The Time-Triggered Wireless Architecture

Romain Jacob
ETH Zurich, Switzerland

Licong Zhang
TU Munich, Germany

Marco Zimmerling
TU Dresden, Germany

Jan Beutel
ETH Zurich, Switzerland

Samarjit Chakraborty
University of North Carolina at Chapel Hill, NC, United States

Lothar Thiele
ETH Zurich, Switzerland

Abstract

Wirelessly interconnected sensors, actuators, and controllers promise greater flexibility, lower installation and maintenance costs, and higher robustness in harsh conditions than wired solutions. However, to facilitate the adoption of wireless communication in cyber-physical systems (CPS), the functional and non-functional properties must be similar to those known from wired architectures. We thus present Time-Triggered Wireless (TTW), a wireless architecture for multi-mode CPS that offers reliable communication with guarantees on end-to-end delays among distributed applications executing on low-cost, low-power embedded devices. We achieve this by exploiting the high reliability and deterministic behavior of a synchronous transmission based communication stack we design, and by coupling the timings of distributed task executions and message exchanges across the wireless network by solving a novel co-scheduling problem. While some of the concepts in TTW have existed for some time and TTW has already been successfully applied for feedback control and coordination of multiple mechanical systems with closed-loop stability guarantees, this paper presents the key algorithmic, scheduling, and networking mechanisms behind TTW, along with their experimental evaluation, which have not been known so far. TTW is open source and ready to use: `https://ttw.ethz.ch`.

2012 ACM Subject Classification Computer systems organization → Real-time system architecture; Computer systems organization → Sensors and actuators; Networks → Sensor networks

Keywords and phrases Time-triggered architecture, wireless bus, synchronous transmissions

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2020.19

Related Version Prior 4-page publication [25]: `https://doi.org/10.23919/DATE.2018.8342127`

Supplementary Material ECRTS 2020 Artifact Evaluation approved artifact available at `https://doi.org/10.4230/DARTS.6.1.5`.
Project webpage – TTW Artifacts – Authors' Contributions (CRediT)

Funding *Marco Zimmerling*: German Research Foundation (DFG) within the Emmy Noether project NextIoT (grant ZI 1635/2-1)
Lothar Thiele: Swiss National Science Foundation program "NCCR Automation"

1 Introduction

For decades, real-time distributed control systems have mostly relied on fieldbuses like CAN, FlexRay, and PROFIBUS. Following the design principles of the Time-Triggered Archi-

tecture (TTA) [31], these systems provide predictability of functional and non-functional properties. Yet, wired systems are increasingly reaching their limits as future cyber-physical systems (CPS) demand higher flexibility and cost efficiency. Low-power wireless communication promises to meet these demands by allowing for an unprecedented degree of mobility and deployment flexibility, avoiding cable breaks or faulty connections, and providing greater robustness to heat, humidity, abrasive substances, and undamped vibrations. However, to be viable for mission-critical CPS, a wireless system must feature timing predictability similar to traditional wired systems. Moreover, short end-to-end delays (tens of milliseconds), high reliability, and multiple years of battery lifetime are required for many applications [3].

The past years have seen significant progress in this direction. In particular, the concept of *synchronous transmissions* [14] has enabled highly reliable and efficient low-power wireless communication protocols that are robust to the unpredictable dynamics of wireless systems. As further detailed in [50], this is because of two main reasons: (*i*) synchronous transmissions enable the design of protocols whose logic is *independent of the time-varying network state*, which leads to a highly deterministic protocol execution regardless of changes in the network; (*ii*) synchronous-transmission-based communication protocols inherently exploit *different forms of diversity*, including sender and receiver diversity as well as temporal and spatial diversity, leading to a reliability as high as 99.9999 % in certain scenarios [14]. In fact, since its inception in 2016, all top three teams at the annual EWSN Dependability Competition have built on synchronous transmissions, demonstrating dependable multi-hop communication even in extreme wireless interference scenarios [40]. Building on this dependable base, protocols have been designed that provide sub-microsecond network-wide time synchronization while abstracting from the complexity of the underlying dynamic network topology, thus allowing to reason about a low-power wireless network as if it were a shared bus [13].

In light of this recent progress, we ask: *Is it possible to design a reliable, adaptive, and efficient time-triggered architecture for low-power wireless multi-hop networks with formal guarantees on end-to-end delays among distributed application tasks?* If so, it would take wireless systems a decisive step closer to wired systems by providing similar abstractions and guarantees as those researchers, engineers, and operators are used to, thus enabling powerful new CPS applications in industry, healthcare, energy, and many other domains.

Challenges. There are four key technical challenges standing in the way of a time-triggered architecture for low-power wireless multi-hop networks.

1. *Real-time communication in the face of radio duty cycling.* In a fieldbus, a node can listen idly without incurring costs, allowing it to react immediately to a request. In a low-power wireless bus [13], instead, a node must turn its radio off whenever possible to save energy, which renders the node unreachable until the next scheduled wake-up. To reduce energy costs due to idle listening, wireless protocols schedule communication rounds, where all nodes wake up, exchange messages, and go back to sleep (e.g., [13, 20, 22, 44]). Deciding when rounds take place and which nodes can send messages in each round to meet real-time deadlines at minimum energy costs is a complex scheduling problem; for example, the allocation of messages to rounds resembles combinatorial NP-hard bin packing [33].
2. *Coupling of communication and application tasks.* What ultimately matters in a CPS are the real-time constraints among application tasks executing on distributed devices. Hence, task executions and message exchanges over the wireless network must be coupled to guarantee end-to-end deadlines. A common approach for wired fieldbuses is to statically co-schedule all tasks and messages [2, 5, 12] to minimize delays by solving a satisfiability modulo theories (SMT) [11, 19, 41] or a mixed integer linear programming (MILP) [6]

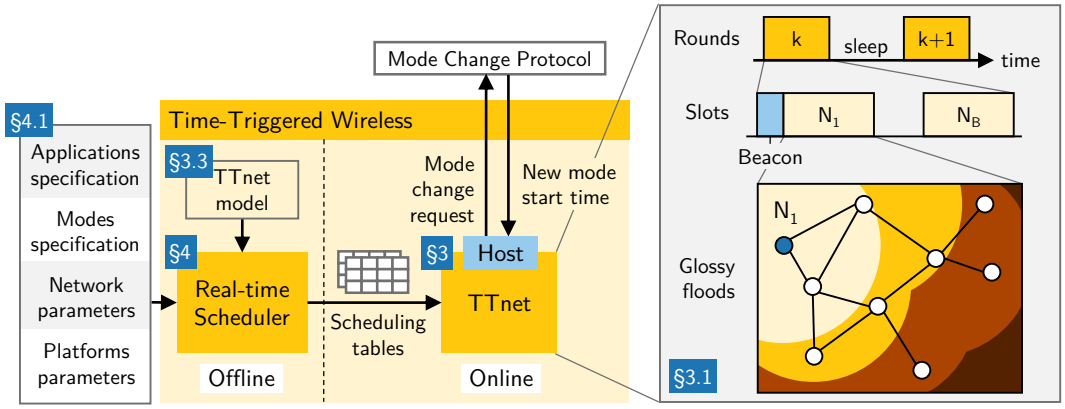


Figure 1 High-level overview of the Time-Triggered Wireless (TTW) architecture.

problem. To apply this approach to a wireless bus, we must embed the above-mentioned bin-packing problem into the schedule synthesis. This, however, introduces non-linear constraints that cannot be easily handled in a SMT or MILP formulation (see § 4.3).

3. *Dependencies across modes.* Static co-scheduling lacks runtime adaptability. This is often mitigated by enabling the system to switch at runtime between multiple operation modes, each having a pre-computed scheduling table [16]. The real-time guarantees that can be provided across mode changes depend on the mode-change protocol [9] and how the modes are scheduled. For example, if the periodicity of a task should be preserved across a mode change, this task must have the same schedule in both modes. A naïve approach to handle such dependencies across modes is a single MILP formulation with a global objective function. However, the poor scalability of the scheduling problem (NP-hard [27]) becomes a bottleneck for any realistic CPS scenario with many modes, while holding no guarantee that the computed schedules perform efficiently in terms of energy.
4. *Worst-case communication time.* The synthesis of scheduling tables requires an accurate model of the timing of all operations, including the worst-case execution time of tasks *and* the worst-case communication time of messages. While staple methods exist for tasks [45], a predictable protocol implementation is needed that yields accurate upper bounds on the time required for exchanging messages across a multi-hop network.

Contributions. This paper presents Time-Triggered Wireless (TTW), a wireless architecture for multi-mode CPS. By addressing all of the above challenges, TTW provides: (*i*) highly reliable and efficient communication across dynamic low-power wireless multi-hop networks; (*ii*) formal guarantees on end-to-end delays among distributed application tasks; (*iii*) runtime adaptability through mode changes that respect the tasks' periodicity constraints. With this, TTW does not only significantly advance the state of the art in real-time wireless systems, but also represents a solid foundation for the design of dependable wireless CPS.

As shown in Figure 1, TTW consists of two main components: a system-wide real-time scheduler that executes offline and a communication stack called TTnet that runs online on distributed low-power wireless devices. Based on the application specification (e.g., tasks, messages, modes) and system parameters (e.g., number of nodes, message sizes), the scheduler synthesizes optimized scheduling tables for the entire system. These tables are loaded onto the devices, and at runtime each device follows the table that corresponds to the current mode. TTnet provides a generic interface to implement different mode-change protocols [9].

Our design of the real-time scheduler uses concepts from network calculus [34] and novel heuristics to addresses challenges **1.**–**3.** Further, our design of TTnet addresses challenge **4.** by exploiting synchronous transmissions, in particular the deterministic behavior of Glossy floods [14]. As indicated for node N_1 in the bottom right corner of Figure 1, a Glossy flood sends a message to all nodes in a multi-hop network with a reliability that can exceed 99.9 % in certain networks and challenging conditions [13, 40], yet the time this process takes can be confined to a time slot of known length. TTnet groups a series of such time slots into rounds to save energy. This yields a highly timing-predictable protocol execution for which we devise timing and energy models as input for the TTW real-time scheduler.

We implement TTW on physical platforms and open-source our implementation [24]. Using this implementation, we perform real-world experiments on 27 low-power wireless nodes of the FlockLab testbed [35]. The results demonstrate that, for the settings we tested, our models are highly accurate and provide accurate upper bounds for the schedule synthesis; for example, our timing model overestimate the length of a round in TTnet by at most 0.7 ms. The results also show that our scheduling techniques can effectively reduce the energy cost of wireless communication in TTW and make the scheduling problem tractable: the solving time for one mode on a standard laptop PC ranges from a few seconds to a few minutes depending on the complexity of the mode (e.g., number of tasks and messages).

