Time-Triggered Flow Scheduling for Synchronization-Deviation-Tolerant TSN

Jiamin Cui*, Cheng Long*, Zonghui Li*, Rui Wang†, Kang G. Shin‡
*School of Computer Science & Technology, Beijing Jiaotong University, Beijing, China, 100044
†Beijing Smart-chip Microelectronics Technology Co. Ltd, Beijing, China
‡Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, USA
Email: cuicui1938@outlook.com, long.c@bjtu.edu.cn, lizonghui@bjtu.edu.cn,
wangrui1@sgchip.sgcc.com.cn, kgshin@umich.edu

***Abstract*—Time-sensitive networking (TSN) has been widely used in industrial automation and automotive applications by precisely opening and closing the gates of packet queues. However, both clock drift and network congestion will likely result in timing misalignment when the clock synchronization protocol gPTP (802.1 AS) is used. This timing misalignment may, in turn, cause failure in forwarding packets at scheduled times and hence unexpected delay jitters, or even miss application deadlines. To address this acute problem, we propose a novel TSN scheduling algorithm, called `SDT-TSN` (Synchronization-Deviation-Tolerant TSN), to ensure that packets can still arrive on time and be transmitted deterministically even in the presence of inexact time synchronization. First, we formalize the linear constraint model of flow scheduling to maximize the tolerance of inexact time synchronization. Then, we propose an optimal algorithm based on SMT (Satisfiability Modulo Theories) and a fast heuristic algorithm to solve the packet scheduling problem under inexact network synchronization. `SDT-TSN` is the first to derive the maximum tolerable time-synchronization deviation. Finally, we evaluate `SDT-TSN`, demonstrating its capability of eliminating packet-forwarding failures due to the commonly-used/assumed constant time-synchronization deviation and increasing the tolerable synchronization deviation from $140\mu s$ to $480\mu s$.**

***Index Terms*—Time-Sensitive Networking (TSN), Time-Aware Shaper (TAS), Network Synchronization Deviation, Clock Drift**

I. Introduction

In the era of Industry 4.0, various industrial applications, such as factory automation and equipment control, require highly deterministic data transmission services [1]. Such applications must communicate flows of data with precise timing to meet their specific timing requirements. Traditional switched Ethernet lacks deterministic packet-forwarding services due to non-deterministic queueing delays. Time-Sensitive Networking (TSN) [2] has been developed and deployed to meet the stringent requirements for deterministic packet delivery of these applications. By integrating time synchronization and traffic shaping mechanisms, TSN traffic may coexist with other network traffic while ensuring the timely transmission of time-critical traffic.

This work was supported in part by the National Key Research and Development Project under Grant 2022YFB3303702, the National Natural Science Foundation of China under Grant 62221001, Grant 62341127, Grant 62272034. (*Zonghui Li is the corresponding author.*)

The IEEE 802.1 Qbv standard introduces a Time-Aware Shaper (TAS) [3] to provide time isolation for deterministic packet transmission. The core of TAS is the time-aware gating mechanism that controls the transmission gate of a queue. When the gate of a queue is "Open", packets in the queue can be transmitted. When the queue's gate is "Closed", packets must wait until its gate's state changes to "Open". The gate of a packet queue is controlled periodically according to a time-based Gate Control List (GCL). So, TAS requires time synchronization across the entire network including switches and terminals to open and close the gates at scheduled precise time instants [4].

TSN clock synchronization is specified in the IEEE 802.1 AS, which is the generic Precision Time Protocol (gPTP) standardized by IEEE 802.1 TSN Working Group [5]. The gPTP protocol is to designate a node as the master clock in the system to broadcast the reference clock value periodically, and other nodes take the master clock as the root to form a tree structure to receive clock values/signals and align them with the master clock to achieve time synchronization. The master clock in gPTP can be specified by default or dynamically selected by the Best Master Clock Algorithm (BMCA) [6]. Clock drift is the change of a device clock gradually deviating from the master clock time. This deviation can be due to various factors, such as oscillator ageing, changes in temperature, humidity, or atmospheric pressure, which are, by nature, unavoidable. Depending on network load, the time-synchronization packets may be delayed or even lost, thus failing timely compensation for clock drift and causing synchronization accuracy to change dynamically [7].

Conventional TSN scheduling methods usually constrain the time slot with a constant synchronization deviation known *a priori* [8], and transmit packets in different time slots to achieve conflict-free transmission of time-critical data flows. If the constant deviation is set too large, the time slot that can be allocated will be reduced, thus making the flow unschedulable. If it is set too small, the network will not be able to cope with the dynamic change of synchronization deviation, creating flow conflicts. Therefore, how to maximize the tolerable time-synchronization deviation subject to the constraints of network resources and flow requirements is an important problem and hence the focus of this paper.

This paper makes the following contributions:

- **SDT-TSN scheduling model** that maximizes the network's tolerance of time-synchronization deviations.
- **Two solution algorithms**: One is the SMT-based optimal algorithm for static/offline flow scheduling and the other is the heuristic fast scheduling algorithm that can handle a large number (thousands) of flows.
- **Improved tolerance of time-synchronization deviation**: SDT-TSN eliminates packet-forwarding failures of conventional methods that rely on a constant time-synchronization deviation. SDT-TSN is also the first to provide the maximum tolerable time-synchronization deviation. Our evaluation results demonstrate SDT-TSN's increase of the tolerable time-synchronization deviation from $140\mu s$ to $480\mu s$ in various application scenarios.

The rest of this paper is organized as follows. We introduce background and motivations in Section II and present a novel Time-Triggered (TT) scheduling model in Section III. The implementation and evaluation of the proposed algorithms are detailed in Sections IV and V, respectively. Section VI discusses the related work, followed by the conclusion in Section VII.

II. Background and Motivations

A. Basic Terminology

The topology of a network can be formally described as an undirected graph $G(V, E)$, where the terminal devices and switches are vertices V and the links between two physical devices are edges E. The link between two physical devices defines the data link in both directions. Let L denote the set of data links:

$$\forall v_1, v_2 \in V : \{v_1, v_2\} \in E \Rightarrow [v_1, v_2] \in L, [v_2, v_1] \in L.$$

where $\{v_1, v_2\}$ is the undirected physical link and $[v_1, v_2]$ is the directed data link from v_1 to v_2. A sequence of data links forms a datapath which is a directed path from a sender device to a receiver device. A datapath p_i for flow f_i from the sender device v_0 to the receiver device v_{n+1} can be expressed as:

$$p_i = [[v_0, v_1], \ldots, [v_n, v_{n+1}]].$$

The datapath contains n devices/switches, i.e., $v_1, v_2, \ldots, v_n$. v_0 is the sender device and v_{n+1} is the receiver device. Let F denote the set of all flows and f_i be the i-th flow in F. $f_i^{[v_k,v_l]}$ denotes flow f_i transmitted over the data link $[v_k, v_l]$. offset$(f_i^{[v_k,v_l]})$ is the sending time of flow f_i from v_k to v_l. Thus, a flow can be represented as:

$$f_i = \{\text{period}(f_i), \text{length}(f_i)\} \cup \{\text{offset}(f_i^{[v_k,v_l]}) | [v_k, v_l] \in p_i\}.$$

This describes the sending period, the packet length of flow f_i, and its sending time along the datapath in each node.

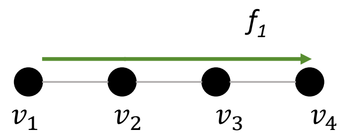


Fig. 1: An example (line) network.

