SMT-based Task- and Network-level Static Schedule Generation for Time-Triggered Networked Systems[*]

Silviu S. Craciunas
TTTech Computertechnik AG
scr@tttech.com

Ramon Serna Oliver
TTTech Computertechnik AG
rse@tttech.com

ABSTRACT

In Ethernet-based time-triggered networks, like TTEthernet, a global communication scheme, for which the schedule synthesis is known to be an NP-complete problem, establishes contention-free windows for the exchange of messages with guaranteed low latency and minimal jitter. However, in order to achieve end-to-end determinism at the application level, software tasks running on the end-system nodes need to obey a similar execution scheme with tight dependencies towards the network domain. In this paper we address the simultaneous co-synthesis of network as well as application schedules for preemptive time-triggered tasks communicating in a switched multi-speed time-triggered network. We use Satisfiability Modulo Theories (SMT) to formulate the scheduling constraints and solve the resulting problem using a state-of-the-art SMT solver. Furthermore, we introduce a novel incremental scheduling approach, based on the demand bound test for asynchronous constrained-deadline periodic tasks, which significantly improves scalability for the average case without sacrificing schedulability. We demonstrate the performance of our approach using synthetic network topologies and system configurations.

1. INTRODUCTION

Ethernet-based time-triggered networks, like TTEthernet [30] (SAE AS6802 [14]), enable the integration of mixed-criticality communicating systems requiring different degrees of determinism in scalable, switched network topologies. TTEthernet is used for safety-critical real-time applications within aerospace and industrial domains allowing critical traffic with guaranteed end-to-end latency (time-triggered communication) to co-exist with flexible non-critical rate-constrained and best-effort traffic. Examples of time-triggered networks successfully deployed include the TTP-based communication systems for the flight control computer of Embraer's Legacy 450 and 500 jets and the distributed electric and environmental control of the Boeing 787 Dreamliner, while technologies like TTEthernet are part of the NASA Orion Multi-Purpose Crew Vehicle scheduled for the first test flight later this year [12].

In TTEthernet the transmission of time-triggered messages (i.e. *tt-messages*) is statically configured at precise instants of time. A global communication scheme, the *tt-network-schedule*, defines transmission and reception time windows for each frame being transmitted, received, or forwarded by any end-system or switch within the multi-hop communication path. The tt-network-schedule accounts for end-to-end latency, message length, as well as additional constraints derived from other used resources, e.g., message buffers. At run-time, a network-wide time synchronization protocol guarantees the cyclic execution of the schedule with sub-microsecond precision [15, p. 186].

The production and consumption of data payload transported by tt-messages is often performed by periodic software tasks (i.e. tt-tasks) following a similar time-triggered scheme (through static table-driven scheduling) on the end-system CPUs[1]. The end-to-end latency is then subject to both scheduling domains: on the one hand the distributed network schedule, *tt-network-schedule*; and on the other hand the multiple dependent end-system schedules, *tt-task-schedules*. The composition of the two scheduling domains is crucial to extend the high end-to-end deterministic guarantees of the network layer to include the application level.

In this paper, we consider multi-hop fully switched TTEthernet networks, such as the one depicted in Figure 1, in which different end-systems execute software tasks communicating via the time-triggered traffic class of TTEthernet. Separate sequential schedule synthesis, either by scheduling the network first and using the result as input for the task schedule synthesis [7], or the complementary approach [11], does not cover the whole solution space. Considering tasks and messages as part of the same scheduling problem enables an exhaustive search of the whole solution space guaranteeing that, if a feasible schedule exists, it will be found.

In this respect, we extend the time-triggered paradigm to include the application layer on end-systems providing a complete end-to-end deterministic chain. Our approach extends previous work by Steiner [28] by considering the co-generation of static tt-network- and tt-task-schedules for preemptable interdependent application tasks with arbitrary

[*]The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 610640 (DREAMS).

RTNS 2014 , October 8 - 10 2014, Versailles, France
Copyright 2014 ACM 978-1-4503-2727-5/14/10 ...$15.00.
http://dx.doi.org/10.1145/2659787.2659812.

[1]We will use task and tt-task as well as message and tt-message interchangeably in this paper.

communication periods over a multi-speed switched TTEthernet network. We model the CPUs as *self-links* on the end-systems and schedule virtual frames (explained below) representing non-preemptable chunks of preemptable tt-tasks. With this abstraction we define a set of general constraints, allowing a Satisfiability Modulo Theories (SMT) solver to synthesize a system-wide tt-schedule. Moreover, we present a novel incremental approach for creating tt-schedules, based on the utilization demand bound analysis for asynchronous tasks [3] scheduled using the earliest deadline first (EDF) algorithm [19], which significantly improves our scalability factor. We evaluate our approach using synthetic workloads and show that our algorithm scales for medium to large networked systems achieving guaranteed high determinism and minimal jitter.

In Section 2, we define the networked system model, which we later use in Section 3 to construct logical constraints that correctly describe a combined network and task schedule. In Section 4 we introduce two scheduling algorithms based on SMT, which we then evaluate using synthetic benchmarks in Section 5. We present related research in Section 6 and conclude the paper in Section 7.

2. NETWORKED SYSTEM MODEL

TTEthernet establishes a multi-hop layer 2 switched network via *full-duplex* multi-speed links (e.g. 100 Mbit/s, 1 Gbit/s, etc.). We formally model the networked system as a directed graph $G(\mathcal{V}, \mathcal{L})$. The set of vertexes ($\mathcal{V}$) comprises the set of communication nodes (switches and end-systems) while the edges ($\mathcal{L} \subseteq \mathcal{V} \times \mathcal{V}$) represent the directional communication links connecting the nodes. Since we consider bi-directional physical links, we have that $\forall [v_a, v_b] \in \mathcal{L} \Rightarrow [v_b, v_a] \in \mathcal{L}$, where $[v_a, v_b]$ is an ordered tuple that represents a directed logical network link connecting vertex $v_a \in \mathcal{V}$ to $v_b \in \mathcal{V}$. In addition to the network links, we also consider tasks running on the end-system CPUs. We model the CPU as a directional *self-link*, which we call CPU link, connecting an end-system vertex with itself.

A network or CPU link $[v_a, v_b]$ between nodes $v_a \in \mathcal{V}$ and $v_b \in \mathcal{V}$ is defined by the tuple

$$\langle [v_a, v_b].s, [v_a, v_b].d, [v_a, v_b].mt, [v_a, v_b].b \rangle,$$

where $[v_a, v_b].s$ is the speed coefficient, $[v_a, v_b].d$ is the link delay, $[v_a, v_b].mt$ is the macrotick, and $[v_a, v_b].b$ is the maximum buffer constraint. In the case of a CPU link, the macrotick represents the granularity of the time-line that the real-time operating system (RTOS) of the respective end-system recognizes. A typical macrotick for a time-triggered RTOS ranges from a few hundreds of microseconds to several milliseconds [4, p. 266]. In the case of a network link the macrotick is the time-line granularity of the physical link. Typically, the TTEthernet time granularity is around $60ns$ [16] but larger values are commonly used. The link delay refers to either the propagation and processing delay on the medium in case of a network link or the queuing and software overhead for a CPU link. The speed coefficient is used for calculating the transmission time of the frame on a particular physical link based on its size and the link speed, –for a network link the speed coefficient represents the time it takes to transmit one byte. Considering the minimum and maximum frame sizes in the Ethernet protocol of 84 and 1542 bytes, respectively, the frame transmission time, for example, on a 1Gbit/sec link would be $0.672\mu sec$ and

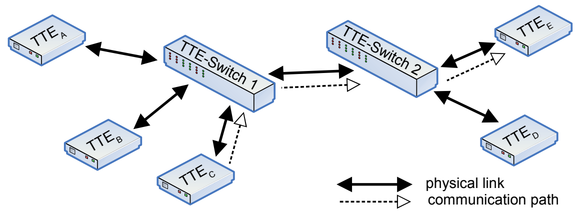


Figure 1: A TTEthernet network with 5 end-systems and 2 switches.