In addition, TTW already proved its utility for practical wireless CPS representative of emerging applications, such as remote control in chemical plants and cooperative robotics in smart manufacturing. Specifically, TTW was essential in enabling fast and reliable feedback control and coordination of multiple physical systems over low-power wireless multi-hop networks with closed-loop stability guarantees despite mode changes and mobile nodes [7, 36].[1] In these works, we integrated TTW with a mode-change protocol and analyzed the worst-case end-to-end jitter; thanks to the predictability of TTW, this jitter can be made negligible ($\leq 50\,\mu$s) for typical CPS applications, which we empirically validated (see [7, 36] for details).

Difference to prior paper. This paper significantly extends a prior 4-page publication [25] by (*i*) incorporating mode changes in the system model and the formulation of the scheduling problem, (*ii*) providing an implementation of TTW on physical platforms, and (*iii*) using this implementation to evaluate TTW and validate our models through real-world experiments.

2 Related Work

TTW is most closely related to prior work on reliable and predictable solutions for wireless sensor-actuator networks and real-ime distributed control systems based on fieldbuses.

In the wireless domain, numerous standards and protocols have been proposed for low-power multi-hop wireless networks, including WirelessHART, ISA100, and TSCH [44] from industry and several proposals from academia (e.g., [10, 39, 18]). Closest to TTW is Blink [49], which also builds on the concept of synchronous transmissions, in particular the Low-Power Wireless Bus (LWB) [13], to achieve adaptive real-time communication with guarantees on packet deadlines. While certainly important and useful, the key difference to TTW is that all these solutions consider only the network resources. They do not take into account the scheduling of application tasks on the distributed devices, and can therefore not influence the end-to-end delays and jitter that ultimately matter from an application's perspective. The

[1] Public demo: `https://youtu.be/AtULmfGkVCE`.
Mobility experiment: `https://youtu.be/19xPHjnobkY`.

only prior work that has looked at this end-to-end problem is DRP [26]. To provide end-to-end guarantees on packet deadlines, DRP decouples tasks and messages as much as possible, aiming for efficient support of sporadic or event-triggered applications. Compared with TTW, which tightly couples tasks and messages by co-scheduling them, DRP incurs significantly higher worst-case delays, and is thus not suitable for demanding CPS applications [3].

In the wired domain, a lot of work has been done on time-triggered architectures, such as the Time-Triggered Protocol (TTP) [32], the static-segment of FlexRay [15], and Time-Triggered Ethernet [30]. Like TTW, many recent works in this area also use SMT or MILP based methods to synthesize and/or analyze static (co-)schedules for those architectures [5, 12, 41, 43, 46]. The key difference to TTW, however, is that these approaches assume that a message can be scheduled at any point in time. While this is a perfectly valid assumption for a wired system, it is not compatible with the use of communication rounds in a wireless setting. § 5 shows that using rounds greatly reduces the energy consumed for communication, but it makes the schedule synthesis more complex, as detailed in § 4.

3 Communication Stack

We introduce TTW, a time-triggered architecture that supports the design and implementation of dependable wireless CPS. This section presents the design, implementation, and analytical models of TTW's communication stack called TTnet. TTW's real-time scheduler, described in § 4, uses these models to synthesize optimized scheduling tables.

3.1 TTnet Design

To support a wide spectrum of CPS applications possibly involving control and coordination of multiple physical systems over large distances, TTW requires a low-power wireless stack that provides reliable many-to-all communication over multiple hops. Further, the communication delays must be as short as possible and tightly bounded to support fast physical systems (e.g., mechanical systems with dynamics of tens of milliseconds). Finally, network-wide time synchronization is essential to reduce delays and achieve high performance and efficiency.

TTnet addresses these requirements by taking inspiration from the Low-Power Wireless Bus (LWB) [13] and improving latency and efficiency by using a different scheduling strategy. The basic communication scheme is illustrated on the right side of Figure 1, where distributed nodes are wirelessly interconnected and communicate using a sequence of Glossy floods [14]. As shown for when node N_1 sends its message, the flood spreads like a wave through the multi-hop network (note the different colors) so all nodes in the network can receive the message. During the flood, nodes that receive the message at the same time also retransmit the message at the same time. This technique, known as synchronous transmissions, is both fast (it achieves the theoretical minimum latency for one-to-all flooding) and extremely reliable [50]. Theoretical and empirical studies have shown that the few messages losses that do occur due to the vagaries of wireless communication are largely decorrelated, which greatly simplifies analytical modeling and control design [29, 48]. During a Glossy flood, the nodes also time-synchronize themselves with sub-microsecond accuracy [14]. Finally, since the protocol logic is independent of the network topology, it is highly robust to changes inside the network (e.g., due to mobile or failing devices) and external interference.

As shown in Figure 1, TTnet performs multiple Glossy floods in consecutive slots within a round; between two rounds all nodes have their wireless radio turned off to conserve energy. The duration of the slots and rounds as well as their associated energy costs can be accurately modeled (§ 3.3). Which nodes get to send their messages in each slot is determined by

TTW's real-time scheduler. The scheduler can abstract the entire multi-hop network with all its complexities and dynamics as a single shared resource with a known bandwidth, just like a wired fieldbus or a uniprocessor. This is because TTnet nodes appear to share a common clock and every messages is distributed to all nodes irrespective of the network topology.

TTnet exploits the fact that the schedules are computed offline to minimize communication delays. Instead of letting a dedicated host node compute the schedules between rounds and distribute them at the beginning of each round as in LWB, the host in TTnet only needs to send the ID of the current round in a short beacon (see Figure 1). The nodes can then use the ID to look up the corresponding scheduling table in their local memory. As a result, the rounds in TTnet are "more compact," which minimizes delays and improves efficiency.

The beacons also contain all information required to reliably switch between different operation modes at runtime. In addition to the round ID, each beacon carries a mode ID and a trigger bit. Modes and rounds have unique IDs with a known mapping of rounds to modes. By default, a beacon includes the current mode ID and the trigger bit is set to 0. A mode change happens in two phases. First, the mode change is announced by a beacon with the new mode ID. In a later round, the trigger bit is set to 1, which triggers the mode change; the new mode starts at the end of that round. This two-step procedure lets the nodes prepare for an upcoming mode change; for example, by stopping the execution of application tasks that are not present in the new mode. Hence, TTnet provides a generic interface to implement different mode-change protocols. The mode-change protocol itself is independent of TTnet and can be freely designed or chosen among existing protocols [9].

Sending beacons in every round ensures a safe operation of TTnet: If a node fails to receive one, it does not participate in the communication until it receives again a beacon. Thus, it is guaranteed that all participating nodes know the current round and mode IDs.

3.2 TTnet Implementation

We implement TTnet using Baloo [22], a design framework for communication stacks based on synchronous transmissions. Baloo provides a customizable round structure and allows the user to implement the high-level logic of a communication stack via a programming interface based on callback functions. Low-level operations such as time synchronization and radio control are handled by a middleware layer integrated in Baloo.

Our TTnet implementation uses the static configuration mode of Baloo. The beacons are implemented using the user-bytes field of Baloo's control packets with a payload size of 2 bytes. The implementation runs on any physical platform supported by Baloo. Because of the need for timing predictability in TTW, we focus on a platform that is based on the dual-processor platform (DPP) architecture [8]. The DPP combines two arbitrary processors with an interconnect called Bolt [42]. Bolt provides predictable asynchronous message passing between two processors using message queues with first-in-first-out (FIFO) semantics, one for each direction. This allows to dedicate each processor to a specific task, execute these tasks in parallel, and compared to traditional dual- or multi-core platforms the DPP offers formally verified bounds on the interference between the processors [42]. The specific platform we use is composed of a 32-bit ARM Cortex-M4 based MSP432P401R running at 48 MHz and a 16-bit CC430F5147 running at 13 MHz. The former is dedicated to the execution of application tasks (e.g., sensing, actuation, and control), while the latter is dedicated to TTnet using a low-power wireless radio operating at 250 kbps in the 868 MHz frequency band. Our implementation is open source and freely available; refer to [24] for details.

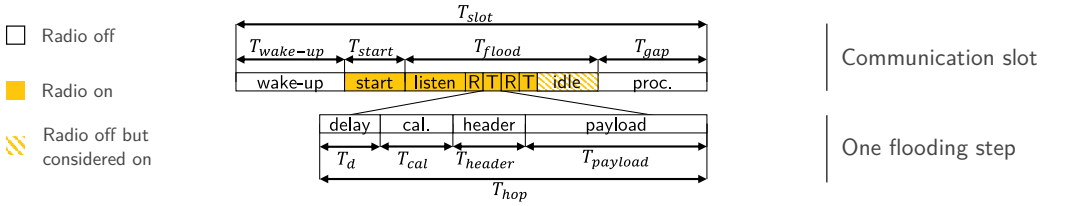


Figure 2 Detailed timings during a slot in TTnet. *At the slot level, the colored boxes identify phases where the radio is on. In the idle phase, the radio is off, but the duration depends on a node's distance to the initiator of the Glossy flood. To estimate the energy savings of rounds, we make the pessimistic assumption that the radio is on for T_{flood}, as specified in Equation* (7).

3.3 TTnet Model

The real-time scheduler in TTW requires timing and energy models of TTnet's operation to synthesize the scheduling tables. These models must be tight bounds to avoid delays and energy costs.