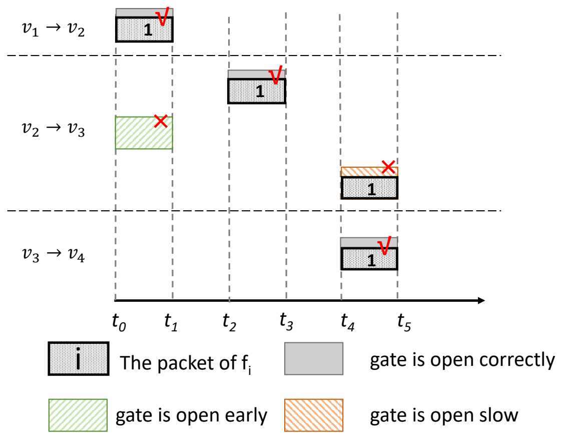


Fig. 2: Examples of TT flow scheduling in the presence of time-synchronization deviation.

B. Motivation

Time synchronization is fundamentally important for deterministic transmission in TSN, and addressing time-synchronization deviation/skew is, therefore, essential for ensuring communication reliability.

Fig. 1 provides an illustrative example with a periodic flow, f_1 sent from v_1 to v_4. Assuming offset$(f_1^{[v_1,v_2]})$=t_0, offset$(f_1^{[v_2,v_3]})$=t_2. If the time-synchronization deviation between v_1 and v_2 is larger than offset$(f_1^{[v_2,v_3]})-$offset$(f_1^{[v_1,v_2]})$, flows cannot be transmitted at scheduled time instants due to untimely opening of gates in the presence of time-synchronization deviation as illustrated in Fig. 2. If v_2 has a faster clock, the gate in v_2 will open earlier at the slot $t_0 \sim t_1$, but during that slot, the packets of f_1 will not arrive at v_2. As a result, f_1 will miss its own slot, and take up the subsequent slot that may have been allocated to other flows. On the other hand, if v_2 has a slower clock, the gate of f_1 will open late at the slot $t_4 \sim t_5$, thus delaying the transmission of f_1 from v_2. Such delayed packets may overflow the limited switch memory or violate the end-to-end (E2E) deadline. So, the time-synchronization deviation will destroy the determinism of time-critical flows.

To maximize the deviation tolerance, we again use flow f_1 to demonstrate between which devices time-synchronization deviations should be considered. Let $\mu^{[v_x,v_y]}$ represent the time-synchronization deviation between v_x and v_y. First, we need to consider the synchronization deviation between two adjacent nodes/devices due to their sequential timing dependencies for forwarding packets, such as $\mu^{[v_1,v_2]}$, $\mu^{[v_2,v_3]}$ in the flow f_1, while $\mu^{[v_3,v_4]}$ need not be maximized because v_4 is a terminal device and does not transmit/forward packets. Second, it is also necessary to maximize the $\mu^{[v_1,v_3]}$ because the E2E delay is calculated by subtracting the sending time of the first hop from that of the last hop, plus the transmission delay on the last hop. In summary, for a certain datapath p_i of any flow from v_0 to v_{n+1}, the time-synchronization deviations that need to be maximized include $\mu^{[v_0,v_1]}, \mu^{[v_1,v_2]} \dots \mu^{[v_{n-1},v_n]}, \mu^{[v_0,v_n]}$.

III. SDT-TSN Scheduling Model

This section details the time-triggered scheduling model of SDT-TSN.

A. Linear Constraints With Time-Synchronization Deviation

We introduce the time-synchronization deviation as a variable in the linear constraints of Steiner's TT scheduling model [9]. The use of $\mu^{[v_x,v_y]}$ allows for a more realistic description of the synchronization deviation between every pair of devices, rather than simply setting it to be a constant as in the prior work. In what follows, we, therefore, introduce the constraints on $\text{offset}(f_i^{[v_x,v_y]})$ and $\mu^{[v_x,v_y]}$.

1) Conflict-Free Constraint : Packets of any two distinct flows f_i and f_j cannot be sent on a link $[v_k, v_l]$ at the same time; a *conflict* will otherwise occur on the link. We are only concerned with the offset of v_k in data link $[v_k, v_l]$, and hence there is no need to consider the synchronization time deviation between nodes/devices. This constraint can be formalized as:

$$\begin{aligned}
&\forall [v_k, v_l] \in L, \forall f_i, f_j \in F,\\
&\forall a \in \left[0 \dots \left(\frac{\text{LCM}(f_i,f_j)}{\text{period}(f_i)} - 1\right)\right], \forall b \in \left[0 \dots \left(\frac{\text{LCM}(f_i,f_j)}{\text{period}(f_j)} - 1\right)\right]:\\
&\left((f_i \neq f_j) \wedge \exists f_i^{[v_k,v_l]} \wedge \exists f_j^{[v_k,v_l]}\right) \Rightarrow\\
&\begin{pmatrix} a \times \text{period}(f_i) + \text{offset}(f_i^{[v_k,v_l]}) \geq \\ b \times \text{period}(f_j) + \text{offset}(f_j^{[v_k,v_l]}) + \text{length}(f_j) \end{pmatrix} \vee\\
&\begin{pmatrix} b \times \text{period}(f_j) + \text{offset}(f_j^{[v_k,v_l]}) \geq \\ a \times \text{period}(f_i) + \text{offset}(f_i^{[v_k,v_l]}) + \text{length}(f_i) \end{pmatrix}
\end{aligned} \tag{1}$$

where $\text{LCM}(f_i, f_j)$ represents the least common multiple (LCM) of the periods of f_i and f_j.

One way to reduce the scheduling complexity is to divide the timeline into equal-sized slots and then schedule a packet per slot. The simplified equation is presented as:

$$\begin{aligned}
&\forall [v_k, v_l] \in L, \forall f_i, f_j \in F,\\
&\forall a \in \left[0 \dots \left(\frac{\text{LCM}(f_i,f_j)}{\text{period}(f_i)} - 1\right)\right], \forall b \in \left[0 \dots \left(\frac{\text{LCM}(f_i,f_j)}{\text{period}(f_j)} - 1\right)\right]:\\
&\left((f_i \neq f_j) \wedge \exists f_i^{[v_k,v_l]} \wedge \exists f_j^{[v_k,v_l]}\right) \Rightarrow\\
&a \times \text{period}(f_i) + \text{offset}(f_i^{[v_k,v_l]}) \neq b \times \text{period}(f_j) + \text{offset}(f_j^{[v_k,v_l]})
\end{aligned} \tag{2}$$

2) Path-Dependency Constraint: For any flow f_i traversing two adjacent links $[v_x, v_j]$ and $[v_j, v_y]$, its packets are always received and then forwarded by node v_j. When there exists a time-synchronization deviation $\mu^{[v_x,v_j]}$, v_j's forwarding time after deducting $\mu^{[v_x,v_j]}$ is still after v_j's receiving time. So, the constraint with $\mu^{[v_x,v_j]}$ is formalized as:

$$\begin{aligned}
&\forall [v_x, v_j], [v_j, v_y] \in p_i :\\
&\text{offset}(f_i^{[v_x,v_j]}) + pdelay^{[v_x,v_j]} + fdelay^{[v_x,v_j]}\\
&\leq \text{offset}(f_i^{[v_j,v_y]}) - \mu^{[v_x,v_j]}
\end{aligned} \tag{3}$$

where $fdelay^{[v_x,v_j]}$ is the forwarding delay at node v_x, and $pdelay^{[v_x,v_j]}$ is the propagation delay on the link $[v_x, v_j]$.

