# Flow Scheduling for Conflict-Free Network Updates in Time-Sensitive Software-Defined Networks

Zaiyu Pang, Xiao Huang, Zonghui Li, Sukun Zhang, Yanfen Xu, Hai Wan, and Xibin Zhao

***Abstract*—The digital transformation of industry requires industrial control networks provide high flexibility and determinacy. Time-sensitive software-defined networking that combines time-sensitive networking and software-defined networking is a new network paradigm which provides both real-time transmission feature and network flexibility. During network updates, the transmission consistency needs to be maintained. However, previous mechanisms mostly target on the proper schedule transition, which cannot guarantee no frame loss and also introduces extra update overhead. The article proposes a novel flow schedule generation model which guarantees no frame loss during network updates even with the basic two-phase update mechanism and introduces no extra update overhead. Two algorithms are designed for the model to adapt to different application scenarios: the offline algorithm poses better schedulability, whereas the online one consumes less time with slightly decreased schedulability. The experiments on two real-world industrial networks demonstrate our mechanism achieves zero frame loss without extra update overhead compared to existing methods, and the online algorithm saves 40% execution time with at most 10% schedulability decrease when the bandwidth utilization is less than 50%.**

***Index Terms*—Integer linear programming (ILP)-based scheduling, network updates, time-sensitive networking (TSN), time-triggered (TT) networks.**

Manuscript received November 12, 2019; revised March 22, 2020, April 19, 2020, and May 11, 2020; accepted May 21, 2020. Date of publication May 28, 2020; date of current version November 20, 2020. This work was supported in part by the NSFC Program under Grant U19A2062 and Grant U1801263, in part by the Project funded by China Postdoctoral Science Foundation under Grant 2019M660439, and in part by Key-Area Research and Development Program of Guangdong Province under Grant 2020B010164001. Paper no. TII-19-4970. *(Corresponding author: Hai Wan.)*

Zaiyu Pang, Xiao Huang, Sukun Zhang, Hai Wan, and Xibin Zhao are with the Key Laboratory for Information System Security, Ministry of Education, Beijing National Research Center for Information Science and Technology, School of Software, Tsinghua University, Beijing 100084, China (e-mail: pangzaiyu@163.com; hx0698@gmail.com; zsk18@mails.tsinghua.edu.cn; wanhai@tsinghua.edu.cn; zxb@tsinghua.edu.cn).

Zonghui Li is with the Beijing Key Laboratory of Transportation Data Analysis and Mining, School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China (e-mail: zonghui.lee@gmail.com).

Yanfen Xu is with the CRRC Qingdao Sifang Rolling Stock Research Institute Co. Ltd., Qingdao 266031, China (e-mail: xuyanfen@srsri.com).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2020.2998224

## I. Introduction

THE LAST decades witnessed a strong momentum to unify the industrial control network and ethernet, especially under the context of Industry 4.0 [1]. However, standard ethernet frame delivery does not assure time-critical properties for industrial control applications. The time-triggered (TT) communication paradigm [2] is a cost-efficient solution where time-critical frames, aka TT frames, are deterministically transmitted at statically scheduled precise times while the other frames, aka BE frames, are forwarded in a best-effort (BE) fashion. Networks following this TT communication paradigm are called TT networks, which include both TTEthernet [3] and time-sensitive networking (TSN). TTEthernet is standardized as AS6802 [4] for the automation and aerospace industry by the Society of Automotive Engineers (SAE) group. It defines a fault-tolerant strategy to maintain synchronized time across a distributed system of hosts and switches. TSN is another standard defined by IEEE 802.1 TSN task group as an extension to IEEE 802.3 for real-time transmission. It synchronizes all devices in the real-time communication with IEEE 802.1AS [5] (based on IEEE 1588 [6], a precision clock synchronization protocol). To facilitate TT flow schedule computation and improve TT network flexibility, time-sensitive software-defined networking (TSSDN) is proposed in [7] and [8].

TSSDN introduces time sensitive features to general software-defined networking (SDN) architecture by employing real-time network techniques and protocols. As a result, it is real-time, deterministic, and flexible. In TSSDN, TT frames are transmitted in a TT manner on the data plane which consists of full-duplex ethernet switches and end systems (hosts). These devices are precisely clock synchronized using the precision time protocol (PTP) [6]. End systems send TT frames at a constant bit rate to multiple destinations with fixed time-periods, which requires a flow schedule to plan all data transmission. The TT flow schedule computed by the network controller contains transmission times of each flow at each device. For instance, Fig. 1 is an example network of TSSDN with a controller, five switches, and seven hosts. A TT flow is transmitted in the network from $H_1$ to $\{H_2, H_3\}$. The relative frame sending times to the global macro period are 1, 3, 7, 11 at devices $H_1, S_1, S_2$, and $S_3$, respectively. These concrete time points are calculated by the scheduler synthesizing flow routes and other requirements, e.g., flow periods, end-to-end delays, frame lengths, and so forth.

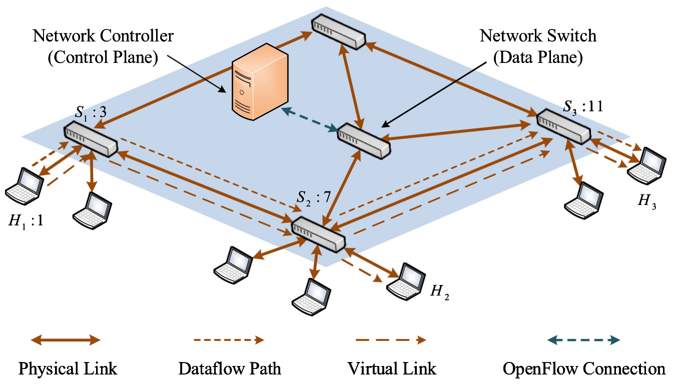


Fig. 1. Example network of TSSDN. The controller is located on the control plane. The data plane is a TT network with five switches and seven hosts. The dataflow paths are $[[H_1,S_1],\ [S_1,S_2],\ [S_2,H_2]]$ (not depicted) and $[[H_1,S_1],\ [S_1,S_2],\ [S_2,S_3],\ [S_3,H_3]]$ (depicted). The union of two paths forms the virtual link set $\{[H_1,S_1],\ [S_1,S_2],\ [S_2,H_2],\ [S_2,S_3],\ [S_3,H_3]\}$. The frame sending times at devices on the path are 1, 3, 7, 11, respectively.

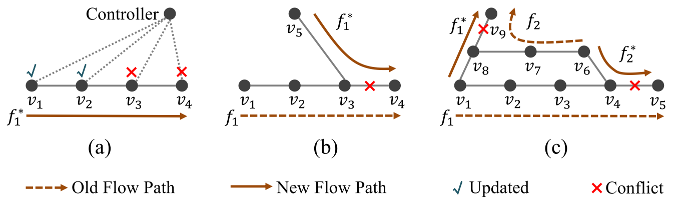


Fig. 2. Anomalies in network updates for TSSDN. (a) Error forwarding at $v_3, v_4$. (b) Conflicts at $[v_3, v_4]$. (c) Deadlock between $f_1, f_2$.

In traditional application scenarios, the TT flow schedule is generated in advance with the network topology and flow specifications known *a priori*, and it does not change during the network operation process. Therefore, previous scheduling researches mostly focus on developing schedule computation methods for static networks with fixed TT flows [9]–[12]. Nevertheless, the emergence of dynamic applications [13], [14], e.g., the real-time Internet of Things (RT-IoT) [15] and train communication networks [16], poses new challenges to the industrial network, i.e., the network must be capable of adapting to frequent changes due to the network topology change (link failure, for example), device insertion and removal as well as TT flow addition and deletion. New flow schedule should be computed and updated to switches and hosts. During the update, the existence of both old and new schedules can easily cause frame loss. Specifically, the old flow frames could be forwarded to a wrong destination due to the mixed use of both schedules and new flow frames may collide with old flow frames if their departure time is identical at a specific device. Safety-critical applications demand no frame loss during network schedule configuration updates and the update duration needs to be minimized.

However, existing update mechanisms [13], [17]–[25] cannot guarantee no update frame loss to achieve transmission consistency and some of them also introduce high update overhead. Hence, in the article, we propose a novel flow schedule generation mechanism to achieve update transmission consistency without extra update overhead. The contributions of the article are as follows.

1) *System model:* We establish a new ILP scheduling model for conflict-free network updates, which guarantees no frame loss in network updates and introduces no extra update duration as well.
2) *Implementation:* We develop two algorithms for different application scenarios. The offline algorithm consumes more time but provides better schedulability while the online one runs faster with slightly lower schedulability.
3) *Experiments:* We conduct experiments on two real industrial network topologies and demonstrate the effectiveness and scalability of our model and algorithms.

The rest of the article is organized as follows. Section II presents the related work and motivation. Section III builds an enhanced ILP scheduling model. Then Section IV describes the offline and online algorithms. Section V evaluates the mechanism in two industrial networks. Finally, Section VI concludes the article.