Let T_r be the length of a TTnet round. A round consists of a beacon slot plus up to B_{max} regular slots, in which Glossy floods are executed (see Figure 1). $T_r(L, B)$ denotes the length of a round with B regular slots having the same payload size L. The structure of a slot is illustrated in Figure 2. Accordingly, the length of a slot T_{slot} can be decomposed as follows

$$T_{slot} = T_{wake-up} + T_{start} + T_{flood} + T_{gap} \tag{1}$$

First, the nodes wake up ($T_{wake-up}$) and switch on their radio (T_{start}). Then, the Glossy flood executes (T_{flood}) followed by a small gap time (T_{gap}) in which the nodes can process the received packet. As explained in [47], the total length of a flood T_{flood} can be expressed by

$$T_{flood} = (H + 2N - 1) * T_{hop} \tag{2}$$

where H is the network diameter and N is the maximum number of times a node transmits during a Glossy flood. Let T_{hop} be the time needed for one protocol step, that is, the duration of one (synchronous) transmission during a flood. This quantity can be decomposed as

$$T_{hop} = T_d + T_{cal} + T_{header} + T_{payload} \tag{3}$$

where T_d is an initial radio delay, T_{cal} is the time needed to calibrate the radio clock (taking time equal to the transmission of L_{cal} bytes), and T_{header} and $T_{payload}$ are the times needed to transmit the packet header consisting of L_{header} bytes and the message payload consisting of L bytes. Using a radio with a transmit bitrate of R_{bit}, the transmission of L bytes takes

$$T(L) = 8L/R_{bit} \tag{4}$$

We divide the length of a slot T_{slot} into T^{on} and T^{off}, the times spent with the radio on and off, respectively. To this end, we make the conservative assumption that the radio stays on for the entirety of T_{flood}, as illustrated in Figure 2 and explained in the caption.

$$T_{slot}(L) = T^{off} + T^{on}(L) \tag{5}$$

$$T^{on}(L) = T_{start} + (H + 2N - 1) * (T_d + 8(L_{cal} + L_{header} + L)/R_{bit}) \tag{6}$$

$$T^{off} = T_{wake-up} + T_{gap} \tag{7}$$

With this, the length of a round with B regular slots and a common payload size L is

$$T_r(L, B) = T_{slot}(L_{beacon}) + B * T_{slot}(L) + T_{preprocess} \tag{8}$$

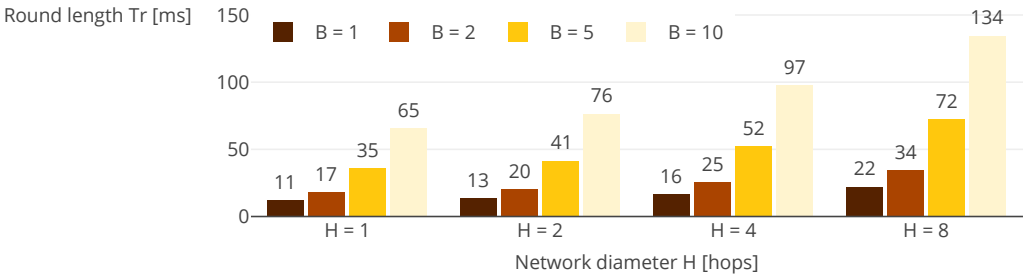


Figure 3 Example values of round length computed using the TTnet model for a payload size of 16 bytes and $N = 2$ transmissions during a Glossy flood. *Fewer slots lead to shorter rounds (e.g., 52 ms for $B = 5$ slots across a 4-hop network), which ultimately allows for shorter end-to-end delays.*

where L_{beacon} is the payload size of a beacon and $T_{preprocess}$ is the time needed by our TTnet implementation to prepare for an upcoming round (e.g., retrieve messages from send queue).

In addition to the timings of a TTnet round, we derive a model for the relative energy savings obtained by grouping multiple slots into the same round. As discussed in § 3.1, sending beacons is necessary to ensure a safe protocol operation. Without rounds, each regular message must still be preceded by a beacon to provide the same guarantee. In this case, the transmission time $T_{wo/r}$ for B regular messages of size L is given by

$$T_{wo/r}(L, B) = B * (T_{slot}(L_{beacon}) + T_{slot}(L)) \tag{9}$$

The relative energy savings obtained by a round-based design thus amount to

$$E = (T^{on}_{wo/r} - T^{on}_{r})/T^{on}_{wo/r} \tag{10}$$

with $T^{on}_{wo/r} = B * (T^{on}(L_{beacon}) + T^{on}(L))$ and $T^{on}_{r} = T^{on}(L_{beacon}) + B * T^{on}(L)$.

Using these expressions and the parameters measured for or used by our TTnet implementation (detailed in [24]), we can determine the round length T_r and the energy savings E for different number of slots per round B, message payload sizes L, network diameters H, and number of transmissions during a Glossy flood (N). For $L = 16$ byte and $N = 2$, Figure 3 plots the resulting round length T_r for different network diameters and slots per round. We can see, for example, that it takes less than 100 ms to send 10 messages in a 4-hop network. Real-world experiments using 27 distributed wireless nodes on a public testbed demonstrate that these models are very accurate (§ 5.1).

4 Real-Time Scheduler

Equipped with a timing-predictable implementation of the TTnet communication stack and a corresponding model that provides accurate upper bounds, we now turn to the second main component of TTW: the real-time scheduler. This section first defines the system model and scheduling problem, then describes the single-mode and multi-mode schedule synthesis.

4.1 System Model

Before we can formulate and solve the scheduling problem we face in TTW, we provide a formal system model of all relevant hardware and software components and their interactions.

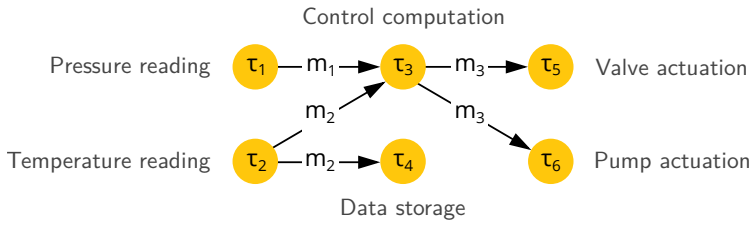


Figure 4 An example application and its precedence graph $\mathbb{P}$. *The application execution starts with sensing tasks τ_1 and τ_2. After messages m_1 and m_2 with the sensor readings are received by the controller, actuation values are computed (τ_3), sent to the actuators (m_3), and applied (τ_5 and τ_6).*

Platform. Each node in the system is a *platform* that executes tasks and exchanges messages using TTnet. All tasks have a known worst-case execution time (WCET) on that platform. Targeting state-of-the-art wireless CPS platforms with two processors (e.g., LPC541XX, VF3xxR, DPP [8]), each node can simultaneously execute tasks and exchange messages.

Applications. Let $\mathcal{A}$ denote the set of distributed *applications* in the system. An application is composed of *tasks* and *messages*, each with a unique task or message *id*, that are coupled by precedence constraints as described by a directed acyclic graph, where vertices and edges represent tasks and messages, respectively. We denote by $a.\mathbb{P}$ the *precedence graph* of application a (see Figure 4 for an example). Each application executes at a periodic interval $a.p$ called the *period*. An application execution is completed when all tasks in $\mathbb{P}$ have been executed. All tasks and messages in $a.\mathbb{P}$ share the same period $a.p$. If the same task belongs to applications with different periods, it can be equivalently modeled as two different tasks. Applications are subject to real-time constraints: an application *relative deadline*, denoted by $a.d$, represents the maximum tolerable *end-to-end delay* to complete the application execution. The deadline can be arbitrary, without a specific relation to the period $a.p$. Some applications may require to keep the same schedule (e.g., the same task offsets) when switching between different operation modes, for example, to guarantee the stability of control loops under arbitrary mode switches [7]. We call these *persistent applications* and denote them by $\mathcal{A}_P \subset \mathcal{A}$.

Tasks. We denote by $\mathcal{T}$ the set of all *tasks* in the system. A node executes at most one task at any point in time. Since interactions with the physical world like sensing and actuation should not be interrupted, we consider non-preemptive task scheduling. Each task τ is mapped to a given node $\tau.map$ on which it executes with WCET $\tau.e$. The *task offset* $\tau.o$ represents the start of the task execution relative to the beginning of the application execution. A task can have an arbitrary number of preceding messages, which must all be received before the task can start to execute. $\tau.prec$ denotes the set of preceding message *id*s. Within one application, each task is unique; however, the same task may belong to multiple applications (e.g., the same sensing task may source different feedback loops). If so, we consider that these applications have the same period.

Messages. Let $\mathcal{M}$ be the set of all *messages*. Every message m has at least one preceding task, that is, a task that needs to finish before the message can be transmitted. The set of preceding task *id*s is denoted by $m.prec$. The *message offset* $m.o$ relative to the beginning of the application execution represents the earliest time message m can be allocated to a TTnet round for transmission (i.e., after all preceding tasks are completed). The *message deadline*

Table 1 Inputs and outputs of the scheduling problem we solve in TTW.

Inputs	$\mathcal{N}$	Set of nodes in the system
	$\mathcal{A}$, $\mathcal{A}_P$	Set of applications and persistent applications (including periods, deadlines, and precedence graphs)
	$\mathcal{O}$	Set of operation modes (including mode priorities)
	$\mathbb{M}$	Mode graph
	$\tau.p$, $m.p$	Task and message periods, inherited from the application
	$\tau.map$	Mappings of tasks to nodes
	$\tau.e$	Task worst-case execution time (WCET) given $\tau.map$
	B_{max}	Maximum number of slots per round
	L_{max}	Maximum message payload size
	H	Network diameter (in number of hops)
	N	Number of transmissions in a Glossy flood [14]
Outputs	$\tau.o$	Task offsets
	$m.o$, $m.d$	Message offsets and deadlines
	$r.t$, $r.[B_{max}]$	Round starting times and allocation vectors

$m.d$ relative to the message offset represents the latest time the message transmission must be completed (i.e., the earliest offset of subsequent tasks). The payload of all messages is upper-bounded by L_{max}. Messages are not necessarily unique: multiple edges of $a.\mathbb{P}$ can be labeled with the same message m, which captures the case of multicast or broadcast transmissions (see Figure 4). If a message belongs to multiple applications, we consider that these applications have the same period.

Modes. We denote with $\mathcal{O}$ the set of operation *modes*. These modes represent mutually exclusive phases in the system execution (e.g., init, normal, and failure modes), each having its own schedule. A mode has a unique *priority* $M.prio$. We write $a \in M$ to denote that application a executes in mode M. When unambiguous, we use M to denote the set of all applications that execute in mode M. The *hyperperiod* LCM of mode M is the least common multiple of the periods of all applications in M. Possible transitions between modes at runtime are specified by the *mode graph* $\mathbb{M}$ (see Figure 6). The mode graph is undirectional, that is, a transition from M_i to M_j implies that it is also possible to switch from M_j to M_i.

Network and rounds. We denote with $\mathcal{N}$ the set of all *nodes* in the network. The following four network parameters must be specified by the user of the TTW architecture:

- L the payload size of regular messages, in bytes;
- B_{max} the maximum number of slots in a TTnet round;
- N the maximum number of transmissions of a node during a Glossy flood;
- H the estimated diameter of the network, in number of hops.

As explained in § 3, communication over the network is scheduled in *rounds* r. The schedule of mode M has R_M rounds. We consider TTnet rounds as atomic, that is, they cannot be interrupted. Thus, the ordering of messages in a round is irrelevant.