3) Bounded Switch Memory Constraint: In practice, a flow f_i cannot occupy v_j's cache for too long. Hence, we set the maximum time for a flow to occupy v_j's cache as $memboundtime^{[v_j,v_y]}$. In the presence of time-synchronization deviation $\mu^{[v_x,v_j]}$, v_j's forwarding time even after adding $\mu^{[v_x,v_j]}$ must still be within the cache's occupancy time limit. So, the constraint with $\mu^{[v_x,v_j]}$ is formalized as:

$$\begin{aligned}
&\forall [v_x, v_j], [v_j, v_y] \in p_i :\\
&\text{offset}(f_i^{[v_j,v_y]}) + \mu^{[v_x,v_j]} - (\text{offset}(f_i^{[v_x,v_j]}) + pdelay^{[v_x,v_j]}\\
&+ fdelay^{[v_x,v_j]}) \leq memboundtime^{[v_j,v_y]}.
\end{aligned} \tag{4}$$

4) Simultaneous Relay Constraint: In the case of multicast [10], a flow f_i is sent from v_j to multiple links at the same time where we only focus on the sending time from v_j. The time-synchronization deviation does not affect multicast, and hence the constraint is formalized as :

$$\begin{aligned}
&\forall [v_j, v_b] \in p_k, [v_j, v_d] \in p_i :\\
&\text{offset}(f_i^{[v_j,v_b]}) = \text{offset}(f_i^{[v_j,v_d]})
\end{aligned} \tag{5}$$

5) End-to-End Deadline Constraint: The E2E delay cannot exceed the worst-case delay, $\text{deadline}(f_i)$, required by the underlying application. In the presence of time-synchronization deviation $\mu^{[v_x,v_k]}$, v_k's forwarding time even after adding $\mu^{[v_x,v_k]}$ should not exceed $\text{deadline}(f_i)$, and hence the constraint with $\mu^{[v_x,v_j]}$ is formalized as:

$$\begin{aligned}
&\forall [v_x, v_j] \in first(f_i), \forall [v_k, v_l] \in last(f_i), \forall f_i \in F\\
&\text{offset}(f_i^{last(f_i)}) + pdelay^{last(f_i)} + fdelay^{last(f_i)}\\
&- \text{offset}(f_i^{first(f_i)}) + \mu^{[v_x,v_k]} \leq \text{deadline}(f_i)
\end{aligned} \tag{6}$$

where $\text{first}(f_i)$ is the first link in the datapath p_i of flow f_i, and $\text{last}(f_i)$ is its last link.

B. Objective Function

There are two types of variables in the above constraints. $\text{offset}(f_i^{[v_x,v_y]})$ is the correct time of opening a gate to send a packet. $\mu^{[v_x,v_y]}$ is the time-synchronization deviation in which a packet can still be sent through the gate successfully.

We want to tolerate a time-synchronization deviation as large as possible. If we use the sum of deviations as the objective function to maximize the tolerance, it may cause some synchronization deviations to be too large or too small. As a result, those small synchronization deviations will become bottlenecks and will likely lead to the failure of their tolerance. So, we would like to maximize the minimum synchronization deviation as:

$$obj_{val} = max\left\{\min_{[v_x,v_y] \in L}\left\{\mu^{[v_x,v_y]}\right\}\right\} \tag{7}$$

IV. Implementation

We propose two solution algorithms for the TT scheduling.

A. SMT-Based Scheduling

The SMT (Satisfiability Modulo Theories) solver is commonly used to check the satisfaction of first-order logic formulas. It can handle complex logical formulas with boolean variables, functions and predicate symbols, and determine if there is a set of assignments that make the entire formula true.

Each SMT-based scheduling method requires two stages [9]: (a) generating scheduling constraints and adding them to the "logical context" of solver; (b) calling the solver. Therefore, the degree of design freedom of the TT-scheduler is determined by the inter-relationship of these two stages. Our approach is to perform incremental scheduling, frequently alternating phases (a) and (b), dealing with only a portion of the constraints at a time. In particular, the SMT solver we use is the z3-solver [11].

For incremental scheduling instead of generating scheduling constraints for all packets at once, only a small subset is generated at a time and then the SMT solver is called. Each time when the solver returns successfully, the scheduled results from this solution are added to the logical context. In the case of a failed solver's return, we backtrack and increase the size of the subset of flows to be scheduled by merging them with the flows in the previous step. When the SMT solver does not terminate within a given timeout, or the size of the subset grows beyond a pre-defined limit, we abort the scheduling method. The incremental TT scheduler maximizing time-synchronization deviation is described in Algorithm 1 and follows the formal steps as:

(i) Initialize the z3 solver.
(ii) Add the constraints of a batch of flows, set the objective to be maximized, and begin solving the constrained optimization problem.
(iii) If the constraints within this batch can be successfully solved, update the batch to the next one, and store the results in *answers*.
(iv) If the constraints within this batch cannot be solved, take out the results of the previous batch from *answers* and move the current *batch_start* back to merge with the flows in the previous step, re-initializing the z3 solver.
(v) Add *offset* and *time deviation* obtained from the constraints that have been solved so far to the z3 solver, and repeat steps (ii)–(v) until all flows are scheduled or timeout.

The larger the batch size $BatchNum$, the more constraints are added into the SMT solver, and thus the more difficult to solve, the closer it will degenerate into the basic SMT scheduling. The worst case of the incremental algorithm is the basic SMT scheduling that solves all constraints of flows at once. The SMT-based incremental algorithm is optimal, but since the problem is NP-complete, it is necessary to set a timeout for the incremental algorithm. Such an optimal algorithm is suitable for offline scheduling.

B. Heuristic Scheduling Algorithm

Since the SMT-based scheduling scheme has an inherent performance bottleneck, we propose a heuristic scheduling algorithm for a large number (thousands) of flows.

Algorithm 1: TT-SMT-Tolerance

```
input  : FlowSet $F$,BatchNum $num$
output: Offset,Clock Deviation
$answers$ ← empty stack;
$batch\_start$ ← 0;
$batch\_end$ ← $batch\_start$ + $num$;
$solver$ ← z3.solver();
while not all flows are scheduled Or not timeout do
    add_constraints(batch_start,batch_end,
    scheduled_flows);
    $scheduled\_flows$ ← F[0:$batch\_end$ ];
    $solver$.maxmize($get\_aim\_val$);
    if solver.check() == z3.sat then
        $answer$ ← solver.model();
        Push $answer$ to $answers$;
        $batch\_start$ ← $batch\_end$;
        $batch\_end$ ← $batch\_end$ + $num$;
    else
        $answers$.pop;
        $answer$ ← $answers$.top;
        $batch\_start$ ← $batch\_start$- $num$;
        if $batch\_start < 0$ then
            Exception(schedule error!)
        $solver$ ← z3.solver();
        Delete $scheduled\_flows$[$batch\_start$:]
    for decl in answer.decls() do
        symbol ← decl.name();
        result ← $answer$ [decl];
        if symbol is about Offset then
            add_constraint (symbol==result);
        if symbol is about Time Deviation then
            add_constraint (symbol≤result);
```

For each flow, the total amount of time that can be deferred is its E2E deadline minus its transmission delay. To maximize the minimum time-synchronization deviation, we need to evenly divide and distribute the total amount of time that can be deferred over all the links of the flow.

1) Definition of Clock Tolerance Window: To tolerate the time-synchronization deviation, we define a *Clock Tolerance Window* (CTW) at each hop/link as the amount of time of a flow that can be deferred without violating its E2E deadline. In general, the CTW is composed of two parts: (i) the evenly distributed delay margin across all hops (ECTW), and (ii) additional slack time gained from underutilized clock windows at previous hops (Untapped). ECTW of f_i, denoted as $\mathrm{ectw}(f_i)$ is defined as:

$$\mathrm{ectw}(f_i) = \frac{\mathrm{deadline}(f_i) - \mathrm{delay}(f_i)}{|p_i|}, \tag{8}$$

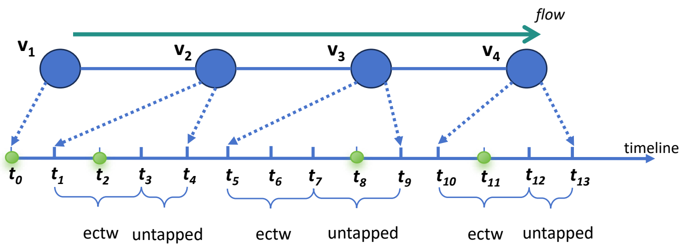


Fig. 3: The evolution illustration of *untapped* values across hops.

where $\text{delay}(f_i)$ is the minimum E2E transmission delay of f_i:

$$\text{delay}(f_i) = \sum_{[v_i,v_j]\in p_i} (pdelay^{[v_i,v_j]} + fdelay^{[v_i,v_j]}). \quad (9)$$

Thus, the actual CTW on each hop $[v_k, v_l] \in p_i$ is computed as:

$$\text{ctw}(f_i^{[v_k,v_l]}) = \text{ectw}(f_i) + \text{untapped}(f_i^{[v_k,v_l]}), \quad (10)$$

where $\text{untapped}(f_i^{[v_k,v_l]})$ represents the cumulative unused time from the source of f_i up to node v_k. This CTW is recursively updated based on the remaining time in the synchronization window from the previous hop, as illustrated in Fig. 3. The difference in the CTW across hops primarily depends on the evolution of the untapped clock window.

