Lv et al. Journal of Cloud Computing (2023) 12:141
https://doi.org/10.1186/s13677-023-00518-3

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Making programmable packet scheduling time-sensitive with a FIFO queue

Qianru Lv[1†], Xuyan Jiang[1†] and Xiangrui Yang[1*]

Abstract

Time-Sensitive Networking (TSN) is an emerging technology for real-time and non-real-time hybrid networked systems. TSN is standardized by IEEE 802.1 TSN Task Group and is becoming widely used in various scenarios including the cloud network. However, existing programmable packet schedulers such as PIFO, PIEO, and AIFO in programmable switches either lack the ability to express most scheduling algorithms in TSN or introduce intolerable on-chip memory overhead (e.g., strict-priority queues). This makes programmable switches and NICs incapable of providing deterministic forwarding.

In this paper, we present AIAO (Admission-In-Admission-Out), a new set of programmable scheduling primitives using just a single FIFO to support typical TSN scheduling algorithms, as well as other popular work-conserving algorithms. AIAO is inspired by AIFO but improves it with a group of high-speed packet ingress/egress admission control triggered by high-precise and globally synchronized time, thus being able to support time-sensitive scheduling. We implement AIAO and evaluate it with FPGA-based TSN switches. The preliminary results show that AIAO guarantees correctness for a typical TSN scheduling algorithm with minimal logic and memory overhead.

Keywords Time-sensitive networking, Programmable data plane, Packet scheduling, FIFO

Introduction

Today, there is an emerging need to deploy time-critical applications and non-time-critical applications on the same network infrastructure [1, 2]. Motivated by this trend, TSN is proposed and standardized by IEEE 802.1 TSN Task Group to support deterministic end-to-end communication for time-critical applications over non-deterministic Ethernet [3]. At the core of TSN is a time-precise packet scheduling mechanism on both switches and NICs. The mechanism guarantees deterministic packet forwarding for time-sensitive (TS) flows. Together with precise traffic injection control on the sender side, the time-critical packet is able to arrive at the receiver at a deterministic latency without heavy jitter. Currently, most existing packet scheduling mechanisms in TSN require globally synchronized time on each switch and time-precise packet ingress/egress control for time-sensitive packets in a non-work-conserving manner [2–5]. Non-work-conserving algorithms allow a link to be idle rather than sending an ineligible packet. Conversely, work-conserving algorithms do not let a link idle if there exists a packet waiting to be scheduled on it.

On the other side of the spectrum, programmable data plane such as RMT [6], dRMT [7] and Trio [8] has been widely adopted in many scenarios such as data centers, bringing benefits like software-defined policies, in-network computing, and network visualization into different use-cases. For instance, HPCC [9] leverages the in-band telemetry to implement fine-grained on-path congestion control. SwitchML [10] and OmniReduce [11] use the Very-Long-Instruction-Word (VLIW) Action Engine in PISA architecture [6] to support in-network parameter aggregation, boosting the

[†]Qianru Lv and Xuyan Jiang contributed equally to this work.

*Correspondence:
Xiangrui Yang
yangxiangrui11@nudt.edu.cn
[1] College of Computer, National University of Defense Technology, No.109 Deya Street, Changsha 410073, Hunan, China

training performance for distributed machine learning systems. It is reasonable that such features may also be beneficial to time-critical applications that require TSN [12].

However, supporting TSN scheduling algorithms efficiently on the programmable data plane is non-trivial. In recent years, a number of representative general packet scheduling primitives such as PIFO (Push-In-First-Out) [13], PIEO (Push-In-Extract-Out) [14] and AIFO (Admit-In-First-Out) [15] are proposed to implement various packet scheduling algorithms with user-customized scheduling program. But those primitives above are either not able to provide deterministic packet forwarding latency with reasonable jitter or too resource-heavy to be implemented onto off-the-shelf switch chips. On the other hand, schedulers like PIPO [16] are able to support the TSN scheduling algorithms above but still need to implement priority queues on chip. Making things worse, at least one queue will be left unused when TSN features are not implemented. Such inefficiency is generally intolerable for switch chip design.

In this paper, we propose a set of high-speed and programmable packet scheduling primitives named Admission-In Admission-Out (AIAO), which is designed to implement a broad spectrum of scheduling algorithms including those in TSN with minimal on-chip memory overhead. AIAO is inspired by the recently proposed programmable scheduler AIFO [15], which uses only one FIFO queue to implement complex packet schedulers but is more expressive to support time-sensitive scheduling algorithms that are necessary to ensure deterministic packet forwarding. The fundamental difference between AIAO and AIFO has two folds: On the one hand, AIAO allows the data plane to maintain the expected eligible time of the last packet in the FIFO queue so that the ingress admission control is capable of granting time-sensitive packets into the queue according to synchronized time. On the other hand, AIAO implements an egress admission control that sends out the packet from the head of the queue. This combination enables dynamic and resource-efficient packet scheduling while maintaining the flexibility to adapt to both work-conserving and non-work-conserving requirements.

The main contributions of this paper are:

1) A hardware primitive design of AIAO, which uses just one FIFO queue to support both work-conserving and non-work-conserving packet scheduling algorithms, especially for TSN.
2) A programming framework demonstrates how typical TSN schedulers such as Time-Aware Shaper [2] (TAS) and Cyclic Queuing and Forwarding [4] (CQF) can be efficiently programmed using AIAO.
3) A preliminary evaluation of AIAO on Xilinx Zynq 7020 FPGAs, demonstrating its feasibility and resource efficiency in expressing TSN scheduling algorithms.

The remainder of the paper is organized as follows: Section II presents the background on programmable packet scheduling and the motivation of AIAO. Section III describes the fundamental design of AIAO's architecture and components. Section IV details the implementation and Section V discusses the preliminary evaluation of AIAO. Finally, Section VI discusses related works and Section VII concludes the paper.

Background and motivation

In this section, we first discuss existing programmable scheduler primitives, then introduce the time-sensitive scheduling model, and finally illustrate our motivation with a simple example.

Programmable scheduler

The programmable scheduler is a key component providing software-defined scheduling abilities for programmable packet processing architectures such as PISA/RMT [6] and Trio [8]. PIFO [13] is the first among existing methods that enable such abilities.

PIFO

PIFO leverages a tree of PIFO blocks (i.e., PIFO mesh) to support work-conserving and non-work-conserving scheduling methods. Besides a PIFO queue, each block also has a ranking computing module that determines the order of packet enqueue. By the module's programmability, the PIFO mesh can determine how the packets are scheduled to the egress port. However, the drawbacks come in three folds: First, the implementation of PIFO-based schedulers requires quite an amount of SRAM [13, 15], which is quite rare and shared globally by components like hash tables and state memory on the switch. Second, PIFO requires packet reordering on the hardware queue, which so far lacks an efficient way to deliver and thus causes scalability issue with the packet processing speed's increase. Last and most importantly, PIFO-based methods only decide the order when packets enqueue, without a comprehensive control on packet dequeue, which is crucial for time-sensitive packet scheduling. To our knowledge, no off-the-shelf switch chip implements PIFO because of the reasons above.