## II. Related Work

TT flow schedule computation is a complex problem *per se*, which already proved to be NP-Complete [9]. Diverse scheduling approaches have been proposed, e.g., satisfiability modulo theories (SMT)-based [9]–[12], mixed integer linear programming (MILP)-based [8], [16], [26]–[28], and mixed criticality (MC) scheduling [29], etc. On the other hand, online scheduling mechanisms for dynamic requirements are proposed in [16], [30], [31], and [32] to compute new schedules quickly with a solver for specific application scenarios. These researches are primarily for the optimization of scheduling speed and overhead without considering the deterministic transmission during network updates. However, the network reconfiguration must guarantee the physical control persistence, i.e., frame loss number is strictly bounded by application-specific worst-case conditions even no frame loss. The network persistence is satisfied with a good schedule before and after network update whereas frames may be lost or incorrectly forwarded due to mixed configurations while the network configuration is updating. Despite various network update mechanisms proposed to solve this problem recently, they can be classified into three categories.

*Two-phase updates:* The entire update course is divided into two phases [18], [22], [23]. In the unobservable phase, the controller deploys new schedule to each device without enabling it. Then in the one-touch phase, the controller initializes ingress devices to send flows with a new version number. These new flows are forwarded by the new schedule, while those old version flows are processed under the old schedule. The main drawbacks of this mechanism are additional flow table space headroom and the hardness to achieve congestion-free updates if the sending time of a new flow at a device is identical to that of some old flow coincidentally. For instance, in Fig. 2(b), frames of legacy flow $f_1$ and new flow $f_1^*$ both exist in the network and their same departure time at $v_3$ directly causes sending conflicts.

*Ordered updates:* At each round, the controller waits until all switches completed their updates, and only then invokes the next round. Ludwig *et al.* [24], [25] aim to minimize the number of sequential controller interactions when transitioning from the initial to final update stage, and consider secure network updates in the presence of middle boxes [33]. Ordered updates require no extra flow table space. However, at the given time, some flow frames may have not been delivered to its destination devices and remain in the network. These legacy frames will be forwarded with new schedule after the update, which could still cause transimission inconsistency. For instance, in Fig. 2(a), network elements $v_1$, $v_2$, $v_3$, and $v_4$ are updated at the same time, however, legacy flow frames remaining at $[v_2, v_3]$ will be incorrectly forwarded at $v_3, v_4$.

Recently, researchers introduce time into network updates to avoid transmission inconsistency and minimize update overhead.

*Timed updates:* Timed updates integrate TT mechanisms with above fundamental update techniques. Mizrahi *et al.* [19] enable a data-plane timed implementation for two-phase mechanism to optimize update memory overhead. Zheng *et al.* [20], [21] analyse congestion-free ordered updates of multiple flows in a time-extended SDN. Li *et al.* [13] propose an enhanced reconfiguration mechanism for deterministic transmission in TT networks, which introduces update delays for different flows to avoid potential congestion. Whereas these methods function flawlessly on normal scenarios, they are not capable of handling the deadlock situation shown in Fig. 2(c). The legacy flow $f_1$ conflicts with $f_2^*$ at dataflow link $[v_4, v_5]$, which indicates $f_1$ should update prior to $f_2$ to avoid the conflict. Meanwhile, the conflict between $f_2$ and $f_1^*$ at $[v_8, v_9]$ demands $f_2$ update before $f_1$. Under such circumstance, frame loss is inevitable for traditional update methods. Further, timed updates introduce extra update delays and require hosts to have the ability to switch flow version tags at precise time.

To our best knowledge, this article is the first work to solve the deadlock problem in timed updates so that frame loss can be totally avoided during network updates. To tackle the deadlock situation, old and new schedules need to be decoupled. Hence, the article focuses on generating a conflict-free new schedule to keep update consistency in the article.

## III. System Model

### A. Basic Terminologies and Concepts

The physical topology of a network can be formally modeled as an undirected graph $G(V, E)$ where $V$ is the set of all switches and hosts and $E = \{(v_1, v_2) | v_1, v_2 \in V \text{ and } v_1, v_2 \text{ are connected by a physical link}\}$ represents the tuple set of linked devices. Fig. 1 depicts an example topology of TSSDN. A physical connection between two vertices actually defines two directed "dataflow links." The set of dataflow links is denoted by $L$:

$$\forall (v_1, v_2) \in E \Rightarrow [v_1, v_2] \in L, [v_2, v_1] \in L$$

where $(x, y)$ denotes an unordered pair and $[x, y]$ ordered. A sequence of dataflow links $l_i$ forms a dataflow path. A dataflow path in a network is a directed path starting from a given vertex (the sender) and ending at another vertex (the receiver). A dataflow path from $H_1$ to $H_3$ is depicted by dotted arrows in Fig. 1. A dataflow path $p$ from a sender $v_0$ to a receiver $v_n$ is formalized by the sequence of its dataflow links

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

A TT flow is a union of periodic TT frames. All flows transferred in the network form the set $F$. Since a flow $f_i \in F$ can be delivered from a source to multiple destinations, the individual dataflow links on paths between the sender and each single receiver together form virtual links. The set of virtual links is denoted by $f_i.vl$ which is formally the union of all dataflow paths: $f_i.vl = \cup p_i$, $p_i$ is a path of $f_i$ (see Fig. 1). A TT flow $f_i$ can be fully specified as follows:

$$f_i = \{f_i.prd, f_i.len, f_i.ddl, f_i.src, f_i.dst, f_i.vl\}. \quad (1)$$

$f_i.prd, f_i.len, f_i.ddl, f_i.src, f_i.dst$ represent the period, frame length, end-to-end delay, single sender, and receiver set of $f_i$, respectively. All these except $f_i.vl$ are provided by the application requirements. In the general scheduling problem, $f_i.vl$ can be calculated by the scheduler, but considering the performance degradation, we take the route of $f_i$ as input to the schedule algorithm in the article. Moreover, $f_i^{[v_k, v_l]}.ofst$ (abbreviation of offset) is used to denote the departure time of $f_i$ on dataflow link $[v_k, v_l]$, further simplified to $f_i^{v_k}.ofst$ as departure times are identical for different $v_l$. If $v \in f_i.dst$, $f_i^v.ofst$ denotes the reception time of flow $f_i$ at host $v$. In this article, our objective is to compute the following schedule $S$.

$$S = \{f_i^{v_k}.ofst | [v_k, v_l] \in f_i.vl, f_i \in F\} \cup \\ \{f_i^v.ofst | v \in f_i.dst, f_i \in F\}.$$

Frames of flow $f_i$ in $n$th period are sent from $v_k$ at $f_i^{v_k}.ofst + n * f_i.prd$. Then the frames go through a series of delays to reach the destination, including the processing delay $d^{\text{proc}}$, queuing delay $d^{\text{que}}$, sending delay $d^{\text{send}}$, and propagating delay $d^{\text{prop}}$. $d^{\text{proc}}, d^{\text{send}}, d^{\text{prop}}$ are constant and known *a priori*. Through scheduling TT flows with no sending time overlaps, $d^{\text{que}}$ is regarded as deterministic. Then the accurate arrival time of flow $f_i$ at vertex $v_l$ is

$$f_i^{[v_k, v_l]}.arrival = f_i^{[v_k, v_l]}.ofst + f_i^{[v_k, v_l]}.d^{\text{proc}} \\ + f_i^{[v_k, v_l]}.d^{\text{send}} + f_i^{[v_k, v_l]}.d^{\text{prop}}.$$

To be concise, we use $f_i^{[v_k, v_l]}.dl$ to represent the sum of above three delays in the rest of our article.

### B. Basic ILP Scheduling Model

Integer linear programming (ILP) is a broadly accepted standard method for network scheduling problems with readily available and highly optimized solvers. To formalize the enhanced ILP scheduling model for network updates, the basic TT flow scheduling problem needs to be modeled first.

#### 1) ILP Inputs:

1) Network topology as an undirected graph, $G = (V, E)$;
2) Flow specifications as a set of tuple 1, $F$;
3) Delay set, $D = \{f_i^{[v_k, v_l]}.dl \mid f_i \in F, [v_k, v_l] \in f_i.vl\}$.

Note that the route of each flow is included in $f_i.vl$, a path tree with $f_i.src$ as the root and $f_i.dst$ as leaves.

*2) ILP Variables:*

1) Flow departure times. $\{f_i^{v_k}.ofst | f_i \in F,\ [v_k, v_l] \in f_i.vl\}$;
2) Flow reception times. $\{f_i^{v}.ofst \mid f_i \in F, v \in f_i.dst\}$;
3) Auxiliary variables, $\{R_{i,j}^{[v_k,v_l]}\}, \{K_{i,j}^{[v_k,v_l]}\}$, where $f_i, f_j \in F, [v_k, v_l] \in f_i.vl \cap f_j.vl$. These variables enable scheduling constraints to be linear.

*3) Objective Function:* For the scheduling problem *per se*, finding a feasible solution is sufficient since the schedule is only a set of times meeting required constraints. Thus, the objective function can be omitted, i.e., set to 0. On the other hand, the objective function can be tweaked to attain better network features. For example, it can be set as below to minimize the end-to-end delay of TT flows

$$\min\left(\sum(f_i^{v}.ofst - f_i^{f_i.src}.ofst)\right), f_i \in F, v \in f_i.dst.$$

Nevertheless, the solver costs more time to find an optimal solution, so objective functions are all set to 0 for better performance in the article.

