End-to-End Schedulability of Virtualized Distributed Time-Triggered Systems

Jan Ruh
TTTech Computertechnik AG
Vienna, Austria
jan.ruh@tttech.com

Silviu S. Craciunas
TTTech Computertechnik AG
Vienna, Austria
silviu.craciunas@tttech.com

ABSTRACT

In most distributed safety-critical applications, real-time tasks executing on multi-core multi-SoC platforms communicate critical messages over a deterministic communication backbone. Virtualization is increasingly used in such systems to enhance scalability and enforce spatial isolation while still maintaining the timing properties of critical tasks. Using time-triggered (TT) scheduling on the computation, the communication, and the virtualization layer enables strict end-to-end timing guarantees with tight latency and jitter bounds. In this paper, we introduce for the first time a formal system and scheduling model to harmonize these different layers of the global scheduling problem for virtualized distributed TT real-time systems. We also describe two offline algorithms to generate TT schedules for all layers that guarantee end-to-end latency requirements. The two schedulers offer trade-offs between schedule synthesis runtime and virtualization overhead. We derive synthetic benchmarks from real-world system properties to show the scalability and schedulability of our approaches.

CCS CONCEPTS

• **Computer systems organization → Reliability**.

KEYWORDS

Time-triggered scheduling, Hypervisor, Real-time.

ACM Reference Format:
Jan Ruh and Silviu S. Craciunas. 2024. End-to-End Schedulability of Virtualized Distributed Time-Triggered Systems. In *The 32nd International Conference on Real-Time Networks and Systems (RTNS 2024), November 7–8, 2024, Porto, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3696355.3696364

ACKNOWLEDGMENTS

The project AIMS5.0 is supported by the Chips Joint Undertaking and its members, including the top-up funding by National Funding Authorities from involved countries under grant agreement no. 101112089.

RTNS 2024, November 7–8, 2024, Porto, Portugal

1 INTRODUCTION

Time-triggered (TT) distributed systems are often used in highly critical real-time domains like aerospace [5, 23] and appear progressively in other safety-critical applications from the automotive [5, 20, 41, 58], and railway [50] to industrial domains [6, 30], due to its benefits in terms of determinism, stability, and compositionality [20, 42, 44, 45, 58, 64]. In a recent survey of applied real-time systems, 68% industrial practitioners have responded that static offline scheduling is used to improve timing predictability, while 54% employ TT scheduling in their systems [3]. In most distributed safety-critical applications, real-time tasks execute on multi-core multi-SoC platforms and communicate critical messages over a deterministic communication backbone. TT scheduling is used on one or more of these layers: on the computation layer [14, 46, 47], the communication layer (via e.g., Flexray [55], TTP [36], TTEthernet [62], or TSN [28]), or even on the system virtualization layer [49, 63]. TT scheduling on the hypervisor level, a.k.a. time partitioning, has been extensively studied [24, 25, 33, 49, 63] due to its relevance for safety-critical and mixed-critical system architectures. While TT scheduling on any one of the computation and communication layers will increase the timing predictability of the system, having a TT scheduling approach on all layers, i.e., from task execution in the guest operating system, via the virtual CPUs (VCPUs) in the hypervisor, to the network, enables strict end-to-end timing guarantees with tight latency and jitter bounds. Moreover, utilizing TT scheduling on all layers enables a precise coordination of task execution and message transmission across the entire distributed system. The strict design-time end-to-end guarantees enabled by TT scheduling on all layers help in the certification process and offer mathematical guarantees for hard real-time systems.

Despite the technological maturity of time-partitioning [24, 25, 33, 49, 63] and its integration with time-triggered communication [10, 11, 53, 54, 66], we are still short of a formal definition of what constitutes a correct schedule in a virtualized distributed TT system. As a result, in this paper, we introduce for the first time a formal system and scheduling model to harmonize the different dimensions of the global scheduling problem, i.e., the computation, communication, and virtualization layers, for virtualized distributed TT systems. Moreover, we generalize the correctness constraints of the TSN network scheduling layer to non-strictly-periodic communication slots. Then, we introduce two offline schedulers to generate TT schedules for all layers that guarantee end-to-end latency requirements and offer trade-offs between schedule synthesis runtime and virtualization overhead. The first scheduler, called iSMT, encodes the problem into a Satisfiability Modulo Theories (SMT) [4, 17] formulation and minimizes virtualization overhead by solving the problem incrementally. The second approach,

called iSMT-EDF, uses iSMT to schedule the network layer and improves the scalability of iSMT at the expense of increased virtualization overhead by simulating an Earliest-Deadline-First (EDF) schedule with a simple heuristic to find task and VCPU schedules. We derive synthetic benchmarks from real-world system properties to show the scalability and schedulability of our approaches.

We describe related work in Section 2 and introduce the system model in Section 3. We formulate the correctness constraints in Section 4, followed by a description of our scheduling algorithms (Section 5) and experimental evaluation (Section 6). We conclude the paper in Section 7.

2 RELATED WORK

DREAMS [2] describes building blocks for mixed-critical system architectures integrating multicore systems-on-a-chip interconnected by on-and off-chip networks. One such building block is a virtualization layer featuring time-partitioning via the Xtratum hypervisor [16]. Gala et al. [25] make a case for TT VCPU scheduling to enable hosting real-time applications in private clouds. They describe time-triggered scheduling for the Linux kernel utilizing it to dispatch VCPUs according to a static schedule table implementing TT VMs. Time partitioning is also present in the commercially available PikeOS *microvisor*[1] [24, 33, 49, 63]. R. Kaiser [33] proposed a scheduler featuring dedicated sub-schedulers for real-time and non-real-time VMs. For real-time VMs, the hypervisor differentiates TT, periodic, event-driven, and sporadic VMs. VM priorities introduce precedence rules so that the hypervisor always dispatches real-time before non-real-time VMs whereas the precedence between real-time VMs remains configurable, i.e., it is possible to define which event-driven VMs may preempt a specific TT VM. Two white papers [49, 63], elaborate on TT operation of PikeOS. H. Theiling [63] reports activities in projects funded by the European Union, exploring the integration of TT off- and on-chip communication and TT VMs. He described how TT operation requires synchronization slots to align VM execution with the communication schedule. Motzkus and Oezer [49] describe PikeOS's scheduler using background and foreground time-partitions, which has been patented in [24], and how it enables EN 50128 certified TT multicore scheduling.

Kerstan et al. [34] describe how to transform periodic tasks to guarantee their timely execution inside single time slot periodic partitions [48] using EDF or rate monotonic (RM). The approach is not applicable to multiple time slot periodic partitions [48] and does not address end-to-end timing requirements as they arise in modern distributed real-time systems. In [51], the authors explore constraints arising from scheduling a fully TT virtualized system, i.e., including VMs and tasks. The resulting overlapping constraint for fully TT virtualized systems and derived optimization criteria constitute necessary yet not sufficient constraints for the correctness of a hierarchical schedule table.

Another related research direction is the hierarchical scheduling framework [40, 43, 60] which is based on abstracting periodic virtual resources that are guaranteed a certain bounded service on the physical resource but this guarantee is arbitrarily allocated over the resource period [21, 59]. Originally, hierarchical scheduling research was focused on global schedulability analysis with local scheduler abstractions under EDF or FP (e.g. [59]). In [39] the authors present the use of hierarchical scheduling in Xen in which the guest OS domains are scheduled using periodic servers. In [65], a two-level hierarchical scheduler with real-time guarantees is implemented in the L4/Fiasco micro-kernel with L4Linux as a virtual machine (VM). Dependencies between tasks in different partitions are considered in [32] such that on the hypervisor level, the schedule fragments can be assigned to time partitions in such a way that both real-time requirements and precedence relations are maintained. Our work is the first to consider end-to-end latency constraints in time-triggered virtualized environments, harmonizing the time-triggered execution on different levels of the scheduling heirarchy.

[1]Gernot Heiser coined the term microvisor emphasizing the convergence of modern microkernel and hypervisor architectures [26].

3 SYSTEM MODEL

We model a virtualized distributed real-time system as a directed graph $G(\mathcal{V}, \mathcal{L})$ in which the vertices $\mathcal{V} = \mathcal{V}^N \bigcup \mathcal{V}^S$ are end-systems (ES) $\mathcal{V}^N$ and switches (SW) $\mathcal{V}^S$, and the edges $\mathcal{L} \subset \mathcal{V} \times \mathcal{V}$ represent the bi-directional, full-duplex physical links between the nodes. Each ES $n \in \mathcal{V}^N$ can host a finite set of *virtual machines* (VMs) sharing n's hardware resources. To harmonize the notation, we say that a *native node* hosts a single VM without virtualization overhead. VMs execute a finite set of dependent and independent tasks using time-triggered (TT) scheduling. Dependent tasks communicate with each other via streams scheduled on a TSN backbone.

We assume an end-system is a symmetric multiprocessing (SMP) platform with a table-driven dispatcher in the hypervisor and the VMs. The hypervisor dispatches VMs' VCPUs on the available physical processor cores, and the guest operating systems dispatch periodic TT tasks on VCPUs according to coordinated precomputed schedule tables. When considering schedule tables of different ES, their dispatchers operate on distinct yet synchronized timelines that depend on the underlying clocks' granularity and the *clock synchronization precision*. Typically, a node $n \in \mathcal{V}$ clock's granularity (*microtick*) $\mu_n > 0$ equals the inverse of its oscillator's frequency.

The *macrotick* (slot length) η_n divides the scheduling timeline on the guest OS and hypervisor levels of a node $n \in \mathcal{V}$ into equal slices, representing the smallest granularity for scheduling [31, 35]. Typically, the macrotick is either equal or a multiple of the microtick μ_n. A static schedule on a node $n \in \mathcal{V}$ consists of a set of slots $\{[t \cdot \eta_n, (t+1) \cdot \eta_n)\}$ that repeats after a time interval called the schedule cycle or hyperperiod. Note the trade-off between schedulability and scheduler runtime when choosing the macrotick, as smaller values will increase schedulability, but the search for the schedule table (and the size of the schedule table and the context-switching overhead) may increase. To not overburden the notation, we assume that $\forall n \in \mathcal{V} : \eta_n = \mu_n$, noting that a generalization to macroticks that are a multiple of the microtick is trivial (c.f. Section 4.1).

3.1 Network and Communication Model

The communication backbone is a time-sensitive network (TSN) [28] with IEEE 802.1Qbv [27] scheduled traffic and IEEE 802.1As-rev [29] clock synchronization capabilities. IEEE 802.1Qbv [27] defines a

time-aware gate mechanism that opens or closes egress ports, allowing or denying the forwarding of frames. At runtime, the state of the time-aware gate changes according to a precomputed cyclic schedule table (c.f. [13]) known as a Gate Control List (GCL), triggered by the local clock of the device [12]. To ensure a common time base, the local clocks of the devices are synchronized with the IEEE 802.1AS-rev protocol within a known clock synchronization precision. The hypervisor on compute nodes virtualizes its local clock granting itself and all hosted VMs access to the fault-tolerant network time [57] following a dependent clock paradigm [9, 56]. Note that our method also works for other time-triggered networks like TTEthernet (SAE AS6802) [62], with changed network correctness conditions (c.f. Section 4.4.2), as described in [15]. We model the network and communication streams similarly to [13].

A stream $s_i \in \mathcal{S}$ is a communication requirement with period T_i from a sender $a \in \mathcal{V}$ to a receiver $b \in \mathcal{V}$. For each stream, we assume a predefined route from sender to receiver via intermediate nodes $v_1, ..., v_m \in N$, and encode it as a finite ordered set

$$s_i = \{(a, v_1), (v_1, v_2), ..., (v_{m-1}, v_m), (v_m, b)\}.$$

Each stream $s_i \in \mathcal{S}$ periodically releases stream jobs that must be transmitted within their respective period instances. Contrary to previous work [13, 52, 61], we assume a more generic non-strictly periodic model, i.e., each frame of a stream can be transmitted at a different time in each period instance. We denote the finite ordered set of jobs of a stream s_i that are released until the hyperperiod with $\mathcal{J}_i^{xy}$. We use the superscript xy to denote the instance of the stream on each communication link $(x, y) \in s_i$.