PIEO

To overcome the scalability issue and improve programmability on the packet dequeue of PIFO [14], PIEO, on the other hand, implements a programmable

predicate-based filtering [14] for packet dequeue at the cost of lower packet rate. The filtering partially solves the third issue we discussed above but is still too expensive (it needs multiple SRAM-based FIFOs for reordering) and has scalability issues to implement.

AIFO

Unlike PIFO and PIEO, AIFO [15] reduces the on-chip SRAM overhead aggressively by consuming only a single FIFO queue for scheduling. The key insight is that up-to-date data center networks (and networks that prioritize low-latency over other metrics) prefer shallow buffer, which makes *packet drop decision* the more important metric over *packet order decision*. Thus, the AIFO-based scheduler only computes the packet rank before enqueue stage and drops the packet according to the computed rank. While the AIFO-based scheduler can express most typical scheduling algorithms at very high packet rates and low SRAM overhead, it fails to implement time-sensitive algorithms because of Head-of-Line (HoL) blocks.

Time-sensitive scheduling model

TS and non-TS flows

Unlike standard Ethernet, flows in TSN are divided into time-sensitive (TS) flows and non-TS flows according to their real-time requirements. TS flows are a stream of static, periodic packets requiring stringent end-to-end delay and ultra-low jitter, i.e., determinism. To ensure the stringent determinism, when a TS packet is transmitted from each switch along its route path is planned in advance by the planning algorithm [17]. In other words, planning algorithms synthesize TS flows' *transmission time table*. To guarantee stringent determinism, TS packets must be scheduled according to their pre-planned transmission timetable. Non-TS flows are usually dynamic and have relaxed requirements of determinism, whose transmission cannot be planned in advance.

To guarantee determinism for TS flows, TSN introduces two time-sensitive schedulers: TAS and CQF. TAS and CQF schedule frames in a time-triggered manner, i.e., schedule a specific frame at a specific time. TAS and CQF are based on a mechanism named *time gating*, as shown in Fig. 1.

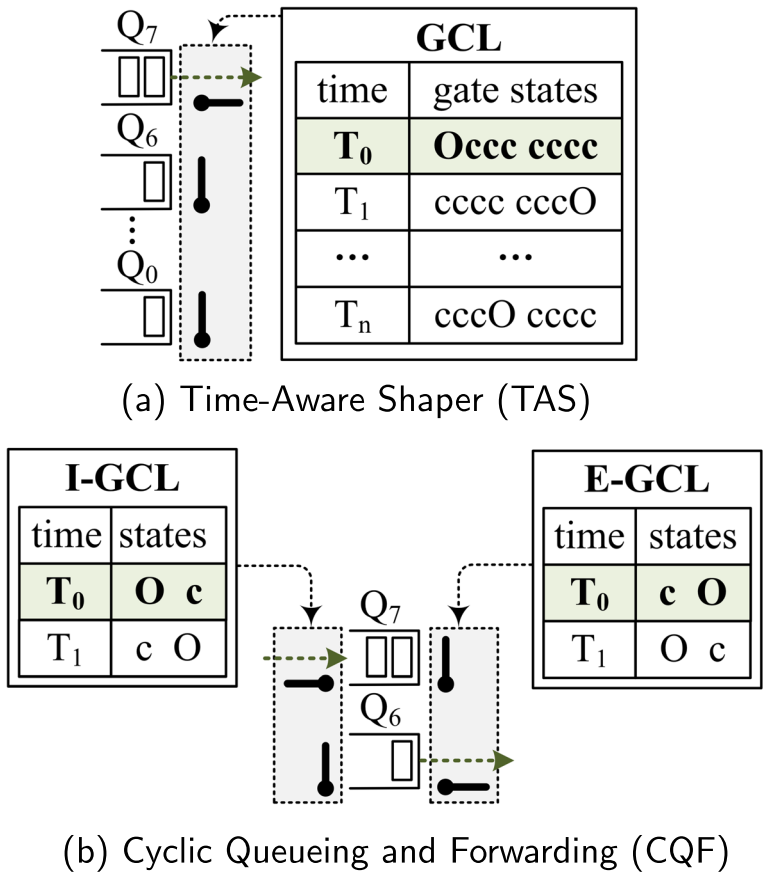


Fig. 1 Time-sensitive scheduling model

Time gating mechanism

The time gating mechanism sets gates before or after packet transmission queues. A gate can be in one of two states: open or closed. Frames can only pass a gate when it is in the open state. The time when a gate switches its state is controlled by a Gate Control List (GCL) and a globally synchronized clock. The pre-planned transmission time table computed by planning algorithms is configured into GCL. GCL works in a cyclic way. It starts with the first entry, moves to the last entry, and then rolls back to the first entry again. The global clock is achieved by the time synchronization standard, IEEE 802.1AS [1], which is beyond the scope of this paper. The detailed workflow of the two schedulers is as follows.

TAS

TAS sets egress gates after each transmission queue as shown in Fig. 1(a). The standard Ethernet often has eight transmission queues, so TAS often has eight gates. A GCL controls all eight gates. For example, at time T_0, the gate after Q_7 is open, and the other seven gates are closed according to *Occc cccc* in the GCL. The GCL in TAS is computed by traffic scheduling algorithms.

CQF

Unlike TAS, CQF set two ingress gates before two transmission queues and two egress gates after two transmission queues, as shown in Fig. 1(b). I-GCL controls two ingress gates, and E-GCL controls two egress gates. Two queues work in a ping-pong manner according to I-GCL and E-GCL. At the time T_0, the ingress gate of Q_7 is open, and egress gates of Q_6 are open. Thus frames from upstream nodes queue up in Q_7, and frames queueing in Q_6 can be transmitted to a downstream node. At time T_1, the opposite happens.

In summary, TAS and CQF use timed-gate to isolate frames in the time domain, thus enabling contention-free transmission.

Motivating example

This paper aims to design a set of primitives that require minimal on-chip SRAM resources but support both time-sensitive scheduling and other classic algorithms that PIFO/PIEO/AIFO is supposed to support.

