Genetic Algorithm for Scheduling Time-Triggered Traffic in Time-Sensitive Networks

Maryam Pahlevan[1] and Roman Obermaisser[2]
University of Siegen, 57068 Siegen, Germany
Emails: {maryam.pahlevan@uni-siegen.de, roman.obermaisser@uni-siegen.de}

***Abstract*— Time-Sensitive Networking (TSN) is introduced as a series of Ethernet extensions to address strict temporal constraints of modern mission-critical applications. TSN offers determinism using global Time-Triggered (TT) transmission schedules. Most of existing scheduling solutions ignore inter-dependence of routing and scheduling problems and derive the design space of system implementations only from scheduling constraints. This strategy limits the capability of former approaches to compute a global schedule of TT communication for several real-time systems. In this paper, we present a heuristic scheduling approach based on a genetic algorithm. Our approach combines the routing and scheduling constraints and generates static global schedules using joint constraints in a single-step. The number of scheduling possibilities within the design space that is derived from joint routing and scheduling constraints increases in comparison to the approaches that only use the fixed routing. Thereby, the schedulability is improved by our solution. Our genetic-based approach also considers the distribution of real-time applications, multicast patterns and interdependencies of TT flows in the scheduling process. Due to optimized task binding and resource allocation, the experimental results show a significant enhancement of schedulability, TT transmission efficiency and resource utilization compared to the state-of-art solutions.**

I. INTRODUCTION

In modern industrial systems, the increasing number of sensors and computing nodes results in complex network designs and a high volume of data exchanges over the communication links. Although Ethernet fulfills the high demand of bandwidth and the seamless connectivity of vendor-specific devices, it does not offer real-time capabilities which are essential for cyber physical systems. Time Sensitive Networking (TSN) [1] is a set of standards that introduces several Ethernet extensions and a novel scheduling mechanism called Time-Aware Shaper (TAS). TAS uses time as a correctness criterion instead of a metric for the performance measurements [2]. This feature enables TAS to provide determinism, low latency and low jitter for safety-critical applications.

In TSN networks, a fault-tolerant synchronization protocol (i.e. IEEE 802.1ASrev [3]) ensures that all devices are synchronized to a global time and provides the basis for deterministic TT communication. In addition, each port of a device dedicates some queues to the TT flows and the rest is used for the non-TT communication. A TSN capable device transmits messages according to the Gate Control List (GCL). The GCL determines at each time instant which queue is eligible to send out messages. TAS applies the port's GCL with respect to the global time. Therefore, it guarantees that the allocated time slot for a TT flow will not be occupied with any other message (including other TT flows and non-TT messages). The port-specific GCL along with a fault-tolerant clock synchronization enables Ethernet-based networks to meet strict timing constraints of mission-critical applications.

In TSN like in other TT protocols, the global schedule of TT communication is computed off-line due to the complex nature of the scheduling problem. After that the port-specific GCLs are calculated using the global schedule. Despite GCL advantages, the scheduling problem arising from the GCL synthesis is NP-complete. Therefore, it is hard to come up with algorithms that are applicable to different network topologies and scalable to large systems.

To generate a global schedule, both knowledge of the network topology and the TT flow specifications are required. A large number of network devices, switches and links of many Ethernet-based systems results in numerous possible routes and consequently a tremendous number of schedule possibilities for each TT flow. The feasibility of running real-time applications over different end-systems makes the search space of the legitimate schedules even bigger. Therefore, the optimization algorithms for the search space exploration are a key element for the deployment of TSN. Several works discuss the scheduling problem of time-triggered networks by reducing the complexity of the problem using several assumptions. For instance, the majority of TT scheduling solutions calculate the global schedules regardless of routing possibilities. In other words, they ignore the impact of routing on the scheduling constraints and use fixed routes which are generated separately as an input to the schedulers. This simplified abstraction of the scheduling problem (e.g. using fixed routing) may lead to the failure of the schedule generation, although the system is schedulable [4]. A few recent works [4], [5] focus on joint routing and scheduling for the global schedule computation. They all use the ILP-based approach which is rather slow and not scalable to solve large scheduling problems. In addition, these solutions do not consider multicast TT messages, inter-flow dependencies and distributed real-time applications.

In this paper, we present two different scheduling algorithms: the first one is based on genetic algorithms, while the second one is based on list scheduling and only a reference for comparison with the first solution. In contrast to state-

978-1-5386-7108-5/18/$31.00 ©2018 IEEE

of-art scheduling solutions, our Genetic Algorithm (GA) generates port-specific GCL imposing routing and scheduling constraints at the same time. This solution transforms two separate sets of routing and scheduling constraints into one set of constraints and solves the scheduling problem considering multicast communication and interdependencies of TT flows in a single-step. The main goal of GA is to satisfy the deadlines, while optimizing the TT transmission makespan and the overhead of TT communication. To achieve this goal, GA minimizes the gap between TT time slots and consequently reduces the number of guard bands that are introduced before every TT time slot to avoid interference with other flows. In addition, our solution provides an opportunity to distribute safety-critical applications over available end-systems rather than placing them on a certain device. This approach is beneficial for mission-critical applications (e.g. driving assistance) that require considerable amounts of computational power.

