IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received March 10, 2020, accepted March 29, 2020, date of publication April 1, 2020, date of current version April 17, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2984749

A Mixed-Critical Consistent Update Algorithm in Software Defined Time-Triggered Ethernet Using Time Window

JUN LU, HUAGANG XIONG, FENG HE, ZHONG ZHENG, AND HAORUO LI
School of Electronic and Information Engineering, Beihang University, Beijing 100191, China

Corresponding author: Feng He (robinleo@buaa.edu.cn)

ABSTRACT Software Defined Networking (SDN) presents a tremendous opportunity for developing abstract management of network updates. However, network updates introduce challenges in terms of consistency. Many consistent update algorithms are proposed for Ethernet, but there is seldom update algorithm for distributed time-triggered system to satisfy requirements of strict real-time and mixed-critical. The contribution of this paper is to present a distributed update mechanism and a mixed-critical update algorithm for the time-triggered (TT) and rate-constrained (RC) traffics to reduce the memory space complexity, computational time complexity, update time, and keep per-packet consistent strictly. More specifically, consistent update is defined and some basic update algorithms are described. And then a packet-based network model is built to describe network resources. Subsequently, the update mechanism is changed for the TT and RC traffics. Meanwhile, a mixed-critical update algorithm, which contains a time window-based update and a rate-constrained update for TT and RC traffics respectively, is proposed. Whether the mixed-critical update algorithm is consistent updates or not is proved logically. Finally, experiments analyzed the performance of the mixed-critical update algorithm, compared with four conventional updates. The results show that there is no black hole, loop and inconsistent path for the mixed-critical update algorithm. The most important thing is that the memory space complexity is reduced by 19.93% at least, compared with two-phase update. Meanwhile, the computational time complexity and update time are reduced by 17.14% and 7.96% respectively, compared with time-based update. The mixed-critical update algorithm provides a method to optimize policy update tasks for distributed time-triggered system, so as to improve the ability of system reconfiguration.

INDEX TERMS Consistent update, distributed time-triggered system, software defined networking, space complexity, time complexity.

I. INTRODUCTION

Time-triggered Ethernet (TTE) is a deterministic real-time network for mixed-critical real-time systems, such as industrial automation, aerospace, and aviation [1], [2]. As prerequisite, a global Time-triggered (TT) schedule table in TTE is required for TT traffic to avoid collision among different time-triggered windows [3]. However, a TT frame might have to wait a full transmission period in the ES before being scheduled out, because of asynchrony between tasks and the network in end systems (ESes) [4]. Software Defined Networking (SDN) is a central management system for routinely performing frequent policy and configuration updates by updating flow table which contains the packet paths [5]. Since the centralized controller knows the entire network information, the initial and final configurations can be optimal [6]. SDN was introduced into TTE to enhance real-time and determinacy of TT messages called Software Defined Time-triggered Ethernet (SDTTE) [7]. SDN can also be used for rate-constrained (RC) traffic and best-effort (BE) traffic to update flow tables, which contain message paths.

However, it can be difficult to reconfigure the network correctly and efficiently for SDN, some incorrect transient behaviors are often introduced during network updates. These behaviors impact consistency which contains per-packet consistency and per-flow consistency [8]. The former allows

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

transmitting a packet in either the old path or the new path only, and the latter requires a stream of related packets to be handled by the unique path [9]. Per-packet consistency should be first considered for network reconfiguration in practice. For per-packet consistent update, incorrect transient behaviors main contain congestion, black hole and loop [9].

Congestion is existed in multi-packet update, it may result in unforeseen transient link overload or even congestions [10]–[12]. Loop path and black hole may occur at single packet update. The former leads to waste network resources [13]–[18]. The latter may drop packets during updating flow table [19]. As a basic bug, black hole is often analyzed together with loop [20]–[22]. The three kinds of bugs sometimes are eliminated together [23]–[25].

No matter how complicated the above update algorithms are, they belong to the ordered update which is an obvious way to solve consistent problem. According to the per-packet consistent constraint, the ordered update can avoid most of bugs. But there is a complex algorithm for the ordered update to complete the multi-flow update step-by-step [26]–[32]. The simplest algorithm is two-phase update where the version tag is used to distinguish the new path from the old [9], [33]. The two-phase update is always consistent, but it costs too many Ternary Content Addressable Memory (TCAM) resources, because a large number of flow table entries need concurrent during the whole updating process. Subsequently, Mattos presented a simple algorithm called ''reverse update'' to solve consistent problem with a small quantity of calculation, installation, and collection [34]. Based on the above basic update algorithms, many revised consistent updates are presented [35]–[37]. However, the update time is too long.

Mirrokni first presented a simple polynomial time framework for reduced-path decomposition in multipath routing [38]. Subsequently, Černỳ used polynomial time to optimal consistent network updates [6]. Meanwhile, Mizrahi presented time-based update by clock synchronization to reduce inconsistency rate and update time [39]–[41]. Moreover, Zheng presented time-based update to reroute the updates of multiple network flows with a congestion-free manner in a synchronized SDN [42]. The update time has obvious reduction. But it needs complex computational time which is a Non-deterministic Polynomial Complete (NP-C) problem. Subsequently, Nguyen proposed distributed network update while preserving various consistency properties, the update time and computational time are reduced greatly [43], [44].

Most bugs can be avoided by the above algorithms. However, if a packet is transmitted during updating flow table, it may be a mixed path from old and new paths. Considering TT traffic and RC traffic must have strictly defined paths in TTE, the reliability and security of the TT traffic and RC traffic can not be guaranteed by the above algorithms.

To solve the above problem, we propose a mixed-critical update algorithm which uses the time constraint of TT traffic and the Bandwidth Allocation Gap (BAG) constraint of RC traffic to guarantee the reliability and security of the TT traffic and RC traffic. The primary contributions of this paper are summarized as follows.

1) The update mechanism is changed from centralized control to distributed control for TT traffic and RC traffic in SDTTE. The controller should just provide the boundaries to switches on the old and new paths. The control does not have to provide precise update time. So the central processing unit (CPU) computational complexity can be reduced obviously.
2) A mixed-critical consistent update algorithm is proposed to realize consistency and improve performance for TT and RC traffic in SDTTE.
3) The consistency and performance of our mixed-critical consistent update are demonstrated theoretically and experimentally. The results show that our algorithm satisfies requirements of strict real-time and mixed-critical. Meanwhile, the performance of the mixed-critical update algorithm is improved.

The remainder of this paper is organized as follows. Section 2 describes basic update algorithms and explains consistent update. And then section 3 builds network model and distributed update mechanism. Subsequently, section 4 presents a new mixed-critical update algorithm. In detail, two consistent updates corresponding to TT and RC traffics are presented, respectively. They can satisfy the requirements of strict real-time and mixed-critical, simplify the existing update algorithms, and keep consistency. Section 5 compares different consistent update algorithms by using a simulation experiment based on OMNeT++. Subsequently, the performance of these algorithms, which contains the properties of consistency (black hole, loop, and inconsistent path), memory space complexity, computational time complexity, and update time, are analyzed. Meanwhile, some phenomena in the experiments are described and illustrated. Finally, some conclusions are given in section 6.

II. BACKGROUND

A. BASIC UPDATE ALGORITHMS

1) ORDERED UPDATE

Ordered update programs the specific switch update ordering to avoid conflicts. Sequential update and reverse update are representative ordered updates. Sequential update is performed on the network devices along the new path, except the ingress switch which is updated finally. Reverse update is applied on the reverse sense, from the destination to the source. Namely, it updates the network devices in the reverse sense of the new path in the network [34].

2) TWO-PHASE UPDATE

Two-phase update introduces the network configuration version tag to stamp virtual links. Thus, the version number becomes a property of virtual links. A virtual link $vl_{p,r}$ can be distinguished from another just by the version tag. The two-phase update works by first installing the new configuration on internal ports, but only enabling the new configuration for a packet containing the correct version number. It updates the ingress ports one-by-one to stamp packets with the new version number [9].