Round r contains B_r slots (at most B_{max}), each of which is allocated to a unique message m. The *allocation vector* $r.[B_r]$ contains the *id*s of the messages that are allocated to the slots in round r, where $r.B_s$ denotes the allocation of the s-th slot. The *starting time* $r.t$ is the start of the round relative to the beginning of the mode's hyperperiod. Using the models from § 3.3, the length of a round and its energy cost can be determined.

Algorithm 1 Pseudo-code of the single-mode schedule synthesis.

Input: mode M, $a \in M$, $\tau.map$, $\tau.e$, B_{max}, T_o
Output: $Sched(M)$
  $LCM = hyperperiod(M)$
  $R_{max} = floor(LCM/T_o)$
  $R_M = 0$
  while $R_M \leq R_{max}$ **do**
    formulate the MILP for mode M using R_M rounds
    ($Sched(M)$, *feasible*) = *solve*(MILP)
    if *feasible* **then return** $Sched(M)$
    end if
    $R_M = R_M + 1$
  end while
  return 'Problem infeasible'

4.2 Scheduling Problem

Based on our system model, we are now in the position to formally state the scheduling problem we have to solve in TTW. Given all modes, applications, task-to-node mappings, and WCETs, for each mode M the remaining variables define the *mode schedule* $Sched(M)$

$$Sched(M) = \left\{ \begin{array}{l|l} \tau.o,\ m.o,\ m.d & \forall\ a \in M,\ (\tau, m) \in a.\mathbb{P} \\ r_k.t,\ r_k.[B_{max}] & \forall\ k \in [1,\ R_M] \end{array} \right\}$$

Table 1 lists all inputs and outputs of the scheduling problem in TTW. A schedule for mode M is said to be *valid* if all applications executing in mode M meet their end-to-end deadlines. The scheduling problem to solve thus consists of finding valid schedules for all modes $M \in \mathcal{O}$ such that the following two objectives are met:

(O1) The number of communication rounds is minimized, which results in minimizing the energy required for wireless communication.

(O2) All persistent applications $\mathcal{A}_P \subset \mathcal{A}$ seamlessly switch between modes, that is, their schedules remain the same across all possible mode changes.

4.3 Single-Mode Schedule Synthesis

TTW statically synthesizes the schedule of all tasks, messages, and communication rounds by solving a MILP. We first look at the single-mode case, effectively showing how to achieve objective **(O1)**, whereas § 4.4 considers the multi-mode case and thus objective **(O2)**.

The schedule of a mode M is computed for one hyperperiod, after which it repeats itself. To minimize the number of rounds used while taming computational complexity, we solve the problem sequentially as described in Algorithm 1. Each MILP formulation considers a fixed number of rounds R_M to be scheduled, starting with $R_M = 0$. The number of rounds is incremented until a feasible solution is found or the maximum number of rounds R_{max} that fit into one hyperperiod is reached. So, even without an explicit objective function, Algorithm 1 guarantees by construction that if the problem is feasible, the synthesized schedule uses the minimum number of rounds, thereby minimizing the communication energy costs.

Each MILP formulation contains a set of classical scheduling constraints such as: precedence constraints between tasks and messages must be respected, the applications' end-to-end deadlines must be satisfied, nodes process at most one task simultaneously, communication rounds must not overlap, and rounds must not be allocated more then B_{max} messages. These constraints can be easily formulated based on our system model. However, we must also guarantee that the allocation of messages to rounds is valid. Specifically,

(C1) messages must be served in rounds that start after their release time;

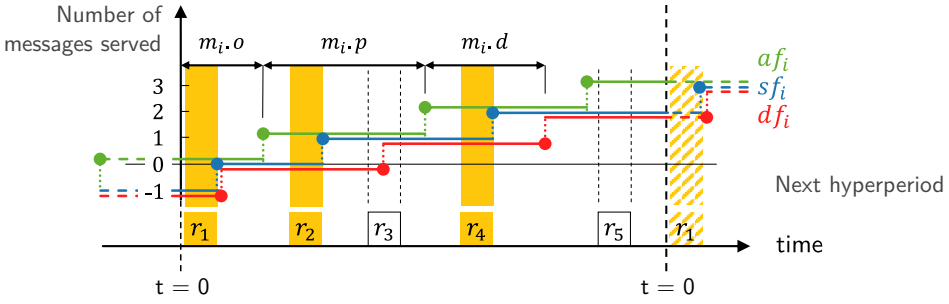


Figure 5 Example showing arrival (af), demand (df), and service (sf) functions for message m_i. *The lower part of the chart shows the five round, r_1 to r_5, that are scheduled within the hyperperiod. Message m_i is allocated a slot in the colored rounds, that is, in rounds r_1, r_2, and r_4. Allocating m_i to r_3 instead of r_2 would be invalid because r_3 does not finish before m_i's deadline, thus violating constraint* **(C2)**. *By contrast, allocating m_i to r_5 instead of r_1 would be valid and result in $r_0.B_i = 0$.*

(C2) messages must be served in rounds that finish before their deadline.

In other words, we must integrate the bin-packing problem that arises when allocating messages to rounds within the MILP formulation. This is a non-trivial challenge and a major difference compared with the existing approaches for wired architectures (e.g., [12]).

To address this challenge, we first formulate the constraints **(C1)** and **(C2)** using *arrival*, *demand*, and *service* functions – denoted by af, df, and sf– inspired by network calculus [34] (see Figure 5 for an illustration). These three functions count the number of message instances that are released, have passed deadlines, and have been served since the beginning of the hyperperiod. It must hold that

$$\forall\, m_i \in \mathcal{M},\ \forall\, t, \qquad df_i(t) \leq sf_i(t) \leq af_i(t) \tag{11}$$

$$\text{with} \qquad af_i :\ t \longmapsto \left\lfloor \frac{t - m_i.o}{m_i.p} \right\rfloor + 1 \tag{12}$$

$$\text{and} \qquad df_i :\ t \longmapsto \left\lceil \frac{t - m_i.o - m_i.d}{m_i.p} \right\rceil \tag{13}$$

However, as the service function sf stays constant between the rounds, we can formulate constraints **(C1)** and **(C2)** as follows: $\forall\, m_i \in \mathcal{M},\ \forall\, j \in [1..R_M]$,

$$\textbf{(C1)} \qquad sf_i(r_j.t + T_r) \ \leq\ af_i(r_j.t) \tag{14}$$

$$\textbf{(C2)} \qquad sf_i(r_j.t) \ \geq\ df_i(r_j.t + T_r) \tag{15}$$

The arrival and demand functions are step functions, which cannot be directly used in a MILP formulation. However, we observe that

$$\forall\, k \in \mathbb{N}, \qquad af_i(t) = k \quad \Leftrightarrow \quad 0 \leq t - m_i.o - (k-1) * m_i.p \ <\ m_i.p \tag{16}$$

$$\text{and} \qquad df_i(t) = k \quad \Leftrightarrow \quad 0 \ < \ t - m_i.o - m_i.d - (k-1) * m_i.p \leq\ m_i.p \tag{17}$$

This allows us to introduce, for each message $m_i \in \mathcal{M}$ and each round r_j, $j \in [1..R_M]$, two integer variables k^a_{ij} and k^d_{ij} that we constrain to take the values of af and df at the time points of interest, namely $r_j.t$ and $r_j.t + T_{r_j}$. More formally, we can write

$$0 \leq r_j.t - m_i.o - (k^a_{ij} - 1) * m_i.p \ <\ m_i.p \tag{18}$$

$$0 \ < \ r_j.t + T_{r_j} - m_i.o - m_i.d - (k^d_{ij} - 1) * m_i.p \leq\ m_i.p \tag{19}$$

$$\text{Thus,} \quad (18) \quad \Leftrightarrow \quad af_i(r_j.t) = k^a_{ij}$$

$$(19) \quad \Leftrightarrow \quad df_i(r_j.t + T_{r_j}) = k^d_{ij}$$

Finally, we must express the service function sf, which counts the number of message instances served by *the end* of each round. Recalling that $r_k.B_s$ denotes the *id* of the message that is allocated to the s-th slot of round r_k, we have that for any time $t \in [\, r_j.t + T_{r_j} \,;\, r_{j+1}.t + T_{r_j} \,[$ the number of instances of message m_i served is

$$\sum_{k=1}^{j} \sum_{s=1}^{B} r_k.B_s \quad s.t. \quad B_s = i$$

As visible in Figure 5, it can happen that $m.o + m.d > m.p$, resulting in $df(0) = -1$ according to Equation (13). This situation arises when a message is released at the end of a hyperperiod and thus has its deadline in the *next* hyperperiod. To account for this, we introduce for each message m_i a variable $r_0.B_i$ that is set to the number of such "leftover" message instances at $t = 0$. With this, for each $m_i \in \mathcal{M}$ and $t \in [\, r_j.t + T_{r_j} \,;\, r_{j+1}.t + T_{r_j} \,[$,

$$sf_i : t \longmapsto \sum_{\substack{k=1 \\ s.t.\ r_k.t + T_{r_k} < t}}^{j} \sum_{\substack{s=1 \\ s.t.\ B_s = i}}^{B} r_k.B_s - r_0.B_i \tag{20}$$

Based on the above reasoning, we can express constraints **(C1)** and **(C2)** in the MILP formulation using Equations (18) and (19) and the following two equations

$$(14) \quad \Leftrightarrow \quad \sum_{k=1}^{j} \sum_{\substack{s=1 \\ s.t.\ B_s = i}}^{B} r_k.B_s - r_0.B_i \;\leq\; k_{ij}^{a} \tag{21}$$

$$(15) \quad \Leftrightarrow \quad \sum_{k=1}^{j-1} \sum_{\substack{s=1 \\ s.t.\ B_s = i}}^{B} r_k.B_s - r_0.B_i \;\geq\; k_{ij}^{d} \tag{22}$$

4.4 Multi-Mode Schedule Synthesis

The multi-mode schedule synthesis is a multi-objective problem: As stated in § 4.2, the number of communication rounds in each mode should be minimized **(O1)**, while the schedule of all persistent applications should remain unchanged across mode changes **(O2)**. Objective **(O2)** induces dependencies between the different modes, which cannot be handled by solving a set of independent single-mode schedule synthesis problems.