In addition to the GA, we develop a heuristic list scheduler which first finds the shortest paths between all end-systems and then uses these fixed routes to solve the scheduling problem. In the experiment section, we apply the proposed solutions to different sets of TT flows and network designs. We also compare the simulation results of the single-step GA with our two-phase list scheduler. The results demonstrate that GA improves the scheduling capability and transmission efficiency of TT communication over our list scheduler considerably.

The remainder of the paper is structured as follows: In Section II, related work is discussed. Section III introduces the system model used in this work. Section IV formulates the scheduling problem of TT communication. The joint routing and scheduling constraints are defined in Section V. Section VI describes our GA while Section VII details the list scheduler, which serves as a reference implementation of existing scheduling solutions and also as a baseline for the experimental analysis. In the following section, experimental results are evaluated. The last section concludes the paper.

II. Related Work

Since time-triggered systems necessitate a global schedule, in recent years several works addressed the scheduling problem of TAS which is also introduced in the IEEE 802.1Qbv [6] extension. In [7], the authors first determine scheduling constraints of TAS in multi-hop switched networks and then compute the valid schedule using Satisfiability Modulo Theories (SMT) and Optimisation Modulo Theories (OMT) solvers. They also verify that GCL offers deterministic delivery of TT messages. Pop et al. synthesize the GCL for each TSN capable device's port using ILP. This work computes the global schedule while it allocates resources to TT flows optimally. The authors in [8] developed a hybrid genetic algorithm to generate the static schedule table of TT frames in Time-Triggered Ethernet (TTE) networks. This work optimizes the number of allocated TT time slots and consequently improves the transmission efficiency of TTE communication. All aforementioned solutions, first calculate the valid routes of TT frames and then use the fixed routing information for computing the global schedule. As a result, they neglect the vital role of routing in the scheduling process.

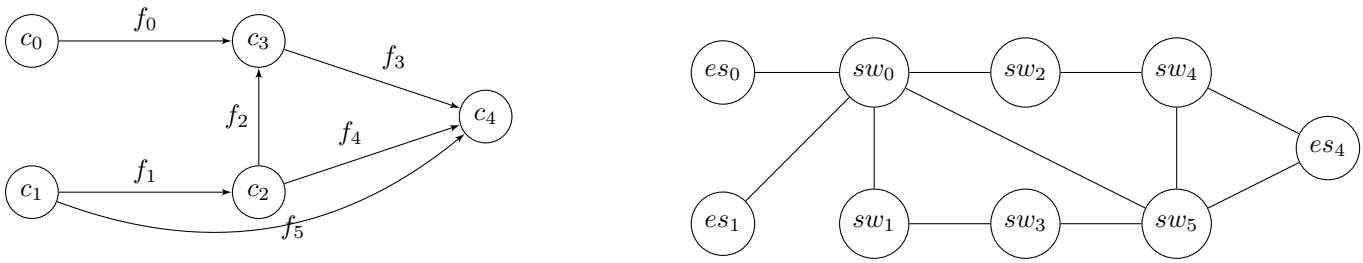
Smirnov et al. propose a set of Pseudo-Boolean (PB) constraints to solve routing and scheduling problems of TT communication in a single-step. Furthermore, this work employs multi-objective optimization to the design space deriving from joint routing and scheduling constraints. This implementation first uses a time-consuming ILP approach to define the design space from PB constraints. Then, it applies the NSGA-II optimization algorithm to the exploration model. This solution does not support application-specific periods for different TT flows which is essential for TSN deployment. Instead it is assumed that all scheduled traffic is sent in the same cycle. In addition to this study, the authors in [5] develop the ILP-based scheduling solution for the joint routing and scheduling problem and evaluate the experimental results of different traffic patterns and network topologies using two performance metrics (i.e. end-to-end delay and scheduling capability). Due to the ILP-based scheduling process, the aforementioned solutions are rather time-consuming specifically for large-scale time-triggered networks. They also do not consider multicast and inter-flow dependencies in their experiments.

We develop a fast GA to address the interdependence of routing and scheduling constraints. This approach computes the global schedule within reasonable time intervals and it is scalable to modern industrial networks with an ever increasing number of elements. Our solution also makes the distribution of real-time applications feasible using valid task bindings and resource allocations. In addition to the aforementioned advantages, the GA supports multicast transmission patterns and inter-flow dependencies which to the best of our knowledge are not integrated in any TSN scheduling solution yet.

III. System Model

In this paper, we model the network topology and TT flows with two separate graphs: An architecture graph and an application graph. An architecture graph is presented by an undirected graph G_A (R, E_l). This graph comprises end-systems and TSN switches $R = ES \bigcup S$ as vertexes and the duplex links between them as edges. An application graph is shown by a directed acyclic graph G_C (C, E_f). G_C consists of computational tasks as vertexes and the TT flows transmitted between the tasks as edges. Since each task can have multiple successor tasks, this system modeling supports multicast TT flows as well [9]. Our scheduling algorithms use the described graphs as inputs. Figure 1 presents an example of the system model.