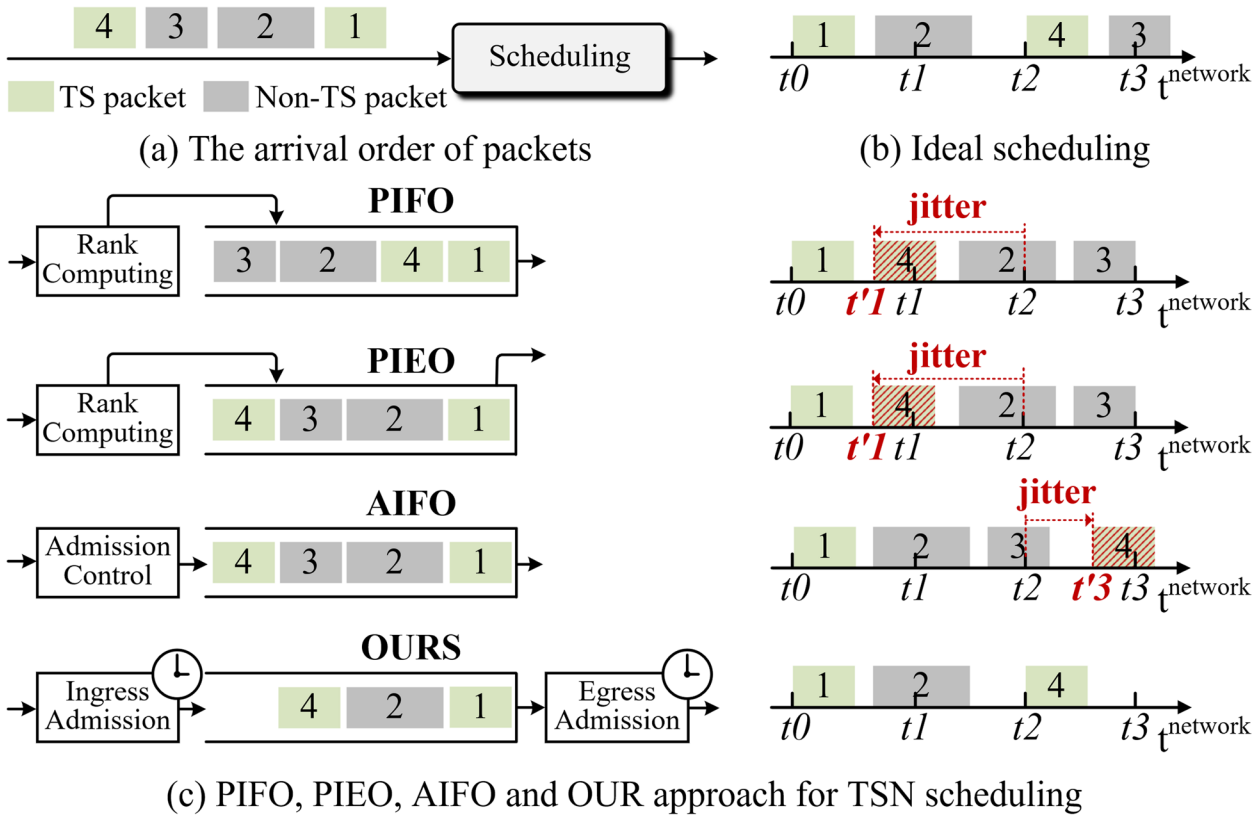


Fig. 2 Motivating time-sensitive scheduling example

To demonstrate what is missing in up-to-date programmable scheduler primitives, we examine the examples in Fig. 2. We consider separately using PIFO, PIEO, and AIFO for TS and non-TS data transfer. The arrival order of packets and the ideal scheduling are shown in Fig. 2(a) and (b). The TS packets *pkt1* and *pkt4* should be scheduled at $t0$ and $t2$. And the non-TS packets *pkt2* and *pkt3* can only occupy the remaining bandwidth.

In the PIFO and PIEO cases in Fig. 2(c), both *pkt1* and *pkt4* will be scheduled first, as time-sensitive data has a higher rank. However, because there is no time-sensitive egress control, *pkt4* experiences uncertain jitters. The scheduling time of *pkt4* is not aligned with the planned time $t2$, and is aligned with $t'1$ instead, which violates the deterministic requirements. In the AIFO case, the admission control first allows all four packets into the queue. And the scheduling order is the arrival order of packets due to its FIFO queue. When *pkt3* finishes transmission, *pkt4* has missed its planned transmission time $t2$, and is scheduled at $t'3$ instead. Thus, due to *pkt3*, *pkt4* experience queuing delays waiting for *pkt3* to be scheduled out of the pipeline, which causes *pkt4* to miss its pre-planned scheduling time $t2$. Instead, *pkt4* is scheduling at $t'3$, violating determinism. In summary, all three schedulers above fail to meet deterministic requirements as they all lack deterministic ingress/egress co-scheduling for TS packets.

Although AIFO does not support TSN features, AIFO inspires us that a single FIFO queue can approximate PIFO behaviors while using the minimum hardware resources, thus maintaining scalability. Based on this observation, we add an admission control before and after a FIFO, which is shown in the last case in Fig. 2(c). Ingress Admission decides the right set of packets to admit into the FIFO queue. Egress Admission supports time-sensitive scheduling. Ingress Admission and Egress Admission have the knowledge of the globally synchronized clock. Since *pkt3* hinders the scheduling of *pkt4*, Ingress Admission drops *pkt3* because *pkt3* is a non-TS packet so it just needs best-effort transmission. The Egress Admission controls the time when *pkt4* is scheduled. Although *pkt3* is dropped, the TS packets *pkt1* and *pkt4* can be scheduled at their pre-planned transmission time, which is the same as the ideal scheduling in Fig. 2(b).

Design

Design insights

Before diving into the details, there are two key observations regarding packet scheduling in TSN:

- *The transmission order of TS packets on a switch may not be identical to their arrival order.* Since we aim to use a single FIFO to schedule TS packets, TS packets in the queue should be arranged in strict ascending order of their transmission time. It is different from AIFO because the packets in the queue of AIFO are not necessarily arranged in strict rank order (see Fig. 3 in [15] for details). However, the arrival order of TS packets may not be the transmission order. We call it disordered problem. The disordered problem of TS packets needs to be solved when using a single FIFO. TS packets need to be cached when disordered packets exist.
- *Non-TS packets must not interfere with the scheduling of TS packets.* Non-TS packets share the same FIFO queue with TS packets, and there is no physical isolation between them. We call it shared queue problem. If a non-TS packet is admitted into the FIFO queue unreasonably, it may hinder the scheduling of subsequent TS packets. To maintain the correctness of TS scheduling, non-TS packets need to be aggressively dropped when there is only a single FIFO.

Based on the observations above, the major technical challenges we tackle are the disordered problem and the shared queue problem. Thus we present Admission-In-Admission-Out (AIAO). The key idea of AIAO is to: 1) maintain the packet order in the FIFO in the increasing order of transmission time and 2) decide whether a non-TS packet can be admitted into the FIFO. To address the challenges, AIAO proposes a programming framework consisting of three components: a FIFO queue, Ingress Admission and Egress Admission, among which Ingress Admission is the key component to solve the challenges.

Programming framework

The high-level framework of AIAO is shown in Fig. 3. And the pseudocode of AIAO is shown in Algorithm 1. AIAO has three underlying components: a primary FIFO queue, Ingress Admission and Egress Admission. The function of each component is as follows.

