electronics MDPI

Article

A Period-Aware Routing Method for IEEE 802.1Qbv TSN Networks

Kai Huang [1], Jingkang Wu [1], Xiaowen Jiang [1,*], Dongliang Xiong [2], Kaitian Huang [3], Hao Yao [4], Wenyuan Xu [2], Yonggang Peng [2] and Zhili Liu [5]

[1] Institute of VLSI Design, Zhejiang University, Hangzhou 310027, China; huangk@vlsi.zju.edu.cn (K.H.); 21860243@zju.edu.cn (J.W.)
[2] Department of Electrical Engineering, Zhejiang University, Hangzhou 310027, China; xiongdl@zju.edu.cn (D.X.); wyxu@zju.edu.cn (W.X.); pengyg@zju.edu.cn (Y.P.)
[3] Electric Power Research Institute, China Southern Power Grid (CSG), Guangzhou 510623, China; huangkt@csg.cn
[4] Digital Grid Research Institute, China Southern Power Grid (CSG), Guangzhou 510670, China; yaohao@csg.cn
[5] Hangzhou Sec-Chip Technology Co., Ltd., Hangzhou 310012, China; liuzhili@sec-chip.com
* Correspondence: xiaowen_jiang@zju.edu.cn

Abstract: The IEEE 802.1Qbv standard provides deterministic delay and low jitter guarantee for time-critical communication using a precomputed cyclic transmission schedule. Computing such transmission schedule requires routing the flows first, which significantly affects the quality of the schedule. So far off-the-shelf algorithms like load-balanced routing, which minimize the maximum scheduled traffic load (*MSTL*), have been used to accommodate more time-triggered traffic. However, they do not consider that the bandwidth utilization of periodic flows is decentralized and their criteria for bottleneck of scheduling are imprecise. In this paper, we firstly explore the combinability among different periods of flows, which can measure their ability to share bandwidth without conflict. Then, we propose a novel period-aware routing algorithm to reduce the scheduling bottleneck, thus more flows can be accommodated. The experiment results show that the success rate of scheduling is significantly improved compared to shortest path routing and load balanced routing.

Keywords: IEEE 802.1Qbv; TSN; routing algorithms; period-aware; schedulability

check for updates

Citation: Huang, K.; Wu, J.; Jiang, X.; Xiong, D.; Huang, K.; Yao, H.; Xu, W.; Peng, Y.; Liu, Z. A Period-Aware Routing Method for IEEE 802.1Qbv TSN Networks. *Electronics* **2021**, *10*, 58. https://doi.org/10.3390/electronics10010058

Received: 14 December 2020
Accepted: 28 December 2020
Published: 31 December 2020

1. Introduction

In the field of industrial control and automotive networks, critical traffic such as real-time control information requires bound low delay and low jitter. However, the traditional Ethernet lacks the guarantee of real-time and certainty of communication. Although the rising real-time Ethernet can provide certain guarantees in real-time and certainty, there are some defects such as inconsistent standards and unclear development direction [1–4]. To address this limitation, the IEEE time sensitive network (TSN) task group has developed a set of unified standards based on AVB (audio video bridging) protocol, which is not only compatible with traditional Ethernet, but also ensures real-time performance and low jitter. These standards consist of time synchronization, paths selection, path reservation, fault tolerance, scheduling, traffic shaping and so on [5,6]. As part of the TSN standards, it is committed to providing deterministic network delay and low jitter guarantee for time triggered (periodic) traffic flow in real-time control, and can be compatible with best-effort flows and AVB flows [7].

To realize the time-aware scheduling defined in 802.1Qbv, precise time synchronization protocols, such as IEEE 802.1AS, are required. Such protocols enable all bridges and end stations in the network to synchronize their local clocks with an accuracy of less than 1 µs. A pre-computed communication schedule is sent to every end station in the system, instructing them when to send which frame to the network, and the end station executes the communication schedule precisely [8–10]. In this way, communication interference occurring in standard Ethernet can be prevented, and definite delays and low jitter can be guaranteed during message delivery. Such a mechanism is usually implemented as offline scheduling because of the large amount of computation [11,12].

Generally, the TSN traffic control mechanism can be considered from two aspects: (1) the scheduling (i.e., when to send which frame to be transmitted); and (2) the routing (i.e., selecting a path from source to destination for each flow) [13]. With regard to the scheduling problem, some formulas or restrictions that meet the time requirements are usually introduced, and are solved by integer linear programming (ILP) and satisfiable modular theory (SMT) [14]. Such a scheduling task is an NP-hard problem [15], and thus, solving routing and scheduling problems jointly [8] is too complex to meet the computing time limit for large-scale networks. Therefore, the typical solution is to solve the routing problem first and then the scheduling problem.

Usually, the same off-the-shelf algorithms, which are used for routing best-effort traffic, are used to route time-triggered flows. As with shortest path routing, it minimizes the number of hops for routing traffic [16]. These strategies may result in too many flows trying to simultaneously traverse the same path, reducing the schedulability of the set of time-triggered flows. Naresh et al. proposed an ILP (integer linear programming) based routing algorithm to reduce maximum scheduled traffic load (*MSTL*), and thus improve the schedulability [5]. Furthermore, since the runtime of ILP based algorithms is very high, the authors of literatures [3,4] developed a heuristic to reduce *MSTL*, respectively. However, they did not take different periods of flows into account, hence not being available for time-triggered flows with different periods. Literatures [6,8] considered "period" as a factor in choosing the path, but they omitted that bandwidth utilization of periodic flows is dispersed, which is essential to precisely evaluate the scheduling bottleneck. To the best of our knowledge, there is no work on TSN routing that pays attention to the difference in sharing bandwidth between different periods of flows.

In this article, we address the following research question: How to improve the schedulability of time-triggered flows in routing stage? To get over the aforementioned hurdles and better address this issue, we therefore make the following contributions:

- We first analyze the combinability among various periods of flows and show the ability of different periods of flows to utilize the decentralized bandwidth.
- We introduce a new metric to measure the bottleneck of scheduling. Moreover, we propose a period-aware routing algorithm for time-triggered flows and improve their schedulability.
- Finally, we evaluate the proposed algorithm from three dimensions: Load, topology and flow type. The experiment results demonstrate that our method can achieve higher schedulability compared to the shortest path algorithm and the load balanced algorithm.

The rest of the paper is organized as follows. The related work is discussed in Section 2. In Section 3, we describe the system model and the problem under consideration. Then the motivation case is presented in Section 4. In Section 5, we analyze the combinability among flows in detail, based on which we proposed a period-aware routing algorithm for time-triggered flows. We evaluate our algorithm using an ILP-based schedule platform in Section 6. Finally, we conclude in Section 7.

2. Related Work

Routing algorithms have been well studied and some advanced results have been proposed [17,18]. Nevertheless, the contributions to TSN routing schemes only gradually emerged ten years ago [3]. Rapid spanning tree protocol and shortest path algorithm are widely used for TSN, adopting the convention of typical Ethernet [19]. Furthermore, the IEEE 802.1Qca provides explicit control over routing and mechanisms and recommends constrained shortest path first routing for TSN transmissions [20,21]. However, as previously described, shortest path routing is likely to reduce the schedulability for time-triggered flows.

In recent studies, some works [22,23] extended the routing problem to the topology planning problem. While in practical applications, the greater demand is to find the best routing scheme based on the given topology. In addition, some researches [8,24] focused on jointly solving the routing problem and scheduling problem. Although such a method can achieve better scheduling quality, it is too time-consuming for large-scale networks. Consequently, most of the methods are to solve the routing problem first, and then solve the scheduling problem.

There are many optimization goals for routing problems, such as scalability, slack, delay, et cetera [25–27], but the core issue is schedulability, because if schedulability cannot be guaranteed, other indicators become meaningless. In terms of schedulability, the main consideration is how to distribute the load as evenly as possible. This is the so-called load balance. Arif and Atia [28] proposed a load balanced routing model for TSN, but it focused on improving the circuit delay rather than schedulability. Nayak et al. [5] proposed two ILP-based routing algorithms to optimize the *MSTL*. However, the runtimes of the ILP-based method are too long, and thus detrimental to scalability. Load balanced routing heuristics are developed in literature [3,4], but they did not consider different periods of flows. Therefore, it does not meet the needs of practical application where different control devices transmit the messages at different frequencies. Although "period" has been considered as a factor for path selection in literatures [6,8], they did not recognize the decentralized bandwidth utilization of periodic flows and they just changed the metric to evaluate *MSTL* by taking "period" into account. Therefore, these methods are not accurate in measuring the scheduling bottleneck. In contrast, we find that flows with different periods have different abilities to utilize the dispersed bandwidth, and use the concept "combinability" to evaluate it. By carefully analyzing the combinability among flows with different periods, we propose a heuristic to reduce the scheduling bottleneck and increase the probability of a set of time-triggered flows to be successfully scheduled.