The scheduling possibilities of TT communication are derived from mapping the application graph to the architecture graph. To compute a valid system implementation, first each computational task is assigned to a certain end-system. As a next step, the TT messages which are transmitted between a computational task and the predecessor tasks are mapped to



(a) Application graph

(b) Architecture graph

Fig. 1: An example of system model

the physical links that connect the sender end-system to one or multiple receiver end-systems.

In this model, computational tasks are intended to generate TT messages. Therefore, they can only run on end-systems whereas TSN switches relay TT messages. It is noteworthy that the physical links $e \in E_{\text{l}}$ are bidirectional. Therefore, if a TT frame is traversing a specific link in one direction, simultaneously another TT message can be transmitted over the same link in reverse direction. In our system model, we assume that all TSN switches and end-systems are synchronized to the global time.

IV. Problem Formulation

In TSN networks, switches and end-systems exchange three types of traffic: TT flows, AVB streams and Best Effort (BE) messages. Our scheduling algorithms are developed to generate a valid GCL so that the AVB streams and BE traffic which do not have strict timing requirements, do not interfere with the transmission of TT frames. Consequently our algorithms only compute the static schedule table of TT messages and non-TT frames (including AVB and BE) are sent when no TT message is scheduled.

Each TT flow $f \in E_{\text{f}}$ is identified by $f.sender \in C$, $f.receiver \in C$, $f.size$, $f.period$ and $f.deadline$. Since in TSN a TT flow can consist of more than one frame, $f.size$ is equal to the number of TT frames which are sent consecutively in one $f.period$ multiplied by the frame's length. As TT frames are sent periodically, the f.period attribute is used to specify the periodicity of a TT flow. The f.deadline field determines the maximum permissible end-to-end latency. For simplicity purposes, we assume a TT flow remains in a TSN capable device just for the processing time which is computed as follows:

$$t_{\text{f.processing}} = ProcessingRate_{\text{device}} \times f.size$$

The GCL of each port of a TSN capable device is determined by our scheduling algorithm and it reflects the injection time of TT flows routed via that device. The $f.InjecTime$ determines when the sender end-system starts transmitting the TT flow just after execution of corresponding computational task ($c.ExTime$). In order to offer deterministic TT communication in synchronized and scheduled networks, all port-specific GCLs begin simultaneously and repeat over the Least Common Multiple (LCM) of all $f.period$ values called hyper-period.

V. Scheduling and routing constraints

We combine the scheduling constraints definition in [7], [10] with the routing constraints as follows:

1) Each computational task is assigned to exactly one end-system. The end-system where the task can run on, is chosen from the eligible end-systems $c.CanRunOn$ for that specific task. The network designers provide this information within the application graph using the knowledge of application requirements and end-system capabilities.

$$\forall c \in C, es \in c.CanRunOn : c.processor = es$$

2) To eliminate loops, each frame of a TT flow can pass through a certain node at most once. The f.route consists of all adjacent links which form the path from the sender to the receiver. It is noteworthy that f.route is set to one of routing possibilities between the sender and the receiver.

$$l_i = (u, v) \in E_{\text{l}} : R = \{(l_i, .., l_{i+n}), ..., (l_j, .., l_{j+m})\}$$

$$r \in R : f.route = r$$

3) Each TT flow can be routed through a specific link, if it can have an exclusive access to the physical link for the duration of $f.TransDelay$ just after the transmission starts. The transmission delay of a TT flow on a certain link is calculated as follows:

$$l_i \in E_{\text{l}} : f_{\text{l}i}.TransDelay = \frac{f.size}{\text{l}_i.bandwidth}$$

It is important to note that we assume a TSN capable device (including end-systems and switches) dedicates only one queue per port to TT traffic. Hence, to eliminate interleaving of different TT flows in a single TT queue, the device follows the flow isolation constraint introduced in [7]. We reflect this constraint by considering an exclusive access to the egress port and the attached link for a period of $t_{\text{f.processing}} + f_{\text{l}i}.TransDelay$.

This constraint is applied to all adjacent links in $f.route$. For each link, the time interval of exclusive access is calculated with respect to the period of $t_{\text{f.processing}} + f_{\text{l}i}.TransDelay$ on previous adjacent link within $f.route$. For minimizing the makespan of TT applications, we do not permit any gap between the

value of $t_{\text{f.processing}} + f_{\text{l}}.TransDelay$ on two subsequent links of $f.route$. This means the buffering of TT frames is not allowed in our system model and devices follow the store and forward approach for switching TT packets.
$\forall(l_i, l_{\text{i+1}}) \in f.route$:

$$f_{\text{l}_{\text{i+1}}}.InjecTime = f_{\text{l}_{\text{i}}}.InjecTime + t_{\text{f.processing}} +$$