Each stream $s_i \in \mathcal{S}$ has a payload with a given size S_i, which can also be larger than the maximum transmission unit (MTU), resulting in a set of frames, each with size less than or equal to the MTU. Hence, each stream job $J_{ij}^{xy} \in \mathcal{J}_i^{xy}$ can feature multiple frames of the same stream since the total payload of the stream can exceed the MTU, which we denote with $\mathcal{F}_{ij}^{xy}$ where $|\mathcal{F}_{ij}^{xy}| = \lceil \frac{S_i}{MTU} \rceil$. Each frame $f_{ijk}^{xy} \in \mathcal{F}_{ij}^{xy}$ has a transmission duration $f_{ijk}^{xy}.L$ when transmitted over a link $(x, y) \in s_i$ with a given speed $s^{x,y}$ in bits/second. The transmission duration is computed as $f_{ijk}^{xy}.L = \frac{S_{ij} \cdot 10^9 \cdot 8}{s^{xy}}$, where S_{ij} is the size of the frame f_{ijk}^{xy} in bytes. The transmission time of a frame $f_{ijk}^{xy} \in \mathcal{F}_{ij}^{xy}$ on link $(x, y) \in s_i$ is denoted with $f_{ijk}^{xy}.\phi$ relative to the period. The absolute scheduling offset of a frame f_{ijk}^{xy} in period instance j is $f_{ijk}^{xy}.\phi + j \cdot T_i$.

3.2 Task Model

A periodic real-time (RT) task τ_i is defined by the tuple $(R_i, D_i, C_i, T_i, A_i)$ in which R_i is the release time, D_i is the relative deadline, C_i is the WCET, T_i is the period, and A_i is the task's resource affinity. We denote the set of all periodic RT tasks with $\mathcal{T}$. Switching from one task to another on a core c introduces a context-switching overhead equal to $\delta_\tau(c)$.

Each task $\tau_i \in \mathcal{T}$ periodically releases jobs that must complete execution within their deadline. For the tasks, we again assume a non-strictly periodic model, i.e., each task job can be executed at a different time in each period instance. We denote the finite ordered set of jobs of a task τ_i that are released until the hyperperiod with $\mathcal{J}_i$. Since we allow preemption, each job $J_{ij} \in \mathcal{J}_i$ is comprised of a finite

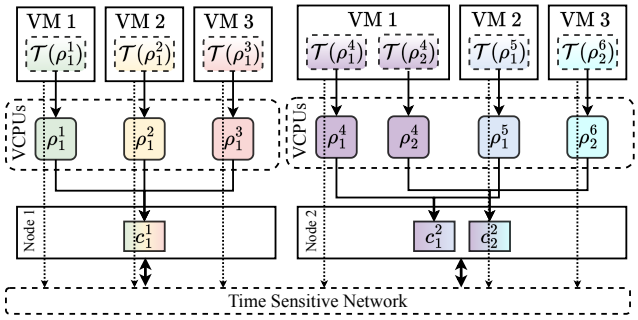


Figure 1: System featuring two nodes with six VMs

list of *job segments* $\mathcal{X}_{ij}$. A job segment $x_{ijk} \in \mathcal{X}_{ij}$ is defined by its offset $x_{ijk}.\phi$ and its length $x_{ijk}.L$, where the offset is the placement of the segment relative to the period. The absolute scheduling offset of a segment x_{ijk} in the period instance is $x_{ijk}.\phi + j \cdot T_i$. Please note that the number and size of segments are part of the scheduling problem and are not fixed (c.f. Section 5). For this section, we are only interested in the correctness constraints of an existing solution, hence we assume a given number of segments for each job.

A pair of periodic RT tasks communicate with each other via a TSN stream. For two tasks $\tau_i, \tau_j \in \mathcal{T}$, we indicate a data dependency between, i.e., τ_j reads input data from τ_i via flow $s_k \in \mathcal{S}$, by writing $\tau_i \xrightarrow{s_k} \tau_j$. The end-to-end latency of a data dependency is expressed as $E(\tau_i \xrightarrow{s_k} \tau_j)$. We assume single-rate dependencies, where the periods of sender and receiver tasks, as well as the stream period, are identical. A more general multi-rate dependency [7, 8] is possible but has to be explicitly specified in terms of dependencies between individual task jobs and frame instances. To simplify the notation, we assume that multi-rate dependencies have been broken down into single-rate job-level dependencies, e.g. via the heuristic in [7, 8].

While we only consider periodic tasks, we note that sporadic tasks can be readily integrated into time-triggered scheduling via a second-level scheduler [22, 31, 46, 54]. Aperiodic, best-effort tasks can run in the background (e.g., via bandwidth servers [1]) whenever there is an idle slot in the TT schedule or there is slack in the system via TT tasks finishing earlier than their WCET assumption.

3.3 System Virtualization Model

We assume that a host node $n \in \mathcal{V}$ is an SMP system, where C^n denotes node n's set of cores that share a main memory. On each ES node $n \in \mathcal{V}$, multiple TT VMs are running that share the physical compute and memory resources but have access to a dedicated network interface card (NIC) connected to the TSN backbone (c.f. Figure 1). Each VM is modeled as a set of VCPUs. The set of all VCPUs running on a core c is defined using $\mathcal{P}(c)$, where each VCPU is assigned to run on exactly one physical core, with no migration allowed at runtime.

Akin to task jobs, we say that a TT VCPU $\rho_i \in \mathcal{P}(c)$ is comprised of a set of segments $\mathcal{Z}_i$, where we define each segment $z_{ij} \in \mathcal{Z}_i$ by an offset $z_{ij}.\phi$ and a length $z_{ij}.L$. Contrary to tasks, VCPU segments do not have a period and their offset is relative to the hyperperiod. Switching from one VCPU to another on a core c introduces a switching overhead equal to $\delta_\rho(c)$. Periodic RT tasks

Symbol	Level	Description
$G(\mathcal{V}, \mathcal{L})$	System	Distributed system with nodes $\mathcal{V}$ and links $\mathcal{L}$
$\mathcal{V} = \mathcal{V}^N \bigcup \mathcal{V}^S$	System	End-systems (ES) $\mathcal{V}^N$ and switches (SW) $\mathcal{V}^S$
$\mathcal{L} \subset \mathcal{V} \times \mathcal{V}$	System	Bi-directional, full-duplex links between nodes
μ_n	System	Microtick granularity of node $n \in \mathcal{V}$
η_n	System	Scheduling granularity (Macrotick) of node $n \in \mathcal{V}$
$c \in \mathcal{C}^n$	System	A core of the set of all cores $\mathcal{C}^n$ of node n
$\delta_\tau(c)$	System	Task context-switching overhead on core c
$\delta_\rho(c)$	System	VCPU switching overhead on core c
Π	Network	Clock synchronization precision in the network
Δ^{xy}	Network	Propagation delay of link (x, y)
$\mathcal{S}$	Network	Set of all streams in the system
$s_i \in \mathcal{S}$	Network	Stream with period T_i and size S_i
$\mathcal{J}_i^{xy}$	Network	Set of jobs of a stream s_i on link $(x, y) \in \mathcal{L}$.
$\mathcal{F}_{ij}^{xy}$	Network	Set of frames of a stream job $J_{ij}^{xy} \in \mathcal{J}_i^{xy}$
$f_{ijk}^{xy} \in \mathcal{F}_{ij}^{xy}$	Network	Frame k of job J_{ij}^{xy} of stream s_i on link (x, y)
$f_{ijk}^{xy}.\phi$	Network	Scheduling offset of frame f_{ijk}^{xy} on link (x, y)
$f_{ijk}^{xy}.L$	Network	Transmission duration of frame f_{ijk}^{xy} on link (x, y)
$\tau_i \in \mathcal{T}$	Task	A periodic real-time task from the set of tasks $\mathcal{T}$
$\tau_i \xrightarrow{s_k} \tau_j$	Task	Data dependency between τ_i and τ_j via stream s_k
$E(\tau_i \xrightarrow{s_k} \tau_j)$	Task	The end-to-end latency of dependency $\tau_i \xrightarrow{s_k} \tau_j$
R_i, D_i, C_i, T_i, A_i	Task	Release, deadline, WCET, period, affinity
$J_{ij} \in \mathcal{J}_i$	Task	The job j from the set of all jobs $\mathcal{J}_i$ of task τ_i
$\mathcal{X}_{ij}$	Task	The finite set of job segments of job J_{ij}
$x_{ijk} \in \mathcal{X}_{ij}$	Task	Job segment k of the segment set $\mathcal{X}_{ij}$ of job J_{ij}
$x_{ijk}.\phi$	Task	Scheduling offset of task segment x_{ijk} of τ_i
$x_{ijk}.L$	Task	Size of task segment x_{ijk} of task τ_i
$\mathcal{P}(c)$	Hypervisor	The set of all VCPUs running on a core c
$\rho_i \in \mathcal{P}(c)$, $\mathcal{Z}_i$	Hypervisor	A TT VCPU comprised of a set of segments $\mathcal{Z}_i$
$\mathcal{T}(\rho_i)$	Hypervisor	The set of real-time tasks allocated to VCPU ρ_i
$\mathcal{Z}_i$	Hypervisor	The set of all segments of VCPU ρ_i
$z_{ij} \in \mathcal{Z}_i$	Hypervisor	Segment j from the segment set $\mathcal{Z}_i$ of VCPU ρ_i
$z_{ij}.\phi$	Hypervisor	Scheduling offset of segment z_{ij} of VCPU ρ_i
$z_{ij}.L$	Hypervisor	Size of VCPU segment z_{ij} of VCPU ρ_i

Table 1: Summary of notations

require assignment to TT VCPUs. We denote the set of periodic RT tasks allocated to a virtual node's TT VCPU ρ with $\mathcal{T}(\rho)$. Note that while a VCPU can be assigned to any core in its affinity list, all VCPUs of a VM have to be assigned to cores of the same physical node. For the correctness constraints, we assume that the schedule and the assignment of VCPUs to cores are known.

4 CORRECTNESS CONSTRAINTS

In this section, we formalize the constraints that define a correct schedule in the context of a virtualized time-triggered distributed real-time system. Here, it is essential to differentiate between checking and creating a schedule. For now, we are only interested in the former and define the constraints such that they are independent of how the schedules were created. Ideally, we might use the same constraints, e.g., in an SMT solver, to also create the schedules. However, SMT-based schedulers are not always feasible for large systems since the scheduling problem is NP-complete (checking the correctness of an existing solution can be done in polynomial time). Please note that, in general, the number and size of task and VCPU segments, as well as the assignment of tasks to VCPUs and VCPUs to cores, are part of the scheduling problem and not fixed. In this section, we are only interested in the correctness constraints of an existing solution, and hence, we assume a given number of segments and a given assignment. We show how to solve the scheduling problem using an incremental SMT approach with a metaheuristic in Section 5.1 and simulating EDF in Section 5.2. Also, please note that we express all offsets in microticks on the respective node. For easier readability, we summarized the formal notations used in the correctness constraints in Table 1.

4.1 Task Correctness Constraints

We start by defining the correctness constraints of scheduled segments arising out of task requirements. A schedule for a given set of tasks is correct if all tasks' job segments fulfill basic constraints on their assigned resource. Firstly, job segments must comply with the limits set by release, deadline, and WCET of their tasks.

