33RD CONGRESS
OF THE INTERNATIONAL COUNCIL
OF THE AERONAUTICAL SCIENCES
STOCKHOLM, SWEDEN, 4-9 SEPTEMBER, 2022

ICAS 2022 SWEDEN

CONCEPT LEVEL EXPLORATION OF IMA-BASED NETWORKED PLATFORMS WITH MIXED TIME-SENSITIVE COMMUNICATION REQUIREMENTS

Rodrigo Saar de Moraes & Simin Nadjm-Tehrani

Department of Computer and Information Science, Linköping University, Linköping, Sweden

Abstract

In this paper, we apply a grammar-based approach to generate computation and communication platforms for avionic applications with mixed classes of time-sensitive communication messages. Then, we propose an evolutionary algorithm to schedule communication in the platform considering the interaction between time-triggered and bandwidth-constrained traffic. Together, the platform generation approach and the scheduling algorithm support the exploration of avionic systems at the concept level.

Keywords: platform architecture exploration; time-sensitive networks; mixed-critical communication; communication scheduling; timeliness analysis

1. Introduction

Due to their distributed computation structure, new IMA systems show an increasing demand for deterministic communication behavior. However, guaranteeing such behavior is still a challenge for the avionics industry. One part of the problem is the choice of a communication standard, which is heavily dependent on the type of application and its requirements. The other is the conceptualization and design of computation and communication platforms (henceforth referred to as a platform) which must provide adequate resources to meet the requirements of the aforementioned application.

While Ethernet (IEEE 802.3) has long been the preferred communication standard for a wide range of non-time-sensitive application domains due to its cost-effectiveness, its lack of determinism is a challenge to its application in time-sensitive domains. To adapt Ethernet to meet the demand for deterministic communication behavior and guarantee the interoperability of different flows within the same network, a series of standards such as Time-Sensitive Networking (TSN, part of the IEEE 802.1 standard), time triggered Ethernet (TTEthernet) and Avionics Full-Duplex Switched Ethernet (AFDX) have been proposed.

Simultaneously, applications with different criticality levels require different guarantees. Some require strict time-triggered communication, others require rate-constrained communication and bounded latencies, and the least strict can even accept best-effort communication.

As the most restrictive of the traffic classes, Time-triggered (TT) traffic, requires schedule tables that define the exact queue transmission times of frames on every egress port along the route of the respective traffic flows. The generation of these schedule tables, usually called gate-schedules or gate control lists, however, is an NP-hard problem, which can become even harder to solve once the routing of traffic flows is considered. Among several techniques described in the literature to solve the problem, the most common are heuristics, such as genetic algorithms [11] [12], and Satisfiability/Optimization Modulo Theories (SMT/OMT) [5].

Rate-constrained (RC) traffic, on the other hand, is a little bit less restrictive. While TT traffic requires the static scheduling of messages in the network, RC traffic allows the sender to transmit at any time but enforces the separation of different flows using two limiting parameters: the minimum duration between two successive frames at the source (known as the bandwidth allocation gap, or BAG); and

maximum frame size. These parameters enable network controllers to restrict the amount of data sent in each flow, which can be used to derive an upper bound on the network latency on the flows. While not as deterministic as TT traffic, RC traffic can still provide enough guarantees on end-to-end latency while being easier and cheaper to implement since it does not require different nodes to be perfectly synchronized.

Even though both traffic classes work very well individually, real-world systems with heterogeneous application subsystems frequently see the interaction of these two traffic classes in the links of the platform. As TT traffic is scheduled for transmission at specific slots, while RC frame can be sent almost at any time, their interaction creates inferences that can undermine the real-time guarantees on determinism or latency of the individual traffic classes.

Despite the advent of many standards for time-sensitive traffic (i.e. TSN, TTEthernet, AFDX, etc.), the topology design part of the problem still remains. Timeliness guarantees and communication determinism are inherently related to the platform on which we deploy the application. Therefore, chosing the correct platform is an important step in making sure applications behave accordingly once deployed. Chosing platforms is a complex optimization problem due to the vast number of choices and different paradigms to be considered when deciding on the architecture and topology.

Due to the complexity of the problem, several techniques have been proposed to tackle platform design throughout the decades. Some works such as Fischer et al. [9], Zhang et al.[13] and Dengiz et al.[8], use genetic algorithms to generate industrial network topologies considering real-time constraints. Other works resort to more robust and exact approaches, such as linear optimization or constraint-satisfaction problems [3]. A third approach is to use hybrid techniques, such as Feng et al. [10], who uses use a control-flow-based heuristic to generate architectures compliant with AFDX specifications and real-time performance requirements.