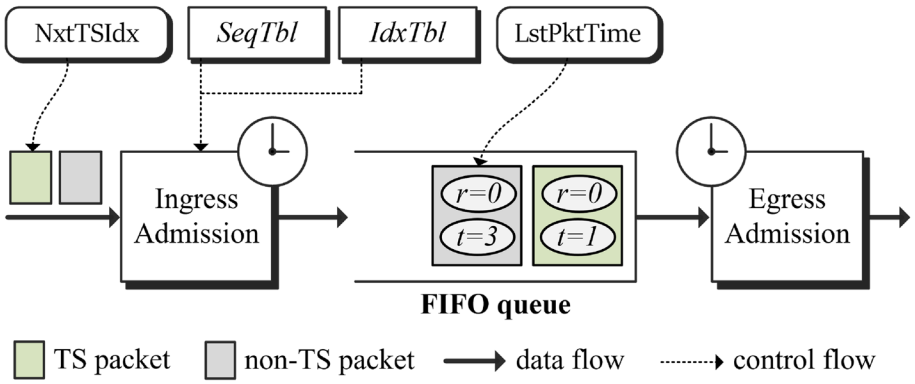


Fig. 3 AIAO framework. For convenience, t_{elig} is abbreviated as *t*, and *rank* is abbreviated as *r*

```
1  function Ingress(pkt)                                  ▷ Ingress Admission
2      pkt.rank ←getRank(pkt)
       pkt.t_elig ←getEligibleTime(pkt)
       flag, pkt' = isAdmitted(pkt)
       if flag == ADMIT then
3          Queue.enqueue(pkt)
4      else if flag == DROP then
5          Queue.drop(pkt)
6      else if flag == STORE then
7          pass
8      end
9  end function

10 function Egress()                                      ▷ Egress Admission
11     if Queue is not empty and Queue.head.t_elig ≥ t_current then
12         pkt ← Queue.dequeue()
13     end
14 end function
```

Algorithm 1 AIAO transactions

FIFO

A primary FIFO queue maintains the scheduling order for enqueued packets. A packet is enqueued into the tail of the queue and dequeued from the head of the queue.

Ingress Admission (IA)

IA is located before the FIFO queue. IA has two functions, as shown in Lines 1-12 in Algorithm 1. The first function assigns the attributes, rank and eligible time, to the packets (Lines 2-3). Each packet has five attributes in AIAO, which will be introduced later. The second function is deciding whether packets can be admitted into the FIFO queue according to *rank* and t_{elig} of the packet (Line 4-11).

To solve the disordered problem and shared queue problem, IA implements two state variables, `LstPktTime` and `NxtTSIdx`, and two tables, Index Table (abbreviated as *IdxTbl*) and Sequence Table (abbreviated as *SeqTbl*). For the disordered problem, IA uses `NxtTSIdx`, *IdxTbl* and *SeqTbl* to obtain the eligible time of TS packets within the time complexity of `O(1)`. Besides, IA uses a few registers for TS packets' reordering. For the shared queue problem, IA uses `LstPktTime` to track the transmission time of the last packet in the FIFO queue and `NxtTSIdx` to track the next TS packet to be enqueued.

IA supports three programming functions, i.e., *getEligibleTime*, *getRank* and *isAdmitted*, which could express different planning algorithms by assigning attributes to

packets with the help of state variables, tables and registers mentioned above.

Egress Admission (EA)

EA is located at the end of the queue. The function of EA is shown in Lines 13-17 in Algorithm 1. When the queue is not empty and the eligible time of the first packet in FIFO exceeds or is equal to the current network time (Line 14), EA schedules the first packet out of the queue (Line 15).

The following section introduce the AIAO programming framework from three aspects: Packet attributes, queue primitives and programming functions.

Packet attributes

Each packet in AIAO is assigned five attributes:

- *type*: This attribute indicates whether the packet is a TS or non-TS packet. Packet scheduling algorithms treat the two types of packets differently.
- *flow_id*: This attribute indicates to which flow the packet belongs.
- *period_id*: This attribute indicates that the packet is the *period_id*-th packet of a flow during the hyper period. The hyper period is the least common multiple of all flow periods. The transmission timetable for TS packets repeats over during the hyper period. In practice, such attributes can be tagged on the sender side and thus can be obtained when the packet arrives.
- *eligible_time*(t_{elig}): This attribute indicates when the packets are eligible to be transmitted. Once that time is reached, the packet should be sent immediately. Note that the network-wide global synchronization clock is used as the wall clock. The t_{elig} of TS and non-TS packets are calculated differently. The t_{elig} of TS packets is pre-calculated and static, while the t_{elig} of non-TS packets needs to be calculated dynamically based on the pre-planned t_{elig} of TS packets.
- *rank*: This attribute indicates the priority of a packet and is identical to *rank* in AIFO. The *rank* of packets is considered the same in the TSN scheduling algorithm. This attribute enables AIAO to express other non-TSN scheduling algorithms.

Queue primitives

AIFO supports five primitive operations atop the FIFO queue and registers, shown in Table 1. *enqueue(pkt)* support inserting a sequence of packets into the FIFO queue due to the disordered problem, which will be discussed in "Expressiveness" section. *store(pkt)* and *retrieve(addr)* are built atop registers that are used to cache disordered TS packets.

Programming functions

There are three programming functions that the programmers can leverage to program the AIAO scheduler to support TSN and non-TSN scheduling algorithms.

- *getEligibleTime* function: This function takes the packet of a flow to be enqueued as an argument and assigns a possible eligible time for that packet as dictated by the scheduling algorithm being programmed. The eligible time of a TS packet is calculated in advance. The eligible time of a non-TS packet is determined dynamically based on the eligible time of the last packet in the FIFO queue and the eligible time of the next TS packet to be enqueued. Two tables, *IdxTbl* and *SeqTbl*, are used to record the eligible time of TS packets. `LstPktTime` state variable is used to track the eligible time of the last packet in the FIFO queue, and `NxtTSIdx` state variable is used to track the index of the next TS packet that is to be enqueued. This index is used to look up the eligible time of this packet in the *SeqTbl*.
- *getRank* function: This function takes the packet of a flow to be enqueued as an argument and assigns *rank* to the packets as dictated by the scheduling algorithm being programmed.
- *isAdmitted* function: This function takes a packet of a flow as an argument and determines whether the packet can be admitted to the queue based on the *rank* or t_{elig} of the packet. The function has three types of return results, *ADMIT*, *STORE* and *DROP*. *DROP* means a packet cannot enter the FIFO. *ADMIT* means a packet immediately enters the FIFO. *STORE* means a TS packet is temporarily stored in a register. When AIAO is programmed to support TSN, t_{elig} is usually used to determine whether to be admitted. When AIAO is used to program non-TSN algorithms, the *rank* is often used to determine whether to be admitted.

Table 1 AIAO primitives

Primitive	Function
enqueue(pkt)	inserts a packet or a sequence of packets into the end of the FIFO queue
dequeue()	takes a packet from the head of the FIFO queue
drop(pkt)	discards a packet, preventing it from entering the FIFO queue
store(pkt)	puts a TS packet into a register and returns the register's address
retrieve(addr)	retrieves a TS packet from the register indicated by its address *addr*

Note that the possible high-level programming language of AIAO is out of the scope of this paper. However, existing network programming languages such as P4 [18] and Domino [19] can be used to program a scheduler as needed in AIAO. We leave the details of such a programming language for future work.