Constraint 1 (Release Time and Deadline)**.** *Given the virtualized distributed real-time system defined by the graph $G(\mathcal{V}, \mathcal{L})$, we have:*

$$\begin{aligned}&\forall n \in \mathcal{V}, \forall c \in \mathcal{C}^n, \forall \rho \in \mathcal{P}(c),\\ &\forall \tau_i \in \mathcal{T}(\rho), \forall J_{ij} \in \mathcal{J}_i, \forall x_{ijk} \in \mathcal{X}_{ij} :\\ &(x_{ijk}.\phi \cdot \mu_c \geq R_i) \wedge (x_{ijk}.\phi \cdot \mu_c + x_{ijk}.L \leq D_i)\end{aligned}$$

Each segment of a task job needs to be at least as long as its context-switching overhead since, otherwise, tasks do not progress. Additionally, the sum of the segment sizes of a job is at least as large as the worst-case execution time of the respective task, including the context-switching overhead times the number of segments.

Constraint 2 (Job Segment Size Constraint)**.** *Given the virtualized distributed real-time system defined by the graph $G(\mathcal{V}, \mathcal{L})$, we have:*

$$\begin{aligned}&\forall n \in \mathcal{V}, \forall c \in \mathcal{C}^n, \forall \rho \in \mathcal{P}(c),\\ &\forall \tau_i \in \mathcal{T}(\rho), \forall J_{i,j} \in \mathcal{J}_i :\\ &(\forall x_{i,j,k} \in \mathcal{X}_{ij} : x_{i,j,k}.L \geq \delta_\tau(c)) \wedge\\ &\left(\sum_{\forall x_{i,j,k} \in \mathcal{X}_{ij}} x_{i,j,k}.L \geq C_i + |\mathcal{X}_{i,j}| \cdot \delta_\tau(c) \right)\end{aligned}$$

Moreover, for any pair of tasks assigned to the same resource, their job segments' execution windows cannot overlap.

Constraint 3 (Task Overlapping Constraint)**.** *Given the virtualized distributed real-time system defined by the graph $G(\mathcal{V}, \mathcal{L})$, we have:*

$$\begin{aligned}&\forall n \in \mathcal{V}, \forall c \in \mathcal{C}^n, \forall \rho_1, \rho_2 \in \mathcal{P}(c),\\ &\forall \tau_i \in \mathcal{T}(\rho_1), \forall \tau_l \in \mathcal{T}(\rho_2), \forall J_{ij} \in \mathcal{J}_i, \forall J_{lm} \in \mathcal{J}_l,\\ &\forall x_{ijk} \in \mathcal{X}_{ij}, \forall x_{lmn} \in \mathcal{X}_{lm}, x_{ijk} \neq x_{lmn} :\\ &(x_{ijk}.\phi \cdot \mu_c + j \cdot T_i \geq x_{lmn}.\phi \cdot \mu_c + m \cdot T_l + x_{lmn}.L) \vee\\ &(x_{lmn}.\phi \cdot \mu_c + m \cdot T_l \geq x_{ijk}.\phi \cdot \mu_c + j \cdot T_i + x_{ijk}.L)\end{aligned}$$

We also have to make sure that all segments of a task are allocated to one and the same VCPU.

Constraint 4 (Unique Assignment Constraint)**.** *Let $\mathcal{P} = \bigcup_{\forall n \in \mathcal{V}, \forall c \in \mathcal{C}^n} \mathcal{P}(c)$ be the set of all TT VCPUs in the distributed system defined by the graph $G(\mathcal{V}, \mathcal{L})$, then we have:*

$$\bigcap_{\forall \rho \in \mathcal{P}} \mathcal{T}(\rho) = \emptyset.$$

Moreover, all segments of a task need to satisfy the affinity constraint, i.e., they need to run on a VCPU assigned to a core from the set of allowed cores.

Constraint 5 (Affinity Constraint)**.** *Given the virtualized distributed real-time system defined by the graph $G(\mathcal{V}, \mathcal{L})$, we have:*

$$\forall n \in \mathcal{V}, \forall c \in C^n, \forall \rho \in \mathcal{P}(c), \forall \tau_i \in \mathcal{T}(\rho) : c \in A_i$$

4.2 End-to-End Correctness Constraints

Tasks located on distinct nodes may communicate via streams over the TSN backbone yielding an end-to-end latency constraint that imposes a maximum delay between the start of a sender task and the completion time of a receiver task. Network nodes within a deterministic Ethernet network $G(\mathcal{V}, \mathcal{L})$ share a common synchronized *network time* characterized by its precision. The clock synchronization precision captures the maximum time difference between two nodes' local clocks [37]. Hence, the difference between the end of the execution of the receiving task's job and the beginning of the execution of the sending task's job is less or equal to the given end-to-end latency minus the clock synchronization precision Π.

Constraint 6 (End-to-End Constraint)**.** *Given the virtualized distributed real-time system defined by the graph $G(\mathcal{V}, \mathcal{L})$, we have:*

$$\begin{aligned}
&\forall a, b \in \mathcal{V}, \forall c_a \in C^a, \forall c_b \in C^b, \forall \rho_a \in \mathcal{P}(c_a), \forall \rho_b \in \mathcal{P}(c_b),\\
&\forall \tau_i \in \mathcal{T}(\rho_a), \tau_l \in \mathcal{T}(\rho_b) \text{ s.t. } \tau_i \xrightarrow{s_h} \tau_l, \forall J_{ij} \in \mathcal{J}_i, \forall J_{lm} \in \mathcal{J}_l :\\
&\max_{\forall x_{lmn} \in \mathcal{X}_{lm}} \{x_{lmn}.\phi \cdot \mu_{c_b} + x_{lmn}.L\} -\\
&\min_{\forall x_{ijk} \in \mathcal{X}_{ij}} \{x_{ijk}.\phi \cdot \mu_{c_a}\} \leq E(\tau_i \xrightarrow{s_h} \tau_j) - \Pi
\end{aligned}$$

Furthermore, we have to align the transmission schedule of the stream s_k along its route to the sender and receiver task schedule. Here, we assume that the frame is buffered in the network stack until the message is ready to be sent on the TSN network. The propagation delay Δ^{xy} describes the time it takes a frame to travel on the physical medium of a link from the source to the destination.

Constraint 7 (Task Alignment Constraint)**.** *Given the set of streams $\mathcal{S}$ and the virtualized distributed real-time system defined by the graph $G(\mathcal{V}, \mathcal{L})$, we have:*

$$\begin{aligned}
&\forall a, b \in \mathcal{V}, \forall c_a \in C^a, \forall c_b \in C^b, \forall \rho_a \in \mathcal{P}(c_a), \forall \rho_b \in \mathcal{P}(c_b),\\
&\forall \tau_i \in \mathcal{T}(\rho_a), \tau_j \in \mathcal{T}(\rho_b) \text{ s.t. } \tau_i \xrightarrow{s_h} \tau_j, \forall J_{ik} \in \mathcal{J}_i, \forall J_{jk} \in \mathcal{J}_j :\\
&\left(\max_{\forall x_{ikl} \in \mathcal{X}_{ik}} \{x_{ikl}.\phi \cdot \mu_{c_a} + x_{ikl}.L\} \leq \min_{\forall f^{ax}_{hkl} \in \mathcal{F}^{ax}_{hk}} \{f^{ax}_{hkl}.\phi \cdot \mu_a\} \right) \wedge\\
&\left(\min_{\forall x_{jkl} \in \mathcal{X}_{jk}} \{x_{jkl}.\phi \cdot \mu_{c_b}\} \geq \right.\\
&\left. \max_{\forall f^{yb}_{hkl} \in \mathcal{F}^{yb}_{hk}} \{f^{yb}_{hkl}.\phi \cdot \mu_y + f^{yb}_{hkl}.L\} + \Delta^{yb} + \Pi \right)
\end{aligned}$$

Essentially, all segments of the sender task (in any period instance) have to be scheduled (relative to the period) before the start of the transmission of the associated stream on the same node. Moreover, the start of any segment of the receiver task (in any job instance) has to be past the reception of the stream on the receiver node. Since the frame is scheduled on the last hop before the receiver node, we have to include the transmission duration and the precision between the receiver node and the last device (switch) on the stream's route before the receiver node. With the *min* and *max* operators, we express that the latest segment of the sender task in each period instance has to be scheduled before the earliest frame of the corresponding period instance of the stream. Similarly, the earliest segment of the receiver task in each period instance has to be scheduled after the latest frame of the corresponding period instance of the stream, also considering the transmission duration to the receiver node and the network precision. Since the sender task τ_i, receiver task τ_j, and stream s_h have the same period, we impose these constraints for the same period instances (jobs) via common subscripts, i.e., x_{ikl}, X_{jkl}, and frame instances f^{ax}_{hkl} on the sender node as well as f^{yb}_{hkl} on the last hop before the receiver node.

In Constraints 6 and 7 the two network nodes' clocks' granularity, which is given by their microtick μ_x, might differ. Hence, we scale the offset and the segment length by the microtick. In contrast, we do not scale the clock synchronization precision and the end-to-end latency requirement as they are given in the network time.

If the macrotick is a multiple of the microtick (and not as assumed so far, equal), we impose a macrotick constraint that forces the offset of tasks to be a multiple of the macrotick.

Constraint 8 (Macrotick Constraint)**.** *Given the virtualized distributed real-time system defined by the graph $G(\mathcal{V}, \mathcal{L})$, we have:*

$$\begin{aligned}
&\forall n \in \mathcal{V}, \forall c \in C^n, \forall \rho \in \mathcal{P}(c),\\
&\forall \tau_i \in \mathcal{T}(\rho), \forall J_{ij} \in \mathcal{J}_i, \forall x_{ijk} \in \mathcal{X}_{ij} :\\
&(x_{ijk}.\phi \cdot \mu_c) \bmod \eta_c = 0
\end{aligned}$$

Note that we define this constraint for task segments, but we can also apply it to VCPU segments and frame instances. However, to preserving readability, we do not explicitly add this constraint in the following sections and assume that $\forall n \in \mathcal{V} : \eta_n = \mu_n$.

4.3 System Virtualization Constraints

First, we need to ensure that no VCPU segments scheduled on the same physical core overlap in the temporal domain.

Constraint 9 (Virtual CPU Overlapping)**.** *Given the virtualized distributed real-time system defined by the graph $G(\mathcal{V}, \mathcal{L})$, we have:*

$$\begin{aligned}
&\forall n \in \mathcal{V}, \forall c \in C^n, \forall \rho_i, \rho_m \in \mathcal{P}(c),\\
&\forall z_{ij} \in \mathcal{Z}_i, \forall z_{mn} \in \mathcal{Z}_m, z_{ij} \neq z_{mn} :\\
&(z_{ij}.\phi \cdot \mu_c \geq z_{mn}.\phi \cdot \mu_c + z_{mn}.L) \vee\\
&(z_{mn}.\phi \cdot \mu_c \geq z_{ij}.\phi \cdot \mu_c + z_{ij}.L)
\end{aligned}$$

Next, we need to ensure that the length of a VCPU segment is greater than or equal to the sum of all the task segments that are scheduled within it, plus the overhead of switching between VCPUs on the hypervisor level. To define the constraint, we need a helper set O_{ij} that defines the set of task segments that fully overlap with the given VCPU segment z_{ij}. Hence, for a segment z_{ij}, belonging to VCPU ρ_i assigned to core c, we have:

$$\begin{aligned}
O_{ij} = \{x_{lmn} | &\forall \tau_l \in \mathcal{T}(\rho_i), \forall J_{lm} \in \mathcal{J}_l, \forall x_{lmn} \in \mathcal{X}_{lm} :\\
&(x_{lmn}.\phi \cdot \mu_c + m \cdot T_l \geq z_{ij}.\phi) \wedge\\
&(x_{lmn}.\phi \cdot \mu_c + m \cdot T_l + x_{lmn}.L \leq z_{ij}.\phi \cdot \mu_c + z_{ij}.L)\}.
\end{aligned}$$

Constraint 10 (Virtual CPU Size Constraint)**.** *Given the virtualized distributed real-time system defined by the graph $G(\mathcal{V}, \mathcal{L})$, we have:*

$$\forall n \in \mathcal{V}, \forall c \in \mathcal{C}^n, \forall \rho_i \in \mathcal{P}(c), \forall z_{ij} \in \mathcal{Z}_i : \\ z_{ij}.L \geq \delta_\rho(c) + \sum_{x_{lmn} \in O_{ij}} x_{lmn}.L$$

Finally, we need to ensure that all task segments that overlap with a given VCPU segment are assigned to that VCPU.

