Work-In-Progress: Safe and Secure Configuration Synthesis for TSN using Constraint Programming

Niklas Reusch, Paul Pop
Technical University of Denmark
Kongens Lyngby; Email: nikre@dtu.dk

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

***Abstract*—Time-Sensitive Networking (TSN) extends IEEE 802.1 Ethernet for safety-critical and real-time applications in several areas, e.g., automotive, aerospace or industrial automation. However, many of these systems also have stringent security requirements, and security attacks may impair safety. Given a TSN-based distributed architecture, a set of applications with tasks and messages, as well as a set of security and redundancy requirements, we are interested to synthesize a system configuration such that the real-time, safety and security requirements are satisfied. We use the Timed Efficient Stream Loss-Tolerant Authentication (TESLA) low-resource multicast authentication protocol to guarantee the security requirements, and redundant disjunct message routes to tolerate link failures. We consider that the tasks are scheduled using static cyclic scheduling and that the messages use the time-sensitive traffic class in TSN, which relies on schedule tables (called Gate Control Lists, GCLs) in the network switches. A configuration consists of the schedule tables for tasks as well as the disjoint routes and GCLs for messages. We propose a Constraint Programming-based formulation for this problem and we evaluate it on several test cases.**

I. Introduction

Many modern Cyber-Physical Systems (CPSs) are becoming increasingly more interconnected with the outside world opening new attack vectors [1], [2] that may also compromise safety. Therefore, the security aspects should be equally important to the safety aspects. Time-Sensitive Networking (TSN) [3], which is becoming the standard for communication in several application areas (e.g. automotive to industrial control), is a set of amendments to the IEEE 802.1 standards, equipping Ethernet with the capabilities to handle real-time mixed-criticality traffic with high bandwidth. Available traffic types are Time-Triggered (TT) traffic for real-time applications, Audio-Video Bridging (AVB) for applications that need bounded latency, but do not have hard real-time requirements, and Best-Effort (BE) traffic for non-critical applications.

We assume that safety-critical applications are scheduled using static cyclic scheduling and use the TT traffic type with a given *Redundancy Level* (RL) for communication. We consider that the task-level redundancy is addressed using solutions such as replication, and we instead focus on the safety and security of the communication in TSN. The real-time safety requirements of critical traffic in TSN networks are enforced through offline-computed schedule tables, called Gate Control Lists (GCLs), that specify the sending and forwarding times of all critical frames in the network. Previously, the synthesis of these schedules has taken into account different safety aspects, such as frame collision, temporal isolation, synchronization error, jitter constraints, and end-to-end latency requirements, but usually did not consider security aspects [4].

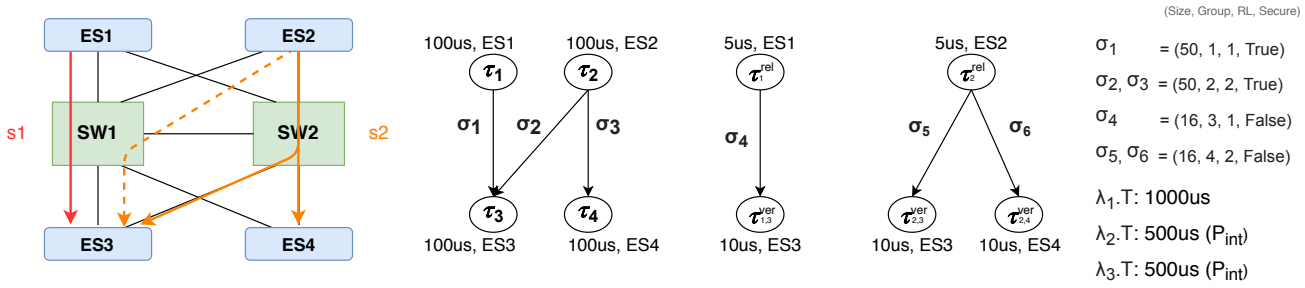
Since link and connector failures in TSN could result in fatal consequences, the network topology uses redundancy, e.g., derived with methods such as [5]. In TSN, IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER) enables the transmission of duplicate frames over disjoint routes, implementing merging of frames and discarding of duplicates. Regarding security, Timed Efficient Stream Loss-tolerant Authentication (TESLA) [6] has been investigated as a low resource authentication protocol for several networks, such as FlexRay and TTEthernet [7] networks. Scheduling time-sensitive traffic in TSN is non-trivial (and fundamentally different from TTEthernet), because TSN does not allow the control of individual frames as is the case in TTEthernet. Instead, only the status of the queue gates can be controlled via GCLs which may lead to non-determinism of frame transmission at runtime [4]. Additionally, adding security mechanisms such as TESLA after the scheduling stage is oftentimes not possible without breaking real-time constraints, e.g. on end-to-end latency, and degrading the performance of the system [7].