$12.336\mu sec$, respectively. For a CPU link, the speed coefficient is used to allow heterogeneous CPUs with different clock rates, resulting in different WCETs for the same task. The maximum buffer constraint represents the maximum amount of time a frame of any size can reside in the message queues of the transmitting node (switch or end-system).

We denote the set of all tt-tasks in the system by Γ. A tt-task $\tau_i^{v_a} \in \Gamma$ running on the end-system v_a is defined, similar to the periodic task model from [19], by the tuple

$$\langle \tau_i^{v_a}.\phi, \tau_i^{v_a}.C, \tau_i^{v_a}.D, \tau_i^{v_a}.T \rangle,$$

where $\tau_i^{v_a}.\phi$ is the offset, $\tau_i^{v_a}.C$ is the WCET, $\tau_i^{v_a}.D$ is the relative deadline, and $\tau_i^{v_a}.T$ is the period of the task. Note that, a tt-task is pre-assigned to one end-system CPU and does not migrate during run-time. Hence, all task parameters are scaled according to the macrotick and speed of the respective CPU link. We denote the set of all tasks that run on end-system v_a by Γ^{v_a}.

We model communication through the network via the concept of virtual links, where a virtual link is a logical data-flow path in the network from one sender node to one receiver node over which a tt-message is transmitted. The concept is similar to [1] but extended to include the tt-tasks associated with the generation and consumption of the data running at the end-systems. We distinguish three types of tt-tasks, namely, *producer*, *consumer*, and *free* tt-tasks. Producer tasks generate messages that are being sent on the network, consumer tasks receive messages that arrive from the network, and free tasks have no dependency towards the network. Note that we assume the actual instant of sending and receiving messages occurs at the end and at the beginning of producer and consumer tasks, respectively. Schedulability may improve by considering the exact moment in the execution of a task where the message is sent or received (cf. [9]).

A virtual link $vl_i \in \mathcal{VL}$ from a producer task running on end-system v_a to a consumer task running on end-system v_b, routed through the nodes (i.e. switches) $v_1, v_2, \ldots, v_{n-1}, v_n$ is expressed, similar to [28], as

$$vl_i = [[v_a, v_a], [v_a, v_1], [v_1, v_2], \ldots, [v_{n-1}, v_n], [v_n, v_b], [v_b, v_b]].$$

Additionally $vl_i.max\_latency$ denotes the maximum allowed latency between the start of the producer task and the end of the consumer task. For communicating tasks, a virtual link is composed by the path through the network and the two end-system CPU links $[v_a, v_a]$ and $[v_b, v_b]$. For a free task $\tau_i^{v_a} \in \Gamma$, a virtual link vl_i is created with $vl_i = [[v_a, v_a]]$.

Our goal is to schedule virtual links considering the task- and network-levels combined. Hence, we take both the tt-message that is sent over the network and the computation time of both producer and consumer tt-tasks and unify these through the concept of frames.

Let $\mathcal{M}$ denote the set of all tt-message in the system. We model a tt-message $m_i \in \mathcal{M}$ associated with the virtual link vl_i by the tuple $\langle T_i, L_i \rangle$, where T_i is the period and L_i is the size in bytes. For the network links, a frame is understood as the instance of a tt-message scheduled on a particular link. For CPU links, we model tasks as a set of sequential virtual frames that are transmitted (or scheduled) on the respective CPU link. Since we consider preemptive tasks, we generate as many virtual frames as are needed for a fully preemptive task based on the CPU macrotick and speed. We defined $\tau_i^{v_a}.C$ to be the WCET of the task scaled according to the macrotick and speed of the CPU link it is running on. Therefore we have $\tau_i^{v_a}.C$ non-preemptable chunks (or virtual frames) of a task $\tau_i^{v_a}$.

In order to generalize frames scheduled on physical links and virtual frames scheduled on CPU links we say that a virtual link vl_i will generate sets of frames on every link (CPU or network) along the communication path. In the case of a network link the set will contain only one element, which is the (non-preemptable) frame instance of tt-message m_i on the respective link, whereas in the case of a CPU link the cardinality of the set will be given by the computation time of the task generating the virtual frames. We denote the ordered set of all frames $f_{i,j}^{[v_a,v_b]}$ of virtual link vl_i scheduled on a (CPU or network) link $[v_a, v_b]$ by $\mathcal{F}_i^{[v_a,v_b]}$, the ordering being done by frame offset. Furthermore, we denote the first and last frame of the set $\mathcal{F}_i^{[v_a,v_b]}$ with $f_{i,1}^{[v_a,v_b]}$ and $last(\mathcal{F}_i^{[v_a,v_b]})$, respectively.

We use a similar notation to [28] to model frames. A frame $f_{i,j}^{[v_a,v_b]} \in \mathcal{F}_i^{[v_a,v_b]}$ is defined by the tuple

$$\langle f_{i,j}^{[v_a,v_b]}.\phi, f_{i,j}^{[v_a,v_b]}.T, f_{i,j}^{[v_a,v_b]}.L \rangle,$$

where $f_{i,j}^{[v_a,v_b]}.\phi$ is the offset in macroticks of the frame on link $[v_a, v_b]$, $f_{i,j}^{[v_a,v_b]}.T$ is the period of the frame in macroticks, and $f_{i,j}^{[v_a,v_b]}.L$ is the duration of the frame in macroticks. For a network link we have

$$f_{i,1}^{[v_a,v_b]}.T = \left\lceil \frac{T_i}{[v_a, v_b].mt} \right\rceil, f_{i,1}^{[v_a,v_b]}.L = \left\lceil \frac{L_i \times [v_a, v_b].s}{[v_a, v_b].mt} \right\rceil.$$

A tt-task $\tau_i^{v_l} \in \Gamma$ yields a set of frames $f_{i,j}^{[v_l,v_l]}, j = 1, 2, \ldots, \tau_i^{v_l}.C$, where each frame has size 1 and a period equal to the scaled task period, i.e., $f_{i,j}^{[v_l,v_l]}.T = \lceil \frac{\tau_i^{v_l}.T}{[v_l,v_l].mt} \rceil$.

Note that it is possible in our model to specify that some (or all) tasks are nonpreemptive by allowing a task to generate a single frame with length equal to the WCET of the task. Our approach implicitly supports nonpreemptive task schedule synthesis as it is a subproblem of preemptive task schedule synthesis.

3. SCHEDULING CONSTRAINTS

Creating static time-triggered tt-schedules for networked systems, like the one described in this paper, generally reduces to solving a set of timing constraints. In this section we construct, based on our system model, the mandatory constraints to correctly schedule, in the time-triggered sense, both tt-tasks and tt-messages. Some of our constraints (namely those in Sections 3.2, 3.3, 3.4, and 3.8) are similar to the contention-free, path-dependent, end-to-end transmission, and memory constraints from [28] but generalized according to our system definition to include virtual frames generated by tasks, different macrotick granularity, and different link speeds.

3.1 Frame constraints

For any frame scheduled on either a network or CPU link, the offset cannot take any negative values or any value that would result in the scheduling window exceeding the frame period. Therefore, we have

$$\forall vl_i \in \mathcal{VL}, \forall [v_a, v_b] \in vl_i, \forall f_{i,j}^{[v_a,v_b]} \in \mathcal{F}_i^{[v_a,v_b]} :$$
$$\left(f_{i,j}^{[v_a,v_b]}.\phi \geq 0\right) \wedge \left(f_{i,j}^{[v_a,v_b]}.\phi \leq f_{i,j}^{[v_a,v_b]}.T - f_{i,j}^{[v_a,v_b]}.L\right).$$

The constraint restricts the offset of a frame to be inside the period of the generating tt-task or tt-message and, moreover, to have a value for which the whole frame fits inside the said period.