Expressiveness

In order to demonstrate the expressiveness of AIAO, we first present the detailed expression of the most representative TSN schedulers, i.e., TAS and CQF. Then we show how AIAO could support non-TSN scheduling algorithms.

1) Expressing TAS. TAS algorithm using AIAO is shown in Algorithm 2. TAS calculates t_{elig} of TS and non-TS packets differently (Lines 2 and 5). The t_{elig} of TS packets are planned by planning algorithms in advance [17], which is recorded by *IdxTbl* and *SeqTbl*. *IdxTbl* is used to obtain the index of a packet in *SeqTbl* within the time complexity of `O(1)` (Line 3), and *SeqTbl* is used to obtain the t_{elig} of the packet within the time complexity of `O(1)` (Line 4).

```
1  function getEligibleTime(pkt)
2      if pkt.type is TS then
3          index←LookupIdxTbl(pkt)
            pkt.t_elig ←LookupSeqTbl(pkt)
4      else
5          pkt.t_elig ←LstPktTime
6      end
7  end

8  function getRank(pkt)
9      pkt.rank ←0
10 end

11 function isAdmitted(pkt)
12     if pkt.type is TS then
13         index←LookupIdxTbl(pkt)
            if index ≠ NxtTSIdx then
14             addr←store(pkt)
                Update addr into SeqTbl
                return STORE
15         else if index == NxtTSIdx then
16             Update SeqTbl, LstPktTime, NxtTSIdx
                pkt_set = []
                while addr in SeqTbl[NxtTSIdx] do
17                 pkt←retrieve(addr)
                    add pkt into pkt_set
18             end
19             return ADMIT, pkt_set
20         end
21     else if pkt.type is non-TS then
22         next_TS_time ← time in SeqTbl[NxtTSIdx]
            if pkt.t_eligible + size/speed ≤ next_TS_time and no TS packets in the
            registers then
23             return ADMIT, pkt
24         else
25             return DROP
26         end
27     end
28 end
```

Algorithm 2 TAS algorithm using AIAO

The t_{elig} of a non-TS packet depends on the eligible time of the last packet in the FIFO queue. When a non-TS packet arrives, its earliest possible eligible time is when the last packet in the FIFO queue finishes its transmission. The earliest possible eligible time is tracked by `LstPktTime` state variable. Thus the t_{elig} of a BE packet is identical to `LstPktTime` (Line 6).

Since TAS schedules packets according to their eligible time, the *rank* of each packet could be set to be the same, which is 0 in this paper (Line 10).

The implementation of *isAdmitted* functions is non-trivial. In general, all TS packets are always admitted, and non-TS packets are not admitted if they interfere with the scheduling of TS packets, i.e., the shared queue problem. Besides, the disordered problem of TS packets should be handled carefully.

When a TS packet arrives (Line 13), if the index of this packet in the *SeqTbl* (Line 14) is not identical to `NxtTSIdx` state variable (Line 15), it means this packet is not the packet being waited for, but the packet was sent after the packet was waited for, which is the disordered problem. Thus the current packet enters the register, and the address of this register is returned (Line 16). The returned address needs to be updated in the address field of the corresponding table entry in *SeqTbl* (Line 17). The packet is temporarily stored in the register. Thus *STORE* is returned (Line 18). If the current packet is the TS packet being waited for (Line 19), there is no disordered problem. The tables and state variables need to be updated (Line 20). First, the address field of the packet in *SeqTbl* is reset to null to indicate that the packet does not need to be buffered. Second, the TS packet is now the last packet in the FIFO queue and then the t_{elig} of this packet is updated to `LstPktTime`. Third, `NxtTSIdx` should be updated to the current index+1 (Line 14). If the packets corresponding to the new `NxtTSIdx` have arrived in advance, the packets are taken out from their register, and the update operation is repeated (Lines 16-18). All the TS packets are admitted into the FIFO queue, and *ADMIT* is returned (Line 26).

When a BE packet arrives (Line 28), if `LstPktTime` plus its transmission delay does not exceed the t_{elig} of the next TS packet, and there are no TS packets in the registers (Line 30), it means that this BE packet will not interfere with the scheduling of the next TS packet, then the BE packet is admitted into the FIFO queue (Line 31). Otherwise, the BE packet is dropped (Line 33).

2) Expressing CQF. CQF algorithm using AIAO is shown in Algorithm 3. Like TAS, CQF assigns the t_{elig} of TS and non-TS packets differently (Lines 2 and 4). In CQF, the t_{elig} of TS packets is also planned in advance. The t_{elig} of TS packets is recorded by *IdxTbl* only. Because there is no disordered problem in CQF, so there is no need to use *SeqTbl*. Thus CQF can obtain the t_{elig} of packets from *IdxTbl* within the time complexity of `O(1)` directly(Line 3). Note that the t_{elig} of multiple TS packets sent in the same time slot is when the time slot starts. The t_{elig} of multiple TS packets is identical. This does not affect the egress scheduling of the FIFO queue. A non-TS packet's t_{elig} is tracked by `LstPktTime` (Line 5). Similarly, the rank of packets in CQF is identical and thus is set to be 0 (Line 9).

```
1  function getEligibleTime(pkt)
2      if pkt.type is TS then
3          pkt.t_eligible ←LookupIdxTbl(pkt)
4      else if pkt.type is non-TS then
5          pkt.t_eligible←LstPktTime
6      end
7  end
8  function getRank(pkt)
9      pkt.rank ←0
10 end
11 function isAdmitted(pkt)
12     if pkt.type is TS then
13         update LstPktTime
           return ADMIT, pkt
14     else if pkt.type is non-TS then
15         if pkt.t_eligible + size/speed ≤ NxtTSIdx then
16             update LstPktTime
               return ADMIT, pkt
17         else
18             return DROP
19         end
20     end
21 end
```

Algorithm 3 CQF algorithm using AIAO

Since TS packets in CQF do not have the disordered problem, TS packets need not be reordered before being admitted into the FIFO queue. As a result, when a TS packet arrives, it is directly admitted into the FIFO queue (Line 14). Besides, `LstPktTime` should add the transmission delay of the TS packet(Line 13).

For non-TS packets, since the eligible transmission time of TS packets is a coarse-grained time slot (the typical value of the time slot is usually over 100μs), the time at the back of a time slot that is not used by

TS packets can be used for the scheduling of non-TS packets. Whether a non-TS packet can enter the FIFO queue depends on whether the non-TS packet affects the scheduling of TS packets in the next time slot. The moment when the next time slot starts is taken from `NxtTSIdx` (Line 21). Note that `NxtTSIdx` in CQF has a different meaning in TAS. If the `LstPktTime` plus the transmission delay of this non-TS packet does not exceed `NxtTSIdx`, the non-TS packet is admitted (Line 18). Otherwise, it is dropped (Line 20). Besides, `NxtTSIdx` needs to be updated to the moment when the transmission of the packet ends (Line 17). Note that for each increase in the length of the slot in the global clock, `NxtTSIdx` automatically adds 1.