$$f_{\text{l}_{\text{i}}}.TransDelay$$

4) In our system model, TT flows are not restricted to one period and can be transmitted over different cycles. For this reason, the time interval of exclusive access on every link of $f.route$ is calculated considering the periodic accesses of other TT flows which traverse the same physical links throughout their paths from the senders to the receivers.
5) Each computational task can start only when the TT flows that are sent by the predecessor tasks towards this task are delivered. In other words, the task can start transmitting TT messages only when the computing job is executed and all predecessor TT flows are received. The flow's end to end delay determines the time interval between the injection time of flow and its arrival time at certain destination.
$\forall f \in E_{\text{f}}, \forall \overline{f} \in pre(f), c \in C, f.sender = c$:

$$\overline{f}.e2eDelay = \sum_{l \in \overline{f}.route} t_{\overline{f}.\text{processing}} + \overline{f}.TransDelay$$

$$\overline{f}.InjecTime + \overline{f}.e2eDelay + c.ExTime$$

$$\leq f.InjectTime$$

This constraint reflects inter-flow dependencies and provides an opportunity to transmit TT flows based on predefined priorities.
6) Each TT flow that is sent by a computational task must be delivered to the successor task within the flow's deadline.
$\forall f \in E_{\text{f}}$:

$$f.InjecTime + f.e2eDelay \leq f.deadline$$

VI. GA IMPLEMENTATION

The scheduling problem of time-triggered networks can be solved by combining bin-packing and a genetic algorithm [8]. We introduce a Genetic Algorithm (GA) to generate the valid schedule with optimized transmission time of TT messages. To be more specific, the main goal of our GA is to minimize the makespan of TT communication by optimizing the end-to-end delay as a measurement metric.

$$\min(\max_{\forall f \in E_{\text{f}}}(f.InjecTime + f.e2eDelay))$$

A. Individual definition

In GA, a genome builds an individual. Each genome contains an array of genes. For resource allocation and task binding, GA needs one gene per task. Each task specific gene contains all end-system IDs that the task can run on (i.e. introduced in c.CanRunOn). In addition, each TT flow is mapped to one gene. The flow specific gene includes the flow's routing possibility indexes. Each gene is encoded by sets of integer numbers.

B. Population initialization

The GA was implemented using GAlib [11] that provides different genetic algorithms in C++. The GA first initializes an individual using information derived from the system model (including G_A and G_C). Then, it generates an initial population. In each generation, the GA chooses the individuals with the best fitness and creates a new population of individuals using the simple-point crossover operator. As a result, the best individuals are preserved for the next generation.

C. Fitness function

The fitness function assigns a fitness score to each individual. In the GA, the fitness function first computes the global schedule of each individual and returns the makespan as a fitness value. After that, the GA evaluates the eligibility of individuals based on their fitness scores and selects the ones with the best fitness for creating the next generation.

Algorithm 1 Fitness Function

```
1: procedure FITNESS(Genome g)
2: makespan ← 0
3: E_f.sorted ← sort flows based on interdependencies
4: ∀f ∈ E_f.sorted:
5:     ST ← f.sender
6:     RT ← f.receiver
7:     ST.processor ← p ∈ ST.CanRunOn task's genes
8:     RT.processor ← p ∈ RT.CanRunOn task's genes
9:     f.route ← r ∈ R using flow's genes
10:    f.InjecTime ← find earliest feasible time slot
11:    f.arrival ← f.InjecTime + f.e2eDelay
12:    if f.arrival > f.deadline then return infinity
13:    RT.startTime ← max(RT.startTime, f.arrival)
14:    RT.finishTime ← RT.startTime + RT.ExTime
15:    makespan ← max(makespan,RT.finishTime)
16: return makespan
```

Algorithm 1 presents the GA's fitness function in more details. In this function, first we sort the TT flows based on their interdependencies. For each TT flow in $E_{\text{f.sorted}}$, the sender and receiver tasks are assigned to the available end systems using task-specific genes in a genome. The f.route is also selected from the possible routes between sender and receiver using the flow's genes. For finding all possible routes we use the multiplication of adjacency matrix approach.

After initialization, the function using constraints (3) and (4) finds the earliest time instant that the sender task can access all adjacent links in the f.route exclusively. If the flow's injection time violates the constraint (6), it means that the individual leads to an infeasible system-wide schedule. Therefore, the function returns infinite as a fitness score to eliminate inheritance of infeasible individuals to the next generation. The line 13 corresponds to constraint (5) and updates the start time of the receiver task accordingly. In the last line, the function returns the makespan as a fitness score. The makespan corresponds to the time instant that all computational tasks are finished. In each generation, the best solutions (individuals with minimum makespan) are stored and the new population of individuals in the next generation is compared to the current best candidate. If the individual's makespan is bigger than the current minimum makespan, the individual will not be carried over to the next generation. Consequently, the GA converges faster to a feasible global schedule.

The GA's objective is to find the global schedule with the minimum makespan. This optimization process has the following advantages: 1) The scheduling capability will be improved, since the transmission time of TT flows is optimized. 2) The optimized makespan leads to more compact transmission schedules of TT flows. Therefore, the number of guard bands that is reserved before each TT time slot to avoid interference with non-TT traffic reduces significantly. 3) This also results in a better bandwidth utilization and smaller waiting time of non-TT messages which are blocked due to exclusive TT time slots.