3.2 Link constraints

The most essential constraint that needs to be fulfilled for time-triggered networks is that no two frames that are transmitted on the same link are in contention, i.e., they do not overlap in the time domain. Similarly, for CPU links, no tasks running on the same CPU may overlap in the time domain, i.e., no two chunks from any task may be scheduled at the same time. Given two frames, $f_{i,j}^{[v_a,v_b]}$ and $f_{k,l}^{[v_a,v_b]}$, that are scheduled on the same link $[v_a, v_b]$ we need to specify constraints such that the frames cannot overlap in any period instance. Let $\mathcal{F}$ denote the set of all frames in the system.

$$\forall [v_a, v_b] \in \mathcal{L}, \forall \mathcal{F}_i^{[v_a,v_b]}, \mathcal{F}_k^{[v_a,v_b]} \subset \mathcal{F},$$
$$\forall f_{i,j}^{[v_a,v_b]} \in \mathcal{F}_i^{[v_a,v_b]}, \forall f_{k,l}^{[v_a,v_b]} \in \mathcal{F}_k^{[v_a,v_b]},$$
$$\forall \alpha \in \left[0, \frac{HP_{i,j}^{k,l}}{f_{i,j}^{[v_a,v_b]}.T} - 1\right], \forall \beta \in \left[0, \frac{HP_{i,j}^{k,l}}{f_{k,l}^{[v_a,v_b]}.T} - 1\right]:$$
$$\Big(f_{i,j}^{[v_a,v_b]}.\phi + \alpha \times f_{i,j}^{[v_a,v_b]}.T \geq$$
$$f_{k,l}^{[v_a,v_b]}.\phi + \beta \times f_{k,l}^{[v_a,v_b]}.T + f_{k,l}^{[v_a,v_b]}.L\Big) \vee$$
$$\Big(f_{k,l}^{[v_a,v_b]}.\phi + \beta \times f_{k,l}^{[v_a,v_b]}.T \geq$$
$$f_{i,j}^{[v_a,v_b]}.\phi + \alpha \times f_{i,j}^{[v_a,v_b]}.T + f_{i,j}^{[v_a,v_b]}.L\Big),$$

where $HP_{i,j}^{k,l} \overset{def}{=} lcm(f_{i,j}^{[v_a,v_b]}.T, f_{k,l}^{[v_a,v_b]}.T)$ is the hyperperiod of the two frames being compared. Please note that the contention-free constraints from [28] compare two frames over the cluster cycle (the hyperperiod of all frames in the system) whereas our approach only considers the hyperperiod of the two compared frames.

Although the macrotick is typically set as the granularity of the physical medium, we can use this parameter to reduce the search space simulating what in [28] is called a scheduling "raster". Hence, increasing the macrotick length –for a network or CPU link– reduces the search space for that link, making the algorithm faster, but also reduces the solution space. However, since the typical macrotick lengths of network and CPU links are several orders of magnitude apart, it may only make sense to take advantage of the scheduling raster in network links.

3.3 Virtual link constraints

We introduce virtual link constraints which describe the sequential nature of a communication from a producer task to a consumer task. The generic condition that applies for network as well as for CPU links is that frames on sequential links in the communication path have to be scheduled sequentially on the time-line. Virtual frames of producer or consumer tasks are special cases of the above condition. All virtual frames of a producer task must be scheduled *before* the scheduled window on the first link in the communication path. Conversely, all virtual frames of the consumer task must be scheduled *after* the scheduled window on the last network link in the communication path.

End-to-end communication with low latency and bounded jitter is only possible if all network nodes (which have independent clock sources) are synchronized with each-other in the time domain. TTEthernet provides a fault-tolerant clock synchronization method [31] encompassing the whole network which ensures clock synchronization. On a real network, the precision achieved by the synchronization protocol is subject to jitter in the microsecond domain. Hence, we also consider, similar to [35], the synchronization jitter which is a global constant and describes the maximum difference between the local clocks of any two nodes in the network. We denote the synchronization jitter (also called network precision) with δ, where typically $\delta \approx 1\mu sec$ [15, p. 186].

$$\begin{aligned}&\forall vl_i \in \mathcal{VL}, \forall [v_a, v_x], [v_x, v_b] \in vl_i :\\ &[v_x, v_b].mt \times f_{i,1}^{[v_x,v_b]}.\phi - [v_a, v_x].d - \delta \geq\\ &[v_a, v_x].mt \times (last(\mathcal{F}_i^{[v_a,v_x]}).\phi + last(\mathcal{F}_i^{[v_a,v_x]}).L).\end{aligned}$$

The constraint expresses that, for a frame, the difference between the start of the transmission window on one link and the end of the transmission window on the precedent link has to be greater than the hop delay for that link plus the precision for the entire network.

3.4 End-to-End Latency constraints

Let $src(vl_i)$ and $dest(vl_i)$ denote the CPU links on which the producer task and, respectively, the consumer task of virtual link vl_i are scheduled on. We introduce latency constraints that describe the maximum latency of a communication from a producer task to a consumer task, namely

$$\begin{aligned}&\forall vl_i \in \mathcal{VL} :\\ &dest(vl_i).mt \times (last(\mathcal{F}_i^{dest(vl_i)}).\phi + last(\mathcal{F}_i^{dest(vl_i)}).L) \leq\\ &src(vl_i).mt \times f_{i,1}^{src(vl_i)}.\phi + vl_i.max\_latency.\end{aligned}$$

In essence, the condition states that the difference between the end of the last chunk of the consumer task and the start of the first chunk of the producer task has to be smaller than or equal to the maximum end-to-end latency allowed. In this paper we consider the maximum end-to-end latency to be smaller than or equal to the message period (which is the same as the period of the associated tasks).

3.5 Task constraints

For any sequence of virtual frames scheduled on a CPU link, the first virtual frame has to start after the offset of the task and the last virtual frame has to be scheduled before the deadline of the task. Hence, we have

$$\begin{aligned}&\forall v_a \in \mathcal{V}, \forall \tau_i^{v_a} \in \Gamma^{v_a} :\\ &\left(f_{i,1}^{[v_a,v_a]}.\phi \geq \tau_i^{v_a}.\phi\right) \wedge \left(last(\mathcal{F}_i^{[v_a,v_a]}).\phi \leq \tau_i^{v_a}.D - \tau_i^{v_a}.C\right).\end{aligned}$$

3.6 Virtual frame sequence constraints

For a CPU link, we check in the condition in Section 3.2 that the scheduling windows of virtual frames generated by different tasks do not overlap. Additionally, we have to check that virtual frames generated by the same task do not overlap in the time domain. This condition can be expressed similar to the condition in Section 3.2, however, we express it, without losing generality, in terms of the ordering of the virtual frame set.

$$\begin{aligned}&\forall v_a \in \mathcal{V}, \forall \tau_i^{v_a} \in \Gamma^{v_a}, \forall j \in \left[1, \left(\left|\mathcal{F}_i^{[v_a,v_a]}\right| - 1\right)\right] :\\ &f_{i,j+1}^{[v_a,v_a]}.\phi \geq f_{i,j}^{[v_a,v_a]}.\phi + f_{i,j}^{[v_a,v_a]}.L.\end{aligned}$$

3.7 Task precedence constraints

Task dependencies are usually expressed as precedence constraints [5], e.g., if task $\tau_i^{v_a}$ and $\tau_j^{v_b}$ have precedence constraints ($\tau_i^{v_a} \prec \tau_j^{v_b}$) then $\tau_i^{v_a}$ has to finish executing before $\tau_j^{v_b}$ starts. Even though these dependencies arise typically between tasks co-existing on the same CPU, we generalize dependencies between tasks executing on any end-system. Task dependencies are partially expressed in [28] as frame dependencies in the sense that one frame is scheduled before another frame, which can be used to specify aspects of the existing task schedule. We introduce constraints for simple task precedences in our model as follows

$$\begin{aligned}&\tau_i^{v_a} \prec \tau_j^{v_b} \Rightarrow [v_b, v_b].mt \times f_{j,1}^{[v_b,v_b]}.\phi \geq\\ &[v_a, v_a].mt \times (last(\mathcal{F}_i^{[v_a,v_a]}).\phi + last(\mathcal{F}_i^{[v_a,v_a]}).L).\end{aligned}$$

Note that both tasks must have the same period or “rate”. Multi-rate precedence constraints (*extended precedences* as they are called in [10]) are subject for future work.