A straightforward approach would be to synthesize the schedules of all modes at once based on a single MILP formulation. This approach, however, has two caveats. First, the resulting schedule synthesis problem is NP-hard [27] and thus scales poorly as the number of modes increases, which becomes a bottleneck for realistic CPS applications requiring a high degree of system adaptability. Second, a global objective function must be defined, such as minimizing the total number of rounds across all modes, which still gives no guarantee that the corresponding communication energy costs are effectively minimized: at runtime, if the system remains almost always in a certain mode, it may be better to minimize the number of rounds in that particular mode even if this implies a larger number of rounds overall.

To address these caveats, we solve the multi-mode schedule synthesis problem sequentially using heuristics, as commonly done in related approaches [28, 37, 41]. Algorithm 2 summarizes our approach. The modes are scheduled based on their priority $M.prio$. The mode with the highest priority (i.e., $M.prio = 1$) is scheduled first according to the single-mode synthesis outlined in Algorithm 1. Then, the mode with the second-highest priority (i.e., $M.prio = 2$)

Algorithm 2 Pseudo-code of the multi-mode schedule synthesis.

Input: Applications specification; modes specification; network parameters; system parameters
Output: { $Sched(M_i)$ for $M_i \in \mathcal{O}$ }

```
inheritance_constraints = ∅
for all M_i ∈ O in order of decreasing mode priority M_i.prio do
    add inheritance_constraints to mode M_i
    ( Sched(M_i) , feasible ) = single_mode_synthesis(M_i)
    if feasible then
        add Sched(M_i) to inheritance_constraints
    else
        return 'Problem infeasible'
    end if
end for
return { Sched(M_i) for M_i ∈ O }
```

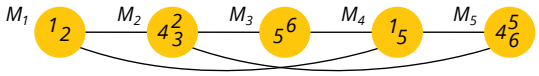


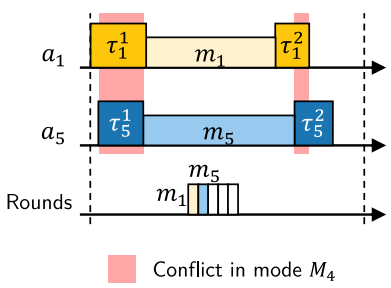
Figure 6 Mode graph $\mathbb{M}$ for Examples 1 and 6. *Five modes are depictd by circles and possible transitions between them by edges. Numbers inside each circle indicate which of the six applications, a_1 to a_6, execute in the corresponding mode, for example, $M_1 = \{a_1, a_2\}$. Mode M_i has priority i.*

is considered; however, its specification is first extended with *inheritance constraints* to guarantee that all persistent applications can seamlessly switch between all modes considered so far. The process repeats until all modes are scheduled.

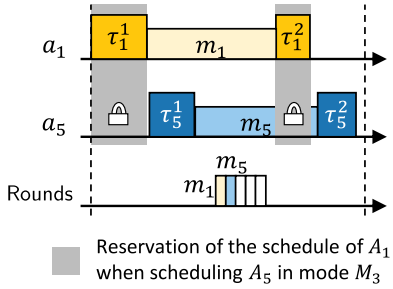
This sequential approach addresses the scalability of the multi-mode problem and provides a reasonable heuristic for minimizing the number of executed rounds and thus the communication energy costs. Indeed, these costs ultimately depend on how much time the system spends in each mode. It is reasonable to assume that an application domain expert knows in which modes the system likely operates most of the time. These modes are assigned a higher priority and therefore scheduled first by Algorithm 2. A mode with a lower priority is subject to more inheritance constraints, so it may schedule more than the minimal number of rounds to achieve **(O2)**. This may lead to a sub-optimal energy cost of lower-priority modes, which is nevertheless acceptable as the system likely spends less time in these modes.

In the following, we address the problem of deriving the set of inheritance constraints that are *necessary* and *sufficient* to guarantee objective **(O2)**. We first formalize the continuity constraints necessary to satisfy **(O2)** and characterize how these constraints may lead to conflicts between the mode schedules. Then we derive the minimally restrictive inheritance constraints that are necessary and sufficient to prevent such conflicts while satisfying **(O2)**.

▶ **Example 1.** Let us consider the mode graph in Figure 6 and assume that all applications are persistent. The modes are scheduled sequentially, starting with the highest-priority mode M_1, which is freely scheduled. When mode M_2 is scheduled, the schedule for application a_2 is inherited from mode M_1 to guarantee **(O2)** and the schedules for applications a_3 and a_4 are synthesized without constraints. In M_3, the specified applications a_5 and a_6 are new and can be scheduled without constraints. Then, in mode M_4, the specified applications a_1 and a_5 have been previously scheduled and thus must be inherited **(O2)**. However, as mode M_3 has been scheduled without constraint, the schedule synthesized for a_5 may be incompatible with that of a_1 from mode M_1. This leads to a conflict in M_4 and thus renders the sequential synthesis of the multi-mode problem infeasible, as illustrated in Figure 7.



(a) a_5 is scheduled in mode M_3 without considering the previously computed schedule of a_1, which leads to a conflict in mode M_4.



(b) a_5 is scheduled in mode M_3 considering the schedule of a_1 as reserved. A compatible schedule for a_5 is found, preventing a conflict in M_4.

Figure 7 Representations of the schedule of applications a_1 and a_5 from Example 1. For sake of illustration, we consider that all tasks are mapped to the same node. a_1 and a_5 are scheduled in modes M_1 and M_3, and must both be inherited in mode M_4. In Figure 7a, overlapping task schedules result in a conflict, while in Figure 7b this was prevented by reserving a_1's schedule.

4.4.1 Continuity Constraints

The schedule synthesis returns the application schedules (i.e., task and message offsets, and message deadlines) and the round schedules (i.e., starting times and allocation vectors). We represent an application schedule through a scheduling function s as defined below.

▶ **Definition 2** (Scheduling function). *The scheduling function s, defined over the set of applications $\mathcal{A}$, returns for a given application a all parameters characterizing the schedule of a. The schedule of application a is denoted by $s(a)$. $s_M(a)$ denotes the schedule of application a in mode M. The scheduling function is extended to sets of applications as follows*

$$\forall\, S \subset \mathcal{A}\,, \quad s(S) = \bigcup_{a \in S} s(a)$$

All persistent applications $a \in \mathcal{A}_P$ must keep the same schedule across mode changes, which leads us to the definition of continuity constraints.

▶ **Definition 3** (Continuity constraint). *The set of all continuity constraints is given by*

$$\forall\, a \in \mathcal{A}_P, \forall\, (M_i, M_j) \in \mathcal{O}^2,\ a \in M_i \wedge a \in M_j \ \wedge \mathbb{M}(M_i, M_j) = 1 \ \Rightarrow\ s_{M_i}(a) = s_{M_j}(a) \quad (23)$$

▶ **Definition 4** (Schedule domains). *The schedule domains of an application are the possibly multiple subsets of modes in which the application schedule must remain the same.*

▶ **Corollary 5.** *Two modes M_i and M_j belong to the same schedule domain of an application $a \in \mathcal{A}_P$ if and only if (i) a is scheduled in both modes, that is, $a \in M_i \ \wedge \ a \in M_j$, and (ii) there is a possible transition between the two modes, that is, $\mathbb{M}(M_i, M_j) = 1$.*

Proof. Multiple modes belong to the same schedule domain because of a continuity constraint. The formalization of the schedule domains directly follows from Definition 3 ◀

The schedule domains of an application can be extracted from the mode graph $\mathbb{M}$. One approach entails considering the sub-graph $\mathbb{G}_A$ of $\mathbb{M}$ that contains only the modes in which application a is specified – every connected component of $\mathbb{G}_A$ is a schedule domain of a.

Any non-persistent application that is present in multiple modes can be replaced by a distinct application with the same parameters in the respective modes. Similarly, any persistent application with multiple schedule domains can be replicated into distinct applications

having one scheduling domain each (illustrated by Example 6). This leads us to consider in the following that all applications are persistent and have a single schedule domain.

▶ **Example 6.** Consider again the mode graph in Figure 6. Application a_6 has two schedule domains, $\{M_3\}$ and $\{M_5\}$. This can also be modeled by two distinct applications $a_{6.3}$ and $a_{6.6}$ executing in M_3 and M_6. Application a_1 has only one schedule domain, $\{M_1, M_4\}$. If a_1 was not persistent, the continuity constraint would not apply and a_1 could be equivalently modeled by two distinct applications $a_{1.1}$ and $a_{1.4}$ executing in M_1 and M_4.

Continuity constraints can cause conflicts leading to the failure of the multi-mode synthesis although a solution may exist. In particular, if a given mode M belongs to the schedule domains of two different applications, which have been independently scheduled in higher-priority modes, there is a risk of conflict as the inherited schedules may be incompatible. We now formalize the notions of (virtual) legacy applications and conflicting modes. $\overline{X} = \mathcal{A} \setminus X$ denotes the complement of X. For each mode M_i, we define four application sets.

- **Known applications** are the applications previously scheduled in higher-priority modes. The set of known applications of mode M_i is denoted by $K_i = \cup_{j=1}^{i-1} M_j$.
- **Free applications** are the newly scheduled applications in mode M_i, that is, no higher-priority mode belongs to the schedule domain of these applications. The set of free applications of mode M_i is denoted by $F_i = M_i \cap (\mathcal{A} \setminus K_i) = M_i \cap \overline{K_i}$.
- **Legacy applications** are the applications previously scheduled in higher-priority modes that must be scheduled in mode M_i. Since applications have a single schedule domain, M_i necessarily belongs to the same schedule domain as these higher-priority modes and the legacy application schedules must be inherited. The set of legacy applications of mode M_i is denoted by $L_i = M_i \cap K_i$.
- **Virtual legacy applications** are the applications previously scheduled in higher-priority modes that are not scheduled in mode M_i. The set of virtual legacy applications of mode M_i is denoted by $VL_i = (\mathcal{A} \setminus M_i) \cap K_i = \overline{M_i} \cap K_i$. The virtual legacy applications of M_i are not executed in M_i; they simply have been scheduled in higher-priority modes. As illustrated in Example 1, it may be necessary to reserve the space for some of these virtual legacy applications to avoid future inheritance conflicts.

The schedules of two applications A and B are said to be in conflict if two tasks, one from A and one from B, are mapped to the same node and are scheduled in overlapping time intervals. We denote by $s(A) \cap s(B) \neq \emptyset$ the property that A and B are in conflict.

