Computing, Performance and Communication Systems (2024)
Clausius Scientific Press, Canada

DOI: 10.23977/cpcs.2024.080109
ISSN 2371-8889 Vol. 8 Num. 1

Improved Heuristic Joint Routing and Scheduling in Time-Sensitive Networking

Li Chuang[1,a,*], Wang Zhong[1,b]

[1]Xi'an Research Institute of High Technology, Xi'an, Shaanxi, 710025, China
[a]chuangliwhisper@163.com, [b]dsp863wang@163.com
[*]Corresponding author

Keywords: Time-Sensitive Networking; Traffic scheduling; Heuristic algorithms; Routing; Search strategy

Abstract: The joint routing and scheduling problem in Time-Sensitive Networking is a Non-deterministic Polynomial problem that can be solved using heuristic algorithms. This paper proposes an improved heuristic scheduling algorithm, optimizes its search strategy, proposes a shorter path first strategy, standardizes the solution space of the heuristic algorithm, improves the search speed of the algorithm, and carries out scenario solving experiments and compares the solving efficiency with the other five algorithms. The results show that the improved heuristic scheduling algorithm has a greater improvement in the solution efficiency, compared with the other five algorithms to improve the overall solution efficiency of more than 17%, and is more adaptable to the complex topology and task flow of the scene.

1. Introduction

Time-Sensitive networking (TSN) is a new network architecture that is able to be compatible with traditional Ethernet while providing users with low latency, low jitter, and deterministic traffic transmission performance. It is increasingly being recognized in fields such as autonomous driving, smart factories, and remote healthcare[1],[2]. The related protocols of Time-Sensitive Networking are still being improved by the Time-Sensitive Networking task group. Traffic shaping scheduling is one of the key technologies of Time-Sensitive Networking [3],[4],[5]. The IEEE802.1Qbv protocol specifies a time-aware shaper that utilizes a gating table for scheduling time-sensitive flows. Efficiently solving the gating table is a critical aspect of research in traffic shaping and scheduling technology[6].

Fixed route traffic scheduling is a method of solving the gating table for task flows based on topology and task flow information, using shortest path routing. However, this strategy has a limited solution space and may fail when a path has a high route occupancy rate, leading to gating table resolution failure. The joint routing solving strategy expands the solution space for scheduling problems, not limited to fixed shortest paths. It solves the gating table while meeting transmission requirements, and effectively enhances traffic scheduling success rates through combined routing and scheduling strategies.

The routing and scheduling problem has been proven to be an NP-hard problem [7]. Various approaches, such as Satisfiability Modulo Theories, Integer Linear Programming, and heuristic

algorithms, have been employed to address routing and scheduling problems. Compared to the other two methods, the heuristic algorithm offers faster solving speeds and can quickly adapt to application scenarios with changes in topology or task flow. Therefore, enhancing the solving efficiency and scheduling success rate of the heuristic algorithm is a significant problem, crucial for meeting the efficiency requirements of dynamic scheduling. This paper proposes a more efficient shortest path first search strategy, based on the heuristic scheduling algorithm discussed in article [8], which significantly improves the solving efficiency of the heuristic scheduling algorithm.

2. Related Work

Previous studies employing heuristics for traffic scheduling can be categorized into two groups: those utilizing heuristics for traffic scheduling with fixed routes, and those employing heuristics for traffic scheduling with joint routes.

For traffic scheduling with fixed routes, Jin et al. [9] proposed the Move-Forward Schedule (MFS) heuristic algorithm, which reduces the use of scheduling tables and improves the efficiency of traffic scheduling. Zhang et al. [10] designed a traffic-aware scheduling algorithm with a cyclic interval search function, which compresses the invalid search space and improves the scheduling speed. Vlk et al. [11] proposed a self-sufficient scheduling algorithm capable of handling large-scale traffic scheduling problems in a relatively short time without relying on third-party solvers.Bujosa et al. [12] introduceda rapid heuristic scheduing algorithm that utilizes multiple time-triggered queues to significantly enhance the efficiency for generating traffic schedules. The aforementioned works employ heuristic algorithms and specifically address traffic scheduling for fixed routes, thereby enhancing the efficiency of traffic scheduling. Nevertheless, the absence of routing union and limited solution space hinders the identification of viable solutions.