In this work, we apply NetGAP, our previous work [7] that provides a method based on graph-grammars for various concept level trade-offs (including security, fault tolerance, timeliness) to an aerospace related use case where communication timeliness is in focus. More specifically, we use NetGAP to automatically generate candidate platform designs that are guaranteed to perform well under strict communication constraints for application with mixed traffic classes. As a new contribution we propose the use of a fast genetic algorithm to generate and analyze a communication schedule that considers the interaction of time-triggered traffic and rate-constrained traffic on the platform. We illustrate the methodology with the application to an industrial use case composed of a mix of TT and RC traffic flows and evaluate a candidate platform generated by the grammar-based method. In the discussion, we show that the two methods together are a fast and useful solution to support the exploration of avionic systems at the concept level.

The paper is organized as follows: Section 2 provides an overview of the platform generation process used on this paper. Section 3 provides details of the methodology such as the implementation of the genetic algorithm. Section 4 shows the results of the application of the proposed methodology. Meanwhile, Section 5 presents our conclusions and suggestions for future works.

2. Background

2.1 On graph-grammar-based approach to platform generation

For the sake of completion and readability, in this section we restate some parts of the methodology framework from our previous work [7] that relate to the generation of candidate topology designs with graph grammars.

In our methodology, candidate platform designs are abstracted as directed graphs in which nodes represent platform components (i.e. processing modules or switches) and edges represent physical network links between the components. In this framework, a candidate platform design is constructed through the iterative application of graph transformations, orderly adding or removing nodes and/or edges, until a candidate design with the desired characteristics is found. We express each of these transformations as a rule in a custom-built networked systems graph-grammar [6] that limits the topology arrangement of the platform. An example of a networked systems grammar is given in Listing 1, a deeper view into the lexical and semantics of the proposed grammar can be found in the original paper [7]. Figure 1 illustrates the iterative process of generating a candidate design using the

gramar in Listing 1.

Listing 1: A simple grammar example

r0 : $\phi \Rightarrow S$ # Adds switch node to an empty graph
r1 : $S \Rightarrow$ S $\leftrightarrow M$ # Adds module M to switch S
r2 : $S_1 \Rightarrow S_1 \leftrightarrow S_2$ # Adds switch S_2 to switch S_1
r3 : $S_1, S_2 \Rightarrow S_1 \leftrightarrow S_2$ # Connects two switches S_1 and S_2

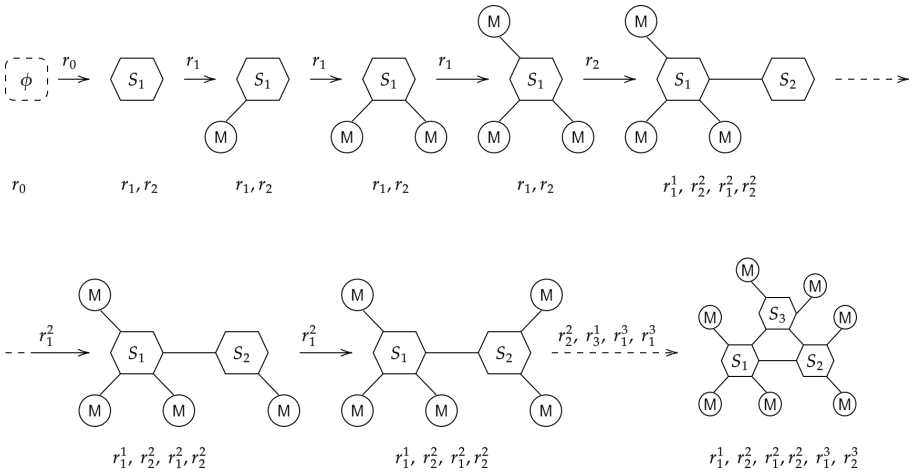

Figure 1 – Grammar-based generation of a networked platform using the grammar of Listing 1. The list of rules below the diagrams indicates the set of possible rules available on each step of the process. Meanwhile, the labels above the arrows indicate the rule applied to proceed to the next step.

As illustrated in Figure 1, each step of the process generates a graph representing a partial platform design. For each partial design, there might exist multiple rules that can be applied to the graph at that stage. Furthermore, the application of a rule in any given stage limits and influences and limits the types of platforms obtained from that point on in the process. The choice of the best rule to apply at each step is, therefore, crucial.