▶ **Definition 7** (Conflict-free). *A set of applications S is said to be conflict-free when there is no conflict between the schedules of the applications in S. We denote by $CF(S)$ the property that S is conflict-free, and $\overline{CF(S)}$ denotes that the set S is in conflict. Formally,*

$$CF(S) \quad \Leftrightarrow \quad \bigcap_{A \in S} s(A) = \emptyset$$

A mode is conflict-free if its legacy applications are conflict-free. In other words, $\forall M_i \in \mathcal{O}$,

$$CF(M_i) \quad \Leftrightarrow \quad CF(L_i)$$

The schedule $Sched(M_i)$ of mode M_i is valid only if $CF(M_i)$.

▶ **Corollary 8.** *A valid schedule for mode $M_i \in \mathcal{O}$ can only exist if the legacy applications of M_i are conflict-free, that is,*

$$CF(L_i) \Leftarrow \text{“Sched}(M_i)\text{ is feasible”}$$

Proof. Using Example 1 as a counter-example, $\overline{CF(L_4)}$ makes it impossible to derive a valid schedule for M_4. ◀

4.4.2 Minimal Inheritance Constraints

The single-mode schedule synthesis is complete: If the problem is feasible for mode M_i, then Algorithm 1 finds a valid schedule. In particular, the scheduled mode is conflict-free, $CF(M_i)$. In the multi-mode case, certain applications are subject to continuity constraints, which are satisfied by fixing the schedules of legacy applications L_i in the MILP formulation for mode M_i. However, by Corollary 8, this leads to a feasible schedule only if $CF(L_i)$.

Example 1 shows that inheriting legacy applications is not sufficient to prevent conflicts. Thus, we now derive the subset of virtual legacy applications VL_i of mode M_i that is necessary and sufficient to reserve in order to guarantee the absence of a conflict due to continuity constraints. More formally, our goal is

$$\forall\, k \in [1..i-1],\ \text{“}Sched(M_k) \text{ is feasible”} \quad \Rightarrow \quad CF(L_i) \tag{24}$$

To this end, we first formalize the constraints on the $Sched()$ function such that continuity constraints are enforced and conflicts are prevented as follows

$$\begin{aligned} &Sched : \mathcal{O} \longmapsto Sched(M) \\ &\quad s.t. \quad CF(M_i) \\ &\qquad \forall\, a \in L_i \cap M_j,\ j < i,\ s_{M_i}(a) = s_{M_j}(a) \\ &\qquad \forall\, a \in F_i,\ s(a) \cap s(\widetilde{VL_i}^a) = \emptyset \end{aligned} \tag{25}$$

The first constraint in Equation (25) ensures that the schedule is valid. The second one enforces the continuity constraints of applications. The third constraint enforces Equation (24) based on the idea that the free applications F_i in mode M_i must be compatible with the schedules of some virtual legacy applications. Theorem 9 derives for any free application the minimal set of such virtual legacy applications.

▶ **Theorem 9** (Minimal virtual legacy sets). *For mode M_i and free application $a \in F_i$, the minimal set of virtual legacy applications $\widetilde{VL_i}^a$ necessary and sufficient to satisfy* (24) *is*

$$\widetilde{VL_i}^a = \{X \in VL_i \mid \exists\, j > i, \quad a \in L_j \ \wedge \ X \in L_j\} \tag{26}$$

Proof. Please refer to Appendix A for the proof of this theorem. ◀

Finally, we set the sum of all message deadlines, $\sum_{m_i \in \mathcal{M}} m_i.d$, as objective function of the MILP solver. Maximizing this allows to limit the constraints inherited between the different modes and improves the schedulability of the multi-mode schedule synthesis problem.

5 Experimental Evaluation

Our experimental evaluation of TTW answers the following questions:

- Are our timing and energy models of the TTnet stack accurate (§ 5.1)?
- How big are the energy savings due to TTnet's round-based communication (§ 5.1)?
- Can the minimal inheritance constraints of TTW's real-time scheduler effectively reduce the number of rounds while respecting the persistency of applications (§ 5.2)?
- How long does it take to solve the multi-mode schedule synthesis problem (§ 5.3)?

All the evaluation artifacts (implementation, raw data, and scripts) are openly available [24].

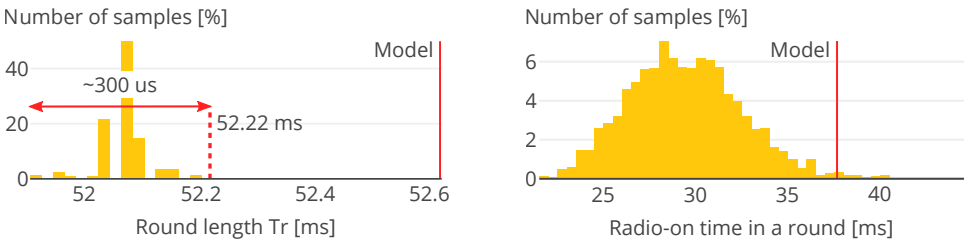


Figure 8 Distributions of round length (left) and radio-on time (right) measurements from all 27 nodes on the FlockLab testbed, collected in one series of 60 runs with 5 slots per round and a payload size of 16 bytes. *Our TTnet models are accurate; the timing model is empirically safe.*

5.1 TTnet Model Validation and Efficiency of Communication Rounds

We begin by validating our TTnet model and investigating the efficiency of rounds. Before discussing our results, we detail the evaluation scenario and the design of our experiments.

Evaluation scenario. Using 27 DPP nodes on the FlockLab testbed [1], we program TTnet to execute one round of B regular slots, followed by B rounds with one regular slot each. For each of these rounds, we collect the round length and the radio-on time. Both values are measured in software using a 32 kHz timer, which gives a measurement resolution of about 30 μs. For $H = 4$ and $N = 2$, we test different number of slots per round B and payload sizes L. For each setting, we measure the round length and radio-on time experienced by each individual node in the network, and we compare the results with our TTnet model.

Experimental design. We design our real-world experiments using TriScale [4], a framework that facilitates the reproducibility of networking evaluations by allowing to make performance claims with quantifiable confidence. TriScale distinguishes *metrics*, which are computed over one run of the evaluation scenario, and *key performance indicators* (KPIs), which capture the expected performance across a series of runs. Concretely, KPIs are percentiles of the underlying distribution of the metric estimated with a certain level of confidence [4].

We consider two performance dimensions: the empirical worst-case length of a round and the average per-node energy savings due to the use of rounds. As we are interested in the worst-case, we take the maximum measurement across all nodes as metric for one run. The true worst-case across any run is the 100th percentile of the metric distribution, which cannot be estimated with a finite number of runs. We thus take as KPI the 95th percentile of our round time metric with a desired confidence level of 95%. For the average energy savings, we use the median values across nodes as metric, and estimate the 5th percentile of that metric with 95% confidence. TriScale indicates that a minimum of 59 runs are needed for these estimations. It has been shown that the experimental conditions on FlockLab exhibit seasonal components [23]. Therefore, to avoid any bias in our results, we perform the series of runs over a span of one week during which we randomly schedule 60 runs for each setting. To investigate the reproducibility of the results, we perform 3 series of tests over the course of six months. We test our TTnet implementation with 5, 10, and 30 slots per round, and a payload of 8, 16, and 64 bytes. This results in a total of 1620 individual runs.

Results – round length. The experimental results in Table 2 confirm that our TTnet model provides a highly accurate estimate of the length of a round. The results are extremely stable across series: the largest difference in the KPI values corresponds to our measurement

Table 2 KPIs from the TTnet model validation for different series and the corresponding model estimates of the round length T_r and the energy savings E. †*marks series in which, due to construction work taking place in the building where FlockLab is deployed, the number of collected metric values is insufficient to compute the KPIs; the reported values are then the maximum round length and the minimum energy saving across all runs in the series.*

Payload	Slots per round	Round length T_r [ms]				Energy savings E [%]			
L [bytes]	B [.]	Series 1	Series 2	Series 3	Model	S. 1	S. 2	S. 3	Model
8	5	42.3	42.3	42.3†	42.52	25	31	29†	34
	10	77.21	77.24	77.24†	77.52	34	36	34†	38
	30	217.01	217.01	217.01	217.52	38	39	39	41
16	5	52.22	52.22	52.22†	52.52	25	24	22†	28
	10	97.04	97.08	97.08†	97.52	28	29	30†	32
	30	276.52	276.52	276.49†	277.52	32	33	32†	34
64	5	104.77	104.74	104.74†	105.02	8	8	9†	14
	10	202.09	202.12	202.09	202.52	8	12	12	16
	30	591.49	591.52	591.49†	592.52	13	11	14†	17

resolution of about 30 μs. Concretely, this means that the largest round length measured by any node is essentially the same. Furthermore, the measured KPI values are very close to and consistently lower than the model estimates. By definition of the KPI, we can claim that, with 95% probability, at least 95% of the runs of the evaluation scenario would yield a maximum round length that is less than or equal to the KPI value [4], and thus smaller than the model estimate. The left plot in Figure 8 shows the distribution of the round length measurements from all the nodes collected during one series of 60 runs. We observe that the distribution is very narrow (less than 300 μs of spread); this is because all operations in a TTnet round are time-triggered, so the measurement differences between nodes only come from the small differences in execution time of Baloo's end-of-round processing.

▶ Remark 10. The first series of test revealed a bug in the time synchronization software which led to an erratic behavior of certain nodes in some corner cases. This bug was fixed before the second series and the erratic data filtered out. In a previous version of this work [21], we presented a single case where one node measured a round time *larger* than the model value. After closer investigation, it appears that this too was a consequence of the time synchronization bug, which we failed to detect and filter. The analysis script has been adapted accordingly; please refer to the TTW artifacts for more details [24].

Results – energy savings. The results in Table 2 also show that the energy savings from the use of rounds in TTnet are significant: the savings are around 30% for a payload size of 16 bytes, which is representative of real-world CPS scenarios [17, 36, 38]. In general, the more slots are packed into a round (increasing B) and the smaller the payload size (decreasing L), the higher the energy savings. Since our energy KPI estimates the 5th percentile of the energy savings, we can claim that, with 95% probability, at least 95% of the runs in our evaluation scenario yield an average energy saving larger than or equal to the KPI value. The right plot in Figure 8 shows the distribution of radio-on time measurements from all the nodes, collected in one series of 60 runs. We see that the nodes experience significant differences in radio-on time during a round. This is expected since the nodes turn off their radio as soon as they have transmitted N times during a Glossy flood, which happens earlier for nodes that are closer to the initiator of the flood.

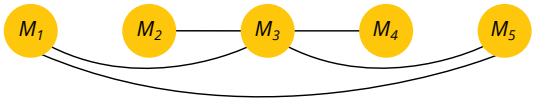


Figure 9 Mode graph used in the evaluation scenario of § 5.2.

