IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 24, 2019, accepted January 3, 2020, date of publication January 7, 2020, date of current version January 14, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2964690

Real-Time Scheduling of Massive Data in Time Sensitive Networks With a Limited Number of Schedule Entries

XI JIN[1,2,3], CHANGQING XIA[1,2,3], NAN GUAN[4], CHI XU[1,2,3], DONG LI[1,2,3], YUE YIN[5], AND PENG ZENG[1,2,3]

[1]State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110016, China
[2]Institutes for Robotics and Intelligent Manufacturing, Chinese Academy of Sciences, Shenyang 110016, China
[3]Key Laboratory of Networked Control Systems, Chinese Academy of Sciences, Shenyang 110016, China
[4]Department of Computing, The Hong Kong Polytechnic University, Hong Kong
[5]School of Economics, Liaoning University, Shenyang 110036, China

Corresponding authors: Xi Jin (xijin@ieee.org) and Peng Zeng (zp@sia.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1700200, in part by the National Natural Science Foundation of China under Grant 61972389, Grant 61903356, and Grant 61803368, in part by the Youth Innovation Promotion Association of the Chinese Academy of Sciences, in part by the Liaoning Provincial Natural Science Foundation of China under Grant 2019-YQ-09, Grant 20180520029, and Grant 20180540114, and in part by the China Postdoctoral Science Foundation under Grant 2019M661156.

ABSTRACT Time sensitive networks support deterministic schedules over Ethernet networks. Due to their high determinism, high reliability and high bandwidth, they have been considered as a good choice for the backbone network of industrial internet of things. In industrial applications, the backbone network connects multiple industrial field networks together and has to carry massive real-time packets. However, the off-the-shelf time-sensitive network (TSN) switches can deterministically schedule no more than 1024 real-time flows due to the limited number of schedule table entries. The excess real-time flows have to be delivered by best-effort services because the switch only supports the two scheduling services. The best-effort services can reduce average delay, but cannot guarantee the hard real-time constraints of industrial applications. To make the limited number of schedule table entries support more real-time flows, first, we relax scheduling rules to reduce the requirement for schedule table entries and formulate the process of transmitting packets as a satisfiability modulo theories (SMT) specification. Then, we divide the SMT specification into multiple optimization modulo theories (OMT) specifications so that the execution time of solvers can be reduced to an acceptable range. Second, we propose fast heuristic algorithms that combine schedule tables and packet injection control to eliminate scheduling conflicts. Finally, we conduct extensive evaluations. The evaluation results indicate that, compared to existing algorithms, our proposed algorithm requires only one-twentieth the number of schedule entries to schedule the same flow set.

INDEX TERMS Industrial Internet of Things, massive data, real-time scheduling, time sensitive networks.

I. INTRODUCTION

Industrial internet of things (IIoT) has changed the way that manufacturing is carried out [1]. For example, IIoT supplies connectivity between customers and production lines so that customers can guide the production process directly [2], and the IIoT connectivity allows data exchange among all industrial devices so that the whole production process can change rapidly to adapt to new products [3]. Industrial networks are the most fundamental and important part of IIoT. To satisfy the strict requirements of industrial applications, industrial networks have to transmit data deterministically, such as precisely transmitting control data, supplying adequate bandwidth for video streams, and managing each packet in massive-machine-type communications.

Many industrial wired and wireless networks have been proposed to handle these high-requirement scenarios, e.g., WIA-FA [4], WIA-PA [5], RT-WiFi [6], WirelessHP [7], WirelessHART [8], TTEthernet [9], and time sensitive networks (TSNs) [10]. However, there is still no single network

The associate editor coordinating the review of this manuscript and approving it for publication was Chunsheng Zhu.

framework that can guarantee all industrial requirements. Heterogeneous networks constituted by a wired backbone network and multiple wireless field networks are considered as the future solutions of industrial networks, for example, the wired fronthaul/backhaul and cell sites in 5G [11]. The wireless field networks, which are flexible and easy to deploy, connect massive devices to the upper management; the wired backbone network, which has high bandwidth and high reliability, carries all data transmitted by the wireless field networks. Thus, the wired backbone network should support all performance-optimization technologies that wireless field networks have done. If not, the optimization effect achieved in wireless field networks will be reduced, and even totally disappear, in wired backbone networks. For example, when the backbone network supports only best-effort services, regardless of which field network is used, the end-to-end delay of a packet is still uncontrollable and nondeterministic.

TSNs are a good choice for the backbone network because they support deterministic schedules over Ethernet networks, and provide high-bandwidth, high-reliability and high-determinism connectivity. Almost all performance indicators of TSNs are better than those of wireless field networks. However, a performance-optimization problem that wireless field networks do not need to consider but that backbone networks have to handle is how to transmit massive, real-time data deterministically. Wireless field networks support real-time communications, but do not need to transmit massive data because a single field network will not cover a large-scale region. However, when multiple field networks submit their data to the backbone TSN, the backbone TSN must consider how to handle massive data. There is no related work that has considered how to schedule massive data in TSNs under real-time constraints (as described in Section II). Therefore, in this paper, we will propose real-time scheduling algorithms that can transmit massive data deterministically in TSNs.

There are two challenges to carrying out real-time scheduling of massive data. First, in off-the-shelf TSN switches, e.g., [12]–[14], the limited capacity of schedule tables makes it difficult for them to handle massive data. TSN switches support both deterministic and nondeterministic communications. For deterministic communications, the schedule tables record the precise times to control egress ports so that the transmitting times of forwarded flows can be precisely controlled. However, due to the limited number of schedule table entries in switch chips, the off-the-shelf TSN switches support no more than 1024 deterministic data flows [12]–[14]. The excess deterministic data flows have to be delivered by best-effort services. Although a new large-capacity chip can be created to support more deterministic data flows, the relevant time costs and price costs cannot be ignored. Second, massive data with different requirements cannot be aggregated to reduce the difficulty of scheduling. TSN switch chips do not support aggregating functions. Only micro control units (MCUs) that are connected to switch chips can aggregate data packets. However, if data packets are sent to MCUs, the exchange of data packets between the switch chips and MCUs will require plenty of time and thus reduce the system performance. Therefore, data can be aggregated only before they are injected into the TSNs and disaggregated only when they leave the TSNs, i.e., the aggregated data packets must have the same path, the same release time and the same real-time constraint. This severe aggregation is not suitable for flexible and diversified machine-to-machine communications. Therefore, our proposed scheduling algorithms should be constrained by the capacity of schedule tables and non-aggregated.

To schedule massive real-time data packets using a limited number of schedule entries, we make the following contributions.

(1) In some cases, TSN switches do not need to record port controls for each packet. For example, when a switch forwards packets immediately after it receives them, the precise transmitting time is determined by its previous switch. Thus, we relax the constraint that switches have to record the port controls for all packets and formulate the process of transmitting packets as a satisfiability modulo theories (SMT) specification [15]. However, the execution time of SMT solvers increase exponentially with the problem size. To obtain a feasible solution in an acceptable time, we divide the SMT specification into multiple optimization modulo theories (OMT) specifications [16]. The dividing strategy makes the execution times of solvers increase linearly with the problem size. Our evaluations indicate that in 2 days, the OMT-based algorithm can solve 40000 packets, while the SMT-based algorithm can solve only approximately 400 packets.

(2) The problem we focus on is NP-complete. There is no polynomial time algorithm for finding a solution for our problem. Hence, we adopt heuristic algorithms to improve the scalability of our work. First, the NGC algorithm is presented to eliminate all conflicts by adjusting the injection times. In this algorithm, switches do not need to record any time to transmit packets, and once a packet is injected into the network, it can be transmitted to its destination without interruption. Then, we analyze the upper bound of end-to-end delays in this algorithm and find two factors that make data flows difficult to schedule. Based on the analysis, we propose the MF algorithm that uses schedule entries to support post-scheduling adjustments. The evaluation results indicate that, compared to existing algorithms, the MF algorithm requires only one-twentieth the number of schedule entries to schedule the same 32000 packets.

The rest of the paper is as follows: Section II first reviews two categories of related work, including specification-based algorithms and heuristic algorithms, and then explains why the existing work cannot be used in our problem. Section III details our system model and problem. Section IV and Section V propose specification-based algorithms and heuristic algorithms to solve our problem, respectively. Section VI evaluates our proposed algorithms based on massive test cases. Section VII discusses the underlying meaning of this paper. Section VIII concludes the paper.

II. RELATED WORK

Real-time scheduling has been widely studied in uniprocessor/multiprocessor systems [17], [18] and networked systems [19], [20]. However, the algorithms designed for other systems cannot obtain the expected results in TSNs due to their special transmitting mechanisms. TSNs represent a set of IEEE standards, in which the IEEE standard 802.1Qbv defines the architecture of TSN schedulers, but does not specify scheduling algorithms. In some recent papers, the TSN scheduling problem has been considered. The work in [21] formulates the scheduling problem as SMT and OMT problems, and then calculates static schedules using an off-the-shelf solver. In addition to SMT/OMT solvers, integer linear programming (ILP) solvers can be used to generate schedules. The work in [22] proposes various ILP formulations to generate incremental schedules, and its results are close to the optimal solutions of the corresponding static scheduling problem. The work in [23] formulates the joint routing and scheduling problem as an ILP and explores the real limitations by evaluating extensive test cases with widely varying parameters. The work in [24] uses an ILP approach to maximize the resources available for audio-video-bridging and best-effort traffic in TSNs. Although the results obtained by off-the-shelf solvers are optimal or close to optimal, their execution times become unacceptable as the system size increases.

Heuristic algorithms are faster than off-the-shelf solvers and have been used in TSNs. The work in [25] proposes a K-shortest path heuristic and a meta-heuristic randomized adaptive search procedure to solve the joint routing and scheduling problem for time-triggered traffic, and at the same time reduce the end-to-end delays of audio-video-bridging traffic. The work in [26] develops two algorithms to schedule multicast time-triggered flows: the first algorithm is based on genetic algorithm and generates schedules under real-time constraints; the second one is based on fixed routings and implemented as a list scheduler. The work in [27] presents a heuristic incremental scheduling algorithm to minimize the impact of reconfiguring schedules on existing flows. The work in [28] proposes a configuration agent architecture and a simple scheduling heuristic such that the schedules of dynamic flows can be quickly generated and configured. However, these algorithms do not consider the capacity limitation of network switches.

The work in [29], [30] introduces capacity limitations into their problem models to make the scheduling problem more similar to real systems. However, the method proposed in [29] generates schedules using solvers, and its formulation is to optimize the individual jitter of each flow; the objective optimized in [30] is to minimize the maximal end-to-end delay, i.e., the problem focused on by [30] is not a typical real-time scheduling problem with deadlines. The work [31] splits a problem into several subproblems so that capacity limitations can be ignored. Although the similar method can be used in TSNs to distribute data flows to multiple switches, the cost of the extra switches is unaffordable to users when massive flows need to be distributed. In contrast, our proposed algorithms are for a normal real-time scheduling model with a limited number of schedule entries and can quickly generate schedules for massive data packets.

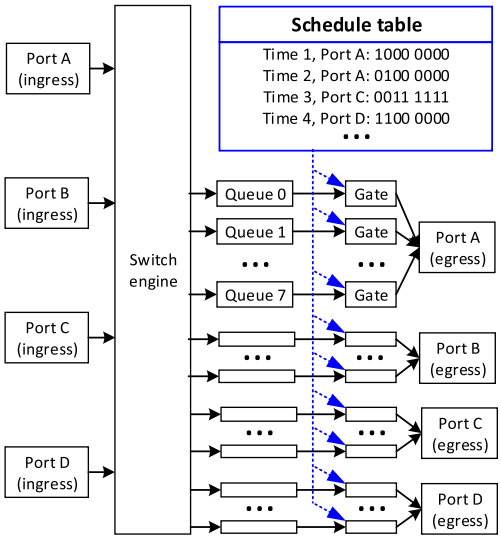


FIGURE 1. Architecture of a TSN switch with 4 ports.