By nature, this type of problem can be encoded as a tree, in which each node represents a partial platform design, which we will call the state of the system, and each edge represents the action of applying a transformation rule to the graph. In our methodology, we use a Monte-Carlo tree search algorithm (MCTS)[4] to explore the tree and decide which action to take at every step.

In general words, the Monte-Carlo tree search is an iterative process that allows for efficient evaluation of tree branches by means of Monte-Carlo sampling. Starting at the root node, multiple random simulations are run in succession in order to estimate which branch of the tree leads to the best solution to the problem. During each of these simulations, a tree branch is selected and expanded into its sub-branches. After the expansion, a random sub-branch is chosen and a depth-first search is performed until a leaf node representing a complete platform design is found. The leaf node is then evaluated according to a and it's fitness is back-propagated through the tree. In the end, the action leading down the branch with the highest fitness is chosen and the transformation rule associated to this action is applied to the graph.

3. Methodology

The methodology presented in this work consists of a two-step process. In the first step, a graph-grammar-based approach is used to generate a candidate time-sensitive network platform to support a real-time application. In the second step, the platform generated in the previous step is analysed for schedulability of time-triggered data using a genetic algorithm approach that considers the interplay between time-triggered and rate-constrained traffic classes. Note that the methodology we present in this paper is centered around guiding and supporting the exploration of concept platform designs and does not aim to find optimal solutions to the addressed problems.

3.1 Part 1: Platform Generation

Up to this point, the platform generation process discussed in Section 2 is very generic and does not define what type of platform should be generated. In order for the platform generation approach to know what it should optimize for, we need to provide it with three important pieces of information: a system-specific grammar; an application model; and a fitness function to define how to evaluate a candidate platform with respect to the requirements of an application. In the context of this work, we will define our platform based on the grammar presented in Listing 1. The avionic application will be presented later in Section 4.1, for now, it suffices to know it contains both time-triggered and rate-constrained traffic classes. The evaluation function, in turn, is presented below.

3.1.1 Platform generation evaluation function

The platform generation objective is expressed in the form of a maximization problem. The optimization objective, as implemented, is a weighted sum of three characteristics of interest: path redundancy for time-triggered traffic; estimated latency; and total platform cost.

$$max \quad \frac{\alpha(redScore) + \beta(latScore) + \gamma(costScore)}{\alpha + \beta + \gamma} \tag{1}$$

The $redScore$ term represents the ratio between the number of time-triggered flows with redundant paths between source and destination (tt_{red}) and the total number of time-triggered traffic flows (tt_{total}). Behind this term is the assumption that time-triggered flows are usually of the utmost importance for the system, therefore, they must be resilient to failures of network components in their path. Furthermore, having multiple routing options to choose from when trying to define a static schedule for these flows helps avoiding overloaded paths, which potentially contributes to lower end-to-end latency. This term is defined by:

$$redScore = \frac{tt_{red}}{tt_{total}} \tag{2}$$

Meanwhile, the $latScore$ is used to estimated the overall latency experienced by the application traffic flows on this platform. The term is a negative exponential composed of 3 evaluation metrics relating to network performance. The first metric, b_{max}, is related to the maximum bandwidth utilization observed in any link e_{ij} between two nodes n_i and n_j. This metric assumes that all traffic flows will be routed following the shortest path between it source and destination, and is defined as the ratio between the current allocated bandwidth and the maximum theoretical bandwidth supported by link e_{ij}. The second metric, v_t represents the number of links in which $b_{max} \geq 1$, meaning link in which the utilization is bigger than 100% (which should not be possible). The third and last metric, h_avg is the average number of network nodes each traffic flow has to go through when traversing the network through it's shortest path. $latScore$ is defined as:

$$latScore = e^{(b_{max} * h_{avg} - v_t)} \tag{3}$$

Finally, the last term $costScore$ is the current solution cost in monetary units. To calculate the costs, each node on the network is assigned a value of 100 units and each link is assigned a value of 1 unit.

3.2 Part 2: Network Traffic Schedule generation

In order to guarantee whether our generated platform is able to support a set of applications with different classes of time-sensitive traffic, we also need to consider whether we can provide assurances that the data going through the network will be delivered in time. In this second part of the process we consider the interplay between time-triggered traffic and the rate-constrained traffic in network links to establish a feasible routing scheme for the time-triggered flows in order to minimize the latency for both traffic classes. Simultaneously, we generate a valid static-schedule for all time-triggered traffic flows and generate a gate-schedule to control the access of time-triggered flows to the network links. As with the platform-generation step, the process is designed to support the exploration of concept platform designs and does not aim to find optimal solution.

