SMT-based Synthesis of TTEthernet Schedules: a Performance Study

Francisco Pozo*, Guillermo Rodriguez-Navas*, Hans Hansson*, and Wilfried Steiner†
*School of Innovation, Design and Engineering, Mälardalen University
Västeras, Sweden
Email: {francisco.pozo, guillermo.rodriguez-navas, hans.hansson}@mdh.se
†TTTech Computertechnik AG
Vienna, Austria
Email: wilfried.steiner@tttech.com

***Abstract*—Time-triggered networks, like TTEthernet, require adoption of a predefined schedule to guarantee low communication latency and minimal jitter. The synthesis of such schedules is a problem known to be NP-complete. In the past, specialized solvers have been used for synthesizing time-triggered schedules, but more recently general-purpose tools like Satisfiability Modulo Theories (SMT) solvers have reported synthesis of large network schedules in reasonable time for industrial purposes. An interesting characteristic of any general-purpose tool is that its configuration parameters can be tuned in order to fit specific problems and achieve increased performance. This paper presents a study identifying and assessing which SMT solver parameters have the highest impact on the performance when synthesizing schedules for time-triggered networks. The results show that with appropriate values of certain parameters, the time can be reduced significantly, up to 75% in the best cases compared to previous work.**

I. Introduction

The time-triggered paradigm [1] defines an offline schedule to ensure separation between time-triggered and non time-triggered traffic while providing high reliability in the most critical parts of the network. These offline schedules can be seen as a contract in which there is an agreement in the way the network resources are shared. For time-triggered communication, the synthesis amounts to allocating all the frames on the communication links while fulfilling a set of constraints.

The problem of synthesizing a schedule can be formulated as a constraint problem where a valid offline schedule is obtained when every constraint is satisfied. Specialized solvers have been designed to solve these specific constraints problems [2], [3], but recently a very powerful type of general-purpose constraints solver called Satisfiability Modulo Theory (SMT) has emerged. SMT solvers can solve decision problem expressed in first-order logic with respect to background theories [4]. Two examples of state of the art SMT solvers are Yices and Z3.

The size and complexity of time-triggered networks are steadily increasing, which is a challenging problem for the synthesizer tool. An evaluation of the viability of SMT-based synthesis of schedules was performed in [5], which reported synthesis of schedules of industrial size networks (1000 frames) in around half an hour. In the conclusion of the evaluation, the author proposed different options to reduce the time of the synthesis, by reducing the number of constraints. Identifying how to reduce the synthesis time but maintaining the same number of constraints is a relevant industrial problem: only faster synthesis of schedules will allow faster exploration of different frame assignments in the network; this is required in order to be able to find feasible schedules for larger networks in reasonable time.

In this paper we study how the configuration of the general-purpose SMT solver impacts the performance for synthesizing time-triggered network schedules. The study does not only focus on the configuration parameters of the SMT solver itself, but also how to use it for our purpose. We consider TTEthernet constraints definitions and the case study networks in conjunction with the schedule synthesizer developed by Steiner [5]. In addition, we have developed a new schedule synthesizer, which uses an up-to-date SMT solver that provides more configuration parameters and better performance.

The paper starts with an explanation of the basic formalism of the time-triggered TTEthernet network, the definition of the synthesis of its schedule and the constraints that have to be accomplished (Section II). Then the SMT-based strategy to synthesize schedules is explained with the different parameters that can be configured (Section III). Results of the study show a synthesis time improvement up to 75% in the best case compared to previous works for networks containing a large number of constraints (Section IV). The paper ends with a summary of the improvements brought by the configuration of the different parameters of the SMT solver (Section V).

II. Background

A. Network definition

TTEthernet networks are defined as multi-hop networks that are composed by end systems, switches and communication links. End systems can only be connected to switches by communication links whereas switches can be connected to end systems and other switches. The physical topology of a multi-hop network is defined by an undirected graph $G(V, E)$ where V is the set of end systems and switches and E are the communication links. Figure 1 is an example of a TTEthernet network with two switches, five end systems and seven communication links.