3. System Model and Problem Definition

In this section, we present our system model and the problem that we focus on. The main notations that we use are listed in Table 1.

Table 1. The main notations that we use.

Notation	Explanation
src	The source node of a flow.
dst	The destination node of a flow.
siz	The size of a flow measured by the length of time consumed during propagation.
prd	The cycle period of a flow.
$MSTL$	The maximum scheduled traffic load.
LCM	The least common multiple of the periods of all flows.
GCD	The greatest common divisor of the periods of all flows.
BW_u	The used bandwidth.
BW_t	The total bandwidth,
BW_r	The residual bandwidth.
num	The number of flows that can be accommodated.
wgt	The specific weight of a flow.
SOW	Sum of wgt.
$MSOW$	The maximum sum of wgt.
$hops$	The number of edge/link in a path.

3.1. System Model

We model the time-sensitive network as a directed graph $G = (V, E)$, where the set $V = ES \cup S$ consists of a finite set ES of end stations and S of switches or bridges, and the set $E = \{(vi, vj) \mid vi, vj \in V, \text{and } vi, vj \text{ are connected.}\}$ includes a set of full duplex edges or links. Note that each full duplex edge is regarded as two separate directional edges (vi, *vj*) and (*vj*, *vi*), where the former node is the source and the latter node is the destination. We assume the whole network is precisely synchronized. An example of network topology is shown in Figure 1.

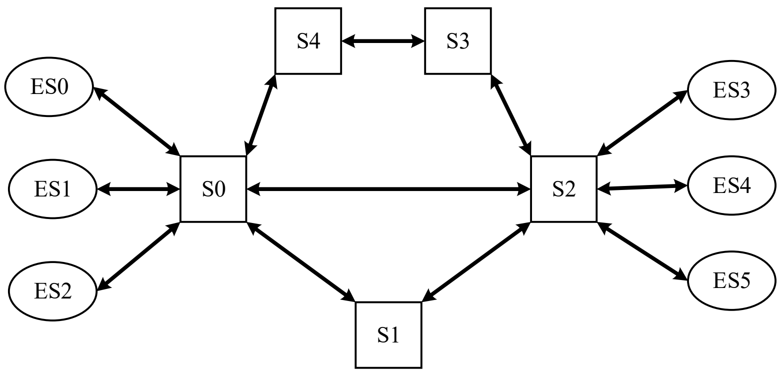


Figure 1. Sample network topology consisting of six end stations and five bridges.

A set of unicast time-triggered flows F is considered and the flow mentioned later refers to time-triggered flows in default. These flows are transmitted within their own cycle and the hyper cycle (the least common multiple of the periods of all flows). When all flows are released simultaneously, we can safely restrict our attention to a single hyper cycle (the first one) [3]. Each flow $\in F$ is denoted by a 4-tuple $f \equiv (src, dst, siz, prd)$, where src, $dst \in V$ are the flow's source and destination node, respectively. *Siz* defines the size of the flow by the length of time consumed during propagation. For example, the bandwidth of TSN is 100 Mb/s and the length of the flow is 100 Byte, then the "*siz*" for the flow is 8 µs. Moreover, *prd* is the cycle period of the flow. For simplicity, we assume the minimum time unit of our system is 1 µs and the unit for *siz* and *prd* is microsecond by default. Since we mainly focus on the schedulability rather than latency, we assume that any valid solution meets the flow's deadline.

3.2. Problem Definition

The considered flows must be transmitted based on a commonly used no-wait schedule [5,6,15]. Given the network topology and a set of flows with different periods, the routing algorithm finds a path for every given flow from its source to destination, aiming to improve the success rate of scheduling.

4. Motivation Example

In this section, we analyze the shortcomings of the existing load balanced routing algorithm [1,3,4] to better illustrate our motivation.

When selecting the path for a flow, we mainly focus on how to reduce the bandwidth utilization of the bottleneck edge. Maximum scheduled traffic load (*MSTL*) is regarded as the bottleneck in existing methods [3–5] and they proposed different algorithms to minimize the *MSTL*, which is computed as

$$MTSL = \max(\sum siz) \tag{1}$$

or

$$MTSL = \max(\sum siz \times LCM/prd) \tag{2}$$

Since we only focus on how the flows are transmitted within a hyper cycle, the hyper cycle can be regarded as the total bandwidth and it will be allocated to the flows. Actually, Equation (2) computes the maximum bandwidth used in a hyper cycle. Letting BW_u be the used bandwidth, we have

$$BW_u = \sum siz \times LCM/prd \tag{3}$$

According to existing methods, the smaller the *MSTL* is, the more bandwidth is left, and the more flows it can hold, because they assume that the remaining bandwidth can be fully utilized. As is shown in Figure 2a, it is reasonable when the cycle is not considered and the flows can be tightly bunched. Nevertheless, when we take the cycle into consideration, it is inappropriate to use the residual bandwidth to measure the capacity to accommodate flows, because they are decentralized as is shown in Figure 2b. Therefore, the remaining bandwidth may be useless for some flows.

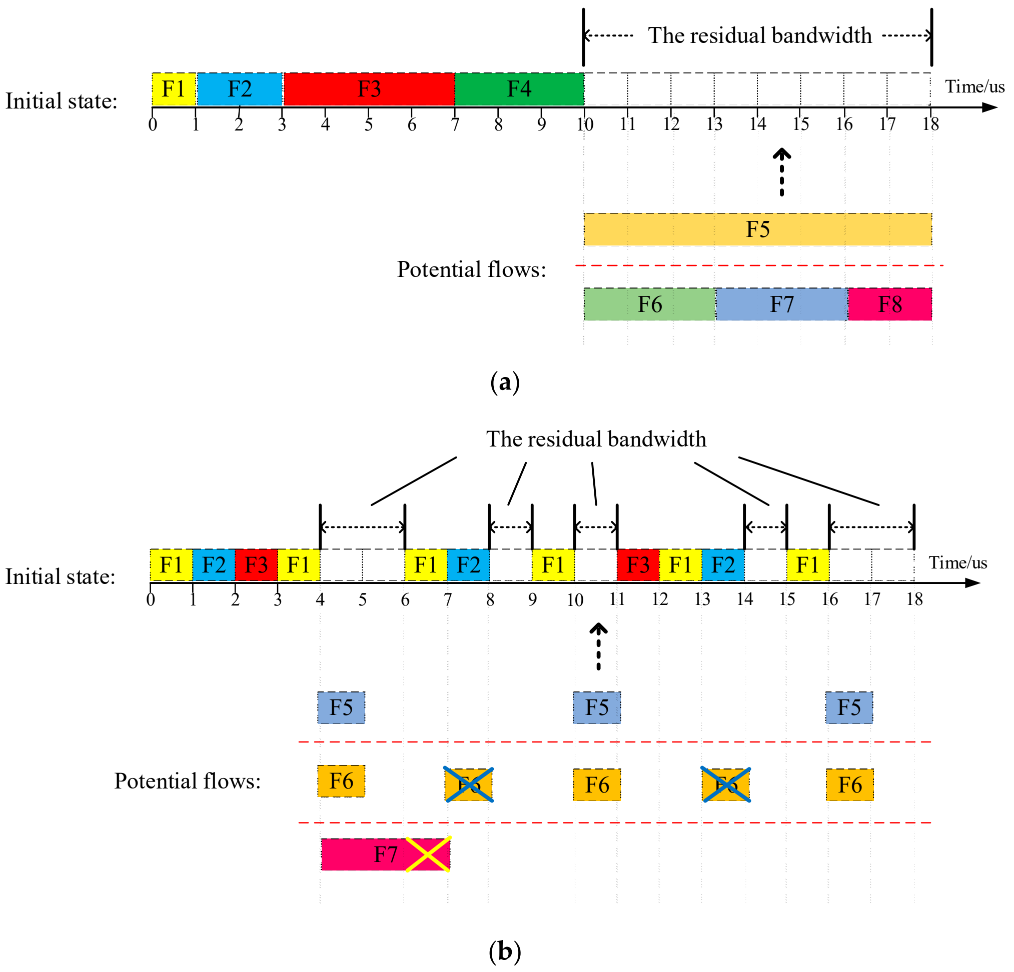


Figure 2. (**a**) The residual bandwidth and potential flows for aperiodic flows; (**b**) the residual bandwidth and potential flows for periodic flows.

