Integrated Cost Optimization and Preemptable Scheduling for Real-Time Ethernet Applications

Ayla Babazade, Soheil Samii, Ahmed Rezine
Dept. Computer and Information Science
Linköping University
Sweden
{ayla.babazade, soheil.samii, ahmed.rezine}@liu.se

***Abstract*—Ethernet communication technology is growing in deployments across many real-time embedded systems. Through the IEEE 802.1 standardization group, Ethernet has been enhanced with several protocols and traffic scheduling policies, commonly known as Time-Sensitive Networking (TSN). Time-triggered communication is a common approach in TSN, but unfortunately not able to meet timing constraints for applications with very stringent real-time requirements. Past work has addressed this issue by combining time-aware scheduling with frame preemption, another key TSN standard. While frame preemption enhances schedulability, it comes with additional hardware cost for endpoints and switches. To reconcile the requirements of low-cost, real-time applications, this paper addresses the joint problem of minimizing hardware preemption cost and scheduling real-time communications across packet-switched Ethernet networks. We propose a monolithic SMT solution as well as efficient heuristic approaches to minimize cost and synthesize preemptable time-triggered network schedules. The proposed approaches have been evaluated through extensive experiments.**

***Index Terms*—time-sensitive networks, frame preemption, scheduling, switch selection, Ethernet**

I. Introduction

Many real-time embedded and cyber-physical systems are designed with onboard real-time networks that interconnects various sensors, actuators, and computing nodes that all need to communicate to realize a set of features and functions. Several real-time and safety-critical application domains are increasingly challenged with the need to expand the communication bandwidth. Prominent practical examples can be found in industrial automation, avionics, and automotive electronics. In the automotive industry, for example, new advanced driver assistance and automated driving applications are driving the expansion of communication bandwidth requirements by several orders of magnitude compared to traditional in-vehicle communication networks. As such, Ethernet communication technology is being increasingly adopted in many real-time application areas.

To meet new real-time and dependability requirements, the IEEE 802.1 Time-Sensitive Networking (TSN) Task Group has developed a set of standards to enhance switched Ethernet with real-time and dependability capabilities. These standards include time synchronization protocols (802.1AS-2011) [1], enhancements for scheduled time-triggered traffic (802.1Qbv-2015) [2], credit-based shaping (IEEE Std 802.1Qav-2009), and frame preemption (802.1Qbu-2016 and 802.3br-2016) [3], to name a few examples.

Time-triggered Ethernet communication is a popular approach for many real-time applications and has been studied and deployed widely [4]. Time-triggered and time-aware scheduling has been studied and adopted widely in safety-critical real-time systems, both at processor [5] and network levels [6], [7], [8], [9], due to its high predictability capabilities [10], [11]. Despite this, there are applications with strict real-time constraints that, for a given network architecture, are not schedulable with traditional time-triggered, packet-switched scheduling. To address these scenarios, system designers need to look for changes to the physical network topology and upgrades to the speed of certain communication links. While such approaches enable a schedule to be defined, meeting the timing constraints of the application, these come at a high and often unacceptable cost. Changing the network topology in a vehicle platform, for example, implies a change of the underlying wiring harness—which adds cost and manufacturing complexity, and sometimes is infeasible due to practical and commercial constraints. Upgrading link speeds involve changes of the physical layer, which in turn involves changes of integrated circuits, board redesign, and connector changes—all of this adding significant cost and complexity.

An alternative, more cost-efficient approach, is to leverage the frame preemption capability in TSN. This enables the introduction of frame preemption in the overall time-triggered network schedule, which in turn creates opportunities to meet deadline constraints that are not possible to meet in traditional nonpreemptive communication scheduling. While frame preemption is a more cost effective solution, it does not come at zero cost difference as the MAC layer of a preemption-aware network port has unique logic compared to a traditional network port. Moreover, network switches that support preemption are more expensive than traditional Ethernet switches. Thus, adding preemption capability to a given network must be done in a selective manner to be able to meet timing constraints while minimizing the added cost of preemption support. Zhou et. al. has developed an SMT-based approach for preemptable TSN scheduling [12]. Stuber et. al. propose a survey of Time-Aware Shaper (TAS) scheduling algorithms for TSN [13]. Recently, Xue et. al. performed a systematic comparison of different TSN scheduling approaches,

2325-1301/25/$31.00 ©2025 IEEE
DOI 10.1109/RTCSA66114.2025.00025

possibly with frame preemption [14]. However, the problem of determining which ports in the network should be preemptable remains open.

Contribution: This paper addresses the problem of joint network scheduling and network cost optimization. We consider a practical setting in which the selection of preemptable ports is constrained through an actual set of switches, each with a certain number of preemptable ports and associated cost. This cosntraint stems from that, in a real-life setting, a system designer can select from a given product portfolio of network switches from a number of vendors. We propose three approaches to minimize the cost of overall network topology with frame preemption. These approaches ensure that we use frame preemption on links between switches (two connected ports of two connected switches) only in situations where Time-triggered Scheduling is not sufficient. As demonstrated in this paper, we can choose more affordable switches for the topology, while meeting all of the scheduling deadlines. We propose three approaches: 1) SMT-based Monolithic approach is effective and efficient in switch allocation and scheduling TSN network with time-triggered communication and frame preemption, 2) SMT-based Monolithic approach with Time slicing to insure scaling, and 3) Heuristic approach which besides being effective and efficient is more scalable than monolithic and Time sliced approach. Our solutions provide complete switch allocation and port-to-port allocation for given TSN network topology.

II. System and Network Model

A network topology is a connection of nodes and edges. A node is either a network switch or an endpoint. Nodes are connected to each other via edges. An edge is a full-duplex link that allows simultaneous communication between two nodes in both directions. To connect two nodes, a link is established between one port of one node and one port of another node.

For ports with frame preemption support, the egress MAC (Media Access Control) interface is divided into two categories as depicted in Figure 1: the eMAC category for express traffic and the pMAC category for preemptable traffic. The eMAC category is used for messages that are able to preempt an ongoing transmission (the messages that are allocated to eMAC are determined as part of the network scheduling process). These messages are therefore queued at the eMAC interface. The pMAC category is used for messages that cannot preempt an ongoing transmission. In other words, messages in the pMAC interface may be interrupted and preempted during transmission if an eMAC message is queued while the pMAC message is being transmitted. Typically, messages in pMAC are less time-sensitive; formally, the set of messages queued in pMAC are determined as part of the network scheduling process, like already mentioned for eMAC messages. In conclusion, only an eMAC message can preempt a pMAC message currently in transmission. Messages within the same category cannot preempt one another (e.g., two eMAC messages will each transmit to conclusion, in the order the messages were queued up). This results in a single level of preemption within TSN.[1]

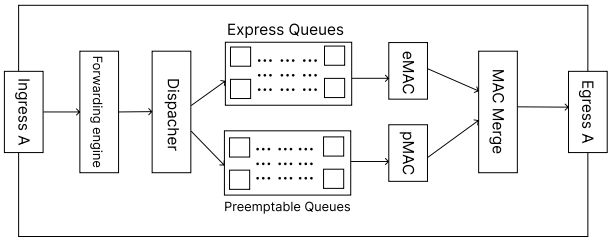

Fig. 1: TSN switch architecture

The format of a pMAC frame can vary due to the nature of preemption. At the sender's side, a message may be transmitted in multiple segments, each with a different format depending on the preemption state. This ensures that critical eMAC traffic is transmitted first, while preemptable pMAC traffic can be resumed once the eMAC message has been completed.