3) TIME-BASED UPDATE

Time-based update uses accurate time to trigger consistent updates. It reduces the duration time required by switches to maintain duplicate policy rules for the same flow [40].

B. PER-PACKET CONSISTENCY

Per-packet consistent updates reduce the number of scenarios, a programmer must consider from multi-path to just two paths: for every packet, it is as if the packet flows are transmitted in the network by using the old path completely before the update occurs, or using the new path completely after the update occurs. Inconsistent update contains black hole, and loop, inconsistent path as follows.

1) BLACK HOLE

Black hole means the packet loss in the flow update process. The incoming packet must not be dropped. The process of sequential update is shown in Fig. 1, where the flow table entries in switches are updated from the old path 1–2–3–4–5 to the new path 1–6–3–7–5.

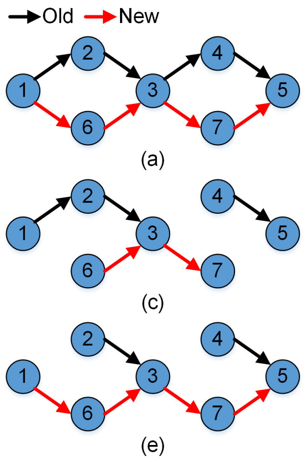


FIGURE 1. Sequential update flowchart with black hole. (Nodes 1–7 are switches, source/destination ESes are hidden).

In sequential update, the ingress switch 1 should be updated finally. The flow tables of switches are updated along the direction 6-3-7-1. First, $SW6$ adds the new flow table entry in Fig. 1(b), as $SW6$ is the new switch for the packet. And then the $SW3$ changes from the old flow table entry to the new flow table entry in Fig. 1(c), as $SW3$ exists in both the old path and the new path. If a packet is transmitted at this moment, the packet may be lost, because there is no receiving node. And then $SW7$ adds the new flow table entry in Fig. 1(d), the packet may be lost at this time. Thereafter, $SW1$ changes the new flow table entry in Fig. 1(e), the black

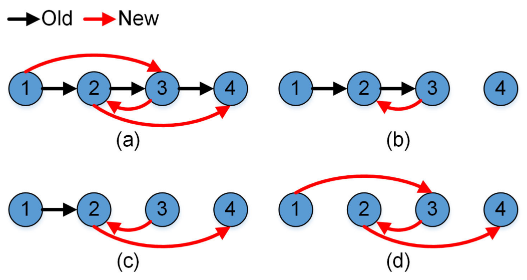


FIGURE 2. Sequential update flowchart with loop path. (Nodes 1–4 are switches, source/destination ESes are hidden).

hole will disappear. Finally, the flow table entries in $SW2$ and $SW5$ are deleted, it is called as garbage collection.

2) LOOP PATH

Loop means that a packet reaches an switch more than once. The network scenario shown in Fig. 2(b) illustrates the loop path problem, where the old path is 1–2–3–4 and needs to be updated to 1–3–2–4.

When the sequential update is also used, the flow tables of switches are updated along the direction 3–2–1. The first update is $SW3$ in Fig. 2(b). If a packet is sent from $SW3$ or $SW2$ at this moment, the path for a packet may become loop (2–3–2) to lead iterative flooding. After the $SW2$ is updated, the iterative flooding could be eliminated.

3) INCONSISTENT PATH

Inconsistent path means that a packet may be transmitted along neither the old path nor the new path as shown in Fig. 3, where the old path is 1–2–3–4–5 and the new path is 1–6–3–7–5.

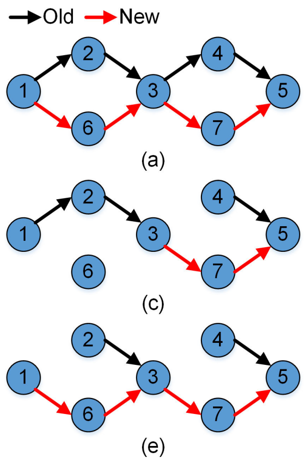


FIGURE 3. Reverse update flowchart with inconsistent path. (Nodes 1–7 are switches, source/destination ESes are hidden).

According to reverse update, flow table will be updated along the path 7–3–6–1. The first update switch is $SW7$ in Fig. 3(b). Subsequently, The $SW3$ is updated as shown in Fig. 3(c). If there is a packet transmitted in the current path at this moment, the path for a packet may be another mixed path 1–2–3–7–5. Similarly, the inconsistent path also exists after $SW6$ is updated as shown in Fig. 3(d). After $SW1$ is updated as shown in Fig. 3(e), the inconsistent path could disappear. Finally, the flow table in the $SW2$ is collected as a garbage.

III. NETWORK ARCHITECTURE

A. NETWORK MODEL

A TTE network is a full duplex switched network with time-triggered and event-triggered mechanism. A TTE network can be formally modeled as a directed graph $G(V, E)$, where the set of vertices V comprises the communication nodes (switches and end systems, *SWs* and *ESes*) and the edges E represent the directional communication links between nodes, similar to Zhao [45].

And a vertex $v_i \in V$ can be denoted by the following tuple.

$$v_i =< \alpha_i, \beta_i, \gamma_i > \tag{1}$$

where α_i is the arrival curve which is obtained by all packets arrived at the vertex v_i. β_i is the service curve in the vertex v_i. γ_i denotes the size of occupied TCAM resources in the vertex v_i.

$$\alpha_i = \rho_i(t + \sigma_i/\rho_i)^+ = \begin{cases} \sigma_i + \rho_i t, & t > 0 \\ 0, & others \end{cases} \tag{2}$$

$$\beta_i = S_i(t - D_i)^+ = \begin{cases} S_i(t - D_i), & t > 0 \\ 0, & others \end{cases} \tag{3}$$

where ρ_i is the sustained bitstream speed in the vertex v_i, σ_i is maximum burstiness caused by packet length, S_i is the service speed, D_i is inherent delays caused by the switch.

For an edge $e_{i,j}$ with the bandwidth of physical links $C_{e_{i,j}}$ (e.g. 100 Mbit/s, 1 Gbit/s, etc.), it can be given by

$$e_{i,j} = v_i - v_j \in E, \quad v_i, v_j \in V \tag{4}$$

A VL is a logical data-flow path in the network from one sending node to receiving nodes. A typical virtual link $vl_{p,r}$ from a producer task running on an ES v_{e1} to a consumer task running on an ES v_{e2}, routed through the switches $v_1, \cdots, v_n$ is expressed.

$$\begin{aligned} vl_{p,r} &= e_{e1,1}e_{1,1}e_{1,2}\cdots e_{n,n}e_{n,e2} \\ &= v_{e1} - v_1, v_1 - v_1, v_1 - v_2, \cdots, v_n - v_n, v_n - v_{e2} \\ &= v_{e1} - v_1 - v_2 - \cdots - v_n - v_{e2} \end{aligned} \tag{5}$$

where $vl_{p,r}$ is a virtual link (VL) for the r^{th} update of a message m_p, and $e_{n,n} = v_1 - v_1$ denotes a packet delivering in a switch, it can be ignored.

Let M denote the set of all messages in the system. We model a message $m_p \in M$ by the tuple.

$$m_p =< P_p, L_p > \tag{6}$$

where P_p is the period and L_p is the size in bytes. A packet $m_{p,q}$ can map the q^{th} instance of the message m_p.

$$m_{p,q} =< T_{p,q}, vl_{p,r}, P_p, L_p > \tag{7}$$

where $T_{p,q}$ is the generated time of a packet $m_{p,q}$ in an end system (ES).

In the TTE network, there are three traffic classes, TT traffic, RC traffic and BE traffic.