3.8 Memory constraints

The time-triggered paradigm enables us to define the memory constraints for the switches and end-system nodes as an upper bound on the time that a frame can reside inside the transmitting queue of a node, i.e., the time a frame can logically remain on a CPU or network link.

$$\begin{aligned}&\forall vl_i \in \mathcal{VL}, \forall [v_a, v_x], [v_x, v_b] \in vl_i :\\ &[v_x, v_b].mt \times f_{i,1}^{[v_x,v_b]}.\phi - [v_a, v_x].mt \times f_{i,1}^{[v_a,v_x]}.\phi \leq [v_a, v_x].b.\end{aligned}$$

Note that an alternative method to directly constrain the memory utilization in terms of frame and buffer lengths implies a non-trivial extension that may require quantifiers in the logical formulas. The added level of expressiveness, however, does not justify the inherent increase in complexity and the required added run-time to solve the problem.

4. SMT-BASED CO-SYNTHESIS

Satisfiability Modulo Theories (SMT) checks the satisfiability of logic formulas in first-order formulation with regard to certain background theories like linear integer arithmetic ($\mathcal{LA}(\mathbb{Z})$) or bit-vectors ($\mathcal{BV}$) [2, 27]. A first-order formula uses variables as well as quantifiers, functional and predicate symbols, and logical operators [21]. Scheduling problems are

Algorithm 1: One-shot SMT schedule synthesis

```
Data: $G(\mathcal{V},\mathcal{L}), \mathcal{VL}, \mathcal{M}, \Gamma$
Result: $\mathcal{S}$ (tt-schedule)
begin
    $\mathcal{S} \leftarrow \emptyset$;
    if Check($\mathcal{V},\Gamma$) $\wedge$ Check($\mathcal{VL},\mathcal{M}$) then
        $\mathcal{C} \leftarrow$ Assert($G(\mathcal{V},\mathcal{L}), \mathcal{VL}, \mathcal{M}, \Gamma$);
        $\mathcal{S} \leftarrow$ SMTSolve ($\mathcal{C}$);
    return $\mathcal{S}$;
```

easily expressed in terms of constraint-satisfaction in linear arithmetic and are thus suitable application domains for SMT solvers; a good use-case presentation of using SMT for job-shop-scheduling can be found in [8]. At its core, our scheduling algorithm generates assertions (boolean formulas) for the logical context of an SMT solver based on the constraints defined in Section 3 where the offsets of frames are the variables of the formula. For a satisfiable context, the SMT solver returns a so-called *model* which is a solution (i.e. a set of variable values for which all assertions hold) to the defined problem.

4.1 One-shot scheduling

The *one-shot* method (Algorithm 1) considers the whole problem set including all tt-tasks on all end-systems as well as all tt-messages. The inputs of the algorithm are the network topology $G(\mathcal{V},\mathcal{L})$, the set of virtual links $\mathcal{VL}$, the set of tt-messages $\mathcal{M}$, and the set of tt-tasks Γ. The output is the set $\mathcal{S}$ of frame offsets or the empty set if no solution exists.

First, the utilization on each end-system is verified (through the `Check` function) to be lower than 100% using the simple polynomial utilization-based test (cf. [19])

$$\forall v_a \in \mathcal{V} : \sum_{\tau_i^{v_a} \in \Gamma^{v_a}} \frac{\tau_i^{v_a}.C}{\tau_i^{v_a}.T} \leq 1.$$

This test is necessary but not sufficient, i.e., if the test fails, the system is definitely not schedulable since the demand of the task set exceeds the CPU bandwidth on at least one end-system, however, if the test passes, the system may or may not be schedulable. A similar check is employed for all network links and the corresponding frames since, in general, the density of feasible systems is less than or equal to 1 [18].

If the check is successful, the algorithm adds the constraints defined in Section 3 to the solver context $\mathcal{C}$ (`Assert`) and invokes the SMT solver (`SMTSolve`) with the constructed context as described above. The solution $\mathcal{S}$ (the solver model), if it exists, contains the values for the offset variables of all frames and is used to build the tt-schedule.

The producer, consumer, and free tt-tasks as well as the tt-messages may generate, depending on the system configuration, a very large number of frames that need to be scheduled. It is known that such scheduling problems (which reduce to the bin-packing problem) are NP-complete [28]. Hence, the scalability of the one-shot approach may not be suitable for applications with hundreds of tt-messages and large network topologies.

In order to improve the performance for network-only schedule synthesis, Steiner [28] proposes an incremental backtracking approach which takes only a subset of the frames at a time and adds them to the SMT context. If a partial solution is found, additional frames and constraints are added until either the complete tt-network-schedule is found or a partial problem is unfeasible. In the case of unfeasibility, the problem is backtracked and the size of the increment is increased. In the worst case the algorithm backtracks to the root, scheduling the complete set of frames in one step.

The performance improvement due to the incremental backtracking method may be sufficient when only scheduling network messages. However, when co-scheduling messages and tasks in large systems, the number of virtual frames due to tasks running on end-systems renders the problem impracticable. Moreover, our experiments with an incremental version of our one-shot algorithm have shown that it performs best when the utilization is low (which is often true for network links) since there is enough space on the links to incrementally add new frames without having to move the already scheduled ones. However, on CPU links, the utilization due to tasks is usually high, resulting in the incremental backtracking method performing worse than in the average case. Hence, the incremental backtracking approach proposed by Steiner is not suitable for our purpose.

We present in the next section a novel incremental algorithm specifically tailored for task scheduling that reduces the runtime of combined task/network scheduling for the average case by taking into account the different types of tasks executing on end-systems.

4.2 Demand-based scheduling

Free tasks account for a significant amount of the total frames that need to be scheduled. However, these tasks do not present any dependency towards the network nor other end-system tasks. Hence, they do not need to be considered from the network perspective, but only from the end-system perspective.

The main idea behind the *demand-based* method (Algorithm 2) is to schedule only communicating tasks via the SMT solver and check, afterward, if the resulting schedule on all end-systems is feasible when adding the corresponding free tasks. In [7] we have introduced a method to generate optimal static schedules using dynamic priority scheduling algorithms. We considered tasks as being asynchronous with deadlines less than or equal to periods (i.e., constrained-deadline task systems) and generated static schedule by simulating the EDF algorithm until the hyperperiod. We employ a similar method here for scheduling free tasks. In this way, free tasks do not add to the complexity of the SMT context but are scheduled separately, resulting in improved performance for the average case. This improvement does not come at the expense of schedulability. We guarantee this by doing an incremental approach that in a first step schedules communicating tasks and checks if, for any end-system, the resulting schedule after adding the free tasks would be schedulable. If this is the case, the free tasks are scheduled by simulating EDF until the hyperperiod. If the resulting system is not schedulable the algorithm increases the SMT formulation by adding only those free tasks that make the solution unfeasible and runs the solver over the increased set. This is done incrementally until either a solution is found or the whole set of free tasks has been added to the SMT problem without finding a solution.