3) Expressing non-TSN scheduler. AIAO can also express non-TSN scheduling algorithms. Take the shortest remaining processing time (SRPT) that AIFO can express as an example. AIAO can achieve the same effect as AIFO. The principle of SRPT is that flows with the shortest remaining bytes of flows are scheduled first. Thus the remaining bytes of flows can be used as *rank* that can be implemented in *getRank* function. Packets whose ranks are larger than a certain rank are dropped, and those smaller than a certain rank enter the FIFO queue, which can be implemented in the *isAdmitted* function. The t_{elig} of packets is set to be 0, which can be implemented in the *getEligibleTime* function. Once the FIFO queue is not empty, the packets can be scheduled immediately. In summary, AIAO can also express algorithms other than TSN scheduling algorithms.

Implementation

This section briefly presents how AIAO can be implemented on the hardware. As shown in Fig. 4, the fundamental feature of AIAO is that it only contains a single FIFO queue for packet scheduling. To support time-sensitive algorithms, high-precision global time via 802.1AS [1] is provided to both IA and EA modules to issue time-sensitive decisions. Each packet is assigned a unique description containing the attributes mentioned above: *type*, *flow_id*, *period_id*, *eligible_time*, and *rank*.

Next, we present how the scheduling decision is issued and executed by AIAO. In the first step (①), a packet descriptor is issued to the scheduler when a packet arrives at the packet processing pipeline. Then AIAO reads the packet descriptor by the order they enter the pipeline and places the descriptor in a pre-queue cache according to the packet type attribute (the second step ②). For non-TS packets, the descriptor will be fed directly to the Rank Selector, waiting to enqueue or to be dropped by the Rank Selector. Because TS packets may arrive disordered, a Reordering Module is implemented before an enqueue operation is issued. Specifically, when the packet descriptor of a TS packet arrives, the *flow_id* and *period_id* are matched against the *IdxTbl* to obtain the time index, which indicates the packet order in each TS flow. Based on this index, the Reordering Module can reorder the TS packets before the enqueuing operation.

In the third step(③), for TS packets, the larger value of the time index and the `NxtTSIdx` is matched against the *SeqTbl* to get the eligible time for the next TS packet

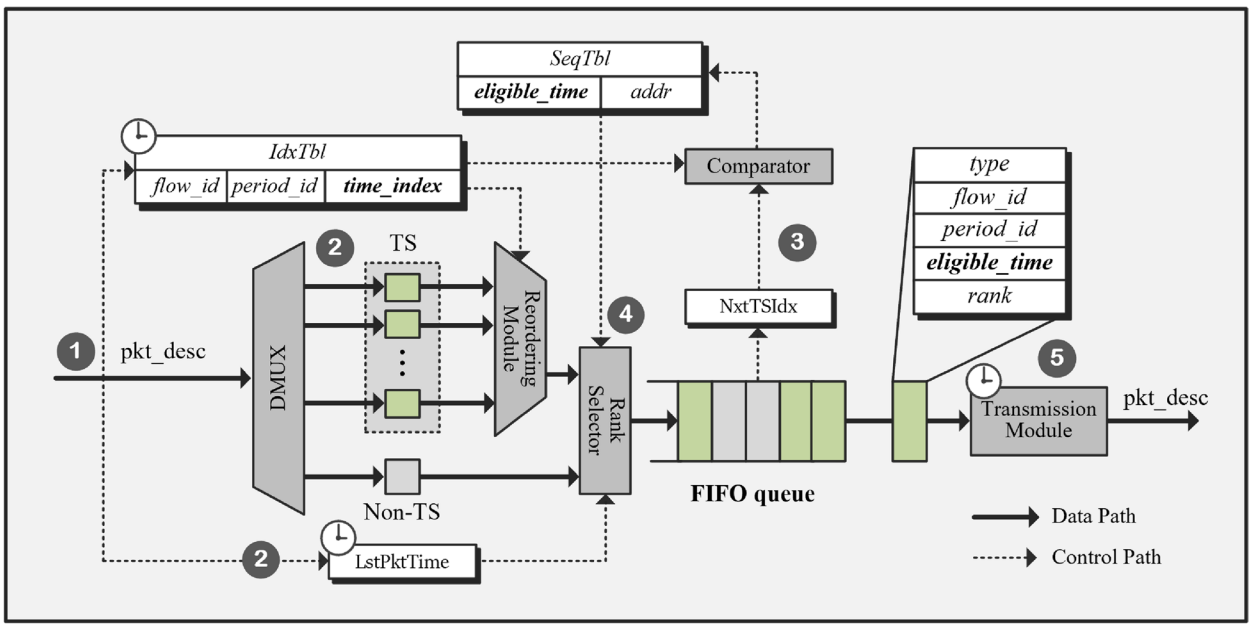


Fig. 4 AIAO Primitives Implementation

that is expected to enqueue. The Rank Selector is the key scheduling module in the ingress stage. It reads the register `LstPktTime`, which indicates when the last packet in the FIFO queue is expected to finish the dequeue operation. Then it calculates if the arriving packet should enqueue by comparing the eligible time of the next TS packet and the expected dequeue time of the current non-TS packet (④). Next, the Rank Selector grants the next packet based on the result and issues the enqueue operation. Finally, both `NxtTSIdx` and `LstPktTime` are updated accordingly. In the last step(⑤), Transmission Module issues a dequeue operation when the current time exceeds the eligible time of the packet on the head of the queue.

As all the tables, the reordering registers and the FIFO queue are implemented using SRAM on-chip, AIAO can run at reasonably high frequency in a deep-pipelined manner. We left the discussion of the size of the FIFO queue, *IdxTbl*, and *SeqTbl* for a future expended version because of the space limitation.

Preliminary results

At the time just before the submission, we implemented an early-version prototype (within an open-source TSN switch project) of AIAO on Xilinx Zynq 7020 (125Mhz) and evaluate it on a ring topology with 6 nodes. We briefly demonstrate the resource usage and feasibility of AIAO in this section.