For joint routing and traffic scheduling, Pahlevan et al. [8] introduced a heuristic list scheduler that effectively addressed the interdependence of routing and scheduling constraints, leading to significant improvements in scheduling efficiency. Falk et al. [13] proposed a memory-efficient method based on a configuration conflict graph, which demonstrated superior scheduling performance. Atallah et al. [14] combined iterative scheduling technology with perceptual multipath routing technology to minimize conflicts and achieve excellent results in managing large-scale networks with high utilization. The aforementioned studies employ diverse methods to enhance the scheduling performance and efficiency of heuristic algorithms. However, there is scope for further improvement in addressing efficiency issues in scenarios characterized by complex network topologies and task flows.

Xue et al. [15] categorized seventeen notable traffic scheduling solutions and conducted a comprehensive comparison of these algorithms. They emphasized the strengths and weaknesses of each approach in different scenarios, thereby offering valuable insights for future research and development in the field of TSN. This paper utilizes the open-source code and dataset provided by Xue et al. [15] to assess the algorithm's performance.

3. System Model and Problem Description

3.1. System Model

In this paper, a directed graph G is employed to depict the network topology, which consists of a set of nodes V and a set of links E. The node set V is denoted as $V = \{1, 2, , , n\}$, where each node is represented by an ID. Similarly, the link set E is represented by a collection of unidirectional links, denoted as $E = \{(1,2),(1,n),(2,1),,,(n,1)\}$. It is essential for the network topology G to be a strongly

connected graph.

3.2. Traffic Model

The traffic set F records transmitted traffic f within the network topology, denoted as $F=\{f_1, f_2,,, f_i\}$. These traffic attributes include flow sequence number i, source address s, destination address d, packet size m, traffic period p, delay requirement t, and jitter requirement j, denoted as $f=(i,s,d,m,p,t,j)$.

3.3. Problem Description

Based on identifying the optimal path for the task flow, joint routing and scheduling determine the transmission timing at the source node and configure the gate control list of the time-aware shaper at the switch node to satisfy the deterministic delay requirements of the task flow.

Compared to the Integer Linear Programming-based solution, heuristic algorithms offer faster solving speeds while still achieving feasible solutions. However, to effectively adapt to dynamic scheduling scenarios involving changes in topology and task flow, heuristic algorithms require even faster solving speeds.

Currently, efforts to enhance the efficiency of heuristic algorithms primarily concentrate on three aspects: designing heuristic functions, minimizing search space, and optimizing search strategies.

In the context of traffic scheduling with joint routing, the routing hop count of a task flow can serve as a cost index in heuristic functions. Considering the routing hop count in the heuristic function enhances the algorithm's ability to achieve optimal solutions more efficiently.

During the heuristic algorithm solving process, finding a feasible solution within the solution space is crucial. However, a larger solution space can diminish solving speed; hence, preemptively eliminating infeasible solution paths enhances algorithmic efficiency. In joint routing traffic scheduling, constraints based on factors like path occupancy can reduce the search space, thereby speeding up heuristic algorithm solutions.

During heuristic algorithm execution, searching the solution space consumes the majority of the time. Improving the search strategy can expedite the discovery of feasible solutions.

4. Heuristic Scheduling Algorithm and Optimization Strategy

4.1. Heuristic Scheduling Algorithm

The primary procedure of the heuristic scheduling algorithm in Paper [8] is detailed in Algorithm 1. Step 2) involves computing all paths in the network topology for each task flow f_i, while in Step 4), task flows are sorted in descending order based on their route hop count, prioritizing those with longer paths first. Step 8) guarantees the attainment of the routing and scheduling arrangement that minimizes delay for each task flow f_i.