Table 3 Solving times for each mode for three inheritance approaches (in seconds).

	M_1	M_2	M_3	M_4	M_5
No inheritance	5	≈ 0	7	294	≈ 0
Minimal inheritance	8	≈ 0	≈ 0	216	≈ 0
Full inheritance	3	33	2	43	578

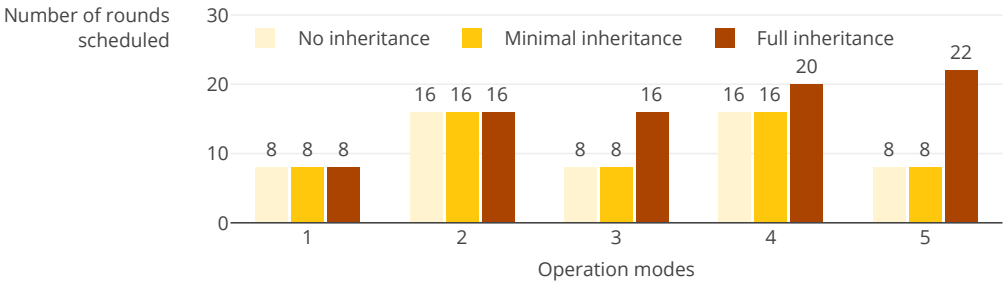


Figure 10 Number of rounds scheduled in each mode for three inheritance approaches. *We consider the number of rounds scheduled over 80 s, the least common multiple of the modes' hyperperiod.*

5.2 Effectiveness of TTW's Minimal Inheritance Approach

We now focus on TTW's real-time scheduler and investigate whether the minimal inheritance constraints from § 4.4.2 help keep the total number of scheduled rounds low, while guaranteeing that persistent applications keep the same schedule across mode changes.

Scenario. We consider a scenario with 13 nodes running 15 different persistent applications. There are in total 45 tasks and 30 messages. The periods and deadlines vary between 10 s and 80 s. The applications execute in 5 different modes; Figure 9 shows the mode graph with all possible transitions. We synthesize schedules for the 5 modes on a standard laptop PC. Besides our *minimal inheritance* approach, we consider two baselines for comparison: (*i*) *no inheritance*, which yields the minimum number of rounds under the (false) assumption that all applications are non-persistent; and (*ii*) *full inheritance*, which makes the (pessimistic) assumption that all applications executing in mode M_i are also executing in M_j. In contrast to *no inheritance*, the *full inheritance* baseline guarantees continuity across mode changes but may find problems to be non-schedulable although a feasible solution exists.

Results. Figure 10 shows for all approaches the number of rounds scheduled in each mode. We can see that with *full inheritance* the number of rounds steadily increases as it assumes that all previously scheduled application are still executing. This not only wastes energy, it also limits scalability as the number of modes grows. Our *minimal inheritance* approach performs and scales significantly better as it considers only the relevant constraints. In fact, in this particular scenario, *minimal inheritance* performs optimally: it does not schedule more rounds than the absolute minimum, which is captured by the *no inheritance* approach.

5.3 Offline Solving Time of TTW's Real-Time Scheduler

Although TTW targets CPS scenarios in which the scheduling tables are synthesized before the system operation starts (e.g., to a priori check that closed-loop stability can be guaranteed under given schedules [36, 7]), the solving time of the real-time scheduler is a relevant factor.

Scenario. Using the scenario from the previous experiment, we measure the solving time for the three different inheritance approaches on a standard laptop PC.

Results. We observe from Table 3 that the per-mode solving time ranges between less than a second and ten minutes, depending on the complexity of the mode. For example, modes M_3 and M_4 contain the most applications, leading to more constraints in the formulation. We also see that *minimal inheritance* does not increase the overall solving time compared to *no inheritance*. This is because, by reserving some applications' schedule, certain problem variables are fixed, which reduces the computational load. However, if too many variables are fixed, as shown by *full inheritance*, the resulting problem may become harder to solve: more rounds are required, which may increases the number of variables and thus the solving time.

6 Conclusions

This paper presented TTW, a time-triggered architecture for wireless CPS. TTW provides guarantees on end-to-end deadlines by statically co-scheduling all tasks and messages in the system, while supporting runtime adaptability via mode changes that respect the schedules of persistent applications. Our design of TTW's real-time scheduler addressed key challenges concerning the formulation and tractability of the scheduling problem. We leveraged synchronous transmissions to design a highly reliable, timing-predictable, and efficient low-power wireless communication stack that is robust to network dynamics. We believe that TTW takes wireless systems a major step closer the wired systems, which opens up several exciting opportunities for future CPS applications that seemed so far out of reach.

References

1 –. FlockLab. `http://flocklab.ethz.ch/`. Last accessed: 2020-04-14.

2 Tarek Abdelzaher and Kang Shin. Combined task and message scheduling in distributed real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 1999. `doi:10.1109/71.809575`.

3 Johan Åkerberg, Mikael Gidlund, and Mats Björkman. Future research challenges in wireless sensor and actuator networks targeting industrial automation. In *2011 9th IEEE International Conference on Industrial Informatics*, 2011. `doi:10.1109/INDIN.2011.6034912`.

4 Anonymous. TriScale: A Framework Supporting Reproducible Performance Evaluations in Networking. In *Zenodo*, 2020. `doi:10.5281/zenodo.3656819`.

5 Mohammad Ashjaei, Nima Khalilzad, Saad Mubeen, Moris Behnam, Ingo Sander, Luis Almeida, and Thomas Nolte. Designing end-to-end resource reservations in predictable distributed embedded systems. *Real-Time Systems*, 2017. `doi:10.1007/s11241-017-9283-6`.

6 Akramul Azim. *Scheduling of Overload-Tolerant Computation and Multi-Mode Communication in Real-Time Systems*. Doctoral Thesis, University of Waterloo, 2014. URL: `http://hdl.handle.net/10012/8973`.

7 Dominik Baumann, Fabian Mager, Romain Jacob, Lothar Thiele, Marco Zimmerling, and Sebastian Trimpe. Fast Feedback Control over Multi-hop Wireless Networks with Mode Changes and Stability Guarantees. *ACM Transactions on Cyber-Physical Systems*, 2019. `doi:10.1145/3361846`.

8 Jan Beutel, Roman Trüb, Reto Da Forno, Markus Wegmann, Tonio Gsell, Romain Jacob, Michael Keller, Felix Sutton, and Lothar Thiele. The Dual Processor Platform Architecture: Demo Abstract. In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*, IPSN '19, Montreal, Quebec, Canada, 2019. ACM. doi:10.1145/3302506.3312481.

9 Tianyang Chen and Linh T. X. Phan. SafeMC: A System for the Design and Evaluation of Mode-Change Protocols. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2018. doi:10.1109/RTAS.2018.00021.

10 Octav Chipara, Chengjie Wu, Chenyang Lu, and William Griswold. Interference-Aware Real-Time Flow Scheduling for Wireless Sensor Networks. In *2011 23rd Euromicro Conference on Real-Time Systems*, 2011. doi:10.1109/ECRTS.2011.15.

11 Silviu S. Craciunas and Ramon Serna Oliver. SMT-based Task- and Network-level Static Schedule Generation for Time-Triggered Networked Systems. In *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*, RTNS '14, Versaille, France, 2014. ACM. doi:10.1145/2659787.2659812.

12 Silviu S. Craciunas and Ramon Serna Oliver. Combined Task- and Network-level Scheduling for Distributed Time-triggered Systems. *Real-Time Systems*, 2016. doi:10.1007/s11241-015-9244-x.

13 Federico Ferrari, Marco Zimmerling, Luca Mottola, and Lothar Thiele. Low-power Wireless Bus. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, New York, NY, USA, 2012. ACM. doi:10.1145/2426656.2426658.

14 Federico Ferrari, Marco Zimmerling, Lothar Thiele, and Olga Saukh. Efficient network flooding and time synchronization with Glossy. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2011. URL: https://ieeexplore.ieee.org/document/5779066.

15 FlexRay. ISO 17458-1:2013–Road vehicles–FlexRay communications system–Part 1: General information and use case definition. Standard, International Organization for Standardization (ISO), Geneva, Switzerland, 2013. URL: https://www.iso.org/standard/59804.html.

16 Gerhard Fohler. Changing operational modes in the context of pre run-time scheduling. *IEICE transactions on information and systems*, 1993. URL: https://pdfs.semanticscholar.org/272b/615266e763369e903dcb0b966e22077f127c.pdf.

17 Samira Hayat, Evşen Yanmaz, and Raheeb Muzaffar. Survey on Unmanned Aerial Vehicle Networks for Civil Applications: A Communications Viewpoint. *IEEE Communications Surveys Tutorials*, Fourthquarter 2016. doi:10.1109/COMST.2016.2560343.

18 Tian He, John A. Stankovic, Chenyang Lu, and Tarek Abdelzaher. SPEED: A stateless protocol for real-time communication in sensor networks. In *23rd International Conference on Distributed Computing Systems, 2003. Proceedings.*, 2003. doi:10.1109/ICDCS.2003.1203451.

19 Jia Huang, Jan Olaf Blech, Andreas Raabe, Christian Buckl, and Alois Knoll. Static scheduling of a Time-Triggered Network-on-Chip based on SMT solving. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2012. doi:10.1109/DATE.2012.6176522.

20 International Electrotechnical Commission (IEC). Industrial networks - Wireless communication network and communication profiles - WirelessHART. https://webstore.iec.ch/publication/24433. Last accessed: 2020-04-14.

21 Romain Jacob. *Leveraging Synchronous Transmissions for the Design of Real-Time Wireless Cyber-Physical Systems.* Doctoral Thesis, ETH Zurich, 2020. Accepted: 2020-02-26T08:48:57Z. doi:10.3929/ethz-b-000401717.

22 Romain Jacob, Jonas Bächli, Reto Da Forno, and Lothar Thiele. Synchronous Transmissions Made Easy: Design Your Network Stack with Baloo. In *Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks*, 2019. doi:10.3929/ethz-b-000324254.

23 Romain Jacob, Reto Da Forno, Roman Trüb, Andreas Biri, and Lothar Thiele. Wireless Link Quality Estimation on FlockLab - and Beyond, 2019. doi:10.5281/zenodo.3354717.

24 Romain Jacob and Licong Zhang. TTW Artifacts - Initial release, 2020. doi:10.5281/zenodo.3759222.

25 Romain Jacob, Licong Zhang, Marco Zimmerling, Jan Beutel, Samarjit Chakraborty, and Loth Thiele. TTW: A Time-Triggered Wireless design for CPS. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018. doi:10.23919/DATE.2018.8342127.