- A TT traffic needs a global synchronized clock to support the TT schedule table which is a flow table with the sending/receiving time window.
- A RC traffic has a Bandwidth Allocation Gap (BAG) which is the minimum time interval between two consecutive packets of a RC traffic in the source end system and a maximum allowable delay (MAD) which limits the delay boundary. Generally, the two constraints have the same value, we define the value in P_p. Because RC traffic is event-triggered and has no period P_p. Meanwhile, RC traffic needs a flow table to storage the traffic paths $vl_{p,r}$. Generally, if system is not reconfigured, the RC flow table should be static to guarantee the reliability and security of the RC traffic.
- A BE traffic is a low-priority Ethernet traffic without time-triggered and BAG constraints, but the delay time should pay respect to retransmission timeout (RTO) with automatic repeat-request (ARQ). Meanwhile, BE traffic also needs a flow table to storage the traffic paths $vl_{p,r}$. However, the BE flow table doesn't have to be static.

B. NETWORK FRAMEWORK

To increase the time determinacy of TTE degraded by the asynchrony between tasks and the network in end systems, Software Defined Networking (SDN) is introduced into TTE called software defined time-triggered Ethernet (SDTTE) as shown in Fig. 4 [7]. And TTE switches must have the ability of software defined path.

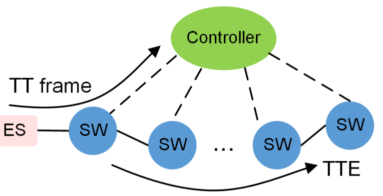


FIGURE 4. The network architecture of Software Defined TTE.

When the system need be reconfigured, there is no flow table or a huge difference between the generated time $T_{p,q}$ and the triggered time, the switch would report the TT message to a centralized controller so that the centralized controller can obtain the received time of the TT traffic from the source ES to the first connected switch.

When the system need not be reconfigured, we defined a threshold and time drift which are relative to the first generated time $T_{p,q+1}$ for a TT traffic after last update. If the time drift exceeds the threshold, the flow table will be reconfigured online. The threshold avoids big jitters for TT traffic during updating the online TT schedule table.

No matter what the state of the system is, the TT schedule tables required by TTE switches can be updated by the corresponding controller as follows.

C. DISTRIBUTED UPDATE MECHANISM

A logically-centralized controller directly manages the distributed network by configuring the update mechanisms. Since flow tables are installed in switches, it needs a feasible update table, which contains order and time to complete updating all flow tables step-by-step [26]–[28]. However, obtaining a feasible update table is NP-C problem, calculation is complex [20]–[22].

Considering time constraint and BAG constraint, the distributed update mechanism is only used for the TT and RC traffic. While the centralized controller provides the time window constraints. When update requests come for the TT and RC traffic, we calculate the new TT schedule table and RC flow table first. According to old and new schedule/flow table, the time window of the update table is obtained by our mixed-critical algorithm. In the new TT schedule table and RC/BE flow table, each schedule/flow table entry must be sent before the time window in update table so that switches can calculate the precise update time by Earliest Deadline First (EDF) algorithm. At this moment, the precise update time in a switch is a subset of the update time in controller. If the consistent update can be realized by conventional centralized control, it can be realized by the distributed control.

The transmission delay between controller and each switch is stable and same for each switches, because they are single hop. And we can calculate the max transmission delay by maximum round-trip time (RTT) once. The arrival time had better close to the lower bound of time window, it can reduce the waiting time of new flow table so that the TCAM resources can be occupied less.

IV. MIXED-CRITICAL UPDATE

In ethernet, we does not have enough constraint to set time window. So general ethernet can not realize the above distributed control mechanism. In SDTTE, the time constraint and BAG constraint make the update mechanism of distributed control possible for TT traffic and RC traffic, respectively. According to the update mechanism of distributed control for TT traffic and RC traffic, a mixed-critical update algorithm in SDTTE is illustrated as follows.

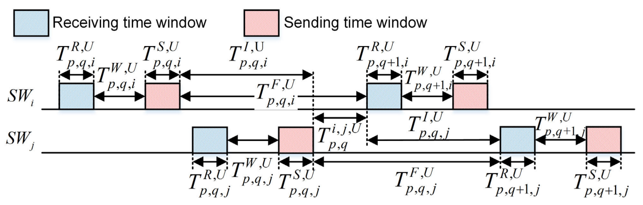


FIGURE 5. **Scheduling process of the TT traffic in switches.**

A. TIME WINDOW-BASED UPDATE FOR TT TRAFFIC

When there is no updated TT schedule table for TT messages in switches, they are scheduled as shown in Fig. 5, where $T^{R,U}_{p,q,i}$ denotes the receiving time window of the packet $m_{p,q}$ with the path version U in the switch i, $T^{W,U}_{p,q,i}$ is the waiting delay, and $T^{S,U}_{p,q,i}$ denotes the sending time window. $T^{F,U}_{p,q,i}$ is the time window from the sending deadline for a packet $m_{p,q}$ to the receiving start time for the next packet $m_{p,q+1}$ in the switch i, $T^{i,j,U}_{p,q}$ is the common part of time windows $T^{F,U}_{p,q,i}$ and $T^{F,U}_{p,q,j}$ in two switches i and j, and $T^{I,U}_{p,q,i}$ denotes the remaining part of time window for a time window $T^{F,U}_{p,q,i}$ in the switch i, namely, $T^{I}_{p,q,i} = T^{F,U}_{p,q,i} - T^{i,j,U}_{p,q}$.

If a packet $m_{p,q+1}$ is transmitted by using a new path first, the corresponding TT schedule rule need be updated before the packet is generated $m_{p,q+1}$, $T^{F,N}_{p,q,n}$ is the time window from the sending deadline for a packet $m_{p,q}$ to the new receiving start time for the next packet $m_{p,q+1}$ with the path version N in the switch n, and $T^{i,j,N}_{p,q}$ is the common part of time windows $T^{F,N}_{p,q,i}$ and $T^{F,N}_{p,q,j}$.

For switches i and j in the new path, the common part of time windows between the old path and the new path can be described by

$$T^{i,j}_{p,q} = \begin{cases} T^{i,j,O}_{p,q} \cap T^{i,j,N}_{p,q}, & i,j \in SW_{p,O} \\ T^{i,j,N}_{p,q}, & others \end{cases} \tag{8}$$

where $SW_{p,O}$ is the switch set for a message m_p with the old path O.

The simplest way to solve the updating problem is to let the update operation take place in the common time window,

$$T^{C}_{p,q} = \bigcap_{i,j \in SW_{p,N}} T^{i,j}_{p,q} \tag{9}$$

If the common time window $T^{C}_{p,q}$ exists, all switches in the new path can update their TT schedule tables at the same time window. But the common time window $T^{C}_{p,q}$ don't always exist. In other words, $T^{C}_{p,q} = \phi$ may often occur in multi-hop network. The simplest update is not a universal method and can not be used to update TT schedule table.

The switches in the new path N should update their TT schedule tables separately. The packet $m_{p,q+1}$ should not be disturbed by the old TT schedule table, so the update must happen before both the old and the new receiving time windows. Without knowing if there are any updates, the certain time window $T^{F}_{p,q,n}$ can be described by

$$\begin{aligned} T^{F}_{p,q,n} &= T^{F,O}_{p,q,n} \cap T^{F,N}_{p,q,n} \\ &= \left[\Theta^{S,O}_{p,q,n}, min\left(\Theta^{R,O}_{p,q+1,n}, \Theta^{R,N}_{p,q+1,n}\right)\right] \end{aligned} \tag{10}$$

where $\Theta^{S,O}_{p,q,n}$ is the sending deadline of a packet $m_{p,q}$ in the switch n with the old TT schedule table, $\Theta^{R,O}_{p,q+1,n}$ is the receiving beginning time of a packet $m_{p,q+1}$ with the old TT schedule table and $\Theta^{R,N}_{p,q+1,n}$ is the receiving beginning time of a packet $m_{p,q+1}$ with the new TT schedule table.