To illustrate the untapped CTW, let us consider a flow whose offset on hop $[v_1, v_2]$ is set to t_0. Suppose the ECTW is 2 slots and the initial untapped value is also 2 slots. At each hop, the untapped value is updated according to the portion of the CTW that was not consumed at the previous hop. For example, if the transmission time at node v_2 is scheduled at t_2, and its allocated CTW spans from t_1 to t_4, then 2 slots (t_2 to t_4) remain unused, and hence the untapped value for the next hop, v_3, is updated to 2. Similarly, if the transmission time at v_3 is set to t_8, and the CTW at v_3 ranges from t_5 to t_9, then only 1 slot remains unused (t_8 to t_9), and the untapped value for node v_4 is updated to 1.

In summary, as long as each node defers the transmission within its allocated CTW, the accumulated delay will not violate the E2E deadline constraint of a flow. This mechanism also maximizes the minimum time-synchronization deviation between devices. For any flow f_i, the maximum theoretical value of the minimum tolerable time-synchronization deviation is $\text{ectw}(f_i)$.

2) Offset Allocation based on CTW: For each flow f_i, the sending time instant at the first hop $[v_0, v_1]$ is initialized to 0. For the subsequent hops, suppose the offset of the previous hop $[v_x, v_k]$ has already been determined to be $\text{offset}(f_i^{[v_x,v_k]})$. Then, the earliest possible sending instant at the next hop $[v_k, v_l]$ can be computed as:

$$\text{offset}_{\text{st}}(f_i^{[v_k,v_l]}) = \text{offset}(f_i^{[v_x,v_k]}) + pdelay^{[v_x,v_k]} + fdelay^{[v_k,v_l]}. \quad (11)$$

The latest allowed sending instant $\text{offset}_{ed}(f_i^{[v_k,v_l]})$ is determined by the size of CTW at the current hop:

$$\text{offset}_{\text{ed}}(f_i^{[v_k,v_l]}) = \text{offset}_{\text{st}}(f_i^{[v_k,v_l]}) + \text{ctw}(f_i^{[v_k,v_l]}) \quad (12)$$

Algorithm 2: Hop-Based Scheduling

```
input : Hop $h$,FlowSet $F$
output: Schedule Table
1  $flows \leftarrow get\_flows\_on\_hop\ (F,h)$;
2  $compute\_ctw(flows)$;
3  $sort\_with\_ctw(flows)$;
4  for $f$ in $flows$ do
5      $h\_pre \leftarrow get\_hop\_pre\ (f,h)$;
6      $\text{offset}_{\text{st}} \leftarrow$ f[$h\_pre$].offset+1;
7      $\text{offset}_{\text{ed}} \leftarrow \text{offset}_{\text{st}}$ + $get\_ctw\ (f,h)$;
8      for $o$ = $\text{offset}_{\text{ed}}$ to $\text{offset}_{\text{st}}$ do
9          f[$h$].offset $\leftarrow o$;
10         if $f$ is not conflicts with lower-ctw flow then
11             break;
12     if $f$ is out of constraints then
13         Set $f$ to be $unschedulable$;
14     else
15         f[untapped] $\leftarrow \text{offset}_{\text{ed}}$ - f[$h$].offset
16 return Schedule Table
```

Therefore, the valid range for assigning the sending offset at hop $[v_k, v_l]$ lies between $\text{offset}_{\text{st}}$ and $\text{offset}_{\text{ed}}$. To achieve the objective of maximizing the minimum synchronization deviation, the available CTW at each hop should be utilized as fully as possible. Thus, for each hop, we search backward within the offset range to find the latest available valid time slot (i.e., a slot not occupied by other flows), and assign it as the transmission offset.

Since multiple flows may traverse the same hop, the order of scheduling these flows also impacts the utilization of available slots. Those flows scheduled later will have fewer remaining valid time slots. Therefore, to minimize conflicts and improve overall schedulability, we prioritize flows with smaller CTWs, as they are more constrained and benefit from being scheduled earlier. The detailed hop-level scheduling procedure is described in Algorithm 2.

3) Hop Order Decisions Based on CTW: To determine the transmission offsets and CTWs for flows traversing a given hop $[v_k, v_l]$, we must first identify the order in which hops are scheduled. This order depends on the relative timing dependencies between hops and can be represented using a precedence diagram. Based on this diagram, we apply topological sorting [12] to derive a valid scheduling sequence, enabling hop-based scheduling to proceed in a way of satisfying dependencies.

Flow Precedence Diagram. For each flow $f_i \in F$, suppose its routing path is $p_i = [[v_0, v_1], \ldots, [v_k, v_l], [v_l, v_m], \ldots, [v_{j-1}, v_j]]$. We construct a directed graph $G'(V', E')$ based on the routing paths of all flows in F. The graph is constructed as follows:

(i) Each hop $[v_x, v_y]$ in the path of a flow is represented as a vertex $v' \in V'$ in the precedence graph and v' can be

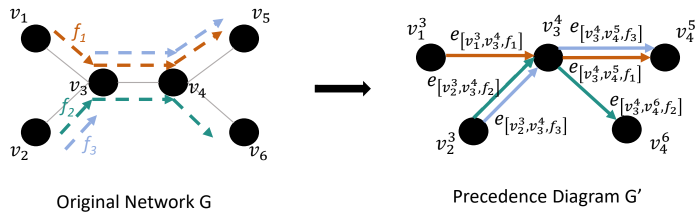


Fig. 4: Construction of a precedence diagram.

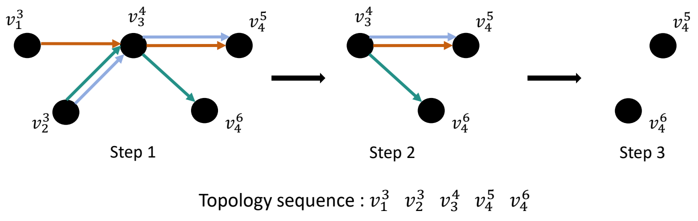


Fig. 5: The topological sorting of precedence diagram.

represented as v_x^y.

(ii) For any two consecutive hops of a flow, such as $v'_a = [v_k, v_l]$ and $v'_b = [v_l, v_m]$, a directed edge $e' = [v'_a, v'_b] \in E'$ is added to indicate that v'_a must be scheduled before v'_b.

Fig. 4 illustrates an example of a thus-constructed precedence diagram.

Hop Topological Sorting. A directed edge $[v_x, v_y]$ in the precedence diagram G' indicates that the scheduling of hop v_y depends on the completion of hop v_x. In other words, v_y can only be scheduled after v_x.

We employ topological ordering to resolve the path dependency constraints while scheduling flows. At each step of topological sorting, we select a node $v' \in V'$ with no incoming edges (i.e., no unscheduled predecessor). Each such node corresponds to a hop in the original flow graph and is scheduled using the hop-based scheduling described in Algorithm 2. The overall topological scheduling based on the precedence diagram is illustrated in Fig. 5.