*4) Constraints:*

1) Flow period constraints: For every flow, the sending time at the source device should less than its period
$$\forall f_i \in F : 0 \le f_i^{f_i.src}.ofst < f_i.prd.$$
Note that $f_i^{f_i.src}.ofst$ denotes the frame departure time of $f_i$ at its source $f_i.src$, which may be different from the frame generation time of $f_i$ at $f_i.src$. If that is the case, the frame generation time should be set to the scheduled departure time, i.e., $f_i^{f_i.src}.ofst$, to achieve better real-time constraint.
2) End-to-end transmission delay constraints: For every flow, all frames must arrive at all destinations in time
$$\forall f_i \in F, v \in f_i.dst : 0 < f_i^{v}.ofst - f_i^{f_i.src}.ofst \le f_i.ddl.$$
3) Sending order constraints: The prerequisite for frame transmission is the frame has arrived, i.e., the sending time at $v_l$ must be greater than that at $v_k$ for dataflow link $[v_k, v_l]$
$$\forall f_i \in F, [v_k, v_l] \in f_i.vl : f_i^{v_l}.ofst > f_i^{v_k}.ofst + f_i^{[v_k,v_l]}.dl.$$
4) No sending time overlap constraints: Flow sending times should have no any overlaps so as to avoid congestion
$$\begin{aligned}\forall m, n \in \mathbb{Z}, f_i, f_j \in F, [v_k, v_l] \in f_i.vl \cap f_j.vl :&\\ m \cdot f_i.prd + f_i^{v_k}.ofst&\\ \ge n \cdot f_j.prd + f_j^{v_k}.ofst + f_j^{[v_k,v_l]}.dl \vee&\\ n \cdot f_j.prd + f_j^{v_k}.ofst&\\ \ge m \cdot f_i.prd + f_i^{v_k}.ofst + f_i^{[v_k,v_l]}.dl.& \quad (2)\end{aligned}$$
Owing to nonlinear elements $\forall m, n \in \mathbb{Z}$ and $\vee$, the constraint is transformed using $R, K$. The final linear form is shown as below. The proof will be stated later
$$\begin{aligned}\forall f_i, f_j \in F, [v_k, v_l] \in f_i.vl \cap f_j.vl :&\\ (R_{i,j}^{[v_k,v_l]} > f_j^{[v_k,v_l]}.dl) \wedge (R_{i,j}^{[v_k,v_l]} < \gcd(f_i.prd, f_j.prd)&\\ - f_i^{[v_k,v_l]}.dl) \wedge (f_i^{v_k}.ofst - f_j^{v_k}.ofst =&\\ K_{i,j}^{[v_k,v_l]} \cdot \gcd(f_i.prd, f_j.prd) + R_{i,j}^{[v_k,v_l]})&\\ \wedge(0 \le R_{i,j}^{[v_k,v_l]} < \gcd(f_i.prd, f_j.prd)).& \quad (3)\end{aligned}$$

A feasible solution to this ILP provides a suitable schedule $S$ for flows in $F$, i.e., the proper values of $f_i^{v_k}.ofst$ and $f_i^{v}.ofst$.

## C. Enhanced ILP Scheduling Model

The basic ILP scheduling model maintains transmission consistency without network updates. Extra constraints are required to compute new schedule for network updates.

Flow specifications in new network configuration is denoted as $F^*$ with the superscript $^*$ meaning new. Similarly, $f_i^*$ denotes a flow in $F^*$. According to Li's paper [13], for any two TT flows $f_i \in F, f_j^* \in F^*$, if they are updated at the same time, the conflict between $f_i$ and $f_j^*$ at dataflow link $[v_k, v_l] \in L$ is equivalent to

$$\begin{aligned}\exists\, m, n \in \mathbb{Z}, m \ge 0, n \ge 0, \text{such that}&\\ (f_i^{f_i.src}.ofst + m \cdot f_i.prd < f_j^{*f_j^*.src}.ofst + n \cdot f_j^*.prd)&\\ \wedge (f_i^{v_k}.ofst + m \cdot f_i.prd = f_j^{*v_k}.ofst + n * f_j^*.prd).& \quad (4)\end{aligned}$$

To eliminate prospective conflicts, above conditions are inverted to give theorem 1.

*Theorem 1:* For any two TT flows $f_i \in F, f_j^* \in F^*$, if they are updated simultaneously, no potential conflict between $f_i$ and $f_j^*$ at dataflow link $[v_k, v_l] \in L$ is equivalent to

$$\begin{aligned}\forall\, m, n \in \mathbb{Z}, m \ge 0, n \ge 0 :&\\ (f_i^{f_i.src}.ofst + m \cdot f_i.prd \ge f_j^{*f_j^*.src}.ofst + n \cdot f_j^*.prd)&\\ \vee (f_i^{v_k}.ofst + m \cdot f_i.prd \ne f_j^{*v_k}.ofst + n \cdot f_j^*.prd).& \quad (5)\end{aligned}$$

Theorem 1 includes nonlinear components $\forall m, n \in \mathbb{Z}$ and $\vee$. We first state Lemma 1 to apply $\forall$ operator to only the second inequality.

*Lemma 1:* For any two TT flows $f_i \in F, f_j^* \in F^*$, if they are updated simultaneously, no potential conflict between $f_i$ and $f_j^*$ at dataflow link $[v_k, v_l] \in L$ is equivalent to

$$\begin{aligned}(f_i^{v_k}.ofst - f_i^{f_i.src}.ofst \le f_j^{*v_k}.ofst - f_j^{*f_j^*.src}.ofst)&\\ \vee (\forall\, m, n \in \mathbb{Z}, m \ge 0, n \ge 0 :&\\ (f_i^{v_k}.ofst + m \cdot f_i.prd \ne f_j^{*v_k}.ofst + n \cdot f_j^*.prd)).& \quad (6)\end{aligned}$$

*Proof:* First prove condition (6) can deduce (5), i.e., (6) $\Rightarrow$ (5). It is trivial that condition (5) holds when $f_i^{v_k}.ofst + m \cdot f_i.prd \ne f_j^{*v_k}.ofst + n \cdot f_j^*.prd$ holds for any integer $m, n$ in (6). If there exist $m', n'$ such that $f_i^{v_k}.ofst + m' \cdot f_i.prd = f_j^{*v_k}.ofst + n' \cdot f_j^*.prd$, we add it to $f_i^{f_i.src}.ofst - f_i^{v_k}.ofst \ge f_j^{*f_j^*.src}.ofst - f_j^{*v_k}.ofst$ which is a direct deformation of the first inequality in (6), then we get $f_i^{f_i.src}.ofst + m' \cdot f_i.prd \geq f_j^{*f_j^*.src}.ofst + n' \cdot f_j^*.prd$. Hence, conditions in (5) still hold. Next, prove (5) $\Rightarrow$ (6) by proving its inverse negative proposition $(6) \Rightarrow (5)$. The inverse of (6) is as follows:

$$(f_i^{v_k}.ofst - f_i^{f_i.src}.ofst > f_j^{*v_k}.ofst - f_j^{*f_j^*.src}.ofst)$$
$$\wedge (\exists\, m, n \in \mathbb{Z}, m \geq 0, n \geq 0 :$$
$$f_i^{v_k}.ofst + m \cdot f_i.prd = f_j^{*v_k}.ofst + n \cdot f_j^*.prd). \quad (7)$$

The inverse of (5) is as follows:

$$\exists\, m, n \in \mathbb{Z}, m \geq 0, n \geq 0 :$$
$$((f_i^{f_i.src}.ofst + m \cdot f_i.prd < f_j^{*f_j^*.src}.ofst + n \cdot f_j^*.prd)$$
$$\wedge (f_i^{v_k}.ofst + m * f_i.prd = f_j^{*v_k}.ofst + n * f_j^*.prd)). \quad (8)$$

Assuming there exist $m', n'$ such that $f_i^{v_k}.ofst + m' \cdot f_i.prd = f_j^{*v_k}.ofst + n' \cdot f_j^*.prd$ holds in (7), then add this equation to $f_i^{f_i.src}.ofst - f_i^{v_k}.ofst < f_j^{*f_j^*.src}.ofst - f_j^{*v_k}.ofst$ which is a deformation of the first inequality in (7). We get $f_i^{f_i.src}.ofst + m' \cdot f_i.prd < f_j^{*f_j^*.src}.ofst + n \cdot f_j^*.prd$. Therefore, conditions in (8) holds, which means (5) $\Rightarrow$ (6). ■

Lemma 1 is a "raster" form with periods split into equal slots [9]. An accurate form with actual delays introduced is in Theorem 2.