Contributions: We jointly address the safety and security requirements during the configuration of the TSN-based distributed CPSs, which means deciding on the scheduling of tasks, finding redundant disjoint routes for messages and generating the GCLs, such that the real-time, safety and security requirements are satisfied. We propose a Constrain Programming (CP) formulation combining constraints imposed by redundancy requirements, TSN and TESLA, and evaluate it on several test cases, including real-life applications.

II. Preliminaries

TSN: The key mechanisms that enable deterministic temporal behavior in TSN are the clock synchronization protocol defined in IEEE 802.1ASrev, which provides a common time reference with bounded precision, and the timed-gate functionality (IEEE 802.1Qbv) enabling the predictable transmission of communication streams according to the predefined times encoded in the GCLs. Using our CP-formulation we can determine the disjoint paths for 802.1CB FRER frames and 802.1Qbv-compatible GCLs for the network switches.

TESLA: TESLA provides a resource efficient way to do asymmetric authentication in a multicast setting [8]. It has been proposed as an alternative for resource constrained CPSs to traditional authentication approaches, e.g. symmetric authentication using shared keys (compromised if one node is taken over) or asymmetric authentication using public-key cryptography (too computationally expensive). The source of asymmetry in TESLA is a time-delayed key disclosure. TESLA uses cryptographic message authentication codes (MAC), e.g. HMAC. For further details we refer the reader to [8]. We use a modification of TESLA for real-time automotive systems described in [7], where bandwidth and computational resources are scarce. The modification is that instead of appending the currently disclosed key to every single frame, it is disclosed only once per interval in its own redundant frame. We assume that the protocol is bootstrapped by creating the necessary key-chains and distributing initial keys.



(a) Architecture (b) App. λ_1: (c) Sec. App. λ_2: (d) Sec. App. λ_3:

Fig. 1: Example Architecture and Applications

III. Models

A. Architecture and Application Models

We model our TSN network as a directed graph. The nodes of the graph are either end-systems (ESs) or switches (SWs), the edges represent directional network links. Each link has a certain speed. An ES runs a real-time operating system in which tasks run according to a periodic static schedule. See Fig. 1a for a small example architecture, where a black link represents a full-duplex link with 10 Mbit/s.

We use a similar application model to [7]. An application is modeled as a directed acyclic graph. The nodes represent *tasks*, the edges represent messages between tasks called *signals*, which define data dependencies. An application repeats periodically and has a set of function paths (FPs) which model a chain of tasks with an end-to-end deadline constraint.

A task is executed on the processor of a certain ES with a certain worst-case execution time (WCET). Moreover, it needs all the required incoming signals to arrive before it executes, and it produces its outgoing signals at the end of its execution.

A signal models a message and data dependency between a source task and a destination task. Source and destination task can be assigned to different ESs, in which case the signal has to travel across the network in an TSN stream. A signal has a certain size given in bytes and it can belong to a group. Signals belonging to the same ES and group are part of a multicast transmission and will be packed into the same TSN stream.

Critical signals have to deliver their data even in the case of permanent link failures and we assume that the appropriate *redundancy level* (RL), which captures the number of disjoint paths necessary, has been provided by the system designer. We call a signal and its carrier stream *critical* if the RL is larger than one. In addition, we call them *secure* if they are to be authenticated using TESLA.