Constraint 11 (Virtual CPU Assignment)**.** *Given the virtualized distributed real-time system defined by the graph $G(\mathcal{V}, \mathcal{L})$, we have:*

$$\forall n \in \mathcal{V}, \forall c \in \mathcal{C}^n, \forall \rho_i \in \mathcal{P}(c), \forall z_{ij} \in \mathcal{Z}_i, \forall x_{lmn} \in O_{ij} : \\ \tau_l \in \mathcal{T}(\rho_i)$$

4.4 Network Constraints

4.4.1 Deterministic Ethernet Constraints. The necessary correctness constraints for scheduled traffic in IEEE 802.1Qbv, which were introduced in [13] only apply to a strictly periodic transmission of frames, i.e., each frame of a stream instance on a link has one offset and each frame instance is sent at the same time in each period instance. We use a more generic non-strictly periodic model and extend the correctness conditions from [13] to this model. We refer the reader to [13] for an in-depth explanation of the constraints and their relation to the IEEE 802.1Qbv timed-gate mechanism.

First, any frame's offset on a link must be within 0 and the period.

Constraint 12 (Frame Constraint)**.** *Given the set of streams $\mathcal{S}$ and the virtualized distributed real-time system defined by the graph $G(\mathcal{V}, \mathcal{L})$, we have:*

$$\forall s_i \in \mathcal{S}, \forall (x, y) \in s_i, \forall f_{ij}^{xy} \in \mathcal{F}_i^{xy} : \\ (f_{ij}^{xy}.\phi \cdot \mu_x \geq 0) \wedge (f_{ij}^{xy}.\phi \cdot \mu_x + f_{ij}^{xy}.L \leq T_i)$$

Frames of different streams and frames of the same stream are not allowed to overlap when transmitted on the same egress port. Hence, any two frames (generated by the same or different streams) are not allowed to overlap in any period instance until the hyperperiod of the two involved stream periods.

Constraint 13 (Link Constraint)**.** *Given the set of streams $\mathcal{S}$ and the virtualized distributed real-time system defined by the graph $G(\mathcal{V}, \mathcal{L})$, we have:*

$$\forall (x, y) \in \mathcal{L}, \forall s_i, s_l \in \mathcal{S} \text{ s.t. } (x, y) \in s_i \wedge (x, y) \in s_l, \\ \forall J_{ij}^{xy} \in \mathcal{J}_i^{xy}, \forall J_{lm}^{xy} \in \mathcal{J}_l^{xy}, \forall f_{ijk}^{xy} \in \mathcal{F}_{ij}^{xy}, \forall f_{lmn}^{xy} \in \mathcal{F}_{lm}^{xy}, f_{ijk}^{xy} \neq f_{lmn}^{xy} : \\ \left(f_{ijk}^{xy}.\phi \cdot \mu_x + j \cdot T_i \geq f_{lmn}^{xy}.\phi \cdot \mu_x + m \cdot T_l + f_{lmn}^{xy}.L\right) \vee \\ \left(f_{lmn}^{xy}.\phi \cdot \mu_x + m \cdot T_l \geq f_{ijk}^{xy}.\phi \cdot \mu_x + j \cdot T_i + f_{ijk}^{xy}.L\right)$$

Next, we have to ensure that a flow's transmission on a path from sender to receiver must follow the sequential order of its frames, i.e., a frame can only be scheduled for transmission on a subsequent link after the previous link has fully received it. The granularity of a node $x \in \mathcal{V}$ clock is termed its *macrotick* μ_x. When defining constraints crossing several network links, we must consider the network time precision and nodes' macroticks.

Constraint 14 (Flow Transmission Constraint)**.** *Given the set of streams $\mathcal{S}$ and the virtualized distributed real-time system defined by the graph $G(\mathcal{V}, \mathcal{L})$, we have:*

$$\forall s_i \in \mathcal{S}, \forall (a, x), (x, b) \in s_i, \\ \forall J_{ij}^{ax} \in \mathcal{J}_i^{ax}, \forall J_{ij}^{xb} \in \mathcal{J}_i^{xb}, \forall f_{ijk}^{ax} \in \mathcal{F}_{ij}^{ax}, \forall f_{ijk}^{xb} \in \mathcal{F}_{ij}^{xb} : \\ f_{ijk}^{xb}.\phi \cdot \mu_x - \Delta^{xb} - \Pi \geq f_{ijk}^{ax}.\phi \cdot \mu_a + f_{ij}^{ax}.L$$

4.4.2 IEEE 802.1Qbv Constraints. In contrast to TTEthernet, an 802.1Qbv stream can consist of multiple frames. The number of egress queues is hardware-dependent, and 802.1Qbv streams share an egress queue with the same priority. In [13], the authors propose a *flow isolation constraint* to prevent jitter due to the interleaving of frames from different streams and discuss the ramifications concerning increased performance yet reduced solution space compared to a *frame isolation constraint*. Hence, we extend the frame isolation constraint from [13]. For a stream $s_i \in \mathcal{S}$ and on link $(a, b) \in s_i$, let P_i^{ab} denote the port on node $a \in \mathcal{V}$ that connects to node $b \in \mathcal{V}$ on which the stream transmits the frames. As before, we generalize the constraint from [13] to our non-strictly periodic model.

Constraint 15 (Frame Isolation Constraint)**.** *Given the set of streams $\mathcal{S}$ and the virtualized distributed real-time system defined by the graph $G(\mathcal{V}, \mathcal{L})$, we have:*

$$\forall (x, y) \in \mathcal{L}, \forall s_i, s_l \in \mathcal{S} \text{ s.t. } (x, y) \in s_i \wedge (x, y) \in s_l, i \neq l, \\ \forall J_{ij}^{xy} \in \mathcal{J}_i^{xy}, \forall J_{lm}^{xy} \in \mathcal{J}_l^{xy}, \forall f_{ijk}^{xy} \in \mathcal{F}_{ij}^{xy}, \forall f_{lmn}^{xy} \in \mathcal{F}_{lm}^{xy} : \\ \left(P_i^{ab} \neq P_j^{ab}\right) \vee \\ \Big(\left(f_{lmn}^{ab}.\phi \cdot \mu_a + m \cdot T_l + \Pi \leq f_{ijk}^{xa}.\phi \cdot \mu_x + j \cdot T_i + \Delta^{xa}\right) \vee \\ \left(f_{ijk}^{ab}.\phi \cdot \mu_a + j \cdot T_i + \Pi \leq f_{lmn}^{ya} \cdot \mu_y + m \cdot T_l + \Delta^{ya}\right)\Big)$$

If the underlying network backbone is TTEthernet instead of TSN, this constraint is not needed and all other constraints can be applied as defined above (see also the comparison between the correctness constraints of TSN and TTEthernet in [15]).

5 OFFLINE SCHEDULING

We can reduce the scheduling problem to placing and dimensioning boxes on a two-dimensional plane, where one axis represents a discrete time base with origin zero and the other axis a countable set of computing and networking resources. The resulting decision problem, i.e., if for a given set of tasks executing on VCPUs communicating via streams, there exists a schedule that fulfills Constraints 1 to 15, is NP-complete. In many cases the scheduling problem also faces optimization criteria, i.e., to minimize end-to-end latency or to reduce context-switching overhead. Specifically, we are interested in schedules minimizing VCPU context-switching time [51]. Furthermore, there is an interdependency between scheduling and resource assignment. For this work, we focus on scheduling virtualized distributed real-time systems thus we presume that the assignment of tasks to VCPUs and VCPUs to CPU cores is given whereas the number and size of task and VCPU segments is determined by the scheduler. We present an incremental SMT-based

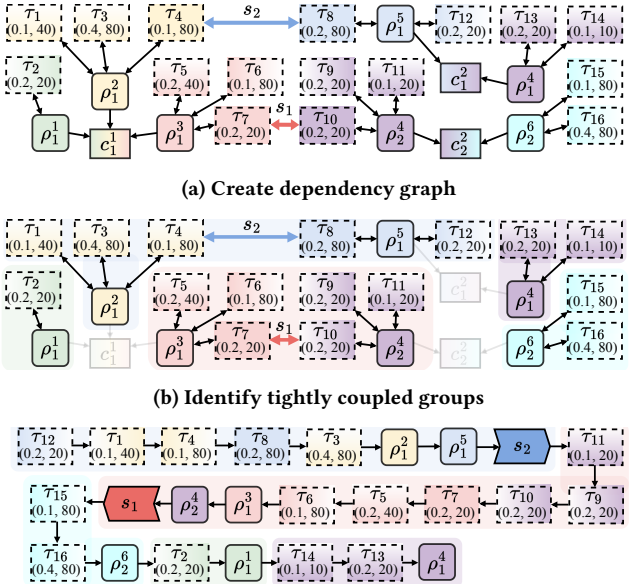
scheduler (iSMT) that implements Constraints 1 to 15 and minimizes virtualization overhead yet does not scale with increasing system size. In response, we explore a more efficient approach that uses iSMT to schedule the network and simulates Earliest-Deadline-First (EDF) at the expense of increased virtualization overhead.

5.1 Incremental SMT-based Scheduler

Satisfiability Modulo Theories (SMT) determine the satisfiability of first-order logic formulas under a selected a background theory [17]. In the case of our scheduling constraints, we use linear integer arithmetic $(\mathbb{Z}, +, \geq)$ as the background theory. For a given set of constraints formulated in $(\mathbb{Z}, +, \geq)$, an SMT solver yields a *model*, i.e., a variable assignment that satisfies the formulas if a solution exists or reports their unsatisfiability. However, even though SMT concerning $(\mathbb{Z}, +, \geq)$ is decidable, it is NP-hard, so an efficient algorithm to solve arbitrary sets of constraints is not known. Steiner [61] first applied SMT to the scheduling of TTEthernet and proposed an incremental approach, successively adding constraints of a variable-sized subset of frames to the solver's context. If the solver returns a model for a subset of frames, the incremental scheduler fixes the frames in the TT schedule. Otherwise, the scheduler backtraces, increasing the size of the subset of frames. This has shown a substantial runtime improvement in the average case [61].

Extending the network scheduling problem to consider task-level and system virtualization requirements to facilitate the integration of network and processor-level schedules in virtualized system architectures complicates incremental solving substantially. Firstly, including task-level and system virtualization requirements naturally increases the constraints required to solve the scheduling problem. Furthermore, combined stream and task-level requirements and logically grouping tasks on VCPUs introduce dependencies. Suppose the scheduler *sequentially* adds streams and dependent tasks to the SMT context. In that case, the solver cannot consider dependent constraints when instantiating variables, e.g., the interdependence between the last job segment offset and size of a sender task and the transmission offset of a stream's first frame (c.f. Constraint 7). This results in augmented backtracking since the solver extends the increment until it includes all dependent stream and task constraints, potentially resulting in an inefficient increment that includes all stream and task constraints. Additionally, integrating system virtualization constraints introduces further dependencies between sender and receiver tasks as well as tasks that are not participating in a stream by logically grouping them on VCPUs. As a result, *the order* in which the scheduler adds groups of schedulable entities (tasks, VCPUs, and streams) to the solver's context is crucial to avoid inefficient backtracking.

Given a virtualized distributed real-time system $G(\mathcal{V}, \mathcal{L})$ and a set of schedulable entities $\Sigma = \mathcal{T} \bigcup \mathcal{P} \bigcup \mathcal{S}$, the iSMT scheduler produces job segment offsets and sizes for schedulable entities if a schedule exists or the empty set if it cannot find a solution. The execution of the iSMT scheduler is split into two phases. First, we create a dependency graph to identify tightly coupled groups of schedulable entities and derive an order in which groups are incrementally added to the SMT context. Afterward, the main scheduler loop generates constraints for a given set of groups, freezing found



(a) Create dependency graph

(b) Identify tightly coupled groups

(c) Sort tightly coupled schedulable groups

Figure 2: Dependency analysis

offsets if the increment is satisfiable, or backtracking if it is unsatisfiable. In the main scheduler loop, we apply a meta-heuristic to minimize the number of VCPU preemptions.