*Theorem 2:* For any two TT flows $f_i \in F, f_j^* \in F^*$, if they are updated at the same time, no potential conflict between $f_i$ and $f_j^*$ at dataflow link $[v_k, v_l] \in L$ is equivalent to

$$(f_i^{v_k}.ofst - f_i^{f_i.src}.ofst \leq f_j^{*v_k}.ofst - f_j^{*f_j^*.src}.ofst) \quad (9)$$
$$\vee (\forall\, m, n \in \mathbb{Z}, m \geq 0, n \geq 0 :$$
$$(f_i^{v_k}.ofst + m \cdot f_i.prd \geq$$
$$f_j^{*v_k}.ofst + n \cdot f_j^*.prd + f_j^{*[v_k,v_l]}.dl)$$
$$\vee (f_j^{*v_k}.ofst + n \cdot f_j^*.prd \geq$$
$$f_i^{v_k}.ofst + m \cdot f_i.prd + f_i^{[v_k,v_l]}.dl)). \quad (10)$$

The proof below can also be used to prove the transformation of (2) in no sending time overlap constraints with little tweak. Since the representation in Theorem 2 is still not a linear form, we give the following equivalent condition to (10):

$$\forall f_i \in F, f_j^* \in F^*, [v_k, v_l] \in f_i.vl \cap f_j^*.vl :$$
$$((f_i^{v_k}.ofst - f_j^{*v_k}.ofst) \bmod \gcd(f_i.prd, f_j^*.prd) \geq$$
$$f_j^{*[v_k,v_l]}.dl) \wedge ((f_j^{*v_k}.ofst - f_i^{v_k}.ofst)$$
$$\bmod \gcd(f_i.prd, f_j^*.prd) \geq f_i^{[v_k,v_l]}.dl). \quad (11)$$

*Lemma 2:* (10) is equivalent to (11), that is, (10)$\Leftrightarrow$(11).

*Proof:* First prove $(11) \Rightarrow (10)$ by contradiction. Provided condition (11) holds but there exist $a, b \in \mathbb{Z}$, such that $(a \cdot f_i.prd + f_i^{v_k}.ofst < b \cdot f_j^*.prd + f_j^{*v_k}.ofst + f_j^{*[v_k,v_l]}.dl) \wedge (b \cdot f_j^*.prd + f_j^{*v_k}.ofst < a \cdot f_i.prd + f_i^{v_k}.ofst + f_i^{[v_k,v_l]}.dl)$, without loss of generality, assume $f_i^{v_k}.ofst - f_j^{*v_k}.ofst \geq 0$ and let integer $g = \gcd(f_i.prd, f_j^*.prd)$, $x_1 = f_i.prd/g$, $x_2 = f_j^*.prd/g$, $t = bx_2 - ax_1$, then we have

$$f_i^{v_k}.ofst - f_j^{*v_k}.ofst < bx_2 \cdot g - ax_1 \cdot g + f_j^{*[v_k,v_l]}.dl$$
$$= t \cdot g + f_j^{*[v_k,v_l]}.dl. \quad (12)$$

$f_i^{v_k}.ofst - f_j^{*v_k}.ofst$ can be represented by $r \cdot g + s$ where $r \in \mathbb{Z}, 0 \leq s < g$. By replacing $f_i^{v_k}.ofst - f_j^{*v_k}.ofst$ with $r \cdot g + s$ in (12), we have $(r - t) \cdot g < f_j^{*[v_k,v_l]}.dl - s$. Since $(f_i^{v_k}.ofst - f_j^{*v_k}.ofst) \bmod g \geq f_j^{*[v_k,v_l]}.dl$, we get $f_j^{*[v_k,v_l]}.dl \leq s < g$. Combining these two inequalities, we have $(r - t) \cdot g < f_j^{*[v_k,v_l]}.dl - s \leq 0$, i.e., $r < t$. On the other hand, $f_j^{*v_k}.ofst - f_i^{v_k}.ofst < a \cdot f_i.prd - b \cdot f_j^*.prd + f_j^{*[v_k,v_l]}.dl = (-t) \cdot g + f_i^{[v_k,v_l]}.dl$. Let $f_j^{*v_k}.ofst - f_i^{v_k}.ofst = r' \cdot g + s'$. By applying the same technique as above, we have $(r' + t) \cdot g < f_i^{[v_k,v_l]}.dl - s' \leq 0$, i.e., $r' < -t$. Therefore, we get

$$r < t < -r'.$$

Notice that $f_i^{v_k}.ofst - f_j^{*v_k}.ofst = -(f_j^{*v_k}.ofst - f_i^{v_k}.ofst)$, then it is deduced that $r \cdot g + s = -r' \cdot g - s' = (-r' - 1) \cdot g + g - s'$. And thus, $r + r' = -1$, then $r < t < r + 1$ which is a contradiction.

Next, we prove $(10) \Rightarrow (11)$ by proving its inverse negative proposition $(11) \Rightarrow (10)$. Without loss of generality, we assume $(f_i^{v_k}.ofst - f_j^{*v_k}.ofst) \bmod g < f_j^{*[v_k,v_l]}.dl$ and still let $f_i^{v_k}.ofst - f_j^{*v_k}.ofst = r \cdot g + s$, then we have $s < f_j^{*[v_k,v_l]}.dl$. We suppose $f_i^{v_k}.ofst - f_j^{*v_k}.ofst \geq 0$, then $r \geq 0$. The proof is identical if $f_i^{v_k}.ofst - f_j^{*v_k}.ofst < 0$. To prove (10), we need to find $a', b' \in \mathbb{Z}$ to satisfy

$$f_i^{v_k}.ofst - f_j^{*v_k}.ofst < (b'x_2 - a'x_1) \cdot g + f_j^{*[v_k,v_l]}.dl$$
$$\wedge\, f_j^{*v_k}.ofst - f_i^{v_k}.ofst < (a'x_1 - b'x_2) \cdot g + f_i^{[v_k,v_l]}.dl. \quad (13)$$

Let $b'x_2 - a'x_1 = r$. Since $s < f_j^{*[v_k,v_l]}.dl$, then we have

$$f_i^{v_k}.ofst - f_j^{*v_k}.ofst = r \cdot g + s$$
$$< r \cdot g + f_j^{*[v_k,v_l]}.dl.$$

On the other side, we have $f_j^{*v_k}.ofst - f_i^{v_k}.ofst < -r \cdot g < -r \cdot g + f_i^{[v_k,v_l]}.dl$ due to $f_i^{v_k}.ofst - f_j^{*v_k}.ofst \geq r \cdot g$. Hence, (13) holds when $b'x_2 - a'x_1 = r$. Finally, because $x_1 = f_i.prd/g, x_2 = f_j^*.prd/g$, then $x_1$ and $x_2$ are relatively prime, i.e., $\gcd(x_1, x_2) = 1$. Otherwise, $\gcd(f_i.prd, f_j^*.prd) > g$ which is a contradiction. As per Bézout's identity, there exist $a', b' \in \mathbb{Z}$, such that $\gcd(x_1, x_2) = b'x_2 - a'x_1 = 1$, then it can be deduced immediately that $\forall r \in \mathbb{Z}, \exists a', b' \in \mathbb{Z}, s.t.\ b'x_2 - a'x_1 = r$. Thus, the lemma is proved. ■

The mod operator in (11) can be eliminated by expanding with auxiliary variables $R$ and $K$ as follows:

$$f_j^{*[v_k,v_l]}.dl < R_{i,j*}^{[v_k,v_l]} < \gcd(f_i.prd, f_j^*.prd) - f_j^{*[v_k,v_l]}.dl$$
$$f_i^{v_k}.ofst - f_j^{*v_k}.ofst =$$
$$K_{i,j*}^{[v_k,v_l]} \cdot \gcd(f_i.prd, f_j^*.prd) + R_{i,j*}^{[v_k,v_l]}$$
$$0 \le R_{i,j*}^{[v_k,v_l]} < \gcd(f_i.prd, f_j^*.prd).$$

Another auxiliary variable $X$ is introduced to replace $\vee$ in Theorem 2. $X_{i,j*}^{[v_k,v_l]} = 1$ if and only if $f_i^{v_k}.ofst - f_i^{f_i.src}.ofst \le f_j^{*v_k}.ofst - f_j^{*f_j^*.src}.ofst$. Thus, the final linear constraint to avoid congestion in network update is specified in Theorem 3.

