Noname manuscript No.
(will be inserted by the editor)

Packet-size Aware Scheduling Algorithms in Guard band for Time Sensitive Networking

Chuwen Zhang · Yi Wang · Ruyi Yao · Boyang Zhou · Liang Cheng · Yang Xu · Xiaoguang Li · Jian Cheng · Bin Liu

the date of receipt and acceptance should be inserted later

Abstract As an emerging and promising technology, Time Sensitive Networking (TSN) can be widely used in many real-time systems such as Industrial Internet of Things (IIoT) and Cyber Physical System (CFS). TSN, while ensuring the bounded latency and jitter, also exhibits the disadvantage of not being able to effectively use the bandwidth resources in the guard band. This is because the past studies put emphasized efforts on pursuing the deterministic packet forwarding for time-sensitive data traffic, but they overlooked the issue of low guard band utilization. In this paper, we propose an algorithm family named Packet-size Aware Shaping (PAS), which is inspired by abstracting the problem of utilizing the guard band to a classic Precedence-Constrained Knapsack Problem (PCKP). PAS works with the existing TSN standards, having achieved the goal of guaranteeing the end-to-end latency for scheduled time-sensitive applications while fully utilizing the available bandwidth in the guard band for others. Further, we have proposed and implemented several hardware designs for both the current standard TSN scheduler and the programmable one. The simulation results show that PAS family can achieve satisfying performance in maximizing the resource utilization in guard band. The synthesis results on Xilinx Vivado show that our proposed Multi-group Push-In-First-Out (MPIFO) scheduler can achieve 100 Mpps scheduling rate for 1024 scheduling items, which is fast enough to purchase the high-speed TSN.

Keywords TSN · programmable packet scheduling · guard band · PCKP

Chuwen Zhang · Bin Liu
Department of Computer Science and Technology, Tsinghua University, Beijing, China
Tel.: +86-13621164840
E-mail: chuwen1992@gmail.com;lmyujie@gmail.com

Yi Wang · Xiaoguang Li · Jian Cheng · Bin Liu
Peng Cheng Lab, Shenzhen, China

Yi Wang
Southern University of Science and Technology, ShenZhen, China;

Ruyi Yao · Yang Xu
Fudan University, Shanghai, China

Boyang Zhou · Liang Cheng
Lehigh University, Bethlehem, USA

1 Introduction

With the emergence of Industrial Internet of Things (IIoT) and Cyber Physical System (CFS), ultra low end-to-end latency and jitter (e.g., a few microseconds) need to be guaranteed for time-sensitive traffic. The current Ethernet technology can provide end-to-end communication with high bandwidth up to tens or even hundreds of Gbps, but cannot guarantee the low latency due to its best-effort and time-insensitive nature. On the other hand, although some specialized real-time communication technologies can guarantee low transmission lantency and jitter, the network communication bandwidth they can provide is very limited. For example, the communication protocol CAN (International Organization for Standardization 2003) and FlexRay (International Organization for Standardization 2013) are operating at 0.5 and 10 Mbps, far behind the increasing bandwidth demands of existing industrial applications. Faced with this situation, several time-synchronized Ethernet-based communication technologies have been introduced, such as EtherCAT (Jansen and Buttner 2004) and TTEthernet (Kopetz and Bauer 2003; Kopetz et al. 2005). However, they cannot achieve flexible and efficient traffic scheduling for time-sensitive and best-effort traffic simultaneously within IEEE 802.3 networks, which is exactly the demand of more and more industrial applications such as industrial automation and automotive self-driving. To address these issues, Time Sensitive Networking (TSN), therefore, is proposed to achieve ultra low end-to-end latency and jitter for the time-sensitive traffic while providing the maximum possible bandwidth for the best-effort traffic.