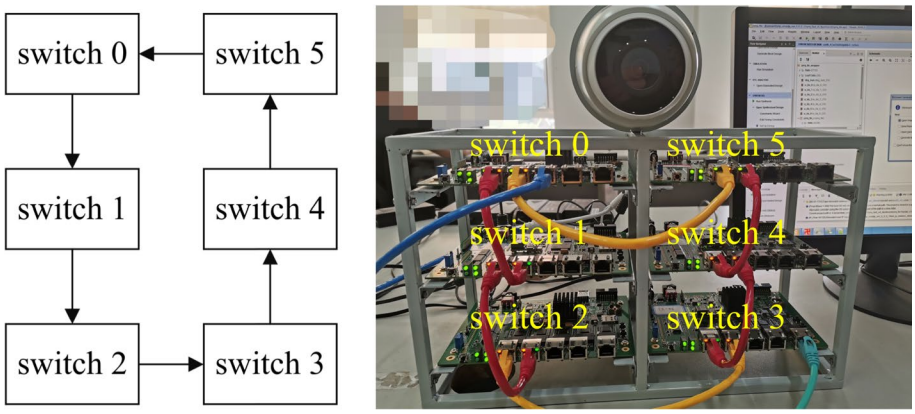
Table 2 Resources usage of a TSN switch when implementing standard CQF and AIAO-based CQF

Hardware Implementation	Slice LUTs	Block RAMs
Standard CQF	25536 (48.00%)	82.50 (58.93%)
AIAO-based CQF	26419 (49.66%)	71.05 (50.75%)

Firstly, we compare the FPGA resource usage between the TSN switch with a standard CQF scheduler and the TSN switch with an AIAO-based CQF scheduler. The result is shown in Table 2. Though the AIAO-based scheduler consumes slightly more (1.6%) LUT than the baseline, it reduces the BRAM usage by over 11 36k BRAM (8%). Such reduction is possible using one FIFO queue instead of eight in standard CQF. The resource usage comparison between AIAO and other scheduler primitives, such as PIFO and PIEO, is still ongoing.

Next, we demonstrate whether AIAO is programmable enough to support typical TSN scheduling algorithms. Thus, we focus on the end-to-end delay of TS streams in the preliminary experimental results. Note that current results can only prove the feasibility of AIAO. Other experiments, such as stress testing with non-TS flows and comparison with other scheduling primitives, are undergoing.

We run the experiment on a ring topology with 6 switches as shown in Fig. 5. The output ports of the switches are equipped with AIAO. The period of the TS flow is $16777.216\mu s$, which means the minimum interval between two consecutive packets is $16777.216\mu s$, and the packet size is 512 bytes. The size of a time slot in CQF configuration is $131.072\mu s$. When CQF scheduling is used, the end-to-end delay d of a TS flow is between $(h-1)\times slot \le d \le (h+1)\times slot$, where h is the number of hops of the flow and $slot$ is the size of a time slot. $(h-1)\times slot$ and $(h+1)\times slot$ are the TS flow's theoretical lower and upper bound. We set up four experiments, each with a different number of hops for the flow. The four experiments send the flow from switch 0 to switch 2, switch 3, switch 4 and switch 5. That is, the



(a) Topology (b) Topology demo

Fig. 5 Experimental topology

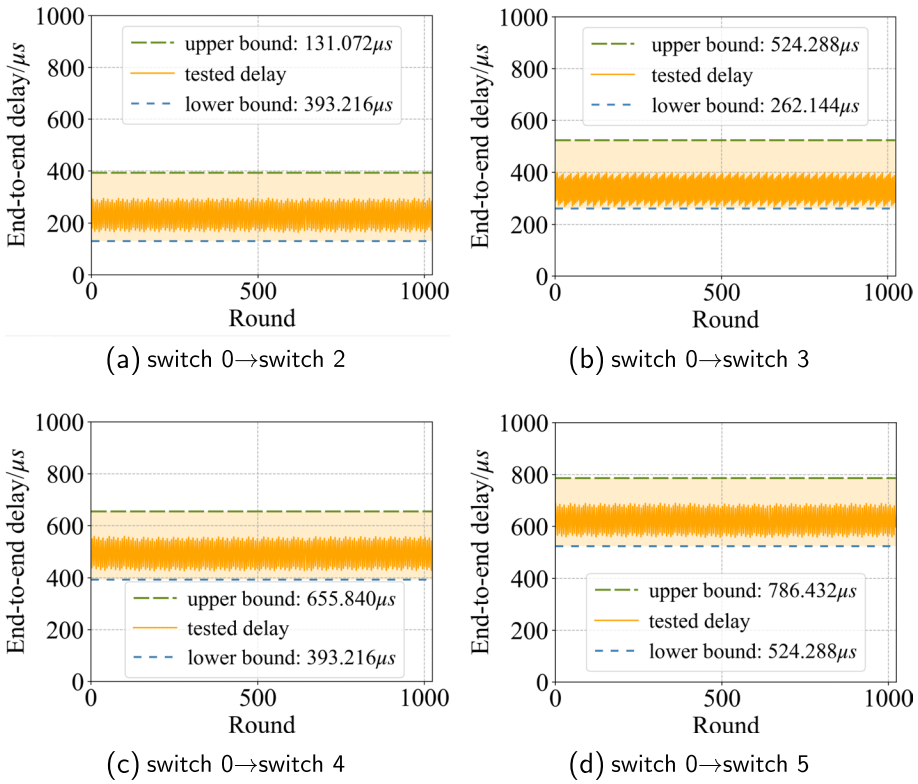


Fig. 6 The end-to-end delay in CQF

number of hops is increased from 2 to 5. We collect the end-to-end delay of this TS flow during 1024 periods/rounds. The results are shown in Fig. 6.

As the number of hops increases, the end-to-end delay of the TS flow increases. However, its end-to-end delay is always within the theoretical upper and lower bounds. For example, the hop number of the flow in Fig. 6(b) is 3, so the lower bound of delay is $(3-1) \times 131.072 = 262.144\mu s$ and the upper bound of delay is $(3+1) \times 131.072 = 524.288\mu s$. The end-to-end delay of this flow always fluctuates within the upper and lower bounds. The results show that it is feasible to express CQF with AIAO. Compared to the original CQF scheduler in TSN, AIAO uses only a single FIFO queue.

Related work

Programmable packet scheduling. Programmable packet scheduling is the opposite of fix-function packet scheduling. This field starts with PIFO [13]. PIFO is a popular packet scheduling primitive, which allows a packet to be pushed into an arbitrary queue position and dequeued only from the head of the queue. At the same time, the scalability of PIFO is extremely limited. SP-PIFO [20] and AIFO [15] are two primitives which aim to approximate PIFO. PIEO is another scheduling primitive, but it cannot be pipelined [16]. However, those primitives cannot support the deterministic packet forwarding required by TSN scheduling algorithms. As far as we know, PIPO [16] is the only scheduling primitive that supports TSN features but consumes several priority queues. Besides, when PIPO is not used to express the TSN scheduling algorithm, it has at least one idle priority queue. Such inefficiency is intolerable for chip designs, especially since priority queues are such critical resources [15].

TSN scheduling. There are two features in TSN packet scheduling. First, flows are divided according to their quality of service requirements, among which TS flows require the most stringent determinism. Second, TSN introduces the network-wide global synchronization clock [1], and the scheduling of TS flows should refer to the global synchronization time. Typical TSN scheduling algorithms include TAS [2], CQF [4], CBS [21], ATS [5], etc. Among them, TAS and CQF can ensure strict determinism for TS traffic, which is the most significant feature that distinguishes TSN from traditional Ethernet scheduling. This paper mainly focuses on the expression of these two algorithms.