*Theorem 3:* For any two TT flows $f_i \in F, f_j^* \in F^*$, if they are updated at the same time, there is no potential conflict between $f_i$ and $f_j^*$ at the dataflow link $[v_k, v_l] \in L$ iff

$$f_i^{v_k}.ofst - f_i^{f_i.src}.ofst \le$$
$$f_j^{*v_k}.ofst - f_j^{*f_j^*.src}.ofst + M(1 - X_{i,j*}^{[v_k,v_l]})$$
$$\wedge f_i^{v_k}.ofst - f_i^{f_i.src}.ofst >$$
$$f_j^{*v_k}.ofst - f_j^{*f_j^*.src}.ofst - M \cdot X_{i,j*}^{[v_k,v_l]}$$
$$\wedge R_{i,j*}^{[v_k,v_l]} > (1 - X_{i,j}^{[v_k,v_l]}) f_j^{*[v_k,v_l]}.dl - C \cdot X_{i,j}^{[v_k,v_l]}$$
$$\wedge R_{i,j*}^{[v_k,v_l]} < (1 - X_{i,j}^{[v_k,v_l]}) \cdot$$
$$(\gcd(f_i.prd, f_j^*.prd) - f_i^{[v_k,v_l]}.dl) + C \cdot X_{i,j*}^{[v_k,v_l]}$$
$$\wedge f_i^{v_k}.ofst - f_j^{*v_k}.ofst =$$
$$K_{i,j*}^{[v_k,v_l]} * \gcd(f_i.prd, f_j^*.prd) + R_{i,j*}^{[v_k,v_l]}$$
$$\wedge 0 \le R_{i,j*}^{[v_k,v_l]} < \gcd(f_i.prd, f_j^*.prd)$$
$$\wedge X_{i,j*}^{[v_k,v_l]} \in \{0,1\} \wedge R_{i,j*}^{[v_k,v_l]}, K_{i,j*}^{[v_k,v_l]} \in \mathbb{Z} \tag{14}$$

where $M$ is a sufficiently large integer satisfying $0 \le f_i^{v_k}.ofst - f_i^{f_i.src}.ofst < M, 0 \le f_j^{*v_k}.ofst - f_j^{*f_j^*.src}.ofst < M$, and $C$ is an integer large enough to satisfy $0 \le R_{i,j*}^{[v_k,v_l]} < C$.

Hence, the enhanced ILP scheduling model is as follows.

*ILP Inputs:*

1) Network topology as undirected graph, $G^* = (V^*, E^*)$;
2) Specifications of the old flows $F$ and new flows $F^*$.
3) The set of delays, $D = \{f_i^{[v_k,v_l]}.dl | f_i \in F \cup F^*, [v_k, v_l] \in f_i.vl\}$.

*ILP Variables:*

1) Flow sending times. $\{f_i^{v_k}.ofst | f_i \in F^*, [v_k, v_l] \in f_i.vl\}$;
2) Flow reception times. $\{f_i^{v}.ofst \mid f_i \in F^*, v \in f_i.dst\}$;
3) Auxiliary variables, $\{R_{i,j}^{[v_k,v_l]}\}$, $\{K_{i,j}^{[v_k,v_l]}\}$, where $f_i, f_j \in F^*$, $[v_k, v_l] \in f_i.vl \cap f_j.vl$ and $\{R_{i,j*}^{[v_k,v_l]}\}$, $\{K_{i,j*}^{[v_k,v_l]}\}$, $\{X_{i,j*}^{[v_k,v_l]}\}$ where $f_i \in F$, $f_j^* \in F^*$, $[v_k, v_l] \in f_i.vl \cap f_j^*.vl$.

*Objective Function:* 0.

**Algorithm 1:** Offline Enhanced Scheduler.

```
1:  function
    OFFLINESCHEDULER($G^*, F, S, F^*, D$)    ▷$G^*$: new topology, $F$: old flow set, $S$: old basic schedule, $F^*$: new flow set, $D$: delay set
2:    $F^* \leftarrow$ SORT ($F^*$)    ▷Sort $F^*$ by flow period in ascending order
3:    $vars \leftarrow$ BUILDVARS ($G^*, F, F^*$)    ▷Build ILP variables
4:    $obj \leftarrow 0$    ▷Objective function is 0
5:    for $i \leftarrow 1$ to SIZE($F^*$) do
6:      for $j \leftarrow i$ to 1 do
7:        $F_s \leftarrow \{f_k^* \in F^* | j \le k \le i\}$
8:        $cons \leftarrow$ BUILDOFFLINECONS($vars$, $G^*$, $F$, $S$, $F^*$, $D$, $F_s$)    ▷Build ILP constraints for $F_s$
9:        $solved \leftarrow$ SOLVE($vars$, $cons$, $obj$)    ▷If feasible, SOLVE returns True and stores values in $vars$
10:       if $solved$ = True then break
11:     if $solved$ = False then break
12:   if $solved$ = True then
13:     return GENERATESCHEDULE($vars$)
14:   else return Infeasible
```

*Constraints:* Constraints of the basic ILP model plus no network update congestion constraints in Theorem 3.

## IV. Implementation for Enhanced Model

This section elaborates the implementation of the enhanced ILP scheduling model. Two algorithms, i.e., offline and online algorithms, are designed to serve different application purposes.

### A. Offline Algorithm

The most naïve method to compute the schedule is simply building constraints all at once and then putting them into the ILP solver. However, this simple method is slow and memory-intensive. For millions of variables and constraints which are quite common in TT flow scheduling, it runs for hours to find a feasible solution and costs gigabytes of memory. It is unacceptable in real-world applications. Hence, we present offline Algorithm 1 for the enhanced ILP scheduling model inspired by greedy and backtracking strategies. Compared to the naïve method, Algorithm 1 does not solve all constraints together but instead uses solved values in previous iterations to decrease the constraint number. The constraints of current flow $f_i$ are established at each iteration by BUILDOFFLINECONS using solved values of $\{f_k \in F^* | 1 \le k < j\}$. Then if feasible, advance to build and solve next flow, otherwise, retrace solved flows one by one, i.e., replacing the fixed values of previous flows in constraints with flexible variables, and try to solve them again. Algorithm 1 spends less time than the simple one with a large solution space and most importantly, it brings a tremendous memory usage decrease. In the worst case, it degrades to the naïve method with no schedulability loss.

**Algorithm 2:** Online Enhanced Scheduler.

```
1:  function ONLINESCHEDULER($G^*, F, S, F^*, S^*, D$)   ▷$G^*$: new topology, $F$: old flow set, $S$: old basic schedule, $F^*$: new flow set, $S^*$: new basic schedule, $D$: delay set
2:    $F_c \leftarrow \{f_j^* | f_i, f_j^*$ satisfy (4), $f_j^* \in F^*, \forall f_i \in F\}$   ▷$F_c$ contains new flows which may conflict with flows in $F$
3:    $F_p \leftarrow F^* \setminus F_c$
4:    $F_c \leftarrow$ SORT($F_c$), $F_p \leftarrow$ SORT($F_p$)   ▷Sort $F_c, F_p$ by flow period in ascending order
5:    $vars \leftarrow$ BUILDVARS($G^*, F, F^*$)
6:    for $i \leftarrow 1$ to SIZE($F_c$) do
7:      for $j \leftarrow i$ to 1 do
8:        $F_s \leftarrow \{f_k^* \in F_c | j \leq k \leq i\}$
9:        $cons \leftarrow$ BUILDONLINECONS($vars$, $G^*$, $F$, $S$, $F^*$, $S^*$, $D$, $F_s$)
10:       $solved \leftarrow$ SOLVE($vars$, $cons$, $obj$)
11:       if $solved =$ True then break
12:     if $solved =$ False then
13:       for flow $f_p^*$ in $F_p$ do
14:         $F_s \leftarrow F_s \cup \{f_p^*\}$
15:         $cons \leftarrow$ BUILDONLINECONS($vars$, $G^*$, $F$, $S$, $F^*$, $S^*$, $D$, $F_s$)
16:         $solved \leftarrow$ SOLVE($vars$, $cons$, $obj$)
17:         if $solved =$ True then break
18:       if $solved =$ False then break
19:   if $solved =$ True then
20:     return GENERATESCHEDULE($vars$)
21:   else returnInfeasible
```

## B. Online Algorithm

Despite the simplicity of the offline algorithm, its all flow computation manner makes it time-consuming if the number of flow is quite large. As a result, Algorithm 2 is proposed as a fast online scheduler to compute the conflict-free schedule when network updates arise. Different from the offline algorithm, Algorithm 2 does not calculate the available schedule for all flows in $F^*$. Instead, it merely computes the schedule for those that may conflict with legacy flows in $F$. The flow trim can dramatically reduce the execution time to generate a feasible solution. To achieve this goal, a basic schedule for network $(G^*, F^*)$ is required for conflict-free flows. There are various adaptive scheduler proposed to compute new schedule for network updates quickly, such as adaptive scheduling for TT train networks [16], incremental flow scheduling in TSSDN [27], etc. The basic schedule fast generation is beyond the scope of this article.

Algorithm 2 first traverses all $(f_i, f_j^*)$ pairs to search for conflict flows by verifying formulas in (4), and then the similar greedy and backtracking strategies to those in Algorithm 1 are still used. All flows in $F_c$, the conflict flow set including far fewer flows than $F^*$, are built constraints on and solved with the prior schedule of $F_p$ in $S^*$. BUILDONLINECONS leverages both solved values of flows in $F_c$ and existing schedule information in $S^*$.

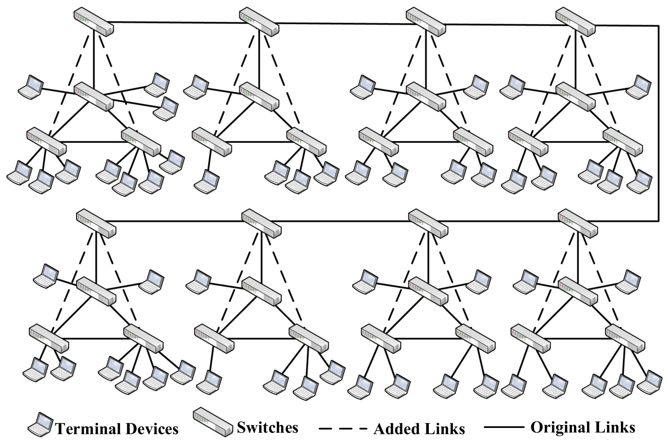


Fig. 3. Train network topology with 54 end systems and 32 switches.