A message has a route that it follows from source to the destination. Endpoints in the network act as sources or destinations for messages. Each message is generated according to a given period and has a relative end-to-end deadline. Scheduling of messages with different periods is done for a hyper-period. The hyper-period of a set of messages is the Least Common Multiplier (LCM) of all message periods. Hyper-periods are used in TSN scheduling to synchronize the transmission times of messages with different periods, ensuring that all message instances and their deadlines are accommodated within a unified time frame. The transmission time of the payload of the message on the link is given by $\gamma = \frac{\alpha}{l}$, where γ is transmission time, α is the payload of the message, and l is the speed of the link traversed by the message. A schedule results in values for start and end times of transmission for message instances within a hyper-period. The obtained end time of transmission of each message instance should be smaller or equal to the corresponding deadline. Once the schedule for a hyper-period is established, it will be repeated cyclically. Each cycle of the hyper-period ensures that all messages are transmitted within their respective deadlines, allowing for consistent and predictable communication over the network.

The network topology is modeled as a graph $G = (N, E)$, where a node $n_i \in N$ is a network switch or an endpoint. We write n_i^t to mean the port number t (for t in $\{1, 2,...,e_i\}$) of node n_i, where e_i is the number of ports that node n_i has. Switches manage the flow of data between devices (endpoints) through dedicated communication links. E is the set of links

[1]While, theoretically, an arbitrary hierarchy of preemption can be supported, like task scheduling in operating systems, hierarchical preemption adds to the complexity of the state machine in the MAC. The IEEE 802.3 standardization groups decided for a single-level preemption, which is a practical trade-off among multiple factors such as cost, area overhead, and real-time performance.

TABLE I: Messages for the motivational example

Message	Source	Destination	Period = Deadline	Transmission time
m_1	n_1	n_5	6	1
m_2	n_1	n_6	18	4

in G. Links always connect ports with interfaces of the same category (i.e., they will never connect a port with frame preemption support to one that does not support preemption). A link $[n_{i_1}^{t_1}, n_{i_2}^{t_2}] \in E$ will therefore involve same category ports $n_{i_1}^{t_1}$ and $n_{i_2}^{t_2}$ (i.e., either both with frame preemption or both without). Network links are full-duplex, meaning they support bidirectional communication. Figure 2 depicts a network topology with three switches (n_2, n_3 and n_4) and three endpoints (n_1, n_5 and n_6).

We use a switch repository to choose appropriate switches or endpoints for a given network topology. A switch repository consists of switches of different brands, specifying the total number of ports, the number of ports that support frame preemption, and the piece cost of each switch on the market. Switches that support frame preemption generally hold higher cost tags compared to those that do not[2]. We are interested in the total cost of the network, which is given by the sum of the selected switches from the switch repository.

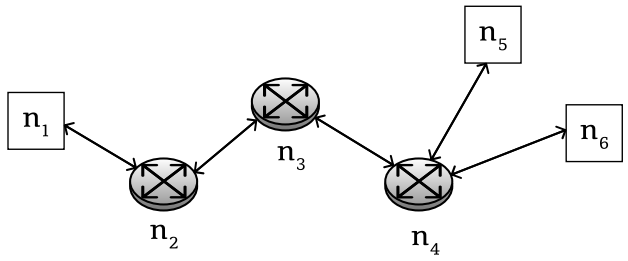

Fig. 2: The network topology for the motivational example

III. Motivational Example

Let us consider the network topology depicted in Figure 2 and assume we want to schedule the messages described in Table. I.

It is impossible for the network to schedule these messages without frame preemption. Indeed, any schedule will miss some deadlines. Figure 3 describes such a schedule for all message instances across all links in the route, for one hyper-period. Here, none of the links supports preemption (preemption supporting links will have the nodes at their extremities written in bold characters). As can be seen by the schedule for links [n_3, n_4] and [n_4, n_5], the deadline constraint for the second instance of message m_1 is not met as it had to wait for the long transmission time of the m_2 instance.

[2]The piece cost of a switch is affected by many factors, examples being number of ports, production volume, procurement model, time on the market, and the vendor itself, to name a few. Our abstraction does not restrict the impact of any of the aforementioned dimensions on the piece cost.

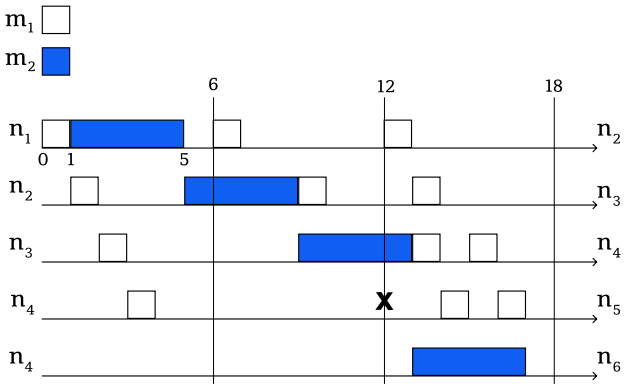

Fig. 3: Scheduling with none of the switches supporting preemption for motivational example: deadline constraint for the second instance of message m_1 cannot be satisfied (see links [n_3, n_4], [n_4, n_5])

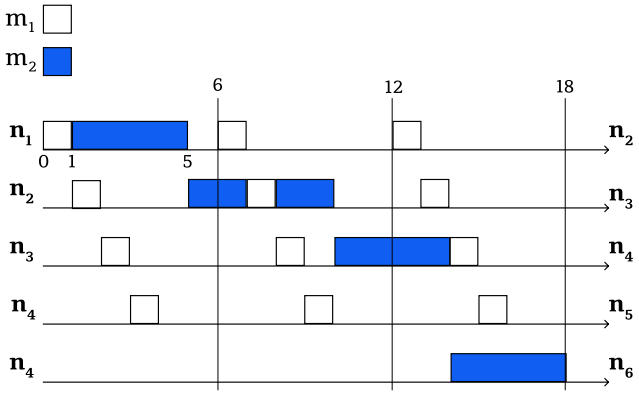

Fig. 4: Scheduling with all of the switches supporting preemption in the motivational example. All messages can meet their deadline, but the cost of the topology is the highest.