It needs installation time for updating flow table, so in the detailed process for TT traffic updating, the installation time should be counted in the decision of updating window. Without loss of generality, we assume that the maximum installation time of a flow table is d_m. The formula (10) can be converted into

$$T^F_{p,q,n} = \left[\Theta^{S,O}_{p,q,n}, min\left(\Theta^{R,O}_{p,q+1,n}, \Theta^{R,N}_{p,q+1,n}\right) - d_m\right] \quad (11)$$

Definition 1: Time window update changes a flow table entry in switch n during the time window $T^F_{p,q,n} \in T^F_{p,q}$ as shown in Algorithm 1, when a flow table for a packet need be updated.

Algorithm 1 Time Window Update

```
1:  $\textbf{\textit{Data}}$ : $vl_{old}, vl_{new}, schedule_{old}, schedule_{new}$;
2:  $\textbf{\textit{Result}}$ : $T^F_{p,q}$;
3:  $\textbf{\textit{Initialize}}$ : $T^F_{p,q} \leftarrow \emptyset$;
4:  for $i$ 1 $to$ $n$ $by$ 1 do
5:      if $SW_i \in vl_{new}$ then
6:          $\textbf{\textit{calculate}}$ : $T^F_{p,q,n}$, according to equation (11);
7:          $\textbf{\textit{update}}$ : $T^F_{p,q} \leftarrow T^F_{p,q,n}$;
8:      end if
9:  end for
10: return $T^F_{p,q}$;
```

Theorem 1: Time window update for TT traffic is a per-packet consistent update.

Proof of Theorem 1: There is no transmitted TT message during the corresponding update, so black hole and loop could never occur.

Meanwhile, each packet is scheduled by the either old or new TT schedule table.

So time window-based update for TT traffic is a per-packet consistent update. □

B. RATE-CONSTRAINED UPDATE FOR RC TRAFFIC

There is no schedule table for the RC traffic, so time window update can not be applied to RC traffic directly. The simplest referable update is the two-phase update. But it takes up a lot of TCAM resources during update. We present the rate-constrained update for RC traffic to reduce the requirement of TCAM resources.

Definition 2: The rate-constrained update for RC traffic is shown in Algorithm 2. More specifically, it updates new added switches SW_n first. And then based on the time $T^{lastold}_{p,q-1}$ that the last packet using old path arrives at its first switch, the rate-constrained update performs path updates of common switches SW_c belonging to both old path and new path during the time window between the worst-case end to end delay and the maximum allowable delay. Finally, it performs garbage collection to delete the flow table entry in old switches SW_o immediately.

In Algorithm 2, D_p denotes the worst-case end to end delay, it can be computed by network calculus (NC) using equation (2) (3) [45]. The time window $T^F_{p,q}$ limits the update time of common switches SW_c.

$$T^F_{p,q} = \left[T^{lastold}_{p,q-1} + D_p, T^{lastold}_{p,q-1} + P_p - d_m\right) \quad (12)$$

where the maximum allowable delay is the same as its period.

Algorithm 2 The Rate-Constrained Update

```
1:  $\textbf{\textit{Data}}$ : $vl_{old}, vl_{new}, P_p, D_p$;
2:  $\textbf{\textit{Result}}$ : $SW_n, SW_c, SW_o, T^F_{p,q}$;
3:  $\textbf{\textit{Initialize}}$ : $SW_n \leftarrow \emptyset, SW_c \leftarrow \emptyset, SW_o \leftarrow \emptyset$;
4:  $\textbf{\textit{calculate}}$ : $T^F_{p,q}$, according to equation (12);
5:  for $i$ 1 $to$ $n$ $by$ 1 do
6:      if $SW_i \in vl_{new}$ then
7:          if $SW_i \notin vl_{old}$ then
8:              $SW_n \leftarrow SW_i$;
9:          else
10:             $SW_c \leftarrow SW_i$;
11:         end if
12:     else
13:         if $SW_i \in vl_{old}$ then
14:             $SW_o \leftarrow SW_i$;
15:         end if
16:     end if
17: end for
18: return $SW_n, SW_c, SW_o, T^F_{p,q,n}$;
```

Theorem 2: The rate-constrained update is a per-packet consistent update.

Proof of Theorem 2: If there is no common switch between the old path and the new path, namely, the old path and the new path are mutually independent, so black hole and loop can not occur, and packets follow the per-packet consistency. In the case of no common switch, rate-constrained update is a per-packet consistent update. Meanwhile, the second step can be ignored.

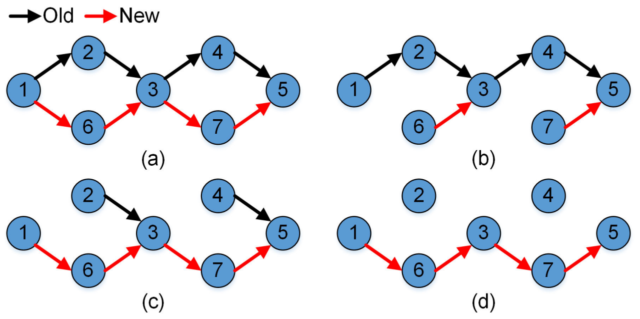


FIGURE 6. An instance of the rate-constrained update flowchart without black holes. (Nodes 1–7 are switches, source/destination ESes are hidden).

If there are some common switches between the old path and the new path, rate-constrained update has three steps. After updating new added switches, the packet will continue transmitting on old path as shown in Fig. 6(b). Due to BAG constraint is same as MAD, there is no packet between the worst-case end to end delay D_p and MAD so that there is sufficient time for updating common switches. And the garbage collection is performed immediately. When a new packet is generated, it will be transmitted on the new path. Thus rate-constrained update is a per-packet consistent update.

According to the above classified discussions, the rate-constrained update for RC traffic is a per-packet consistent update. □

When old and new paths are shown in Fig. 3(a), the rate-constrained update performs the three steps update. It updates new added switches $SW6$ and $SW7$ first. And then all common switches are updated during the time window $T^F_{p,q}$. Finally, the garbage collection is performed. It eliminates black holes as shown in Fig. 6.

Meanwhile, when old and new paths are shown in Fig. 2(a), there is no new switch for new path, every switch belongs to common switch sets. The rate-constrained update only performs the second step, all switches are updated during the time window $T^F_{p,q}$. It eliminates loops.

C. CONVENTIONAL UPDATE FOR BE TRAFFIC

The rate-constrained update can not be applied to the RC traffic directly, because there is no generated time constraint for the BE traffic, not like either TT traffic or RC traffic. The RC traffic can uses more loose strategy to update flow table, such as conventional time-based update.

For transmission control protocol (TCP) pattern, BE traffic must satisfy retransmission timeout (RTO) constraint to limit the end to end delay, which can be gotten by the maximum round-trip time (RTT) from the sending time of a message to the receiving time of the corresponding acknowledgement message (ACK). Garbage collection is performed with RTO for TCP pattern or a default value for user datagram protocol (UDP) pattern [40].

V. CONSISTENT UPDATE EXPERIMENT

A. SIMULATION MODEL

The self-designed software based on OMNeT++ is used to perform the experiment to evaluate the performance of the mixed-critical update algorithm. The switch module and the controller module are similar to Klein and Salih [46], [47]. But the buffers in switch module and the centralized controller module are divided into three types, TT buffers, RC buffers, and BE buffers.

An ES module is displayed in Fig. 7, where Sflow modules generate all packets with different traffic types, Rflow modules receive all packets with different traffic types, flowDispatch module is responsible for dispatching packets to different Sflow/Rflow modules, buffer modules temporarily store different packets, and scheduler module realizes scheduling for different traffics according to different traffic constraints. flowCheck module is used to check whether the received packets satisfy rules, such as the TT schedule table and BAG constraint.