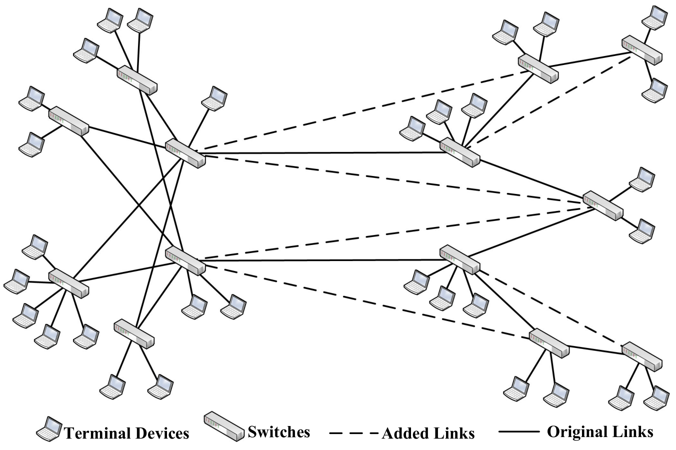


Fig. 4. Orion CEV network topology with 31 end systems and 13 switches.

If current $F_s$ is infeasible, the algorithm backtracks to previous flows in $F_c$. However, only retracing $F_c$ is insufficient to keep the schedulability, and thus $F_p$ is also retraced if no solution found for all constraints of flows in $F_c$. In this way, the schedulability is maintained while in most cases, a feasible solution can be found in a short time.

## C. Comparison of Two Algorithms

The offline and online algorithms are designed to fit into different application scenarios. The idea of offline algorithm is much simpler, which works just like an ordinary TT flow scheduler, but with some extra constraints to avoid frame loss during network updates. Since it needs to compute the schedule of all flows in $F^*$, it consumes a lot of time if the number of flow is large. Also due to this computation strategy, it poses a high schedulability. On the contrary, online algorithm does not compute flow schedule all at the beginning. It attempts to only recompute those flows potentially causing conflicts during updates. In most cases, these "conflict" flows are much fewer and thus online algorithm can obtain a feasible schedule swiftly. However, there might be no feasible solution by only

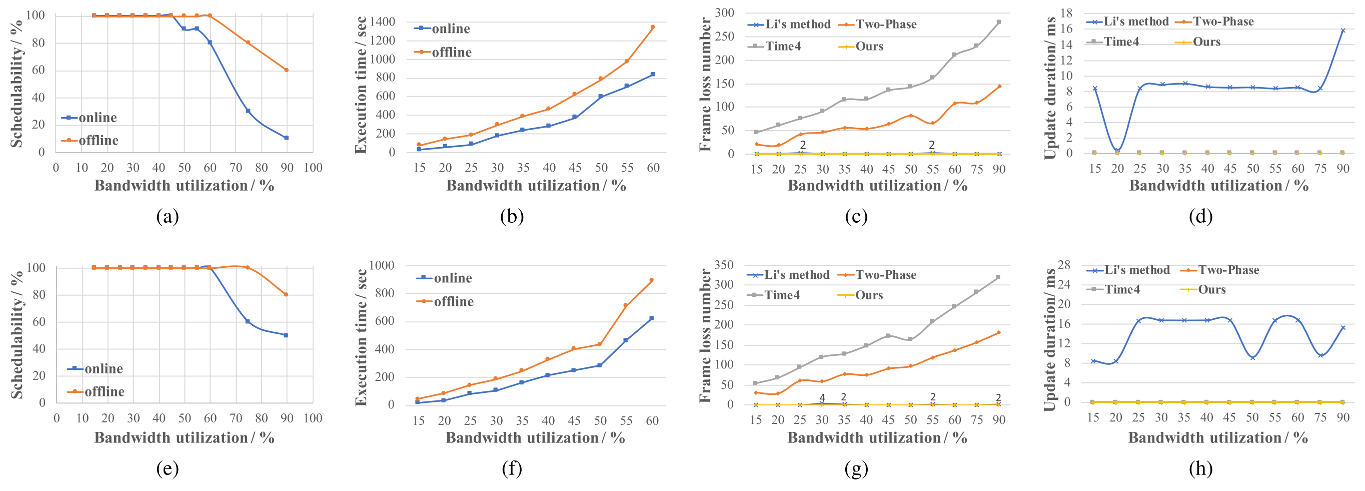


Fig. 5. First row includes experiment results of the train network and the second row includes those of the Orion CEV network. (a) and (e) Relationship between the schedulability and different bandwidth utilization thresholds with respect to both online and offline algorithms. (b) and (f) Average execution time of both algorithms for varying bandwidth utilization. (c) and (g) Comparison of frame loss numbers of Time4, two-phase, Li's method, and ours. (d) and (h) Comparison of update time duration of Time4, two-phase, Li's method, and ours.

TABLE I
SCHEDULABILITY AND EXECUTION TIME OF ONLINE AND OFFLINE ALGORITHM FOR THE TRAIN NETWORK

| **Bandwidth utilization (%)** | | **15** | **20** | **25** | **30** | **35** | **40** | **45** | **50** | **55** | **60** | **75** | **90** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **schedulability (%)** | online | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 90 | 90 | 90 | 30 | 10 |
| | offline | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 80 | 60 |
| **execution time (s)** | online | 28 | 56 | 84 | 175 | 239 | 283 | 373 | 596 | 708 | 833 | 1103 | 1192 |
| | offline | 76 | 142 | 183 | 294 | 386 | 467 | 625 | 785 | 975 | 1340 | 2115 | 2436 |

TABLE II
SCHEDULABILITY AND EXECUTION TIME OF ONLINE AND OFFLINE ALGORITHM FOR THE ORION CEV NETWORK

| **Bandwidth utilization (%)** | | **15** | **20** | **25** | **30** | **35** | **40** | **45** | **50** | **55** | **60** | **75** | **90** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **schedulability (%)** | online | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 30 | 10 |
| | offline | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 80 |
| **execution time (s)** | online | 18 | 34 | 84 | 106 | 161 | 214 | 250 | 283 | 466 | 623 | 977 | 1124 |
| | offline | 45 | 86 | 146 | 189 | 245 | 327 | 403 | 435 | 713 | 892 | 1486 | 1660 |

recomputing "conflict" flows. In this case, online algorithm traces back all flows which can cause more running time than offline algorithm does. In practical applications, if the network bandwidth utilization is high ($\geq 75\%$) or the network update is complex and significant involving lots of device and flow changes and causes numerous conflict flows, offline algorithm would be a better solution. In other cases, online algorithm should be fast and satisfactory.

## V. Evaluation and Analysis

Two typical scenarios, namely topology changes and flow variations due to requirement changes, cause the network update in TSSDN. We test two real industrial network topologies in such two scenarios, employing both online and offline algorithms. To evaluate the quality and performance of our mechanism comprehensively, we choose different bandwidth utilization thresholds and compare our mechanism with Time4 [34], two-phase [18] and Li's method [13].

### A. Experiment Setup

*1) Topology Selection:* A typical ethernet is usually based on three topologies: star, tree, and line. Industrial networks are often enhanced with redundant topologies for reliable transmission and thus the ring topology is also widely used. To evaluate the proposed mechanism on the four topologies, we select two typical and significant TSSDN application scenarios, the train network [35] (Fig. 3) and the Orion Crew Exploration Vehicle (CEV) network [36] (Fig. 4) as test topologies.

*2) TT Scheduler:* Two TT schedulers, the basic and enhanced scheduler, are implemented based on the basic ILP scheduling model (Section III-B) and the enhanced one (Section III-C), respectively. The former is used to compute the normal TT schedule, while the latter is aimed at computing the enhanced schedule for network updates. Both online and offline algorithms is implemented in the enhanced scheduler for comparison.

*3) Flow Generation:* All flows are generated randomly. The sole source and multiple destinations of each flow are randomly selected from all terminal devices to cover the unicast and multicast broadcast. Flow periods are chosen randomly from $\{8, 16, 32\,\text{ms}\}$ and frame lengths are generated between 64 and 1518 B randomly. The end-to-end delay of all flows is set to fixed 16 ms.

*4) Experiment Flow:* Given the old and new network configurations, the basic TT scheduler is used first to generate schedules $S, S^*$ and then the enhanced TT scheduler is applied to generate new conflict-free schedules. Finally the schedule is verified by checking conflicts between old and new flows. Schedule results and execution time of our enhanced ILP scheduler are collected. To compare with other update methods, we also record frame loss numbers and update duration (measured as the difference between minimum and maximum flow update times).

Since there are diverse available ILP solvers with different features and performance, our prototype system is built on MATLAB R2019a and YALMIP R20190425 [37] for comparison convenience. After comparing two dominant solvers, IBM ILOG CPLEX 12.9 and Gurobi 8.1, CPLEX is chosen due to its higher speed (about 2 times faster than Gurobi). All experiments are performed on a Ubuntu 19.04 workstation with an Intel Core i7-9700 CPU and 16GB DDR4 RAM.