The inputs of the algorithm are, as before, the network

Algorithm 2: Demand-based SMT schedule synthesis

```
Data: $G(\mathcal{V},\mathcal{L}), \mathcal{VL}, \mathcal{M}, \Gamma$
Result: $\mathcal{S}$ (tt-schedule)
begin
    $\mathcal{S} \leftarrow \emptyset$;
    if Check($\mathcal{V},\Gamma$) $\wedge$ Check($\mathcal{VL},\mathcal{M}$) then
        $f \leftarrow false$;
        $\Gamma_{edf} \leftarrow \Gamma_{free}$;
        $\Gamma_{smt} \leftarrow \Gamma \setminus \Gamma_{free}$;
        while $f \neq true$ do
            $\mathcal{C} \leftarrow$ Assert($G(\mathcal{V},\mathcal{L}), \mathcal{VL}, \mathcal{M}, \Gamma_{smt}$);
            $\mathcal{S} \leftarrow$ SMTSolve ($\mathcal{C}$);
            if $\mathcal{S} \neq \emptyset$ then
                $\Gamma_d \leftarrow$ DemandCheck($\mathcal{V}, \mathcal{S}, \Gamma_{edf}$);
                if $\Gamma_d \neq \emptyset$ then
                    $\Gamma_{edf} \leftarrow \Gamma_{edf} \setminus \Gamma_d$;
                    $\Gamma_{smt} \leftarrow \Gamma_{smt} \cup \Gamma_d$;
                else
                    $f \leftarrow true$;
                    if $\Gamma_{edf} \neq \emptyset$ then
                        $\mathcal{S} \leftarrow \mathcal{S} \cup$ EDFSim($\mathcal{V}, \mathcal{S}, \Gamma_{edf}$);
            else
                $f \leftarrow true$;
    return $\mathcal{S}$;
```

topology $G(\mathcal{V},\mathcal{L})$, the set of virtual links $\mathcal{VL}$, the set of messages $\mathcal{M}$, and the set of tasks Γ (cf. Algorithm 2). Like in the one-shot method, the utilization on all end-systems and all network links is verified first (`Check` function).

We define the following helper sets. The set of free tasks Γ_{free} is the set containing all tasks that are neither producer nor consumer tasks and which are not dependent on other tasks. We also introduce the set of tasks scheduled with SMT (Γ_{smt}) and the set of tasks scheduled with EDF (Γ_{edf}).

Initially, Γ_{edf} is equal to the set of free tasks Γ_{free} and $\Gamma_{smt} = \Gamma \setminus \Gamma_{free}$ is the set of remaining tasks from Γ. We repeat the following steps until either a solution is found or the set Γ_{edf} is empty. First, we add the constraints defined in Section 3 based on the tasks in Γ_{smt} to the solver context $\mathcal{C}$ (`Assert`) and then invoke the SMT solver (`SMTSolve`) with the constructed context. If no solution exists we exit from the loop and return the empty set. If there exists a partial solution $\mathcal{S} \neq \emptyset$, we check (via the function `DemandCheck`) the demand of the resulting system together with the tasks which have not yet been scheduled (the tasks in Γ_{edf}).

The demand check is based on the necessary and sufficient feasibility condition for constrained-deadline asynchronous tasks with periodic execution under EDF (cf. [3]). The test constructs a set of intervals between any release and any deadline over a certain time-window. In each of these intervals the demand of the executing tasks is checked to be smaller than or equal to the supply (the length of the interval). In our case, for every end-system, the set of tasks is derived from the already scheduled tasks in Γ_{smt} and the tasks in Γ_{edf}. The already scheduled tasks in Γ_{smt} have fixed scheduled intervals according to their virtual frames whereas the tasks in Γ_{edf} will be treated as EDF tasks.

For every end-system $v_a \in \mathcal{V}$ the function `DemandCheck` generates a set $\widetilde{\Gamma}^{v_a}$ of virtual periodic tasks, where every virtual task $\widetilde{\tau_k}^{v_a}$ is defined by the tuple $\langle \widetilde{\tau_k}^{v_a}.\phi, \widetilde{\tau_k}^{v_a}.C, \widetilde{\tau_k}^{v_a}.D, \widetilde{\tau_k}^{v_a}.T \rangle$, consisting, as before, of the offset, the WCET, the relative deadline, and the period of the virtual task, respectively. For every task $\tau_i^{v_a} \in \Gamma_{edf}$ we generate a virtual task $\widetilde{\tau_k}^{v_a}$ with a one to one translation of the task parameters. Additionally, for every frame offset[2] $f_{i,j}^{[v_a,v_a]}.\phi \in \mathcal{S}$ we generate a virtual task $\widetilde{\tau_k}^{v_a}$ with $\widetilde{\tau_k}^{v_a}.\phi = f_{i,j}^{[v_a,v_a]}.\phi, \widetilde{\tau_k}^{v_a}.C = 1, \widetilde{\tau_k}^{v_a}.D = 1$, and $\widetilde{\tau_k}^{v_a}.T = f_{i,j}^{[v_a,v_a]}.T$.

We use the necessary and sufficient feasibility condition from [3, 23] for every generated virtual task set $\widetilde{\Gamma}^{v_a}$, namely

$$\forall v_a \in \mathcal{V}, \forall t_1 \in \Phi^{v_a}, \forall t_2 \in \Delta^{v_a}, t_1 < t_2 :$$

$$\sum_{\widetilde{\tau}_i^{v_a} \in \widetilde{\Gamma}^{v_a}} \widetilde{\tau}_i^{v_a}.C \times \left(\left\lfloor \frac{t_2 - \widetilde{\tau}_i^{v_a}.\phi - \widetilde{\tau}_i^{v_a}.D}{\widetilde{\tau}_i^{v_a}.T} \right\rfloor - \left\lceil \frac{t_1 - \widetilde{\tau}_i^{v_a}.\phi}{\widetilde{\tau}_i^{v_a}.T} \right\rceil + 1 \right)_0 \leq t_2 - t_1,$$

where

$$\Phi^{v_a} \stackrel{def}{=} \{a_{i,j}^{v_a} = \widetilde{\tau}_i^{v_a}.\phi + j \times \widetilde{\tau}_i^{v_a}.T | \widetilde{\tau}_i^{v_a} \in \widetilde{\Gamma}^{v_a}, j \geq 0, a_{i,j}^{v_a} \leq \lambda^{v_a}\},$$

$$\Delta^{v_a} \stackrel{def}{=} \{d_{i,j}^{v_a} = a_{i,j}^{v_a} + \widetilde{\tau}_i^{v_a}.D | \widetilde{\tau}_i^{v_a} \in \widetilde{\Gamma}^{v_a}, j \geq 0, d_{i,j}^{v_a} \leq \lambda^{v_a}\},$$

$$\lambda^{v_a} = max(\{\widetilde{\tau}_i^{v_a}.\phi | \widetilde{\tau}_i^{v_a} \in \widetilde{\Gamma}^{v_a}\}) + 2 \times lcm(\{\widetilde{\tau}_i^{v_a}.T | \widetilde{\tau}_i^{v_a} \in \widetilde{\Gamma}^{v_a}\}).$$

The sets Φ^{v_a} and Δ^{v_a} of arrivals and absolute deadlines, respectively, define intervals in which the demanded execution time of running tasks has to be less than or equal to the processor capacity [3, 23]. If the test is fulfilled on every end-system, we know that applying EDF to the task sets will result in a feasible schedule. In this case, the function `DemandCheck` returns an empty set. We schedule the remainder of the tasks by running an EDF simulation (`EDFSim`) on each end-system of the entire virtual task set (composed of both scheduled and unscheduled tasks) until the hyperperiod. The EDF simulation will return the static schedule for the tasks in Γ_{edf} which will complete the partial solution $\mathcal{S}$. If the schedulability condition is not fulfilled on some end-system, the function `DemandCheck` returns the set (Γ_d) of tasks that have contributed to the intervals where the demand was greater than the supply. These tasks are removed from the set Γ_{edf} and added to the set Γ_{smt} and the procedure is repeated. The loop terminates ($f \leftarrow true$) when either a full solution is found or the SMT solver could not synthesize a partial schedule for Γ_{smt}.

Note that in the worst case, the algorithm may perform worse than the one-shot method due to the intermediary steps in which partial solutions were unfeasible. If none of the partial solutions were feasible, in the last step, the demand-based algorithm has to solve the same input set as the one-shot method.