VII. LIST SCHEDULER

We also developed a scheduling algorithm based on list scheduling [10]. Since our list scheduler (LS) solves the scheduling problem of TT communication using fixed routings, we use LS to evaluate GA in terms of schedulability.

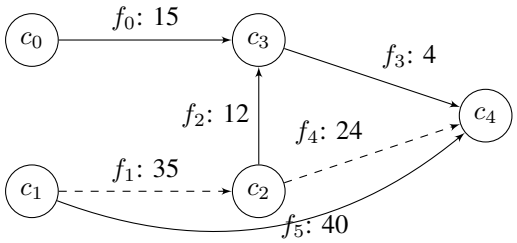


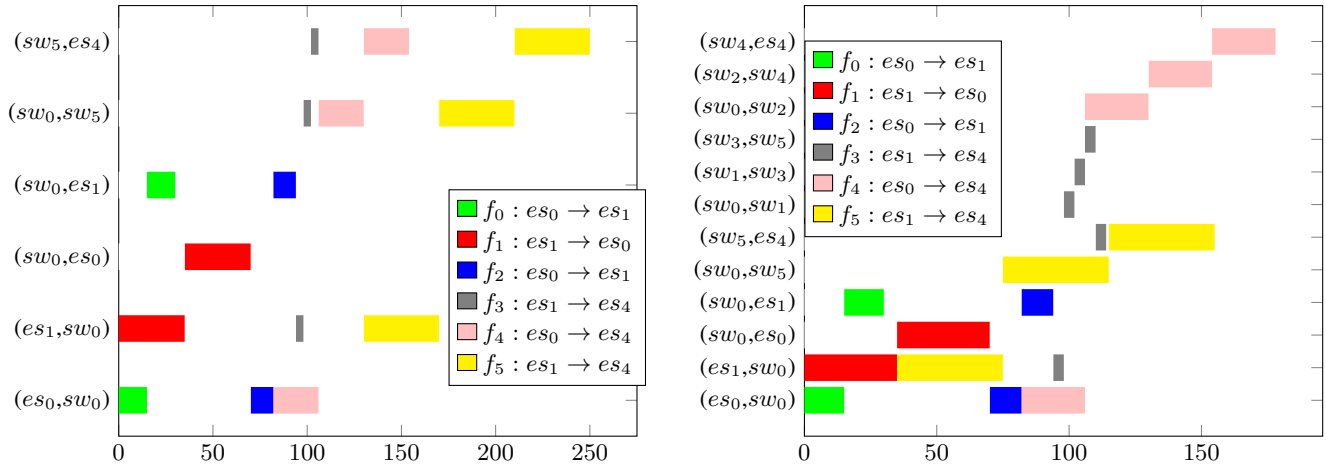
Fig. 2: An application graph, weight on each edge is $Cost_f$

As shown in Algorithm 2, LS first calculates the priority of each task using the critical path. The task's critical path defines the longest path from a predecessor task to the task according to the communication cost (i.e. $Cost_f = t_{\text{f.processing}} + f_{\text{l}_\text{i}}.TransDelay$). For example, in Figure 2 the critical path of task c_4 is shown in dashed-lines and the priority is set to 59 (i.e. $Cost_{f_1} + Cost_{f_4}$). After that, LS sorts computational tasks based on their priorities. In line 5, LS schedules tasks from highest priority to the lowest one. For each task, first the task's incoming TT flows are retrieved. If the task does not have any incoming flow or all predecessor tasks are already scheduled, an available end-system from $c.CanRunOn$ is allocated to the task. Then, for each incoming TT flow, the algorithm finds the earliest injection time using the shortest path between the sender and the receiver and updates the schedule's makespan accordingly. If the flow's injection time violates constraint (6), another end-system is assigned to the task and the same procedure is repeated. The reason is that the previous chosen end-system which violates constraint (6) leads to an infeasible solution. If the task needs to receive TT flows from other computational tasks before it can start transmitting TT messages (as formulated in constraint 5), LS tries to schedule all preceding tasks first as is shown in line 10. For instance, LS specifies the injection time of f_0, f_1 and f_2 before it allocates a time slot to f_3.

Algorithm 2 List Scheduler

```
1:  procedure ListScheduler
2:  makespan ← 0
3:  assign priority to each computational task
4:  C_c.sorted ← sort tasks descendingly based on priorities
5:  ∀c ∈ C_c.sorted is not scheduled:
6:      makespan ← Scheduler(c)
7:  return makespan
8:  procedure Scheduler(Task c)
9:      if task c is not scheduled and has incoming TT flows then
10:         ∀f ∈ E_f.incoming: Scheduler(f.sender)
11:         is_pred_tasks_scheduled ← true
12:     else if is_pred_tasks_scheduled or task c has no child then
13:         for p ∈ c.CanRunOn do
14:             for f ∈ E_f.incoming do
15:                 c.processor ← p
16:                 f.route ← ShortestPath(sender, receiver)
17:                 f.InjectTime ← FindEarliestTime
18:                 f.arrival ← f.InjectTime + f.e2eDelay
19:                 if f.arrival > f.deadline then:
20:                     go to the next end-system
21:                 c.StartTime ← max(c.StartTime, f.arrival)
22:     makespan ← max(makespan, c.StartTime + c.ExTime)
23: return makespan
```