If all switches in the network support frame preemption and message m_1 is assigned to an eMAC interface while message m_2 is assigned to a pMAC interface in all of the egress ports, then a possible schedule is shown in Figure 4. This schedule respects the deadlines of all message instances.

However, if we assume (to simplify the presentation) that any port that does not support preemption costs 1 unit while any port that supports preemption costs 5 units, then this topology uses five preemptable links, resulting in 10 preemption supporting ports and hence a total cost of c_N = 50 units.

Importantly, even if all links support preemption (captured by the figure by having all nodes written in bold characters) the depicted schedule uses preemption only once on the link [n_2, n_3]. This means that there is no need for all of the switches to support frame preemption. Figure 5 shows a schedule where

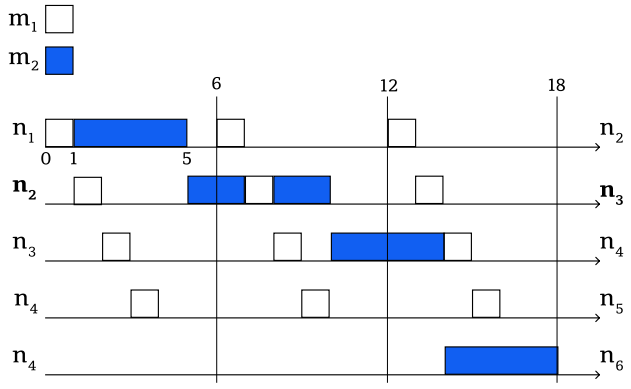

Fig. 5: Scheduling with two of the switches (link $[n_2, n_3]$) supporting preemption for motivational example: all messages can meet their deadline and the cost is optimized.

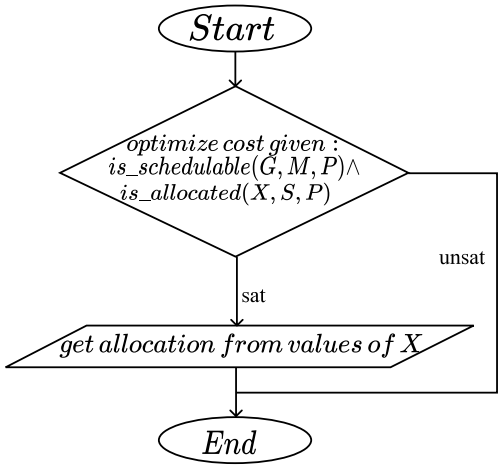

Fig. 6: A Monolithic SMT based approach

only nodes n_2 and n_3 need to be linked via preemptive interfaces. In this case, four out of the five links in the network need not involve preemption, leading to two preemptive ports and eight non-preemptive ports and resulting in a total cost of c_N = 18 units while still meeting deadlines' constraints for all message instances. As a result, the cost was reduced from 50 to 18 units, illustrating that judicious switch and port allocation can maintain schedulability while minimizing cost.

IV. Problem formulation

This paper addresses the joint problem of minimizing hardware preemption cost and scheduling real-time communications across packet-switched Ethernet networks. For this it takes as input:

- A network topology given by graph $G = (N, E)$.
- A set M of $|M|$ messages where each $m \in M$ comes with period $\texttt{period}(m)$, a transmission time $\texttt{ttime}(m)$, a deadline $\texttt{deadline}(m)$ and a route $\texttt{route}(m)$. A route is a simple path from the message source to its destination.
- A set S of $|S|$ switches and endpoints where each $s \in S$ comes with a cost $\texttt{cost}(s)$ and with a total number of ports $\texttt{ports}(s)$ (of which the number of preemptable ports are denoted by $\texttt{pports}(s)$). Observe that $\texttt{pports}(s) \leq \texttt{ports}(s)$

A solution to the problem will:

- Choose a switch for each node in the topology, i.e., perform switch allocation.
- Decides which (ports connected to) links are preemptable, i.e., perform port allocation.

The objective is to assign to each node in the network a switch from the switch repository in order to allow enough frame preemption to ensure schedulability while minimizing the amount of needed preemptable ports to simultaneously minimize the cost of the entire network.

V. SMT Formulation for switch allocation

We describe in the following an SMT formulation to solve the switch/port allocation and cost minimization problem described in the previous section. Our formulation will allow an optimizing SMT solver to minimize network cost while solving switch and port allocation and ensuring schedulability.

a) TSN scheduling: Several works use SMT solvers to solve TSN schedulability problems [4], [15], [16]. None, to the best of our knowledge, combines it with cost minimization and with switch and port allocation. We leverage on Zhou et. al. SMT based formulation where the challenge is to find a TSN schedule regardless of preemption [12]. For this, SMT constraints are introduced to obtain schedules in graph topologies where ports are assumed to allow frame preemption. We use these SMT constraints to obtain, given a set of messages M with fixed routes, a possible TSN scheduling of the messages (i.e., concrete values of start and end transmission times of each message instance on each link) for a given network topology where the set of preemptable ports is given. Using a "preemptability variable" p_i^t for each port n_i^t and given start and end transmission times of messages for each link, we constrain p_i^t to equal 1 if one of the messages passing by n_i preempts another on the same link, and 0 if no message instance preempts another on that port. These "preemption variables" only reflect whether a solution needed preemption on some port. We propose in this paper three approaches for solving the allocation and minimization problem from III. All three approaches use these schedulability constraints that track preemptability. We write:

$$\texttt{is\_schedulable}(G, M, P) \tag{1}$$

to refer to the TSN scheduling constraints given a graph topology G, a set of messages M, and a set P of "preemptability variables" for all ports in G.

b) Switch and port allocation: To solve the allocation problems we need to choose a suitable switch for each node in the graph. We capture this choice using the decision variables:

$$x_i^j \in \{0,1\}, i = 1, ...|N|, j = 1, ...|S| \tag{2}$$

We use $x_i^j = 1$ to mean that switch with index j is associated to node n_i. We now introduce our constraints for switch allocation for a given topology:

1) To make sure that only one switch or endpoint is selected for each node we introduce following constraint:

$$\sum_{j=1}^{|S|} x_i^j = 1, \text{for each } i = 1, ...|N| \tag{3}$$

2) Constraint to map the ports from topology to selected switch. For any node n_i and any s_j:

$$x_i^j = 1 \Rightarrow \begin{cases} \texttt{ports}(s_j) \geq e_i \\ \sum_{t=1}^{e_i} p_i^t \leq \texttt{pports}(s_j) \\ e_i - \sum_{t=1}^{e_i} p_i^t \leq \texttt{ports}(s_j) - \texttt{pports}(s_j) \end{cases} \tag{4}$$

This will make sure that selected switch has enough preemptable and non-preemptable ports for each node.