Authenticating secure signals in our system with TESLA means that some additional applications, tasks, signals and streams are necessary, similar to [7]. We need to generate, send and verify a key in each interval for each set of ESs that communicate via a secure stream. Thus, for each ES that is sending signals over the network, we create a key authentication application with a period P_{int}, the interval length of TESLA.

Each of these application consists of one key release task scheduled on the sending ES and key verification tasks on each ES that receives some signals from the sending ES where the WCET of these tasks is scaled according to the speed of the respective ES. A key release just requires sending a message to the network while a key verification also requires one hash-computation.

Key signals are transmitted in dedicated streams where the size of the payload depends on the key size of the chosen MAC function. The redundancy level of all key signals is set to the max. RL of any signals on the sending ES. Moreover, each secure stream will get a MAC appended, thus increasing the frame size by the MAC length of the chosen MAC function. For each stream the MAC has to be generated at the sender and verified at the receiver, which requires one hash-computation on that ES.

B. Fault and Threat Model

Reliability models discussed in [5] (e.g., Siemens SN 29500) indicate that the most common type of permanent hardware failures are due to link failures (especially physical connectors) and that ESs and SWs are less likely to fail. Our disjoint routing can guarantee the transmission of the signal of RL n despite any $n-1$ link failures. For example, signals σ_2 and σ_3 in Fig. 1 have a RL of 2 and can thus tolerate the failure of any one link in Fig. 1a.

Similar to the threat model in [7], we assume that an attacker is capable of gaining access to some ESs or SWs in our system. We consider that the attackers have the following abilities: they know about the network schedule and the content of the streams on the network; they can control (block, delay, replay) frames which are routed through the SW they control; they can attempt to masquerade as another ES they do not control by faking the source address of streams they send; they have access to the key material in the ES they control; they can flood the network with many frames. The combination of TESLA and TSN protocol features (e.g., authentication, scheduling and filtering) can prevent such attacks.

IV. Problem Formulation

Given the architecture, application and security models introduced earlier, we are interested to synthesize an optimized configuration such that: all applications are schedulable (i.e., the end-to-end latency of FPs is less than their deadlines), the redundancy requirements of all signals are met (i.e., we can find disjoint routes for all critical signals) and the security constraints imposed by TESLA are fulfilled. Once the applications are schedulable, we are interested to minimize the latency of the FPs. Hence, synthesizing a configuration means determining: the routing of streams, the schedules for tasks and for messages (as 802.1Qbv GCLs) as well as the TESLA interval duration P_{int} for key releases.

Let us illustrate the problem using the architecture and applications in Fig. 1. Our applications period is 1,000 µs. P_{int} is determined to be 500 µs. Our application λ_1 has two function paths $\rho_1 = [\tau_1, \tau_3]$ and $\rho_2 = [\tau_2, \tau_4]$ both with a deadline of 1,000 µs. Signal σ_1 is packed into stream $s1$, σ_2

and σ_3 are packed into $s2$. The key signals of λ_2 and λ_3 are packed into their own separate streams (not shown).

With this setup, an optimized routing is depicted in Fig. 1a. A simple schedule can be seen in Fig. 2a. The red blocks visualize the transmission of streams $s1$ and $s2$ on network links. The blue blocks visualize tasks executed on the processing element of the end-systems. To guarantee deterministic schedules in TSN we have to isolate the frames in the time domain, leading to the delay of f_1 and thus τ_3. See [4] for a discussion on non-determinism in TSN.

However, this simple schedule does not consider the redundancy or security requirements of the signals. Considering them results in the schedule shown in Fig. 2b. Note how the black dashed line separates the TESLA key release intervals. Streams carrying keys are orange, key generation tasks purple, key verification tasks green and the MAC generation/validation operations on ES red. Especially interesting to see is the delay incurred by the time-delayed release of keys. Tasks τ_3 and τ_4 can only be executed after the keys authenticating $s1$ and $s2$ have arrived in the second interval, and after key verification and MAC validation tasks have been run.