5.1.1 Dependency Analysis. In Figure 2 we illustrate process of dependency analysis given a task set $\mathcal{T} := \{\tau_1, ..., \tau_{16}\}$ and 2 streams $\mathcal{S} := \{s_1, s_2\}$ being scheduled on 2 nodes with a total of 3 cores with six VMs and VCPUs $\mathcal{P} := \{\rho_1^1, \rho_1^2, \rho_1^3, \rho_1^4, \rho_2^4 \rho_1^5, \rho_1^6\}$ (c.f. Figure 1).

Initially, the scheduler constructs a dependency graph differentiating between *schedulable entity dependencies* and *resource dependencies* as visualized in Figure 2a. A schedulable entity dependency is a bidirectional dependency between two schedulables so that the placement and dimensioning of their jobs' segments are reciprocal. Therefore, if the scheduler adds a pair of dependent schedulable entities to the SMT context in differing increments, we see excessive backtracking since the placement and dimensioning of the first added schedulable entity does not consider the subsequently added entity's constraints. For instance in Figure 2a, stream s_1 from τ_7 to τ_{10} or the assignment of τ_1 to VCPU ρ_1^2 constitute schedulable entity dependencies. Further, a VCPU assigned to a processor core is a resource dependency. Resource dependencies are unidirectional such as VCPU ρ_1^2 executing on core c_1^1 as we can see in Figure 2a. Note that only considering schedulable entity dependencies, clusters of tightly coupled schedulables emerge as shown in Figure 2b, i.e., schedulables whose placement and dimensioning are interdependent, so the scheduler must add them to the SMT context together to prevent backtracking. Lastly, the scheduler sorts the tightly coupled schedulable groups in descending order by their number of streams, the additive inverse of the utilization without overhead, and the additive inverse of the minimal schedulable period within the group yielding an ordered set $\mathcal{H} := \{\{\tau_{12}, \tau_1, ..., \rho_1^5, s_2\}, ..., \{\tau_{14}, \tau_{13}, \rho_1^4\}\}$ (c.f. Figure 2c). As a result, the scheduler solves tightly coupled

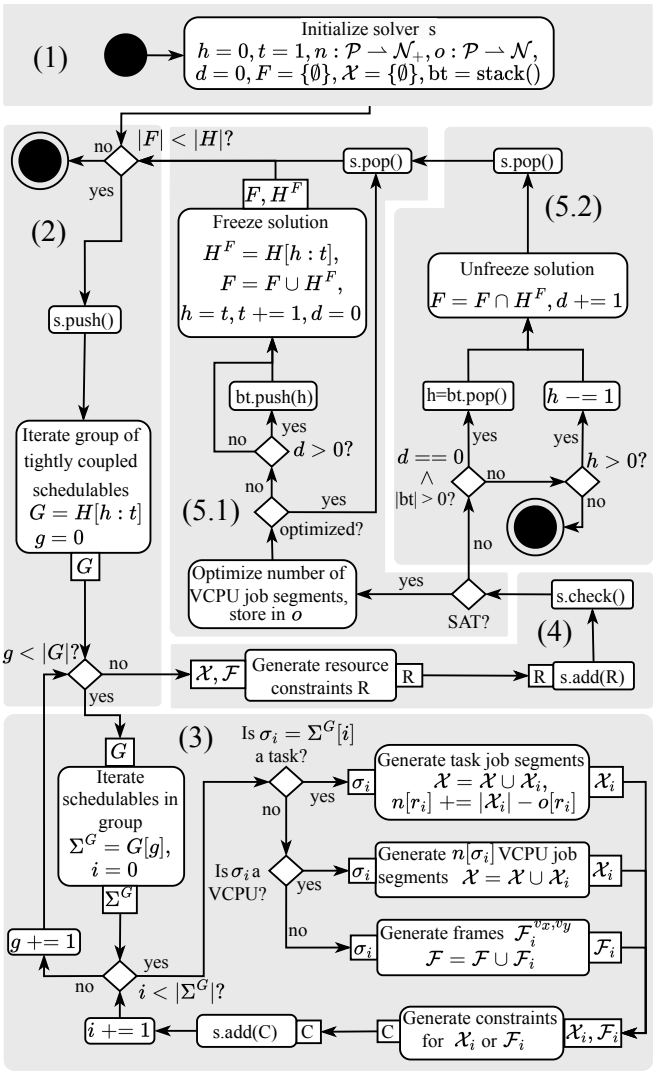


Figure 3: Activity diagram of the incremental scheduler

groups with many streams, contributing to high utilization of computational resources while resources are still sparsely populated.

5.1.2 Incremental Solving with Meta-Heuristics. The sorted set of tightly coupled schedulable groups acts as an input to the iSMT scheduler. We denote the ordered set of schedulable groups as hot schedulable groups $\mathcal{H}$. Figure 3 depicts the scheduling algorithm:

(1) We initialize the internal state: the set of frozen schedulable groups $F = \{\emptyset\}$, a head h and tail t to index the hot schedulable groups, the current backtracking distance d and stack bt, and 2 dictionaries n and o storing for each VCPU the number of required VCPU segments and the optimization summand, respectively.

(2) The central scheduler loop executes as long as the set of frozen schedulable groups is smaller than the set of hot schedulable groups $|F| < |H|$. The head h and tail t determine the size of the increment, i.e., the tightly coupled schedulable groups solved within the same SMT context. For instance, as long as there was no backtracking it is $t - h = 1$ and the central scheduler loop picks a single group to add to the SMT context at a time.

(3) Each group of tightly coupled schedulables $G[g]$ with $g \in \{h, ..., t-1\}$ is sorted first by type (task, VCPU, or stream), and then by period, so that within a group high-frequency tasks are processed first and low-frequency streams last (Figure 2c). The scheduler iterates a group's schedulables $\Sigma^{\mathcal{G}}$ and generates job segment constraints differentiating tasks, VCPUs, and streams. The scheduler keeps track of the number of task segments generated per VCPU in the dictionary n. Hence, once the scheduler processes a group's VCPUs, for each VCPU $\rho = \sigma_i$, the maximum number of required VCPU segments is known and the scheduler generates $n[\rho] - o[\rho]$ job segments for VCPU ρ. If a schedulable is a stream $s = \sigma_i$, for each link $(v_x, v_y) \in s$ we generate frames $\mathcal{F}_i = \{\mathcal{F}_i^{(v_x,v_y)} \mid \forall (v_x, v_y) \in s\}$. The scheduler uses the generated segments and frames to synthesize constraints for the SMT context.

(4) After generating schedulable constraints, the scheduler iterates system resources, including CPU cores, VCPUs, and links, using the set of job segments $\mathcal{X}$ and frames $\mathcal{F}$ to generate and add resource constraints and check the satisfiability of the resulting SMT context.

(5.1) If the solver finds a solution, it optimizes the number of required VCPU segments since it initially generated constraints assuming the maximum number of job segments per VCPU. There are two possible optimizations. Firstly, the SMT solver might return VCPU segments not containing any job segments since it grouped job segments sharing a VCPU together. Secondly, the SMT solver can place two segments belonging to the same VCPU in direct succession so that the resulting VCPU preemption is dispensable. Therefore, for each VCPU ρ_i the scheduler counts the number of empty and consecutive thus redundant VCPU segments, and stores the optimization summand in $o[\rho_i]$. Suppose the scheduler has detected optimizations for at least one VCPU. In that case, it pops the corresponding SMT context and reiterates the same group of tightly coupled schedulables, optimizing the number of VCPU job segments. Once the scheduler cannot further decrease the number of VCPU job segments, it pushes the current SMT context on the solver's stack, freezes the found variable assignments, and adjusts h and t to index the next hot group of tightly coupled schedulables.

(5.2) When the SMT context is unsatisfiable, the scheduler backtracks. If the backtracking distance is zero and there is a head on the backtracking stack, the scheduler restores the head from the backtracking stack and unfreezes the corresponding tightly coupled schedulable groups. If the backtracking distance is greater than 0 or the backtracking stack is empty, but the scheduler can further backtrack, i.e. the head is not at index 0, we decrease the head, and proceed as before with unfreezing the corresponding tightly coupled schedulable groups. If $h = 0$, backtracking has included all tightly coupled schedulable groups in a single SMT context, and the solver found no solution so the scheduler returns an empty set.

Note that iSMT initially generates one segment per task job in scheduler Step (3). To simplify the scheduler's visualization in Figure 3, we omit a subroutine in Step (5.2) that increases the number of generated task segments if backtracking completes without finding a solution ($h = 0$) and reiterates the central loop trying to find a schedule generating more than one segment per task job.

5.2 Simulating Earliest-Deadline-First

The iSMT scheduler aims to minimize the number of VCPU preemptions by starting the search with the minimal number of task segments and the maximum number of required VCPU segments, successively decreasing the number of VCPU segments to cluster task segments sharing a VCPU. However, the iSMT scheduler's runtime scales badly with increasing system size (c.f. Section 6). Therefore, we inroduce iSMT-EDF that invokes iSMT to reserve stream offsets, simulates EDF to schedule tasks (similar to [14]), and applies a simple heuristic to dimension and place VCPU segments.

First, given a virtualized distributed real-time system $G(\mathcal{V}, \mathcal{L})$, a set of tasks $\mathcal{T}$, and a set of streams $\mathcal{S}$, iSMT-EDF isolates all dependent tasks $\tau_i, \tau_j \in \mathcal{T}' \subset \mathcal{T}$ with $\tau_i \xrightarrow{s_k} \tau_j$ for $s_k \in \mathcal{S}$. The subset of dependent tasks $\mathcal{T}'$ yields a set of VCPUs $\mathcal{P}'$ so that $\mathcal{T}' = \bigcup_{\forall \rho \in \mathcal{P}'} \mathcal{T}(\rho)$, i.e., the VCPUs in $\mathcal{P}'$ contain only the communicating tasks $\mathcal{T}'$. The resulting set of schedulable entities $\Sigma = \mathcal{T}' \cup \mathcal{P}' \cup \mathcal{S}$ acts as input to the iSMT scheduler, yielding a network schedule and offsets and sizes for task job segments, considering the presence of VCPUs. Subsequently, these offsets and sizes of communicating task job segments identified by iSMT limit release times and deadlines of communicating tasks in the EDF simulation so that they adhere to the corresponding network schedule. We simulate EDF with a configurable microtick, i.e., there is a trade-off between schedulability and runtime for small versus large microticks. Similarly to iSMT, the EDF simulation assumes one VCPU segment per job segment of a task. Hence, for a task $\tau_i \in \mathcal{T}(\rho)$ executing in a VCPU $\rho \in \mathcal{P}(c)$ on a core $c \in \mathcal{C}^n$, in the case of a context switch, the simulation accounts for the task's context-switching overhead $\delta_\tau(c)$ and, if the subsequent task $\tau_j \in \mathcal{T}(\rho')$ executes on a different VCPU, i.e., $\rho \neq \rho'$, also for the VCPU's context-switching overhead $\delta_\rho(c)$. It is $\mathcal{X}(z_{ij}) = \{x^1, ..., x^m\}$ an ordered set of m consecutive job segments sharing a segment $z_{ij} \in \mathcal{Z}_i$ of a VCPU $\rho_i \in \mathcal{P}(c)$, i.e., from $x^1.\phi + x^1.L = x^2.\phi$ to $x^{m-1}.\phi + x^{m-1}.L = x^m.\phi$. Then, VCPU segment z_{ij}'s offset and size are given by $z_{ij}.\phi = x^1.\phi - \delta_\rho(c)$ and $z_{ij}.L = x^m.\phi + x^m.L$ respectively.