The bandwidth of each link is 100Mbit/s. Message sources and destinations are generated randomly. When the period of TT messages is small (such as 2ms and 4ms), TT messages can not be scheduled as the solution space is too big to traverse. So in our experiment, the period of TT messages is selected from set (16ms, 32ms, and 64ms). The BAG of RC messages is a random value among 2ms, 4ms, and 8ms to make full use of bandwidth in the absence of link congestion. The link congestion may lead to packet loss, it influences statistics of black hole.

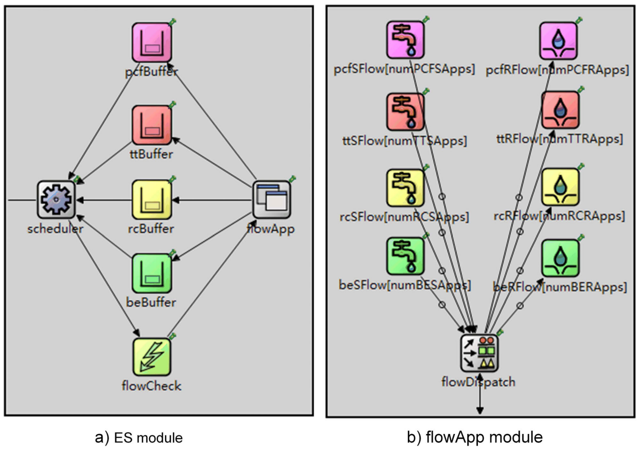


FIGURE 7. Implemented model of the ES.

B. INCONSISTENT UPDATE AND ASSESSMENT

1) NETWORK CONFIGURATION

The inconsistent bugs, which includes black holes, loop and inconsistent path, rarely appear in usual complex networks. Because any one update algorithm eliminates the most of bugs. We find the network topology as shown in Fig. 8, where black holes, loop and inconsistent path can appear with a high probability during the path update. There are four end systems and seven switches in the network, where $SW6$ and $SW7$ are standby switches which are used in the new path. Meanwhile, there is a SDN centralized controller connecting with each switch to dispatch flow tables. The initial physical links are green and blue physical links. To avoid congestion between old paths and new paths during updating, green physical links are break, pink physical links and pink switches are added to reconfigure network topology.

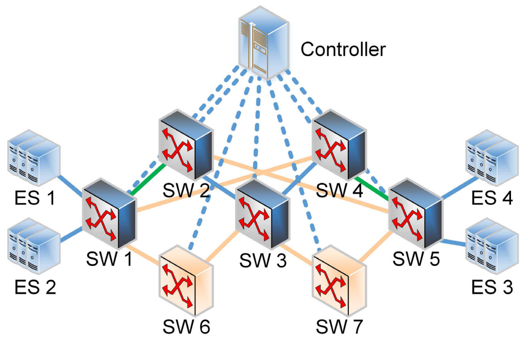


FIGURE 8. Network topology in inconsistent update.

According to the old and new network topologies, each path is updated by another path, except four paths with a single switch, such as $ES1$–$SW1$–$ES2$. And others have the old undirected path ($SW1$–$SW2$–$SW3$–$SW4$–$SW5$). All new paths are subdivided equally into two categories. The one half is updated by reverse paths ($SW1$–$SW4$–$SW3$–$SW2$–$SW5$) similar to Fig. 2(a), and the another half is updated by their alternate paths ($SW1$–$SW6$–$SW3$–$SW7$–$SW5$) as same as Fig. 1(a). According to the section II-B, black hole, loop and inconsistent path are all highly likely to occur in this path update. The results can show the difference among algorithms well.

According to old paths of messages and the rule of new paths, we can obtain the new paths of the random 128 messages. To schedule TT messages in the above network, the number of TT messages can not be too many, it is cautiously set to 16. The number of RC messages and BE messages are randomly set to 60 and 52 respectively. Except the messages in the four paths with a single switch, there are 5 TT messages, 19 RC messages and 15 BE messages updated by its alternate paths. Meanwhile, there are 5 TT messages, 18 RC messages and 17 BE messages updated by its reverse paths. The conventional update algorithms are sequential update, time-based update, two-phase update and reverse update. Our mixed-critical update algorithm will be compared with these four update algorithms in terms of black holes, loops, and inconsistent paths.

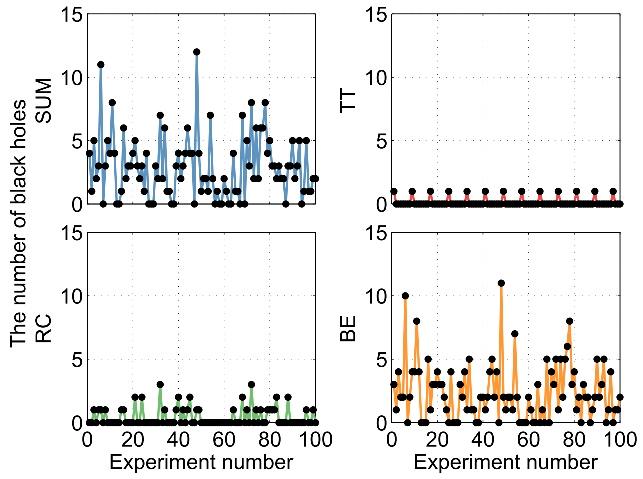


FIGURE 9. **The number of black holes in sequential update.**

2) BLACK HOLE BUGS

Black hole may only occur in the sequential update, when $SW3$ and $SW7$ are updated by its alternate paths. The number of black holes for different types of messages is shown in Fig. 9. The results show that the maximum of black holes in sequential update are 1, 3, and 11 for TT messages, RC messages and BE messages, respectively. So the maximum ratios of black holes are 20%, 15.79%, and 73.33% in alternate paths. All update algorithms have no black hole, except sequential update. The sequential update can not be applied to TT traffic or RC traffic in SDTTE.

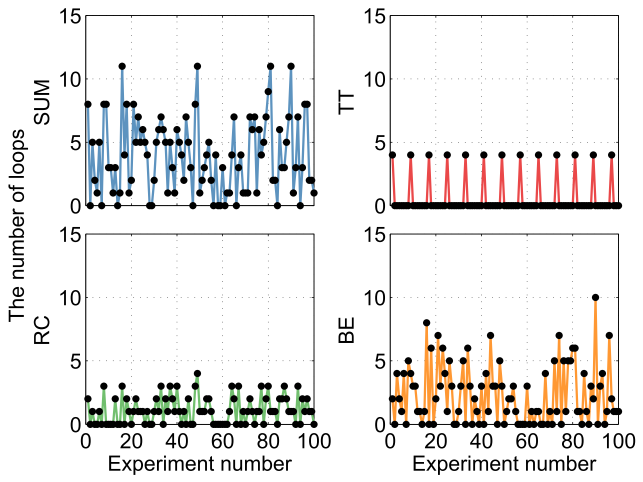


FIGURE 10. **The number of loops in sequential update.**

3) LOOP BUGS

Loop may occur in the sequential update, when $SW3$ and $SW4$ are updated by its reverse paths. The number of loops for different types of messages is shown in Fig. 10. The results show that the maximum of loops in sequential update are 4, 4, and 10 for TT messages, RC messages and BE messages, respectively. So the maximum ratios of loops are 80%, 22.22%, and 58.82% in reverse paths. All update algorithms also have no loop, except sequential update. It also proves that the sequential update can not be applied to TT traffic or RC traffic in SDTTE.