Let BW_t be the total bandwidth, the theoretical residual bandwidth (denoted as BW_r) can be computed as

$$BW_r = BW_t - BW_u \tag{4}$$

For aperiodic flows, it can be calculated that BW_r = 8 in Figure 2a, which means that it can accommodate a flow with *siz* up to 8 or several flows with a total *siz* up to 8. While for periodic flows shown in Figure 2b, it can be calculated that BW_r = 7 according to Equation (4). Three potential flows are added: (1) F5: *siz* = 1, *prd* = 6; (2) F6: *siz* = 1, *prd* = 3; (3) F7: *siz* = 3, *prd* = 18, whose BW_u are 3, 6, 3 respectively. It is obvious that they are all less than BW_r, but only F5 can be accommodated.

This shows that more bandwidth remaining does not mean that more flows can be accommodated. The capacity of different flows to utilize specific remaining bandwidth is different. In other words, some flows are not prone to conflict when sharing the same edge, while others are prone to conflict. Existing methods only focus on bandwidth distribution and ignore this characteristic of different flows. Therefore, we need a new criterion to show whether a specific flow can be accommodated on an edge.

5. Proposed Solution

In this section, we propose a period-aware routing heuristic for time-triggered flows. "Period-aware" means that the period of the flow is regarded as the essential element of path selection. First of all, we proposed the concept of combinability, which helps us to understand the ability of different periods of flows to share bandwidth without conflict. After that we analyze the combinability of different periods of flows and put forward a new metric to measure the bottleneck of scheduling. Then we give an example to illustrate the advantages of our method compared to the load balanced algorithm proposed in literature [3]. Finally, the procedure of our algorithm is described in detail.

5.1. The Combinability of Different Periods of Flows

When more than one time-triggered flow is passing through the same edge, the scheduling algorithm must assure that the transmission time for these flows is not overlapped to avoid congestion on that edge [2]. We put forward the concept of "combinability" to measure the capacity of flows to share bandwidth without conflict. More specifically, high combinability means that flows transmitted via the same edge are not likely to congest, bandwidth can be used more fully and this edge is able to accommodate more flows. It should be pointed out that when flows pass through the same edge, the following two restrictions need to be met:

- The transmission times of flows are not overlapped in one hyper cycle [6].
- The first frame of each flow should be sent within its own period [5].

5.1.1. Examples for Different Combinability

For example, there are two flows passing through the same edge: (1) F1: $siz = 1$, $prd = 3$; (2) F2: $siz = 1$, $prd = 6$. The hyper cycle of these two flows is 6, which means that we only need to care about time 0 to time 6. Supposing F1 is sent at time 0, the possible situations of how F2 is transmitted under the above restrictions is shown in Figure 3.

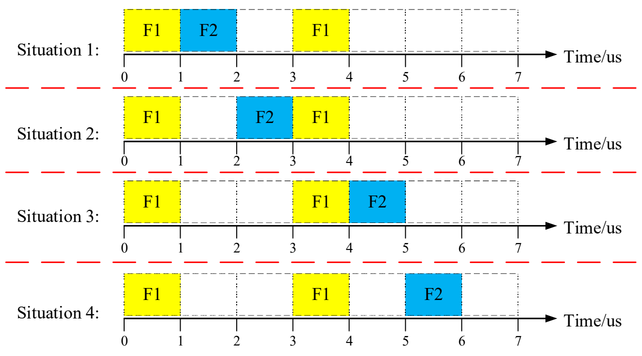


Figure 3. Possible situations for F2 when F1 sent at time 0.

The figure shows that there are up to four feasible sending methods for F2. Moreover, it is not difficult to find out that the edge can accommodate up to four F2 and bandwidth can be fully used. Therefore, there is a high combinability between F1 and F2.

Instead, we consider these two flows: (1) F3: $siz = 1$, $prd = 3$; (2) F4: $siz = 1$, $prd = 6$. The hyper cycle is 12 and the possible situations for F4 is shown in Figure 4 assuming F3 is sent at time 0.

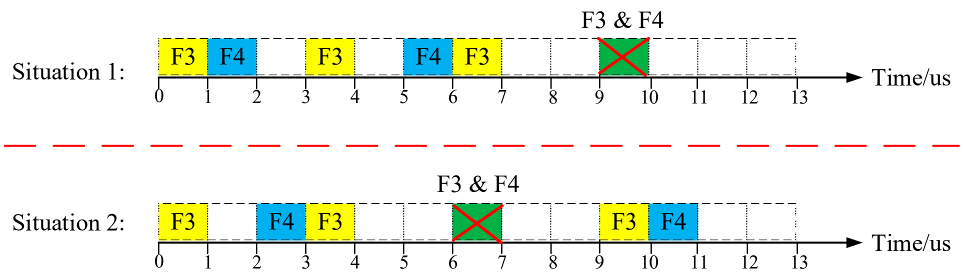


Figure 4. Possible situations for F4 when F3 sent at time 0.

We can see that F3 and F4 congest at time 9–10 in situation 1 while in situation 2 they congest at time 6–7. With the first frame of F4 must sent before time 4 and not congesting with F3, there is no other situation for F4 to be transmitted, which means that the two flows are bound to conflict. Consequently, there is poor combinability (bandwidth utilization is zero) between them and it is not suitable for them to pass through the same edge.

5.1.2. Analysis of the Combinability between Two Flows

From the above examples, we have a preliminary understanding of different combinability. In this section, we analyze the combinability between two flows with different periods in more depth.

In the following analysis, we assume that the *siz* of all the flows are "1" and explore how many other flows can be accommodated when a certain stream exists on the edge. Specifically, as the first example shown in Section 5.1.1, four F2 can be accommodated when an F1 exists. Obviously, we can use this number to measure the combinability between two flows, or whether it is suitable for the latter flow to be transmitted via the specific edge, which we can use to select path for the flow.

Without loss of generality, we assume that flow1 with $prd = p1$ has been assigned to be transmitted through an edge at time 0. Then we analyze how many feasible situations for flow2 with $prd = p2$ are sent through the edge without congestion. We denote the time offset of the first frame of flow2 to be sent as a, and we get $0 \leq a < p2$ under the transmitting time limitation. Then the sending time for flow1 and flow2 would be $n \times p1$ and $(a + m \times p2)$, where m and n are integers (denoted as N). If the two flows congest at an edge, the following equation can be obtained:

$$n \times p1 = a + m \times p2, \tag{5}$$

and such that:

$$a = n \times p1 - m \times p2. \tag{6}$$

Assuming the greatest common divisor (*GCD*, *GCD* also means the *GCD* of periods in this paper) of $p1$ and $p2$ is g, we have

$$\exists m_0, n_0 \in N,\ p1 = n_0 \times g,\ p2 = m_0 \times g, \tag{7}$$

where the *GCD* of n_0 and m_0 is "1", otherwise g cannot be the *GCD* of $p1$ and $p2$. Additionally, a is computed as

$$a = n \times n_0 \times g - m \times m_0 \times g = (n \times n_0 - m \times m_0) \times g. \tag{8}$$

As n, n_0, m and m_0 are integers, it can be concluded that a is an integral multiple of g. Then we explore the possible values for a, before which we quote the famous Bézout's identity [29]:

"For nonzero integers a and b, let d be the greatest common divisor $d = gcd(a,b)$. Then, there exist integers x and y such that: $ax + by = d$."