III. SYSTEM MODEL AND PROBLEM STATEMENT

In this section, we will detail our system model and problem. The symbols used in this paper is summarized in Appendix.

A. NETWORK

A time-sensitive network is characterized by a three-tuple $< N, W, L >$. The node set $N = \{n_1, n_2, ...\}$ includes all switches and gateways. The switch set W is a subset of N. Gateways connect other networks, e.g., RT-WiFi, WIA-PA, and WirelessHP, to TSNs. We use $n_{i,j}$ to denote the j-th port of switch n_i. The element $l_{i,g}$ in the link set L denotes that there is a cable between nodes n_i and n_g. Our algorithms are suitable for all topologies. To describe the model more clearly, we specify the network topology is mesh since mesh topologies can reduce to the other topologies.

The architecture of a TSN switch with 4 ports is shown in FIGURE 1. Ports and cables are both full-duplex. After a packet is injected into a switch from an ingress port, it is switched to its destination egress port according to routing tables, and then stored into a queue of the egress port. Each egress port has multiple queues, and the queue with a smaller ID has lower priority. The packet is stored in the queue that has the same ID as that of the priority code point (PCP) segment in the packet header. To avoid conflicts among packets, one queue is used by at most one packet at a time [21]. Each queue connects to a gate. If the gate is opened, the packet stored in the queue is allowed to use the egress port. Otherwise, the packet waits until the gate is opened. When multiple gates are opened, the nonempty queue with the highest priority uses the egress port.

Gates are controlled by schedule tables. Each entry in the table contains time, port, and gate states [13], [14]. For example, in FIGURE 1, at Time 1, the first entry is applied to Port A, and gate states 1000 0000 mean that only the gate of the first queue, i.e., Queue 0, is opened, and the others are closed. Then, at Time 2, the gate states change to 0100 0000 that mean the gate of Queue 0 is closed, and the gate of Queue 1 is opened. A switch contains at most T entries. If not all entries are used, the time of unavailable entries is marked as 0. Available entries are stored in the order of time, i.e., if Time i and Time j are not zero and $i > j$, then Time $i >$ Time j. After all available entries are executed, the first entry is executed again. Note that the last available entry has the same control information as that of the first entry because the last entry is added to mark the running time of the penultimate entry. The limitation of a table capacity T is the main reason why it is difficult for TSN switches to support massive data communications.

Some related papers adopt other kinds of schedule tables. For example, in the paper [29], a switch has multiple schedule tables (also called gate control lists), and a schedule table corresponds to only one port. Thus, schedule entries do not need to contain port information. Regardless of which kind of schedule tables is adopted, our proposed algorithms can still be used as long as one entry corresponds to only one port.

B. DATA FLOW

Industrial data flows can be divided into two categories: critical flows and noncritical flows. Critical flows are hard real-time, while noncritical flows are best-effort. Our proposed algorithms schedule only critical flows. The remaining resources can be used by noncritical flows. In the following, if no specific description is given, flows refer to critical flows. A flow f_a in the flow set F is characterized by a five-tuple $< p_a, d_a, q_a, \gamma_a, \Pi_a >$, which denote the period, relative deadline, queue ID, end-to-end delay, and routing path, respectively. Each flow generates a packet periodically with a period p_a. In addition to the backbone TSN, packets are transmitted in field networks. Hence, two time intervals in a period are occupied by field networks as shown in FIGURE 2(a). Only the middle interval belongs to the backbone TSN, i.e., our proposed algorithms only consider the middle intervals. The scheduling problem between field networks and the backbone network exceeds the scope of our paper and is not considered here. The lengths of these time intervals can be zero when sources or destinations are gateways. This does not impact our proposed algorithms.

If flows are allowed to start at any time, the schedules cannot be generated due to the unknown conflicts among packets. Thus, we adjust flows to make them start at the same time. The adjustment is easy to implement because all nodes are time synchronized, and the gateways can control the injection times of flows. Then, relative deadlines are equal to the available time intervals in which packets can be transmitted in TSNs as shown in FIGURE 2(b), i.e., $d_a \leq p_a$. Flow f_a releases its k-th packet at time $p_a \times k + 1$, and its absolute

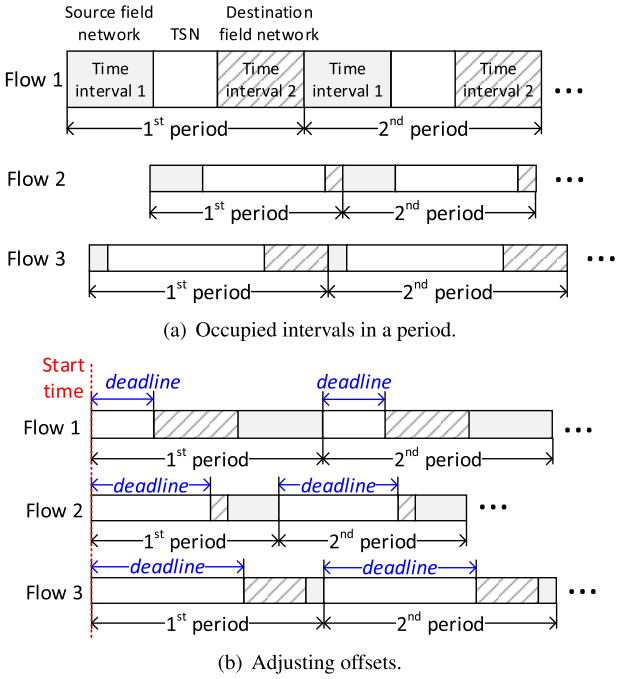


FIGURE 2. Flow model.

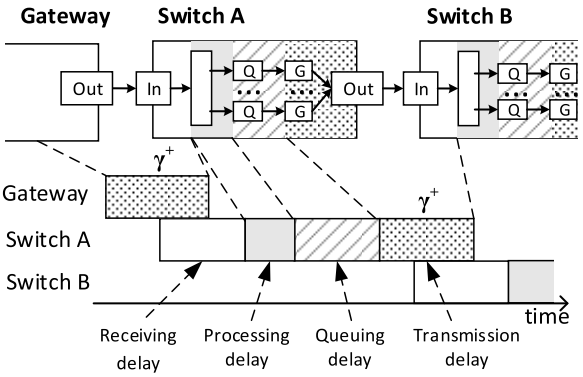


FIGURE 3. Delay of a packet in a switch.

deadline is $p_a \times k + d_a$. We consider only how to schedule in the first hyperperiod H, which is defined as the least common multiple of their periods, i.e., $H = LCM\{p_1, p_2, ...\}$, because after the first hyperperiod, the subsequent schedules are periodically repeated.

The queue ID q_a is written into the PCP segment in the packet header before packets are injected into the TSNs. The routing path Π_a is a node set, i.e., $\Pi_a = \{\pi_{a1}, \pi_{a2}, ...\}$ and $\Pi_a \subseteq N$. Since routing is already well-studied [32], [33], we do not propose any routing algorithm and consider that the routes are already generated based on some existing algorithms.

The delay of a packet is composed of four delays, including the receiving delay, processing delay, queuing delay, and transmission delay γ_a^+, as shown in FIGURE 3. The receiving delay is from the time that the packet enters the switch to the time that the packet is completely received. The processing delay is needed by the switch engine.

Since the processing delay is less than $1\mu s$, we ignore it in our problem. Then, the packet might be blocked in the queue. Thus, the queuing delay depends on schedules, and can be zero. The transmission delay is the amount of time needed to serialize and propagate the packet on the cable. Serializing delays and propagation delays should be listed separately because queues are still occupied during serializing delays. However, propagating a packet in a single cable requires at most 556 *ns* [34], and in TSNs, the synchronization error is approximately $1\mu s$. Therefore, we merge propagation delays and serializing delays together. Queues are occupied in all of the delays. The time durations spent in different nodes overlap each other since different parts of a serialized packet are processed in two adjacent nodes simultaneously. We use delay γ_a to denote the time it takes for a packet to transmit from its source to its destination without interruption.

TSNs adopt the IEEE 802.1AS standard to synchronize nodes. The only effect of the synchronization standard on our problem is that time synchronization introduces time errors to transmission delays. To simplify the description, we consider that the time synchronization errors are contained in transmission delays and will not be mentioned in the following.

C. SCHEDULING PROBLEM

To deterministically schedule massive data packets, the following four rules have to be followed.

1) A queue cannot be shared between critical packets and noncritical packets even at different times. Noncritical packets are uncontrollable. We cannot predict when noncritical packets will occupy queues and how noncritical packets will affect critical packets. Therefore, to make critical packets controllable, we assign several dedicated queues to them, and then the remaining queues can be used by noncritical queues.

2) Although critical packets and noncritical packets share the same network resources, scheduling critical packets does not require considering the noncritical packets. To maximize the bandwidth of noncritical data flows, their gates are always opened. However, noncritical transmissions cannot preempt critical transmissions because the queues used by critical packets have higher priorities than those used by noncritical packets. Although, critical packets can preempt noncritical packets, we still do not need to consider the preemption process. This is because guard bands, which are inserted before critical packets to keep them from being affected by unfinished noncritical packets, can be integrated into noncritical packets.

3) When multiple queues have critical packets, they cannot be opened at the same time. For example, there are two packets in two queues, and their gates are opened. In a regular case, the high-priority packet can use the egress port, and the low-priority packet is blocked. However, if the high-priority packet loses, the low-priority packet will be sent to the cable before its expected time. Then, the subsequent transmissions will be affected, and their determinism will be difficult to guarantee. Therefore, we do not permit the queues that are storing critical packets to be opened at the same time.

4) If a critical queue has no packets, its gate can remain open. Note that even though a high-priority gate is opened, the low-priority queue with opened gate can use the egress port as long as there is no packet in the higher-priority queue [35]. Based on this rule, we can reduce the number of closings, i.e., the number of table entries.

Therefore, based on the four scheduling rules, given the network $< N, W, L >$, the critical flow set F and the number of queues used by critical packets, our objective is to generate schedules for all critical flows to satisfy the following requirements.

- *Real-time requirement*: All of the critical packets are delivered to destinations before their deadlines.
- *No resource conflict*: Every queue and every link serves at most one packet at any time.
- *Resource limitation*: For each switch, the number of schedule entries used is not greater than the limitation T.

Since the schedules must be deterministic, we adopt fine-grained scheduling algorithms. Comparing with coarse-grained algorithms that specify a long time duration to transmit all of the critical packets, our fine-grained algorithms will assign a link and a time duration to each transmission so that end-to-end delays are controllable and deterministic.

A flow set is *schedulable*, if it has a feasible schedule that meets all the requirements. A scheduling algorithm is called *optimal*, if it can find a feasible schedule whenever one exists. If there are enough table entries to close the gates after each packet is sent, our problem is the same as the problem in [29]. Since the problem in [29] is NP-complete, our problem is also NP-complete.

To solve our NP-complete problem, we propose multiple algorithms that are briefly introduced in FIGURE 4. First, since SMT algorithms are optimal for NP-complete problems, we use SMT specifications to state our problem, and use SMT solvers to generate schedules (in Section IV-A). However, the execution time of SMT solvers increases exponentially with the problem size. For simple scheduling problems, SMT solvers have to spend tens of minuets to generate the schedules of tens of flows [21], [29]. Thus, SMT solvers cannot solve our SMT specification in acceptable time because our problem model is more complex than that of any existing work. Therefore, based on the SMT specification we propose an OMT-based algorithm that can significantly reduce the execution time (in Section IV-B).