3) Making sure that two ports on the same link both support frame preemption:

$$p_{i_1}^{t_1} = p_{i_2}^{t_2} \quad \text{for any} \quad [n_{i_1}^{t_1}, n_{i_2}^{t_2}] \in E \tag{5}$$

4) Cost function calculates the total cost of the topology:

$$Cost = \sum_{i=1}^{|N|} \sum_{j=1}^{|S|} x_i^j \texttt{cost}(s_j) \tag{6}$$

Given the set X of decision variables introduced in 2, the set of switches S and the set of "preemptability variables" P, We will write

$$\texttt{is\_allocated}(X, S, P) \tag{7}$$

to refer to the conjunction of the constraints described in 2, 3, 4, 5, and 6.

One way to solve the problem discussed in Section IV is to ask an optimizing SMT solver (we used the default optimization feature of Z3 [17]) to solve both the schedulability problem (by satisfying $\texttt{is\_schedulable}(G, M, P)$) and the allocation problem (by also satisfying $\texttt{is\_allocated}(X, S, P)$) while obtaining a lowest cost. We call this approach the *Monolithic SMT* approach. It is depicted in Figure 6. Assuming termination, this approach will result in a lowest cost while solving the allocation and the schedulability problems. If it fails, then the problem is guaranteed to not have a solution: either because the messages are not schedulable even if frame preemption is enabled on all ports, or because no switch allocation can introduce enough preemption. Our experiments do confirm that this approach suffers from scalability issues. This result in the approach often timing out leaving the user without a solution, let alone an optimal one.

VI. Proposed heuristic approaches

To address the poor scalability of Monolithic approach described above, propose three heuristics: Time slicing, incremental Time slicing, and our score based heuristic.

A. (Incremental) Time slicing of hyper-periods

Time slicing has been used [18] to increase the chances of termination when using SMT formulations of scheduling. The idea is to divide the hyper-period into slices. This reduces the number of message instances that need to be scheduled. When time slicing our monolithic SMT formulation, the SMT solver will solve scheduling for messages which start time happens in the slice, switch allocation, and cost minimization one slice at a time, while assuming the values of scheduling from previous slices, as well as the variables p_i^t that were assigned to 1 (preemptable) from previous time slices. After each slice is solved, the schedule for each instance of m within the slice, and the p_i^t that were assigned as preemptable are retrieved and given to the next slice as constants. The ports that were not assigned preemption are kept as a decision variable, as in monolithic SMT formulation, for the next slice, as preemption may or may not be needed to find a schedulable solution. Because the preemption decisions are retained from one slice to the next, the switch selection that results from the last slice becomes the output from the overall Time slicing heuristic. This iterative SMT approach results in smaller problems for the SMT solver to handle, enabling better scalability and more efficient resource allocation. We write

$$\texttt{is\_schedulable}_k(G, M, P) \tag{8}$$

to mean the slicing the hyper-period of the $\texttt{is\_schedulable}(G, M, P)$ problem into k slices.

However, this approach will likely miss optimal solutions (or may even fail to find a solution) because of the solutions it had to assume from earlier slices. As a result, it may miss some of the schedules and not give an optimal switch allocation.

An alternative Time sliced based approach consists in trying to solve the problem with fewer time slices first (improving completeness by missing less opportunities to solve and optimize the problem). Only if a fixed timeout is consumed will this incremental approach increase the number of slices. We call this the incremental Time slicing heuristic. It is depicted in Figure 7.

B. Score Based Heuristic Approach

In our score based heuristic approach we separated switch allocation and cost optimization from scheduling. This is achieved by making decisions about whether ports on the same link need to have frame preemption capabilities. The idea then is to introduce preemption greedily based on a score that strives to reflect, for each link, the corresponding need for preemption. Intuitively, our scoring formula for a given link accounts for its utilization and for the "eagerness" of

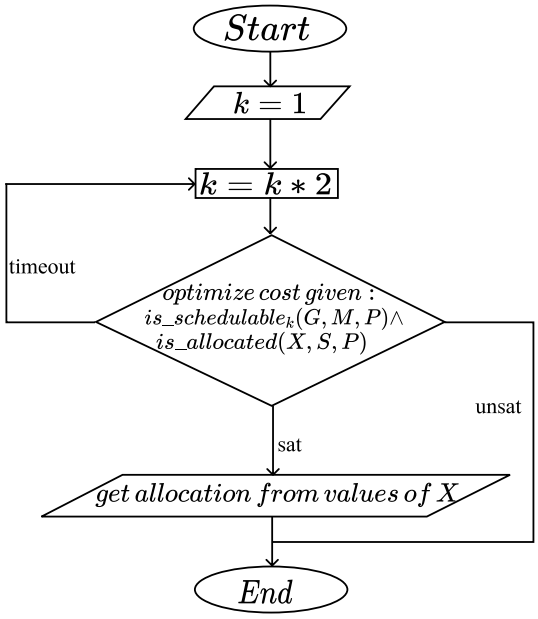

Fig. 7: An incremental Time slicing approach

the messages passing by it. Utilization corresponds to the ratio between transmission times and periods of the messages passing by the considered link. The higher the ratio, the less margins the scheduler has to meet the deadlines. We capture "eagerness" as the ratio between the deadlines and the transmission times of the messages passing by the link. The higher the ratio, the more margins the scheduler will have to schedule the message within the deadline. This is confirmed by our experiments with Monolithic approach. Indeed, we tracked the relation between the number of preemptable ports and the deadline. For instance, Figure 8 shows the number of needed preemptable ports as chosen by the monotlithic approach decreased as deadlines increased (given we keep all other parameters equal).