Apparently, n_0 and m_0 are nonzero integers. Where the GCD of n_0 and m_0 is "1" and $(n \times n_0 - m \times m_0)$ is a linear combination of n_0 and m_0, we can deduce that

$$\exists m_1, n_1 \in N,\ n_1 \times n_0 - m_1 \times m_0 = 1. \tag{9}$$

Note that m and n can be any integer. From Equations (8) and (9), we get

$$\forall k \in N,\ \exists m,\ n \in N,\ a = (n * n_0 - m * m_0) * g = k * (n_1 * n_0 - m_1 * m_0) * g = k * g, \tag{10}$$

only when

$$n = k \times n_1,\ m = k \times m_1. \tag{11}$$

By Equation (10), the value range of a (denoted as A) is

$$A \equiv \{0 \le a < p2 \mid a = k \times g, k \in N\} \equiv \{0, g, 2g, \ldots, p2 - g\}. \tag{12}$$

Therefore, the number of possible values for a are $p2/g$. In other words, the number of situations in which the two flows do not congest is $(p2 - p2/g)$. More generally, let *num* be the number of flows that can be accommodated, such that:

$$num\ = prd - prd/GCD. \tag{13}$$

From the above discussion, we find an expression that can evaluate the combinability between two flows when their *siz* are both "1". As for the situations when *siz* is an arbitrary integer, considering that *siz* is generally much smaller than *prd* in real application scenarios, we can roughly treat a flow with *siz* = *n* as *n* flows with *siz* = 1. Then, *num* is computed as

$$num\ = \lfloor (prd - prd/GCD)/siz \rfloor. \tag{14}$$

Since *prd* and *siz* are the properties of the flow itself, *GCD* is the determinant of the combinability of two flows.

5.1.3. Combinability among Multiple Flows

The above analysis points out the combinability between two flows and the upper limit that can accommodate one flow when the other flow exists, but it is not enough to select a path for a newly added flow. A metric that can be applied to a real-world scenario, where multiple periods of flows are transmitted via a same edge, is needed.

Equation (13) indicates the maximum number of flows that can be accommodated when the other flow exists and *siz* are both "1". By converting this number to the specific weight (denoted as *wgt*) of each flow, we get

$$wgt = 1/num\ = 1/(prd\ - \frac{prd}{GCD}). \tag{15}$$

As for the situation when *siz* is an arbitrary value, by Equation (14), we have

$$wgt = 1/num\ = siz/(prd\ - \frac{prd}{GCD}). \tag{16}$$

Then the sum of *wgt* (denoted ad *SOW*) is computed as

$$SOW\ = \sum siz/(prd\ - \frac{prd}{GCD}). \tag{17}$$

The higher the *SOW*, the closer to the upper limit of the flows that can be accommodated, and the more difficult it is to accommodate more flows. It means that *SOW* can be used to measure the scheduling bottleneck when there are flows with two kinds of periods.

Similar to *MSTL*, we propose using the maximum *SOW* (denoted as *MSOW*) as the bottleneck of scheduling, and we get

$$MSOW = \max\left(\sum \frac{siz}{prd - \frac{prd}{GCD}}\right). \tag{18}$$

As *LCM* is a constant for the same set of flows, comparing Equation (2) and Equation (18), we find that there is only a difference between the two by a coefficient related to *GCD*. This shows that when considering the scheduling bottleneck, not only the total load on an edge must be considered, but also the combinability of flows passing through the same edge. When choosing a path for a flow, we should select a path whose *MSOW* is the least after adding the flow to it.

As shown in Figure 5, there are three candidate paths for F7 (*prd* = 6, *siz* = 1) to select. Table 2 shows the *SOW*, *TSL* (scheduled traffic load according to Equation (2)) and number of future flows that can be accommodated (*NOF*) after selecting different paths. It can be seen that the smaller the *SOW*, the more flows it can accommodate in the future. However, there is no such inverse relationship between *TSL* and *NOF*. It demonstrates that *MSOW* is more suitable than *MSTL* to measure the scheduling bottleneck. Notably, if there are two flows whose *GCD* is "1", arranging the same path for them will leave the flows unschedulable. This can be avoided by using *MSOW* as the metric of path selection, but not by *MSTL*. Consequently, trying to lower the value of *SOW* helps to accommodate more flows.

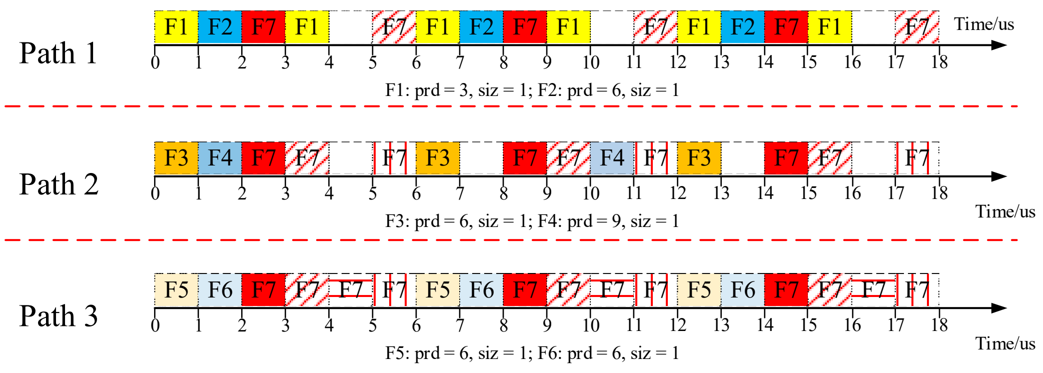

Path 1
F1 F2 F7 F1 F7 F1 F2 F7 F1 F7 F1 F2 F7 F1 F7
Time/us
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
F1: prd = 3, siz = 1; F2: prd = 6, siz = 1
Path 2
F3 F4 F7 F7 F7 F3 F7 F7 F4 F7 F3 F7 F7 F7
F3: prd = 6, siz = 1; F4: prd = 9, siz = 1
Path 3
F5 F6 F7 F7 F7 F7 F5 F6 F7 F7 F7 F7 F5 F6 F7 F7 F7 F7
F5: prd = 6, siz = 1; F6: prd = 6, siz = 1

Figure 5. The capacity to accommodate future flows after selecting a different path for F7.

Table 2. SOW, TSL[1] and NOF[2] after selecting different paths.

PATH	SOW	TSL	NOF
Path 1	1	12	1
Path 2	2/3	8	2
Path 3	3/5	9	3

[1] TSL: Scheduled Traffic Load according to Equation (2). [2] NOF: Number of future flows that can be accommodated.

5.2. The Proposed Routing Algorithm

The analysis in Section 5.1 indicates that *MSOW* is an appropriate metric to measure the scheduling bottleneck, based on which we proposed a period-aware routing algorithm (denoted as PAR). To facilitate the understanding of our algorithm, we first explain some of the symbols used in it.

Definition 1. *We denote each flow $\in F$ by a 4-tuple $f \equiv$ (src, dst, siz, prd) in Section 3, and we add an attribute "class" (initial value is 0) to it, which is determined by our algorithm. We use f.prd to* *represent the period of the corresponding flow, and the representation methods for other attributes are similar.*

Definition 2. *Three attributes "prd_list" (initial value is {}), "gcd" (initial value is 0) and "SOW" (initial value is 0) are added for each edge in the network. The first represents the set of periods of all flows passing through the edge (repeated elements will be merged into one element), while the second represents the greatest common divisor of all periods in the set. Additionally, "SOW" is the sum of wgt mentioned in Section 5.1.3.*

Definition 3. *The function least_common_multiple(F) returns the least common multiple of the periods of all flows* $\in F$.

Definition 4. *The function greatest_common_divisor(list) returns the greatest common divisor of the number in list.*

Definition 5. *The function get_wgt(f) returns the wgt for the flow f using Equation (16).*

Definition 6. *The function all_simple_paths(G, src, dst) returns all simple paths in the graph G from src to dst, where simple path is a path with no repeated modes.*

5.2.1. Period-Aware Routing Algorithm

The whole procedure of the period-aware routing algorithm is presented in Algorithm 1. In general, the algorithm consists of two stages: sorting and routing. Firstly, the flow set is sorted according to the sub-algorithm, Sorting_the_flows. Additionally, the path is selected for each flow in turn. When choosing a path for a flow, all the possible paths are found in the first step (line 4). The function *all_simple_paths* can be easily implemented using NetworkX [30], a popular Python language package for exploring and analyzing the network. All possible paths are traversed and the best path for the flow is found using the procedure Find_best_path (line 5).