Second, since the OMT-based algorithm is still restricted by third-party solvers, to further improve the scalability of our proposed algorithms, we propose heuristic algorithms to make a tradeoff between performance and efficiency. Our first heuristic algorithm adopts the minimum number of schedule entries to transmit packets (in Section V-A), and then we analyze why the first heuristic algorithm cannot schedule more packets (in Section V-B). Based on the analysis, we propose a second heuristic algorithm that

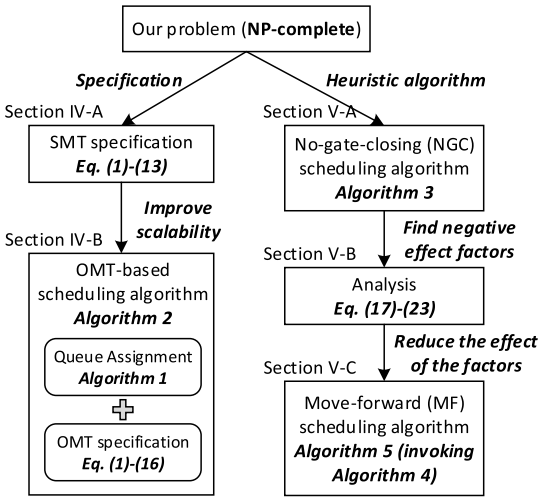


FIGURE 4. Overview.

adopts the unused schedule entries to transmit more packets (in Section V-C).

IV. ALGORITHMS BASED ON SMT/OMT

SMT specifications are based on first-order logic, and a SMT problem is to determine whether the specification is satisfiable or not. OMT is an extension of SMT and combines SMT with optimization procedures. To describe our problem in detail, we formulate the problem as an SMT specification. Then, we divide the problem into multiple OMT specifications to improve its scalability.

A. SMT SPECIFICATION

Our objective is to assign queues to flows and compute a schedule table E_i for each switch n_i ($\forall n_i \in W$). Each entry $e_{i,y}$ in table E_i is denoted as $< t_{i,y}, v_{i,y,1}, \ldots, v_{i,y,V}, c_{i,y,1}, \ldots, c_{i,y,Q} >$. The entry starts to run at time $t_{i,y}$. Not all entries in the schedule table E_i are used. If the entry is not used, its $t_{i,y}$ is set to 0. A switch has V ports, and each egress port has Q queues. If $v_{i,y,j} = 1$ ($j \in [1, V]$), entry $e_{i,y}$ is available to Port j. If $c_{i,y,r} = r$ ($r \in [1, Q]$), the gate of the r-th queue is opened. Otherwise, it is closed. We use symbol k ($k \in [0, \frac{H}{p_a})$) to distinguish the packets that are generated by flow f_a. For the k-th packet generated by flow f_a, its b-th hop transmission is called transmission $\tau^k_{a,b}$. We use $\alpha^k_{a,b}$ and $\beta^k_{a,b}$ to denote the starting time and the ending time of transmission $\tau^k_{a,b}$, i.e., $\beta^k_{a,b} - \alpha^k_{a,b} = \gamma^+_a$. Thus, the SMT problem has to respect the following constraints.

1) Range constraint: Based on our model and problem, the ranges of all variables are as follows.

$$\begin{aligned}&\forall n_i \in W, \quad \forall y \in [2, T],\ t_{i,1} = 0,\\ &((t_{i,y} = 0) \wedge \bigwedge_{\forall z \in [y+1, T]} (t_{i,z} = 0))\\ &\vee((t_{i,y} > 0) \wedge (1 \le t_{i,y-1} < t_{i,y} \le H)).\end{aligned} \tag{1}$$

$$\forall n_i \in W, \quad \forall y \in [1, T],\ \forall r \in [1, Q],\ c_{i,y,r} \in \{0, r\}. \tag{2}$$

$$\forall n_i \in W, \quad \forall y \in [1, T],\ \forall j \in [1, V],\ v_{i,y,j} \in \{0, 1\}. \tag{3}$$

$$\forall f_a \in F, \quad 1 \le q_a \le Q. \tag{4}$$

$$\begin{aligned}&\forall f_a \in F, \quad \forall b \in [1, |\Pi_a|),\ \forall k \in [0, \frac{H}{p_a}),\\ &k \times p_a + 1 \le \alpha^k_{a,b} < \beta^k_{a,b} \le d_a + k \times p_a,\\ &\beta^k_{a,b} - \alpha^k_{a,b} = \gamma^+_a.\end{aligned} \tag{5}$$

2) Path sequence constraint: In a routing path, the $(b+1)$-th hop starts after the b-th hop finishes.

$$\forall f_a \in F, \quad \forall b \in [1, |\Pi_a| - 1),\ \forall k \in [0, \frac{H}{p_a}),\ \alpha^k_{a,b+1} > \beta^k_{a,b}. \tag{6}$$

3) Real-time constraint: All packets have to be delivered to their destinations before their absolute deadlines.

$$\forall f_a \in F, \quad \forall k \in [0, \frac{H}{p_a}),\ \beta^k_{a,|\Pi_a|-1} \le d_a + k \times p_a. \tag{7}$$

This equation has been included in Constraint 1).

4) Link conflict constraint: If two packets use the same link, the times during which they pass through the link cannot overlap.

$$\begin{aligned}&\forall f_a, f_g \in F, a \ne g, \quad \forall b \in [1, |\Pi_a|),\ \forall h \in [1, |\Pi_g|),\\ &\forall k \in [0, \frac{H}{p_a}), \forall m \in [0, \frac{H}{p_g}),\\ &\neg(\pi_{a,b} = \pi_{g,h}) \vee \neg(\pi_{a,b+1} = \pi_{g,h+1})\\ &\vee(\alpha^k_{a,b} > \beta^m_{g,h}) \vee (\alpha^m_{g,h} > \beta^k_{a,b}).\end{aligned} \tag{8}$$

5) Queue conflict constraint: If two packets use the same queue, the times at which they are stored in the queue cannot overlap.

$$\begin{aligned}&\forall f_a, f_g \in F, a \ne g, \quad \forall b \in [2, |\Pi_a|),\ \forall h \in [2, |\Pi_g|),\\ &\forall k \in [0, \frac{H}{p_a}), \quad \forall m \in [0, \frac{H}{p_g}),\\ &\neg(\pi_{a,b} = \pi_{g,h}) \vee \neg(\pi_{a,b+1} = \pi_{g,h+1}) \vee \neg(q_a = q_g)\\ &\vee(\beta^k_{a,b} < \alpha^m_{g,h-1}) \vee (\beta^m_{g,h} < \alpha^k_{a,b-1}).\end{aligned} \tag{9}$$

6) Time validity constraint: Only schedule table E_i is recorded in the TSN switch. There must be a relationship between $\alpha^k_{a,b}$, $\beta^k_{a,b}$ and E_i so that the schedules in switches can guarantee the above constraints. We know that $\beta^k_{a,b}$ is equal to $\alpha^k_{a,b}$ plus the transmission delay γ^+_a. Therefore, we consider only $\alpha^k_{a,b}$ in this constraint. $\alpha^k_{a,b}$ has to satisfy the condition that it is the first opening time after the end of its previous hop. We use $A(x)$ to check the condition. Thus,

$$\begin{aligned}&\forall f_a \in F, \quad \forall b \in [2, |\Pi_a|),\ port(f_a, \pi_{a,b}) = n_{i,j},\\ &\forall k \in [0, \frac{H}{p_a}), \quad \bigvee_{\forall g \in [k \times p_a + 1, k \times p_a + d_a]} ((\alpha^k_{a,b} = g) \wedge A(g)),\end{aligned} \tag{10}$$

where $port(f_a, \pi_{a,b})$ denotes the port that in node $\pi_{a,b}$, f_a is sent from, and

$$\begin{aligned}A(x) = &((x > \beta^k_{a,b-1}) \wedge B(x))\\ &\wedge \bigwedge_{\forall h \in [k \times p_a + 1, x)} ((h \le \beta^k_{a,b-1}) \vee \neg B(h)).\end{aligned} \tag{11}$$

b starts from 2 because the first node in the routing path is not a TSN switch. In $A(x)$, if the value x satisfies the condition, and if, before the value x, there is no value that satisfies the condition, then $A(x)$ is true. $B(x)$ is used to check if the gate is opened at time x.

$$B(x) = \bigvee_{\forall e_{i,y} \in E_i} ((t_{i,y} = 0) \vee (v_{i,y,j} \wedge \bigvee_{\forall r \in [1,Q]} (c_{i,y,r} = q_a) \wedge \bigwedge_{\forall e_{i,z} \in E_i, z \neq y} ((t_{i,z} = 0) \vee \neg v_{i,z,j} \vee (t_{i,z} < t_{i,y} \leq x) \vee (t_{i,z} > x)))). \quad (12)$$

If there exists an entry $e_{i,y}$ that is the last entry to control the queue of flow f_a before time x, and if the entry is to open the queue of flow f_a, then $B(x)$ is true.

7) Time duration validity constraint: During the time duration $[\alpha^k_{a,b}, \beta^k_{a,b}]$, the corresponding gate must always be opened to send $\tau^k_{a,b}$ continuously.

$$\forall f_a \in F, \quad \forall k \in [0, \frac{H}{p_a}), \ \forall b \in [2, |\Pi_a|), \ part(f_a, \pi_{a,b}) = n_{i,j}, \bigwedge_{\forall e_{i,y} \in E_i} ((t_{i,y} = 0) \vee \neg v_{i,y,j} \vee ((t_{i,y} < \alpha^k_{a,b}) \vee (\beta^k_{a,b} < t_{i,y}) \vee ((\alpha^k_{a,b} \leq t_{i,y} \leq \beta^k_{a,b}) \wedge \bigvee_{\forall r \in [1,Q]} (c_{i,y,r} = q_a)))). \quad (13)$$

If an entry is in the time duration, i.e., $\alpha^k_{a,b} \leq t_{i,y} \leq \beta^k_{a,b}$, the corresponding gate must be opened.

B. AN ALGORITHM BASED ON OMT

As described in FIGURE 4, to improve the scalability of our proposed algorithms, we break down an SMT problem into multiple OMT problems. These OMT problems are the subproblems of the original SMT problem and independent of each other. For each subproblem, an OMT solver needs to be invoked once to schedule Δ packets, where Δ is a parameter given by users or selected based on the execution time of the OMT solver to make a trade-off between execution time and efficiency. As the problem size increases, the number of subproblems increases, while the subproblem size is not changed. Thus, the OMT solver is invoked more times, but the execution time of each invocation is basically unchanged. Therefore, the execution time of scheduling all the packets increases linearly with the problem size.

Before dividing the original problem, we must assign queues to flows because if the queue assignment is divided into multiple subproblems, the packets that belong in the same flow may be assigned to different queues. Algorithm 1 shows our assignment method. The heavier the workload is, the greater the number of conflicts that occur, resulting in more gate controls being needed to isolate transmissions. Our objective of assigning queues is to balance their workloads.

In Algorithm 1, we first calculate the utilization ρ_a that a flow introduces to a node (line 1). Then, each flow selects the queue with the minimal workload to use (lines 3–5), and the utilization of the selected queue is updated (line 6). $\hat{\rho}_{t,i,j,g}$ denotes the utilization of the g-th queue in $n_{i,j}$ at time t. To find the maximum $\hat{\rho}$, all possible t should be checked. However, we know that all flows release packets at time 1, i.e., the maximum utilization must occur at time 1. Therefore, we consider only $\hat{\rho}_{1,-,-,-}$ in the algorithm. The time complexity of Algorithm 1 is $O(|F||Q||N|)$.