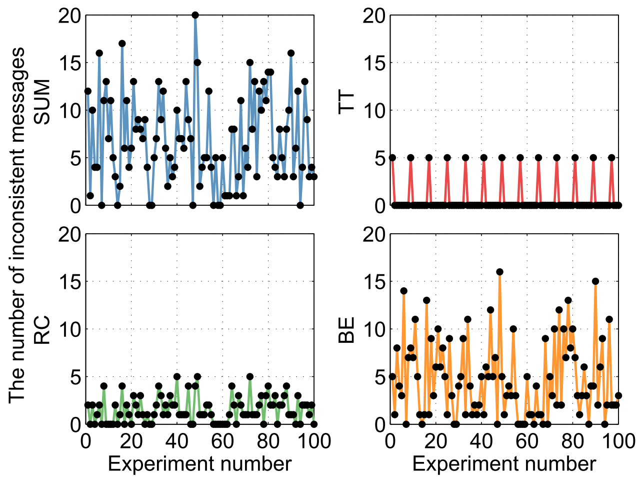


FIGURE 11. **The number of inconsistent paths in reverse update.**

4) INCONSISTENT PATH BUGS

Inconsistent path may occur in the sequential update and reverse update, especially in reverse update. Inconsistent path bugs may exist in reverse update, if there are transmitted packets during updating corresponding flow tables. The number of inconsistent paths for different traffics in reverse update is shown in Fig. 11. The results show that the maximum of loops in reverse update are 5, 5, and 16 for TT messages, RC messages and BE messages, respectively. So the maximum ratios of inconsistent paths in reverse update are 50%, 13.51%, and 50% in all update paths. The results show that reverse update can not be applied to TT traffic or RC traffic. Because TT traffic and RC traffic must satisfy requirements of strict real-time and mixed-critical in SDTTE.

C. PERFORMANCE ANALYSIS

1) NETWORK CONFIGURATION

To analyze the performance of the mixed-critical update algorithm, the above special network is not suitable. To finish experiments efficiently with a general computer (CPU is i3-M380, and memory is 4G), we build a larger suitable network, where flow table, TT schedule table and update table can all be calculated in 30 minutes. The network is mesh network shown in Fig. 12, all core switches *SWs* are interconnected with each other. And each core switch is connected with 2 edge switches *ESWs*. All 8 edge switches *ESWs* are connected with 8 ESes. So there are 64 ESes in the network totally. Meanwhile, there is a SDN centralized controller connected with all *SWs* and *ESWs* to dispatch flow tables.

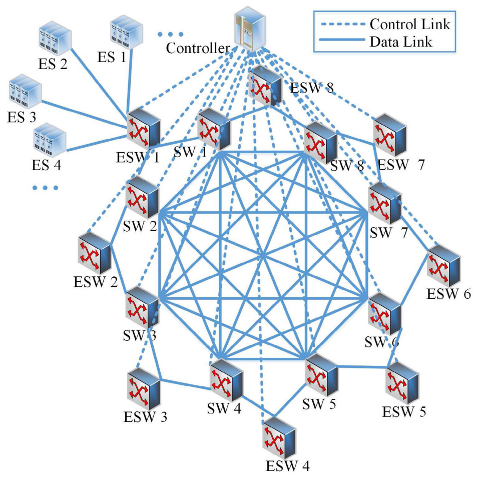


FIGURE 12. Network topology in performance analysis.

We generate different numbers of messages with the random paths to analyze algorithm performance. According to actual situations in distributed real-time system, the ratio of TT, RC, and BE traffics is 1:3:4. The conventional update algorithms are sequential update, time-based update, two-phase update and reverse update. Our mixed-critical update algorithm will be compared with these four update algorithms in terms of memory space complexity, computational time complexity, and update time as follows.

2) MEMORY SPACE COMPLEXITY

The space complexity describes that an update algorithm occupies the sizes of TCAM resources. Different traffics have

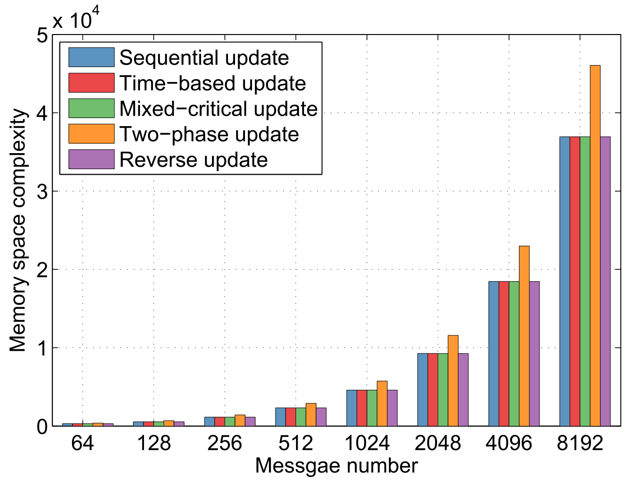


FIGURE 13. The memory space complexity of all update algorithms with different amount of messages.

different memory information, such as TT traffic has the triggered time uniquely. To eliminate the impact of the number of randomly generated different traffics, the maximum number of flow table entries in the whole network can reflect the memory space complexity better. The memory space complexity of all update algorithms are shown in Fig. 13. The results show that the maximum number of flow table entries processed by two-phase update is more than others. Namely, Two-phase update needs the more TCAM resources than others. Except two phase update, the memory space complexities of other different update algorithms are same. The updates without version tag reduce TCAM resource requirements by an average of 19.93%.

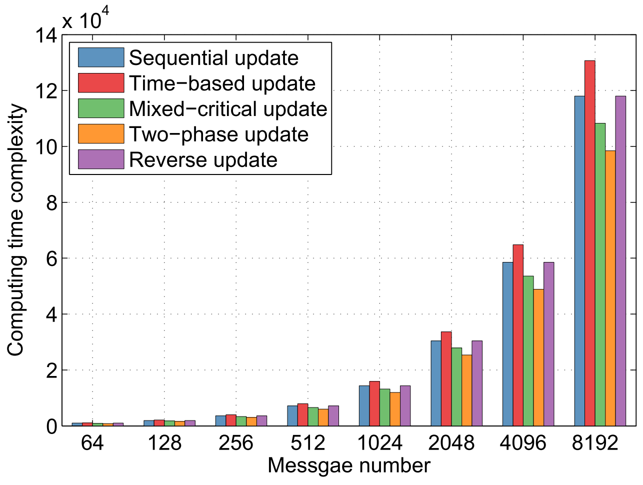


FIGURE 14. The time complexity of all update algorithms with different amount of messages.

3) COMPUTATIONAL TIME COMPLEXITY

Computational time complexity is the number of CPU instructions. Different consistent update algorithms also have different computational time complexities for updating the flow table as shown in Fig. 14. The result shows that the computational time complexity of two-phase update is the least than others. Namely, it takes the least computing resources to update flow table, compared with others. However, the mixed-critical update has the lowest time complexity in all four updates without version tag. The mixed-critical update reduced computational time complexity by an average of 17.14%, compared with time-based update.

4) UPDATE TIME

Update time of different update algorithms with different amount of messages are shown in Fig. 15. The results show that the update time of two-phase update is also the least than others. However, the mixed-critical update has the lowest time complexity in all four updates without version tag. The mixed-critical update reduced update time by an average of 7.96%, compared with time-based update.

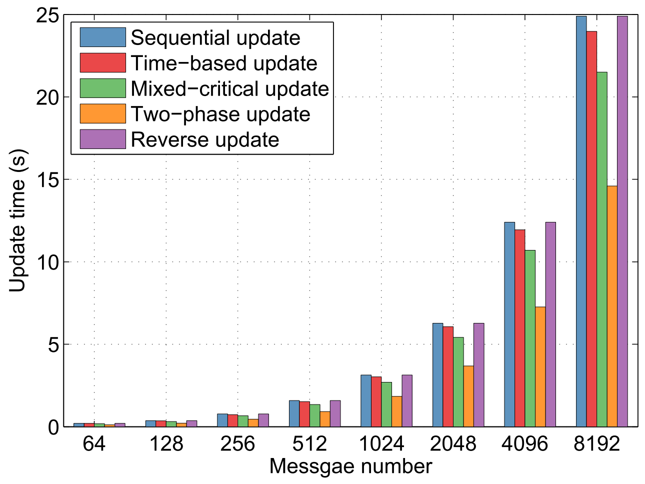


FIGURE 15. The update time of all update algorithms with different amount of messages.