Physical communication links are bidirectional and composed by two directed unidirectional *dataflow links* that connect two vertices, one for each direction. A sequence of dataflow links forms a *dataflow path* that connects a vertex (sender) with another vertex (receiver). In a TTEthernet network the sender and receiver can only be end systems, whereas the other vertices of the dataflow path can only be switches. As seen in Figure 1, $v_1 = v_s$ is the sender, $v_6 = v_r$ is the receiver and the other vertices in the dataflow path v_4

978-1-4673-7711-9/15/$31.00

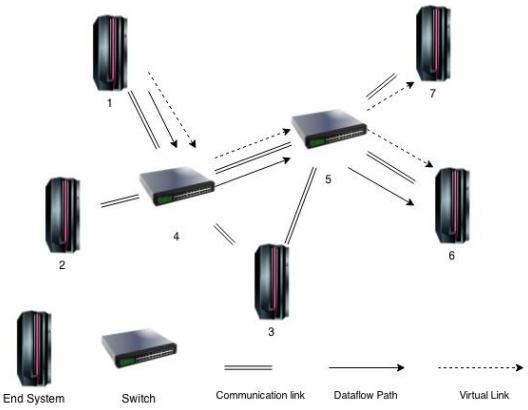


Fig. 1: TTEthernet Network with two switches, five end systems and seven communication links

and v_5 are switches, forming the dataflow path denoted by $p = ((v_1, v_4), (v_4, v_5), (v_5, v_6))$.

TTEthernet also allows multiple receivers for a unique sender with the implementation of *virtual links* similarly to ARINC 664-p7. A virtual link is a union of one or more dataflow paths. Moreover, we can see the union of dataflow paths as a directed tree structure where the sender is the root and the receivers are leafs. In Figure 1, the virtual link tree has v_1 as root and v_6, v_7 as leafs.

Information between the sender and receivers is communicated by means of frames where the route from the sender to the receivers is defined by the virtual link. An *instance* of a frame is a specific frame being transmitted in a specific dataflow link. Therefore a frame is defined by a collection of instances of the frame, formulated as $f^{(v_i,v_j)}$, where the data is transmitted from v_i to v_j.

TTEthernet has three different classes of frames:

- Time-Triggered (TT): has hard real-time requirements and is scheduled offline.
- Rate-Constrained (RC): with soft real-time requirements, but with guaranteed bandwidth; is scheduled online.
- Best-Effort (BE): follows the Standard Ethernet policy.

B. Scheduling rationale

In this paper we will focus only on TT frames as they are the only ones whose schedule is synthesized. An instance of a TT frame is defined by a triple with the period, length and offset $f^{(v_i,v_j)} = \{f.period, f.length, f^{(v_i,v_j)}.offset\}$. The period and length of the TT frame are given a priori by the designer of the network. The task of the *tt-scheduler*, the synthesizer of schedules developed in this work, is to assign a specific value to the offset for all the instances of TT frames.

The offset value of every TT frame instance has to identify the transmitting time from the first and all the iterations until the network stops. A cyclic scheduling of all the TT frames, with a *hyper-period*, is defined by the least common multiple of all the TT frames periods [6].

Another inconvenience is the high complexity of scheduling with the resolution of the clock of the network (one clock tick of the synchronization protocol. To reduce complexity, one of the most used techniques is the division of the hyper-period into equally sized slots called *raster* [7]. The size of the raster slots is the time needed for the transmission of the longest instance of the frame on any dataflow link.

C. Constraints definition

The assignment of the value to the offset of the TT frames is limited by a series of constraints of the TTEthernet network. We will mention the constraints that are implemented in the *tt-scheduler*. Due to space limitations, the formal specification of the constraints is not discussed here. Interested readers are referred to [5].

- Avoid-collision: only one frame can be transmitted at the same time through a given dataflow link.
- Ensure-causality: controls the correct sequence of transmission of all the instances of the frame in their path from the sender to the final receivers.
- Avoid-buffer-overflow: to be able to always receive the TT frames and not having to discard frames because the switch's memory is full, we define a constraint on the number of time slots that a frame can be stored in the switch.
- Simultaneous-relay: optional implementation of TTEthernet required for some implementations, which will dispatch the frame simultaneously to all outgoing dataflow links of the switch.

III. SMT-Based Synthesis of Schedules

The *tt-scheduler* developed in this work is based on the SMT solver Yices, hence all the names of the SMT solver functions are referred to Yices. The same strategy could, however, be implemented in most of the others SMT solvers.