Algorithm 1 QA (Queue Assignment)

Input: a network $< N, W, L >$, a flow set F
Output: $\forall f_a \in F, q_a$

1: $\forall f_a \in F, \rho_a = \frac{\gamma_a}{d_a}$;
2: sort flows in decreasing order of their utilization, where f_1 has the largest utilization;
3: **for** $a = 1$ to $|F|$ **do**
4: $\quad h = \arg\min_{\forall g \in [1,Q]} \{\max_{\forall n_{i,j} \in \Pi_a} \{\hat{\rho}_{1,i,j,g}\}\}$;
5: $\quad q_a$ is assigned the h-th queue;
6: $\quad \forall n_{i,j} \in \Pi_a, \hat{\rho}_{1,i,j,h} = \hat{\rho}_{1,i,j,h} + \rho_a$;
7: **return** all q_a;

Based on the result of Algorithm 1, we divide the original problem into multiple subproblems. There are $\sum_{\forall f_a \in F} \frac{H}{p_a}$ packets in a hyperperiod. We divide them into $\lceil \frac{\sum_{\forall f_a \in F} \frac{H}{p_a}}{\Delta} \rceil$ subsets. First, we sort all packets in the order of their earliest deadline first (EDF) priorities. EDF priorities mean that a packet with an earlier deadline has a higher priority. In the sorted packet set $\{s_1, s_2, ...\}$, the elements $s_{u \times \Delta + 1}, \ldots, s_{(u+1) \times \Delta}$ are assigned to the subset S_u. This assignment method schedules conflicting packets in the same subset so that the OMT solver can resolve conflicts more efficiently.

Since different packet subsets require different numbers of gate control entries, we cannot determine how many entries should be used in each subset. Therefore, the objective of each OMT subproblem is to minimize the maximum number of entries used, i.e.,

$$\text{obj.} \quad \min \bar{e}, \quad (14)$$

$$\text{subject to } \forall n_i \in W, \sum_{\forall e_{i,y} \in E_i} \lambda_{i,y} \leq \bar{e}, \quad (15)$$

$$\forall n_i \in W, \bigwedge_{\forall e_{i,y} \in E_i} (((t_{i,y} = 0) \wedge (\lambda_{i,y} = 0)) \vee ((t_{i,y} > 0) \wedge (\lambda_{i,y} = 1))), \quad (16)$$

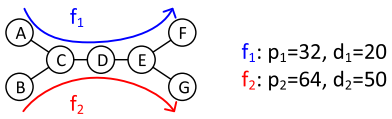
where $\bar{e}$ denotes the maximum number of entries used among all switches, and $\lambda_{i,y}$ denotes if the corresponding entries are used. In addition to the two constraints, Constraints 1) to 7) also need to be guaranteed.

Algorithm 2 shows the process of invoking the OMT solver. For each subset S_u, the OMT solver is invoked to find an optimal solution (lines 1-2). Then, all the variables in the solution are set to constants and cannot be changed in subsequent invocations (lines 3-7). This process is repeated until all subsets finish. Except when invoking the OMT solver, the other codes have $O(n)$ time complexity. n is approximately the number of packets.

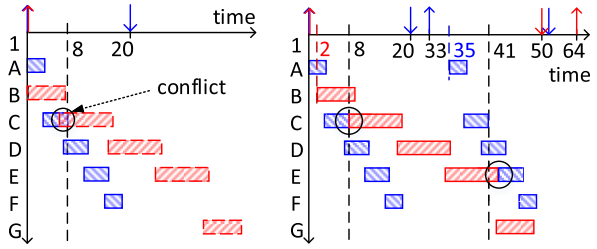
Algorithm 2 OMT-Based Scheduling Algorithm

Input: network $< N, W, L >$, flow set F
Output: all schedule tables

1: **for** each S_u **do**
2: **if** $OMT(S_u)$ found a solution **then**
3: **for** each packet in S_u **do**
4: the variables $\alpha^k_{a,b}$, $\beta^k_{a,b}$ $(\forall b \in [1, |\Pi_a| - 1))$ are assumed to be constants in the following calculation;
5: **for** each $e_{i,y}$ **do**
6: **if** $t_{i,y} > 0$ **then**
7: the entry $e_{i,y}$ can no longer be changed;
8: **else**
9: **return** FAIL;
10: **return** all E_i;



(a) A network and flows.



(b) Transmission conflicts. (c) Schedules without conflicts.

FIGURE 5. An example for algorithm 3.

V. HEURISTIC SCHEDULING ALGORITHMS

To handle massive data, scheduling algorithms should be simple and fast. In this section, we first present a fast scheduling algorithm (in Section V-A), and analyze the end-to-end delays (in Section V-B). Then, based on the analytic result, we modify the fast scheduling algorithm to improve its efficiency (in Section V-C).

A. NO-GATE-CLOSING SCHEDULING

There is a scheduling algorithm that can use one entry and one queue to transmit all packets. We call it the *no-gate-closing* (NGC) scheduling algorithm. In the NGC algorithm, all gates are always opened. After a switch receives a whole packet, it sends the packet to the next hop immediately. Source nodes control the injection times of all packets to avoid conflicts, and no queue is needed to block packets. Therefore, for each egress port, only one queue is used to buffer packets. This transmission rule is similar to that in no-wait packet scheduling [30]. However, we focus on a different problem model, and the no-wait scheduling algorithm is not suitable for massive data due to its execution time.

Algorithm 3 NGC (No-Gate-Closing Scheduling)

Input: network $< N, W, L >$, flow set F
Output: injection times η^k_a

1: $S = S' = \emptyset$;
2: **for** each $t \in [1, H]$ **do**
3: **for** each $f_a \in F$ **do**
4: **if** $t \mod p_a == 1$ **then**
5: $S = S + \{s^k_a\}$, where $k = \lfloor \frac{t}{p_a} \rfloor$;
6: **while** $S \neq \emptyset$ **do**
7: find a packet s^k_a with the highest EDF priority in S;
8: $S = S - \{s^k_a\}$;
9: **if** packet s^k_a misses its deadline **then**
10: **return** FAIL;
11: **if** $NoConflict(s^k_a, t)$ **then**
12: $\eta^k_a = t$;
13: **else**
14: $S' = S' + \{s^k_a\}$;
15: $S = S'$;
16: **if** $S \neq \emptyset$ **then**
17: **return** FAIL;
18: **else**
19: **return** all η^k_a;

The NGC algorithm (as shown in Algorithm 3) uses the classical EDF policy to schedule massive data since it has high real-time performance [18] in many real-time systems. The schedule is nonpreemptive. Once packets start to be transmitted, they cannot be blocked. In a hyperperiod, the network runs from time 1 to time H. At each time t, the newly released packets are added to the set S, which contains all released but unscheduled packets (lines 2–5). Then, the highest priority packet in set S is selected (lines 7–8) and checked by the function $NoConflict(\,)$ to determine whether it can be scheduled without conflicts at time t (line 11). As shown the example in FIGURE 5(a), the paths of flow f_1 and f_2 are $\{A, C, D, E, F\}$ and $\{B, C, D, E, G\}$, respectively, and the corresponding transmissions are shown in FIGURE 5(b) and (c). Up arrows denote release times, and down arrows are deadlines. There is no time gap between successive transmissions since transmissions are not blocked. The two flows generate their first packets at time 1. The packet of f_1 is first transmitted since it has a higher EDF priority than f_2. If the packet of f_2 also starts to be transmitted at time 1, the two packets will conflict in node C as shown in FIGURE 5(b). Hence, the packet of f_2 cannot start at time 1. The packets that have conflicts with the scheduled packets are removed to the set S' to be checked again at the next time (lines 13–15). After all packets in S are checked (line 6), the same process is repeated at the next time. Until the packet does not conflict with the scheduled packets, it can be injected into the network (line 12), such as the packet of f_2 is injected at time 2 as shown in FIGURE 5(c). η^k_a denotes the injection time of packet s^k_a. The second packet of f_1 is released at time 33. However,

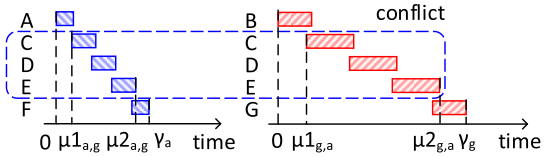


FIGURE 6. Conflict time.

due to the conflict with the first packet of f_2, its injection time is time 35. If a packet misses its deadline (lines 9–10) or some packets are not scheduled in a hyperperiod (lines 16–17), the algorithm cannot find a feasible solution. Otherwise, the injecting times of all packets are generated (lines 18-19). The time complexities of lines 2, 3, 6, 7 and 11 are $O(H)$, $O(|F|)$, $O(|S|)$, $O(|S|)$ and $O(|\Pi_a|)$, respectively. Therefore, the time complexity of Algorithm 3 is $O(|H||S|(|S|+|\Pi_a|))$, i.e., $O(n^3)$.

B. ANALYSIS

We analyze the worst-case end-to-end delay in Algorithm 3 so that we can find the key factors affecting the real-time performance. In the following, we focus on the packet s_a^k, the worst-case end-to-end delay of which is denoted as x. The delay x includes the end-to-end delay γ_a and the conflict delay $D(s_a^k, x)$ introduced by other packets in the time interval $[\sigma_a^k, \sigma_a^k + x]$, where σ_a^k is the release time of s_a^k and is equal to $p_a \times k + 1$. Hence,

$$x = D(s_a^k, x) + \gamma_a. \tag{17}$$

First, we calculate $D(s_a^k, x)$. We use four packet sets $HB(s_a^k)$, $HA(s_a^k)$, $LB(s_a^k)$, and $LA(s_a^k)$ to denote the four kinds of packets that will introduce conflicts to packet s_a^k. The letter H (or L) indicates that the packets in the corresponding sets have higher (or lower) EDF priorities than s_a^k, and the letter B (or A) indicates that the packets in the corresponding sets are released no later than (or after) the release time σ_a^k of packet s_a^k. For example, the set LB includes the packets that have lower priorities and are released no later than the time σ_a^k. A lower-priority packet can delay a higher-priority packet when the higher-priority packet has been blocked by other packets, with the lower-priority packet beginning the transmission process before the higher-priority packet. Since critical packets cannot be preempted, the higher-priority packet has to wait until the lower-priority packet releases the shared resources. For the packet s_a^k, we use the time interval $[\mu 1_{a,g}, \mu 2_{a,g}]$ to denote its conflict interval, where the earliest conflict time $\mu 1_{a,g}$ of flow f_a is the relative start time of the first transmission that uses the same node as that of f_g, and similarly, the last conflict time $\mu 2_{a,g}$ is the relative end time of the last transmission conflicting with f_g (as shown in FIGURE 6). For each packet s_g^m in the four sets, there must be

$$[\sigma_g^m + \mu 1_{g,a}, \sigma_g^m + d_g - (\gamma_g - \mu 2_{g,a})] \cap [\sigma_a^k + \mu 1_{a,g}, \sigma_a^k + x - (\gamma_a - \mu 2_{a,g})] \neq \emptyset. \tag{18}$$

If not, s_g^m does not affect the transmissions of s_a^k.