3.2.1 Genetic Algorithm Scheduler

In this work, a genetic algorithm is used to solve the traffic scheduling problem for a fixed platform and application model with different traffic classes. Figure 2 shows a flowchart of the proposed genetic algorithm. The specifics of the implementation, such as the chromosome representation, the mutation and crossover operators, and the chosen fitness functions are described in the following subsections.

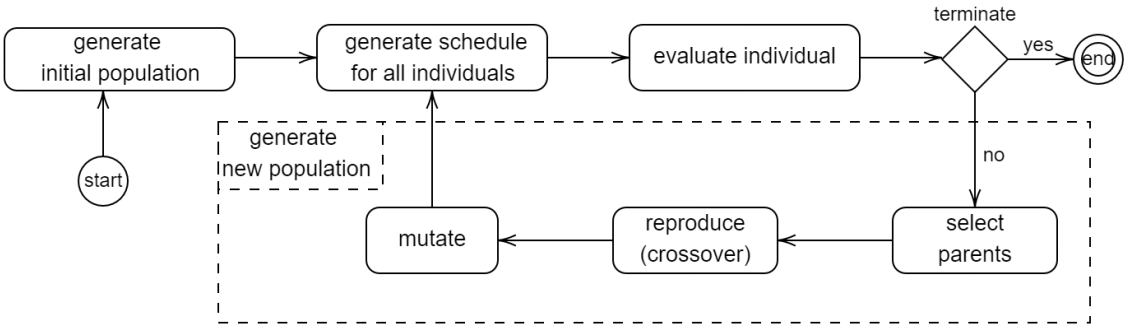

Figure 2 – Structure of the proposed genetic algorithm.

3.2.2 Chromosome representation

As usual, genetic algorithms represent candidate solutions in the form of a genome, with individual genes representing individual characteristics of the solution. In this study, inspiration was taken from Hyeong et al.[11] where genes are used to represent an ordering of flows to be considered during the schedule generation process, and from [12], where genes are used to represent the routing choices for each flow. The implemented genome is composed of two halves composed each of n genes (to total size of $2n$ genes) as shown in Figure 3. The first n genes represent an absolute order in which a flow should be considered when generating a schedule (thus providing this flow with a virtual priority of sorts). Therefore, the first (last) gene in contains the identifier of the traffic flow that was considered first (last) in the schedule generation process. The second set of n genes represents the path chosen for the respective flow from the list of possible paths. Note that, in the second part of the genome, gene n_j gene relates to the path chosen by the flow with identifier j (henceforth f_j) and not by the j_{th} flow from the first half of the genome.

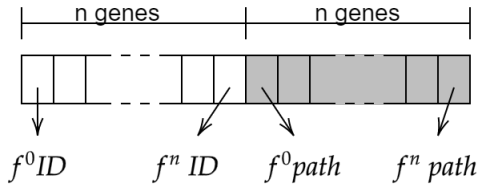

Figure 3 – Chromosome representation of ordering and route.

3.2.3 Mutation and Crossover Operators

Since the genome is divided in two different parts with different characteristics, two different mutation and crossover operators are implemented. In the first part of the genome, the important characteristic is the relative order of the flows. In other words, it is important for flow f_i to come before flow f_j. For these types of problems, the ordered crossover operator (also known as OX operator) is amongst the best performers [1]. For this reason, the OX operator has been chosen for first part of the genome. On the other hand, in the second part of the genome only the absolute value of the gene is important. Therefore, a simple single point crossover operator was implemented for that part. The mutation

operators were chosen according to the same logic. Where order mattered, a RSM operator [2] was chosen. Where order did not matter, genes were mutated randomly.

3.2.4 Schedule Generator

This additional module, is responsible for transforming a genome into a candidate schedule for the time-triggered flows. Taking each flow f_i as they appear in the first half of the genome, the schedule generator injects the flow into the links that compose the path selected for f_i in the second half of the genome. Every time a new flow f_i is inserted into a link e_j, the schedule generator searches for empty slots in the gate-schedule controlling that link considering the necessary guard band, the processing delay on the interface, and the transmission delay for the packet to leave the interface. Once a new slot is found, the gate-schedule for the link e_j is updated and the process restarts for the next link e_{j+1} in the path of f_i. Note that it is assumed that a packet can only be transmitted forward once it has been completely received at the end of a link, therefore the the search for a slot for f_i on the gate-schedule of e_{j+1} starts at the end of the slot of f_i in e_j (there is no pass-through). Once f_i has reached it's destination, the process restarts for the next flow in the ordered portion of the genome. Listing 2 describes the pseudo-algorithm of the schedule generator module.