The principal flow of the *tt-scheduler* is divided in four phases. The first phase parses the information of the input file of the network into internal memory. The input file contains the number of links of the network, the parameters of the constraints and the virtual links of all the TT frames of the network. This information will be used to generate the scheduling constraints in the principal loop. The second phase is the configuration of the Yices Solver and the synthesizer of schedules. Different parameters can be chosen both in the solver and in the synthesizer. The third phase is the principal loop and will be explained in the following paragraphs. The last phase, the output phase, gathers the information of the solver and saves it in the appropriate file format.

A background theory is a collection of symbols and a structure that interpret the symbols [8]. For example, if we have the symbols 0, 1, $+$, $-$ and $<$, interpreted in the usual way, we get a theory that can add, subtract and calculate inequalities of any number coded as boolean. There are many different background theories, some of them are more used than others, and every SMT solver chooses which ones to integrate. The most implemented background theories in the SMT solvers are the linear arithmetic (which we just described), difference arithmetic, non-linear arithmetic, free function, bit-vectors and array. Different background theories can be merged to create new background theories, but not all combinations are possible. Every SMT solver also chooses which background theories can be merged.

The principal loop tries to solve all the constraints and returns the schedule. The approach to solve schedules with SMT-based solvers has two main steps: add constraints into the logic context of the solver with the function $yices\_assert\_formula()$ and solve the logic context with the function $yices\_check\_context()$. In our case, as the number of constraints is too large to be solved in one check, we divide the whole set of constraints into groups and check them

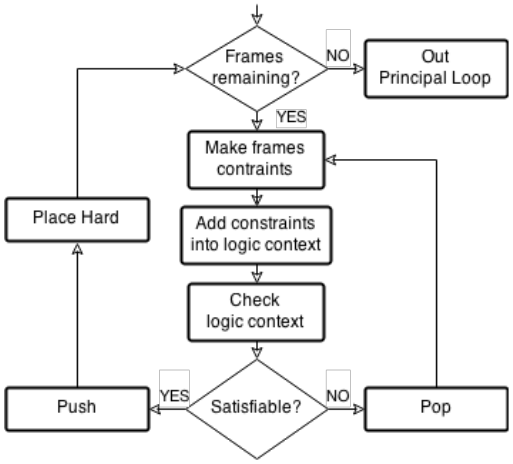


Fig. 2: Principal Loop of the *tt-scheduler*

group by group. Note that every time a new group is checked, the solution of the previous groups of constraints has to be considered. Every group will be be divided by frames, such that every group of constraints refers to a fixed number of frames. This number is configured offline with a variable called $stepsize$.

Figure 2 shows the *tt-scheduler* principal loop. In every iteration of the loop, a $stepsize$ number of frames is scheduled. The loop starts building the constraints of the frames to be scheduled, adding them into the logical context with the $yices\_assert\_formula()$ function and calling the solver with $yices\_check\_context()$ which tries to find a solution that fulfills all the constraints of the logical context. If the solver finds a solution, the check function returns a $status\_sat$ (satisfiable) flag, otherwise it returns a $status\_unsat$ (unsatisfiable) flag. Along with the $status\_sat$ flag, Yices creates a $model$ with a mapping into constant values of all the variables that fulfills the constraints of the logical context, which is a valid schedule of the network.

In the principal loop, when the *tt-scheduler* gets a safistiable flag from Yices, it saves the logic context into an internal stack of Yices with the $yices\_push()$ function for backtracking purposes. To take into account the solution for the next iteration, new constraints are built from the results of the model and are added to the logic context with the $place\_hard()$ function. These iterations will be performed until there are no more frames to schedule, and in the last iteration, the $model$ of Yices will contain the schedule of the whole network. In contrast, if the check context function returns an unsatisfiable flag, the last satisfiable configuration is loaded with the $yices\_pop()$ function and the loop is repeated again with the constraints of all the frames since the last satisfiable configuration. Note that in the case that there is no more satisfiable configuration to be loaded, it means that the current network with these frames constraints cannot be scheduled.