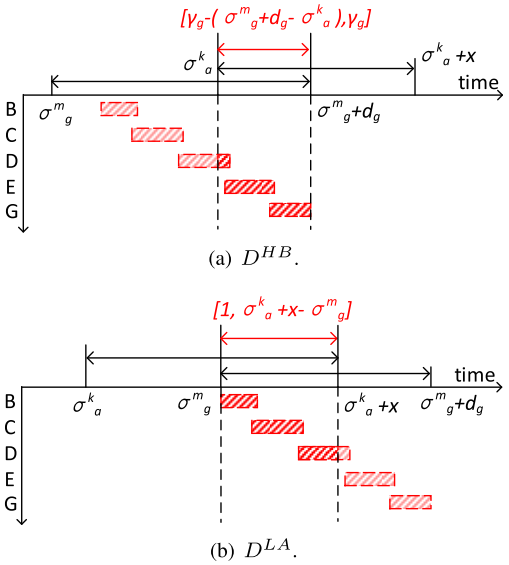


FIGURE 7. Some of the transmissions in a conflict interval.

Once we know the release times and deadlines of all the packets, we can then find the packets of the four sets. Thus,

$$\begin{aligned} D(s_a^k, x) &= \sum_{\forall s_g^m \in HB(s_a^k)} D^{HB}(s_a^k, s_g^m) + \sum_{\forall s_g^m \in HA(s_a^k)} D^{HA}(s_a^k, s_g^m) \\ &+ \sum_{\forall s_g^m \in LB(s_a^k)} D^{LB}(s_a^k, s_g^m) + \sum_{\forall s_g^m \in LA(s_a^k)} D^{LA}(s_a^k, s_g^m). \end{aligned} \tag{19}$$

The calculations of the four delays are as follows.

1) D^{HB}

First, we consider all the transmissions of s_g^m as being in the time interval $[\sigma_a^k, \sigma_a^k + x]$. Thus, the delay introduced by s_g^m is $\sum_{\forall t \in [1,\gamma_g]} C(s_a^k, s_g^m, t)$, where $C(s_a^k, s_g^m, t) = 1$ if s_a^k conflicts with s_g^m when it starts to be transmitted t time units after s_g^m; otherwise, $C(s_a^k, s_g^m, t) = 0$. In some cases, not all the transmissions of s_g^m are in the time interval $[\sigma_a^k, \sigma_a^k + x]$. As shown in FIGURE 7(a), if the front part of s_a^k overlaps with the rear part of s_g^m, the length of the overlapping time is $\sigma_g^m + d_g - \sigma_a^k$. Thus, when part of s_g^m is in the conflict interval, i.e., $(\sigma_g^m + d_g - \sigma_a^k) < \gamma_g$, only the last $(\sigma_g^m + d_g - \sigma_a^k)$ transmissions can introduce delay. Therefore, the delay is $\sum_{\forall t \in [\gamma_g - (\sigma_g^m + d_g - \sigma_a^k), \gamma_g]} C(s_a^k, s_g^m, t)$. We obtain that

$$D^{HB} = \sum_{\forall t \in [\delta^{HB}, \gamma_g]} C(s_a^k, s_g^m, t), \tag{20}$$

where $\delta^{HB} = \min\{1, \gamma_g - (\sigma_g^m + d_g - \sigma_a^k)\}$.

2) D^{HA}

All packets in $HA(s_a^k)$ have higher EDF priorities and are released after σ_a^k. Thus, compared to D^{HB}, there is no case

where only some of the transmissions affect s_a^k. Therefore,

$$D^{HA} = \sum_{\forall t \in [1, \gamma_g]} C(s_a^k, s_g^m, t). \tag{21}$$

3) D^{LB}

All packets in $LB(s_a^k)$ have lower EDF priorities and are released no later than σ_a^k. In the worst case the latter $\gamma_g - 1$ transmissions will conflict with s_a^k. This is the only difference between D^{HB} and D^{LB}. Therefore, the first transmission is excluded, i.e.,

$$D^{LB} = \sum_{\forall t \in [2, \gamma_g]} C(s_a^k, s_g^m, t). \tag{22}$$

4) D^{LA}

All packets in $LA(s_a^k)$ have lower EDF priorities and are released after s_a^k. In the worst case, the latter $\gamma_g - 1$ transmissions will affect s_a^k. As shown in FIGURE 7(b), the overlapping time is $\sigma_a^k + x - \sigma_g^m$. When $\gamma_g > \sigma_a^k + x - \sigma_g^m$, i.e., only some of the transmissions will affect s_a^k, and thus

$$D^{LA} = \sum_{\forall t \in [2, \delta^{LA}]} C(s_a^k, s_g^m, t), \tag{23}$$

where $\delta^{LA} = \min\{\gamma_g - (\sigma_g^m + d_g - \sigma_a^k - x), \gamma_g\}$.

Based on the above analysis and Equation (17), x can be solved by the following recurrent relation:

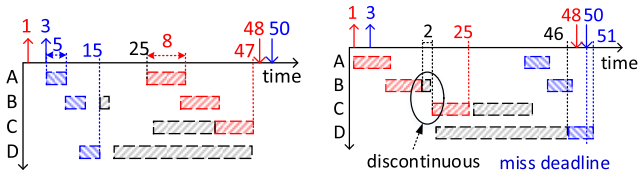
$$\begin{cases} x_0 = \gamma_a \\ x_{i+1} = D(s_a^k, x_i) + \gamma_a. \end{cases} \tag{24}$$

x_{i+1} is calculated repeatedly until x_{i+1} and x_i have the same value. Then, x is assigned as the final x_{i+1}.

From the analysis, we know that there are two key factors that delay flows. The first one is priority inversion, where lower-priority packets delay higher-priority packets. This weakens the efficiency of priorities. The second one is that a packet may suffer all conflicts in series. To cover the worst case, the analysis is pessimistic. However, in an actual schedule, the delay can be reduced by parallelism.

C. MOVE-FORWARD SCHEDULING

In this subsection, we propose a scheduling algorithm to reduce the effect of the two factors on end-to-end delays. First, we schedule packets in the strict order of their priorities. The classical EDF policy is designed for multitask systems, where the deadlines of unreleased tasks may be unknown during algorithm running. Thus, the classical nonpreemptive EDF policy selects tasks to be scheduled from the currently released tasks. However, in our model, all the deadlines are known. Thus, we can schedule packets in the strict order of their EDF priorities (as shown in Algorithm 4) so that lower-priority packets do not delay higher-priority packets. In Algorithm 4, all packets are sorted in decreasing order of their EDF priorities (line 1). Then, for each packet, the algorithm checks whether it can start to be scheduled without conflicting with the other scheduled packets at time t (lines 2–4), where t is from its release time to its deadline. The first no-conflict time is its injection time, and then, the packet can be transmitted without blocking (line 5). If there is no time t that satisfies $NoConflict(s_a^k, t)$, then the packet is moved to the set $\hat{S}$ to await subsequent processing (lines 6–7). The time complexity of Algorithm 5 is $O(|S||H||H|)$.



(a) Continuous scheduling. (b) Discontinuous scheduling.

FIGURE 8. An illustration of property 2.

Algorithm 4 SPS (Strict Priority Scheduling)

Input: network $< N, W, L >$, flow set F
Output: η_a^k and $\hat{S}$

1: add all packets into a set $\vec{S}$ in increasing order of their deadlines;
2: **for** $s_a^k =$ *the first packet* to *the last one* in $\vec{S}$ **do**
3: **for** $t = \sigma_a^k$ to $\sigma_a^k + d_a$ **do**
4: **if** $NoConflict(s_a^k, t)$ **then**
5: $\eta_a^k = t$;
6: **if** there is no t that makes $NoConflict(s_a^k, t)$ true **then**
7: $\hat{S} += \{s_a^k\}$;
8: **return** all η_a^k and $\hat{S}$;

Second, we propose a strategy to improve the parallelism of packets. Recall that in Algorithm 2 and 4, transmissions cannot be blocked. Thus, if two packets conflict with each other, all transmissions of one of them have to be delayed. This leads to a decrease in parallelism. Closing gates is a good way to improve parallelism because the transmissions of a packet can be transmitted discontinuously to use more resource fragmentation. Therefore, our proposed scheduling algorithm closes and opens gates based on the following three properties.

Property 1: If a packet can be transmitted continuously before its deadline, it can also be transmitted discontinuously before its deadline.

Discontinuous scheduling is a special case of continuous scheduling. The resources used by continuous scheduling can also be used by discontinuous scheduling. Therefore, in the worst case, discontinuous scheduling is the same as continuous scheduling.

Property 2: Packet $s_a, s_b \in \vec{S}$, and $a < b$. If $Resource(s_a) \cap Resource(s_b) \neq \emptyset$, and s_a is transmitted discontinuously, then s_a may introduce more delays to the lower-priority packet s_b.

$Resource(s_a)$ denotes the network resources required to transmit packet s_a. FIGURE 8 illustrates Property 2. The unit of the horizontal axis is time, and the letters of the vertical axis correspond to nodes. The red (blue) blocks represent when the

corresponding nodes are used to transmit s_a (s_b). The gray blocks represent the resources that have been occupied by other higher-priority packets. s_a has a higher priority than s_b. One-hop transmissions of s_a and s_b require 8 unit times and 5 unit times, respectively. The overlap time between two adjacent hops is 1 unit time. When s_a is scheduled continuously, it has to be injected into the network at time 25 because there is no continue resource at the front. Thus, s_b can use the idle resources between time 3 and 15 (as shown in FIGURE 8(a)). In this case, the two packets can be transmitted before their deadlines. However, when s_a is discontinuous, it will occupy the resources between time 1 and 25 (as shown in FIGURE 8(b)). The transmissions of s_a are broken between nodes B and C. Since node D is occupied by other packets, the last hop of s_b has to be transmitted between time 46 and 51, i.e., s_b misses its deadline time 50. Therefore, discontinuous scheduling may make packets that are schedulable in other algorithms unschedulable.

Property 3: Packet $s_a, s_b \in \vec{S}$, and $a < b$. For packet s_{a+1},..., s_{b-1}, if there exists a subset $\{s'_1, \ldots, s'_j\}$, and

$$\begin{aligned} \forall g \in [1, j), \quad & (Resource(s'_g) \cap Resource(s'_{g+1}) \neq \emptyset) \\ & \wedge (Resource(s_a) \cap Resource(s'_1) \neq \emptyset) \\ & \wedge (Resource(s'_j) \cap Resource(s_b) \neq \emptyset), \end{aligned} \tag{25}$$

then packet s_a may delay packet s_b.

Based on Property 2, we know that the discontinuous s_a can affect the transmissions of s'_1, and then s'_1 affects s'_2. Repeat this process, and then s'_j affects s_b. Thus, the discontinuous s_a will affect s_b.

Our proposed algorithm is shown in Algorithm 5. We first invoke Algorithm 4 *SPS*() to generate schedules *sch*[ ][ ][ ] (lines 1–2), and create a set $\hat{S}$ to mark all unscheduled packets and a set $\vec{S}$ to mark the packets that affect the unscheduled packets based on Properties 2 and 3 (line 5). If *SPS*() can schedule all packets, the following algorithm does not need to run (lines 3–4). If not, we reschedule the packets in $\hat{S}$ and $\vec{S}$ under the discontinuous scheduling rule (lines 6–23). There are two reasons why our rescheduling algorithm can improve schedulability: first, based on Property 1, rescheduling a scheduled packet does not make it unschedulable; second, discontinuous transmissions will use resource fragmentation at the front and leave more available resources for unscheduled packets.

NoConflict$(s_a^k, h, t) = 1$ denotes that the h-th hop of s_a^k can be scheduled without conflicting with scheduled packets at time t (line 10). For a packet in $\hat{S}$ and $\vec{S}$, we schedule its hops separately before its deadline (lines 8–9 and 16–18). The new schedules are written in the schedule array *sch* (line 11). If two successive hops cannot be scheduled continuously, the entries need to be set to control the corresponding gate (lines 12 and 15). However, if there is no unused entry, the flow set cannot be scheduled (lines 13–14). Repeat this process to schedule all packets in $\hat{S}$ and $\vec{S}$ until a packet misses its deadline (lines 21–22) or all schedules are generated and returned (line 23).