Conclusion

The paper presents AIAO (Admission-In-Admission-Out), a new set of programmable scheduler primitives using a single FIFO to support typical TSN schedulers standardized in IEEE 802.1Q and other popular work-conserving algorithms. AIAO is inspired by AIFO but improves AIFO by a group of high-speed packet ingress/egress admission control triggered by high-precise and globally synchronized time, thus supporting time-sensitive scheduling. This paper discusses AIAO and evaluates it with FPGAs. The preliminary results demonstrate that it guarantees correctness for a typical TSN scheduling algorithm with limited logic and memory overhead.

Authors' contributions

Q.L and X.J wrote the main manuscript text and conducted the experiments and collected the data of the manuscripts. X.Y proposed the basic idea of AIAO, prepared figure 2/4 and helped to implement AIAO on the FPGA-based TSN switch. All authors reviewed the manuscript.

Funding

All authors are funded by National University of Defense Technology.

Availability of data and materials

Not applicable.

Declarations

Ethics approval and consent to participate

Not applicable.

Competing interests

The authors declare no competing interests.

Received: 23 August 2023 Accepted: 13 September 2023
Published online: 09 October 2023

References

1. 802.1AS-2020 - IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications. https://standards.ieee.org/ieee/802.1AS/7121/. Accessed 26 Sept 2023
2. 802.1Qbv-2015 - IEEE Standard for Local and Metropolitan Area Networks - Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic. https://standards.ieee.org/ieee/802.1Qbv/6068/. Accessed 26 Sept 2023
3. Dürr F, Nayak NG (2016) No-wait packet scheduling for IEEE Time-Sensitive Networks (TSN). In: Proceedings of the ACM RTNS conference, Association for Computing Machinery, New York, NY, USA, pp 203–212
4. 802.1Qch-2017: IEEE Standard for Local and Metropolitan Area Networks - Bridges and Bridged Networks - Amendment 29: Cyclic Queuing and Forwarding. https://standards.ieee.org/ieee/802.1Qch/6072/. Accessed 26 Sept 2023
5. 802.1Qcr-2020: IEEE Standard for Local and Metropolitan Area Networks - Bridges and Bridged Networks - Amendment: Asynchronous Traffic Shaping. https://standards.ieee.org/ieee/802.1Qcr/7420/. Accessed 26 Sept 2023
6. Bosshart P, Gibb G, Kim HS, Varghese G, McKeown N, Izzard M, Mujica F, Horowitz M (2013) Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN. In: Proceedings of the ACM SIGCOMM conference, Association for Computing Machinery, New York, NY, USA, pp 99–110
7. Chole S, Fingerhut A, Ma S, Sivaraman A, Vargaftik S, Berger A, Mendelson G, Alizadeh M, Chuang ST, Keslassy I, Orda A, Edsall T (2017) dRMT: Disaggregated Programmable Switching. In: Proceedings of the ACM SIGCOMM conference, Association for Computing Machinery, New York, NY, USA, pp 1–14
8. Yang M, Baban A, Kugel V, Libby J, Mackie S, Kananda SSR, Wu CH, Ghobadi M (2022) Using Trio: Juniper networks' programmable chipset for emerging in-network applications. In: Proceedings of the ACM SIGCOMM conference, Association for Computing Machinery, New York, NY, USA, pp 633–648
9. Li Y, Miao R, Liu HH, Zhuang Y, Feng F, Tang L, Cao Z, Zhang M, Kelly F, Alizadeh M, Yu M (2019) HPCC: High Precision Congestion Control. In: Proceedings of the ACM SIGCOMM conference, Association for Computing Machinery, New York, NY, USA, pp 44–58
10. Sapio A, Canini M, Ho CY, Nelson J, Kalnis P, Kim C, Krishnamurthy A, Moshref M, Ports DRK, Richtarik P (2021) Scaling Distributed Machine Learning with In-Network Aggregation. In: Proceedings of the USENIX NSDI conference, USENIX Association, Santa Clara, USA, pp 785–808
11. Fei J, Ho CY, Sahu AN, Canini M, Sapio A (2021) Efficient sparse collective communication and its application to accelerate distributed deep learning. In: Proceedings of the ACM SIGCOMM conference, Association for Computing Machinery, New York, NY, USA, pp 676–691
12. Yang X, Li C, Yang L, Han C, Li T, Sun Z (2021) Cames: enabling centralized automotive embedded systems with Time-Sensitive Network. In: Proceedings of the ACM SIGCOMM Poster and Demo Sessions, Association for Computing Machinery, New York, NY, USA, pp 85–87
13. Sivaraman A, Subramanian S, Alizadeh M, Chole S, Chuang ST, Agrawal A, Balakrishnan H, Edsall T, Katti S, McKeown N (2016) Programmable Packet Scheduling at Line Rate. In: Proceedings of the ACM SIGCOMM conference, Association for Computing Machinery, New York, NY, USA, pp 44–57
14. Shrivastav V (2019) Fast, scalable, and programmable packet scheduler in hardware. In: Proceedings of the ACM SIGCOMM conference, Association for Computing Machinery, New York, NY, USA, pp 367–379
15. Yu Z, Hu C, Wu J, Sun X, Braverman V, Chowdhury M, Liu Z, Jin X (2021) Programmable packet scheduling with a single queue. In: Proceedings of the ACM SIGCOMM conference, Association for Computing Machinery, New York, NY, USA, pp 179–193
16. Zhang C, Chen Z, Song H, Yao R, Xu Y, Wang Y, Miao J, Liu B (2021) PIPO: Efficient programmable scheduling for Time Sensitive Networking. In: Proceedings of the IEEE ICNP conference, IEEE, Piscataway, NJ, USA, pp 1–11
17. Yan J, Quan W, Jiang X, Sun Z (2020) Injection time planning: Making CQF practical in Time-Sensitive Networking. In: Proceedings of the IEEE INFOCOM conference, IEEE, Piscataway, NJ, USA, pp 616–625
18. P4-16 Language Specification. https://p4.org/p4-spec/docs/P4-16-v1.2.1.html. Accessed 26 Sept 2023
19. Sivaraman A, Cheung A, Budiu M, Kim C, Alizadeh M, Balakrishnan H, Varghese G, McKeown N, Licking S (2016) Packet transactions: High-level programming for line-rate switches. In: Proceedings of the ACM SIGCOMM conference, Association for Computing Machinery, New York, NY, USA, pp 15–28
20. Alcoz AG, Dietmüler A, Vanbever L (2020) SP-PIFO: Approximating Push-In First-Out behaviors using Strict-Priority queues. In: Proceedings of the USENIX NSDI conference. USENIX Association, Santa Clara, USA, pp 59–76
21. 802.1Qav-2009 - IEEE Standard for Local and metropolitan area networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams. https://standards.ieee.org/ieee/802.1Qav/4401/. Accessed 26 Sept 2023

Publisher's Note