1) Scoring system: our scoring mechanism consider the utilization of each link within the network topology. This evaluation is based on the period, the transmission time, and the deadline of all the messages that traverse the link. The score $L_{[n_{i_1}^{t_1}, n_{i_2}^{t_2}]}$ for each link $[n_{i_1}^{t_1}, n_{i_2}^{t_2}] \in E$ in the network topology requires to first identify any message m that passes through the link, i.e., the link $[n_{i_1}^{t_1}, n_{i_2}^{t_2}]$ is part of $\texttt{route}(m)$, then, we use the following weighted sum as score for the link:

$$L_{[n_{i_1}^{t_1}, n_{i_2}^{t_2}]} = \sum_{m \in [\![n_{i_1}^{t_1}, n_{i_2}^{t_2}]\!]} \frac{\texttt{ttime}(m)}{\texttt{period}(m)} * \left(2 - \frac{\texttt{deadline}(m)}{\texttt{ttime}(m)}\right) \tag{9}$$

Where:

- $[\![n_{i_1}^{t_1}, n_{i_2}^{t_2}]\!]$ represents the total number of messages in the system,
- $\texttt{period}(m)$ is the period of message m,

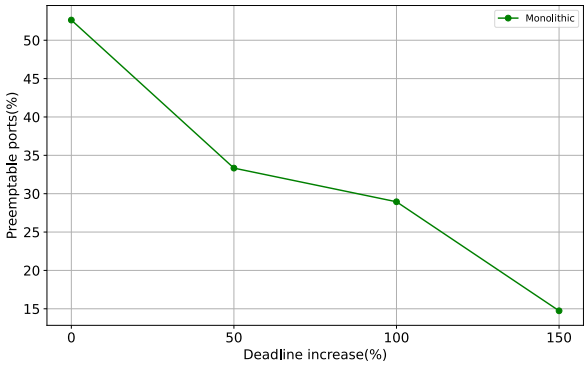

Fig. 8: Number of preemptable ports as message deadlines increase

- $\texttt{ttime}(m)$ is the transmission time for message m,
- $\texttt{deadline}(m)$ is the deadline for message m

Since the network links are full-duplex the score is computed for both directions of each link. This means that the scoring system accounts for both directions, $[n_{i_1}^{t_1}, n_{i_2}^{t_2}]$ and $[n_{i_2}^{t_2}, n_{i_1}^{t_1}]$, separately. The direction with the highest score is then selected as the final score for the link. This approach allows for considering the impact of each direction of communication separately, ensuring that the direction with the highest utilization is prioritized during the scheduling process.

2) Schedulability check: The schedulability check is done by constructing the preemptive TSN schedule through SMT (either Monolithic or Time slicing, depending on the size of the problem and afforded runtime complexity). Note that this does not involve switch selection and optimization as those decisions are part of our heuristic.

3) Frame preemption allocation: In Monolithic approach with SMT, the solver decides whether or not to allocate a switch with the support of frame preemption depending on whether it can construct a schedule for the network topology. With the score based heuristic approach, the decision to enable frame preemption on a link depends on the score of that link $L_{[n_{i_1}^{t_1}, n_{i_2}^{t_2}]}$. Links are first inserted in a max priority queue based on their score Figure 9. Recall that variable p_i^t captures whether port t of node n_i uses frame preemption. By initializing all preemptability variables to zero, we forbid any frame preemption initially. We then try to construct a schedule by using a (Time sliced) query to the SMT solver where we select which ports might use frame preemption. If construction fails, frame preemption capability is added to each link in descending order of their respective score. After each addition, a schedule is constructed to evaluate whether the network can schedule the messages. Our score based heuristic stops adding frame preemption capability to the links when messages are fully scheduled.

Switch selection For each node i, now we have a value to each port p_i^t, thus deciding whether it should support frame

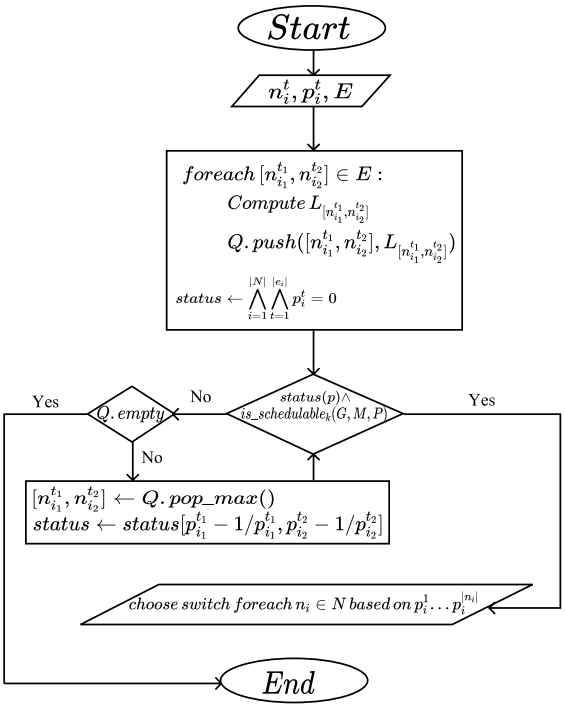

Fig. 9: Our score based heuristic

preemption. As a post-processing step, for each switch, we select the lowest-cost switch from the repository that has sufficient number of preemptable and nonpreemptable ports as decided by our heuristic 4. This minimizes the total switch cost, constrained by the preemption decisions made by our heuristic.

VII. Experiments

In this section we will evaluate the efficiency of the proposed approaches. All of the experiments are conducted on an AMD Ryzen ThreadRipper PRO 3955WX with 64 GB of RAM and running Ubuntu 22. The SMT solver we used to solve the synthesis problem is z3 [17], which is a state-of-the-art SMT solver. In our experiments, we consider all link speeds to be 100 Mbit/s, and message size to be between 60 to 1000 bytes. The period for all messages range from 5 to 100 ms. We consider networks with 22 up to 44 nodes. A dataset with 42 different hardware configuration switches is used for switch repository. Same switch can be chosen for several nodes in the network topology. The network topology is generated randomly using the Erdos-Renyi graph model [19]. All of the experiments have been tested to not schedule without frame preemption. The main problems that are being tackled are scheduling of the messages, switch allocation within the network topology and cost optimization of overall cost of the network topology. We tested 3 approaches.

1) Monolithic approach is fully integrated SMT based switch allocation, cost optimization and scheduling approach. It solves the problems for one hyper-period. The timeout for the experiments with Monolithic approach is 1 hour.

2) Time slicing approach uses the same logic as Monolithic approach. While Monolithic approach solves the problems for whole hyper-period, Time slicing divides the hyper-period into several slices. The slicing is done in an incremental fashion starting with 2 up to 16 multiplied by 2, and are done until the schedule is successfully constructed. This is done to ensure scaling while preserving quality of the solution. For each try to solve the problems the timeout is 15 minutes.

3) Heuristic approach has separate switch allocation and cost optimization algorithms integrated with SMT based scheduling without optimization. Scheduling process is time sliced into 16 slices to ensure scaling. The timeout for the experiments is 1 hour. The timeout is the same with other approaches because Heuristic approach is trying to construct the schedule several times during the switch allocation.

Experiments that time out are assigned timeout time to the runtime results. To indicate that no solution was found, these experiments are assigned the highest cost in the results that are reported.