Algorithm 5 MF (Move-Forward Scheduling)

Input: network $< N, W, L >$, flow set F
Output: schedules *sch*[ ][ ][ ]

1: invoke *SPS*() to get injection times η_a^k and an unschedulable packet set $\hat{S}$;
2: for all scheduled packets, create an array $sch[a][k][h]$ to record the start time of the h-th hop of each packet s_a^k;
3: **if** $\hat{S} == \emptyset$ **then**
4: **return** *sch*[ ][ ][ ];
5: $\forall s_b \in \hat{S}$, all scheduled packets that satisfy Properties 2 and 3 are added into the set $\tilde{S}$;
6: **for** $s_a^k =$ *the first packet* to *the last one* in $\vec{S}$ **do**
7: **if** $s_a^k \in \tilde{S} \cap \hat{S}$ **then**
8: $t = \sigma_a^k$; $h = 1$;
9: **while** $t \leq \sigma_a^k + d_a$ **do**
10: **if** *NoConflict*(s_a^k, h, t) **then**
11: $sch[a][k][h] = t$;
12: **if** the h-th hop and $(h-1)$-th hop are not continuous **then**
13: **if** the source node of the h-th hop does not have unused entries **then**
14: **return** FAIL;
15: set entries to control the corresponding gate in the source node of the h-th hop;
16: t is set to the earliest start time of the next hop;
17: **if** $(++h) == |\Pi_a|$ **then**
18: break;
19: **else**
20: $t++$;
21: **if** $s_a^k \in \hat{S}$ and $h \neq |\Pi_a|$ **then**
22: **return** FAIL;
23: **return** *sch*[ ][ ][ ];

This algorithm first invokes *SPS*() to schedule packets, and then moves the schedules forward. In the process of moving, if transmissions are not continuous, schedule entries are spent to block and unblock packets. The schedules generated by *SPS*() occupy many network resources. Thus, even though we allow all packets to move forward, not many packets can move. Therefore, compared to other scheduling algorithms, this algorithm still spends fewer schedule entries. The number of iterations of the **for** loop in line 4 and **while** loop in line 7 is $O(|S|)$ and $O(|H|)$, respectively. The time complexity of *NoConflict*() is $O(|H|)$. Therefore, the time complexity of Algorithm 5 is $O(|S||H||H|)$.

VI. EVALUATION

In this section, we evaluate our proposed algorithms based on massive test cases. Three metrics are used for performance evaluation: (1) *schedulable ratio* is the percentage of test cases for which an algorithm is able to find a feasible schedule; (2) *the number of schedule entries* means that a switch needs at least these schedule entries to schedule its packets;

TABLE 1. Parameters.

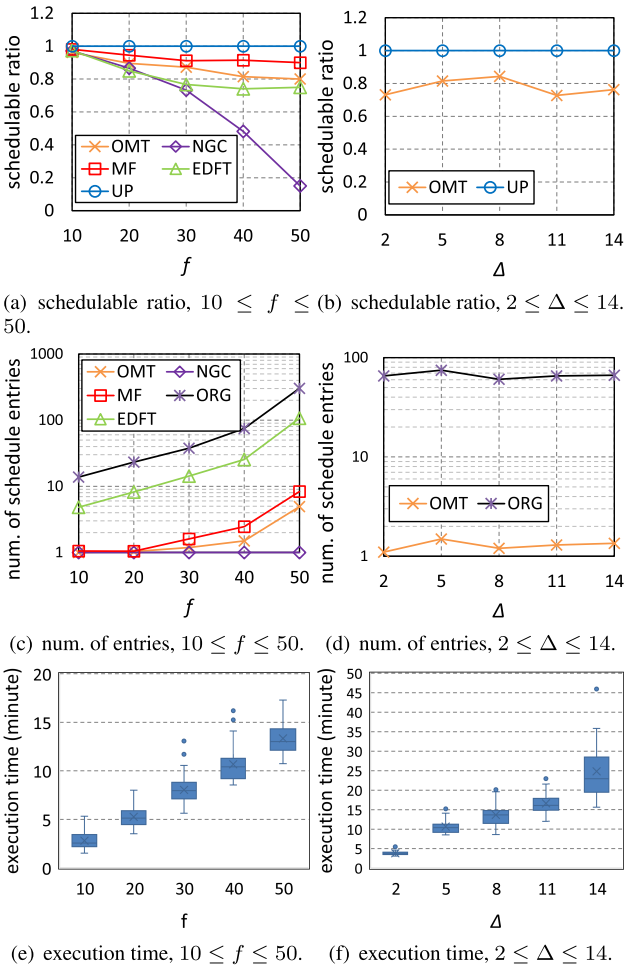
n	Number of switches
Q	Number of queues
f	Number of flows
p	Period range
s	Packet size range
Δ	Number of packets that $OMT()$ solves at one time

and (3) *execution time* is the time required to generate a feasible schedule.

We compare our OMT (Algorithm 2), NGC (Algorithm 3), and MF (Algorithm 5) algorithms with three fundamental methods ORG, EDFT and UP. ORG is the method used in the current TSN switches, in which gates must be closed immediately after they finish a transmission. EDFT schedules each hop based on their EDF priorities. UP is a conservative upper bound of optimal results. Since, to verify the performance of our algorithms, our evaluation results should be compared with optimal results. However, our problem is NP-complete, and SMT/OMT solvers cannot solve large-scale test cases. There is no method to obtain accurate optimal results. Thus, we have to compare our proposed algorithms with the conservative upper bound UP. In UP, if test cases satisfy the following necessary condition, they are considered schedulable. The necessary condition for schedulability is that for each port in a switch, its unidirectional utilization cannot be greater than 100%. Since UP is better than optimal results, if our results are close to UP, they must be close to optimal results. Note that UP is a baseline for comparisons and does not generate schedule tables. OMT is solved by the Microsoft solver Z3 [36], [37]. All algorithms are written in C and run on a Windows machine with a 3.4 GHz CPU and 16GB of memory.

Our test cases are generated based on the given parameters, which are summarized in TABLE 1. For each test case, n switches with Q queues are randomly deployed in a square area. Each switch connects a gateway and three adjacent switches. f flows randomly select their source nodes and destination nodes. Periods and packet sizes are random numbers in the period range p and the packet size range s, respectively. The deadline of a flow is also a random number between its transmission delay and its period. We use a mass of random test cases to verify our algorithms so that their universality can be demonstrated. Before running algorithms, to avoid unnecessary computations, we use UP to pre-filter the unschedulable test cases. If the test cases cannot satisfy UP, they are discarded. The pre-filtering operation is based on the necessary condition and does not affect our evaluation results. Δ is used to control the execution time of OMT.

In the following evaluation, we first compare all the algorithms, and the comparison is shown in FIGURE 9 of Section VI-A. To verify the universality of the algorithms, for each parameter setting, 10000 test cases are generated and solved. Due to the long execution time of Z3, the scale of test cases is limited. Therefore, in Section VI-B, we compare



(a) schedulable ratio, $10 \leq f \leq 50$. (b) schedulable ratio, $2 \leq \Delta \leq 14$.

(c) num. of entries, $10 \leq f \leq 50$. (d) num. of entries, $2 \leq \Delta \leq 14$.

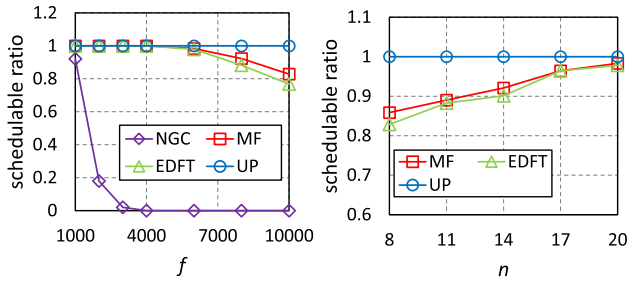
(e) execution time, $10 \leq f \leq 50$. (f) execution time, $2 \leq \Delta \leq 14$.

FIGURE 9. Comparison with Z3.

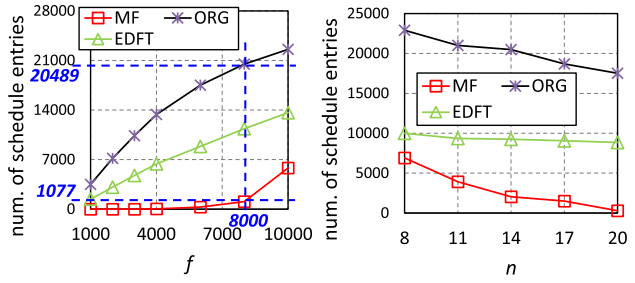
the other algorithms except OMT (Z3) using large-scale test cases. In Section VI-B, we set the parameters to different values so that the algorithms can be tested fully. FIGURE 10 shows the comparisons under varying f and n. FIGURE 11, 12 and 13 show the comparisons under varying Q, p and s, respectively. Thus, our proposed Algorithm 2, 3, 4 and 5 are evaluated completely. Finally, we evaluate Algorithm 1 that assigns queues in OMT, and show the evaluation results in FIGURE 14.

A. EVALUATIONS WITH Z3

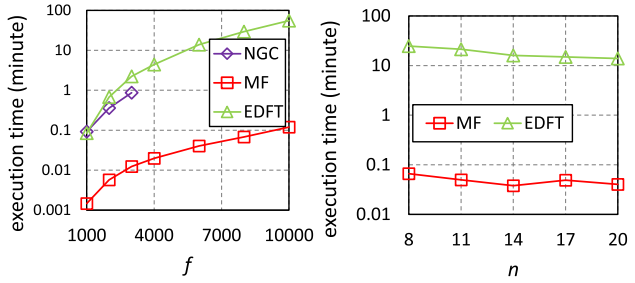
To make test cases solvable by Z3, we set $n = 6$, $Q = 2$, $p = [32, 64]$, $s = [100, 400]$, $f \in [10, 50]$ and $\Delta \in [2, 14]$. The average results of 10000 test cases are shown in the subsequent figures. FIGURE 9 shows the comparison of schedulable ratios, the number of schedule entries, and the execution times under varying f and Δ. Since Δ does not affect the other algorithms except OMT, our proposed algorithms are not shown in FIGURE 9(b) and (d). In FIGURE 9(a) and (b), the schedulable ratios are normalized, with UP as the baseline. As f increases, NGC decrease rapidly, and the others decrease slowly. For NGC, the greater the number of flows, the greater amount of resources that are fragmented, and the



(a) schedulable ratio under varying f. (b) schedulable ratio under varying n.

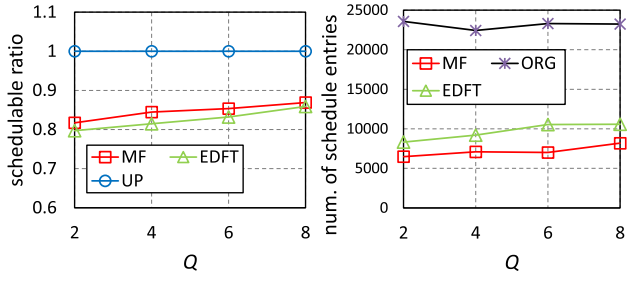


(c) num. of entries under varying f. (d) num. of entries under varying n.



(e) execution time under varying f. (f) execution time under varying n.