TSN is composed of a number of IEEE standards that are still evolving and expanding. Among them, the Time Aware Shaper (TAS) in IEEE Std 802.1Qbv (LAN/MAN Standards Committee 2016c) plays an important role in realizing deterministic forwarding for scheduled traffic, a class of time-sensitive traffic that has a fixed cycle and predictable size. This shaper isolates scheduled traffic from others by an exclusive time window on the base of globally synchronized time from IEEE Std 802.1AS (LAN/MAN Standards Committee 2011), resulting in the transmission of scheduled traffic not being preempted by other traffic. It introduces a guard band (detailed in Section 2) to ensure that the link is available for transmitting the scheduled traffic at the start of its time window. However, except for the packets that are already being transmitted, no other packets will be scheduled in the guard band, even if the guard band is not fully filled, which will cause the remaining band wasted.

As a guard band is reserved as large as the transmission time of the largest Ethernet frame[1], the maximum bandwidth waste in guard band could be $\frac{f \cdot S_{max}}{l}$, where f denotes the window frequency, S_{max} the size of the largest Ethernet frame, and l the link rate. When the window frequency is high and the link rate is low, the waste on the guard band will increase. For example, if there are 100 sensors, each of them sends a packet every 10 ms, and the link rate is 1 Gbps, up to 12% bandwidth will be wasted in the guard band. Further, if the TSN LAN adopts Jumbo frame for efficiency, whose payload is up to 9000 bytes, the theoretical maximum waste will go up to 72% in the above settings!

[1] In this paper, the term of packet actually refers to the Ethernet frame, but we do not distinguish between them, and use them interchangeably.

To reduce the bandwidth waste in the guard band, frame preemption in IEEE Std 802.1br (LAN/MAN Standards Committee 2016a) and 802.1Qbu (LAN/MAN Standards Committee 2016b) is employed, which allows a low-priority packet that is being transmitted to be preempted by a high-priority packet. After the high-priority packet has finished its transmission, the rest of the low-priority packet will resume sending. This, however, depends on the dedicated MAC layer with a complex hardware structure that partitions and assembles a packet on a sender and receiver, respectively.

In this paper, we propose a family of Packet-size Aware Scheduling (PAS) algorithms to make full use of the guard band. The genealogy of the PAS family is shown in Fig. 1. PAS family tries to select eligible packets to fill the remaining band repeatedly until the next scheduled traffic window comes. PAS functions as an auxiliary measure to work with the basic scheduling algorithm such as the Strict Priority Scheduling (SPS), and takes action only in the guard band. Selecting packets to fill the remaining band can be modeled as a Precedence-Constrained Knapsack Problem (PCKP), which is an NP-complete problem. In this paper, we have proposed two solutions: PAS_I and PAS_G. The former tries to get the optimal solution, but it fails running fast enough at line rate, while the latter is a simple but fast greedy algorithm. We design the Binary-Comparator-Tree (BCT) based and Push-in-First-out (PIFO) based structures for the standard TSN scheduler to realize PAS_G. However, as PAS_G needs to traverse N scheduling items to find the best one, it cannot adapt to a programmable scheduler supporting hundreds of per-flow queues. We then further develop PAS_G(M) to approximate PAS_G, where M denotes the number of packet-size based PIFO queues, and design a Multi-group PIFO (MPIFO) scheduler accordingly.

We have made the following major contributions.

- We propose PAS family algorithms to utilize the bandwidth in guard bands more efficiently. We model PAS as a PCKP, and then raise a dynamic programming based algorithm PAS_I and greedy strategy based PAS_G to schedule packets in guard band.
- We design the BCT-based and PIFO-based structures for the standard TSN scheduler with eight output queues according to realize PAS_G. We also design the MPIFO structure for a programmable TSN scheduler according to PAS_G(M), which can support hundreds of per-flow queues at line rate.
- We conduct extensive simulations to evaluate the performance of our PAS family algorithms. Experimental results show that PAS(M) has ever-increased performance as M grows. We prototype the three scheduler designs on FPGA, and the synthesis results on Xilinx Vivado show that MPIFO achieves 100 Mpps scheduling rate when the number of scheduling items is 1024, more faster than the others.