26 Romain Jacob, Marco Zimmerling, Pengcheng Huang, Jan Beutel, and Lothar Thiele. End-to-End Real-Time Guarantees in Wireless Cyber-Physical Systems. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, 2016. doi:10.1109/RTSS.2016.025.

27 Kevin Jeffay, Donald F. Stanat, and Charles U. Martel. On non-preemptive scheduling of period and sporadic tasks. In *Proceedings Twelfth Real-Time Systems Symposium*, 1991. doi:10.1109/REAL.1991.160366.

28 Prachi Joshi, Haibo Zeng, Unmesh D. Bordoloi, Soheil Samii, S. S. Ravi, and Sandeep K. Shukla. The Multi-Domain Frame Packing Problem for CAN-FD. In Marko Bertogna, editor, *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ECRTS.2017.12.

29 Jens Karschau, Marco Zimmerling, and Benjamin M. Friedrich. Renormalization group theory for percolation in time-varying networks. *Scientific Reports*, 2018. doi:10.1038/s41598-018-25363-2.

30 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)*, 2005. doi:10.1109/ISORC.2005.56.

31 Hermann Kopetz and Günther Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 2003. doi:10.1109/JPROC.2002.805821.

32 Hermann Kopetz and G. Grunsteidl. TTP - A time-triggered protocol for fault-tolerant real-time systems. In *FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing*, 1993. doi:10.1109/FTCS.1993.627355.

33 Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Algorithms and Combinatorics. Springer-Verlag, Berlin Heidelberg, third edition, 2006. doi:10.1007/3-540-29297-7.

34 Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Lecture Notes in Computer Science, Lect.Notes Computer. Tutorial. Springer-Verlag, Berlin Heidelberg, 2001. doi:10.1007/3-540-45318-0.

35 Roman Lim, Federico Ferrari, Marco Zimmerling, Christoph Walser, Philipp Sommer, and Jan Beutel. FlockLab: A Testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems. In *Proceedings of the 12th International Conference on Information Processing in Sensor Networks*, IPSN '13, New York, NY, USA, 2013. ACM. doi:10.1145/2461381.2461402.

36 Fabian Mager, Dominik Baumann, Romain Jacob, Lothar Thiele, Sebastian Trimpe, and Marco Zimmerling. Feedback Control Goes Wireless: Guaranteed Stability over Low-power Multi-hop Networks. In *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, ICCPS '19, Montreal, Quebec, Canada, 2019. ACM. doi:10.1145/3302509.3311046.

37 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)*, 2015. doi:10.1109/SIES.2015.7185055.

38 James A. Preiss, Wolfgang Honig, Gaurav S. Sukhatme, and Nora Ayanian. Crazyswarm: A large nano-quadcopter swarm. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017. doi:10.1109/ICRA.2017.7989376.

39 Abusayeed Saifullah, You Xu, Chenyang Lu, and Yixin Chen. Real-Time Scheduling for WirelessHART Networks. In *2010 31st IEEE Real-Time Systems Symposium*, 2010. doi:10.1109/RTSS.2010.41.

40 Markus Schuß, Carlo Alberto Boano, Manuel Weber, and Kay Römer. A Competition to Push the Dependability of Low-Power Wireless Protocols to the Edge. In *Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks*, EWSN '17, USA, 2017. Junction Publishing. doi:10.5555/3108009.3108018.

41 Wilfried Steiner. An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks. In *2010 31st IEEE Real-Time Systems Symposium*, 2010. doi:10.1109/RTSS.2010.25.

42 Felix Sutton, Marco Zimmerling, Reto Da Forno, Roman Lim, Tonio Gsell, Georgia Giannopoulou, Federico Ferrari, Jan Beutel, and Lothar Thiele. Bolt: A Stateful Processor Interconnect. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, SenSys '15, New York, NY, USA, 2015. ACM. doi:10.1145/2809695.2809706.

43 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*, CODES+ISSS '12, Tampere, Finland, 2012. ACM. doi:10.1145/2380445.2380518.

44 Watteyne Watteyne, Thomas, Pere Tuset-Peiro, Xavier Vilajosana, Sofie Pollin, and Bhaskar Krishnamachari. Teaching Communication Technologies and Standards for the Industrial IoT? Use 6TiSCH! *IEEE Communications Magazine*, 2017. doi:10.1109/MCOM.2017.1700013.

45 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 2008. doi:10.1145/1347375.1347389.

46 Licong Zhang, Dip Goswami, Reinhard Schneider, and Samarjit Chakraborty. Task- and network-level schedule co-synthesis of Ethernet-based time-triggered systems. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014. doi:10.1109/ASPDAC.2014.6742876.

47 Marco Zimmerling. *End-to-End Predictability and Efficiency in Low-Power Wireless Networks*. Doctoral Thesis, ETH Zurich, Zürich, 2015. doi:10.3929/ethz-a-010531577.

48 Marco Zimmerling, Federico Ferrari, Luca Mottola, and Lothar Thiele. On Modeling Low-Power Wireless Protocols Based on Synchronous Packet Transmissions. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems(MASCOTS)*, 2013. doi:10.1109/MASCOTS.2013.76.

49 Marco Zimmerling, Luca Mottola, Pratyush Kumar, Federico Ferrari, and Lothar Thiele. Adaptive Real-Time Communication for Wireless Cyber-Physical Systems. *ACM Transactions on Cyber-Physical Systems*, 2017. doi:10.1145/3012005.

50 Marco Zimmerling, Luca Mottola, and Silvia Santini. Synchronous Transmissions in Low-Power Wireless: A Survey of Communication Protocols and Network Services. *arXiv:2001.08557 [cs, eess]*, 2020. arXiv:2001.08557.

A Proof of Theorem 9

We first prove by recurrence that virtual legacy sets defined in (26) are sufficient to satisfy (24).

For the highest priority mode M_1, by definition, $L_1 = \emptyset$, thus $CF(L_1)$. Let us assume that for any $k \in [1..i]$, $Sched(M_k)$ is feasible in the sense of (25). This induces that $CF(S_k)$, hence $CF(L_k)$ for any $k \in [1..i]$. Let us finally assume that L_{i+1} is *not* conflict-free; that is,

$$\overline{CF(L_{i+1})} \quad \Leftrightarrow \quad \bigcap_{A \in L_{i+1}} s(A) \neq \emptyset \quad \Rightarrow \quad \exists\, (A,B) \in {L_{i+1}}^2,\ s(A) \cap s(B) \neq \emptyset \tag{27}$$

$$\Rightarrow \quad \begin{cases} \exists!\, M_a,\ A \in F_a \ \wedge\ a < i+1 \\ \exists!\, M_b,\ B \in F_b \ \wedge\ b < i+1 \end{cases} \tag{28}$$

where ∃! means "there exists a unique." Without loss of generality, we consider $a \leq b$. If $a = b$, then $S_a = S_b$ and $CF(S_a) \equiv CF(S_b)$. Therefore,

$$\bigcap_{A \in S_a = S_b} s(A) = \emptyset \quad \Rightarrow \quad s(A) \cap s(B) = \emptyset \tag{29}$$

This contradicts (27), thus necessarily $a < b$; in other words, mode M_a has higher priority than mode M_b. Therefore, a belongs either to L_b or VL_b by definition of those sets. By hypothesis, $CF(S_b)$ and $(28) : B \in F_b$, thus

$$A \in L_b \quad \Rightarrow \quad s(A) \cap s(B) = \emptyset$$

which again contradicts (27). Hence necessarily, $A \in VL_b$. Furthermore,

$$\left.\begin{array}{l}(28) : i+1 > b \\ (27) : A \in L_{i+1} \\ (27) : B \in L_{i+1}\end{array}\right\} \text{ Taking } b = i \text{ and } j = i+1, \quad (26) \ : \ A \in \widetilde{VL_b{}^B} \tag{30}$$

By hypothesis, $Sched(M_b)$ is feasible, thus $s(B) \cap s(\widetilde{VL_b{}^B}) = \emptyset$, which yields $s(B) \cap s(A) = \emptyset$ and contradicts (27) again. Therefore, the recurrence hypothesis is necessarily false. Hence, if for any $k \in [1..i]$, $Sched(M_k)$ is feasible in the sense of (25), then $CF(L_{i+1})$. By recurrence, we can conclude that the virtual legacy sets as defined by (26) are sufficient to satisfy (24).

We now prove that the virtual legacy sets are also necessary. Let us consider smaller virtual legacy sets than defined by (26), that is, $\exists\, i \in [1..M],\ a \in F_i,\ \widehat{VL_i{}^A} \not\subseteq \widetilde{VL_i{}^A}$. Let us further assume that $Sched()$ is redefined to replace $\widetilde{VL}$ by $\widehat{VL}$. By hypothesis,

$$\exists\, X \in \mathcal{A},\ X \in \widehat{VL_i{}^A} \ \wedge \ X \notin \widetilde{VL_i{}^A} \tag{31}$$

$$\text{Furthermore,} \quad X \in \widetilde{VL_i{}^A} \quad \Rightarrow \quad X \in VL_i \quad \Rightarrow \quad X \notin S_i \tag{32}$$

$$X \in \widetilde{VL_i{}^A} \quad \Rightarrow \quad \exists\, j > i, \quad a \in L_j \ \wedge \ X \in L_j \tag{33}$$

Assuming that $Sched(M_i)$ is feasible, the resulting schedule guarantees that $CF(M_i)$ and $\forall\, a \in F_i,\ s(a) \cap s(\widehat{VL_i{}^A}) = \emptyset$. However, $(31) : X \notin \widehat{VL_i{}^A}$ and $(32) : X \notin S_i$. Hence, schedule $s(a)$ may be synthesized such that $s(A) \cap s(X) \neq \emptyset$. According to $(33) : (A, X) \in {L_j}^2$, which induces a conflict in mode M_j. Hence, we can conclude that no sets $\widehat{VL}$ smaller than $\widetilde{VL}$ are sufficient to satisfy (24).

Overall, this shows that the virtual legacy sets $\widetilde{VL}$ as defined in (26) are both necessary and sufficient for the schedule synthesis method to satisfy (24), that is, to guarantee that inheritance of schedules from legacy applications does not lead to conflicts in lower-priority modes. In other words, $\widetilde{VL}$ from (26) defines the sets of minimally restrictive constraints such that $Sched()$ as defined in (25) satisfies (24). This completes the proof of Theorem 9.