A. Approach Comparison

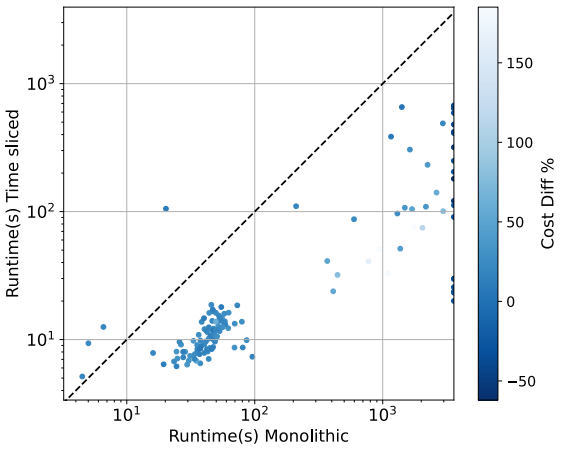

Fig. 10: Runtimes comparison and percentages of cost differences between Time sliced and Monolithic approaches. Monolithic approach always obtains cheaper costs than Time sliced approach when it does not time out.

In this section, we explore how well did the three proposed approaches perform compare to each other. We compare results of runtime and the achieved costs. For this experiment, we used 40 to 80 messages and 22 to 44 nodes in the topology. *1) Monolithic and Time slicing approach:* Figure 10 shows the comparison between runtime and cost of the topology of monolithic and Time slicing approach. The majority of the experiments performed better with Time slicing approach than Monolithic approach in terms of runtime. We can observe

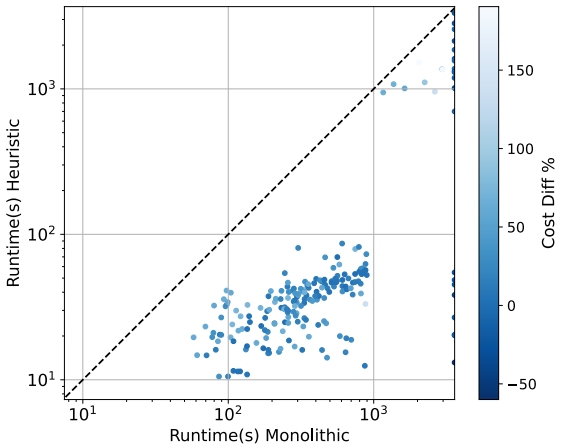

Fig. 11: Runtimes comparison and percentages of cost differences between Heuristic and Monolithic approaches. Monolithic approach always obtains cheaper costs than Heuristic approach when it does not time out.

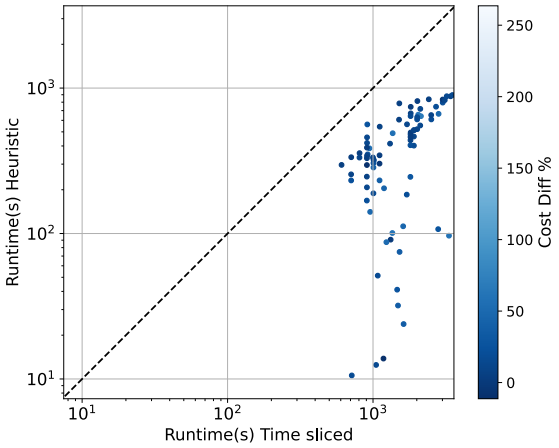

Fig. 12: Runtimes comparison and percentages of cost differences between Heuristic and Time sliced approaches. Heuristic approach always obtains cheaper costs than Time sliced approach.

several of the experiments done by Monolithic approach not terminating and timing out. However, it is important to mention that with small scale experiments, Monolithic approach can perform better that Time slicing because Monolithic is solving the problems for one hyper-period and is thus an optimal solution, whereas Time slicing is dividing the hyper-period making it a heuristic approach. This means that after the problems are solved for the first half, the results will be saved and the second half will need to solve the problem based on the variables in the first half being fixed. This is an important observation because it shows that Time slicing is useful for scaling and doing larger experiments. The cost comparison of the topologies is showing that when both approaches terminated, Monolithic approach shows better results in terms of cost than Time slicing.

2) Heuristic and Monolithic approach: In Figure 11 we can see that all of the experiments fall to right side of the diagonal showing that Monolithic approach demonstrates poorer performance in terms of runtime, often failing to terminate within the specified timeout. However, when Monolithic approach does complete, it generally yields lower costs compared to Heuristic method. The reason why there is no experiments where Monolithic approach performed better in terms of the runtime than heuristic is that heuristic uses Time sliced scheduling while Monolithic does it for the whole hyper-period. The relation between runtime and number of slices is clearly shown in VII-B.

3) Heuristic and Time slicing approaches In Figure 12 we observe that both Heuristic and Time slicing approaches terminated and Heuristic performed better in terms of runtime than Time slicing. We can also see that there are a few cases where Heuristic out did Time slicing almost 10 times. This can be explained by the fact that Time slicing is starting with 2 slices and incrementing (with a factor of 2) if a result is not found within 15 minutes (the time out value we set). That is, Heuristic starts constructing the schedule directly with 16 slices. Heuristic approach accounts for utilization and link ”eagerness” when greedily adding preemption capabilities. This is unlike Time slicing approach which adds preemption to minimizes costs one time slice at a time. Time slicing may therefore yield poor decisions that will badly affect the resulting costs.

B. Scalability and cost comparison

In this section, we will explore the scaling and cost comparison of the proposed approaches as the number of nodes and the number of switches increase. All points in the following graphs represent the average of the experiments. We used the same ranges for periods, deadlines, and message sizes.

For the experiments, we fix the number of messages to be 60 and evaluate the impact of increasing the number of nodes in the topology. Figure 13 and Figure 14 show the results for the increasing amount of nodes.

Monolithic approach: Monolithic approach exhibits the worst runtime performance, showing poor scalability as the number of nodes increases. Its runtime grows rapidly and times out at 40 nodes, making it less suitable for larger topologies due to the high computational costs. As expected, in cases it was able to terminate, it showed the best results in terms of cost effectiveness. This happens because monolithic SMT formulation is an optimal approach.