Algorithm 1:Heuristic scheduling
Input:Traffic set F, Network topology G
Output:Gate Control List GCL, $Routes$
1) for f_i in F do
2) get_all_paths(f_i);

3) end for
4) sort(F , key=rout_num(path(f_i)));
5) for f_i in F do
6) for $path$ in $paths(f_i)$ do
7) find_vacant_time($paths, f_i$);
8) if delay($path$)=minimize then
9) return assigned($path$);
10) end if
11) end for
12) end for
13) if succ then
14) return $GCL, Routes$;
15) else return fail;
16) end if

4.2. Shorter Path Priority Strategy

In the heuristic scheduling algorithm mentioned earlier, traversing all available paths of a task flow to obtain routing and scheduling results incurs a significant time cost. Hence, a strategy prioritizing shorter paths is proposed to optimize the algorithm's search strategy and enhance heuristic algorithm efficiency.

The shorter path priority strategy first standardizes the solution space of Algorithm 1. For steps 1) to 3) of Algorithm 1, the shorter path priority strategy sorts all paths of the task flow based on the number of route hops to standardize the solution space. The specific steps are shown in Algorithm 2.

Algorithm 2: Shorter Path Priority Strategy Section 1
Input:Traffic set F , Network topology G
Output: $sorted\_paths(f_i)$
1) for f_i in F do
2) get_all_paths(f_i);
3) sort($paths(f_i)$,key=rout_num($path$));
4) return $sorted\_paths(f_i)$
5) end for

The shorter priority strategy further improves the search strategy of Algorithm 1. For steps 5) to 12) of Algorithm 1, the shorter path priority strategy prioritizes solving paths with shorter task flows based on the solution space specified in Algorithm 2. If successful, subsequent path solving is stopped directly, saving time and improving the efficiency of the algorithm. The specific steps are shown in Algorithm 3.

Algorithm 3:Shorter Path Priority Strategy Section 2
Input:Traffic set F , $sorted\_paths(f_i)$
Output:Gate Control List GCL
5) for f_i in F do

6) for $path$ in $sorted\_paths(f_i)$ do
7) find_vacant_time($paths, f_i$);
8) if succ then break;
9) end if
10) end for
11) end for

5. Performance Evaluation

5.1. Computing Platform

The hardware platform CPU used for the experiment is Intel®Core™ i7-8750H CPU @ 2.20GHz ×8, operating system is Ubuntu 18.04.6 LTS, Python version is 3.10. For algorithms using ILP, the solver and version used is Gurobi 10.0.3.

5.2. Experimental Setup

The network topology in the dataset adopts four commonly used topology structures, namely linear topology, ring topology, tree topology, and mesh topology[15]. The number of bridges in the topology ranges from 8 to 78, and the scale of task flow ranges from 10 to 220. In the dataset configuration, the topology scale is structured in cycles of 32 scenes, with complexity incrementing progressively within each cycle. Simultaneously, the scale of the task flow increases linearly every 32 scenarios, as depicted in Equation (1), where the node set V, task flow set F, and scenario sequence index $index$ are defined.

$$\begin{cases} V = \{1, 2, , , n\}, n = [(index \bmod 32) / 4] \times 10 + 8 \\ F = \{f_1, f_2, , , f_i\}, i = [index / 32] \times 30 + 10 \end{cases} \tag{1}$$

The improved heuristic scheduling algorithm is compared with the HLS algorithm from article [8], the PCG algorithm from article [13], the RST algorithm from article [14], the LSP algorithm from article [11], and the SNS algorithm from article [10]. The comparison is based on the time required to solve problem scenarios, assessing the efficiency of these algorithms.