The feasibility test[3] is known to be co-NP-hard [17, p. 615]. Therefore, the underlying scheduling problem still remains exponential in the worst case. However, the run-time of the test is highly dependent on the properties of the tasks (periods, harmonicity of periods, hyperperiod, etc.) which, in practice, are not that pessimistic. Thus, the demand method may be more practicable than solving the entire problem using SMT in the average case. Moreover, splitting the problem and solving it using an incremental approach

[2]Frames of the same task scheduled sequentially on the timeline can be joined into a bigger virtual task to increase the performance of the feasibility test.

[3]Note that other tests with pseudo-polynomial complexity [22, 3] could be used instead, but these are only sufficient or deal with restricted task sets.

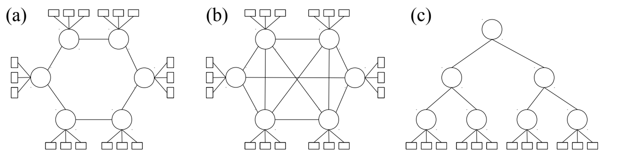


Figure 2: Example network topologies: (a) Ring–size 6, (b) Mesh–size 6, (c) Tree–depth 2. All examples with 3 end systems per switch (leaf nodes only).

Size	Topology	Num Switches	Num End-Systems
Small (S)	*Mesh, Ring*	2	4
	Tree, depth $= 1$	4	6
Medium (M)	*Mesh, Ring*	4	16
	Tree, depth $= 2$	13	36
Large (L)	*Mesh, Ring*	8	48
	Tree, depth $= 3$	15	48

Table 1: Configuration parameters for network configurations of each size.

also reduces the runtime for the average case in which only a few incremental steps are needed.

Naturally, we do not improve the scalability of the underlying SMT solver, rather, we reduce, regardless of the algorithm complexity and without sacrificing schedulability, the size of the SMT problem and hence the number of assertions and frames that place a burden on the solver. Through this we can tackle medium to large problems even in the extended scenario of co-scheduling preemptive tasks together with messages in a multi-hop switched network. Moreover, finding a schedule with the SMT solver becomes harder the more utilized the links become. By eliminating subsets of tasks from the input of the SMT solver we make it easier for the SMT solver to place the (virtual) frames of the remaining tasks, thus shifting the complexity from the SMT solver to the schedulability test.

We show in the next section that the demand method outperforms the one-shot in most cases and results in significant performance improvements leading to better scalability for medium to large input configurations.

5. EVALUATION

We have implemented a prototype tool, called TT-NTSS, for task- and network-level static schedule generation based on the system model, constraint formulation, and scheduling algorithms described above. The underlying SMT solver employed by the tool is Yices v2.2.1 (64bit) [6] using linear integer arithmetic ($\mathcal{LA}(\mathbb{Z})$) without quantifiers as the background theory. We have run all experiments on a 64bit 8-core 3.40GHz Intel *Core-i7* PC with 16GB memory. We have fixed a $1\mu sec$ granularity for the network links, and defined two different network speeds (100Mbit/s and 1Gbit/s).

We analyze the performance of TT-NTSS over a number of industrial-sized synthetic scenarios following the network topologies depicted in Figure 2. For each case we evaluate three network sizes which range from small (i.e. a couple of switches) to large (i.e. tens of switches). We scale proportionally the number of connected end systems and therefore the number of tasks to be scheduled. We define a virtual link between each two communicating tasks executing on distinct randomly-selected end systems. Table 1 summarizes the set of configurations. Message sizes are chosen randomly between the maximum and minimum Ethernet packet sizes, while periods are randomly distributed among three different predefined sets. The WCET of tasks is set proportionally to the task period and the desired CPU utilization bound, rounded to the nearest macrotick multiple. Each end-system runs a total of 16 tasks, of which 8 are communicating and 8 free. VLs are defined between communicating tasks running on randomly selected end-systems. It is a common pattern in industrial applications that communicating tasks (e.g. sensing and actuating) are sensibly smaller than non-communicating ones (e.g. background computation and core functionality). Therefore, we choose to model free tasks to account for approximately 75% of the utilization and communicating tasks for 25%. For the experiments we use 3 different period configurations, namely $\{10, 20, 25, 50, 100\}, \{10, 30, 100\}$, and $\{50, 75\}$ ms. The time-out for each experiment was set to 100 minutes after which the problems were deemed unfeasible. Note that the number of leaves in the tree topology is set to 3 for the small and medium sized networks and 2 for the large, while the tree depth is set to 1, 2, and 3 for small, medium, and large, respectively.

Figures 3 and 4 depict the runtime of the demand-based algorithm compared to the one-shot with different network topologies and period configurations. For these experiments we fixed the macrotick on each end-system to $250\mu s$ and the average utilization of tasks to 50%. The y-axis showing the runtime has a logarithmic scale and the x-axis shows the 3 different sizes for each topology, each size being described by the tuple (switches, total end-systems, total tasks, virtual links). We combine the mesh and ring topologies together in Figure 3 since they have similar sizes in terms of end-systems, switches, tasks, and virtual links. The one-shot method reaches the time-out (100 minutes) even for most medium-sized problems whereas the demand method performs significantly better in all cases scaling up to large network sizes.

The hardware-dependent macrotick for time-driven scheduling in real-time operating systems (RTOS) running on embedded platforms is usually in the range of hundreds of microseconds to a few milliseconds [4, p. 266]. The RTOS developed internally at TTTech (see [7] for a short description) running on a TMS570 MCU [33] with a 180 MHz ARM Cortex-R4F processor has a configurable macrotick in the range of $50\mu s$ to $1ms$. Smaller macroticks increase the re-

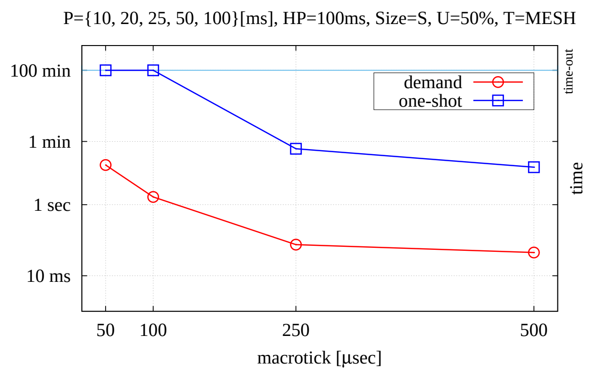
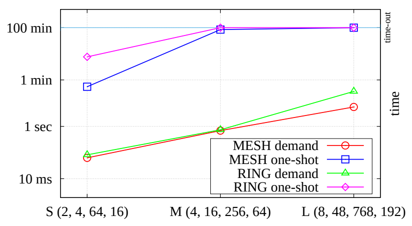
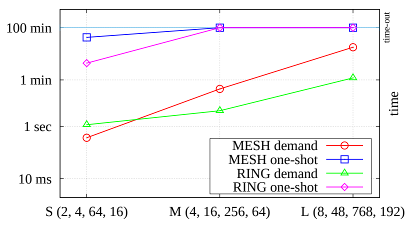


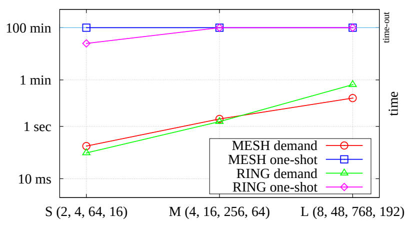
Figure 5: Runtime as a function of the macrotick.