Loop Breaking. In order to obtain a complete topological sequence from topological sorting, no loops can appear in the precedent diagram, and if loops appear, we need to break them.

Fig. 6 shows an example loop in a precedence diagram, where the first step represents the original graph information, the second step constructs the precedence graph, the third step schedules the node with a 0 in-degree, and the fourth step breaks the loop.

There may be more than one loop in the precedence diagram, so there may be multiple loop-breaking operations, each of which will select a certain edge, and the flows involved in that edge will be marked as loop-broken. In Fig. 6, selecting $e'[v_1^2, v_2^3]$ and $e'[v_2^3, v_3^9]$ to break, f_1 will be marked as loop-broken. Loop-breaking becomes successful when a complete topological sequence can be obtained.

For loop-breaking, we need to (1) consider the number of flows to be used for loop-breaking and (2) maximize the minimum time-synchronization deviation (indicated by the CTW value). For (1), we use as few flows for loop-breaking as possible to reduce the scheduling overhead. For (2), we give

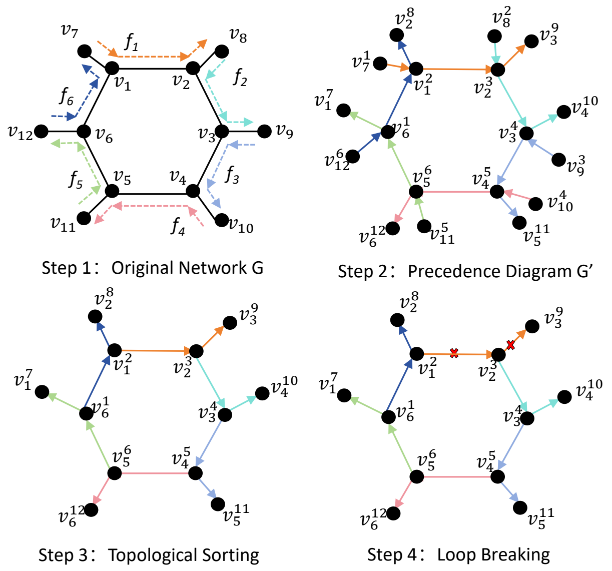


Fig. 6: An example of loop breaking.

priority to a large CTW value for loop-breaking, because the broken edges will be scheduled at the end. When the broken edges begin to be scheduled, the majority of available time slots have already been assigned to schedulable flows. The flow with a large CTW is easier to schedule and will make less impact on maximization of the minimum time-synchronization deviation. So, we synthesize (1) and (2) as:

$$cost = \underset{[h_k,h_l]\in G'(E')}{\operatorname{argmin}} \left\{ \omega \cdot |F_{h_k,h_l}| - (1-\omega) \cdot \min_{f_i \in F_{h_k,h_l}} (\text{ctw}(f_i)) \right\}$$
$$F_{h_k,h_l} = F \cap F^{h_k} \cap F^{h_l}.$$

where F_{h_k} is the set of flows that go through hop h_k, F_{h_k,h_l} is the set of flows that go through both h_k and h_l, and ω indicates the weights of the two loop-breaking factors mentioned above. Selecting the edge with the smaller *cost* is preferred to break loops.

4) Analysis of Time Complexity: Algorithm 3 outlines our heuristic strategy to maximize the minimum time-synchronization deviation. The formal steps are as follows:

(i) Construct the precedence diagram $G'(V', E')$.

(ii) Begin topological sorting, adding vertices v' with 0 in-degree to Q.

(iii) If Q is empty, representing the existence of a loop in G', then execute a loop-breaking operation, add loop-broken flows to S, and update G'.

(iv) If Q is not empty, begin hop-based scheduling with v' in Q, and then delete v' and the directed edge e' starting from v' in G'.

(v) Repeat steps (ii) - (iv) until all vertices are scheduled.

(vi) Schedule the loop-broken flows.

The time complexity of this procedure is analyzed as follows. The number of vertices in G' is $|E|$, since the two adjacent edges of each flow have one edge, the number of edges is $|E| \cdot |F|$. So, the construction of precedence diagram has a time complexity of $O(|F| \cdot |E|)$ and the topological sort

Algorithm 3: TT-Fast-Tolerance

```
input : Network $G$,FlowSet $F$
output: Schedule Table
1  $Q \leftarrow$ empty FIFO queue;
2  $S \leftarrow$ empty stack;
3  $D_{in} \leftarrow$ empty array;
4  $G'(V',E') \leftarrow CreatePreGraph$ ($G$,$F$);
5  for $v' \in V'$ do
6      if $D_{in}$ [$v'$]=0 then
7          Add $v'$ to $Q$;
8  while not all V' are scheduled do
9      while $Q$ is empty do
10         $e' \leftarrow LoopBreaking$ ($E'$);
11         Push $e'$ to $S$
12     while $Q$ is not empty do
13         $v' \leftarrow Q$.remove;
14         Update $D_{in}$;
15         Update $Q$ with $D_{in}$ =0;
16         $HopBasedScheduling$ ($v', F$);
17 while $S$ is not empty do
18     $e' \leftarrow S$.pop;
19     $LoopBreakingScheduling(e')$
20 return Schedule Table;
```

requires $O(V' + E')$ = $O(|F| \cdot |E|)$. Then, we begin hop-based scheduling: for each node v' corresponding to a hop in the original graph, the algorithm calculates the CTW of flows and sorts flows by CTW with $O(|E| \cdot (|F| + |F| \cdot log(|F|))$ = $O(|E| \cdot |F| \cdot log(|F|))$. Scheduling them in the range of size CTW and resolving any conflicts require a computation complexity of $O(|F| \cdot CTW \cdot |F| \cdot \frac{LCM(f_a,f_b)}{slot}))$ = $O(|F|^2 \cdot CTW \cdot \frac{LCM(f_a,f_b)}{slot})$, where $\{f_a, f_b\} = \arg\max_{f_i,f_j \in F, i \neq j} LCM(f_i, f_j)$. Finally, we manage loops in the precedence diagram with $O(|F| \cdot |E|)$ to find which edge to break. So, the TT-Fast-Tolerance algorithm maximizes the minimum time-synchronization deviation with a polynomial-time complexity.

V. Evaluation

A. Simulation Setup

1) Topology Selection: We consider two typical classes of topologies. One is the basic topologies of industrial control networks, including linear, ring, and snowflake, as shown in Figs. 7a, 7b, and 7c. In a linear topology, traffic flow can traverse in both directions, while a ring topology restricts traffic to either clockwise or counter-clockwise direction. The maximum hop count is 15 for both linear and ring topologies. The snowflake topology has the same port count for all but edge switches, and the maximum hop count is set to 17. The other is the complex topologies, e.g., the typical Orion Crew Exploration Vehicle (CEV) [13]. It employs a hybrid network topology, combining elements of star, tree, and ring structures

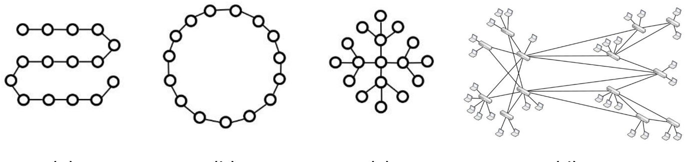


Fig. 7: Network topologies used in evaluation.

as shown in Fig. 7d. This network interconnects a total of 31 end-devices and 13 switches.

2) Resource Setting: The transmission delay over a data link consists of forwarding delay and propagation delay which is defined as $20\mu s$. The maximum packet length is configured to be 1518 bytes, and the network link operates at a line speed of 1000 Mbps. In switches, the limit of the bounded cache, $memboundtime$ is set to $280ms$. We set the $slot$ size to $20\mu s$ that equals to the transmission delay and the $timeout$ to $1000s$.