6. Experimental Results

6.1. Rate of Successful Solving

Table 1: Success rate of solving

Algorithm	Number of task	Success count	Success rate
SPPS	200	171	85.5%
HLS	200	169	84.5%
PCG	200	170	85.0%
RST	200	170	85.0%
LSP	200	114	57.0%
SNS	200	170	85.0%

Each algorithm tackled scenarios of varying scales and topologies in the experimental setup. Table 1 displays the success rates of all algorithms. The enhanced algorithm SPPS(Shorter Path Priority

Strategy) demonstrates a higher success rate compared to its original version and outperforms other algorithms.

6.2. Solving Efficiency

To determine the performance improvement rate of the enhanced algorithm, compare its solving time with that of other algorithms, as indicated in Equation (2), where $rate, t_1$ and t_2 denote the improvement rate, the solving time of other algorithms, and the solving time of the enhanced algorithm.

$$rate = (t_1 - t_2) / t_1 \tag{2}$$

Figure 1 illustrates the comparison of solving efficiency between the original algorithm and the improved heuristic scheduling algorithm. The enhanced algorithm demonstrates comparable performance to the HLS algorithm in scenarios with simpler topologies, while notably enhancing solving efficiency in more complex topology scenarios.

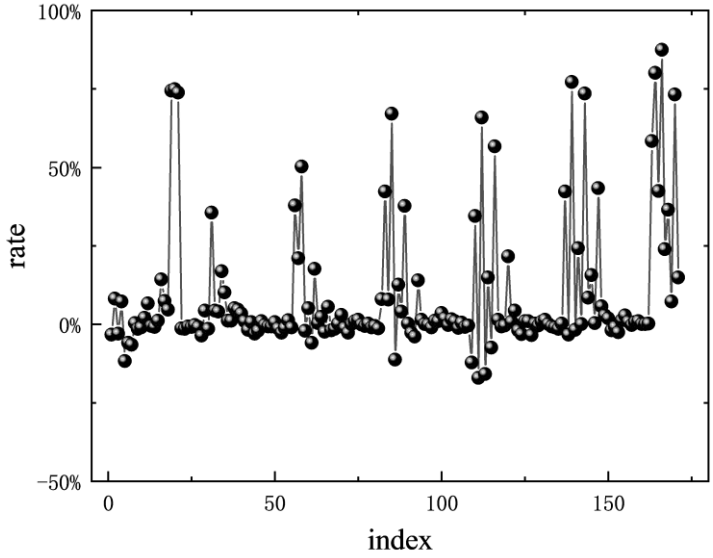

Figure 1: Comparison with HLS algorithm

Figure 2 illustrates the comparison of solving efficiency with the PCG algorithm. The improved heuristic scheduling algorithm shows marginal improvement in solving efficiency in simpler topology scenarios but exhibits significant enhancement in scenarios with more complex topologies. Furthermore, as the task flow size increases, the efficiency improvement rate of the algorithm gradually rises.

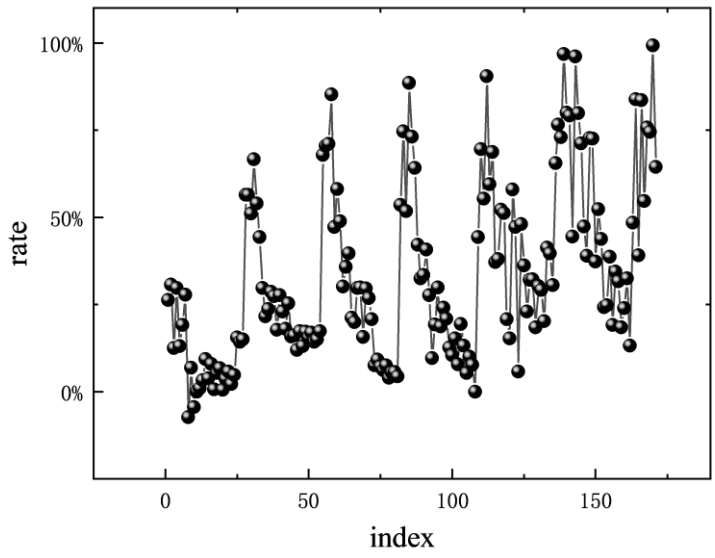

Figure 2: Comparison with PCG algorithm

Figure 3 illustrates the comparison of solving efficiency with the RST algorithm. The enhancement rate of solving efficiency for the improved heuristic scheduling algorithm rises with the task flow size, and its efficiency improvement is notably more pronounced in scenarios with simpler topologies.