Listing 2: Schedule generator module

```
Inputs: genome,
Outputs: gate_schedule, fitness

gate_list ← empty list

fitness  ← array [genome.size]

for flow in genome.flow_list:
   n_cycles ← 1600us/flow.period
   path ← genome.path_list[flow]
   latency  ← array [n_cycles]
   for n in [0,n_cycles]:
      start ← 0*n
      for hop in path_list:
         slot ← find_next_available_slot(gate_list)
         start ← slot.end
      latency[n] ← calculate_cycle_latency()

   fitness[flow]← calculate_fitness(latency)

total_fitness ← sum(fitness)

return gate_schedule, total_fitness
```

Note that, for convenience purposes, the schedule generator works on a fixed scheduling window defined as the least-common-multiplier of the periods of all flows. This scheduling windows is called a hyper-period after [11] and refers to the shortest period in which the transmission pattern of all flows is repeated.

3.2.5 Schedule Generation fitness function

In this work, the fitness function for the evaluation of individuals in the GA is a weighted sum of two terms. The first part of the evaluation function relates to the evaluation of the quality of the schedule of time-triggered traffic. The second, deals with the interaction between time-triggered and rate-constrained traffic.

For the first part, this study takes inspiration from [11] to evaluate the fitness of each individual of the population. In line with the cited work, three indicators of the performance of the time-triggered schedule represented by each solution were chosen.

The first indicator was inspired on the end-to-end latency performance indicator from [11] and reformulated to fit the implementation presented here. In this study, the latency performance indicator expresses the percentual difference between the calculated latency for a given TT flow, and the ideal

latency for the same flow under perfect circumstances (i.e. when there is no queuing and packets are able to traverse a switch with no delay). This metric is defined as follows:

$$LPI = \sum_{i=0}^{n} \frac{1}{k} \sum_{j=0}^{k} \left(\frac{d_{calc}^{k,i} - d_{ideal}^{i}}{d_{ideal}^{i}} \right) \tag{4}$$

where d_{ideal}^{i} represents the ideal latency for flow f_i, and $d_{calc}^{j,i}$ represents the calculated latency for instance j of a packet pertaining to flow f_i.
The second indicator, is the jitter observed on flow f_i. This parameter is calculated as the mean of the standard deviation of the latencies observed for each instance of a packet pertaining to flow f_i during the scheduling window.
The third and last parameter is ratio between the number of individual packet deadline overruns observed across a hyper period and all packets sent during a scheduling window. It is calculated by:

$$DO = \frac{1}{m} \sum_{i=0}^{m} v_i \tag{5}$$

where $v_i = 1$ in case packet i has missed its deadline and $v_i = 0$ otherwise, and m is the total number of packets sent during a hyper-period.
The second part of the evaluation function is a simple evaluation of the bandwidth left over by the time-triggered traffic to the rate-constrained traffic. Since the time-triggered traffic is generally well behaved and easy to control, one can chose its routing as to leave the more direct or busier links available for the rate-constrained traffic flows. Therefore, the idea is to minimize the bandwith taken by time-triggered traffic across all links.

4. Experimental Results

4.1 Avionic Application Use-Case

As part of the assessment of the proposed methodology, the platform generation workflow was applied to a synthetic industrial use-case provided by SAAB AB. The use-case consists of a synthetic avionic application model constructed to look like and have the same properties of those actually deployed by the partner in an early concept design stage. It is divided into two independent parts, a mission-oriented part (MOP) and a flight-critical part (FCP). The mission-oriented part contains 8 applications and 31 data-flows, and the mission-oriented part is composed of 44 applications and 629 data-flows. For the purpose of this study, the processes of the different parts were allocated to processing modules before the platform generation process started. As a result of the allocation, it was established that the platform should contain 14 processing modules. The data-flows pertaining to the MOP part were considered to be time-triggered, while the MOP data-flows were classified as rate-constrained flows.
For convenience, the payloads of the packets composing the time-triggered flows were chosen so that the periods and deadlines of these flows were harmonic. Figure 4 presents the characteristics of the time-triggered flows, while Figure 5 shows a graphical representation of the total bandwidth required by all rate-constrained traffic flows between any two nodes in the platform.