Contrary to iSMT, which initially presumes the maximum number of VCPU segments and optimizes in successive sweeps, EDF-SMT trades subpar VCPU segment placement and dimensioning resulting in an expected higher VCPU preemption overhead of the resulting schedule for improved runtime and scalability.

6 EXPERIMENTS

We compare iSMT and iSMT-EDF concerning their attained schedulability, schedule synthesis runtime, and resulting VCPU context-switching overhead varying the system size, i.e., number of nodes, switches, and flows, and the target task utilization. With both schedulers, iSMT and iSMT-EDF, we intend to demonstrate the applicability of our formal system and scheduling model to synthesize global schedules incorporating tasks, streams, and VCPUs with end-to-end latency guarantees. Conversely, we do not claim that iSMT and iSMT-EDF solve the global scheduling problem efficiently. Indeed, a schedule synthesis runtime of several hours for single problem instances delimited the total number of data points we could generate. We countered high synthesis runtimes by solving up to 14 instances in parallel on an AMD Ryzen 7 2700X with 8 physical cores and two logical cores at 2.2 GHz each, accessing 64 GiB of main memory. The scheduler times out after 2.5 h and reports the subset of scheduled tasks, VCPUs, and stream at that instant.

Table 2: Overview of problem instances' dimensions.

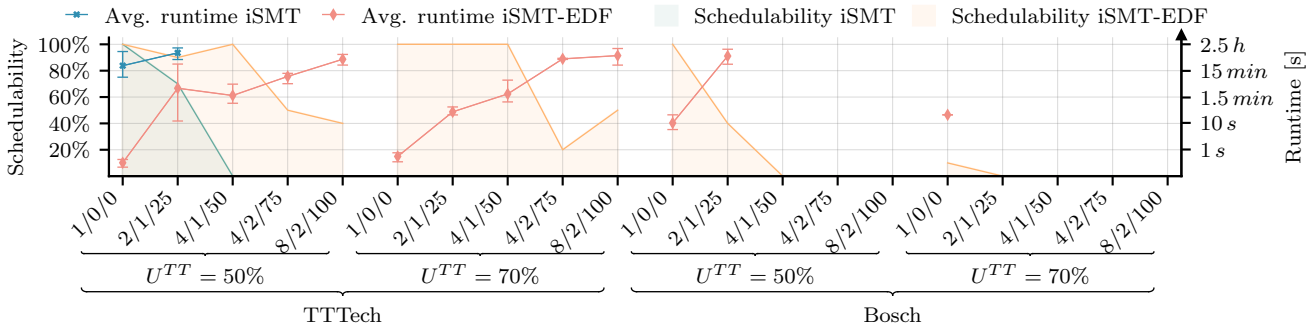
System Params.	Util.	System Size	num. tasks (min., max.)	num. VCPUs (min., max.)
Bosch	50.0%	1/0/0	173, 230	101, 150
		2/1/25	341, 467	219, 289
		4/1/50	732, 874	463, 563
		4/2/75	732, 874	463, 563
		8/2/100	1520, 1766	925, 1103
	70.0%	1/0/0	232, 313	127, 182
		2/1/25	481, 619	248, 354
		4/1/50	1030, 1193	555, 652
		4/2/75	1030, 1193	555, 652
		8/2/100	2233, 2396	1151, 1290
TTTech	50.0%	1/0/0	241, 295	127, 157
		2/1/25	470, 564	251, 322
		4/1/50	1008, 1128	545, 637
		4/2/75	1008, 1128	545, 637
		8/2/100	2089, 2182	1111, 1255
	70.0%	1/0/0	329, 403	147, 196
		2/1/25	679, 803	299, 379
		4/1/50	1400, 1525	649, 754
		4/2/75	1400, 1525	649, 754
		8/2/100	2843, 3048	1309, 1474

We presume a homogeneous distributed system for all experiments in which each node comes with four CPU cores. Accordingly, we infer WCETs relative to the CPU speed. We used realistic values $\delta_\tau(c) = 10\,\mu s$ and $\delta_\rho(c) = 30\,\mu s$ for the task and VCPU context-switching times across all problem instances to evaluate the VCPU context-switching time overhead of resulting schedules.

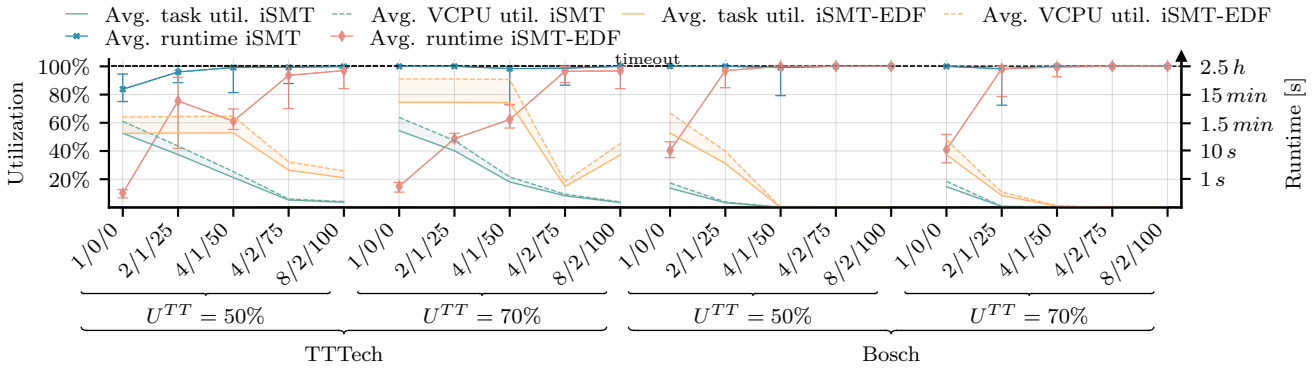
6.1 Increasing System Size

First, we evaluate the effect of increasing system size on the schedulability, schedule synthesis runtime, and VCPU context-switch overhead for two sets of real-world system properties named *Bosch* [38] and *TTTech* [22] benchmarks. We consider 5 system sizes identified by the number of nodes, number of TSN switches, and number of streams. For each system size (#nodes/#switches/#streams) and benchmark, we generated 10 problem instances.

We randomly generate between 64 and 128 VMs with a random number of VCPUs for each host node $n \in \mathcal{V} = \{1, ..., \#nodes\}$ with $|C^n| = 4$ cores. To generate random task sets per CPU core, we use task characteristics as found in automotive applications [22, 38]. For the Bosch benchmark, we randomly select task periods from a set $T^1 = \{1, 2, 5, 10, 20, 50, 100, 200, 1000\}\ ms$ with distribution $P^1 = \{0.03, 0.02, 0.02, 0.25, 0.25, 0.03, 0.2, 0.01, 0.04\}$. Given a task's random period $T_i \in T^1$ at index i, to determine its WCET, we select its corresponding average-case execution time at index i from the set $ACETs = \{5, 4.2, 11.040, 10.090, 8.740, 17.560, 10.530, 2.560, 0.430\}\mu s$ and its minimum and maximum WCET factors f_{min} and f_{max} from the sets $F_{min} = \{1.3, 1.54, 1.13, 1.06, 1.06, 1.13, 1.02, 1.03, 1.84\}$ and $F_{max} = \{29.11, 19.04, 18.44, 30.03, 15.61, 7.76, 8.88, 4.9, 4.75\}$, respectively. Then, we pick a random WCET factor $f \in [f_{min}, f_{max}]$



(a) Schedulability and schedulable instances' runtimes.



(b) Fraction of scheduled tasks and VCPUs within timeout of 2.5 *h*.

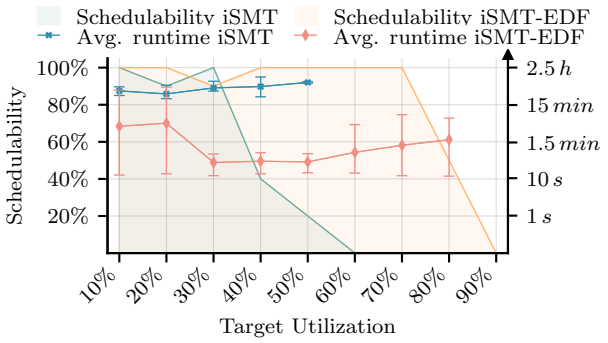
Figure 4: Results for different system sizes (#nodes/#switches/#streams) and sets of problem instances (TTTech, Bosch), with 10 μs microtick, and periods $T_i \in$ Bosch = $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ *ms*, $T_i \in$ TTTech = $\{5, 10, 20, 40, 80\}$ *ms*.

and calculate the task's WCET $C_i = f \cdot ACETs[i]$. We assign each generated task to a VCPU of that core, and the procedure repeats until it reaches a target utilization, not including context-switching overheads. For the TSN backbone, we assume that each communicating VM can directly access a physical network interface card (NIC) connected to a TSN switch. If the TSN backbone consists of more than one switch, the switches form a mesh network. To generate communication streams, we identify pairs of tasks with the same period assigned to VCPUs located on different nodes. Then, we randomly pick a pair of tasks as the source and sink of a stream and select its data size from the set $D = \{1, 2, 4, 8, 16, 32, 64, 3000\}$ byte with size distribution $\{0.35, 0.49, 0.13, 0.008, 0.013, 0.005, 0.002, 0.002\}$ that again is representative of automotive applications [38]. We assume that a stream's end-to-end latency equals its period.

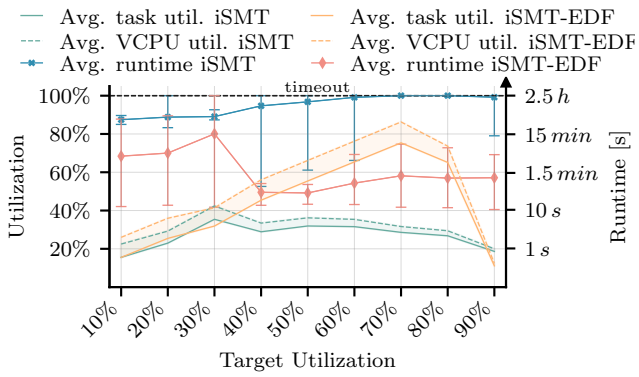
For the TTTech benchmark, we replace the periods and distribution used in the Bosch benchmark with the ones described in [22], i.e., we randomly select task periods from $T^2 = \{5, 10, 20, 40, 80\}$ *ms* with distribution $P^2 = \{0.09166, 0.2666, 0.125, 0.19166, 0.325\}$ and prune the *ACET*, F_{min}, and F_{max} sets so that their dimensions align, i.e., $ACET' = \{11.040, 10.090, 8.740, 17.560, 10.530\}\mu s \subset ACET$, $F'_{min} = \{1.13, 1.06, 1.06, 1.13, 1.02\} \subset F_{min}$, and $F'_{max} = \{18.44, 30.03, 15.61, 7.76, 8.88\} \subset F_{max}$. Table 2 summarizes the minimum and maximum number of tasks and VCPUs generated across the 10 instances for each benchmark, target problem utilization, and system size.

Figure 4 shows the schedulability or attained utilization [%] on the left y-axes and the schedule synthesis runtime on the logarithmic right y-axes. We see that in the case of the TTTech benchmark, all problem instances generated for systems without network communication (1/0/0) and 50% task target utilization, are schedulable with both schedulers (c.f. Figure 4a). However, iSMT-EDF (0.32 ± 0.07 *s*) outperforms iSMT (1413.30 ± 1275.95 *s*) in terms of schedule synthesis runtime. For a system with 25 streams and a task target utilization of 50%, iSMT still schedules 7 out 10 TTTech problem instances within 2.5 *h* whereas iSMT-EDF successfully schedules 90% of TTTech problem instances in 200.73 ± 533.86 *s*. For systems exceeding 25 streams, iSMT times out on all TTTech instances whereas iSMT-EDF retains 100%, 50%, and 40% schedulability for system sizes 4/1/50, 4/2/75, and 8/2/100, respectively, with exponentially increasing synthesis runtime. Note that for the biggest system size 8/2/100 and 50% target utilization, iSMT-EDF schedules TTTech problem instances featuring between 2089 and 2182 tasks and 1151 and 1290 VCPUs (c.f. Table 2). For the TTTech benchmark with 70% target utilization, iSMT consistently times out whereas iSMT-EDF exhibits comparable scalability of schedulability and synthesis runtime as for a 50% target utilization.