## B. Case Study of the Train Network

Fig. 3 is the real topology of an ethernet train network. We choose 12 different bandwidth utilization thresholds of the new network configuration: 15%, 20%, 25%, 30%, 35%, 40%, 45%, 50%, 55%, 60%, 75%, and 90%. Pre-experiments on 30% bandwidth utilization with 10, 20, 30, 40, and 50 test cases show that the results of more test cases are consistent with those of 10 cases, and therefore in formal experiments, 10 test cases are generated for each threshold, in which 5 cases are for the topology change and the rest are for flow number variation. The detailed specifications are the following.

1) Topology change: The topology without dotted links in Fig. 3 is treated as the old topology $G$ and obtain the new topology $G^*$ by adding those dotted links for better redundancy. Flows in $F, F^*$ are identical. They are incrementally generated until reaching certain bandwidth utilization threshold.
2) Various flow numbers: The topology with dotted links in Fig. 3 is utilized for both old and new network configurations. Flows in $F^*$ are generated incrementally until reaching certain bandwidth utilization and flows in $F$ are randomly selected half from $F^*$.

*1) Schedulability and Execution Time:* In real industrial applications, the online scheduler is required to generate an available schedule in seconds when a network update occurs. Since our prototype system is implemented in MATLAB with no special optimization and both algorithms involve no intense matrix and array computation, there is a huge performance gap between our prototype and the practical C/C++ implementation, which is typically 10–100 (average 50) times faster in C/C++ [38], [39]. Therefore, 20 min is set as the online time limit in our experiments, equivalent to 60 s with 20 times speedup in C/C++. Meanwhile, the offline time limit is set to 1 h to balance the performance and quality.

TABLE III
ABBREVIATIONS

| Abbreviation | Implication |
|---|---|
| SDN | Software-Defined Networking |
| TSSDN | Time-Sensitive Software-Defined Networking |
| TSN | Time-Sensitive Networking |
| TT | Time-triggered |
| BE | Best-effort |
| SAE | Society of Automotive Engineers |
| PTP | Precision Time Protocol |
| RT-IoT | Real-Time Internet of Things |
| ILP | Integer Linear programming |
| MILP | Mixed Integer Linear programming |
| SMT | Satisfiability Modulo Theories |
| MC | Mixed-Criticality |
| Orion CEV | The Orion Crew Exploration Vehicle |

The test case is considered schedulable if the scheduler returns a feasible solution within time limit. Fig. 5(a) and Table I illustrate the schedulability of online and offline approaches for the train network. With increasing bandwidth utilization, additional constraints make it harder for the scheduler to find a feasible solution. The online schedulability is always 100% with bandwidth utilization less than 50%. Offline method shows a better schedulability at the cost of consuming more time. Since the network bandwidth utilization is no more than 30% [16], [40], [41] in typical industrial applications, both online and offline algorithms can satisfy the requirement. Fig. 5(b) shows the average algorithm execution time , which is regularly increasing with the bandwidth utilization and the maximum online execution time is under 175 s (around 9 s in C/C++) when bandwidth utilization is less than 30%. The online algorithm approximately saves half the time of offline method.

*2) Frame Loss and Update Time Duration:* Fig. 5(a) and (d) compare the maximum frame loss number and update duration of Time4, two-phase, Li's, and our mechanisms. Both Time4 and two-phase incur a large number of lost frames. Though fewer, Li's method still loses several frames and requires longer time to finish the update than other methods. On the contrary, our method guarantees no frame loss and introduces no extra update time at the same time.

## C. Case Study on Orion CEV Networks

Fig. 4 illustrates the real topology of the Orion CEV network. The same steps in Section V-B are followed to generate test cases, only switching the topology from train network to Orion CEV network.

*1) Schedulability and Execution Time:* Fig. 5(e) and (f) and Table II illustrate the schedulability and execution time of two algorithms in the Orion CEV network, respectively. Compared to the train network schedulability, test cases with bandwidth utilization under 0.6 are solved online properly while almost all test cases are solved by the offline algorithm. And test cases are solved faster than those of the train network. Only 106 s are required to schedule the cases with 30% bandwidth utilization.

TABLE IV
NOTATIONS OF ELEMENTS AND SETS

| Symbol | Implication |
|---|---|
| $v_i$ | the $i$-th device in the network |
| $f_i$ | the $i$-th flow |
| $(x, y)$ | the physical link between device $x$ and $y$ |
| $[x, y]$ | the dataflow link from device $x$ to $y$ |
| $p$ | the dataflow path |
| $f_i.prd$ | the period of flow $f_i$ |
| $f_i.len$ | the frame length of flow $f_i$ |
| $f_i.ddl$ | the end-to-end delay of flow $f_i$ |
| $f_i.src$ | the single sender of flow $f_i$ |
| $f_i.dst$ | the receiver set of flow $f_i$ |
| $f_i.vl$ | the virtual link set of flow $f_i$ |
| $f_i^{[v_k,v_l]}.ofst$ | the sending time of flow $f_i$ at dataflow link $[v_k, v_l]$ |
| $f_i^{v}.ofst$ | the sending time of flow $f_i$ at device $v$ if $v \notin f_i.dst$, otherwise the reception time of flow $f_i$ at $v$ |
| $f_i^{f_i.src}.ofst$ | the sending time of flow $f_i$ at its single source $f_i.src$ |
| $f_i^{[v_k,v_l]}.d^{proc}$ | the processing delay of flow $f_i$ at $[v_k, v_l]$ |
| $f_i^{[v_k,v_l]}.d^{que}$ | the queuing delay of flow $f_i$ at $[v_k, v_l]$ |
| $f_i^{[v_k,v_l]}.d^{send}$ | the sending delay of flow $f_i$ at $[v_k, v_l]$ |
| $f_i^{[v_k,v_l]}.d^{prop}$ | the propagating delay of flow $f_i$ at $[v_k, v_l]$ |
| $f_i^{[v_k,v_l]}.dl$ | the sum of processing, sending and propagating delays of flow $f_i$ at $[v_k, v_l]$ |
| $f_i^{[v_k,v_l]}.arrival$ | the arrival time of flow $f_i$ from $v_k$ to $v_l$ |
| $R_{i,j}^{[v_k,v_l]}, K_{i,j}^{[v_k,v_l]}$ $X_{i,j}^{[v_k,v_l]}$ | the ILP auxiliary variables corresponding to flow $f_i, f_j$ and dataflow link $[v_k, v_l]$ |
| $G$ | the undirected graph modelling the physical topology of a network |
| $V$ | the set of all switches and hosts |
| $E$ | the tuple set of linked devices |
| $L$ | the set of dataflow links |
| $F$ | the set of flows |
| $S$ | the flow schedule |
| $D$ | the set of all delays |
| $\_^*$ | the superscript $*$ means the corresponding value in the new network configuration |

2) *Frame Loss and Update Time Duration:* Fig. 5(g) and (h) explore the maximum frame loss number and update duration in the Orion CEV network, presenting the same relationship as those in Fig. 5(c) and (d). Time4 and two-phase lose numerous frames and Li's method consumes more update time, whereas our method keeps all frames with zero update duration.

## VI. CONCLUSION

We proposed a new flow scheduling mechanism to generate conflict-free schedules for TSSDN network updates. To maintain transmission consistency, the basic ILP model was extended with extra constraints to avoid potential flow conflicts during network updates. Two algorithms were designed for the model. The online algorithm consumed less time with slightly decreased schedulability than the offline one. Experiments on two real industrial networks demonstrated the effectiveness and scalability of our mechanism.

## REFERENCES

[1] M. Hermann, T. Pentek, and B. Otto, "Design principles for industrie 4.0 scenarios," in *Proc. 49th Hawaii Int. Conf. Syst. Sci.*, Jan. 2016, pp. 3928–3937.

[2] H. Kopetz and G. Bauer, "The time-triggered architecture," in *Proc. IEEE*, vol. 91, no. 1, pp. 112–126, Jan. 2003.

[3] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The time-triggered ethernet (TTE) design," in *Proc. 18th IEEE Int. Symp. Object-Oriented Real-Time Distrib. Comput.*, 2005, pp. 22–33.

[4] *Time-Triggered Ethernet*, Aerospace standard AS6802, SAE International Group, Warrendale, PA, USA, Nov. 2011.

[5] *IEEE Standard for Local and Metropolitan Area Networks – Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks– Corrigendum 1: Technical and Editorial Corrections*, IEEE Std 802.1AS-2011/Cor 1-2013 (Corrigendum to IEEE Std 802.1AS-2011), IEEE, pp. 1–128, Sep. 2013.

[6] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002), IEEE, pp. 1–269, Jul. 2008.

[7] R. Kumar *et al.*, "End-to-end network delay guarantees for real-time systems using SDN," in *Proc. IEEE Real-Time Syst. Symp.*, 2017, pp. 231–242.

[8] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive software-defined network (TSSDN) for real-time applications," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, 2016, pp. 193–202.