Algorithm 1 Period-aware routing algorithm (PAR)

```
Input: Network topology $G$; Flows set $F$; Constant $K$
Output: Best path for each flow $Path\_set$
1: $Path\_set \leftarrow \{\}$
2: $Fs \leftarrow$ Sorting_the_flows($F$);
3: foreach $f_i \in Fs$ do
4:     $paths$ = $all\_simple\_paths$ ($G$, $flow.src$, $flow.dst$);
5:     $path^* \leftarrow$ Find_best_path ($G$, $f_i$, $paths$, $K$);
6:     $Path\_set$.append($path^*$);
7: end
8: return $Path\_set$
```

5.2.2. Sorting the Flows

In order to minimize *MSOW*, *GCD* of the flows passing through the same edge should be as large as possible. Consequently, we need to group the flows according to their impact on *GCD* or combinability. For instance, a flow whose *GCD* with every other flow is 1 has poor combinability with other flows, so the flow should use a separate path. Besides, changing the *GCD* of an edge after adding a flow should be avoided, otherwise *SOW* has to be recalculated for the previous flows. For flows that do not affect the *LCM* of the flows set after removal, their *prd* are the factor of others and they may be the *GCD* of an edge. We should route them earlier such that the *GCD* is not likely to change when the following flows are added. Therefore, we divide flows into three categories:

- Class 0: The flows whose *GCD* with every other flow is 1;
- Class 1: The flows do not affect *LCM* of the flows set after removal;

- Class 2: Other flows.

Moreover, it is not likely to reduce the *GCD* after adding a flow with a larger period, so we sort the flows in the order of the period from small to large. The procedures of sorting the flows are presented in Algorithm 2 as follows. In step I (line 1), *LCM* of the flows set (*lcm_old*) is calculated. In step II (lines 3–4), the *LCM* (*lcm_new*) is recalculated after removing the specific flow. In step III (lines 5–11), flows are classified. If the equation in line 5 holds, it means that the flow's *GCD* with every other flow is 1, so it falls into class 0. While if *lcm_new* is equal to *lcm_old*, it is classified as the class 1 and the others will be class 2. The above steps will be repeated until all flows are classified. Finally, in step IV (line 14), flows are sorted from small to large according to class and period.

Algorithm 2 Sorting the flows

```
   Input: Initial flows set F
   Output: Flows set after sorted Fs
1: lcm_old = least_common_multiple(F.prd);
2: foreach $f_i \in F$ do
3:     Remove $f_i$ from F;
4:     lcm_new = least_common_multiple(F.prd);
5:     if (lcm_new == lcm_old/$f_i$.prd) then
6:         fi.class = 0;
7:     else if (lcm_new == lcm_old)
8:         fi.class = 1;
9:     else
10:        fi.class = 2;
11:    end
12:    Append $f_i$ to F;
13: end
14: Fs ← sort F by f.class and f.prd from small to large
15: return Fs
```

5.2.3. Find Best Path

As a core part of our approach, the Find_best_path procedure is shown in Algorithm 3. First of all, we compute the number of edges (denoted as *hops*) in the path (line 1). Then *MSOW* of each path supposing the flow uses the path is calculated (line 3–11). After that, we calculate the cost function for the path (line 12). It is computed as

$$cost \ = MSOW + K * hops. \tag{19}$$

The purpose of our algorithm is to reduce the *MSOW* after all flows are routed. However, if each flow is routed to reduce *MSOW* as the optimization goal, a too-long path may be chosen in the front, causing each flow to add a large load to the entire network. For example, if we choose a path with eight edges for a flow, it will increase the load of this flow to eight edges. In other words, using a too-long path will reduce the bandwidth available for subsequent flows. Therefore, we introduce a user-defined positive integer *K* to penalize the length of the path. After the computation of the *cost*, we restore the state of the network to calculate the *cost* for the next path (lines 13–17). The path with the smallest *cost* is taken as the optimal path after traversing all the paths. Then the network state is changed after the flow chooses the optimal path (lines 20–24).

Algorithm 3 Find best path

```
   Input: Network topology $G$; Flow $f$; Paths set $Paths$; Constant $K$
   Output: Best path for flow $f$
1: foreach $path \in paths$ do
2:     $hops$ = len($path$) $-$ 1;
3:     $MSOW$ = 0;
4:     foreach $edge \in path$ do
5:         $edge.prd\_list$.append ($f.prd$);
6:         $wgt \leftarrow$ Calculate $wgt$ ($edge, f$);
7:         $edge.SOW$ += $wgt$;
8:         if ($SOW > MSOW$) then
9:             $MSOW = SOW$;
10:        end
11:    end
12:    $cost = MSOW + K \times hops$;
13:    foreach $edge \in path$ do
14:        $wgt \leftarrow$ Calculate $wgt$ ($edge, f$);
15:        $edge.prd\_list$.remove ($f.prd$);
16:        $edge.SOW = edge.SOW - wgt$;
17:    end
18: end
19: $path^* \leftarrow arg$ min $cost(path^*, K)$;
20: foreach $edge \in path^*$ do
21:     $wgt \leftarrow$ Calculate $wgt$ ($edge, f$);
22:     $edge.prd\_list$.remove ($f.prd$);
23:     $edge.SOW$ += $wgt$;
24: end
25: return $path^*$
```

As for calculating *wgt*, the procedure is shown in Algorithm 4. Note that the purpose of our previous sorting algorithm is to keep the *GCD* of a bridge unchanged after adding a new flow. If there is only one flow on the edge, we use the *prd* of the flow as *GCD*, which is very likely not change in future procedures. While if the *GCD* of an edge change, *edge.SOW* will have to be recalculated for the previous flows (lines 4–7). Although recalculating the value will increase the complexity of the algorithm, the overall complexity of the algorithm is not high because of the low probability of this happening. Moreover, if the *GCD* of an edge is equal to 1, the value of *wgt* is set to a large enough integer such that the path will not be selected. This can avoid unschedulable situations due to *GCD* being 1.

The four algorithms we proposed have been introduced, and then we analyze the complexity of these algorithms. We use $|F|$, $|N|$ and $|E|$ to denote the number of flows, nodes and edges in the graph, respectively. All flows are traversed in Algorithm 2, so the complexity of Algorithm 2 is $O(|F|)$. The complexity of Algorithm 4 is $O(1)$ since there is no loop in it. There are two layers of loops in Algorithm 3, which are the traversal of the number of paths and the traversal of the edges in the path. We roughly estimate that the number of paths for each flow is $2|E|$ and the number of edges in each path is $|E|/2$. Therefore, the complexity of the Algorithm 3 is $O(|E|^2)$. As for Algorithm1, it traverses all flows, finds all possible paths of each flow, and then selects the optimal path from these paths. In the procedure *all_simple_paths*, the complexity of finding a single path for a flow is $O(|N|+|E|)$ and the complexity of finding all the paths is $O((|N|+|E|)|E|/2)$. Since $|E| > |N|$ in our model, $O((|N|+|E|)|E|/2) < O(|E|^2)$. So we can conclude that the complexity of the Algorithm 1 is $O(|F||E|^2)$.

Algorithm 4 Calculate *wgt*

```
  Input: Edge $edge$, Flow $f$
  Output: $Wgt$ for the $f$ on the $edge$
1: if (len($edge.prd\_list$) == 1)) then
2:   $edge.gcd = f_i.prd$;
3: else
4:   $new\_gcd = greatest\_common\_divisor\ (edge.prd\_list)$;
5:   if ($new\_gcd$ != $edge.gcd$) then
6:     $edge.gcd = new\_gcd$;
7:     Recalculate the $edge.SOW$ for all the previous flows on the $edge$;
8:   end
9: end
10: if ($edge.gcd$ == 1) then
11:   wgt = $D$; (D is a large enough integer.)
12: else
13:   $wgt = get\_wgt\ (f)$;
14: end
15: return $wgt$
```

6. Experiments and Evaluation

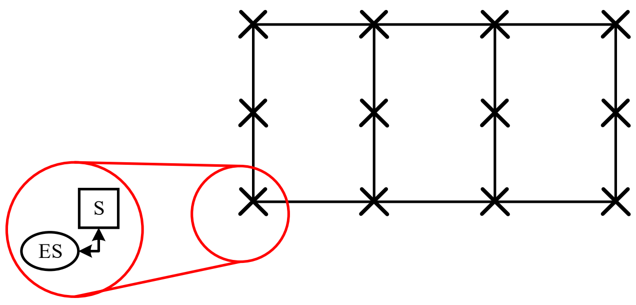
In this section, the evaluation is conducted in terms of the percentage of successfully scheduled flow sets compared to two baseline algorithms. The first algorithm is shortest path routing (denoted as SPR), which is most commonly used in practical applications. The second is a recently published load balanced routing (denoted as LBR) proposed in literature [3]. In order to exclude the influence of irrelevant factors, we evaluated the performance from three dimensions: load, topology and flow type. Based on the method proposed in literatures [6], we built an ILP-based no-wait scheduling platform to evaluate the success rate of scheduling.