In the case of the Bosch benchmark, iSMT times out regardless of the system size whereas iSMT-EDF successfully schedules



(a) Schedulability and schedulable instances' runtimes.



(b) Fraction of scheduled tasks and VCPUs within 2.5 *h*timeout.

Figure 5: Fixed system size (2/1/25) and increasing target utilization, 10 μs microtick, periods $T_i \in$ TTTech = $\{5, 10, 20, 40, 80\}$ ms.

all instances without streams and 4 out of 10 for a target utilization of 50%. For the Bosch benchmark with a target utilization of 70% and system size 1/0/0, iSMT-EDF cannot find a solution for any problem instance except for a single 1/0/0 instance. For the remaining instances with a target utilization of 70%, iSMT-EDF times out. Therefore, in Figure 4b, we show the effective average task and VCPU utilization iSMT and iSMT-EDF realize within 2.5 h runtime per problem instance. Considering the attained average task utilization of both schedulers, the Bosch benchmark uncovers a common weakness of our approach: Scheduling the network using incremental SMT solving. The Bosch benchmark consistently yields problem instances with the maximum possible hyperperiod of 1 s and high-frequency tasks and streams of more than 200 Hz. Thus, even for small system sizes, Bosch problem instances result in over 10^5 constraints in iSMT-EDF's SMT context compared to about 10^4 for TTTech problem instances. Ultimately, the SMT solver in both schedulers poses the bottleneck for more efficient schedule synthesis and in future work we aim to implement a fully heuristic search that does not use any SMT solver.

Lastly, Figure 4b shows the attained task utilization as well as VCPU utilization for both schedulers. The gap between both lines indicates the VCPU context-switching overhead. As expected, we see that for feasible problem instances, iSMT provides schedules with less VCPU context-switching overhead than iSMT-EDF. In numbers, for a target utilization of 50% in the case of schedulable system sizes 1/0/0 and 2/1/25 of the TTTech benchmark, iSMT yields an average VCPU context-switching overhead of 8.4%. Contrary, iSMT-EDF yields an average VCPU context-switching overhead of 11.7% across all system sizes of the TTTech benchmark for a target utilization of 50% and 15.8% with a target utilization of 70%. For the iSMT-EDF and the Bosch benchmarks, we find an average VCPU context-switching overhead of 14.3% for the schedulable problem instances with a system size of 1/0/0 and 2/1/25 and with a target utilization set to 50%.

6.2 Increasing Utilization

In the second set of experiments, we retain a system size of 2/1/25 and generate a task set of size 480, i.e., 60 tasks and 32 VCPUs per core, but increase the target task utilization to evaluate its effect on the schedulability, schedule synthesis runtime, and VCPU context-switching overhead. We use the TTTech benchmark but generate task WCETs using the algorithm from [18, 19]. We omit the Bosch benchmark as initial experiments lead to time-outs for system size 2/1/25, similar to the previous experiments.

Figure 5 shows the schedulability, attained utilization, and runtime for the target utilization increasing from 10% to 90%. From 10% to 30% target utilization iSMT and iSMT-EDF achieve comparable schedulability with lower synthesis runtime of iSMT-EDF compared to iSMT (c.f. Figure 5a). Starting with a target utilization of 40% iSMT times out for 4/10 problem instances, declining to 2/10 for a target utilization of 50%, and always timing out for a target utilization exceeding 50%. Considering the attained utilization in Figure 5b, we can see that iSMT consistently achieves about 30% task utilization within 2.5 h synthesis runtime. In contrast, iSMT-EDF can schedule all generated instances up to 70% target utilization and 50% of problem instances for 80% target utilization with an average runtime below 90 s. For schedulable problem instances from 10% to 30% target utilization, iSMT yields an average VCPU context-switching overhead of 6.8%. In comparison, for schedulable problem instances from 10% to 80% target utilization, iSMT-EDF yields an average VCPU context-switching overhead of 10.7%.

7 CONCLUSION

In this paper we have studied distributed safety-critical applications in which real-time tasks execute in a virtualized environment on multi-core multi-SoC platforms and communicate critical messages over a deterministic communication backbone. We have defined correctness constraints for the schedulability of distributed virtualized safety-critical systems that use time-triggered scheduling on the computation, the communication, and the virtualization layers. Furthermore, we presented a formal system and scheduling model to harmonize these different layers of the global scheduling problem. We also introduced two offline algorithms to generate TT schedules for all layers that guarantee end-to-end latency requirements. The two schedulers offer trade-offs between schedule synthesis runtime and virtualization overhead. We have used synthetic benchmarks derived from real-world systems to show the scalability and schedulability of our offline scheduling approaches.

REFERENCES

[1] Luca Abeni and Giorgio C. Buttazzo. 2004. Resource Reservation in Dynamic Real-Time Systems. *Real Time Syst.* 27, 2 (2004), 123–167. https://doi.org/10.1023/B:TIME.0000027934.77900.22

[2] Hamidreza Ahmadian, Roman Obermaisser, and Jon Perez. 2018. *Distributed Real-Time Architecture for Mixed-Criticality Systems.* CRC Press.

[3] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I. Davis. 2022. A comprehensive survey of industry practice in real-time systems. *Real Time Syst.* 58, 3 (2022), 358–398. https://doi.org/10.1007/S11241-021-09376-1

[4] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. 2009. Satisfiability Modulo Theories. In *Handbook of Satisfiability*, Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh (Eds.). Frontiers in Artificial Intelligence and Applications, Vol. 185. IOS Press, 825–885. https://doi.org/10.3233/978-1-58603-929-5-825

[5] Sanjoy K. Baruah and Gerhard Fohler. 2011. Certification-Cognizant Time-Triggered Scheduling of Mixed-Criticality Systems. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS 2011, Vienna, Austria, November 29 - December 2, 2011*. IEEE Computer Society, 3–12. https://doi.org/10.1109/RTSS.2011.9

[6] Mohammadreza Barzegaran and Paul Pop. 2023. The FORA European Training Network on Fog Computing for Robotics and Industrial Automation. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2023, Antwerp, Belgium, April 17-19, 2023*. IEEE, 1–6. https://doi.org/10.23919/DATE56975.2023.10137067

[7] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. 2016. Synthesizing Job-Level Dependencies for Automotive Multi-rate Effect Chains. In *22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2016, Daegu, South Korea, August 17-19, 2016*. IEEE Computer Society, 159–169. https://doi.org/10.1109/RTCSA.2016.41

[8] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. 2017. End-to-end timing analysis of cause-effect chains in automotive embedded systems. *J. Syst. Archit.* 80 (2017), 104–113. https://doi.org/10.1016/J.SYSARC.2017.09.004

[9] Timothy Broomhead, Laurence Cremean, Julien Ridoux, and Darryl Veitch. 2010. Virtualize Everything but Time. In *9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings*, Remzi H. Arpaci-Dusseau and Brad Chen (Eds.). USENIX Association, 451–464. http://www.usenix.org/events/osdi10/tech/full_papers/Broomhead.pdf

[10] Zhuo Cheng, Jinyun Xue, Haitao Zhang, Zhen You, Qimin Hu, and Yuto Lim. 2020. Scheduling Heterogeneous Multiprocessor Real-Time Systems with Mixed Sets of Task. In *14th IEEE International Conference on Service Oriented Systems Engineering, SOSE 2020, Oxford, UK, August 3-6, 2020*. IEEE, 72–81. https://doi.org/10.1109/SOSE49046.2020.00016

[11] Silviu S. Craciunas and Ramon Serna Oliver. 2016. Combined task- and network-level scheduling for distributed time-triggered systems. *Real Time Systems* 52, 2 (2016), 161–200. https://doi.org/10.1007/S11241-015-9244-X

[12] Silviu S. Craciunas and Ramon Serna Oliver. 2021. Out-of-sync Schedule Robustness for Time-sensitive Networks. In *17th IEEE International Conference on Factory Communication Systems, WFCS 2021, Linz, Austria, June 9-11, 2021*. IEEE, 75–82. https://doi.org/10.1109/WFCS46889.2021.9483602

[13] Silviu S. Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. 2016. Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, October 19-21, 2016*, Alain Plantec, Frank Singhoff, Sébastien Faucou, and Luís Miguel Pinho (Eds.). ACM, 183–192. https://doi.org/10.1145/2997465.2997470

[14] Silviu S. Craciunas, Ramon Serna Oliver, and Valentin Ecker. 2014. 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 2014, Barcelona, Spain, September 16-19, 2014*, Antoni Grau and Herminio Martínez (Eds.). IEEE, 1–8. https://doi.org/10.1109/ETFA.2014.7005128

[15] Silviu S. Craciunas and Ramon Serna Oliver. 2017. An Overview of Scheduling Mechanisms for Time-sensitive Networks. Technical report, Real-time summer school L'École d'Été Temps Réel (ETR).

[16] Alfons Crespo, Ismael Ripoll, and Miguel Masmano. 2010. Partitioned Embedded Architecture Based on Hypervisor: The XtratuM Approach. In *Eighth European Dependable Computing Conference, EDCC-8 2010, Valencia, Spain, 28-30 April 2010*. IEEE Computer Society, 67–72. https://doi.org/10.1109/EDCC.2010.18

[17] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2011. Satisfiability modulo theories: introduction and applications. *Commun. ACM* 54, 9 (2011), 69–77. https://doi.org/10.1145/1995376.1995394

[18] Paul Emberson. [n. d.]. TaskGen v1.0. https://github.com/jlelli/taskgen.

[19] Paul Emberson, Roger Stafford, and Robert I Davis. 2010. Techniques for the synthesis of multiprocessor tasksets. In *Proceedings of the 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*. 6–11.

[20] Rolf Ernst, Stefan Kuntz, Sophie Quinton, and Martin Simons. 2018. The Logical Execution Time Paradigm: New Perspectives for Multicore Systems (Dagstuhl Seminar 18092). *Dagstuhl Reports* 8, 2 (2018), 122–149. https://doi.org/10.4230/DAGREP.8.2.122

[21] Xiang Feng and A.K. Mok. 2002. A model of hierarchical real-time virtual resources. In *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.* 26–35. https://doi.org/10.1109/REAL.2002.1181559

[22] Anaïs Finzi, Silviu S. Craciunas, and Marc Boyer. 2024. Integrating Sporadic Events in Time-triggered Systems via Affine Envelope Approximations. In *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 15–28. https://doi.org/10.1109/RTAS61025.2024.00010

[23] Tom Fleming, Sanjoy K. Baruah, and Alan Burns. 2016. Improving the Schedulability of Mixed Criticality Cyclic Executives via Limited Task Splitting. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, October 19-21, 2016*, Alain Plantec, Frank Singhoff, Sébastien Faucou, and Luís Miguel Pinho (Eds.). ACM, 277–286. https://doi.org/10.1145/2997465.2997492

[24] Robert Kaiser Rudolf Fuchsen. 2005. Method for distributing computing time in a computer system. Patent No. 20090210879A1, Filed November 11st., 2005, Issued Aug. 20th., 2009.

[25] Gautam Gala, Javier Castillo Rivera, and Gerhard Fohler. 2021. Work-in-Progress: Cloud Computing for Time-Triggered Safety-Critical Systems. In *42nd IEEE Real-Time Systems Symposium, RTSS 2021, Dortmund, Germany, December 7-10, 2021*. IEEE, 516–519. https://doi.org/10.1109/RTSS52674.2021.00054

[26] Gernot Heiser and Ben Leslie. 2010. The OKL4 microvisor: convergence point of microkernels and hypervisors. In *Proceedings of the 1st ACM SIGCOMM Asia-Pacific Workshop on Systems, ApSys 2010, New Delhi, India, August 30, 2010*, Chandramohan A. Thekkath, Ramakrishna Kotla, and Lidong Zhou (Eds.). ACM, 19–24. https://doi.org/10.1145/1851276.1851282

[27] IEEE. 2016. 802.1Qbv - Enhancements for Scheduled Traffic. https://standards.ieee.org/standard/802_1Qbv-2015.html. Accessed: 21.05.2024.