4.2 Platform Generation Results

For the purpose of the platform generation part of the solution, the grammar presented in Listing 1 is used in order do describe and to generate generic switched-mesh topologies. Figure 6 shows the final platform generated to support our application. Here, the S terms denote network switches and the M terms denote computation modules. The final platform has a $latScore$ of 0.12, with the maximum bandwidth utilization of 97% being observed in the link *Sg0 - SY10*, while the average number of hops per traffic flow amounts to 2.2. The platforms manifests a $redScore$ of 0.94, meaning 29 of the 31 TT flows had more than on possible route to their destination. The platform is composed of 14 modules, 6 switches and 48 directed links (24 full-duplex links), totalling a cost of 2048 monetary units.
The process of generating the platform in Figure 6 took 56.01s, attesting to the scalability of the approach.

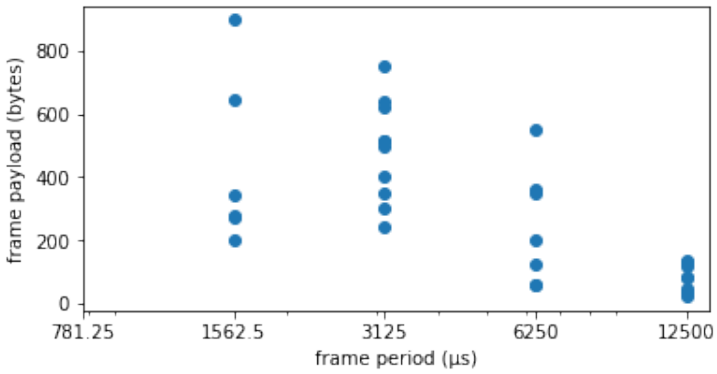

Figure 4 – Payloads and frame periods of the time-triggered traffic

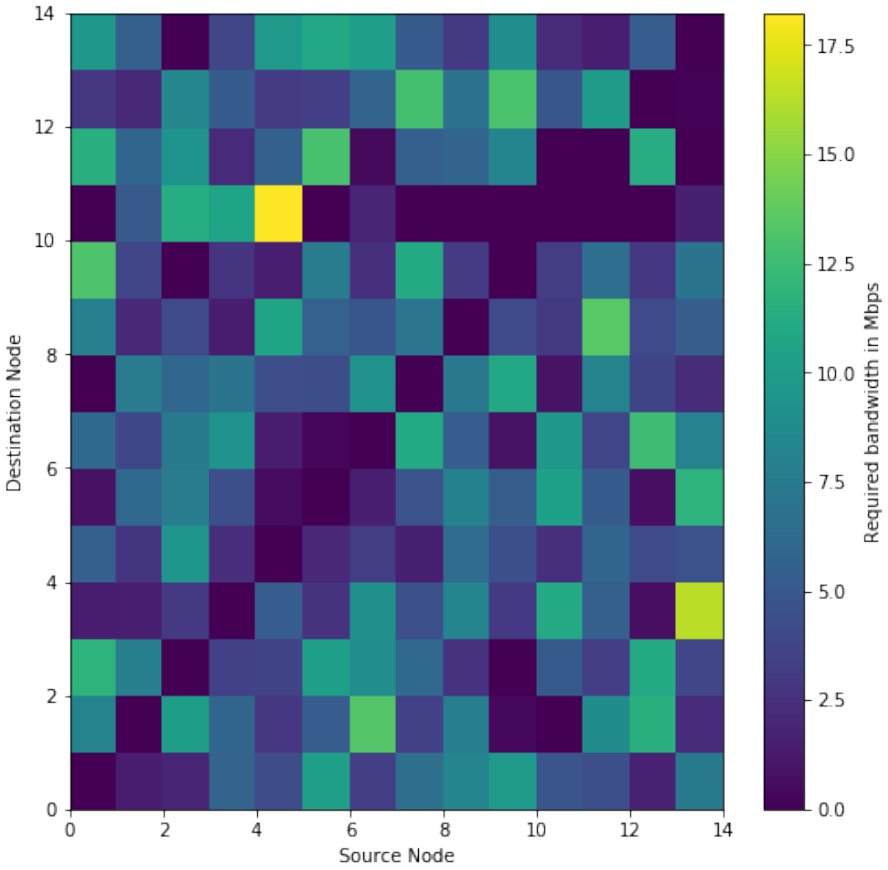

Figure 5 – Combined bandwidth requirements of rate-constrained traffic between nodes

4.3 Schedulability Results

Table 1 shows the parameters used in by the genetic schedulability algorithm during the test runs, in total, 30 experiments were run. Figure 7 shows a comparison between the average latencies obtained by the genetic scheduler and by a random scheduler. It is clear that the genetic scheduler presents lower latencies for almost all flows when compared to the random case, with the difference being exacerbated for flows 3 and 16, which turned out to be very difficult to schedule even for the genetic scheduler. Figure 8 illustrates one gate control list (GCL) obtained by the genetic schedulability algorithm for the first 24 interfaces (out of 50).