D. PHENOMENA ANALYSIS

As shown in Fig. 9, Fig. 10, and Fig. 11, the black holes, loops, and inconsistent path for TT traffic are periodic in the 100 times independent repeat experiments. Because the initial updated instant of the flow table is periodic and TT traffic are transmitted periodically according to TT schedule table. However, RC and BE traffics are not time-triggered, so their black holes, loops, and inconsistent path are not periodic in the experiments.

Except two-phase update, other four algorithms are the same memory space complexity. Because they have the similar operations (add, replace and delete) in the same situation. For the four algorithms, the main differences of their update mechanisms are operating time and operating sequence.

If there are old packets in $SW2 - SW4$ during updating its reverse paths or there are old packets in $SW2 - SW3$ during updating its alternate paths, inconsistent path bugs may exist in time-based update in practice. However, the probability of inconsistent path bugs is very small. Due to the time constraint and BAG constraint, there is no packet in the network before we update corresponding flow table entry of TT and RC traffics. Our mixed-critical update algorithm guarantees requirements of strict real-time and mixed-critical in SDTTE absolutely.

VI. CONCLUSION

Consistency is a critical prerequisite for SDN to ensure network updates. Per-packet consistent updates reduce the number of scenarios, and improve network security. Just two paths must be considered, namely the packet flows through the network completely before the update occurs, or after the update occurs. Due to the black hole, loop, and inconsistent path, network updates are not per-packet consistent. Through analyzing the traffic constraints in SDTTE, there is a potential to optimize the existing updates for time-triggered Ethernet by traffic constraints.

First, the update mechanism is changed from the centralized control to distributed control in SDTTE. The centralized controller need not calculate the precise update time for the TT and RC traffic, Only the boundary of the update time for the TT and RC trafficis constrained by the centralized controller. It reduces the computational time complexity.

And then a mixed-critical update algorithm is presented for the TT and RC traffic to guarantee their reliability and security. At the same time, the mixed-critical update algorithm is per-packet consistent for the TT and RC traffic strictly.

Finally, the 100 independent repeat experiments are realized to analyze the performance of our mixed-critical update algorithm. The statistical results show that the mixed-critical update algorithm are consistent update. Compared with two-phase update, the mixed-critical update reduces TCAM resources by more than 19.93%. Meanwhile, compared with time-based update algorithm, the computational time complexity and update time in the mixed-critical update algorithm reduces by 17.14% and 7.96%.

ACKNOWLEDGMENT

The authors wish to acknowledge the efforts of all the entities contributing to the Open Networking Foundation (ONF) and TTTech. The colleagues who provided insight and stimulating discussions that greatly assisted in the preparation of this paper. The authors gratefully acknowledge the anonymous reviewers for carefully reading the paper and providing constructive comments.

REFERENCES

[1] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The time-triggered Ethernet (TTE) design," in *Proc. 8th IEEE Int. Symp. Object-Oriented Real-Time Distrib. Comput. (ISORC)*, Seattle, WA, USA, May 2005, pp. 22–33, doi: 10.1109/isorc.2005.56.

[2] W. Steiner, G. Bauer, B. Hall, M. Paulitsch, and S. Varadarajan, "TTEthernet dataflow concept," in *Proc. 8th IEEE Int. Symp. Netw. Comput. Appl.*, Cambridge, MA, USA, Jul. 2009, pp. 319–322, doi: 10.1109/nca.2009.28.

[3] H. Kopetz, "Event-triggered versus time-triggered real-time systems," in *Operating Systems Beyond,*. Berlin, Germany: Springer, 1991, pp. 86–101, doi: 10.1007/BFb0024530.

[4] M. Boyer, H. Daigmorte, N. Navet, and J. Migge, "Performance impact of the interactions between time-triggered and rate-constrained transmissions in ttethernet," *Die Medizinische Welt*, vol. 14, no. 2, pp. 217–225, 2016, doi: 10.1017/S0021859600022838.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–79, Mar. 2008, doi: 10.1145/1355734.1355746.

[6] P. Černý, N. Foster, N. Jagnik, and J. McClurg, "Optimal consistent network updates in polynomial time," in *Proc. Int. Symp. Distrib. Comput.* Berlin, Germany: Springer, 2016, pp. 114–128, doi: 10.1007/978-3-662-53426-7_9.

[7] J. Lu, H. Xiong, F. He, and R. Wang, "Enhancing real-time and determinacy for network-level schedule in distributed mixed-critical system," *IEEE Access*, vol. 8, pp. 23720–23731, 2020, doi: 10.1109/access.2020.2970266.

[8] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, "Consistent updates for software-defined networks: Change you can believe in!" in *Proc. 10th ACM Workshop Hot Topics Netw.*, 2011, p. 7, doi: 10.1145/2070562.2070569.

[9] 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, Sep. 2012, doi: 10.1145/2377677.2377748.

[10] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz, "ZUpdate: Updating data center networks with zero loss," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 411–422, doi: 10.1145/2486001.2486005.

[11] J. Zheng, H. Xu, G. Chen, and H. Dai, "Minimizing transient congestion during network update in data centers," in *Proc. IEEE 23rd Int. Conf. Netw. Protocols (ICNP)*, Nov. 2015, pp. 1–10, doi: 10.1109/icnp.2015.33.

[12] W. Wang, W. He, J. Su, and Y. Chen, "Cupid: congestion-free consistent data plane update in software defined networks," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9, doi: 10.1109/infocom.2016.7524420.

[13] J. Fu, P. Sjodin, and G. Karlsson, "Loop-free updates of forwarding tables," *IEEE Trans. Netw. Service Manage.*, vol. 5, no. 1, pp. 22–35, Mar. 2008, doi: 10.1109/tnsm.2008.080103.

[14] L. Shi, J. Fu, and X. Fu, "Loop-free forwarding table updates with minimal link overflow," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2009, pp. 1–6, doi: 10.1109/icc.2009.5199146.

[15] A. Ludwig, M. Rost, D. Foucard, and S. Schmid, "Good network updates for bad packets: Waypoint enforcement beyond destination-based routing policies," in *Proc. 13th ACM Workshop Hot Topics Netw.*, 2014, p. 15, doi: 10.1145/2670518.2673873.

[16] K.-T. Forster and R. Wattenhofer, "The power of two in consistent network updates: Hard loop freedom, easy flow migration," in *Proc. 25th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2016, pp. 1–9, doi: 10.1109/icccn.2016.7568583.

[17] K.-T. Foerster, A. Ludwig, J. Marcinkowski, and S. Schmid, "Loop-free route updates for software-defined networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 328–341, Feb. 2018, doi: 10.1109/tnet.2017.2778426.

[18] A. Basta, A. Blenk, S. Dudycz, A. Ludwig, and S. Schmid, "Efficient loop-free rerouting of multiple SDN flows," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 948–961, Apr. 2018, doi: 10.1109/tnet.2018.2810640.

[19] L. Luo, H. Yu, S. Luo, and M. Zhang, "Fast lossless traffic migration for SDN updates," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 5803–5808, doi: 10.1109/icc.2015.7249247.

[20] A. Ludwig, J. Marcinkowski, and S. Schmid, "Scheduling loop-free network updates: It's good to relax!" in *Proc. ACM Symp. Princ. Distrib. Comput. (PODC)*, 2015, pp. 13–22, doi: 10.1145/2767386.2767412.

[21] K.-T. Forster, R. Mahajan, and R. Wattenhofer, "Consistent updates in software defined networks: On dependencies, loop freedom, and blackholes," in *Proc. IFIP Netw. Conf. (IFIP)*, May 2016, pp. 1–9, doi: 10.1109/ifipnetworking.2016.7497232.