(a) $P_1 = \{10, 20, 25, 50, 100\}ms$

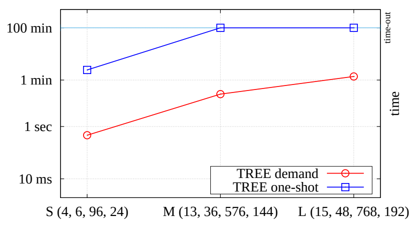


(b) $P_2 = \{10, 30, 100\}ms$

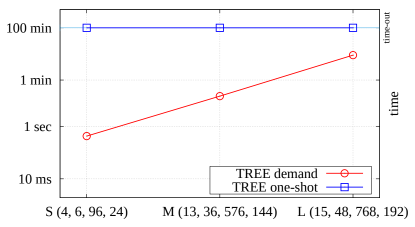


(c) $P_3 = \{50, 75\}ms$

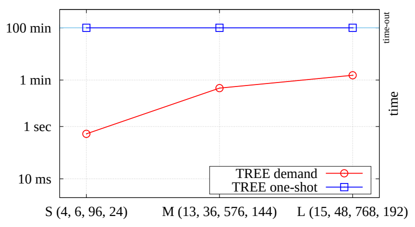
Figure 3: Runtime for mesh and ring topologies with $MT = 250\mu sec$, $U = 50\%$.



(a) $P_1 = \{10, 20, 25, 50, 100\}ms$



(b) $P_2 = \{10, 30, 100\}ms$



(c) $P_3 = \{50, 75\}ms$

Figure 4: Runtime for the tree topology with $MT = 250\mu sec$, $U = 50\%$.

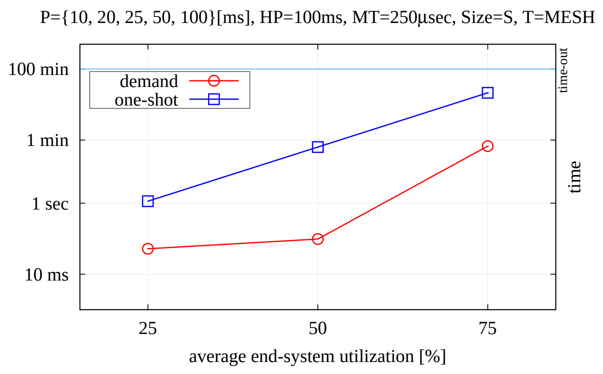


Figure 6: Runtime as a function of the average end-system utilization.

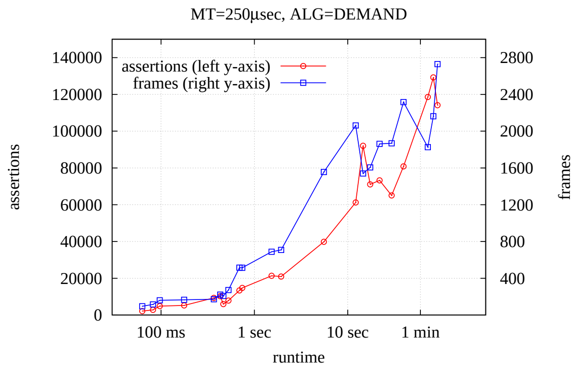


Figure 7: Assertions and frames as a function of the runtime.

sponsiveness of the system but introduce more overhead due to more frequent timer interrupt invocations and context switches. The macrotick also has an impact on the runtime of our method, a bigger macrotick leads to tasks generating less virtual frames but decreases the solution space (similar to the raster method for network links). In Figure 5 we compare the runtime of the one-shot and demand algorithms (logarithmic y-axis) as a function of the macrotick length (x-axis). All values were obtained using the small mesh topology with 50% task utilization, period configuration $\{10, 20, 25, 50, 100\}$, and macrotick values between $50\mu s$ and $0.5ms$. As can be seen, the smaller the macrotick is, the longer it takes to find a schedule due to the increasing number of virtual frames generated by the tasks on the end-system CPUs.

In Figure 6 we compare the runtime of the demand and one-shot methods (logarithmic y-axis) as a function of the average end-system utilization (x-axis) for a small mesh topology where each end-system has a macrotick of $250\mu s$ and the task and message periods are chosen randomly from the period configuration set $\{10, 20, 25, 50, 100\}ms$. The more utilized the end-systems becomes the harder it is for the SMT solver to find a solution. We remind the reader that free tasks account for approximately 75% of the utilization and communicating tasks for around 25%. The demand algorithm eliminates, in the best case, up to 75% of the tasks and therefore, even for a highly utilized end-system, the size of the SMT problem becomes significantly smaller.

The runtime of the scheduling method is dependent on a number of factors, the most important of them being the number of frames that need to be scheduled. However, as can be seen from the previous figures, there is a non-monotonic relationship between the various variables and the runtime of the algorithm. The number of frames has a complex dependency on the macrotick, the hyperperiod, the relation and length of the periods, the topology, etc. It is therefore hard to find a monotonic relationship between these variables and the complexity of the problem. However, there is a monotonic relationship between the number of assertions and the runtime, and, to a lesser degree, between the

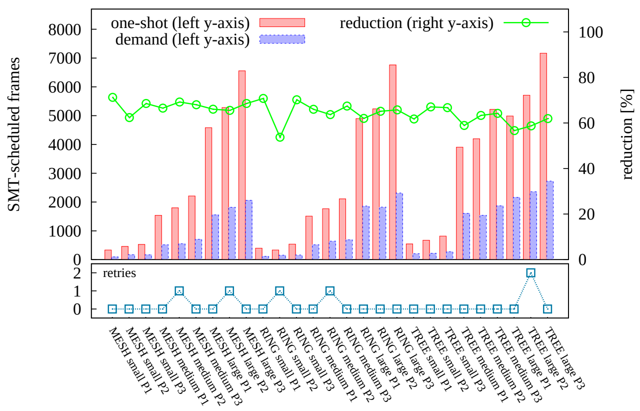


Figure 8: SMT-scheduled frames with the one-shot and demand methods.

number of frames and the runtime. In Figure 7 we plot the number of logical assertions (left y-axis) and frames (right y-axis) as functions of the runtime (logarithmic x-axis). The values were obtained from the previous experiments with all 3 topologies scheduled with the demand-based algorithm with all 3 period configurations and a macrotick of $250\mu s$. We omitted from the figure the one-shot method since most of the experiments reached the time-out, as well as the experiments where the demand-based algorithm needed more than one incremental step due to failed demand checks.

On average, the demand-based algorithm performed significantly better than the one-shot method due to the fact that it eliminates the majority of the virtual frames generated by free tasks from the SMT formulation. In Figure 8 we show the total number of frames (virtual frames from tt-tasks and frame instances of network tt-messages) scheduled with each method as well as how many incremental steps ("retries" sub-plot) were used for the demand method in all the network configurations discussed above. The significant performance improvements result directly from the reduced number of frames (on average a 64.8% reduction) that have to be scheduled with the SMT solver in each case.

Finding a solution for very large input sets still remains an impracticable problem (unless $P = NP$) since SMT solvers, which generalize SAT solvers, have exponential complexity. For inputs that generate an amount of frames and assertions beyond the ranges presented above, the problem becomes intractable, making the proposed method unfeasible. Even though an active community constantly improves the performance of SMT solvers, for very large systems a heuristic method or a combination of heuristics and SMT-based scheduling, at the expense of decreased solution spaces, remains the most promising approach.

6. RELATED WORK

The starting point for our work was [28] in which the author formulates SMT message scheduling constraints for multi-hop time-triggered networks and solves them using Yices [6]. We extend the work done by Steiner threefold. First, we extend the problem definition to include preemptive tasks that run in a table-driven schedule on end-system nodes and formulate the scheduling constraints based on this model. Furthermore, we add support for different link speeds and time-line granularities for both network and CPU links. Finally, based on this extended model, we show how to efficiently create combined task and network schedules with deterministic end-to-end latency that push the time-triggered properties of TTEthernet to the software layers.

Other approaches besides [28] also discuss the generation of message schedules for time-triggered networks without factoring in the producing and consuming tasks. The problem of generating a time-triggered message schedule is extended with rate-constrained traffic considerations by either scheduling reserved slots that correspond to the rate-constrained requirements [29] or by formulating an optimization problem that minimizes the end-to-end delay of rate-constrained frames [32]. The work in [11] addresses the synthesis of time-triggered message schedules for Profinet IO where messages depend on pre-scheduled producer and consumer tasks. Scheduling for time-triggered network-on-chip, where both scheduling points and communication routes of messages are assigned, is studied in [13].