6.1. Experiment Setup

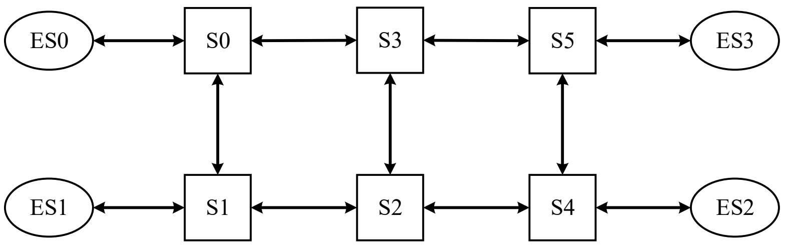
The routing algorithms and scheduling platform were implemented in a Python-based framework, where we created a network and executed the routing algorithm using the NetworkX API and solved the ILP-based scheduling with the Gurobi Optimizer API. The experiments were performed on a PC with an Intel Core i5-8265U processor running at 1.8 GHz and 8 GB RAM.

The topologies used in this article are quoted from literature [6,8]. As is shown in Figure 6a, the topology quoted from literature [8] is a partial mesh topology including 12 end stations and 12 switches. While there are 4 end stations and 6 switches in the topology quoted from literature [6], which have a slightly higher connectivity.

Two groups of candidate flows are presented in Tables 3 and 4. In the first group, the period of each flow is a multiple of 10, which means there is high combinability between every two flows. While there is a big difference in the combinability between two flows and even the *GCD* may be 1 in the second group. The probability of being selected for F1 is relatively low because it has low combinability with other flows. When generating a set of flows, we first specified a group, and then randomly selected flows from it according to their respective probabilities. Besides, the source and destination of each flow were randomly selected. For a single test case, we generated 100 flow sets with the same number of flows.



(**a**)



(**b**)

Figure 6. (**a**) The topology quoted from literature [8] (topology 1); (**b**) the topology quoted from literature [6] (topology 2).

Table 3. The first group of candidate flows (flow group 1).

Flow ID	Prd	Siz	Pro.[1]
F1	10	1	1/6
F2	20	1	1/6
F3	30	2	1/6
F4	40	2	1/6
F5	50	3	1/6
F6	60	3	1/6

[1] The probability to be selected, similarly below.

Table 4. The second group of candidate flows (flow group 2).

Flow ID	Prd	Siz	Pro.
F1	9	1	1/10
F2	10	1	3/10
F3	20	1	3/10
F4	30	2	3/10

6.2. Evaluation Results

A wide range of test cases was used to evaluate our algorithm. We evaluated the performance of the proposed algorithm (PAR) and two baseline algorithms (SPR and LBR) by the scheduling result computed by Gurobi Optimizer. The time limit for scheduling solution was set to 1 min and solution number limit was set to 1, so there were three possible solver outcomes:

- Solved: One sub-optimal schedule found (optimal solution is not pursued in this paper).
- Timeout: Time limit reached without a feasible schedule.
- Infeasible: Cannot be scheduled with the given flows and routes.

6.2.1. Load Dependency

Firstly, three test cases were performed to show the schedulability of the algorithms under different network loads. They used the topology from Figure 6a (topology 1) and flows from Table 3 (flow group 1). We set $K = 0.4$ for PAR here and the result is presented in Figure 7. This figure shows that PAR achieved higher schedulability under different network loads.

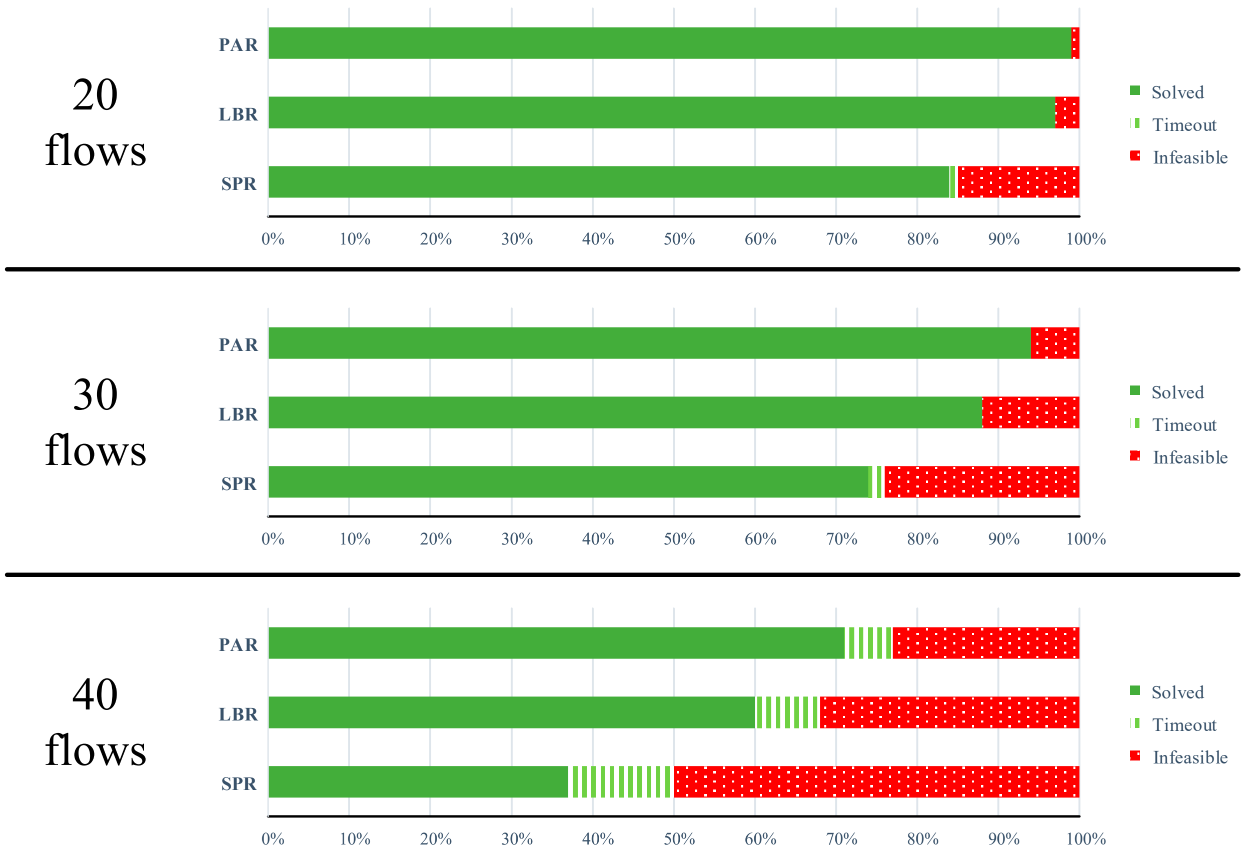


Figure 7. Percentage distribution of solved, timeout and infeasible sets for PAR (period aware routing), LBR (load balanced routing) and SPR (shortest path routing) for different numbers of flows using topology 1 and flow group 1.

Specifically, when there are only 20 flows in the network, flows are not likely to congest, so the advantage of PAR is not obvious. While as the number of the flows increase, the probability of a highly utilized edge causing the infeasibility is growing. In this situation, the benefit of well-designed algorithms to reduce the scheduling bottlenecks is reflected, and the improvement of schedulability is also more obvious. Although the combinability between flows is similar, our algorithm can better reduce the bottleneck than LBR and improve the schedulability by considering the combinability. This suggests that more flows can be accommodated using our algorithm.