3) Flow Requirements: All TT flows in the simulation are randomly generated and follow the IEC/IEEE 60802 industrial automation networking requirements [14]. For each TT flow, a source terminal is randomly selected from the set of all terminals in the network. A destination terminal is then randomly selected from the remaining terminals. Packet lengths are chosen randomly between 64 and 1518 $bytes$. The traffic cycle is randomly selected from $\{2ms$, $4ms$, $8ms$, $16ms$, $32ms$, $64ms$, $128ms$, $256ms$, $512ms\}$. The E2E deadline for each flow is randomly selected between $2ms$ and $256ms$. The E2E deadline should be less than, or equal to the traffic cycle.

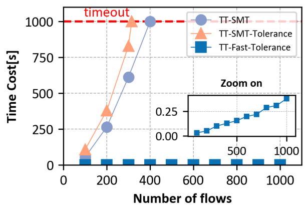
4) Compared Methods:

- TT-SMT: A traditional scheduling scheme that models the scheduling constraints as Satisfactory Modulo Theory (SMT) problems and uses the Z3-solver to solve them.
- TT-SMT-Tolerance: SMT scheduling that maximizes the tolerance for time-synchronization deviation and was introduced in Section IV-A.
- TT-FAST-Tolerance: Fast scheduling using heuristic strategies to maximize the synchronization-deviation tolerance and was described in Section IV-B.

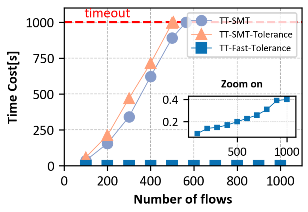
B. Comparison of TT Scheduling Algorithms

We evaluate the traditional scheduling schemes and those that maximize the tolerance of time-synchronization deviation in terms of time cost, runnable flows and the value of the tolerable time-synchronization deviation.

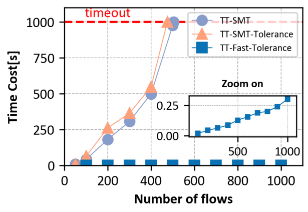
Fig. 8 shows the time cost on different topologies. TT-SMT and TT-SMT-Tolerance are easier to hit a timeout. Within $1000s$, TT-SMT can schedule 400, 564, 504, and 700 flows in linear, ring, snowflake, and CEV topologies, respectively, while TT-SMT-Tolerance can schedule 315, 505, 475, and 850 flows respectively. In the CEV topology, a larger number of flows can be scheduled within the timeout period thanks to the network's extended scale that inherently reduces the probability of flow collisions compared to other topologies. Furthermore, TT-SMT-Tolerance demonstrates superior performance to TT-SMT despite the additional computational overhead



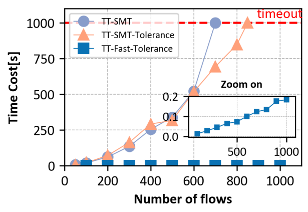
(a) Linear topology.



(b) Ring topology.

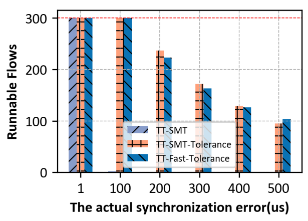


(c) Snowflake topology.

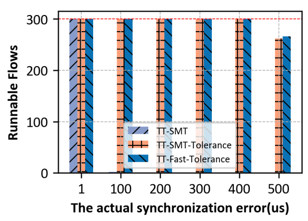


(d) CEV topology.

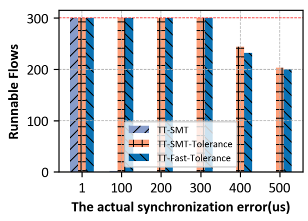
Fig. 8: The time cost on different topologies.



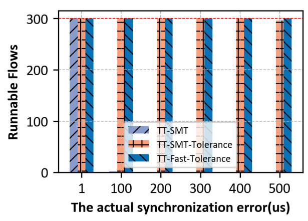
(a) Linear topology.



(b) Ring topology.

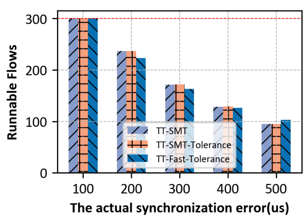


(c) Snowflake topology.

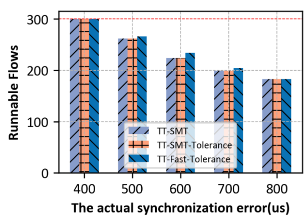


(d) CEV topology.

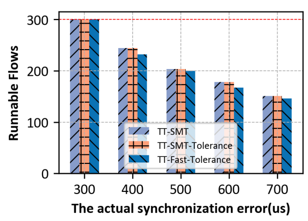
Fig. 9: The runnable flows with actual time deviations on different topologies with 300 flows. The constant deviation is set to $1\mu s$ for TT-SMT.



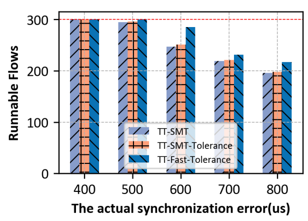
(a) Linear topology.



(b) Ring topology.



(c) Snowflake topology.



(d) CEV topology.

Fig. 10: The runnable flows with actual time deviations on different topologies with 300 flows. The constant deviation is set to the optimal objective value for TT-SMT.

introduced by the variable $\mu^{[v_x,v_y]}$ because more even offset selection by maximizing the minimum time-synchronization deviation reduces backtracking operations significantly. Consequently, TT-SMT-Tolerance achieves faster execution than TT-SMT when handling traffic loads of >600 flows. However, when the load is higher than 850 flows, TT-SMT-tolerance will time out. But, TT-Fast-Tolerance achieves superior scheduling efficiency and can schedule 1000 flows within $0.5s$ on any of the tested topologies.

Fig. 9 shows the number of runnable flows. For TT-SMT, we use $1\mu s$ as the constant synchronization deviation, because the synchronization process of gPTP ensures that the synchronization error is kept within $1\mu s$ in a network with the theoretic maximum hop count of 7. But, in reality, the hop count may be larger than 7 and the frequency of time synchronization may be smaller. Assuming the real synchronization accuracy is selected from $\{1\mu s, 100\mu s, 200\mu s, 300\mu s, 400\mu s, 500\mu s\}$, when the actual synchronization deviation exceeds the objective function value shown in Table I, the number of runnable flows will decrease. As illustrated in Fig. 9, TT-SMT can run 0 flows when the real synchronization accuracy is larger than $1\mu s$. But both TT-SMT-Tolerance and TT-Fast-Tolerance can ensure that most of the flows are runnable even if the time-synchronization deviation is up to $500\mu s$ because they both maximize the tolerance of time-synchronization deviation. Besides, they demonstrate comparable performance in terms of runnable flows, as both methods achieve similar levels of tolerable time synchronization accuracy. Table I presents the optimal and heuristic values of tolerable time-synchronization deviation. TT-Fast-tolerance is close to the optimal value of TT-SMT-tolerance.

Furthermore, we set the constant time-synchronization deviation of TT-SMT to a higher value, e.g., setting it to the optimal tolerable values $\{156.83\mu s, 441.33\mu s, 322.21\mu s, 492.86\mu s\}$ in Table I for linear, ring, snowflake, CEV topologies, respectively. With this setting, Fig. 10 shows the number of runnable flows using different algorithms. TT-SMT has a very similar number of runnable flows to that of our two algorithms, but we cannot predict the optimal objective function value and set it to TT-SMT without using `SDT-TSN`. So, we still rely on the commonly used or experience-based time-synchronization deviation, typically $1\mu s$, to use TT-SMT, leading to the failure of many flows as illustrated in Fig. 9.

These results demonstrate that `SDT-TSN` can eliminate flow failures caused by the commonly-used/experience-based time-synchronization deviation.