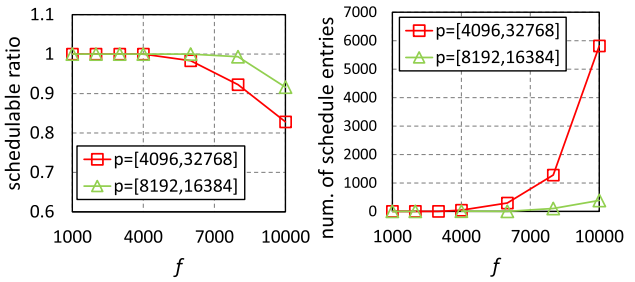
FIGURE 10. Comparison without Z3.



(a) schedulable ratio. (b) number of schedule entries.

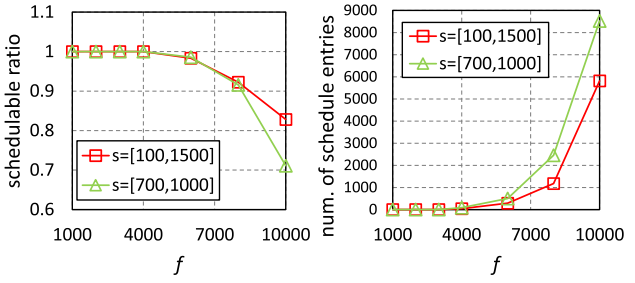
FIGURE 11. Comparison under $n = 8$ and varying Q.

more conflicts that are introduced to high-priority packets. Thus, NGC has the worst schedulable ratio. OMT calls the solver Z3 to minimize the number of schedule entries. In each call, an attempt is made to satisfy the schedulability, while OMT cannot optimize the schedulability between multiple calls. Thus, our proposed MF algorithm has higher schedulable ratios than those of OMT. EDFT provides a flexible strategy to schedule flows. However, it still cannot avoid the conflicts introduced by lower-priority packets. Thus, it is better than NGC but worse than the other algorithms. In FIGURE 9(b), as Δ increases, the schedulable ratio of OMT



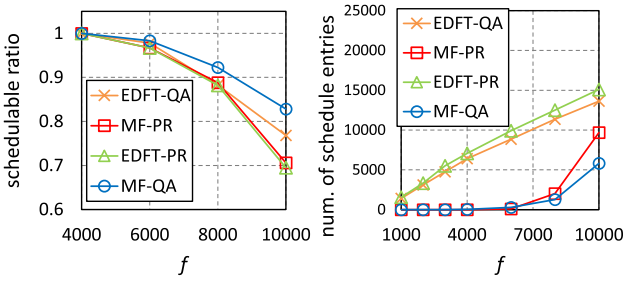
(a) schedulable ratio. (b) number of schedule entries.

FIGURE 12. Comparison under varying p.



(a) schedulable ratio. (b) number of schedule entries.

FIGURE 13. Comparison under varying s.



(a) schedulable ratio. (b) number of schedule entries.

FIGURE 14. Comparison of queue assignment algorithms.

almost does not change. This is because the setting of test cases does not change.

FIGURE 9(c) and (d) show a comparison of the number of schedule entries. As f increases, the other algorithms except NGC increase. NGC is allowed to use only one schedule entry. If one entry is not enough to schedule all flows, NGC cannot generate schedules. For the same network settings, the greater the number of flows, the greater the number of entries that are needed to isolate packets. Hence, the other algorithms exhibit an increase. ORG closes the corresponding gates after finishing transmissions, while only when transmissions need to be blocked do the other algorithms close gates. Therefore, ORG consumes the most entries. In EDFT, if two transmissions are continuous, no schedule entry is needed. Thus, EDFT requires fewer schedule entries than ORG. The objective of OMT is to minimize the number of required schedule entries. Hence, it requires the least number of schedule entries. Although our proposed MF algorithm requires more entries than does OMT, MF has the higher schedulable ratio and a shorter execution time. For all the

test cases in this subsection, MF can finish within 1 ms, while the execution time of OMT is approximately a few minutes. As the number of flows increases, the execution time of OMT increases linearly (as shown in FIGURE 9(e)). Based on this trend, OMT will require less than two days to solve a test case with 10000 flows and 40000 packets. However, for the original SMT specification shown in Section IV-A, it is impossible to obtain the result of any large-scale test case. We tried to solve a test case with 100 flows based on the SMT specification but did not obtain the result within two days. Therefore, when a flow set needs to be scheduled, MF is used first because it is fast and efficient. Then, if MF requires more schedule entries than those supported by network devices, OMT can be used to find a more optimized solution.

B. EVALUATIONS WITHOUT Z3

To fully evaluate our proposed algorithms, we vary every parameter to compare the results. The basic settings are $n = 20$, $Q = 4$, $f = 6000$, $p = [4096, 32768]$, and $s = [100, 1500]$. In the following evaluation, if these parameters are not specified, the basic settings are used. In FIGURE 10, the parameters f and n are changed. FIGURE 10(a) and (b) show schedulable ratios that are normalized with UP as the baseline. NGC is still the worst one. When $f > 3000$, NGC cannot schedule any test case. Thus, in the following comparisons, only FIGURE 10(e) shows three points corresponding to NGC. As f increases, the schedulable ratios decrease. This is because the greater the number of flows, the more difficult the scheduling becomes. MF can avoid priority inversion, while in EDFT, priority inversions still occurs between transmissions. Thus, MF is better than EDFT. As shown in FIGURE 10(b), the schedulable ratios increase as n increases because the greater the number of switches, the fewer the number of conflicts that occur.

FIGURE 10(c) and (d) show the number of required schedule entries. Since the flow set that is difficult to schedule requires more schedule entries, the trend of the lines in the two figures is opposite to that in FIGURE 10(a) and (b). There are approximately 40000 packets in a test case with 10000 flows. ORG has to spend many schedule entries to provide complete isolation for these packets. Our proposed MF algorithm requires the fewest schedule entries because the strict priority scheduling requires only one entry in each switch, and the selection of moving packets limits the number of operations on gates. Compared to ORG, in the best case, MF requires only one-twentieth the schedule entries to schedule the same flow set.

The related work in [29] considers the number of schedule entries in its problem model. However, the work uses only 64 entries to schedule small networks. Due to the unacceptable execution time of SMT solvers, the work cannot use more entries. Although, from the evaluation shown in [29], we can deduce that maybe a switch with 1024 entries can schedule more than 1024 flows, we do not know the exact number of flows and cannot compare it with our results. The datasheet [14] of the NXP switch chip mentions that the chip with 1024 entries can support 1024 deterministic flows. The datasheet does not explain why the number of deterministic flows is 1024. However, this is the only exact number we found. Therefore, we compare it with our results. In our algorithms, as shown in FIGURE 10(c), the switch with 1024 entries can deterministically schedule approximately 8000 flows, which include more than 30000 packets. Thus, our proposed algorithm offers approximately 8 times more schedulability than does the suggestion in the datasheet [14].

FIGURE 10(e) and (f) show the execution time. MF is based on fixed priorities and does not need to find the highest priority packet for each schedule. However, EDFT not only needs to find the highest-priority one at every scheduling time but also finishes only one transmission after each discovery. This is the reason why MF can finish within 1 minute, while EDFT spends up to 1 hour. As f increases, the execution time spent to search the solution spaces increases. As n increases, the execution time slightly fluctuates. The reason is that the scheduling algorithms traverse flows rather than switches.

FIGURE 11 shows the comparison a varying number of queues. To make the difference obvious, we set $n = 8$. As Q increases, the schedulable ratios slightly increase. This is because having more queues helps to eliminate some of the queue conflicts, though the link conflicts still exist. Comparing FIGURE 10(b) and FIGURE 11(a), we find that link conflicts have more impact on schedulable ratios than do queue conflicts. The greater the number of queues, the more entries are needed to control gates. Thus, the number of schedule entries also increases as Q increases.

FIGURE 12 shows the comparison under different p. All the results are solved by our MF algorithm. When the range of periods decreases, the high real-time packets are eliminated. Thus, the test cases are easy to schedule. FIGURE 13 shows the comparison under different packet sizes. When the range of packet sizes becomes narrow, there are no small packets to make use of resource fragmentation. Therefore, the schedulable ratio reduces, and the number of schedule entries increases.

FIGURE 14 shows the comparison between our proposed queue assignment algorithm (QA) and a method (PR) currently used. In current applications, high-priority flows are assigned to high-priority queues. In the compared PR method, flows are evenly assigned to queues based on their priorities. This method does not consider the workload assigned to a switch. If a switch has a greater workload, it needs more entries to isolate flows. Therefore, the methods with PR have lower schedulable ratios and spend more schedule entries than those with our proposed QA method.

VII. DISCUSSION

For scheduling problems of TSNs, there is no benchmark. Thus, we have to vary all the parameters related to networks and flows to generate extensive test cases such that our evaluations can cover as many situations as possible. From the evaluation results, we can conclude that our proposed algorithms

OMT and MF are better than the others. This is because our algorithms use the following two strategies to reconcile the contradiction between the limited number of schedule entries and massive data. The first one is to control the injection times of packets to reduce conflicts. The second one is to use one schedule entry to isolate as many packets as possible. These are also the reasons why our proposed algorithms can tackle the two challenges mentioned in Section I.

Our algorithms are based on software and can be applied to all of the networks following the standard IEEE 802.1Qbv. For example, our algorithms can be used in a 5G system to solve practical issues, since TSNs are defined as the fronthaul networks of 5G systems in the IEEE standard 802.1CM, and transmitting massive data is one of the three use cases supported by 5G. Even if in the future a switch chip may include a large schedule table, our algorithms can still break through the limitation of the chips and make a switch support more communications.

In industrial applications, almost all communications are fixed and known, while bursty communications still exist, such as emergency alarms. Our algorithms are designed to handle known communications. When a bursty packet needs to be transmitted, schedule tables have to be reconfigured. This reconfiguration process will introduce a long delay that may reduce the real-time performance of industrial networks. Therefore, in the future, we will propose distributed scheduling algorithms to handle bursty communications.

VIII. CONCLUSION

TSNs, as the backbone network of industrial internet of things, have to schedule massive real-time data packets. However, the off-the-shelf TSN switches only contain limited schedule entries. This causes that they cannot support massive real-time packets. Therefore, in this paper, we propose the algorithms that can schedule massive real-time data with a limited number of schedule entries. First, to find the optimal results of the NP-complete problem, we formulate it as an SMT specification. The SMT/OMT solver Z3 can find optimal solutions for SMT specifications. However, due to the complexity of the problem, Z3 cannot even find any feasible results for a small test case in two days. Hence, we divide the SMT specification into multiple OMT specifications and still use Z3 to solve them. The evaluation results indicate that the execution time of OMT increases linearly as the number of flow increases. In two days, using the OMT-based algorithm, we can obtain feasible solutions for test cases with 10000 flows. Second, to obtain solutions quickly, we propose heuristic algorithms. Based on an initial algorithm NGC, we find two factors that make flows difficult to schedule. Then, we propose the MF algorithm to reduce the impact of the two factors on the schedulability. Compared with existing heuristic algorithms, our proposed MF algorithm requires only at least one-twentieth the number of schedule tables to schedule the same number of packets. Therefore, if users want to obtain feasible solutions quickly, MF is a good choice; if the usage of entries is the most important, the OMT-based algorithm can be used.

APPENDIX
SYMBOL

The following symbols are used in this paper.

