daniel Prokesch. "Patmos: A Time-predictable Microprocessor". In: *Real-Time Systems* 54(2) (Apr. 2018), pp. 389–423. ISSN: 1573-1383. DOI: 10.1007/s11241-018-9300-4.

[146] Wilfried Steiner, Günther Bauer, Brendan Hall, and Michael Paulitsch. "Time-triggered Ethernet". In: *Time-Triggered Communication*. CRC Press, 2018. Chap. 8, pp. 209–248.

[147] Lin Zhao, Feng He, Ershuai Li, and Jun Lu. "Comparison of Time Sensitive Networking (TSN) and TTEthernet". In: *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*. IEEE. 2018, pp. 1–7.

[148] Luxi Zhao, Paul Pop, and Silviu S Craciunas. "Worst-case latency analysis for IEEE 802.1 Qbv time sensitive networks using network calculus". In: *Ieee Access* 6 (2018), pp. 41803–41815.

[149] Lucia Lo Bello and Wilfried Steiner. "A perspective on IEEE time-sensitive networking for industrial communication and automation systems". In: *Proceedings of the IEEE* 107.6 (2019), pp. 1094–1120.

[150] Huang Chen, Lide Wang, Ping Shen, and Jun Di. "Static Schedule Generation for Time-Triggered Ethernet Based on Fuzzy Particle Swarm Optimization". In: *Chinese Journal of Electronics* 28.6 (2019), pp. 1250–1258.

[151] Casimer DeCusatis, Robert M Lynch, William Kluge, John Houston, Paul Wojciak, and Steve Guendert. "Impact of Cyberattacks on Precision Time Protocol". In: *IEEE Transactions on Instrumentation and Measurement* (2019).

[152] Eleftherios Kyriakakis, Jens Sparsø, and Martin Schoeberl. "InterNoC: Unified Deterministic Communication For Distributed NoC-based Many-Core". In: *th Junior Researcher Workshop on Real-Time Computing*. 2019.

[153] Maja Lund, Luca Pezzarossa, Jens Sparsø, and Martin Schoeberl. "A Time-predictable TTEthernet Node". In: *2019 IEEE 22nd International Symposium on Real-Time Computing (ISORC)*. May 2019, pp. 229–233. DOI: 10.1109/ISORC.2019.00048.

[154] Maryam Pahlevan, Nadra Tabassam, and Roman Obermaisser. "Heuristic list scheduler for time triggered traffic in time sensitive networks". In: *ACM Sigbed Review* 16.1 (2019), pp. 15–20.

[155] Tórur Biskopstø Strøm, Jens Sparsø, and Martin Schoeberl. "Hardlock: Real-time multicore locking". In: *Journal of Systems Architecture* 97 (2019), pp. 467–476.

[156] Robert Wittig, Friedrich Pauls, Emil Matus, and Gerhard Fettweis. "Access Interval Prediction for Tightly Coupled Memory Systems". In: *International Conference on Embedded Computer Systems*. Springer. 2019, pp. 229–240.

[157] Mohammadreza Barzegaran, Anton Cervin, and Paul Pop. "Performance optimization of control applications on fog computing platforms using scheduling and isolation". In: *IEEE Access* 8 (2020), pp. 104085–104098.

[158] Simulink Documentation. *Simulation and Model-Based Design*. 2020. URL: https://www.mathworks.com/products/simulink.html.

[159] Owais Hamid and Sayyid Anas Vaqar. "A comparison of Distributed Data Communications using Ethernet in Aircraft". In: (2020).

[160] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. eng. 2020. DOI: 10.1109/IEEESTD.2020.9120376.

[161] Eleftherios Kyriakakis, Maja Lund, Luca Pezzarossa, Jens Sparsø, and Martin Schoeberl. "A Time-predictable Open-Source TTEthernet End-System". In: *Journal of Systems Architecture* (2020), p. 101744.

[162] Eleftherios Kyriakakis, Jens Sparsø, Peter Puschner, and Martin 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.

[163] Peter Puschner and Raimund Kirner. "Asynchronous vs. synchronous interfacing to time-triggered communication systems". In: *Journal of Systems Architecture* 103 (2020), p. 101690.

[164] Eleftherios Kyriakakis, Jens Sparsø, Peter Puschner, and Martin Schoeberl. "Synchronizing Real-Time Tasks in Time-Triggered Networks". In: *24th International Symposium On Real-Time Distributed Computing (ISORC)*. IEEE. 2021.

[165] Anna Minaeva and Zdeněk Hanzálek. "Survey on Periodic Scheduling for Time-triggered Hard Real-time Systems". In: *ACM Computing Surveys (CSUR)* 54.1 (2021), pp. 1–32.

[166] TTTech. *TTETools - TTEthernet Development Tools v4.4*. URL: https://www.tttech.com/products/aerospace/development-test-vv/development-tools/tte-plan/ (visited on 07/05/2018).