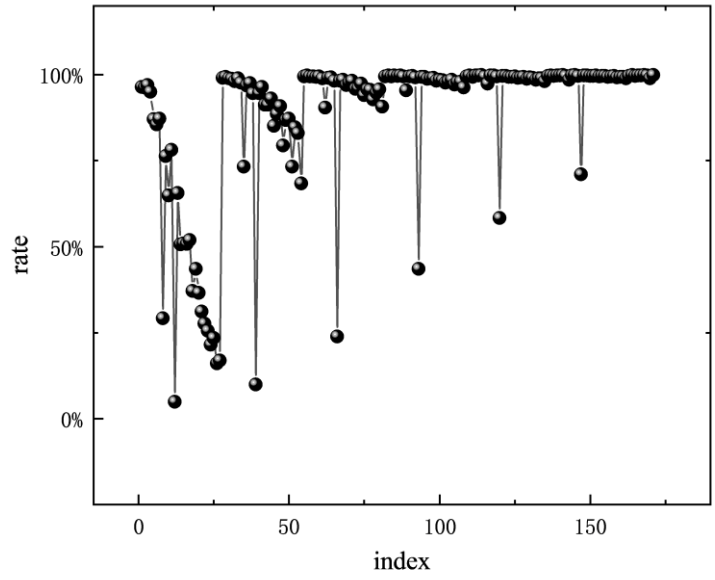

Figure 3: Comparison with RST algorithm

Figure 4 shows the comparison of solving efficiency with LSP algorithm. The improved heuristic algorithm has better performance in solving efficiency in topologies of various scales, and its advantages gradually expand as the topology size increases.

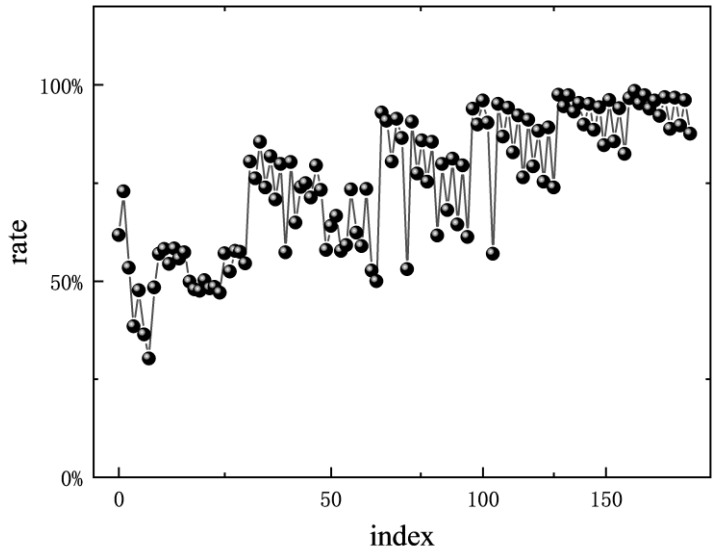

Figure 4: Comparison with LSP algorithm

Figure 5 shows the comparison of solving efficiency with the SNS algorithm. The improved heuristic algorithm has a stable improvement in solving efficiency in topologies of various scales, but lags behind in larger topologies and flow scales.

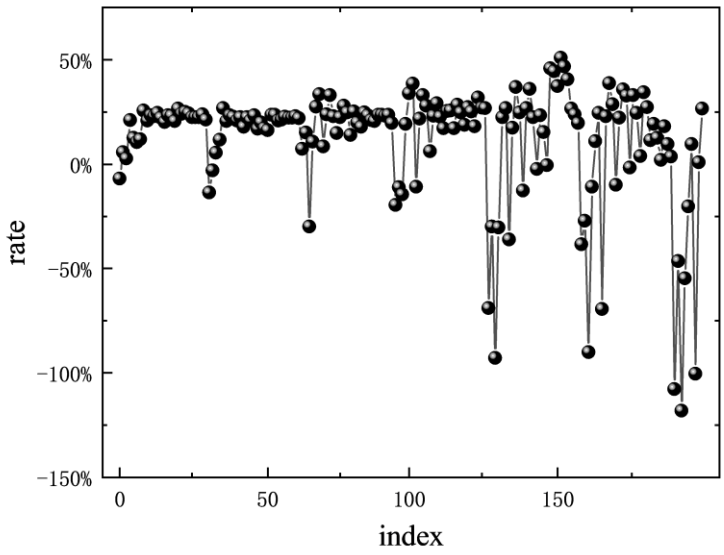

Figure 5: Comparison with SNS algorithm

In terms of total solving time cost, the improved heuristic scheduling algorithm has significantly improved solving efficiency, with a 17% improvement compared to the original HLS algorithm, a 39% improvement compared to the PCG algorithm, a 95% improvement compared to the RST algorithm, a 74% improvement compared to the LSP algorithm, and a 12% improvement compared to the SNS algorithm. Figure 6 shows the overall solving efficiency of each algorithm.

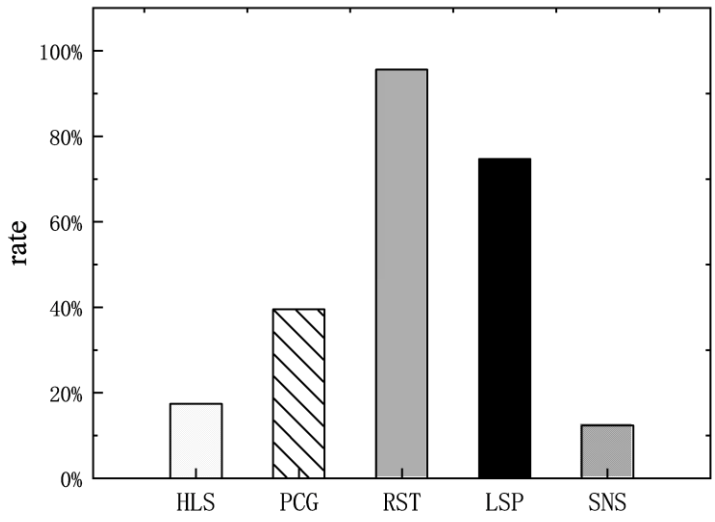

Figure 6: Overall performance comparison

7. Conclusions

In response to the problem of reducing the time cost of heuristic scheduling solutions, the search strategy of heuristic scheduling algorithms has been improved by proposing a shorter path priority strategy, standardizing the solution space, and optimizing the search strategy. The experimental results show that compared with the original algorithm, the solving efficiency is improved by 17%. Compared with the other four algorithms, the solving efficiency is improved by more than 12%, and it is more suitable for scenarios with complex topology and task flow. It can respond faster to scenarios with changes in topology or task flow.

References

[1] Z. Tong, F. Jiaqi, M. Yanying, Q. Siyuan, and R. Fengyuan, "Survey on Traffic Scheduling in Time-Sensitive Networking," Jisuanji Yanjiu yu Fazhan, vol. 59, no. 4, pp. 747–764, 2022.

[2] T. Liu, D. He, Z. Jin, S. Shan, Y. Chen, and Q. Chen, "Research on flow scheduling of train communication based on time-sensitive network," Simul. Model. Pract. Theory, vol. 130, p. 102859, Jan. 2024, doi: 10.1016/j.simpat.2023.102859.

[3] 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, doi: 10.1109/ACCESS.2023.3286370.

[4] Y. Xu and J. Huang, "A Survey on Time-Sensitive Networking Standards and Applications for Intelligent Driving," Processes, vol. 11, no. 7, Art. no. 7, Jul. 2023, doi: 10.3390/pr11072211.

[5] J. Jiang, Y. Li, X. Zhang, M. Yu, C. D. Lee, and S. H. Hong, "Assessing the traffic scheduling method for time-sensitive networking (TSN) by practical implementation," Journal of Industrial Information Integration, vol. 33, p. 100464, Jun. 2023, doi: 10.1016/j.jii.2023.100464.

[6] N. G. Nayak, F. Dürr, and K. Rothermel, "Routing algorithms for IEEE802.1Qbv networks," ACM SIGBED Review, vol. 15, no. 3, pp. 13–18, Aug. 2018, doi: 10.1145/3267419.3267421.

[7] D. Hellmanns, L. Haug, M. Hildebrand, F. Dürr, S. Kehrer, and R. Hummen, "How to Optimize Joint Routing and Scheduling Models for TSN Using Integer Linear Programming," Apr. 2021. doi: 10.1145/3453417.3453421.

[8] M. Pahlevan, N. Tabassam, and R. Obermaisser, "Heuristic list scheduler for time triggered traffic in time sensitive networks," SIGBED Rev., vol. 16, no. 1, pp. 15–20, Feb. 2019, doi: 10.1145/3314206.3314208.

[9] X. Jin et al., "Real-Time Scheduling of Massive Data in Time Sensitive Networks With a Limited Number of Schedule Entries," IEEE Access, vol. 8, pp. 6751–6767, 2020, doi: 10.1109/ACCESS.2020.2964690.

[10] Y. Zhang, Q. Xu, S. Wang, Y. Chen, L. Xu, and C. Chen, "Scalable No-wait Scheduling with Flow-aware Model Conversion in Time-Sensitive Networking," in 2022 IEEE GLOBAL COMMUNICATIONS CONFERENCE (GLOBECOM 2022), in IEEE Global Communications Conference. New York: IEEE, 2022, pp. 413–418. doi: 10.1109/GLOBECOM48099.2022.10001004.

[11] M. Vlk, K. Brejchová, Z. Hanzálek, and S. Tang, "Large-scale periodic scheduling in time-sensitive networks," Computers & Operations Research, vol. 137, p. 105512, Jan. 2022, doi: 10.1016/j.cor.2021.105512.
[12] D. Bujosa, M. Ashjaei, A. V. Papadopoulos, T. Nolte, and J. Proenza, "HERMES: Heuristic Multi-queue Scheduler for TSN Time-Triggered Traffic with Zero Reception Jitter Capabilities," in Proceedings of the 30th International Conference on Real-Time Networks and Systems, Paris France: ACM, Jun. 2022, pp. 70–80. doi: 10.1145/3534879.3534906.
[13] J. Falk, F. Durr, and K. Rothermel, "Time-Triggered Traffic Planning for Data Networks with Conflict Graphs," in 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Sydney, NSW, Australia: IEEE, Apr. 2020, pp. 124–136. doi: 10.1109/RTAS48715.2020.00-12.
[14] A. A. Atallah, G. B. Hamad, and O. A. Mohamed, "Routing and Scheduling of Time-Triggered Traffic in Time-Sensitive Networks," Ieee Trans. Ind. Informat., vol. 16, no. 7, pp. 4525–4534, Jul. 2020, doi: 10.1109/TII.2019.2950887.
[15] C. Xue, T. Zhang, Y. Zhou, and S. Han, "Real-Time Scheduling for Time-Sensitive Networking: A Systematic Review and Experimental Study," arXiv.org, 2023, Accessed: Nov. 03, 2023. [Online]. Available: https://www.semanticscholar.org/paper/Real-Time-Scheduling-for-Time-Sensitive-Networking%3A-Xue-Zhang/450704e2188a 2138 0e46c89 cbc1c26de81581b70