To illustrate the difference between GA and LS, we use the system model in Figure 1. The flow's communication cost and periods are given in Figure 2 and Table I respectively. Table I also shows that LS always uses the shortest paths, while GA finds the routing that leads to a more optimal makespan. The Gantt charts in Figure 3 present the global schedules that were computed by LS and GA. In the Gantt charts, each box presents the time slot that is dedicated to a



(a) LS TT transmission schedule

(b) GA TT transmission schedule

Fig. 3: Transmission schedule of LS and GA

certain TT flow on a specific link. As the graphs show, in GA the makespan of TT flows is improved compared to LS (from 250 μs to 178 μs). The reason is that the fixed routes used in LS cause high traffic load on certain physical links, although other links are under low utilization. In contrast, GA benefits from the load balancing while computing routes using joint scheduling and routing constraints. The enhancement of makespan is important because the TT transmission schedule plays a key role in the complex time-triggered systems that comprise several mission-critical applications with short deadlines.

	period/ deadline (μs)	route LS	route GA
f_0	500 / 100	es_0, sw_0, es_1	es_0, sw_0, es_1
f_1	400 / 350	es_1, sw_0, es_0	es_1, sw_0, es_0
f_2	400 / 250	es_0, sw_0, es_1	es_0, sw_0, es_1
f_3	500 / 380	es_1, sw_0, sw_5, es_4	$es_1, sw_0, sw_1, sw_3, sw_5, es_4$
f_4	500 / 250	es_0, sw_0, sw_5, es_4	$es_0, sw_0, sw_2, sw_4, es_4$
f_5	1000 / 350	es_1, sw_0, sw_5, es_4	es_1, sw_0, sw_5, es_4

TABLE I: TT flow parameters

VIII. EXPERIMENTS AND EVALUATION

A. Experimental Setup

GA and LS both are implemented in C++ and run on a T460 ThinkPad computer with 2.4GHz Intel i5 CPU and 32GB of memory.

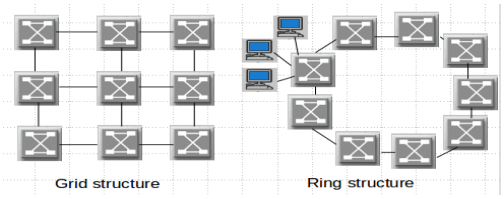


Fig. 4: Topologies used in our experiments. Every switch is connected to either 3, 4 or 5 hosts in a star structure.

We generate several system models (including architecture and application graphs) using the SNAP library [12]. These system models are given as inputs to GA and LS schedulers. We conduct our experiments on two different network topologies: 1) meshed grid and 2) ring. As depicted in Figure 4, each network comprises 9 switches and every switch is attached to 3 to 5 end systems. We consider the ring topology to reflect a common structure of industrial control networks. To evaluate our joint routing and scheduling constraints, we use the meshed grid structure. This topology has a higher connectivity and provides more routing possibilities for every TT message. It is also assumed that all physical links in our experimental networks have a bandwidth of 1 Gbps and all switches require 2 nanosecond for processing each byte.

The flow interdependencies are formulated using a random Forest Fire directed graph [12]. In our synthetic application graph which includes 15 tasks, we use 4 different traffic classes. The characteristics of each traffic class are detailed in Table II. It is noteworthy that the list of eligible end-systems which every task can run on (i.e. $c.CanRunOn$) is chosen randomly. Furthermore, each TT flow's deadline is selected randomly from a range of 200 to 800 microseconds. It is also assumed that all end systems are identical and all computational tasks have the same execution time (i.e. 4 microseconds).

traffic class	$f.size$ (bytes)	$f.period$ (μs)
$class_1$	200	100
$class_2$	400	200
$class_3$	600	300
$class_4$	800	400

TABLE II: Traffic classes parameters

GA initializes the genetic algorithm parameters as follows: population size = 100, the number of generations = 100, the mutation probability = 0.2, the crossover probability = 0.9 and convergence probability = 1.