C. Simulation

We have built an OMNeT++ simulation environment with the INET 4.4.1 framework [15] and simulated PTP packets interfered with BE (Best-effort) traffic in the network. Table II shows the simulation setup on OMNeT++. The maximum time-synchronization deviation between any two devices occurs just before their resynchronization when the time synchronization is done correctly. So, $\Delta T = |\text{drift}(v_a) - \text{drift}(v_b)| \times T_{sync}$, and thus the estimated time-synchronization deviation is $500\mu s$ according to Table II.

Each egress port maintains three priority queues that correspond to different traffic classes. Incoming packets are classified and placed into these queues based on their Priority Code Point (PCP) using the flow identification specified in IEEE 802.1CB [16]. TT flows are assigned to the highest-priority queue. The other two priorities are assigned to BE and PTP flows using different PCP values. We evaluate both cases when PTP flows are given higher priority than BE flows, and vice versa. When the priority of PTP flows is higher than that of BE flows, the time synchronization can be completed correctly, so the time deviation between devices will be corrected in time. Fig. 11 shows that the affected flows of both TT-SMT and TT-SMT-Tolerance remain unchanged, because the time-synchronization deviation is always less than $500\mu s$. If the priority of PTP flows is lower than that of BE flows, timely synchronization becomes unachievable. This leads to time-synchronization deviations exceeding $500\mu s$, the deviation will become larger as the BE bandwidth increases. So, the number of flows affected increases with the increase of BE traffic. But in both cases, TT-SMT-Tolerance has fewer flows affected by the time-synchronization deviation caused by BE traffic, than TT-SMT. That is, TT-SMT-Tolerance ensures that more flows are sent in their scheduled time slots even in the presence of inexact time synchronization because TT-SMT-Tolerance maximizes the tolerance for time-synchronization deviation.

The above results demonstrate that our `SDT-TSN` maintains deterministic transmission for most of TT flows, under the imperfect synchronization caused by the background traffic.

TABLE I: The optimal objective value(μs) of the tolerable time-synchronization deviation on different topologies with 300 flows.

Topology	Linear	Ring	Snow	CEV
TT-SMT	1	1	1	1
TT-SMT-Tolerance	156.832	441.33	322.21	492.86
TT-Fast-Tolerance	140	420	300	480

TABLE II: Simulation Setup

Topology	CEV
Number of Flows	30
Clock Drift	[-500ppm,500ppm]
Synchronization Interval	500ms

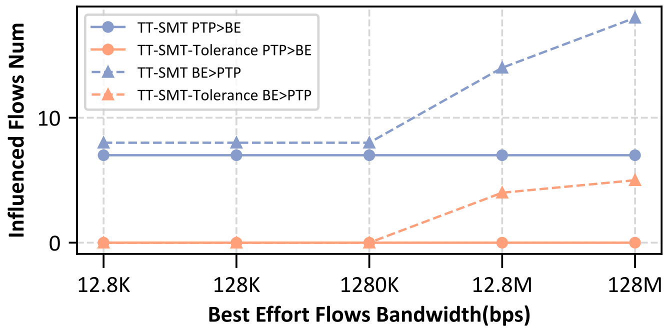
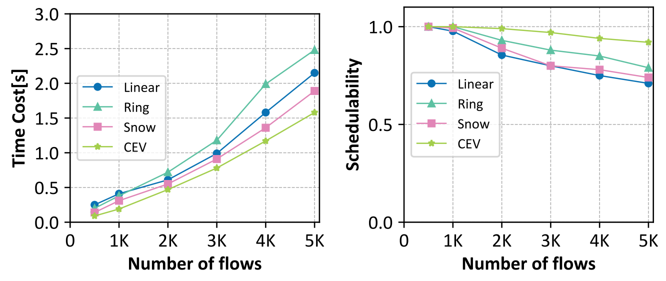


Fig. 11: The number of flows that are affected under different background traffic.



(a) Time cost. (b) Schedulability.

Fig. 12: The performance of TT-Fast-Tolerance on different topologies.

D. Scalability of TT-Fast-Tolerance

We randomly generate multiple sets of flows (each set with thousands of flows) on different topologies to evaluate the scalability of the TT-Fast-Tolerance.

Fig. 12 illustrates the scheduling results of TT-Fast-Tolerance as the number of flows increases from 0 to 5000. In terms of schedulability, it achieves full scheduling for up to 3550, 3950, 3700, 4600 flows in linear, ring, snowflake, and CEV topology, respectively, with the scheduling time cost no more than $2.5s$. The rising trend in time cost with increasing traffic is polynomial-time and is similar to linear-time as illustrated in Fig. 12a. This demonstrates the high efficiency of TT-Fast-Tolerance in handling large-scale traffic.

VI. Related Work

TSN depends on time-synchronization to precisely open and close the gates of different packet queues, and thus ensures deterministic transmission. So, most of works focus on scheduling TT flows to generate the gate configuration, and improving time synchronization for accuracy and reliability.

A. Scheduling TT flows

[9] pioneered the use of SMT solvers to schedule transmission time slots for TT flows along their paths. However, it faces scalability issues in large networks due to NP-Completeness of TT scheduling. The IEEE 802.1Qch standard simplifies TAS by introducing the CQF model, employing two ping-pong queues that alternate periodically for data transmission, instead of gate control of eight queues. [17] translated the

CQF model into multiple constraints and proposed a heuristic scheduling algorithm based on Injection Time Planning (ITP) for resource allocation to TSN flows. [18] proposed a scheduling algorithm based on deep reinforcement learning, which combines the Markov Decision Process (MDP) [19] with Deep Neural Networks (DNN). It uses DNN [20] to make routing and time-slot allocation decisions in the MDP, enabling dynamic scheduling of thousands of TT flows in several hundred seconds. Recently, TSN and 5G are moving towards a converged network. [21] presents a novel way to integrate 5G and TSN by a Double Q-learning based hierarchical particle swarm optimization algorithm (DQHPSO).

These previous scheduling approaches assumed that time synchronization was established well. They considered time-synchronization deviation as constant, typical value $1\mu s$ according to IEEE 802.1AS [5], and aim to address such issues as load-balancing, schedulability, or efficiency of TT flows, without addressing dynamic time-synchronization deviations. When the dynamic deviations are larger than the commonly-used/assumed constant, the deterministic transmission of TT flows will fail due to the timing misalignment of gates.

B. Improving Time Synchronization

On the one hand, to improve the reliability of time synchronization, IEEE 1588 [22] broadcasts its sync messages to maintain time synchronization as long as a network is connected. When a master clock fails, the IEEE 1588 suggests use of the BMC algorithm to select a new master clock. However, it is time-consuming to re-establish time synchronization. IEEE 802.1AS-rev [5] decreases the re-establishment latency by setting up multiple time domains based on multiple master clocks. Upon detection of the failure of a master clock, a device immediately selects the next time domain via register configuration. Besides, to cope with arbitrary abnormal actions of a device such as generating random-content messages, the aerospace standard AS6802 [23] is designed to support both single- and dual-failure cases.

On the other hand, to improve the accuracy of time synchronization, IEEE 1588 [22] suggests adjusting the frequency offset or using a better oscillator such as an oven-controlled crystal oscillator (OCXO) to reduce clock drift between the intervals of time synchronization. However, adjusting frequency offset consumes resources and an OCXO is too expensive to use in industrial embedded devices. [24] designed a Kalman filter-based cooperative synchronization tracking algorithm to accurately tracks both clock offsets and drifts. [25] used a neural network model to reduce the impact of node clock drift on TSN time synchronization. [26] proposed a software-defined time synchronization framework, so-called *Network Clock Ensemble* to obtain stable time accuracy by a synergistic effect between clocks in a packet network. Recently, in the converged network, [27] enhanced the accuracy of 5G-TSN time synchronization by overcoming multi-gNB contention and mobility challenges in industrial environments.