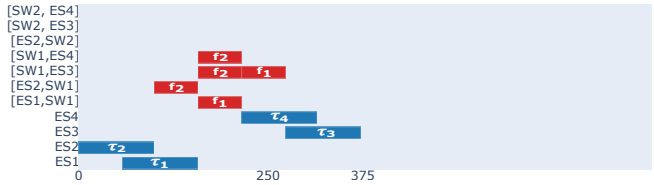
V. Constraint Programming-based Solution

Several related scheduling problems have been successfully solved using Integer Linear Programming [7] and Satisfiability Modulo Theories [4]. Hence, we have decided to use a Constraint Programming (CP)-based solution. In our solution, we decompose the problem into separate constraint optimization problems, determining: (1) P_{int}; (2) stream routes; and (3) task and stream schedules, using determined routes and P_{int}. Although such a problem decomposition does not guarantee optimality, it increases the scalability, and we have found it to obtain good results in practice. For brevity reasons, we do not formalize the constraints here and refer the reader to [9] for a complete formulation.

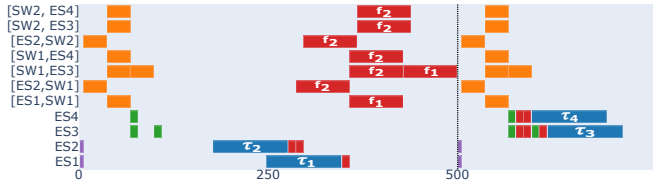
A. P_{int}

We constrain P_{int} to be the maximum value for which we can satisfy the latency requirements of our system, while we fix d to 1, similar to [7].

Constraints: P_{int} is small enough to allow function paths to be authenticated before their deadline. P_{int} is smaller than the minimum period of all applications. P_{int} is a factor of the greatest common divisor of all application periods.



(a) Schedule without security & redundancy: TESLA is not used for security and redundant routing is not used for fault-tolerance



(b) Schedule with security & redundancy: TESLA is applied to secure the streams and redundant routing is used for fault-tolerance

Fig. 2: Example solution showing schedules for the architecture and applications in Fig. 1

Optimization Objective: Maximize P_{int}. This will minimize the CPU and bandwidth overhead of the security applications.

B. Stream Routing

The routing problem consists of finding multiple disjunct (non-overlapping) minimum-cost spanning trees between a source node and one or more destination nodes on a directed weighted graph. Each edge of our graph (network link) is assigned the same weight, such that the minimum-cost spanning tree represents the shortest route. Our solution is inspired by [10].

Iteratively selecting the minimum-cost spanning tree for each redundant route may lead to sub-optimal results (e.g. for trap topologies). Our heuristic is that by minimizing the total number of links used in the disjoint routes, we can improve the latency of FPs.

Routing Constraints: The stream sender must have a successor (on the route), all stream receivers must have a predecessor, and each switch which is a successor of a node must also have a successor. The sum of the bandwidth used by all streams routed through a link may not exceed the available bandwidth. All redundant copies of a stream cannot have common links on their routes and a route may not contain cycles.

Optimization Objective: Minimize the total sum of links used by all routes.

C. Task and Stream Scheduling

The scheduling problem consists of finding a network schedule such that all deadline requirements are satisfied, the TESLA security condition is fulfilled and the TSN frame isolation constraints are applied. Some constraints are adapted from [7] and [4].

Stream Constraints: A stream is only scheduled on the ESs and links of its route; its transmission time on a link is equal to the stream size divided by the link speed. Streams may not be transmitted on a link before they have been fully received at the link's source and they may not overlap on the same link. A stream's execution time (for MAC generation/verification) on an ES is equal to one hash worst-case computation time.

The key corresponding to a stream's MAC may only be released in the interval after the stream has been fully received. The key corresponding to a stream's MAC has to be verified before the MAC can be verified.

For each link connected to an output port of a SW, the streams arriving at all input ports of that SW that want to use this output port, cannot overlap in the time domain (TSN frame isolation for determinism, see [4] for more details).

Task Constraints: A task may not start before its incoming streams have arrived and are authenticated and it has to finish before its outgoing streams can be sent. Tasks' execution does not overlap with themselves or stream MAC generation/verification.