There have been several approaches dealing with task and message scheduling in tandem for time-triggered communication. A recent paper [35] studies task and messages schedule co-synthesis in switched time-triggered networks using a MIP multi-objective optimization formulation. Similar to our work, the authors differentiate between communicating and free tasks, however, their task model is non-preemptive whereas ours allows preemption which increases the solution space on the application level. Another MIP-based approach can be found in [34] where FlexRay bus scheduling is considered. Scheduling preemptive tasks together with time-triggered messages has been analyzed in [25, 24] for fixed-priority scheduled tasks communicating through a TTP bus. Similarly, [20] studies a SAT-based solution for task and message scheduling on bus systems where tasks are scheduled using a fixed-priority assignment.

In [26], a system consisting of communicating event- and time-triggered tasks running on distributed nodes is scheduled in conjunction with the associated bus messages from the dynamic and static domains respectively. A similarity to our work consists in the separate schedulability test (in this case fixed-priority) for event-triggered tasks based on the static schedule of the time-triggered tasks.

Hitherto, all presented methods for task and message schedule co-synthesis deal either with non-preemptive tasks on multi-hop networks, or with preemptive tasks on simple bus network topologies. We consider a more complex problem by including preemptive tasks that communicate in a switched multi-hop multi-speed time-triggered network.

7. CONCLUSION

We have introduced algorithms for the simultaneous co-synthesis of time-triggered schedules for both network messages and preemptive tasks in switched multi-speed time-triggered networks. We have defined the schedulability constraints as Satisfiability Modulo Theories (SMT) formulæ and solved them using the state-of-the-art solver Yices. Moreover, we have shown how to increase, for the average case, the performance of our method through a novel incremental scheduling approach based on the utilization demand bound analysis for asynchronous periodic tasks from classical scheduling theory. Our evaluation, using a variety of synthetic network topologies and system configurations, shows that our approach can tackle medium to large problems efficiently and scales for industrial-sized systems.

8. REFERENCES

[1] ARINC Report 664P7-1. *Aircraft Data Network, Part 7: Avionics Full Duplex Switched Ethernet (AFDX) Network*, Sept. 2009.

[2] Barrett, C., Sebastiani, R., Seshia, S., and Tinelli, C. Satisfiability modulo theories. In *Handbook of Satisfiability*, vol. 185. IOS Press, 2009.

[3] Baruah, S. K., Rosier, L. E., and Howell, R. R. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Syst. 2*, 4 (1990).

[4] Buttazzo, G. C. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Springer-Verlag, 2004.

[5] Chetto, H., Silly, M., and Bouchentouf, T. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Syst. 2*, 3 (1990).

[6] Computer Science Laboratory – SRI International. The Yices SMT Solver. `http://yices.csl.sri.com/`. retrieved 27-Apr-2014.

[7] Craciunas, S. S., Oliver, R. S., and Ecker, V. Optimal static scheduling of real-time tasks on distributed time-triggered networked systems. In *Proc. ETFA* (2014), IEEE Computer Society.

[8] De Moura, L., and Bjørner, N. Satisfiability modulo theories: Introduction and applications. *Commun. ACM 54*, 9 (2011), 69–77.

[9] Derler, P., and Resmerita, S. Flexible static scheduling of software with logical execution time constraints. In *Proc. CIT* (2010), IEEE.

[10] Forget, J., Grolleau, E., Pagetti, C., and Richard, P. Dynamic priority scheduling of periodic tasks with extended precedences. In *Proc. ETFA* (2011), IEEE Computer Society.

[11] Hanzalek, Z., Burget, P., and Šucha, P. Profinet IO IRT message scheduling. In *Proc. ECRTS* (2009), IEEE Computer Society.

[12] Honeywell Aerospace. Application specific integrated circuits based on TTEthernet ready for first Orion test flight. `http://aerospace.honeywell.com/about/media-resources/newsroom`, 2014. retrieved 22-May-2014.

[13] Huang, J., Blech, J. O., Raabe, A., Buckl, C., and Knoll, A. Static scheduling of a time-triggered network-on-chip based on SMT solving. In *Proc. DATE* (2012), IEEE Computer Society.

[14] Issuing Committee: As-2d2 Deterministic Ethernet And Unified Networking. SAE AS6802 time-triggered ethernet. `http://standards.sae.org/as6802/`, 2011. retrieved 20-May-2014.

[15] Kopetz, H. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[16] Kopetz, H., Ademaj, A., Grillinger, P., and Steinhammer, K. The time-triggered ethernet (TTE) design. In *Proc. ISORC* (2005), IEEE.

[17] Leung, J., Kelly, L., and Anderson, J. H. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Inc., 2004.

[18] Leung, J. Y.-T., and Merrill, M. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters 11*, 3 (1980), 115–118.

[19] Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM 20* (1973), 46–61.

[20] Metzner, A., Franzle, M., Herde, C., and Stierand, I. Scheduling distributed real-time systems by satisfiability checking. In *Proc. RTCSA* (2005), IEEE Computer Society.

[21] Moura, L., and Bjørner, N. Satisfiability modulo theories: An appetizer. In *Formal Methods: Foundations and Applications*, vol. 5902. Springer Berlin Heidelberg, 2009, pp. 23–36.

[22] Pellizzoni, R., and Lipari, G. A new sufficient feasibility test for asynchronous real-time periodic task sets. In *Proc. ECRTS* (2004), IEEE Computer Society.

[23] Pellizzoni, R., and Lipari, G. Feasibility analysis of real-time periodic tasks with offsets. *Real-Time Syst. 30*, 1-2 (2005), 105–128.

[24] Pop, P., Eles, P., and Peng, Z. An improved scheduling technique for time-triggered embedded systems. In *Proc. EUROMICRO* (1999), IEEE Computer Society.

[25] Pop, P., Eles, P., and Peng, Z. Schedulability-driven communication synthesis for time triggered embedded systems. *Real-Time Syst. 26*, 3 (2004), 297–325.

[26] Pop, T., Eles, P., and Peng, Z. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Proc. CODES* (2002), ACM.

[27] Sebastiani, R. Lazy satisfiability modulo theories. *JSAT 3*, 3-4 (2007), 141–224.

[28] Steiner, W. An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks. In *Proc. RTSS* (2010), IEEE Computer Society.

[29] Steiner, W. Synthesis of static communication schedules for mixed-criticality systems. In *Proc. ISORCW* (2011), IEEE Computer Society.

[30] Steiner, W., Bauer, G., Hall, B., and Paulitsch, M. TTEthernet: Time-Triggered Ethernet. In *Time-Triggered Communication*, R. Obermaisser, Ed. CRC Press, Aug 2011.

[31] Steiner, W., and Dutertre, B. Automated formal verification of the TTEthernet synchronization quality. In *NASA Formal Methods*, vol. 6617 of *Lecture Notes in Computer Science*. Springer, 2011.

[32] Tamas-Selicean, D., Pop, P., and Steiner, W. Synthesis of communication schedules for ttethernet-based mixed-criticality systems. In *Proc. CODES+ISSS* (2012), ACM.

[33] Texas Instruments. TMS570LS Series 16/32-BIT RISC Flash Microcontroller. `http://www.ti.com/lit/ds/symlink/tms570ls3137.pdf`. retrieved 12-Jun-2014.

[34] Zeng, H., Zheng, W., Di Natale, M., Ghosal, A., Giusto, P., and Sangiovanni-Vincentelli, A. Scheduling the flexray bus using optimization techniques. In *Proc. DAC* (2009), ACM.

[35] Zhang, L., Goswami, D., Schneider, R., and Chakraborty, S. Task- and network-level schedule co-synthesis of Ethernet-based time-triggered systems. In *Proc. ASP-DAC* (2014), IEEE Computer Society.