[22] S. Dudycz, A. Ludwig, and S. Schmid, "Can't touch this: Consistent network updates for multiple policies," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2016, pp. 133–143, doi: 10.1109/dsn.2016.21.

[23] R. Mahajan and R. Wattenhofer, "On consistent updates in software defined networks," in *Proc. 12th ACM Workshop Hot Topics Netw.*, 2013, p. 20, doi: 10.1145/2535771.2535791.

[24] Y. Yuan, F. Ivančić, C. Lumezanu, S. Zhang, and A. Gupta, "Generating consistent updates for software-defined network configurations," in *Proc. 3rd workshop Hot topics Softw. Defined Netw.*, 2014, pp. 221–222.

[25] K.-T. Foerster, S. Schmid, and S. Vissicchio, "Survey of consistent software-defined network updates," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1435–1461, 2nd Quart., 2019, doi: 10.1109/comst.2018.2876749.

[26] N. P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 49–54.

[27] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 15–26, Aug. 2013, doi: 10.1145/2534169.2486012.

[28] S. Brandt, K.-T. Forster, and R. Wattenhofer, "On consistent migration of flows in SDNs," in *Proc. IEEE INFOCOM - 35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9, doi: 10.1109/infocom.2016.7524332.

[29] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 539–550, Aug. 2014, doi: 10.1145/2740070.2626307.

[30] J. McClurg, H. Hojjat, P. Černý, and N. Foster, "Efficient synthesis of network updates," *ACM SIGPLAN Notices*, vol. 50, no. 6, pp. 196–207, Jun. 2015, doi: 10.1145/2813885.2737980.

[31] W. Zhou, D. Jin, J. Croft, M. Caesar, and P. B. Godfrey, "Enforcing customizable consistency properties in software-defined networks," in *Proc. 12th USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, 2015, pp. 73–85.

[32] S. Vissicchio and L. Cittadini, "FLIP the (Flow) table: Fast lightweight policy-preserving SDN updates," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.

[33] S. Luo, H. Yu, and L. Li, "Consistency is not easy: How to use two-phase update for wildcard rules?" *IEEE Commun. Lett.*, vol. 19, no. 3, pp. 347–350, Mar. 2015, doi: 10.1109/lcomm.2015.2388754.

[34] D. M. Ferrazani Mattos, O. C. Muniz Bandeira Duarte, and G. Pujolle, "Reverse update: A consistent policy update scheme for software-defined networking," *IEEE Commun. Lett.*, vol. 20, no. 5, pp. 886–889, May 2016, doi: 10.1109/lcomm.2016.2546240.

[35] X. An, D. Perez-Caparros, and Q. Wei, "Consistent route update in software-defined networks," in *Proc. IEEE 3rd Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2014, pp. 84–89, doi: 10.1109/cloudnet.2014.6968973.

[36] 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. (ICDCS)*, Jun. 2017, pp. 319–327, doi: 10.1109/icdcs.2017.96.

[37] I. Maity, A. Mondal, S. Misra, and C. Mandal, "CURE: Consistent update with redundancy reduction in SDN," *IEEE Trans. Commun.*, vol. 66, no. 9, pp. 3974–3981, Sep. 2018, doi: 10.1109/tcomm.2018.2825425.

[38] V. S. Mirrokni, M. Thottan, H. Uzunalioglu, and S. Paul, "A simple polynomial time framework for reduced-path decomposition in multipath routing," in *Proc. IEEE INFOCOM*, Hong Kong, 2004, pp. 739–749, doi: 10.1109/INFCOM.2004.1354544.

[39] T. Mizrahi and Y. Moses, "Time-based updates in software defined networks," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 163–164, doi: 10.1145/2491185.2491214.

[40] 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, doi: 10.1109/tnet.2016.2529058.

[41] T. Mizrahi, O. Rottenstreich, and Y. Moses, "TimeFlip: Using timestamp-based TCAM ranges to accurately schedule network updates," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 849–863, Apr. 2017, doi: 10.1109/tnet.2016.2608441.

[42] J. Zheng, B. Li, C. Tian, K.-T. Foerster, S. Schmid, G. Chen, J. Wu, and R. Li, "Congestion-free rerouting of multiple flows in timed SDNs," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 968–981, May 2019, doi: 10.1109/jsac.2019.2906741.

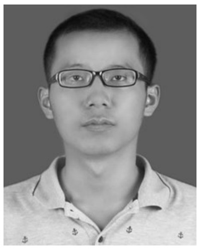
[43] T. D. Nguyen, M. Chiesa, and M. Canini, "Decentralized consistent updates in SDN," in *Proc. Symp. SDN Res. (SOSR)*, 2017, pp. 21–33, doi: 10.1145/3050220.3050224.

[44] G. Li, Y. Qian, C. Zhao, Y. R. Yang, and T. Yang, "DDP: Distributed network updates in SDN," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2018, pp. 1468–1473, doi: 10.1109/icdcs.2018.00150.

[45] 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, doi: 10.1007/s11241-016-9265-0.

[46] D. Klein and M. Jarschel, "An OpenFlow extension for the OMNeT++ INET framework," in *Proc. 6th Int. Conf. Simulation Tools Techn.*, 2013, pp. 322–329, doi: 10.4108/icst.simutools.2013.251722.

[47] M. A. Salih, J. Cosmas, and Y. Zhang, "OpenFlow 1.3 extension for OMNeT++," in *Proc. IEEE Int. Conf. Comput. Inf. Technol.*, Liverpool, U.K., Oct. 2015, pp. 1632–1637, doi: 10.1109/cit/iucc/dasc/picom.2015.246.



JUN LU was born in Jingzhou, Hubei, China. He received the M.S. degree in control engineering from the School of Electronics and Information, Northwestern Polytechnical University, Xi'an, China, in September 2014. He is currently pursuing the Ph.D. degree in communication and information system with the School of Electronic Information Engineering, Beihang University, China.

His research interests include avionics information integration, software-defined networks, and embedded systems.

HUAGANG XIONG received the Ph.D. degree in communication and information system from the School of Electronic Information Engineering, Beihang University, China, in 1998.

He is the Chief of BUAA-TTTech Time-Triggered Technology Joint Laboratory (TTTJL), Beihang University, and he is the Head of the Avionics and Bus Communications Research Team (ABC), School of Electronic Information Engineering, Beihang University. He is currently a Full Professor with Beihang University. He has published over 305 peer-reviewed articles (SCI or EI) and three books. And he has presided more than twenty major projects in total, such as national natural science foundation of China, national 863 program, and civil aircraft research. His researches are focused on communication network theory and technology, avionics integration, airborne networks, and standards.

Dr. Xiong is a member of China aviation electronics standardization committee, the Director of Beijing electronic circuit research association, a member of avionics and air traffic control branch of China Society of Aeronautics and Astronautics, and an expert of civil aircraft scientific research group.

FENG HE received the Ph.D. degree in communication and information systems from the School of Electronic Information Engineering, Beihang University, China, in 2008.

He is currently an Associate Professor with the School of Electronic Information Engineering, Beihang University. In this area, he has published over 76 peer-reviewed articles and two books. He has presided more than ten major projects in total, such as national natural science foundation of China, national 863 program, and civil aircraft research. His research interests concern digital communication technology, communication network theory and technology, avionics integration, software-defined networks, embedded systems, and real-time networks.

ZHONG ZHENG was born in Fuzhou, Jiangxi, China. He is currently pursuing the Ph.D. degree in communication and information system with the School of Electronic Information Engineering, Beihang University, China.

His main research directions include the time-triggered ethernet (TTE) design, scheduling algorithm, delay analysis, and system optimization.

HAORUO LI was born in Chengdu, Sichuan, China. He received the B.S. degree in electronic and information engineering from Beihang University, Beijing, China, in 2018, where he is currently pursuing the M.S. degree with the School of Electronic Information Engineering.

His main research directions are avionics information integration and software-defined networks.