Heuristic Approach: Heuristic approach offers a more favorable runtime compared to Monolithic approach, with better scalability. Although it still experiences an increase in runtime with the number of nodes, it remains more efficient and

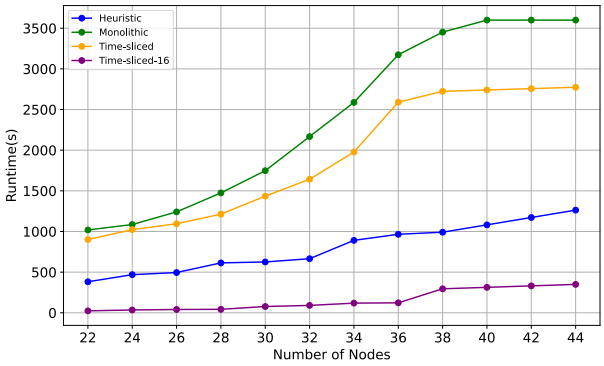

Fig. 13: Runtime of Monolithic, Time sliced, Heuristic approach, and Time slicing approach with 16 slices with 60 messages and increasing amount of nodes in network topology

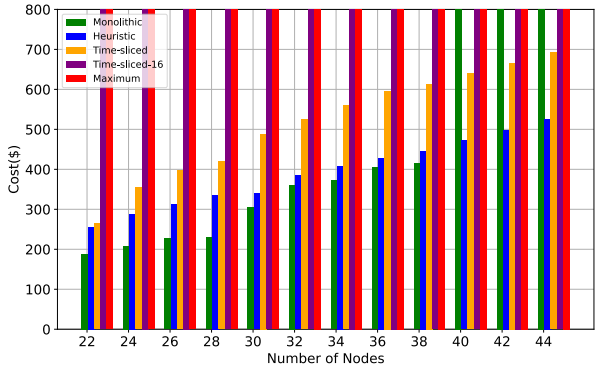

Fig. 14: Cost comparison between Monolithic, Time sliced, Heuristic approach and Time slicing approach with 16 slices with 60 messages and increasing amount of nodes in TSN network topology

demonstrates a balanced cost-effectiveness. Compared to Time slicing approach, our heuristic takes into account utilization of the links when making switch allocation. This makes it more cost effective than Time slicing approach.

Time sliced Approaches: Time sliced approach shows improved runtime efficiency, with the runtime increase being slower than Monolithic which indicates better scalability. It is less efficient than Heuristic approach because it needs to increase the amount of slices and run the formulation again if it was not able to solve the problem with two slices. This iteration contributes to the higher runtime.

Time slicing with 16 slices: Among all approaches, Time slicing with 16 slices demonstrates the best performance in terms of runtime, offering the most efficient scaling, but with the worst cost effectiveness. This is the results of increasing the amount of slicing. With higher amount of slicing, the original search space is heavily reduced (thus, potentially good solutions being eliminated) at the benefit of reducing runtime and increasing scalability.

In the second set of experiments, we fix the number of nodes to 20 and evaluated the impact of increasing number of messages from 50 to 85 incrementally by 5 Figure 15 and Figure 16. The tendency of the graph is the same as in the first experiments.

C. Preemption distribution comparison

For this set of experiments we used 30 messages and topology with 20 nodes. This was done deliberately to ensure that all approaches terminate with a solution before timeout. This way we can observe the performance of all approaches. All data points in the following graphs represent the average of the experiments.

Period decrease, In the first set of experiments, we kept the deadlines and size constant, gradually decreased the period by 20%. As shown in Figure 17, this decrease in period led to a greater demand for preemption, requiring more preemptable ports. A shorter period reduces flexibility in message

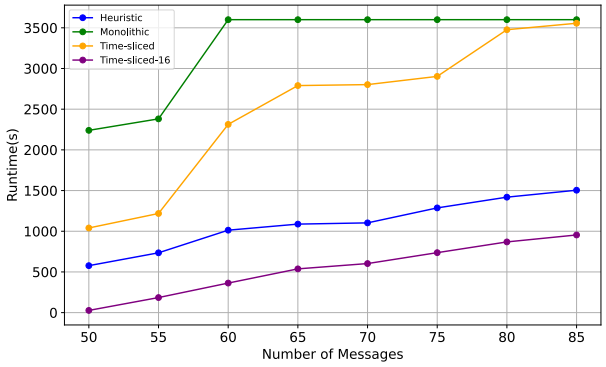

Fig. 15: Scaling of runtime of Monolithic, Time sliced, Heuristic approach and Time slicing approach with 16 slices with 20 nodes and increasing amount of messages in TSN network topology

allocation, making it more challenging to schedule without preemption. Consequently, scheduling becomes less efficient as the period declines due to the limited ability to allocate resources without conflicts.

Size decrease, for the second set of experiments we kept all of the deadlines and periods in the same range, but gradually decreased the value of the size by 33%. Figure 18 illustrates that this size decrease reduces the need for preemption. With shorter message durations, the scheduling process becomes more straightforward due to fewer conflicts.

Deadline increase, in the third set of experiments, we kept the message sizes and periods fixed, incrementally increasing the deadline by 50%. As shown in Figure 19, a deadline increase leads to a reduced need for preemption. A larger deadline relaxes the scheduling constraints, providing more time to schedule tasks without conflicts, thereby reducing reliance on

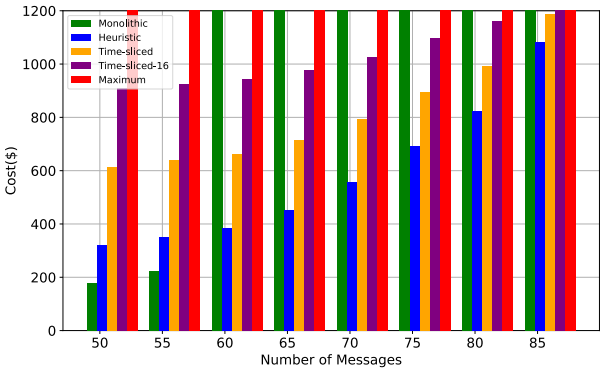

Fig. 16: Cost comparison between Monolithic, Time sliced, Heuristic approach and Time slicing approach with 16 slices with 20 nodes and increasing amount of messages in TSN network topology

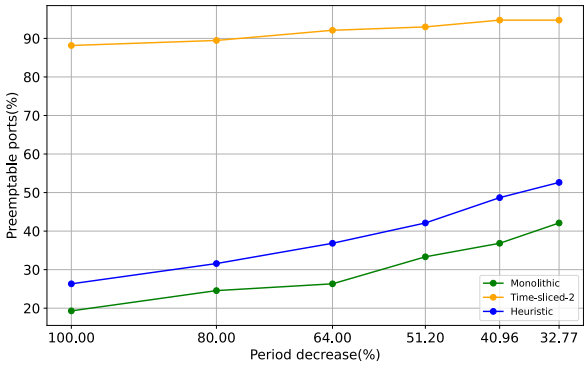

Fig. 17: Distribution of preemptable port in the topology with co-relation to decrease of period of the messages