6.2.2. Topology Dependency

Secondly, we analyze the schedulability of three algorithms using the topology from Figure 6b to exclude the impact of a specific topology. Flow type is the same as Section 5.2.1 while the flow number is smaller since we used a smaller topology. We set $K = 0.3$ for our algorithm and the result is shown in Figure 8. As shown below, the schedulability of the three algorithms as the number of flows increases is similar to the result presented in Figure 7. These test cases demonstrate that PAR is useful in different topologies. It is worth noting that when the flow number is 20, the number of the timeout sets for PAR and LBR is significantly higher than that of SPR. This is because some edges may not be used for SPR and the complexity of ILP-based scheduling is reduced.

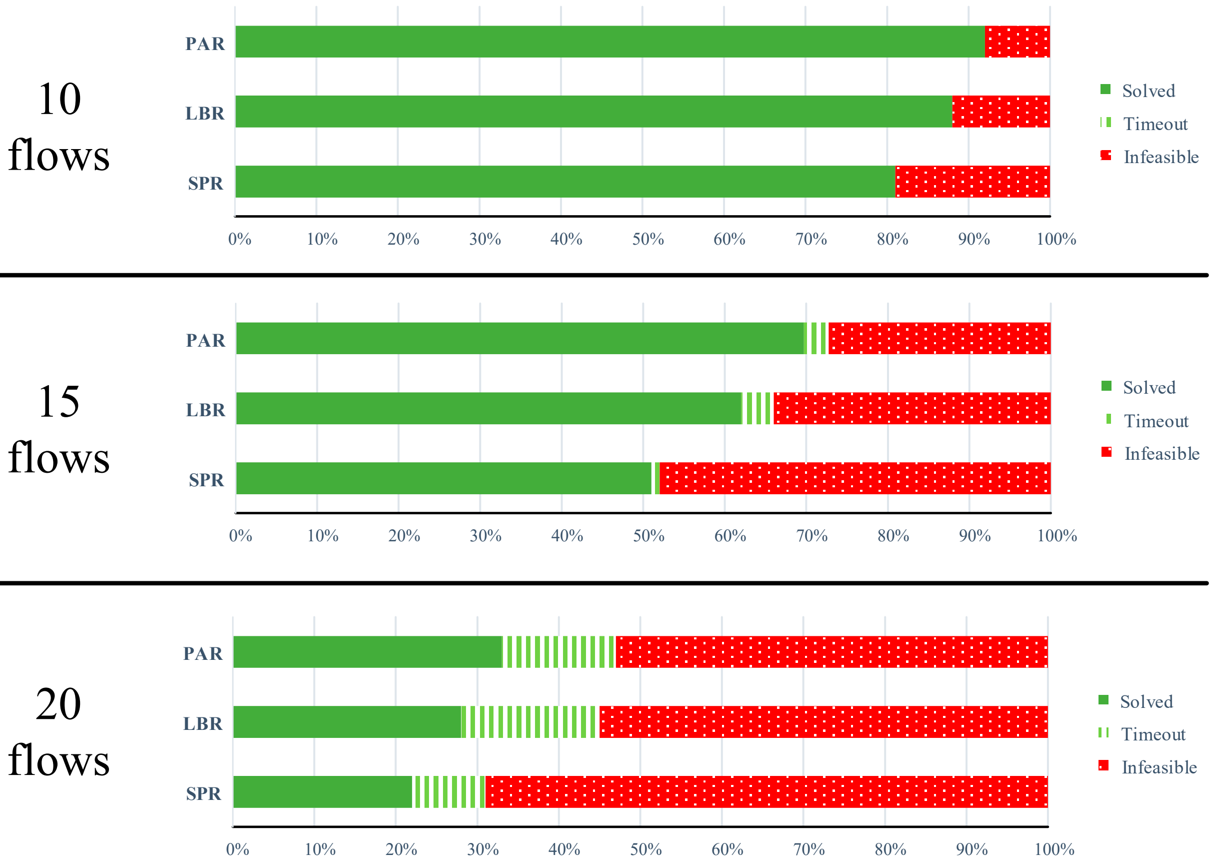


Figure 8. Percentage distribution of solved, timeout and infeasible sets for PAR, LBR and SPR for different numbers of flows using topology 2 and flow group 1.

6.2.3. Flow Type Dependency

Finally, we illustrate the advantages of our algorithm in terms of flow type. Topology 1 and flow group 2 were used in these test cases and we set $K = 0.4$. From Figure 9, we can see that the results of experiment are completely different from the above two. This figure shows that the success rate of LBR and SPR is almost the same while PAR has made a big improvement in this aspect. This is due to the big difference in the combinability between the flows used in these cases. For example, if F1 and F2 of the group 2 are arranged to pass through the same edge, it will directly lead to infeasible situations. In this case, how to make the flow with poor combinability follow disjointed paths, rather than how to optimize the load distribution, becomes the key to the success of scheduling. As LBR only considers load distribution and ignores the combinability between flows, it does not achieve better results than SPR.

In contrast, our algorithm mainly considers combinability in path selection, so flows whose *GCD* is 1 will not share the same edge and the schedulability is doubled. This shows that the proposed algorithm is more adaptable to flows with a variety of periods. It is of practical value under the premise that there are more and more sensors in the system and the types of flows are becoming more and more complex.

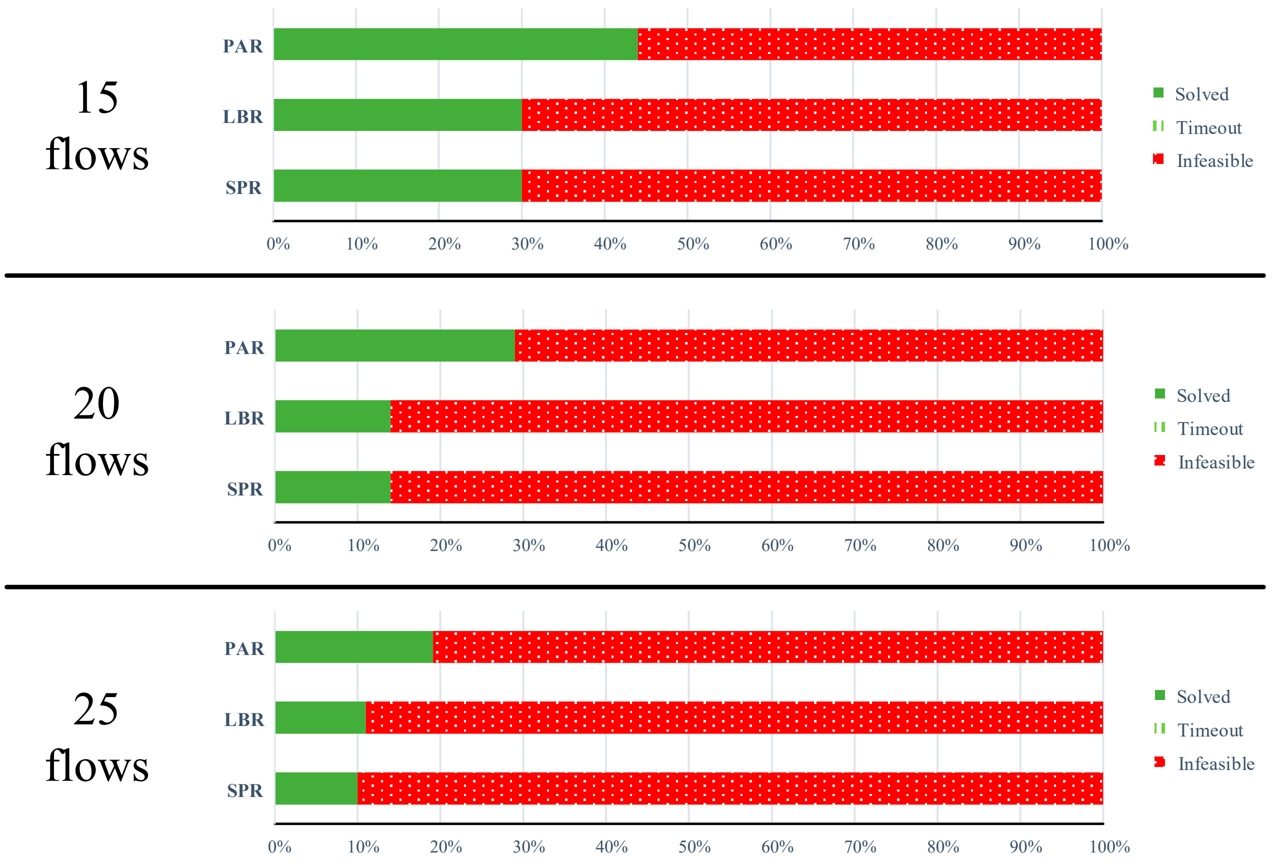


Figure 9. Percentage distribution of solved, timeout and infeasible sets for PAR, LBR and SPR for different numbers of flows using topology 1 and flow group 2.