IV. Study

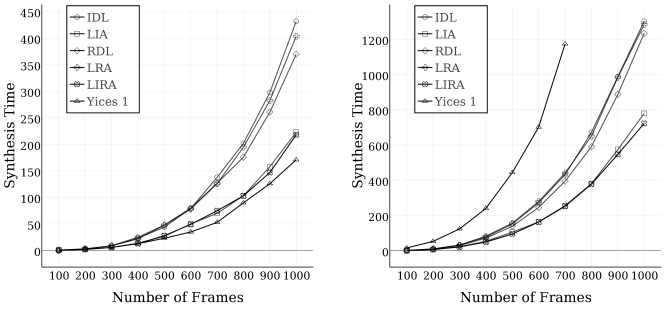
In the following studies we aim to improve the results from [5]. A first approach to reduce the time to synthesize the schedule and be able to study more configuration parameters is to update the SMT solver to the newest version. The previous evaluation has been done with the SMT solver Yices 1, a new version of the solver, Yices 2.2, is available, which exhibits better performance and offers more varied configuration options [9]. The parameter of the SMT solver with strongest effect on the performance is the Background Theories (BTs). Yices 1 only allows two configurations: the default BT, or use only the arithmetic BT, which was used in the previous mentioned evaluation. In contrast, Yices 2 offers a higher number of BTs. There are also possibilities to change parameters outside Yices to try to reduce the time. We study how the length of the steps of every iteration, $stepsize$, impacts the time of synthesis.

To perform the study we ran the synthesizer in 20 different networks from Steiner's evaluation. All the networks have a snowflake topology, 50 dataflow links and a number of frames between 100 and 1000. The snowflake network was chosen because it was reported as one of the more time consuming to synthesize and to be a common configuration for end-users applications. The networks are divided in two groups depending on the distribution: a high distribution (HD) group in which all end systems are broadcasting the frame to all possible receivers; a low distribution group (LD) in which between a third and a half of the end systems are broadcasting and the rest of the end systems belong to two different multicast groups that only exchange frames to the end systems belonging to the same group. As for the configuration parameters of the scheduler, the steps in frames of every iteration is set with $stepsize = 4$. The study was run in a MacBook Pro in OS X Yosemite with a 2,6 Ghz Intel Core i7 and 10 GB of RAM.

A. Yices 2 Background Theories

Nowadays, Yices 2 supports seven BTs: Arrays with eXtensionality (AX), BitVectors (BV), Integer Difference Logic (IDL), Real Difference Logic (RDL), Linear Integer Arithmetic (LIA), Linear Real Arithmetic (LRA) and Uninterpreted Functions (UF). It also supports deactivating all BTs (for propositional logic only) and up to three BTs combined at the same time. Note that not all the BTs are valid to synthesize schedules. Only the arithmetic BTs (IDL, RDL, LIA and LRA) are valid, the other BTs do not interpret the constraints as we need and therefore synthesize invalid schedules. The combination of BTs with at least one arithmetic theory and one or more non-arithmetic theories also generated invalid schedules. The combination of only arithmetic BTs is still valid, but unfortunately the only arithmetic combination that Yices 2 supports is LIA + LRA, constituting the Linear Integer Real Arithmetic (LIRA). To compare the performance between Yices 1 and Yices 2, we used Steiner's synthesizer, which is implemented in Yices 1, using the arithmetic's BT.

Figure 3a shows the synthesis time of the different BTs for the LD networks, the Yices 1 synthesis time is shown too for comparison purposes. Two different groups can be distinguished: a first group with IDL, LRA and RDL BTs; and a second group with LIA and LIRA BTs, which can synthesize the same network schedule in almost half of the time but slightly more time than Yices 1. LIA is the BT that yields lowest synthesis time for our scheduling problem. The combination of LIA with LRA also gives similar synthesis times; this can be due to the fact that Yices 2 does not activate the LRA BT in the LIRA combination, as it does not find any Real variable in the logic context. Similar results can be observed in Figure 3b for HD networks, where the schedule is synthesized with LIA and LIRA in 40 % less time than



(a) LD network (b) HD network

Fig. 3: Synthesis time of different BTs in Yices 2 and Yices 1 for LD and HD networks

the with other BTs. The improvement compared to Yices 1 is about three times less synthesis time needed. There is no data for the 900 and 1000 frames HD networks for Yices 1; this is because the memory that the synthesizer needed was more than 10 GB and the synthesizer stopped. This memory problem does not occur in Yices 2. In a further study about the memory consumption, the Yices 2 synthesizer only uses 4,6 GB of memory for 1000 frames HD networks and 2,2 GB for 800 frames HD networks. So as a side effect of this study, we observed that Yices 2 has much lower memory consumption than Yices 1, which enables scheduling of larger networks.

B. Stepsize

We study whether the number of constraints solved at every iteration in the logical context can also affect the performance of the synthesizer modifying the $stepsize$. The study is done on Yices 2 with the LIA BT as this is the one that performed best. The number of frames of the network was set to 1000 because it is the most complex to synthesize and thus will make the difference in performance more evident.