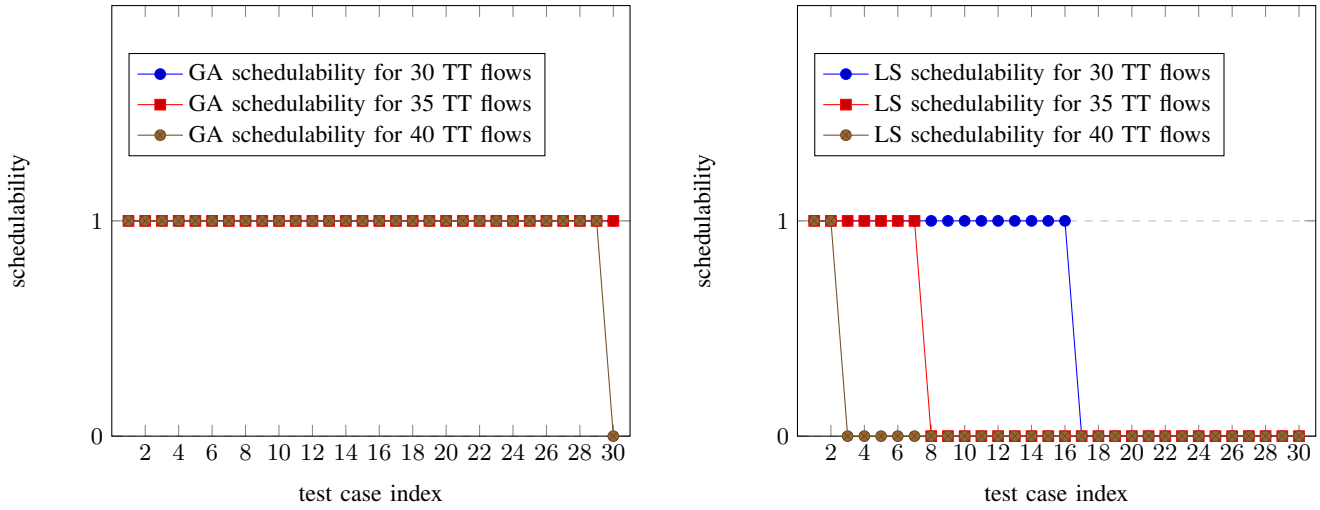


Fig. 5: Schedulability of LS and GA with varying TT loads and the meshed grid topology

B. Experiments and Evaluation

In this section, we evaluate the scheduling capability and efficiency of GA using the experimental results of LS which serve as a baseline for the state-of-art two-phase scheduling solutions.

The first part of the experiments is intended to study the impact of varying load on schedulability and transmission makespan of LS and GA schedulers. For this purpose, we consider the system models with the meshed grid structure (Figure 4) and three different TT traffic loads (i.e. 30, 35 and 40 TT flows). For every traffic load, we generate 30 different flow interdependency patterns although the network topology remains the same.

TT flows	LS Avg makespan (μs)	GA Avg makespan (μs)	Improved ratio
30	168.62	100.25	0.4
35	179.42	121.14	0.32
40	180	142	0.21

TABLE III: Transmission makespans for the meshed grid topology and varying load.

Table III lists the required time for delivering all TT flows to corresponding destinations. The improved ratio of makespan in this table is calculated as follows:

$$ImpRatio = \frac{LS\ average\ makespan - GA\ average\ makespan}{LS\ average\ makespan}$$

GA compared to LS improves TT transmission efficiency on average by 0.31. The enhanced scheduling efficiency of GA implies shorter end-to-end delays, more compact TT global schedules and better utilization of link bandwidth. To achieve the optimal makespan, GA distributes the computational tasks over available end-systems and balances the load over different physical links. GA also schedules different TT flows on a certain link with the minimum gap between their time slots. Furthermore, joining time slots of consecutive TT flows on a specific link leads to the minimum number of guard bands and better resource utilization (e.g. link bandwidth).

TT flows	LS Avg Exec Time (s)	GA Avg Exec Time (s)
30	0.01	52.54
35	0.013	54.61
40	0.014	56.75

TABLE IV: Execution time for the meshed grid topology and varying load.

Table VI presents the average execution time of GA and LS for each traffic load. As simulation results show, LS solves the scheduling problem faster than GA, since LS uses the fixed routing and ignores the key role of routing in the scheduling process. In other words, the fixed routing between each sender and receiver limits the search space and reduces the solving time of LS significantly. In contrast, GA considers all routing possibilities and employs the joint routing and scheduling constraints. Therefore, the design space of system implementations gets bigger and GA requires more time for the global schedule generation.

In addition as results depict, the average execution time of GA and LS increases when the number of flows increases. Because by increasing network load, our schedulers take more time to find a task binding and resource allocation that leads to a valid schedule.

The graphs in Figure 5 depict the schedulability of GA and LS for the above benchmarks. The schedulability ratio of GA for test cases with varying loads is on average 0.98. On contrary, LS scheduling ratio for same test cases is on average 0.27. As the graphs illustrate, the scheduling capability of LS compared to GA decreases significantly when the network utilization (i.e. number of TT messages in our experiments) increases. LS like other state-of-the-art scheduling solutions solves the scheduling and routing problems separately. Consequently, the increasing number of

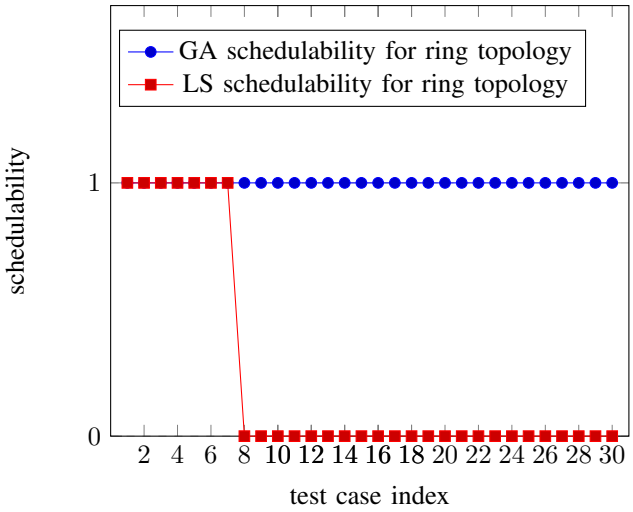


Fig. 6: Schedulability of LS and GA with different topologies

TT flows in LS scheduler may lead to over-utilized links and the violation of constraint 6. GA resolves this issue by transmitting TT flows over different routes and avoiding bottlenecks in routing TT frames. In other words, GA excels LS by examining different routing possibilities during the scheduling and optimization process.