7. Conclusions

In this article, a novel routing heuristic for time-triggered flows was introduced. We discovered that periodic flows use decentralized bandwidth, and put forward the concept of combinability to express the ability of different streams to use dispersed bandwidth. By analyzing the combinability of two or more periodic flows, we proposed a metric to measure the scheduling bottleneck. Moreover, we found that there is only one coefficient related to the greatest common divisor between our metric and the existing methods, and the great common divisor is an important factor to measure the combinability between flows. We implemented test cases to evaluate the schedulability of our algorithm compared to shortest path routing and a recently proposed load balanced routing. The results were analyzed with respect to three dimensions: load, topology and flow type. Additionally, they show that our method can achieve better schedulability regardless of the load and topology in the case of high combinability between flows, which means that more time-triggered flows can be accommodated. As for the circumstances when combinability between flows is poor, our algorithm doubled the schedulability compared to the existing method, while load balanced routing gained no improvement compared with the shortest path routing. In the case of more and more complex flows in the network, better flow type compatibility will have great practical value.

Still, shortcomings exist in our algorithm. As an empirical value in our method, K should be determined according to the characteristics of the flow and the scale of the topology, and the selection of K will greatly affect the performance of the algorithm. In the future, we would develop heuristics considering the scheduling bottleneck globally rather than thinking about it flow by flow, which can no longer rely on an empirical parameter.

Author Contributions: Conceptualization, J.W. and X.J.; data curation, J.W., W.X. and Y.P.; formal analysis, J.W. and X.J.; funding acquisition, K.H. (Kai Huang); investigation, J.W., D.X., K.H. (Kaitian Huang) and H.Y.; methodology, K.H. (Kai Huang), J.W. and X.J.; resources, K.H. (Kai Huang), X.J. and Z.L.; software, J.W.; supervision, K.H. (Kai Huang) and X.J.; validation, K.H. (Kai Huang), J.W. and D.X.; visualization, K.H. (Kai Huang); writing—original draft, J.W.; writing—review and editing, K.H. (Kai Huang), J.W., X.J. and D.X. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key R&D Program of China (2018YFB0904900, 2018YFB0904902).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy.

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

References

1. Bello, L.L.; Steiner, W. A perspective on IEEE time-sensitive networking for industrial communication and automation systems. *Proc. IEEE* **2019**, *107*, 1094–1120. [CrossRef]
2. 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.
3. Ojewale, M.A.; Yomsi, P.M. Routing heuristics for load-balanced transmission in TSN-based networks. *ACM Sigbed Rev.* **2020**, *16*, 20–25. [CrossRef]
4. Singh, S. Routing Algorithms for Time Sensitive Networks. Master's Thesis, University of Stuttgart, Stuttgart, Germany, 2017.
5. Nayak, N.G.; Duerr, F.; Rothermel, K. Routing Algorithms for IEEE 802.1Qbv Networks. *ACM SIGBED Rev.* **2018**, *15*, 13–18. [CrossRef]
6. Atallah, A.A.; Hamad, G.B.; Mohamed, O.A. Routing and Scheduling of Time-Triggered Traffic in Time-Sensitive Networks. *IEEE Trans. Ind. Inf.* **2019**, *16*, 4525–4534. [CrossRef]
7. Messenger, J.L. Time-sensitive networking: An introduction. *IEEE Commun. Stand. Mag.* **2018**, *2*, 29–33. [CrossRef]
8. Schweissguth, E.; Danielis, P.; Timmermann, D.; Parzyjegla, 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.
9. 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.
10. Steiner, W.; Peón, P.G.; Gutiérrez, M.; Mehmed, A.; Rodriguez-Navas, G.; Lisova, E.; Pozo, F. Next generation real-time networks based on IT technologies. In Proceedings of the 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), Berlin, Germany, 6–9 September 2016; pp. 1–8.
11. Guck, J.W.; Van Bemten, A.; Kellerer, W. DetServ: Network models for real-time QoS provisioning in SDN-based industrial environments. *IEEE Trans. Netw. Serv. Manag.* **2017**, *14*, 1003–1017. [CrossRef]
12. 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. [CrossRef]
13. Avni, G.; Guha, S.; Rodriguez-Navas, G. Synthesizing time-triggered schedules for switched networks with faulty links. In Proceedings of the 13th International Conference on Embedded Software, Chengdu, China, 13–14 August 2016.
14. Bingqian, L.; Yong, W. Hybrid-GA based static schedule generation for time-triggered ethernet. In Proceedings of the 2016 8th IEEE International Conference on Communication Software and Networks (ICCSN), Beijing, China, 4–6 June 2016; pp. 423–427.
15. Tindell, K.W.; Burns, A.; Wellings, A.J. Allocating hard real-time tasks: An NP-hard problem made easy. *Real Time Syst.* **1992**, *4*, 145–165. [CrossRef]
16. Pozo, F.; Rodriguez-Navas, G.; Hansson, H. Schedule reparability: Enhancing time-triggered network recovery upon link failures. In Proceedings of the 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Hakodate, Japan, 28–31 August 2018; pp. 147–156.
17. Zhang, D.G.; Zhang, T.; Dong, Y.; Liu, X.H.; Cui, Y.Y.; Zhao, D.X. Novel optimized link state routing protocol based on quantum genetic strategy for mobile learning. *J. Netw. Comput. Appl.* **2018**, *122*, 37–49. [CrossRef]
18. Al-Turjman, F. Cognitive routing protocol for disaster-inspired internet of things. *Future Gener. Comput. Syst.* **2019**, *92*, 1103–1115. [CrossRef]
19. Pop, P.; Raagaard, M.L.; Craciunas, S.S.; Steiner, W. Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks. *IET Cyber Phys. Syst. Theory Appl.* **2016**, *1*, 86–94. [CrossRef]
20. IEEE. *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 24*; IEEE: New York, NY, USA, 2016.

21. Steiner, W.; Craciunas, S.S.; Oliver, R.S. Traffic planning for time-sensitive communication. *IEEE Commun. Stand. Mag.* **2018**, *2*, 42–47. [CrossRef]
22. 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 2018 IEEE 24th International Symposium on On-Line Testing and Robust System Design (IOLTS), Platja D'Aro, Spain, 2–4 July 2018; pp. 151–156.
23. Gavrilut, 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.
24. Falk, J.; Dürr, F.; Rothermel, K. Exploring practical limitations of joint routing and scheduling for TSN with ILP. In Proceedings of the 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Hakodate, Japan, 28–31 August 2018; pp. 136–146.
25. Yu, Q.; Gu, M. Adaptive Group Routing and Scheduling in Multicast Time-Sensitive Networks. *IEEE Access* **2020**, *8*, 37855–37865. [CrossRef]
26. Hofmann, R.; Nikolić, B.; Ernst, R. Slack-based Traffic Shaping for Real-time Ethernet Networks. In Proceedings of the 2019 IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Hangzhou, China, 18–21 August 2019; pp. 1–11.
27. Falk, J.; Dürr, F.; Rothermel, K. Time-Triggered Traffic Planning for Data Networks with Conflict Graphs. In Proceedings of the 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Sydney, Australia, 21–24 April 2020; pp. 124–136.
28. Arif, F.A.R.; Atia, T.S. Load balancing routing in time-sensitive networks. In Proceedings of the 2016 Third International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T), Kharkiv, Ukraine, 4–6 October 2016; pp. 207–208.
29. Bullynck, M. Modular arithmetic before CF Gauss: Systematizations and discussions on remainder problems in 18th-century Germany. *Hist. Math.* **2009**, *36*, 48–72. [CrossRef]
30. Platt, E.L. *Network Science with Python and NetworkX Quick Start Guide: Explore and Visualize Network Data Effectively*; Packt Publishing Ltd.: Birmingham, UK, 2019.