As can be noticed in Figure 4, with a small value of $stepsize$, the synthesis takes much longer, for instance with HD network and if only one frame is scheduler per each iteration, it takes more than 30 minutes. The lowest synthesis time occurs for $stepsize = 9$. There is a range of $stepsize$ where the synthesis time is only slightly above the lowest synthesis time, and when $stepsize$ takes a value in the range $[6 - 22]$, the synthesis time is only slightly above the lowest synthesis time, in no case it exceeds a 25% increase. The large synthesis time for the synthesizer outside the $[6-22]$ $stepsize$ range can be explained as a saturation caused by too many checks for lower $stepsize$, or caused by too many constraints to be solved at the same check for larger $stepsize$.

V. Conclusion

We have presented a performance study for SMT-based synthesis of time-triggered network schedules. We first migrated Steiner's tool [5] to an up-to-date SMT solver, which reduced the synthesis time in half for the most complex networks considered. Moreover, we studied the impact on the performance for the new background theories that an up-to-date SMT solver can offer. Integer Linear Arithmetic background theory reduced the synthesis time up to three times compared with previous works. Last, we have performed a study of the number of frames ($stepsize$) checked in the logical context in every iteration. Our finding is that a range

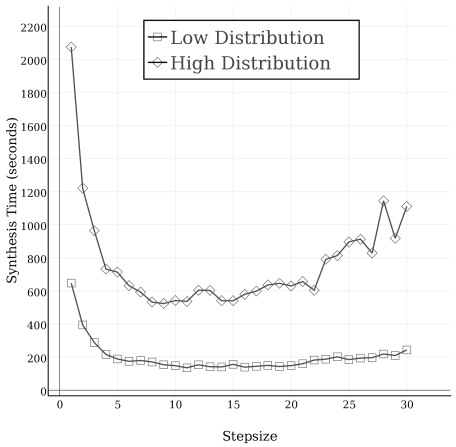


Fig. 4: Synthesis time of different $stepsize$ for 1000-frame networks

of $stepsizes$ ($[6 - 22]$) has better performance than other $stepsizes$ choices.

The improvement in performance brought for an up-to-date SMT solver and an appropriate tune of its parameters allows us to synthesize thousand-frames network in a more than acceptable time for industrial purposes. For future work, we want to develop a network test generator that allow us extend our study to larger networks and different topologies. Also we seek to perform a more rigorous study for the heuristic parameters of SMT solvers in order to keep reducing the synthesis time.

Acknowledgment

The research leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme FP7/2007-2013/ under REA grant agreement n°607727 and from the Swedish Knowledge Foundation (KKS), under project n° 20130048.

References

[1] H. Kopetz and G. Bauer, "The Time-Triggered Architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[2] D.-T. Peng, K. G. Shin, and T. F. Abdelzaher, "Assignment and Scheduling Communicating Periodic Tasks in Distributed Real-Time Systems," *Software Engineering, IEEE Transactions on*, vol. 23, no. 12, pp. 745–758, 1997.

[3] C. Hang, P. Manolios, and V. Papavasileiou, "Synthesizing Cyber-Physical Architectural Models with Real-Time Constraints," in *Computer Aided Verification*. Springer, 2011, pp. 441–456.

[4] L. De Moura and N. Bjørner, "Satisfiability Modulo Theories: Introduction and Applications," *Communications of the ACM*, vol. 54, no. 9, pp. 69–77, 2011.

[5] W. Steiner, "An Evaluation of SMT-based Schedule Synthesis for Time-Triggered Multi-Hop Networks," in *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*. IEEE, 2010, pp. 375–384.

[6] T. P. Baker and A. Shaw, "The Cyclic Executive Model and Ada," *Real-Time Systems*, vol. 1, no. 1, pp. 7–25, 1989.

[7] A. K. Mok and W. Wang, "Window-Constrained Real-Time Periodic Task Scheduling," in *Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings. 22nd IEEE*. IEEE, 2001, pp. 15–24.

[8] L. De Moura and N. Bjørner, "Satisfiability Modulo Theories: An Appetizer," in *Formal Methods: Foundations and Applications*. Springer, 2009, pp. 23–36.

[9] B. Dutertre, "Yices 2.2," in *Computer Aided Verification*. Springer, 2014, pp. 737–744.