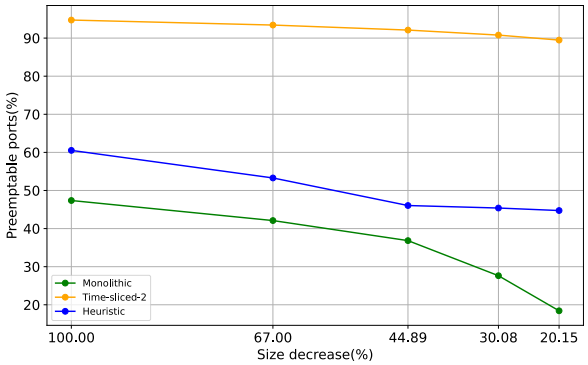

Fig. 18: Distribution of preemptable port in the topology with co-relation to decrease of size of the messages

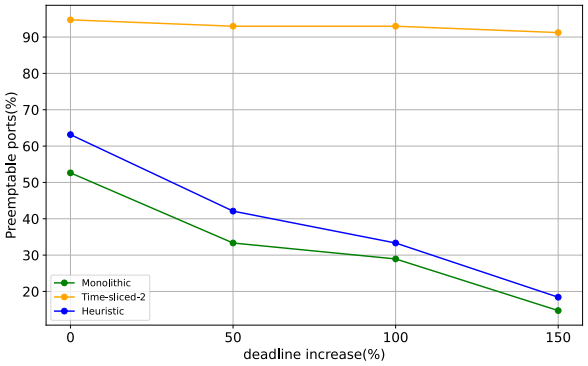

Fig. 19: Distribution of preemptable port in the topology with co-relation to increase of deadline of the messages

preemption. The extended deadline allows more flexibility in message allocation, reducing scheduling complexity.

In Figure 17, Figure 18, and Figure 19 we can see a comparison of performance of three approaches. Monolithic approach gives the most accurate result. However, we need to mention that Heuristic approach is able to approximate Monolithic approach more closely because it takes into account the utilization of each link in descending order. This allows more optimization for Heuristic. Time slicing is done with slicing of hyper-period into 2. It shows very low optimization which results in higher cost for the all topology which was shown in VII-B.

In summary, we conclude that our extensive experiments support our heuristic yielding cost effective preemptable scheduling and switch allocation in a scalable and time-efficient manner for very large real-time Ethernet networks.

VIII. Conclusion

We considered three switch allocation approaches for cost optimization of network topologies integrated with preemptable TSN scheduling of real-time communications. We proposed and evaluated three approaches: Monolithic SMT, incremental Time slicing, and a Score-based Heuristic. Extensive experimental evaluation was presented, demonstrating the quality, effectiveness, and scalability of our approach. By using our approach, system designers are able to use TSN preemption technology in a cost-optimal manner in scenarios where otherwise cost-prohibitive and practically disruptive solutions, involving physical architecture and bandwidth upgrades, would have to be considered. Future works can build on our approach to improve schedulability. For instance, it can be interesting to leverage on our approach to identify streams that might be partitioned into multiple substreams, enabling the usage of multiple routes, thus requiring even less preemption.

ACKNOWLEDGMENT

The research has been partially funded by CUGS (Graduate School in Computer Science) and ELLIIT (Excellence Center at Linköping-Lund in Information Technology).

REFERENCES

[1] "IEEE Standard for local and metropolitan area networks - timing and synchronization for time-sensitive applications in bridged local area networks," tech. rep., 2011.

[2] "IEEE Draft Standard for local and metropolitan area networks-bridges and bridged networks asynchronous traffic shaping," *IEEE P802.1Qcr/D0.5, June 2018*, pp. 1–112, 2018.

[3] "IEEE Standard for local and metropolitan area networks–frame replication and elimination for reliability," *IEEE Std 802.1CB-2017*, pp. 1–102, 2017.

[4] W. Steiner, G. Bauer, B. Hall, and M. Paulitsch, *Time-triggered ethernet*, pp. 181–220. 01 2011.

[5] Y. Zhou, S. Samii, P. Eles, and Z. Peng, "Partitioned and overhead-aware scheduling of mixed-criticality real-time systems," in *2019 24th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 1–6, 2019.

[6] S. S. Craciunas, R. Serna Oliver, and W. Steiner, "Demo abstract: Slate xns–an online management tool for deterministic tsn networks," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 103–104, 2018.

[7] W. Steiner, "An evaluation of smt-based schedule synthesis for time-triggered multi-hop networks," in *2010 31st IEEE Real-Time Systems Symposium*, pp. 375–384, 2010.

[8] Y. Zhou, S. Samii, P. Eles, and Z. Peng, "Asil-decomposition based routing and scheduling in safety-critical time-sensitive networking," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 184–195, 2021.

[9] Y. Zhou, S. Samii, P. Eles, and Z. Peng, "Reliability-aware scheduling and routing for messages in time-sensitive networking," *ACM Trans. Embed. Comput. Syst.*, vol. 20, May 2021.

[10] Z. Jiang, N. C. Audsley, and P. Dong, "Bluevisor: A scalable real-time hardware hypervisor for many-core embedded systems," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 75–84, 2018.

[11] Z. Jiang, K. Yang, N. Fisher, N. Audsley, and Z. Dong, "Pythia-mcs: Enabling quarter-clairvoyance in i/o-driven mixed-criticality systems," in *2020 IEEE Real-Time Systems Symposium (RTSS)*, pp. 38–50, 2020.

[12] Y. Zhou, S. Samii, P. Eles, and Z. Peng, "Time-triggered scheduling for time-sensitive networking with preemption," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 262–267, 2022.

[13] T. Stüber, L. Osswald, S. Lindner, and M. Menth, "A survey of scheduling algorithms for the time-aware shaper in time-sensitive networking (tsn)," *IEEE Access*, vol. 11, pp. 61192–61233, 2023.

[14] C. Xue, T. Zhang, Y. Zhou, M. Nixon, A. Loveless, and S. Han, "Real-time scheduling for 802.1qbv time-sensitive networking (tsn): A systematic review and experimental study," in *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 108–121, 2024.

[15] T. Park, S. Samii, and K. G. Shin, "Design optimization of frame preemption in real-time switched ethernet," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 420–425, 2019.

[16] M. A. Ojewale, P. M. Yomsi, and B. Nikolić, "Multi-level preemption in tsn: Feasibility and requirements analysis," in *2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)*, pp. 47–55, 2020.

[17] L. de Moura and N. Bjørner, "Generalized, efficient array decision procedures," in *2009 Formal Methods in Computer-Aided Design*, pp. 45–52, 2009.

[18] R. Mahfouzi, A. Aminifar, S. Samii, A. Rezine, P. Eles, and Z. Peng, "Stability-aware integrated routing and scheduling for control applications in ethernet networks," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 682–687, 2018.

[19] P. Erdős and A. Rényi, "On random graphs i," *Publ. Math. Debrecen*, vol. 6, pp. 290–297, 1959. 18.