In all cases, there were no deadline overruns and the jitter for each time-triggered flow was reduced to zero (meaning it was possible to maintain constant and uniform latency times across all packets belonging to the same flow). When it comes to scalability, the gate-schedule of Figure 8 was found in $62.55s$, indicating that the method tends to be scalable.

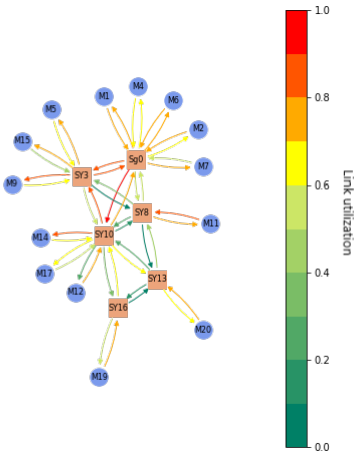

Figure 6 – Platform chosen as target for the schedulability verification. The color of the edges between different nodes represents the observed load on that link.

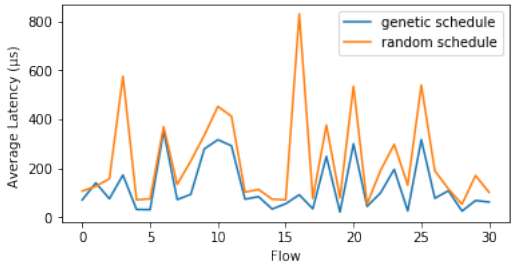

Figure 7 – End-to-end latencies obtained for each flow by a random scheduler and by the genetic scheduler.

network bandwidth	100Mbps
number of generations	60
genome size	62
population size	250

Table 1 – Parameters of the genetic algorithm experiment

5. Conlusion

In this work, we show the application of a grammar-based generation scheme to generate candidate computation and communication platform designs. Building upon the idea of procedural generation and exploration of candidate platforms, we propose an evolutionary method to schedule and analyze a communication schedule that considers the interaction of time-triggered traffic and rate-constrained traffic on the platform. We show that, together both methods scale well and can be used in conjunction to analyze the merits of different platform candidates in terms of determinism and latency of time-triggered and rate-constrained traffic flows (respectively).

Future improvements to the work presented in this study could feature the inclusion different priority classes for time-triggered traffic to the algorithm to improve the determinism of higher-priority traffic flows. Another possibility worth consideration is that of utilizing a different link base encoding for the chromosome representation in the scheduling, creating a per-link priority list for the scheduling of flows in the links (as opposed to the current global priority presented on the algorithm of the paper).

6. Acknowledgements

This work was supported by the Sweden's Innovation Agency – Vinnova, as part of the national projects on aeronautics, NFFP7, project CLASSICS (NFFP7-04890). The authors wish to thank the industrial partners from Saab AB for the use case used in the paper.