N	Node set
W	Switch set
L	Link set
n_i	The i-th node
$n_{i,j}$	The j-th port of n_i
$l_{i,g}$	The link between n_i and n_g
T	Number of entries
F	Flow set
f_a	The a-th flow
p_a	Period of f_a
d_a	Relative deadline
q_a	Queue ID of f_a
γ_a	End-to-end delay of f_a
γ_a^+	Transmission delay of f_a
Π_a	Path of f_a
$\pi_{a,b}$	The b-th node in Π_a
H	Hyperperiod
E_i	Schedule table in n_i
$e_{i,y}$	The y-th entry in E_i
$t_{i,y}$	The time at which $e_{i,y}$ starts to run
$v_{i,y,j}$	If $e_{i,y}$ is available to Port j
$c_{i,y,r}$	If the gate of the r-th queue is opened
V	Number of ports in a switch
Q	Number of queues in an egress port
s_a^k	The b-th packet of f_a
$\tau_{a,b}^k$	The b-th hop of s_a^k
$\alpha_{a,b}^k$	Starting time of $\tau_{a,b}^k$
$\beta_{a,b}^k$	Ending time of $\tau_{a,b}^k$
Δ	Number of packets in a OMT subproblem
ρ_a	Utilization of n_a
$\hat{\rho}_{t,i,j,g}$	Utilization of the g-th queue in $n_{i,j}$ at time t
$S, S', \tilde{S}$	Temporary packet sets
S_u	Packet set in OMT subproblems
$\hat{S}$	Unschedulable packet set
$\vec{S}$	Sorted packet set
s_a	The a-th packet in a packet set
$\bar{e}$	Maximum number of available entries
$\lambda_{i,y}$	Whether $e_{i,y}$ is available
η_a^k	Injection time of s_a^k
σ_a^k	Release time of s_a^k
$[\mu 1_{a,g}, \mu 2_{a,g}]$	Conflict interval
$sch[\,][\,][\,]$	Schedules
n	Number of switches
f	Number of flows
p	Period range
s	Packet size range

REFERENCES

[1] D. Serpanos and M. Wolf, "Industrial Internet of Things," in *Internet-of-Things (IoT) Systems*. Cham, Switzerland: Springer, 2018, pp. 37–54.

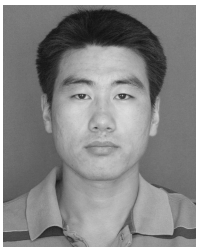
[2] B. Chen, J. Wan, A. Celesti, D. Li, H. Abbas, and Q. Zhang, "Edge computing in IoT-based manufacturing," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 103–109, Sep. 2018.

[3] Z. Wu, Z. Meng, and J. Gray, "IoT-based techniques for online M2M-interactive itemized data registration and offline information traceability in a digital manufacturing system," *IEEE Trans. Ind. Inf.*, vol. 13, no. 5, pp. 2397–2405, Oct. 2017.

[4] W. Liang, M. Zheng, J. Zhang, H. Shi, H. Yu, Y. Yang, S. Liu, W. Yang, and X. Zhao, "WIA-FA and its applications to digital factory: A wireless network solution for factory automation," *Proc. IEEE*, vol. 107, no. 6, pp. 1053–1073, Jun. 2019.

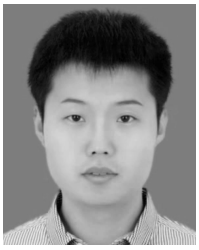
[5] X. Jin, N. Guan, C. Xia, J. Wang, and P. Zeng, "Packet aggregation real-time scheduling for large-scale WIA-PA industrial wireless sensor networks," *Trans. Embed. Comput. Syst.*, vol. 17, no. 5, pp. 1–19, Sep. 2018.
[6] Y.-H. Wei, Q. Leng, S. Han, A. K. Mok, W. Zhang, and M. Tomizuka, "RT-WiFi: Real-time high-speed communication protocol for wireless cyber-physical control applications," in *Proc. IEEE 34th Real-Time Syst. Symp.*, Dec. 2013, pp. 140–149.
[7] Z. Pang, M. Luvisotto, and D. Dzung, "Wireless high-performance communications: The challenges and opportunities of a new target," *IEEE Ind. Electron. Mag.*, vol. 11, no. 3, pp. 20–25, Sep. 2017.
[8] X. Jin, F. Kong, L. Kong, W. Liu, and P. Zeng, "Reliability and temporality optimization for multiple coexisting WirelessHART networks in industrial environments," *IEEE Ind. Electron. Mag.*, vol. 64, no. 8, pp. 6591–6602, Aug. 2017.
[9] L. Zhao, P. Pop, Q. Li, J. Chen, and H. Xiong, "Timing analysis of rate-constrained traffic in TTEthernet using network calculus," *Real-Time Syst.*, vol. 53, no. 2, pp. 254–287, Mar. 2017.
[10] 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 Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 88–145, 1st Quart., 2019.
[11] H. Zhang, N. Liu, X. Chu, K. Long, A.-H. Aghvami, and V. C. M. Leung, "Network slicing based 5G and future mobile networks: Mobility, resource management, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 138–145, Aug. 2017.
[12] Cisco. (2018). *About Time-Sensitive Networking*. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/switches/lan/cisco_ie4000/tsn/b_tsn_ios_support/b_tsn_ios_support_chapter_01.pdf
[13] Innovasic Inc. (2016). *TSN Evaluation Kit Quick Start Guide*. [Online]. Available: https://www.analog.com/media/en/technical-documentation/user-guides/tsn-evaluation-kit-quickstart-guide.pdf
[14] NXP Semiconductors. (2016). *SJA1105 5-Port Automotive Ethernet Switch*. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/SJA1105.pdf
[15] C. Barrett and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Model Checking*. Cham, Switzerland: Springer, 2018, pp. 305–343.
[16] A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani, and C. Stenico, "Satisfiability modulo the theory of costs: Foundations and applications," in *Proc. Int. Conf. Tools Algorithms Construct. Anal. Syst.*, 2010, pp. 99–113.
[17] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 1–44, Oct. 2011.
[18] J. Liu, *Real Time Systems*. Upper Saddle River, NJ, USA: Prentice-Hall, 2000.
[19] A. Sgora, D. J. Vergados, and D. D. Vergados, "A survey of TDMA scheduling schemes in wireless multihop networks," *ACM Comput. Surv.*, vol. 47, no. 3, pp. 1–39, Apr. 2015.
[20] P. Park, S. Coleri Ergen, C. Fischione, C. Lu, and K. H. Johansson, "Wireless network design for control systems: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 978–1013, 2nd Quart., 2018.
[21] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks," in *Proc. 24th Int. Conf. Real-Time Netw. Syst. (RTNS)*. New York, NY, USA: ACM, 2016, pp. 183–192.
[22] 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, Dec. 2018.
[23] J. Falk, F. Durr, and K. Rothermel, "Exploring practical limitations of joint routing and scheduling for TSN with ILP," in *Proc. IEEE 24th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2018, pp. 136–146.
[24] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner, "Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks," *IET Cyber-Phys. Syst., Theory Appl.*, vol. 1, no. 1, pp. 86–94, Dec. 2016.
[25] V. Gavrilut, L. Zhao, M. L. Raagaard, and P. Pop, "AVB-aware routing and scheduling of time-triggered traffic for TSN," *IEEE Access*, vol. 6, pp. 75229–75243, 2018.
[26] M. Pahlevan and R. Obermaisser, "Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks," in *Proc. IEEE 23rd Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2018, pp. 337–344.
[27] M. L. Raagaard, P. Pop, M. Gutierrez, and W. Steiner, "Runtime reconfiguration of time-sensitive networking (TSN) schedules for Fog Computing," in *Proc. IEEE Fog World Congr. (FWC)*, Oct. 2017, pp. 1–6.
[28] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner, "Enabling fog computing for industrial automation through time-sensitive networking (TSN)," *IEEE Commun. Stand. Mag.*, vol. 2, no. 2, pp. 55–61, Jun. 2018.
[29] R. S. Oliver, S. S. Craciunas, and W. Steiner, "IEEE 802.1Qbv gate control list synthesis using array theory encoding," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2018, pp. 13–24.
[30] 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. (RTNS)*. New York, NY, USA: ACM, 2016, pp. 203–212.
[31] F. Xiao and M. Aritsugi, "An adaptive parallel processing strategy for complex event processing systems over data streams in wireless sensor networks," *Sensors*, vol. 18, no. 11, p. 3732, Nov. 2018.
[32] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, "ILP-based joint routing and scheduling for time-triggered networks," in *Proc. 25th Int. Conf. Real-Time Netw. Syst. (RTNS)*, 2017, pp. 8–17.
[33] K. Gopalan, T.-C. Chiueh, and Y.-J. Lin, "Load balancing routing with bandwidth-delay guarantees," *IEEE Commun. Mag.*, vol. 42, no. 6, pp. 108–113, Jun. 2004.
[34] *Commercial Building Telecommunications Cabling Standard*, Standard ANSI/TIA/EIA-568-B.2-2001, Telecommun. Ind. Assoc., 2001.
[35] NXP Semiconductors. (2018). *Open Industrial Linux User Guide*. [Online]. Available: https://www.openil.org/news/files/OpenIL_User_Guide_Rev1.2.pdf
[36] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. 14th Int. Conf. Tools Algorithms for Construct. Anal. Syst.*, 2008, pp. 337–340.
[37] N. Bjørner and P. A. Dung, "νZ—Maximal satisfaction with Z3," in *Proc. 6th Int. Symp. Symbolic Comput. Softw. Sci. (SCSS)*, 2014, pp. 1–9.

XI JIN received the Ph.D. degree in computer science from Northeastern University, China, in 2013. She is currently an Associate Professor with the Shenyang Institute of Automation, Chinese Academy of Sciences. She is also a Committee Member of the China Computer Federation Technical Committee on Embedded Systems (CCF TCEBS). Her research interests include industrial networks, real-time systems, and the Internet of Things.



CHANGQING XIA received the Ph.D. degree from Northeastern University, China, in 2015. He is currently an Assistant Professor with the Shenyang Institute of Automation, Chinese Academy of Sciences. His research interests include wireless sensor networks and real-time systems, especially the real-time scheduling algorithms and smart energy systems.

NAN GUAN received the B.E. and M.S. degrees from Northeastern University, China, in 2003 and 2006, respectively, and the Ph.D. degree from Uppsala University, Sweden, in 2013. He is currently an Assistant Professor with the Department of Computing, The Hong Kong Polytechnic University. His research interests include real-time embedded systems and cyber-physical systems. He received the EDAA Outstanding Dissertation Award, in 2014, and the Best Paper Award of RTSS 2009 and DATE 2013.



CHI XU received the Ph.D. degree from the University of Chinese Academy of Sciences, China, in 2017. He has been with the Shenyang Institute of Automation, Chinese Academy of Sciences, China, since 2013, where he currently is doing his postdoctoral research and serving as an Assistant Professor. His research interests include cognitive radio networks, industrial wireless networks, tactile internet, and 5G URLLC. He is a Voting Member of the IEEE 1918.1 Working Group for Tactile Internet as well as a member of the IEEE 1918.1.1 Task Group for Haptic Codec. He is also a member of the IEEE 1932.1 Working Group for Licensed/Unlicensed Spectrum Interoperability in Wireless Mobile Networks. He serves as a 3GPP standardization delegate in the Technical Specification Group (TSG) for Radio Access Network (RAN).

DONG LI received the Ph.D. degree in mechatronic engineering from the Shenyang Institute of Automation, Chinese Academy of Sciences. He is currently involved in research on industrial networks and software-defined networking for intelligent manufacturing.

YUE YIN received the master's degree in technical economy and management from the Business School of Liaoning University, in 2013. He is currently pursuing the Ph.D. degree in technical economics with the School of Economics, Liaoning University. His research interests include network management and optimization algorithms.

PENG ZENG received the Ph.D. degree from the Shenyang Institute of Automation, Chinese Academy of Sciences. He is currently a Professor with the Shenyang Institute of Automation, Chinese Academy of Sciences. His research interests include industrial communication and wireless sensor networks.