The rest of this paper is organized as follows. Section 2 introduces the background of TSN standards related to packet scheduling. Section 3 proposes PAS concept, models it as a PCKP, and presents PAS_I and PAS_G. Section 4 describes two hardware designs based on PAS_G for the current standard TSN switch and MPIFO based on PAS_G(M) for the programmable TSN scheduler. Section 5 evaluates the performance of PAS family by simulations and implementations. Section 6 surveys the related work on packet scheduling. Finally, Section 7 concludes this paper.

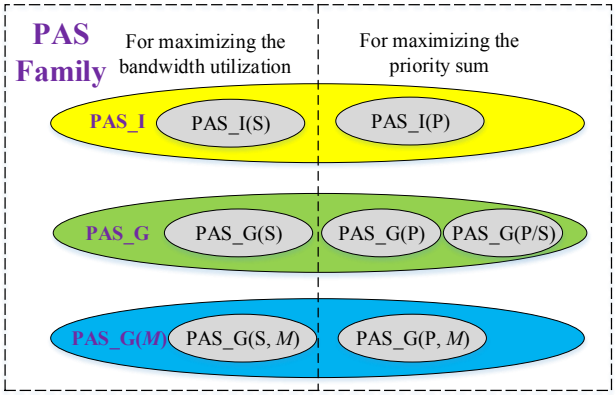


Fig. 1: PAS family is composed of three sub-classes: PAS_I provides an optimal solution based on the dynamic programming; PAS_G provides a local optimal solution based on the greedy strategy on per-flow queues; PAS_G(M) provides a local optimal solution based on the greedy strategy on the M packet-size based queues. According to the optimization objective, we further develop specific algorithms in each sub-class.

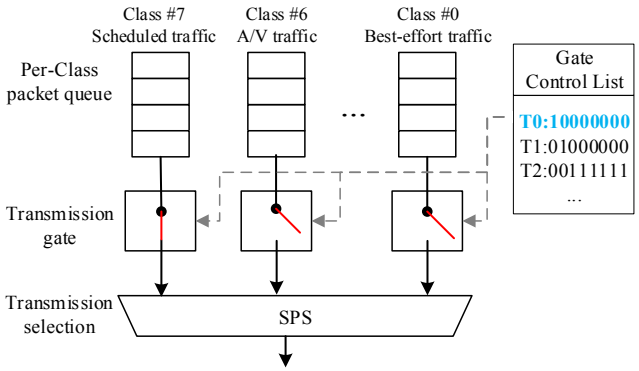


Fig. 2: Structure of the standard TSN scheduler with TAS.

2 Background

Existing TSN standards mainly focus on two types of time-sensitive traffic, i.e., scheduled traffic and Audio/Video (A/V) traffic. The former has a fixed cycle, predictable size, and rigid requirement on the latency and jitter, while the latter is usually bursty with a bulk of packets and has a requirement on bandwidth. As TSN aims to realize the harmonious coexistence of best-effort, scheduled, and A/V traffic, it should has a sophisticated packet scheduler to differentiate and schedule network traffic at line rate. Generally, a packet scheduler specifies when and in what order the queued packets should be transmitted according to its configured scheduling algorithm. To cope with the traffic diversity in TSN, the TSN task group has published several standards for packet scheduling: 1) IEEE

Std 802.1Qbv proposed TAS for isolating scheduled traffic, 2) IEEE Std 802.1Qbu and IEEE std 802.3br proposed frame preemption for reduce the influence of best-effort traffic on the time-sensitive traffic; and 3) IEEE Std 802.1Qav (LAN/MAN Standards Committee 2009) proposed Credit-based shaping (CBS) for A/V traffic. As we mainly focus on efficiently utilizing the remaining bandwidth in the guard band, which can be understood as a side-effect of TAS, we omit the description on CBS. We show the standard TSN scheduler structure in Fig. 2.