[9] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks," in *Proc. 31st IEEE Real-Time Syst. Symp.*, 2010, pp. 375–384.

[10] F. Pozo, W. Steiner, G. Rodriguez-Navas, and H. Hansson, "A decomposition approach for SMT-based schedule synthesis for time-triggered networks," in *Proc. IEEE 20th Conf. Emerg. Technol. Factory Autom.*, 2015, pp. 1–8.

[11] S. S. Craciunas and R. S. Oliver, "SMT-based task- and network-level static schedule generation for time-triggered networked systems," in *Proc. 22nd Int. Conf. Real-Time Netw. Syst.*, 2014, pp. 45–54.

[12] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, 2016, pp. 183–192.

[13] Z. Li *et al.*, "An enhanced reconfiguration for deterministic transmission in time-triggered networks," *IEEE/ACM Trans. Netw.*, vol. 27, no. 3, pp. 1124–1137, Jun. 2019.

[14] P. Vrba and V. Marik, "Capabilities of dynamic reconfiguration of multiagent-based industrial control systems," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 40, no. 2, pp. 213–223, Mar. 2010.

[15] Y. Ge, X. Liang, Y. C. Zhou, Z. Pan, G. T. Zhao, and Y. L. Zheng, "Adaptive analytic service for real-time internet of things applications," in *Proc. IEEE Int. Conf. Web Serv.*, 2016, pp. 484–491.

[16] N. Wang, Q. Yu, H. Wan, X. Song, and X. Zhao, "Adaptive scheduling for multicluster time-triggered train communication networks," *IEEE Trans. Ind. Informat.*, vol. 15, no. 2, pp. 1120–1130, Feb. 2018.

[17] T. Mizrahi and Y. Moses, "Software defined networks: It's about time," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.

[18] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 323–334, 2012.

[19] T. Mizrahi, E. Saat, and Y. Moses, "Timed consistent network updates in software-defined networks," *IEEE/ACM Trans. Netw.*, vol. 24, no. 6, pp. 3412–3425, Dec. 2016.

[20] J. Zheng, G. Chen, S. Schmid, H. Dai, and J. Wu, "Chronus: Consistent data plane updates in timed SDNs," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 319–327.

[21] J. Zheng *et al.*, "Scheduling congestion-free updates of multiple flows with chronicle in timed SDNs," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 12–21.

[22] S. Brandt, K.-T. Förster, and R. Wattenhofer, "On consistent migration of flows in SDNS," in *Proc. IEEE 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.

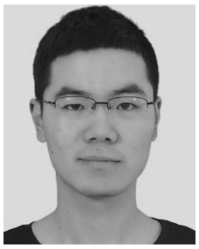
[23] X. Jin *et al.*, "Dynamic scheduling of network updates," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, 2014, pp. 539–550.

[24] A. Ludwig, J. Marcinkowski, and S. Schmid, "Scheduling loop-free network updates: It's good to relax!," in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2015, pp. 13–22.

[25] A. Ludwig, S. Dudycz, M. Rost, and S. Schmid, "Transiently secure network updates," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 44, no. 1, pp. 273–284, 2016.

[26] R. Obermaisser and A. Murshed, "Incremental, distributed, and concurrent scheduling in systems-of-systems with real-time requirements," in *Proc. IEEE Int. Conf. Comput. Inf. Technol.; Ubiquitous Comput. Commun.; Dependable, Auton. Secure Comput.; Pervasive Intell. Comput.*, 2015, pp. 1918–1927.

[27] N. G. Nayak, F. Dürr, and K. Rothermel, "Incremental flow scheduling and routing in time-sensitive software-defined networks," *IEEE Trans. Ind. Informat.*, vol. 14, no. 5, pp. 2066–2075, May 2017.
[28] F. Dürr and N. G. Nayak, "No-wait packet scheduling for IEEE time-sensitive networks (TSN)," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, 2016, pp. 203–212.
[29] K. Lee *et al.*, "MC-SDN: Supporting mixed-criticality scheduling on switched-ethernet using software-defined networking," in *Proc. IEEE Real-Time Syst. Symp.*, 2018, pp. 288–299.
[30] F. Pozo, G. Rodriguez-Navas, and H. Hansson, "Schedule reparability: Enhancing time-triggered network recovery upon link failures," in *Proc. IEEE 24th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, 2018, pp. 147–156.
[31] M. L. Raagaard, P. Pop, M. Gutiérrez, and W. Steiner, "Runtime reconfiguration of time-sensitive networking (TSN) schedules for fog computing," in *Proc. IEEE Fog World Congr.*, 2017, pp. 1–6.
[32] Q. Yu and M. Gu, "Adaptive group routing and scheduling in multicast time-sensitive networks," *IEEE Access*, vol. 8, pp. 37855–37865, 2020.
[33] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using SDN," *ACM SIGCOMM Comput. Commun. Rev.*, ACM, vol. 43, no. 4, pp. 27–38, 2013.
[34] T. Mizrahi and Y. Moses, "Time4: Time for SDN," *IEEE Trans. Netw. Serv. Manage.*, vol. 13, no. 3, pp. 433–446, Sep. 2016.
[35] Q. Yu, X. Zhao, H. Wan, Y. Gao, C. Lu, and M. Gu, "Handling scheduling uncertainties through traffic shaping in time-triggered train networks," in *Proc. IEEE/ACM 25th Int. Symp. Qual. Serv.*, 2017, pp. 1–6.
[36] D. Tămaş-Selicean and P. Pop, "Optimization of TTethernet networks to support best-effort traffic," in *Proc. IEEE Emerg. Technol. Factory Autom.*, 2014, pp. 1–4.
[37] J. Löfberg, "YALMIP : A toolbox for modeling and optimization in MATLAB," in *Proc. Comput. Aided Control Syst. Des. Conf.*, Taipei, Taiwan, 2004.
[38] C. Ennis, L. M. Kuhn, A. Sievers, and S. Russell, "An analysis of MATLAB's software performance interfaced with high-level C language for expediting numerical integration technique," *J. Comput. Sci. Colleges*, vol. 33, no. 4, pp. 84–91, 2018.
[39] T. Andrews, "Computation time comparison between MATLAB and C++ using launch windows," 2012.
[40] *Electronic Railway Equipment: Train Communication Network (TCN): Part 2–6*, IEC 61375-2-6:2018, International Electrotechnical Commission (IEC), Geneva, Switzerland, 2018.
[41] *Industrial Communication Networks: Fieldbus Specifications: Part 1*, IEC 61158-1:2019, International Electrotechnical Commission (IEC), Geneva, Switzerland, 2019.


**Zaiyu Pang** received the B.S. degree in computer science and technology from the School of Computer Science and Technology, Jilin University, Changchun, China, in 2018. He is currently working toward the master's degree in software engineering with the School of Software, Tsinghua University, Beijing, China.

His current research interests include the real-time networks and stream processing systems.

**Xiao Huang** received the B.S. degree in software engineering from the School of Software, Nanjing University, Nanjing, China, in 2019. He is currently working toward the master's degree in software engineering with the School of Software, Tsinghua University, Beijing, China.

His current research interests include the job scheduling on distributed computer clusters.

**Zonghui Li** received the B.S. degree in computer science from the Beijing Information Science and Technology University, Beijing, China, in 2010, and the M.S. degree in integrated circuits and Ph.D. degree in software engineering from the Institute of Microelectronics and the School of Software, Tsinghua University, Beijing, China, in 2014 and 2019, respectively.

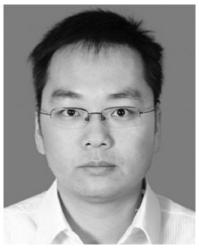
He is currently an Assistant Professor with the School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China. His current research interests include embedded and high performance computing, real-time embedded systems, especially for industrial control networks and fog computing.

**Sukun Zhang** received the B.S. degree in software engineering from the School of Software, Beijing Jiaotong University, Beijing, China, in 2018. She is currently working toward the master's degree in software engineering with the School of Software, Tsinghua University, Beijing, China.

Her current research interests include the real-time networks.

**Yanfen Xu** received the B.S. degree in computer science from Shijiazhuang Tiedao University, Shijiazhuang, China, in 2002.

She joined CRRC Qingdao Sifang Rolling Stock Research Institute CO., LTD., Qingdao, China, and was appointed as Professorate Senior Engineer, in 2017. Her current research interests include train network, train control and management system.


**Hai Wan** received the B.S. degree from National University of Defense Technology, Changsha, China, in 2003, and the Ph.D. degree from Tsinghua University, Beijing, China, in 2011, both in computer science and technology.

From 2011 to 2012, he was a Postdoc Researcher with INRIA, Nancy, France. From 2013, he is with the faculty of Tsinghua University, Beijing, China. He is currently a Research Associate Professor with the School of Software, Tsinghua University. His current research interests include real time systems, embedded systems.

**Xibin Zhao** received the B.S., M.E., and Ph.D. degrees from the School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang, China, in 1994, 2000, and 2004, respectively, all in computer science and technology.

He is currently an Associate Professor with the School of Software, Tsinghua University, Beijing, China. His current research interests include reliability analysis of hybrid network systems and information system security.