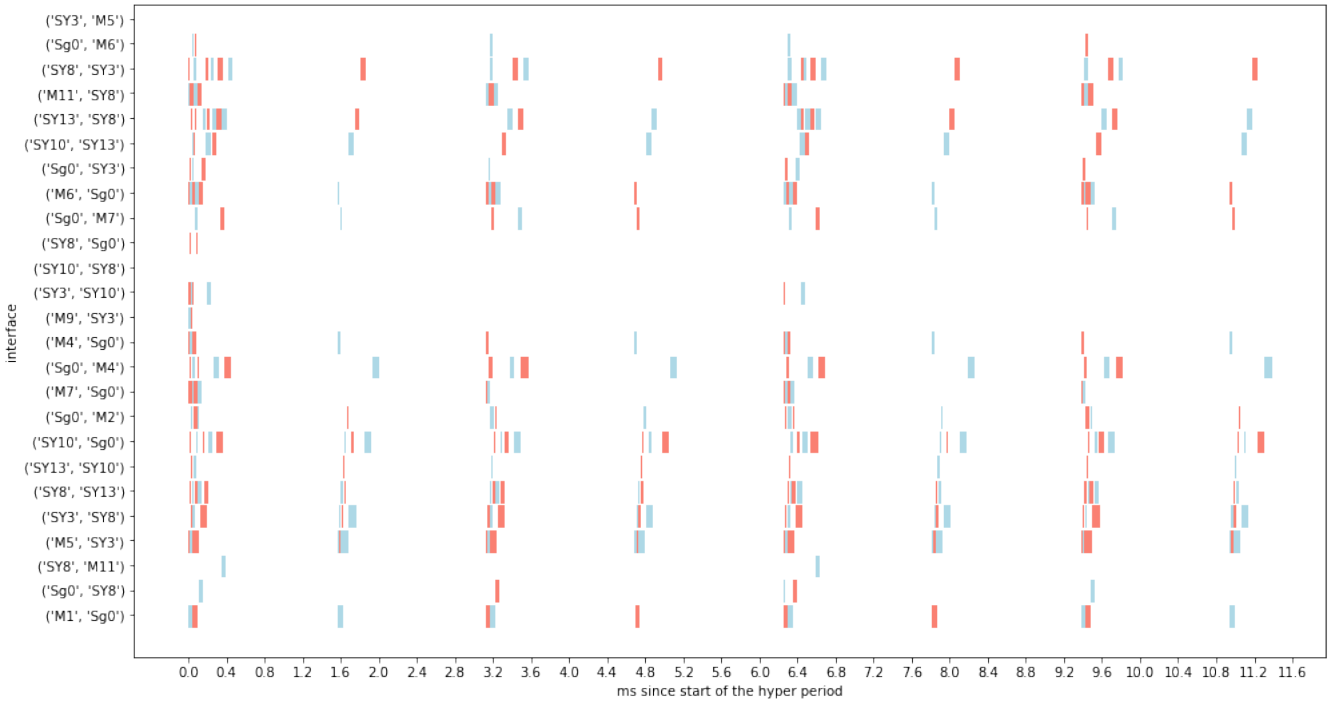

Figure 8 – Gate control list generated by the genetic scheduler for the first 24 interfaces. Colored segments represent an open gate for TT traffic. Two colors are used in order to illustrate the contribution of the individual data packets to gate status.

7. Copyright Statement

References

[1] Otman Abdoun and Jaafar Abouchabaka. A comparative study of adaptive crossover operators for genetic algorithms to resolve the traveling salesman problem. *International Journal of Computer Applications*, 2012.

[2] Otman Abdoun, Jaafar Abouchabaka, and Chakir Tajani. Analyzing the performance of mutation operators to solve the travelling salesman problem. *International Journal of Emerging Sciences*, 2012.

[3] Bjoern Annighoefer, Caroline Reif, and Frank Thieleck. Network topology optimization for distributed integrated modular avionics. In *2014 IEEE/AIAA 33rd Digital Avionics Systems Conference (DASC)*, pages 4A1–1–4A1–12, 2014.

[4] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, pages 216–217, 2008.

[5] Silviu S. Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. Scheduling real-time communication in ieee 802.1qbv time sensitive networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, page 183–192. Association for Computing Machinery, 2016.

[6] F. De La Parra and T. Dean. Survey of graph rewriting applied to model transformations. In *MODELSWARD 2014 - Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development*, pages 431–441, 2014.

[7] Rodrigo Saar de Moraes and Simin Nadjm-Therani. Netgap: A graph-grammar approach for concept design of networked platforms with extra-functional requirements. *under review*, 2022.

[8] B. Dengiz, F. Altiparmak, and A.E. Smith. Local search genetic algorithm for optimal design of reliable networks. *IEEE Transactions on Evolutionary Computation*, 1(3):179–188, 1997.

[9] Christoph Fischer, Maximilian Berndt, Dennis Krummacker, Janis Zemitis, Daniel Fraunholz, and Hans Dieter Schotten. Industrial network topology generation with genetic algorithms. In *2020 International Young Engineers Forum (YEF-ECE)*, pages 25–32, 2020.

[10] Feng He, Zhiyu Wang, and Xiaoyan Gu. Complex real-time network topology generation optimization based on message flow control. In Jing He, Philip S. Yu, Yong Shi, Xingsen Li, Zhijun Xie, Guangyan Huang, Jie Cao, and Fu Xiao, editors, *Data Science*, pages 639–651, Singapore, 2020. Springer Singapore.

[11] Hyeong Jun Kim, Kyoung Chang Lee, and Suk Lee. A genetic algorithm based scheduling method for automotive ethernet. In *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, pages 1–5, 2021.

[12] Maryam Pahlevan and Roman Obermaisser. Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 337–344, 2018.

[13] Lei Zhang, Mattias Lampe, and Zhi Wang. Topology design of industrial ethernet networks using a multi-objective genetic algorithm. In *2011 6th International ICST Conference on Communications and Networking in China (CHINACOM)*, pages 735–741, 2011.