[28] IEEE. 2016. Official Website of the 802.1 Time-Sensitive Networking Task Group. http://www.ieee802.org/1/pages/tsn.html. Accessed: 21.05.2024.

[29] IEEE. 2020. IEEE Std 802.1AS-2020. https://standards.ieee.org/standard/802_1AS-2020.html. Accessed: 21.05.2024.

[30] Haris Isakovic and Radu Grosu. 2016. A heterogeneous time-triggered architecture on a hybrid system-on-a-chip platform. In *25th IEEE International Symposium on Industrial Electronics, ISIE 2016, Santa Clara, CA, USA, June 8-10, 2016*. IEEE, 244–253. https://doi.org/10.1109/ISIE.2016.7744897

[31] Damir Isovic and Gerhard Fohler. 2009. Handling mixed sets of tasks in combined offline and online scheduled real-time systems. *Real Time Syst.* 43, 3 (2009), 296–325. https://doi.org/10.1007/S11241-009-9088-3

[32] Jan Jatzkowski, Marcio Kreutz, and Achim Rettberg. 2017. Hierarchical Multicore-Scheduling for Virtualization of Dependent Real-Time Systems. In *System Level Design from HW/SW to Memory for Embedded Systems*, Marcelo Götz, Gunar Schirner, Marco Aurélio Wehrmeister, Mohammad Abdullah Al Faruque, and Achim Rettberg (Eds.). Springer International Publishing, Cham, 103–115.

[33] Robert Kaiser. 2008. Alternatives for scheduling virtual machines in real-time embedded systems. In *Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems, IIES '08, Glasgow, Scotland, April 1, 2008*, Michael Engel and Olaf Spinczyk (Eds.). ACM, 5–10. https://doi.org/10.1145/1435458.1435460

[34] Timo Kerstan, Daniel Baldin, and Stefan Groesbrink. 2010. Full virtualization of real-time systems by temporal partitioning. In *International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*. 24.

[35] Hermann Kopetz. 1992. Sparse Time versus Dense Time in Distributed Real-Time Systems. In *Proceedings of the 12th International Conference on Distributed Computing Systems, Yokohama, Japan, June 9-12, 1992*. IEEE Computer Society, 460–467. https://doi.org/10.1109/ICDCS.1992.235008

[36] Hermann Kopetz and Günter Grünsteidl. 1993. TTP - A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems. In *Digest of Papers: FTCS-23, The Twenty-Third Annual International Symposium on Fault-Tolerant Computing, Toulouse, France, June 22-24, 1993*. IEEE Computer Society, 524–533. https://doi.org/10.1109/FTCS.1993.627355

[37] Hermann Kopetz and Wilfried Steiner. 2022. *Real-Time Systems - Design Principles for Distributed Embedded Applications, Third Edition*. Springer. https://doi.org/10.1007/978-3-031-11992-7

[38] Simon Kramer, Dirk Ziegenbein, and Arne Hamann. 2015. Real world automotive benchmarks for free. In *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, Vol. 130.

[39] Jaewoo Lee, Sisu Xi, Sanjian Chen, Linh T.X. Phan, Chris Gill, Insup Lee, Chenyang Lu, and Oleg Sokolsky. 2012. Realizing Compositional Scheduling through Virtualization. In *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*. 13–22. https://doi.org/10.1109/RTAS.2012.20

[40] Giuseppe Lipari and Enrico Bini. 2005. A methodology for designing hierarchical scheduling systems. *J. Embedded Comput.* 1, 2 (apr 2005), 257–269.

[41] Martin Lukasiewycz, Reinhard Schneider, Dip Goswami, and Samarjit Chakraborty. 2012. Modular scheduling of distributed heterogeneous time-triggered automotive systems. In *Proceedings of the 17th Asia and South Pacific*

Design Automation Conference, ASP-DAC 2012, Sydney, Australia, January 30 - February 2, 2012. IEEE, 665–670. https://doi.org/10.1109/ASPDAC.2012.6165039

[42] Martin Lukasiewycz, Reinhard Schneider, Dip Goswami, and Samarjit Chakraborty. 2012. Modular scheduling of distributed heterogeneous time-triggered automotive systems. In *Proceedings of the 17th Asia and South Pacific Design Automation Conference, ASP-DAC 2012, Sydney, Australia, January 30 - February 2, 2012*. IEEE, 665–670. https://doi.org/10.1109/ASPDAC.2012.6165039

[43] S.P. Marimuthu and S.C. Chakraborty. 2006. A Framework for Compositional and Hierarchical Real-Time Scheduling. In *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)*. 91–96. https://doi.org/10.1109/RTCSA.2006.7

[44] Shane D. McLean, Emil A. Juul Hansen, Paul Pop, and Silviu S. Craciunas. 2022. Configuring ADAS Platforms for Automotive Applications Using Metaheuristics. *Frontiers in Robotics and AI* 8 (2022), 353. https://doi.org/10.3389/frobt.2021.762227

[45] Ayhan Mehmed, Wilfried Steiner, and Maximilian Rosenblattl. 2017. A Time-Triggered Middleware for Safety-Critical Automotive Applications. In *Ada-Europe*.

[46] Carlo Meroni, Silviu S. Craciunas, Anaïs Finzi, and Paul Pop. 2023. Mapping and Integration of Event- and Time-triggered Real-time Tasks on Partitioned Multi-core Systems. In *28th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2023, Sinaia, Romania, September 12-15, 2023*. IEEE, 1–8. https://doi.org/10.1109/ETFA54631.2023.10275547

[47] Anna Minaeva and Zdenek Hanzálek. 2022. Survey on Periodic Scheduling for Time-triggered Hard Real-time Systems. *ACM Comput. Surv.* 54, 1 (2022), 23:1–23:32. https://doi.org/10.1145/3431232

[48] Aloysius K. Mok, Alex Xiang Feng, and Deji Chen. 2001. Resource Partition for Real-Time Systems. In *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium (RTAS 2001), 30 May - 1 June 2001, Taipei, Taiwan*. IEEE Computer Society, 75–84. https://doi.org/10.1109/RTTAS.2001.929867

[49] Andreas Motzkus and Mehmet Oezer. 2016. *PikeOS Safe Real-Time Scheduling: Adaptive Time-Partitioning Scheduler for EN 50128 certified Multi-Core Platforms*. Technical Report. SYSGO. 6 pages. https://www.sysgo.com/whitepapers

[50] Roman Obermaisser. 2009. Time-Triggered Communication. In *Networked Embedded Systems - Volume 2 of the Embedded Systems Handbook*, Richard Zurawski (Ed.). CRC Press, 14. https://doi.org/10.1201/9781439807620.CH14

[51] Ramon Serna Oliver and Silviu S. Craciunas. 2016. Hierarchical scheduling over off- and on-chip deterministic networks. *SIGBED Rev.* 13, 4 (2016), 14–19. https://doi.org/10.1145/3015037.3015039

[52] Ramon Serna Oliver, Silviu S. Craciunas, and Wilfried Steiner. 2018. IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding. In *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2018, 11-13 April 2018, Porto, Portugal*, Rodolfo Pellizzoni (Ed.). IEEE Computer Society, 13–24. https://doi.org/10.1109/RTAS.2018.00008

[53] Traian Pop, Petru Eles, and Zebo Peng. 2002. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Proceedings of the Tenth International Symposium on Hardware/Software Codesign, CODES 2002, Estes Park, Colorado, USA, May 6-8, 2002*, Jörg Henkel, Xiaobo Sharon Hu, Rajesh Gupta, and Sri Parameswaran (Eds.). ACM, 187–192. https://doi.org/10.1145/774789.774828

[54] Traian Pop, Petru Eles, and Zebo Peng. 2003. Schedulability Analysis for Distributed Heterogeneous Time/Event Triggered Real-Time Systems. In *15th Euromicro Conference on Real-Time Systems (ECRTS 2003), 2-4 July 2003, Porto, Portugal, Proceedings*. IEEE Computer Society, 257–266. https://doi.org/10.1109/EMRTS.2003.1212751

[55] Traian Pop, Paul Pop, Petru Eles, Zebo Peng, and Alexandru Andrei. 2006. Timing Analysis of the FlexRay Communication Protocol. In *18th Euromicro Conference on Real-Time Systems, ECRTS'06, 5-7 July 2006, Dresden, Germany, Proceedings*. IEEE Computer Society, 203–216. https://doi.org/10.1109/ECRTS.2006.31

[56] Jan Ruh, Wilfried Steiner, and Gerhard Fohler. 2021. Clock Synchronization in Virtualized Distributed Real-Time Systems Using IEEE 802.1AS and ACRN. *IEEE Access* 9 (2021), 126075–126094. https://doi.org/10.1109/ACCESS.2021.3111045

[57] Jan Ruh, Wilfried Steiner, and Gerhard Fohler. 2023. IEEE 802.1AS Multi-Domain Aggregation for Virtualized Distributed Real-Time Systems. In *53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2023 - Supplemental Volume, Porto, Portugal, June 27-30, 2023*. IEEE, 70–76. https://doi.org/10.1109/DSN-S58398.2023.00027

[58] Florian Sagstetter, Sidharta Andalam, Peter Waszecki, Martin Lukasiewycz, Hauke Stähle, Samarjit Chakraborty, and Alois C. Knoll. 2014. Schedule Integration Framework for Time-Triggered Automotive Architectures. In *The 51st Annual Design Automation Conference 2014, DAC '14, San Francisco, CA, USA, June 1-5, 2014*. ACM, 20:1–20:6. https://doi.org/10.1145/2593069.2593211

[59] Insik Shin and Insup Lee. 2003. Periodic resource model for compositional real-time guarantees. In *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003.* 2–13. https://doi.org/10.1109/REAL.2003.1253249

[60] Insik Shin and Insup Lee. 2004. Compositional real-time scheduling framework. In *25th IEEE International Real-Time Systems Symposium*. 57–67. https://doi.org/10.1109/REAL.2004.15

[61] Wilfried Steiner. 2010. An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks. In *Proceedings of the 31st IEEE Real-Time Systems Symposium, RTSS 2010, San Diego, California, USA, November 30 - December 3, 2010*. IEEE Computer Society, 375–384. https://doi.org/10.1109/RTSS.2010.25

[62] Wilfried Steiner, Günther Bauer, Brendan Hall, and Michael Paulitsch. 2011. TTEthernet: Time-Triggered Ethernet. In *Time-Triggered Comm.* CRC Press.

[63] Henrik Theiling. 2013. *PikeOS and Time-Triggering*. Technical Report. SYSGO. 6 pages. https://www.sysgo.com/whitepapers

[64] Jia Xu and David Lorge Parnas. 2000. Priority Scheduling Versus Pre-Run-Time Scheduling. *Real Time Syst.* 18, 1 (2000), 7–23. https://doi.org/10.1023/A:1008198310125

[65] Jungwoo Yang, Hyungseok Kim, Sangwon Park, Changki Hong, and Insik Shin. 2011. Implementation of compositional scheduling framework on virtualization. *SIGBED Rev.* 8, 1 (mar 2011), 30–37. https://doi.org/10.1145/1967021.1967025

[66] Licong Zhang, Dip Goswami, Reinhard Schneider, and Samarjit Chakraborty. 2014. Task- and network-level schedule co-synthesis of Ethernet-based time-triggered systems. In *19th Asia and South Pacific Design Automation Conference, ASP-DAC 2014, Singapore, January 20-23, 2014*. IEEE, 119–124. https://doi.org/10.1109/ASPDAC.2014.6742876