These works improve synchronization accuracy or reliability, and thus alleviate time-synchronization deviation, but when the dynamic deviation is greater than the commonly used/assumed constant, the deterministic transmission will fail. In other words, they still do not consider how to tolerate the residual deviation at the scheduling level.

C. Joint TT scheduling and Time Synchronization

Few existing studies consider both TT scheduling and time synchronization together. [28] analyzes the impact of time-synchronization deviations on TT scheduling by deriving tight E2E latency bounds, enabling performance evaluation under clock drifts. However, it did not propose any scheduling solution to the synchronization-deviation problem. In contrast, `SDT-TSN` proactively tolerates such deviations through a practical scheduling scheme that ensures deterministic transmission. While [28] provides a theoretical analysis, `SDT-TSN` makes an important, complementary contribution with a concrete solution. [29] first assumed the worst-case time-synchronization deviation as a constant, and then based on the assumed constant, designed a shadow gate queue as a supplementary of a regular gate queue to receive the packets impacted by the time-synchronization deviation, which is called the *robust TAS* (RTAS). The constant deviation decided the length and time-slot co-configuration of the shadow queue with its regular queue. So, RTAS can only tolerate the assumed synchronization deviation and will cause the overhead of the complex gate co-configuration and management. That is, RTAS still considered the time-synchronization accuracy as a commonly-used/assumed constant. Such a design requires to modify the hardware design and thus cannot be used for the off-the-shelf TSN devices. Furthermore, how to get the worst-case accuracy remains an open problem. In contrast, `SDT-TSN` is the first to derive the maximum tolerable time-synchronization deviation subject to network resource constraints and flow requirements at the scheduling level.

VII. Conclusion

In this paper, we have formulated and solved the problem of maximizing the tolerance of time-synchronization deviation to ensure the deterministic transmission of TT flows. Specifically, we have proposed `SDT-TSN`, a novel time-triggered scheduling model with time-synchronization deviations, and then designed an optimal SMT-based solving algorithm for static/offline scheduling, and a heuristic fast solution algorithm that scales up to thousands of flows. Our extensive evaluation results demonstrate that the heuristic algorithm has good scalability and its performance is close to that of the optimal SMT-based algorithm. The proposed methods eliminate the failure of flows due to the conventional use of a constant time-synchronization deviation, and increase the tolerable time-synchronization deviation from $140\mu s$ to $480\mu s$ in various application scenarios.

Acknowledgment

The authors would like to thank the anonymous reviewers, especially the shepherd, for their insightful reviews and suggestions.

References

[1] Q. Yu and M. Gu, “Adaptive group routing and scheduling in multicast time-sensitive networks,” *IEEE Access*, vol. 8, pp. 37 855–37 865, 2020.

[2] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, “Ultra-low latency (ull) networks: The ieee tsn and ietf detnet standards and related 5g ull research,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 88–145, 2018.

[3] “Ieee standard for local and metropolitan area networks–bridges and bridged networks,” *IEEE Std 802.1Q-2022 (Revision of IEEE Std 802.1Q-2018)*, pp. 1–2163, 2022.

[4] S. S. Craciunas and R. S. Oliver, “Out-of-sync schedule robustness for time-sensitive networks,” in *2021 17th IEEE International Conference on Factory Communication Systems (WFCS)*, 2021, pp. 75–82.

[5] “Ieee standard for local and metropolitan area networks–timing and synchronization for time-sensitive applications,” *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, pp. 1–421, 2020.

[6] S. Rodrigues and J. Lv, “Synchronization in time-sensitive networking: An introduction to ieee std 802.1as,” *IEEE Communications Standards Magazine*, vol. 6, no. 4, pp. 14–20, 2022.

[7] M. Gutiérrez, W. Steiner, R. Dobrin, and S. Punnekkat, “Synchronization quality of ieee 802.1as in large-scale industrial automation networks,” in *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2017, pp. 273–282.

[8] T. Stüber, L. Osswald, S. Lindner, and M. Menth, “A survey of scheduling algorithms for the time-aware shaper in time-sensitive networking (tsn),” *Ieee Access*, vol. 11, pp. 61 192–61 233, 2023.

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

[10] Q. Yu and M. Gu, “Adaptive group routing and scheduling in multicast time-sensitive networks,” *IEEE Access*, vol. 8, pp. 37 855–37 865, 2020.

[11] “Z3prover.” [Online]. Available: https://github.com/Z3Prover/z3

[12] A. B. Kahn, “Topological sorting of large networks,” *Communications of the ACM*, vol. 5, no. 11, pp. 558–562, 1962.

[13] D. Tămaş-Selicean and P. Pop, “Optimization of ttethernet networks to support best-effort traffic,” in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. IEEE, 2014, pp. 1–4.

[14] “Ieee/iec draft international standard time-sensitive networking profile for industrial automation,” *IEEE/IEC P60802/D3.0, August 2024*, pp. 1–194, 2024.

[15] “INET Framework version 4.4.1.” [Online]. Available: https://inet.omnetpp.org/

[16] “Ieee standard for local and metropolitan area networks–frame replication and elimination for reliability,” *IEEE Std 802.1CB-2017*, pp. 1–102, 2017.

[17] J. Yan, W. Quan, X. Jiang, and Z. Sun, “Injection time planning: Making cqf practical in time-sensitive networking,” in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 616–625.

[18] C. Zhong, H. Jia, H. Wan, and X. Zhao, “Drls: A deep reinforcement learning based scheduler for time-triggered ethernet,” in *2021 International Conference on Computer Communications and Networks (ICCCN)*, 2021, pp. 1–11.

[19] G. Shani, D. Heckerman, and R. I. Brafman, “An mdp-based recommender system,” *Journal of machine Learning research*, vol. 6, no. Sep, pp. 1265–1295, 2005.

[20] S. Mittal, “A survey on modeling and improving reliability of dnn algorithms and accelerators,” *Journal of Systems Architecture*, vol. 104, p. 101689, 2020.

[21] X. Wang, H. Yao, T. Mai, S. Guo, and Y. Liu, “Reinforcement learning-based particle swarm optimization for end-to-end traffic scheduling in tsn-5g networks,” *IEEE/ACM Transactions on Networking*, vol. 31, no. 6, pp. 3254–3268, 2023.

[22] “Ieee standard for a precision clock synchronization protocol for networked measurement and control systems,” *IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008)*, pp. 1–499, 2020.

[23] “Aerospace standard AS6802: Time-triggered Ethernet,” SAE International Group, 11 2011.

[24] R. Ji, F. Xu, Y. Tao, T. Feng, and C. Zhao, “Fast kalman filter synchronization algorithm for cooperative synchronization in time sensitive networks,” *Digital Signal Processing*, vol. 143, p. 104255, 2023.

[25] Y. Yang, C. Yao, G. Shou, Y. Liu, Z. Li, H. Li, and Y. Hu, “Clock drift and time error compensation with neural network for time synchronization in tsn,” in *2024 IEEE 7th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, vol. 7. IEEE, 2024, pp. 440–444.

[26] X. Zhang, G. Shou, H. Li, H. Yu, Y. Liu, and Y. Hu, “Enable precise timing with clock ensemble in packet networks,” *IEEE Communications Magazine*, vol. 62, no. 3, pp. 117–123, 2023.

[27] Z. Wang, Z. Li, X. Qiao, Y. Zheng, B. Ai, and X. Song, “Time Synchronization for 5G and TSN Integrated Networking,” *arXiv e-prints*, p. arXiv:2401.17721, Jan. 2024.

[28] A. Ghosh, S. Yousefi, and T. Kunz, “Latency bounds for tsn scheduling in the presence of clock synchronization,” *IEEE Networking Letters*, 2024.

[29] H. Yang, T. Zhang, X. Feng, Y. Ma, L. Zhang, and F. Ren, “Absorbing time synchronization errors using shadow queues in time-sensitive networking,” *IEEE Internet of Things Journal*, 2024.