Function Path Constraint: The end-to-end latency of each function path is smaller than its the deadline.

Optimization Objective: Maximize the sum of all function path laxities, whereby the laxity is the difference of the deadline and latency of a function path.

VI. Evaluation

Our CP approach was implemented in Python 3.7 using the CP-SAT solver from Google OR-Tools. The evaluations were run with an i7-8565U CPU and 16 GB of DDR4-RAM. We have used the following test cases, with varying topologies, see their details in the first five columns of Table I: the example used in this paper (TC0), a realistic automotive test case from General Motors (TC1), a medium-sized automotive

TABLE I: Evaluation of our Constraint Programming approach

					No TESLA				ASAP					OPT			
	ES	SW	Tasks	Signals	Lax.	Bw.	Util.	Runtime	Lax.	Bw.	Util.	Runtime	Missed D	Lax.	Bw.	Util.	Runtime
TC0	4	2	4/9	3/6	1,367	2.88	10	0.1 s	314	6.56	14	0.1 s	0/2	1007	6.56	14	0.1 s
TC1	20	20	47/115	48/96	150977	0.15	2.73	30 min	-3233466	0.69	5.74	6.5 min	48/48	33995	0.69	5.74	30 min
TC2	6	1	24/37	21/28	24718	0.99	18.59	6 min	-3207	4.77	21.01	1 s	4/7	19046	4.77	21.01	20 s
TC3.1	4	2	16/20	5/8	2997	0.64	49.60	0.6 s	-1422	0.71	56.10	0.2 s	4/5	2801	0.71	56.10	1 s
TC3.2	8	4	32/41	12/19	6797	1.05	49.62	2 s	-2447	1.13	57.46	0.6 s	8/12	6330	1.13	57.46	16 s
TC3.3	16	8	64/80	27/40	16000	0.59	49.53	10 s	-5524	0.64	55.83	15 min	19/27	14856	0.64	55.83	21 min
TC3.4	32	16	128/161	52/79	27951	0.32	49.59	30 min	-11417	0.35	55.32	30 min	40/52	26424	0.35	55.32	30 min

case study from [11] (TC2) and synthetic industrial test cases of increasing size inspired by [12] (TC3.1–3.4). All testcases contain security and safety-critical (with RLs between 1–3) applications and the topologies support redundant routing, i.e., disjoint routes can be constructed where necessary. As MAC function for TESLA, we chose *HMAC-MD5* with a key length and MAC length of 16 B and a hash computation time of 10 µs on all ESs. The MTU is 1,500 B and an Ethernet frame has an overhead of 22 B. The link speed is 10 Mbit/s for TC0 and TC2, 100 Mbit/s for TC1 and 1000 Mbit/s for TC3.1-3.4.

We were interested to evaluate the ability of our CP implementations to derive optimized solutions using TESLA, see the results in Table I. We used two CP implementations: *ASAP*, which uses optimized P_{int} and routing, but uses a heuristic where tasks and messages are scheduled as-soon-as-possible when their resources are available without considering end-to-end latency constraints; and *OPT*, which is our full CP solution that also optimizes scheduling according to the optimization objective in Sect. V-C and function-path deadline constraint. ASAP and OPT were compared on the cost function (*Lax* in the table), Bandwidth (*Bw.*, mean bandwidth used on all network links) and the CPU utilization of ESs (*Util.*). Note that a larger *Lax* is better, and when FPs miss their deadlines, the value can be negative. *Runtime* is the time it took to run the CP solver.

In the table we also show the results for an implementation we call *No TESLA*, which is our *OPT* implementation without applying TESLA for security, which means that there are less tasks and signals (first number in the Tasks/Signals column) at the cost of security and safety. As expected, TESLA introduces overheads in terms of *Lax*, *Bw* and *Util*, see the *OPT* columns compared to *No TESLA*. We expect that for larger realistic test cases, where only a small part of applications are security- and safety-critical, the overheads introduced by TESLA via our CP implementations will be relatively small.

As we can see from the table, our proposed OPT solution is able to find feasible solutions. Although applying TESLA introduces overheads compared to *No TESLA*, OPT is able to significantly reduce these overheads through optimization. Regarding scheduling optimization, when comparing OPT to the ASAP solution, which does not optimize the schedules, we can see that OPT improves the laxity results significantly.

Using the synthetic test cases (TC3.1–3.4) of increasing size we evaluate the scalability of our solution. The synthetic test cases feature mesh topologies with increasing number of ESs and SWs. Each ES runs 8 tasks of which 4 communicate over the network. We can see that the runtime increases exponentially with the test case size. We have used a time limit of 30 minutes for all runs. If the solver has finished before this time limit (see the *Runtime* columns), then the optimal solution (considering our problem decomposition) has been obtained; otherwise, it returns the best solution found.

VII. Conclusion

In this paper we have addressed distributed safety-critical cyber-physical systems that use TSN for communication. We have considered mixed-criticality real-time applications that use static cyclic scheduling for tasks and the time-triggered traffic type for messages; applications have both fault-tolerance and security constraints, addressed via redundant routing and TESLA, respectively.

We have developed a method to derive the necessary tasks and signals needed for the use of the TESLA authentication protocol. We have formulated constraints that guarantee the timing properties of applications using TSN and the security properties of TESLA, as well as the redundancy constraints of streams to tolerate permanent link failures. These constraints have been used with a CP solver to synthesize stream routes and combined task and network schedules.

We have evaluated the impact of TESLA in terms of application latency and network bandwidth. As the results show, we are able to derive optimized implementations that can significantly reduce the latency overheads.

In the future we would like to extend this work by comparing the CP implementation with a heuristic, adding more testcases and discussing the threat model in more details.

References

[1] T. Pereira, L. Barreto, and A. Amaral, "Network and information security challenges within Industry 4.0 paradigm," *Procedia Manufacturing*, vol. 13, 2017.

[2] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaaniche, and Y. Laarouchi, "Survey on security threats and protection mechanisms in embedded automotive networks," *Proc. DSN*, 2013.

[3] IEEE, "Official Website of the 802.1 Time-Sensitive Networking Task Group," `http://www.ieee802.org/1/pages/tsn.html`, 2016, accessed: 28.09.2020.

[4] S. S. Craciunas, R. Serna Oliver, M. Chmelik, and W. Steiner, "Scheduling real-time communication in IEEE 802.1Qbv Time Sensitive Networks," in *Proc. RTNS*, 2016.

[5] V. Gavrilut, B. Zarrin, P. Pop, and S. Samii, "Fault-tolerant topology and routing synthesis for ieee time-sensitive networking," in *Proc. RTNS*, 2017.

[6] A. Perrig, R. Canetti, D. Song, and J. D. Tygar, "Efficient and secure source authentication for multicast," in *Proc. NDSS*, 2001.

[7] R. Zhao, G. Qin, Y. Lyu, and J. Yan, "Security-aware scheduling for TTEthernet-based real-time automotive systems," *IEEE Access*, vol. 7, pp. 85 971–85 984, 2019.

[8] A. Perrig, D. Song, R. Canetti, J. D. Tygar, and B. Briscoe, "Timed efficient stream loss-tolerant authentication (tesla): Multicast source authentication transform introduction," Internet Requests for Comments, RFC Editor, RFC 4082, June 2005.

[9] N. Reusch, P. Pop, and S. Craciunas, http://people.compute.dtu.dk/nikre/technical_report.pdf, Tech. Rep., 2020, accessed: 28.09.2020.

[10] Q. D. Pham and Y. Deville, "Solving the quorumcast routing problem by constraint programming," *Constraints*, vol. 17, no. 4, pp. 409–431, 2012.

[11] N. Kandasamy, J. P. Hayes, and B. T. Murray, "Dependable communication synthesis for distributed embedded systems," *Reliability Engineering and System Safety*, vol. 89, no. 1, pp. 81–92, 2005.

[12] S. S. Craciunas and R. S. Oliver, "Combined task- and network-level scheduling for distributed time-triggered systems," *Real-time Systems*, vol. 52, no. 2, pp. 161–200, 2016.