In the default setting, each port possesses eight First-In-First-Out (FIFO) queues to buffer packets that are waiting for transmission. An incoming packet towards an egress port will be dispatched to different FIFO queues according to the Priority Code Point (PCP) value in its VLAN header or the Internal Priority Value (IPV) value if Per-Stream Filtering and Policing (PSFP) of IEEE Std 802.1Qci (LAN/MAN Standards Committee 2017) is enabled. As a result, the packets belonging to the same class is pushed into the same FIFO queue. By default, all packets of scheduled traffic are pushed into the #7 queue with the highest priority. TAS specifies a gate ahead of each FIFO queue to control the queue availability. If a queue's gate is closed or there is no packet in the queue, the queue will be regarded as 'empty'. When the output link is idle, the scheduling algorithm (e.g., SPS) will select the queue with the highest priority among all the 'non-empty' queues to dequeue its head packet. Specially, the open or close states of the gates are controlled by a Gate Control List (GCL), which consists of a series of pairs of time and gate states. Taking the example in Fig. 2, the gate state at time T0 is 10000000, so only the transmission gate for queue #7 is open, resulting in scheduled packets occupying the output link from time T0 to T1. In this way, the time line is split to many time windows, which is a special kind of Time Division Multiple Access (TDMA) in a broad sense. Flexible Time-Triggered Ethernet (FTTE) (Pedreiras et al. 2005; Meyer et al. 2013) also adopts time windows to isolate scheduled traffic, but the window's size and offset are fixed, in contrast to the flexible window obeying the GCL in TSN. To obtain a feasible GCL for a predefined scheduled traffic pattern, many schemes based on Satisability Modulo Theories (SMT) have been proposed (Steiner 2010; Craciunas et al. 2016; Oliver et al. 2018). This paper dose not get involved but assumes a GCL is available reasonably.

However, A/V or best-effort traffic will collide with scheduled traffic if one of their packets is occupying the link at the start of a scheduled flow's window. As shown in Fig. 3, the left is a snapshot of a TSN egress port with four FIFO queues and an SPS scheduler, and we assume no new packet will arrive since then, so packets will be dequeued according to their priorities strictly. For sequence diagram 1), at time $t1$, the transmission of packet $b3$ has not been finished, so the packet $a1$ cannot occupy the link, though its gate is open and its priority is the highest, which we call a collision. The collision may cause prolonged and non-deterministic end-to-end latency for the scheduled traffic. As the sequence diagram 1) shown in Fig. 3, the delayed packet $a2$ cannot be transmitted in its planned window and has to encroach the next window, and such an operation may bring a chain effect.

The root cause of the collision is that a low-priority packet under transmitting cannot stop at the exact starting point of a new window to give its way to the scheduled traffic. The basic IEEE Std 802.1Qbv sets a guard band before the start of each scheduled traffic's window, during which no packet will be allowed

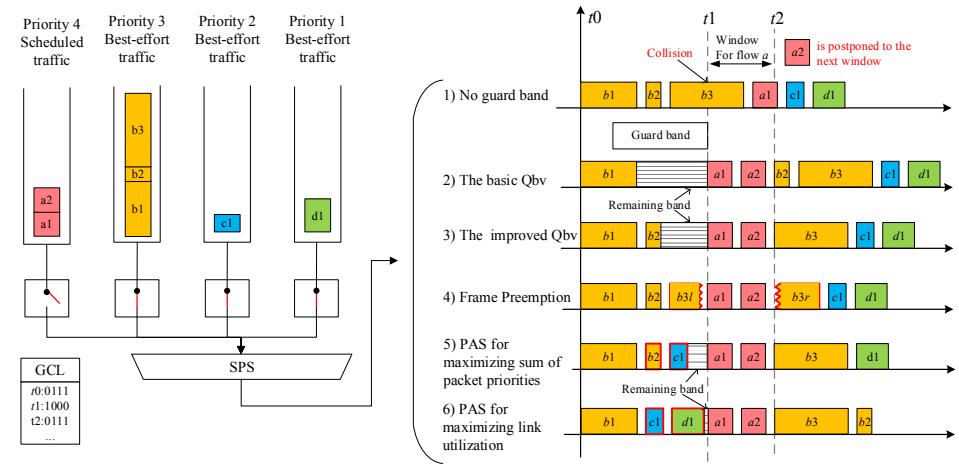


Fig. 3: The left is a snapshot for a TSN egress port with four queues and an SPS scheduler, and the right shows the subsequent packet transmission order for different scheduling algorithms if no new packet is enqueued since then. We specify that a larger priority value denotes a higher priority, so the priority order is $a > b > c > d$.

to transmit except for the on-the-fly packet, resulting in a clean state at the start of the next scheduled traffic's window. The safest approach is setting the guard band to be as long as the transmission time of the largest Ethernet frame as the sequence diagram 2) shown in Fig. 3, but it may waste a large amount of link resources, especially when the windows frequency is high and link rate is low. To relieve this problem, the improved Qbv keeps sending the highest-priority packet until it cannot be filled into the remaining band (assuming the size information can be obtained in advance). As the sequence diagram 3) shown in Fig. 3, packet $b2$ is transmitted because it has the highest priority and its size is small enough to fit in the remaining band. The following packet $b3$, however, is too large to fit the remaining band, causing a waste of bandwidth. Note that the improved Qbv reduces the average remaining band size, but its maximum bandwidth waste can be equal to the maximum guard band minus one byte when the highest-priority packet is as large as the maximum Ethernet frame.

Instead of delaying the packets that may cause collision, frame preemption allows splitting a low-priority packet at the start of a window and resumes the transmission of the remaining half after the window immediately. As the sequence diagram 4) shown in Fig. 3, frame preemption enables the large packet $b3$ to be split into two halves: $b3l$, which can fit the remaining band seamlessly, and $b3r$, which will be scheduled once the window ends. Although frame preemption improves the bandwidth utilization significantly, it has the following shortcomings. First, the ingress and egress ports need a dedicated and complicated hardware structure to support a new sub-MAC layer for eMAC (express Media Access Control) and pMAC (preemptable Media Access Control): eMAC is for express frames which are high-priority and cannot be preempted, while pMAC is for preemptable frames, which need the partition and assembling operations at the sender and receiver,

respectively. The hardware complexity makes it not so appealing. Second, it cannot eliminate the guard band completely because it still needs a guard band for a 124-byte frame. The reason is both two split frames must be no smaller than the minimum Ethernet frame length (64 bytes), a frame that is smaller than 124 bytes (128 bytes minus the 4-byte new CRC) cannot be split.

3 PAS primitive

In this section, we will first explain the motivation of proposing PAS. Then, we will model PAS with two optimization objectives and abstract both of them to a PCKP. Finally, we will raise a dynamic programming based algorithm PAS_I and a greedy strategy based algorithm PAS_G for packet scheduling.

3.1 Motivation

As mentioned above, the basic Qbv sets a guard band ahead of each scheduled traffic window, whose size is equal to the transmission time of the largest Ethernet frame. The improved Qbv approach keeps transmitting the highest priority packets until the remaining band cannot afford. Neither of them takes into account head packets of other queues, leading to a large remaining band waste potentially, even if some low-priority packets can indeed fill in it. As a result, the precious bandwidth resource is wasted. For the frame preemption solution, though it can significantly reduce the bandwidth waste, its dedicated and complicated MAC layer makes it hard to be deployed.

For this situation, we propose PAS, which is aware both the size of each packet to be scheduled and the amount of the remaining band, to make scheduling decisions. PAS is in pursuit of an approximate performance comparable to frame preemption without asking for a dedicated MAC layer. This is feasible in practice because the size of the head packets should be delivered to the scheduler in advance and the high-precision synchronized clock gives a precise remaining time estimation as well as the amount of remaining band by recognizing the product of the remaining time and link rate. By now, the coming question is the methodology of PAS.

In this paper, we target two optional optimization objectives for PAS, maximizing the sum of packet priorities or the bandwidth utilization. As the examples in Fig. 3, we only have three scheduling schemes, $(b2, c1)$, $(b2, d1)$ and $(c1, d1)$, to fill the guard band as much as possible. In order to maximize the sum of packet priorities, we select $(b2, c1)$ as the sequence diagram 5), because they have the largest priority sum. To maximize the bandwidth utilization, we select $(c1, d1)$ as the sequence diagram 6), because they have the largest size sum.

Note that PAS is an auxiliary scheduling algorithm which takes actions only in guard band. In other time periods, the scheduler runs on its configured scheduling algorithms such as SPS.

3.2 Model

We first present some mathematical definitions to model PAS. Assume that PAS is for a TSN egress port with N FIFO queues and the size of the original remaining band is denoted by r. Without loss of correctness, we assume all packets, including the ones arriving within the remaining time, are all buffered in their FIFO queues in advance. We also assume that making a scheduling decision costs zero time. We use N_i to denote the number of queued packets for queue i. For the j-th packet of queue i, $s_{i,j}$ and $p_{i,j}$ denote its size[2] and priority, respectively, where $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots, N_i$. Packets in the same queue must be dequeued in FIFO manner, i.e., successively from the head. Therefore, we use x_i to denote the number of packets to be dequeued in queue i, where $x_i \in \{0, 1, \ldots, N_i\}$.

For the objective of maximizing the sum of packet priorities, packets in a queue should be dequeued in an FIFO manner and the sum of all selected packets' size should be less than r. To sum up, we have the following optimization model.

$$\begin{aligned} &\max \sum_{i=1}^{N} \sum_{j=1}^{x_i} p_{i,j} \\ &s.t. \quad \begin{cases} x_i \in \{0, 1, \ldots, N_i\}, \\ \sum_{i=1}^{N} \sum_{j=1}^{x_i} s_{i,j} \leq r. \end{cases} \end{aligned} \tag{1}$$

This model is indeed equal to a PCKP, where packet size corresponds to the item weight, packet priority corresponds to the item value, and our objective corresponds to select some items obeying some orders (i.e., the PIFO manner) to a fixed-size knapsacka to get the total value as large as possible.

For the objective of maximizing bandwidth utilization, it is equal to maximizing the sum of packet sizes under a given r. We can also abstract it as a PCKP similar as the above. The only difference is that the item value is the packet size rather than the packet priority. Hence, we have the final model as follows.

$$\begin{aligned} &\max \sum_{i=1}^{N} \sum_{j=1}^{x_i} s_{i,j} \\ &s.t. \quad \begin{cases} x_i \in \{0, 1, \ldots, N_i\}, \\ \sum_{i=1}^{N} \sum_{j=1}^{x_i} s_{i,j} \leq r. \end{cases} \end{aligned} \tag{2}$$

3.3 Scheduling algorithms

Now that both the two models can be abstracted as a PCKP, we can use dynamic programming, one of the algorithm solutions for PCKP, to solve these models. The state transfer equation for the models are as follows,

$$Rank[i][s] = \max_{j=0}^{N_i} (Rank[i-1][s - sum_{size}[i][j]] + sum_{rank}[i][j]), \tag{3}$$

[2] The size refers to the real occupation of each packet on the wire, containing the Ethernet frame, eight-byte headers in the physical layer, and 12-byte Inter Frame Gap (IFG)

where $Rank[i][s]$ denotes the maximum sum of ranks (priorities or sizes according to the objective) for the first i queues with the total size constraint s; $sum_{size}[i][j]$, the sum of packet sizes of the first j packets in queue i; and $sum_{rank}[i][j]$, the sum of ranks. