

Article

Redundant Topology Design Method for Schedule Traffic Transmission Based on IEEE802.1CB Standard: Optimizing NW-PSP Scheduling Performance

Jun-Young Han ¹, Young-Hoo Jeon ², Hyeong-Jun Kim ^{3,*} and Suk Lee ^{1,*}

¹ School of Mechanical Engineering, Pusan National University, Busan 46241, Republic of Korea; jyhan@pusan.ac.kr

² CV Electronics Control Test Team, Hyundai Motor Company, Hwaseong 18280, Republic of Korea; yhjeon@hyundai.com

³ Department of Future Automotive Engineering, Gyeongsang National University, Jinju 52828, Republic of Korea

* Correspondence: hj.kim@gnu.ac.kr (H.-J.K.); slee@pnu.edu (S.L.); Tel.: +82-055-772-3642 (H.-J.K.); +82-051-510-3091 (S.L.)

Abstract: This study proposes a method for designing redundant topologies in time-sensitive networking to transmit scheduled traffic (ST) while considering fault tolerance. In general, redundant topologies enhance network fault tolerance and are used in safety-critical applications that demand high reliability. These applications place a premium on real-time network performance, necessitating the consideration of the quality of service for time-critical traffic during the redundant topology design process. The primary objective of this study is to optimize the flowspan, which pertained to the scheduling performance of the no-wait packet scheduling problem during the redundant topology design phase. Specifically, we introduced a methodology for designing topologies tailored to transmit ST using the IEEE802.1CB mechanism. Comparisons with existing topology design approaches revealed the ability of the proposed method to enhance network fault tolerance and yield a topology that reduced flowspan of ST, thereby freeing up bandwidth for time-critical traffic.

Keywords: time sensitive networking; redundant topology; topology design; scheduled traffic; no-wait packet scheduling problem

Citation: Han, J.-Y.; Jeon, Y.-H.; Kim, H.-J.; Lee, S. Redundant Topology Design Method for Schedule Traffic Transmission Based on IEEE802.1CB Standard: Optimizing NW-PSP Scheduling Performance. *Electronics* **2024**, *13*, 88. <https://doi.org/10.3390/electronics13010088>

Academic Editor: Pietro Manzoni

Received: 12 October 2023

Revised: 21 December 2023

Accepted: 22 December 2023

Published: 24 December 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Recently, technologically advanced systems such as autonomous vehicles and factories have been invented [1–3]. Safety-critical applications associated with these systems require high levels of network reliability in real-time to ensure stability [4–6]. However, if a link or bridge fails in an application network, the time required to retransmit the lost frames over an alternate path can be excessively long. The time-sensitive networking (TSN) standard, known as IEEE802.1CB or frame replication and elimination for reliability (FRER), proposes replicating frames and transmits them over multiple paths from the sending node to the destination node [7,8]. This standard enables the network to achieve reliable features.

To apply the FRER standard, a redundant topology that can transmit frames over multiple paths from the origin to destination node must be designed. A redundant topology eliminates the transmission problems caused by network failures. Thus, the loss or delay of frames owing to faults, physical damage, etc., during the communication process can be prevented [9,10]. However, for safety-critical applications such as autonomous vehicles and automation facilities, frames within a redundant topology are transmitted with a simple structure to reduce cost and design complexity [11–13]. Such a simple path

networks can decrease the reliability of safety-critical applications owing to bottlenecks [14,15].

In addition, real-time monitoring is an important consideration [5,16,17]. The IEEE802.1Qbv TSN standard proposes transmitting time-critical traffic as scheduled traffic (ST) using a time-aware shaper (TAS) [18]. Therefore, transmitting ST using a TAS requires the scheduling of ST across the entire network. Consequently, various scheduling methods are used to minimize transmission delay, jitter, and free bandwidth [19–22].

To achieve the primary objectives of TSN, which are real-time operation and reliability, redundant topology designs and traffic scheduling must be considered together. Therefore, a method to design a redundant topology that considers the characteristics of TSN is required [23,24]. Garvrlut et al. [23] proposed a method for designing a topology with redundant paths to carry urgency-based scheduler traffic in a TSN. Three optimization methods were proposed to minimize the topology architecture cost. Atallah et al. [24] designed a topology with redundant paths for transmitting ST.

These studies designed redundancy by considering the characteristics of the traffic used in a TSN. In particular, they aimed to minimize the architectural cost of the constructed topology. However, they did not consider the scheduling performance of the ST, which is important for TSN. According to Nayak et al. [25], scheduling performance changes depending on the transmission path of the ST. Therefore, a redundant topology that considers the performance of the ST must be designed to ensure the free bandwidth of TSN networks.

This study proposes a method for designing a redundant topology for transmitting critical ST based on the IEEE802.1CB standard. The proposed method uses Yen's K -shortest-path (KSP) algorithm to design the topology by generating k -paths and selecting the path that is likely to improve the scheduling performance among the k -paths. In particular, the topology was designed to minimize the flowspan, which is the performance of no-wait packet scheduling problem (NW-PSP) scheduling among the various scheduling methods [22]. Furthermore, the proposed method's performance was verified through comparisons with two topology design methods in a simulation environment.

The rest of this study is organized as follows. Section 2 briefly describes the topology planning method using Yen's KSP and the NW-PSP's performance. Section 3 describes the design of a redundant topology for transmitting ST, considering the architectural cost and the NW-PSP's performance. Section 4 evaluates the performance of three methods: the proposed approach, a method considering only architectural cost, and a method incorporating traffic distribution into the comparison. Finally, Section 5 discusses this study's results and limitations.

2. Background

2.1. Overview of TSN-Based Topology Design

In TSN networks, ensuring fault tolerance is crucial. One technique used for this purpose is the IEEE802.1CB FRER. FRER involves replicating frames and identifying and eliminating duplicate frames, thereby allowing frame transmission via redundant paths. Multiple paths are required to achieve this, which necessitates a redundant network topology.

A common approach for designing a redundant topology involves rendering the entire topology redundant. However, this method results in unnecessary paths and increased topology costs and may not be feasible owing to physical limitations. Therefore, an alternative approach involves duplicating only the specific paths required to transmit important messages.

The method for creating a redundant topology by duplicating specific paths relies on routing algorithms. This process configures the network links to be fully connected, performs routing, and activates links in the topology model. In a typical design process, the

network cost is calculated as the sum of the topology and hop costs. Subsequently, the topology is designed using routing paths that minimize the overall network cost.

However, this approach overlooks the scheduling performance, which is a primary goal in TSN. Figure 1 illustrates the paths routed using different methods in the same topology, along with their corresponding schedules. In (a), the path is routed using the shortest-path algorithm, whereas in (b), the traffic is distributed. Furthermore, (c) and (d) show the schedules for the shortest-path algorithm and distributed traffic, respectively. The ‘flowspan’ in each schedule table suggests a performance metric for NW-PSP, indicating the time required for a frame’s transmission to complete within the schedule duration.

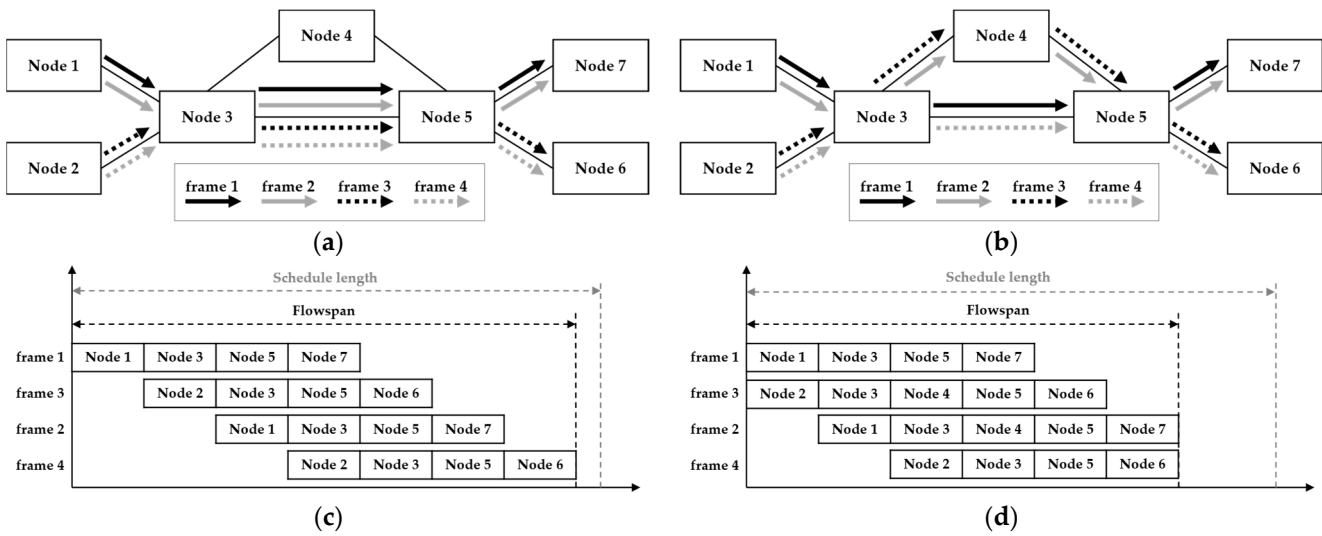


Figure 1. Comparison of NW-PSP’s performance schedules based on routing methods: (a) routes routed with the shortest path algorithm; (b) routes routed with the traffic distribution method; (c) schedules for routes routed with the shortest path algorithm; and (d) schedules for routes routed with the traffic distribution method.

It is evident that the frame transmission delay is low in the schedule generated using the shortest-path algorithm. Distributing traffic increases frame delay while reducing the flowspan. Reducing the flowspan facilitates the scheduling of more frames and minimizes bandwidth waste by reducing the schedule space within the schedule cycle. This highlights the potential drawback of considering only the shortest path, particularly when designing a topology. To minimize bandwidth wastage, the scheduling performance must be considered during the routing stage.

2.2. Overview of Yen’s KSP Algorithm

The proposed topology design method employs Yen’s K-shortest paths (KSP), which is based on Dijkstra’s algorithm for graph modeling to discover diverse K-shortest paths between two nodes. Therefore, to grasp Yen’s KSP thoroughly, it is essential to first understand Dijkstra’s algorithm. Consequently, in this section, we will provide explanations for both algorithms. We will begin by elucidating Dijkstra’s algorithm and subsequently delve into the specifics of Yen’s KSP. This sequential approach aims to offer a clear understanding of the concepts and functionalities of each algorithm.

Dijkstra’s algorithm, named after the computer scientist Edsger Dijkstra, is a well-known method for determining the shortest paths between nodes in a weighted graph. The algorithm starts by selecting an initial node and then iteratively updating the shortest path values to adjacent nodes until the destination node is reached. The steps of the algorithm can be summarized as follows:

Firstly, the algorithm begins by selecting the starting node and setting its shortest path value to 0, while marking all other nodes with an initial infinite distance.

Next, it iterates through each adjacent node, updating the shortest path if a shorter path is discovered through the current node.

The algorithm then selects the node with the shortest path value among the unprocessed nodes.

This process is repeated iteratively until the destination node is reached or all nodes are processed.

Dijkstra's algorithm is guaranteed to find the shortest path from the source node to all other nodes in non-negative weighted graphs, making it widely applicable in areas such as network routing and optimization.

Yen's KSP algorithm is an extension of Dijkstra's methodology, specifically tailored to find multiple diverse shortest paths between two nodes. It introduces diversity by iteratively blocking edges along the initially identified shortest path. The steps are outlined as follows:

Firstly, employ Dijkstra's algorithm to determine the initial shortest path from the source to the destination node.

Subsequently, explore alternative paths by systematically blocking one edge at a time along the initial shortest path. This process generates diverse paths, each distinguished by the obstruction of a single edge.

Finally, if K-shortest paths are required, perform the above steps K times. In each iteration, restore the previously blocked edge and obstruct the next edge along the initial path, producing a set of K distinct paths.

Yen's KSP algorithm is crucial in network design, providing fault tolerance and load distribution by creating diverse paths. As it iterates, it discovers paths different from the initially identified shortest path, adapting to various network conditions.

In summary, Dijkstra's algorithm excels in finding single shortest paths, while Yen's KSP algorithm extends this capability to discover multiple diverse paths, enhancing adaptability and resilience in network design and optimization. Both algorithms showcase the power of graph theory in solving real-world problems.

3. Topology Design Method

Figure 2 shows the topology design method proposed here. This method considers the initial topology connection and ST frame information as the input, routes the transmission paths of the ST frame individually, updates the topology, and generates the topology structure and transmission paths of the ST frames as the output.

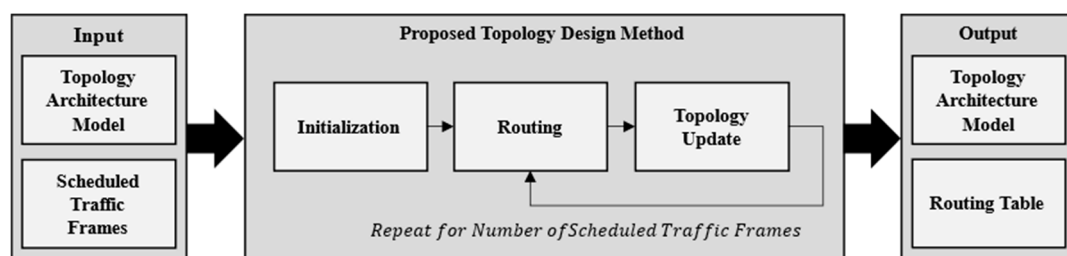


Figure 2. Overview of the proposed topology design method.

In particular, the routing phase used Yen's KSP to generate multiple paths and select the path that minimized the flowspan. The path was selected by assigning a schedule to the schedule table and selecting the path with the shortest transmission completion time. In the case of multiple paths with the same completion time, the path with the least amount of traffic was chosen.

This method minimized bandwidth wastage in ST transmissions by designing an efficient redundant topology. Table 1 presents the parameters used in the proposed method.

Table 1. Symbol descriptions in the algorithm pseudocode.

Symbol	Description
G_{app}	Constructed as a graph model, it is composed of the weight between nodes.
G_{arch}	Constructed as a graph model, it consists of connectivity information among nodes.
$Frame_{ST}$	A set of ST frames (<i>frame</i>) to be delivered over the network.
<i>frame</i>	Comprising information about the sending and receiving nodes, importance, transmission period, and frame length, a single ST frame consists of essential details.
<i>important</i>	A parameter indicating the importance of ST traffic with a value of '0' or '1'. When this value is '0', an ST frame is transmitted via a single path, and when it is '1', it is transmitted via redundant paths.
R	R is a routing table that stores the transmission paths.
r	A path, denoted as r , is composed of path configuration information. In the case of a redundant path, the number of paths is '2', and the information for both paths is included. In the case of a single path, the number of paths is '1', and the information for the single path is included.
S	S is a schedule table that stores the transmission schedule of ST frames.
s	s represents the transmission schedule for a single ST frame, comprising the transmission start time and completion time at each node along path r .

In this study, we used the *ScheduleTableCheck* function to place schedules in the schedule table during routing. It placed the schedule of a frame in the table based on the NW-PSP and output the schedule of the frame. Therefore, the schedule was placed at the fastest transmission location without latency at each node. In addition, the *ScheduleTableCheck* function output the traffic of the most-bottlenecked transmission port as the maximum traffic when a schedule of redundant paths was placed.

Algorithm 1 shows the pseudocode for the topology planning method. The proposed method accepted G_{arch} and $Frame_{ST}$ as input. It planned the topology by sequentially creating paths for transmitting $Frame_{ST}$ and activating the connections of G_{arch} . The output of this process was G_{arch} and R . The topology planning method comprised initialization, path selection, and topology update phases. The topology was planned by repeating the path selection and topology update steps as many times as the number of frames in $Frame_{ST}$.

Algorithm 1. Topology Design Algorithm

```

1:  function TopologyDesign( $G_{arch}$ ,  $Frame_{ST}$ )
2:     $G_{app} \leftarrow \text{InitTopology}(G_{arch})$  # step 1. initialization starting Point
3:     $R \leftarrow \emptyset, S \leftarrow \emptyset$ 
4:     $Frame_{ST} \leftarrow \text{SortPeriodAscending}(Frame_{ST})$ 
5:     $Frame_{ST} \leftarrow \text{SortLengthDescending}(Frame_{ST})$  # step 1. ending Point
6:    for frame in  $Frame_{ST}$  do
7:      if frame.importance is 0 then # step 2. routing starting point
8:         $s, r \leftarrow \text{MakeSinglePath}(\text{frame}, G_{app}, S)$ 

```

```

9:      else if frame.importance is 1 then
10:           $s, r \leftarrow \text{MakeMultiPath}(\text{frame}, G_{app}, S)$ 
11:      end if # step 2. ending point
12:           $R \leftarrow \text{UpdateRoutingTable}(R, r)$  #
          step 3. topology update starting point
13:           $G_{arch}, G_{app} \leftarrow \text{UpdateTopology}(G_{arch}, G_{app}, R)$ 
14:           $S \leftarrow \text{UpdateScheduleTable}(S, s)$  # step 3. ending point
15:  end for
16:  return  $G_{arch}, R$ 

```

The initialization phase initialized the G_{app} to be used for routing and aligned the Frame_{ST} . G_{app} was initialized by calculating the topology cost based on the connection status of G_{arch} . G_{arch} recommended enabling connectivity between the end nodes and switched to facilitate routing. Frame_{ST} was sorted in ascending order by transmission period and then in descending order by the length of the frame. This sorting method ensured that frames with longer lengths were used first, and those with shorter transmission cycles were used instead if they had the same frame length. Consequently, the gaps between the schedules were minimized when they were placed sequentially.

The path-selection phase selected a path for a single ST-frame. If the *importance* of a *frame* was zero, a single path was created; if the *importance* was one, a redundant path was created. Path selection used a schedule table to generate a transmission schedule for a selected path. The topology update step used the generated routes to update the routing table and topology model. In addition, the schedule table was updated using the generated schedule.

In the presented approach, the time complexity of the algorithm responsible for generating a single path is lower than that of the algorithm generating multiple paths. Consequently, the best-case time complexity is expressed as $O(N \cdot F_S)$, where N denotes the number of ST frames, and F_S is the time complexity corresponding to the *MakeSinglePath* function. Conversely, the worst-case time complexity is characterized as $O(N \cdot F_M)$, where F_M corresponds to the time complexity of the *MakeMultiPath* function.

Algorithm 2 shows the pseudocode for the *MakeSinglePath* function used to create a single path. Figure 3's flowchart visually depicts the process. It used Yen's KSP to generate k -paths and selected one of the k -paths to create the path. Yen's KSP routed k -paths by considering the topology cost and hopping of paths. Using route R obtained from routing, we generated a single transmission schedule for the frames and calculated the maximum traffic for each path. Schedule and traffic were stored in *result* as a set of paths. The maximum traffic on a path was that of the port with the highest traffic among the transmission ports of the switches comprising the path.

Algorithm 2. Single Path Routing Algorithm

```

1:  function MakeSinglePath(frame,  $G_{app}, S$ )
2:       $result \leftarrow \emptyset$ 
3:       $R \leftarrow \text{YenKsp}(\text{frame}, G_{app}, k)$ 
4:      for  $r$  to  $R$  do

```

```

5:       $s, traffic \leftarrow ScheduleTableCheck(S, frame, r)$ 
6:       $result \leftarrow res.append(\{r, s, traffic\})$ 
7:      end
8:       $result \leftarrow SortAscendingTraffic(res)$ 
9:       $result \leftarrow SortAscendingEndtime(res)$ 
10:     return  $result[0].s, result[0].r$ 

```

Then, the *result* was sorted in ascending order by *traffic* and in ascending order by the time the transmission was completed in the transmission schedule of the first cycle. The routes and schedules of the top dataset were output in the *result* after sorting. The path selection method selected the path with the fastest transfer completion time in the first cycle. If multiple paths completed a transfer simultaneously, the path with the lowest traffic level was selected. This path selection method increased the schedule density and distributed traffic when creating a single path.

The time complexity of the algorithm for generating a single path varies with the network size and the number of ST frames. For a small network size, bottlenecks intensify, resulting in an increased scheduling time complexity. In this scenario, the worst-case time complexity is denoted as $O(K \cdot N \cdot V)$. For larger network sizes, the worst-case time complexity is determined by Yen's KSP algorithm, resulting in $O(K \cdot V \cdot (E + V \cdot \log V))$. Here, K denotes the number of paths generated using Yen's KSP, N is the quantity of ST frames, V denotes the total number of nodes in the topology, and E denotes the total number of links in the network.

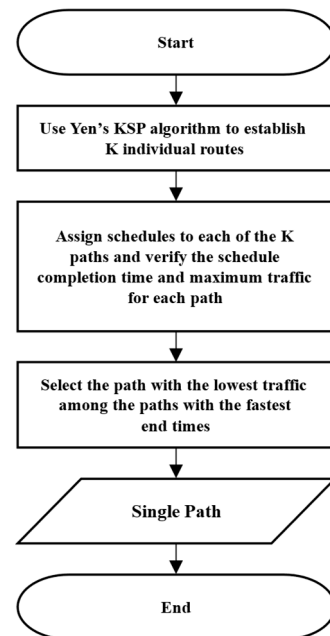


Figure 3. Flowchart for a function that generates a single path.

Algorithm 3 shows the pseudocode for the *MakeMultiPath* function, which created redundant paths. Figure 4's flowchart visually depicts the process. Here, k -paths were created using Yen's KSP. Redundant paths were created based on the paths in created path R_{base} . Subsequently, the path with the fastest transmission completion time in the first cycle of k -paths was selected. The selected path and transmission schedule for the selected path were the output.

Algorithm 3. Multipath Routing Algorithm

```

1:  function MakeMultipath( $frame, G_{app}, S$ )
2:     $result \leftarrow \emptyset$ 
3:     $R_{base} \leftarrow YenKsp(frame, G_{app}, k)$ 
4:    for  $r_{base}$  to  $R_{base}$  do
5:       $G_{app}^* \leftarrow RemoveVertex(G_{app}, r_{base})$ 
6:       $R_{sub} \leftarrow YenKsp(frame, G_{app}^*, k)$ 
7:       $result_{sub} \leftarrow \emptyset$ 
8:      for  $r_{sub}$  to  $R_{sub}$  do
9:         $r \leftarrow r_{base} \cup r_{sub}$ 
10:        $s, traffic \leftarrow ScheduleTableCheck(S, frame, r)$ 
11:        $result_{sub}.append(\{r, s, traffic\})$ 
12:     end
13:      $subres \leftarrow SortAscendingEndtime(result_{sub})$ 
14:      $subres \leftarrow SortAscendingTraffic(result_{sub})$ 
15:      $result.append(result_{sub}[0])$ 
16:   end
17:    $result \leftarrow SortAscendingFlowEndtime(res)$ 
18:   return  $result[0].s, result[0].r$ 

```

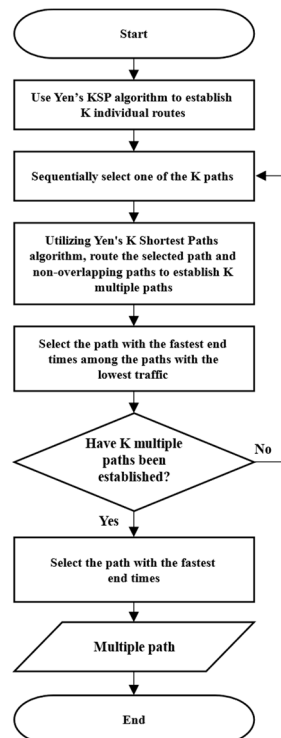


Figure 4. Flowchart for a function that generates a redundant path.

The create redundant path task created a path independent of the base path r_{base} by removing nodes except switches that were not directly connected to the sending and receiving nodes of the base path. Consequently, it generated k -redundant paths using Yen's KSP algorithm. The generated backup paths were combined with the reference path to form a redundant path. The schedule and maximum traffic on the redundant path were then checked and saved in $result_{sub}$.

Next, one of the k -redundant-paths was selected based on the base path r_{base} . The path with the lowest maximum traffic was selected. If there were paths with the same maximum traffic, the path with the fastest transmission completion time in the first cycle was selected. The dataset containing the selected path was stored in $result$.

Next, a redundant path of k -paths in R_{base} was generated and the path that minimized the flowspan was selected. To improve the schedule density, we chose the path with the fastest transfer completion time in the first cycle among the redundant paths. This path selection method considers schedule density and traffic distribution when configuring redundant paths. Therefore, it was likely that the flowspan would be reduced when running the schedule on the completed topology.

The time complexity of the algorithm for generating duplicate paths varies based on the network size and the number of ST frames. In scenarios involving a small network size, bottlenecks become more pronounced, resulting in an increased scheduling time complexity. Specifically, the algorithm's time complexity in this context is denoted as $O(K \cdot N)$. Conversely, for larger network sizes, the worst-case time complexity is derived by multiplying the execution count with the time complexity of Yen's KSP, yielding $O((K + 1) \cdot K \cdot V \cdot (E + V \log V))$. In situations where the network size is small, the worst-case time complexity is calculated as $O(2 \cdot N \cdot V \cdot K^2)$. Here, K denotes the number of paths generated using Yen's KSP, N denotes the number of ST frames, V denotes the total number of nodes in the topology, and E denotes the total number of links in the network.

In conclusion, the proposed method demonstrates varying worst-case time complexity based on the network size and the number of ST frames. For a small network size, the worst-case complexity is determined as $O(N^2 \cdot V \cdot K^2)$. Conversely, for larger network sizes, the worst-case complexity is expressed as $O(N \cdot ((K + 1) \cdot K \cdot V \cdot (E + V \log V)))$. Consequently, in the design of a topology for transmitting a substantial number of ST frames in a small network, there exists a relatively high likelihood of engaging in a significant number of computations.

4. Experiments

In this study, we evaluated the performance of the proposed topology design method by comparing it with two comparators in a simulation environment. The first comparator, Yen's KSP-based topology design (YKTD) method, was based on Atallah et al. [24]. The YKTD method employed Yen's KSP, which used the ratio of hop count and topology cost as the network cost and checked the schedule placement. The second comparator, the flowspan-optimized routing-based topology design (FORTD) method, was derived from Nayak et al. [25], and applied a routing algorithm to minimize the flowspan while enhancing the YKTD method. The FORTD method achieved traffic distribution by calculating the network cost as the sum of the hop count, topology cost, and maximum traffic on the path.

YKTD, FORTD, and the proposed method all used Yen's KSP for topology design. They calculated the network cost $cost_{network}$ using the same approach. The formula for calculating the network cost was:

$$cost_{network} = cost_{hop} + cost_{topology}. \quad (1)$$

The network cost was calculated as the sum of $cost_{hop}$ and $cost_{topology}$. $cost_{hop}$ represented the number of hops in the path, while $cost_{topology}$ was the expense incurred

when activating the elements constituting the path. $cost_{topology}$ was computed as the sum of costs, which was based on the number of activated links and the number of ports on switches.

To evaluate the proposed topology design in various scenarios, we configured 20 different network environments and generated 50 random scenarios for each environment, resulting in 1000 validation scenarios. Table 2 presents the network configuration environments, including the number of end nodes and ST frames transmitted over the network. The maximum number of switch nodes was set to 40 for all network environments.

Table 2. Number of ST frames and nodes configuring the network environment.

Number of End Nodes		Number of ST Frames				
10	100	300	500	700	900	
20	200	400	600	800	1000	
30	300	500	700	900	1100	
40	400	600	800	1000	1200	

Random scenarios involved generating ST frames under various conditions. Table 3 outlines the ST frame generation conditions, which involved randomizing the frame periods between 1 ms and 10 ms, sizes between 200 bytes and 1500 bytes, and designating 50% of all ST frames as critical frames to ensure that 50% of the frames used redundant paths. In particular, during scenario generation, all end nodes were configured for use to prevent excessive bottlenecks from occurring.

Table 3. ST frame generation conditions.

Parameters		Values					
Redundant Path Utilization		50					
Ratio (%)							
Period (ms)		10	5	2.5	2	1.25	1
Length (bytes)	Mix.	200					
	Max.	1500					

The performance was validated on a Windows 10 computer equipped with an Intel i5-7600 CPU and 8 GB RAM. We used Python version 3.7.5 to implement and evaluate different topology design methods. We assessed these methods' performance in terms of flowspan, traffic distribution effect, topology cost, and design time. Following the topology design, we performed NW-PSP scheduling to compare the flowspan performance.

Figure 5 shows the reduction ratios of the flowspan and maximum traffic reduction for the proposed and FORTD methods compared with the YKTD method across the entire set of scenarios. For clarity, each graph sorted the metrics in descending order. The y -axis represented the number of scenarios, and the x -axis represented the performance metrics. Graph (a) shows the reduction ratio of the flowspan, and (b) shows the maximum traffic reduction.

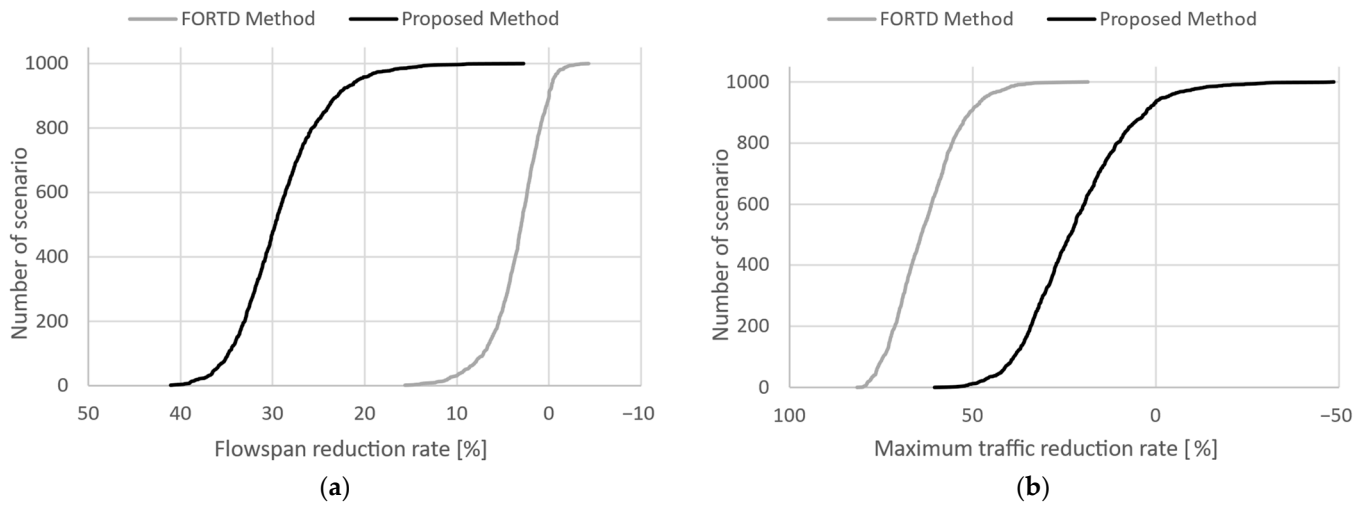


Figure 5. Results of flowspan performance and maximum traffic measurements: (a) flowspan reduction ratio of FORTD and proposed methods compared to YKTD method and (b) maximum traffic reduction ratio of FORTD method and proposed method compared to YKTD method.

The FORTD method reduced the maximum traffic compared with the YKTD method in all scenarios, effectively alleviating bottlenecks. However, the proposed method exhibited more significant bottlenecks than the YKTD method in 7% of the scenarios. Nevertheless, the proposed method achieved a smaller flowspan than the YKTD method in all scheduling scenarios. Conversely, the FORTD method, which excelled in traffic distribution, had the same or a larger flowspan than the YKTD method in 10% of the scenarios.

Figure 6 illustrates the cost growth rate (a) and topology design time (b) as functions of the number of nodes and frames. In most scenarios, the FORTD method incurred higher topological costs than the YKTD method. Conversely, the proposed method incurred higher costs only when the topology size was small, and the topology costs decreased as the network size increased. This behavior was attributed to the FORTD method, which selected dense paths in the schedule even if the hop count was long.

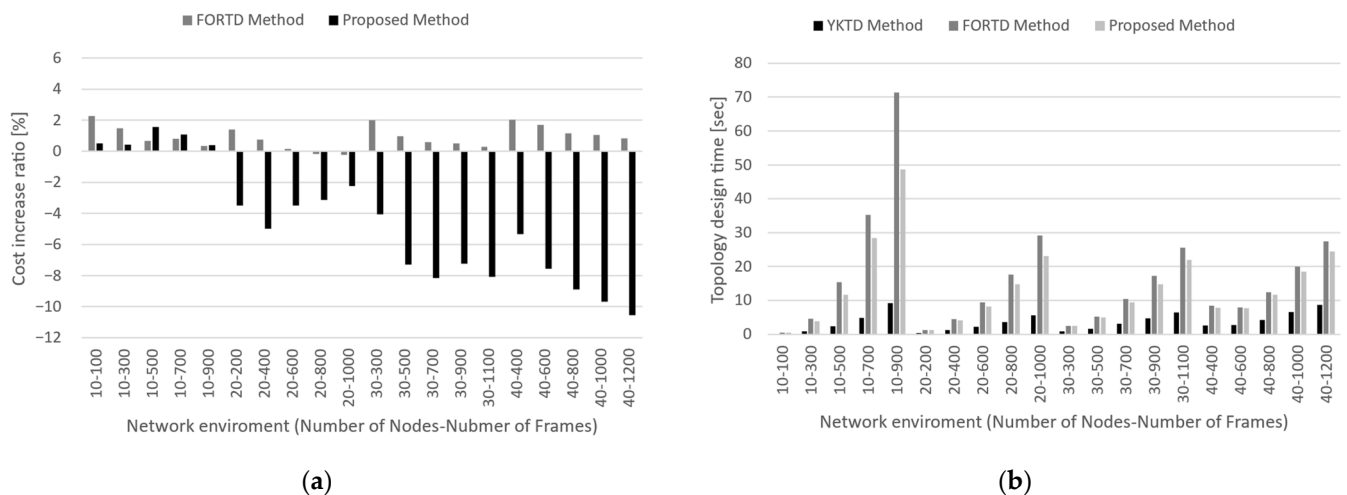


Figure 6. Results of topology cost and topology design time measurements: (a) increased topology cost measurement results of the proposed and FORTD method based on YKTD method and (b) topology design time of each method.

The topology design time increased with the number of frames, particularly when the number of terminating nodes was small. This was because it took more time to check the feasibility of ST schedule placement owing to the complexity of the schedule table

during path selection. Notably, the FORTD method required the most time for topology design. This was because Yen's KSP added the traffic distribution process to routing, thereby increasing the complexity of route discovery.

In summary, the proposed method exhibited a less effective traffic distribution than the FORTD method and required more time for topology design than the YKTD method. However, it reduced the topology costs when the topology size increased, with the topology design requiring a short time of up to 50 s. In particular, the proposed method reduced the flowspan in all scenarios compared with the YKTD method. These results demonstrated that the proposed topology design method could design redundant topologies with improved scheduling performance.

5. Conclusions

This study proposes a method for designing a redundant topology required for ST frame transmission using the fault-tolerant feature of the IEEE 802.1CB FRER standard in TSN networks. The proposed method used Yen's KSP for topology design, establishing two redundant paths for the inherent fault tolerance of critical ST frames. Specifically, during topology design, the method minimized bandwidth waste during ST transmission by activating paths that optimize the performance of the NW-PSP.

The proposed topology design method comprised three main stages: initialization, routing, and topology update. In the initialization stage, the graph model used for the topology design was initialized. In the routing stage, K -paths were generated using Yen's KSP. After placing ST frames in the schedule table using K -paths, the method selected paths with the potential to optimize the scheduling performance, considering the maximum traffic and transmission completion time. Then, the selected paths were used to update the topology, thereby completing the topology design.

To evaluate the performance of the proposed topology design method, we conducted simulations in various network environments while considering the network cost and traffic distribution methods. In conclusion, the proposed method streamlined scheduling and demonstrated notable efficiency in large-scale topologies, thereby providing an effective approach for topology design that minimized bandwidth waste.

However, note that the proposed topology design method focuses exclusively on optimizing the flowspan, which was a crucial performance metric for NW-PSP. Further research is required to assess the impact of the proposed method on scheduling optimization related to guard bands, jitter, and delays.

Another limitation of the proposed method is the absence of consideration of physical constraints during topology design. Therefore, future studies should investigate the integration of physical limitations into the proposed approach.

Author Contributions: Conceptualization, J.-Y.H., H.-J.K. and S.L.; methodology, J.-Y.H. and Y.-H.J.; software, J.-Y.H.; validation, J.-Y.H., Y.-H.J., H.-J.K. and S.L.; writing—original draft preparation, J.-Y.H.; writing—review and editing, J.-Y.H., H.-J.K. and S.L.; visualization, J.-Y.H.; supervision, H.-J.K. and S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by a research grant from Gyeongsang National University in 2023.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Parekh, D.; Poddar, N.; Rajpurkar, A.; Chahal, M.; Kumar, N.; Joshi, G.P.; Cho, W. A review on autonomous vehicles: Progress, methods and challenges. *Electronics* **2022**, *11*, 2162. <https://doi.org/10.3390/electronics11142162>.
2. Faisal, A.; Kamruzzaman, M.; Yigitcanlar, T.; Currie, G. Understanding autonomous vehicles. *J. Transp. Land Use* **2019**, *12*, 45–72.
3. Beliatas, M.J.; Jensen, K.; Ellegaard, L.; Aagaard, A.; Presser, M. Next generation industrial IoT digitalization for traceability in metal manufacturing industry: A case study of industry 4.0. *Electronics* **2021**, *10*, 628. <https://doi.org/10.3390/electronics10050628>.

4. Knight, J.C. Safety critical systems: Challenges and directions. In Proceedings of the 24th International Conference on Software Engineering, Orlando, FL, USA, 19–25 May 2002; pp. 547–550. <https://doi.org/10.1109/ICSE.2002.1007998>.
5. Nasrallah, A.; Thyagaturu, A.S.; Alharbi, Z.; Wang, C.; Shao, X.; Reisslein, M.; ElBakoury, H. Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research. *IEEE Commun. Surv. Tutor.* **2018**, *21*, 88–145. <https://doi.org/10.1109/COMST.2018.2869350>.
6. Gavriluț, V.; Madsen, C.E.L.; Stübert Berger, M.S. Selection of FDI-tolerant time-sensitive networking (TSN)-based architecture for preserving determinism of safety-critical applications. In Proceedings of the 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), Canary Islands, India, 19–21 July 2023; IEEE Publications: Piscataway, NJ, USA, 2023; Volume 2023, pp. 1–8. <https://doi.org/10.1109/ICECCME57830.2023.10253364>.
7. IEEE TSN Working Group [Online]. Available online: www.ieee802.org/1/pages/tsn.html (accessed on 1 July 2021).
8. *IEEE STD 802.1CB-2017*; IEEE Standard for Local and Metropolitan Area Networks—Frame Replication and Elimination for Reliability. IEEE Standard: Piscataway, NJ, USA, 2017; pp. 1–102.
9. Kirrmann, H.; Weber, K.; Kleineberg, O.; Weibel, H. HSR: Zero recovery time and low-cost redundancy for Industrial Ethernet (High availability seamless redundancy, IEC 62439–3). In Proceedings of the IEEE Conference on Emerging Technologies & Factory Automation, Palma de Mallorca, Spain, 22–25 September 2009; IEEE Publications: Piscataway, NJ, USA, 2009; Volume 2009, pp. 1–4. <https://doi.org/10.1109/ETFA.2009.5347037>.
10. Antonova, G.; Frisk, L.; Tournier, J.C. Communication redundancy for substation automation. In Proceedings of the 64th Annual Conference for Protective Relay Engineers, College Station, TX, USA, 11–14 April 2011; IEEE Publications: Piscataway, NJ, USA, 2011; Volume 2011; pp. 344–355. <https://doi.org/10.1109/CPRE.2011.6035636>.
11. Ferrari, P.; Flammini, A.; Rinaldi, S.; Prytz, G.; Hussain, R. Multipath redundancy for industrial networks using IEEE 802.1aq Shortest Path Bridging. In Proceedings of the 10th IEEE Workshop on Factory Communication Systems (WFCS 2014), Toulouse, France, 5–7 May 2014; IEEE Publications: Piscataway, NJ, USA, 2014; Volume 2014, pp. 1–10.
12. Kirrmann, H.; Weber, K.; Kleineberg, O.; Weibel, H. Seamless and low-cost redundancy for substation automation systems (high availability seamless redundancy, HSR). In Proceedings of the 2011 IEEE Power and Energy Society General Meeting, Detroit, MI, USA, 24–28 July 2011; IEEE Publications: Piscataway, NJ, USA, 2011; Volume 2011, pp. 1–7. <https://doi.org/10.1109/PES.2011.6038906>.
13. Hansen, K. Redundancy Ethernet in industrial automation. In Proceedings of the 2005 IEEE Conference on Emerging Technologies and Factory Automation, Catania, Italy, 19–22 September 2005; IEEE Publications: Piscataway, NJ, USA, 2005; Volume 2.
14. Kentis, A.M.; Berger, M.S.; Soler, J. Effects of port congestion in the gate control list scheduling of time sensitive networks. In Proceedings of the 8th International Conference on the Network of the Future (NOF), London, UK, 22–24 November 2017; IEEE Publications: Piscataway, NJ, USA, 2017; Volume 2017, pp. 138–140. <https://doi.org/10.1109/NOF.2017.8251236>.
15. Allan, D.; Farkas, J.; Mansfield, S. Intelligent load balancing for shortest path bridging. *IEEE Commun. Mag.* **2012**, *50*, 163–167. <https://doi.org/10.1109/MCOM.2012.6231293>.
16. Pahlevan, M.; Schmeck, J.; Obermaier, R. Evaluation of tsn dynamic configuration model for safety-critical applications. In Proceedings of the 2019 IEEE International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), Xiamen, China, 16–18 December 2019; IEEE: Piscataway, NJ, USA, 2019; Volume 2019, pp. 566–571. <https://doi.org/10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00086>.
17. Deng, L.; Xie, G.; Liu, H.; Han, Y.; Li, R.; Li, K. A survey of real-time ethernet modeling and design methodologies: From AVB to TSN. *ACM Comput. Surv.* **2023**, *55*, 1–36. <https://doi.org/10.1145/3487330>.
18. *IEEE STD 802.1Qbv-2015*; IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 25: Enhancements for Scheduled Traffic. IEEE Standard: Piscataway, NJ, USA, 2016; 1–57.
19. Craciunas, S.S.; Oliver, R.S.; Chmelík, M.; Steiner, W. 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, Brest, France, 19–21 October 2016; pp. 183–192.
20. Craciunas, S.S.; Oliver, R.S.; Ag, T. An overview of scheduling mechanisms for time-sensitive networks. In *Proceedings of the Real-Time Summer School L'École d'Été Temps Réel (ETR)*; Technical Report; TTTech Computertechnik AG: Vienna, Austria, 2017; Volume 3203, pp. 1551.
21. Schweissguth, E.; Danielis, P.; Timmermann, D.; Parzyjegl, H.; Mühl, G. ILP-based joint routing and scheduling for time-triggered networks. In Proceedings of the 25th International Conference on Real-Time Networks and Systems, Grenoble, France, 4–6 October 2017; pp. 8–17. <https://doi.org/10.1145/3139258.3139289>.
22. Dürr, F.; Nayak, N.G. No-wait packet scheduling for IEEE time-sensitive networks (TSN). In Proceedings of the 24th International Conference on Real-Time Networks and Systems, Brest, France, 19–21 October 2016; pp. 203–212. <https://doi.org/10.1145/2997465.2997494>.
23. Gavriluț, V.; Zarrin, B.; Pop, P.; Samii, S. Fault-tolerant topology and routing synthesis for IEEE time-sensitive networking. In Proceedings of the 25th International Conference on Real-Time Networks and Systems, Grenoble, France, 4–6 October 2017; pp. 267–276. <https://doi.org/10.1145/3139258.3139284>.
24. Atallah, A.A.; Hamad, G.B.; Mohamed, O.A. Fault-resilient topology planning and traffic configuration for IEEE 802.1 Qbv TSN networks. In Proceedings of the 24th International Symposium on On-Line Testing And Robust System Design (IOLTS), Platja D'Aro, Spain, 2–4 July 2018; IEEE Publications: Piscataway, NJ, USA, 2018; Volume 2018, pp. 151–156.

-
25. Nayak, N.G.; Dürr, F.; Rothermel, K. Routing algorithms for IEEE802.1Qbv networks. *SIGBED Rev.* **2018**, *15*, 13–18. <https://doi.org/10.1145/3267419.3267421>.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.