To evaluate the effects of the network topology on GA performance metrics, we repeat the test cases with 30 flows in the first part using the ring structure (Figure 4). To be more specific, the flow interdependency patterns in this set of benchmarks stay unchanged although the switches are connected to each other in a ring topology rather than a meshed grid structure.

According to experimental results which is presented in Figure 6, when the ring topology is used, the LS capability to find a valid schedule declines significantly compared to the GA scheduler. It is important to note the similar number of flows in the ring structure leads to a higher network utilization. Therefore, LS fails to meet timing requirements of more test cases due to the violation of constraint 6. In contrast, GA keeps the schedulability ratio of one in all test cases regardless of a chosen network topology. Since GA overcomes the limitation of LS regarding over-utilized links by balancing loads over different routes. In GA, this strategy becomes feasible by examining a higher number of scheduling possibilities that is derived from the joint scheduling and routing constraints.

IX. CONCLUSIONS

In this paper, we present a genetic scheduling approach for generating global schedules of TT communication. In contrast to state-of-art scheduling solutions with fixed routing, the GA combines the routing and scheduling constraints and computes a system-wide schedule in a single-step. To make task binding and resource allocation feasible, we design a system model in the form of an application graph and an architecture graph. Furthermore, in this paper we define joint scheduling and routing constraints and employ them in the scheduling and optimization process of GA. This novel scheduling approach also provides an opportunity to distribute processing of real-time applications over different end-systems using optimized task bindings.

To have a solid base for comparison, we developed a list scheduler which is a typical example of state-of-art scheduling procedures and solves the routing and scheduling problems separately. The experimental results illustrate the impact of load and network topology on GA and LS performance indicators (i.e scheduling capability and efficiency). In GA, we extend the search space of scheduling possibilities using the joint constraints. This strategy, specifically in presence of high network utilization, enhances the scheduling capability of GA over LS significantly. In the experiments we observed that GA improves the transmission makespan on average by 31 % in comparison to LS. The optimal makespan implies more compressed TT transmission schedule and less number of guard bands.

ACKNOWLEDGMENT

This work was sponsored by Safe4RAIL Project, Grant Agreement No. 730830.

REFERENCES

[1] "Institute of electrical and electronics engineers, time-sensitive networking," in *Time-Sensitive Networking Task Group. http://www.ieee802.org/1/pages/tsn.html*, IEEE, 2017.

[2] M. A. Weiss et al., "Time-aware applications, computers, and communication systems (taaccs)," in *Technical Note (NIST TN)-1867*, 2015.

[3] "Institute of electrical and electronics engineers, inc. 802.1as-rev - timing and synchronization for time-sensitive applications," in *Time-Sensitive Networking Task Group. http://www.ieee802.org/1/pages/802.1AS-rev.html*, IEEE, 2017.

[4] F. Smirnov, M. Glaß, F. Reimann, and J. Teich, "Optimizing message routing and scheduling in automotive mixed-criticality time-triggered networks," in *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*, pp. 1–6, IEEE, 2017.

[5] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, "Ilp-based joint routing and scheduling for time-triggered networks," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, pp. 8–17, ACM, 2017.

[6] "Institute of electrical and electronics engineers, inc. 802.1qbv - enhancements for scheduled traffic," in *Time-Sensitive Networking Task Group. http://www.ieee802.org/1/pages/802.1bv.html*, IEEE, 2016.

[7] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in ieee 802.1 qbv time sensitive networks," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pp. 183–192, ACM, 2016.

[8] L. Bingqian and W. Yong, "Hybrid-ga based static schedule generation for time-triggered ethernet," in *Communication Software and Networks (ICCSN), 2016 8th IEEE International Conference on*, pp. 423–427, IEEE, 2016.

[9] M. Lukasiewycz, M. Streubühr, M. Glaß, C. Haubelt, and J. Teich, "Combined system synthesis and communication architecture exploration for mpsocs," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 472–477, European Design and Automation Association, 2009.

[10] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner, "Design optimisation of cyber-physical distributed systems using ieee time-sensitive networks," *IET Cyber-Physical Systems: Theory & Applications*, vol. 1, no. 1, pp. 86–94, 2016.

[11] "Galib documentation," in *http://lancet.mit.edu/galib-2.4/*, 2017.

[12] "Snap library 4.0, user reference documentation," in *https://snap.stanford.edu/snap/doc/snapuser-ref/index.html*, 2017.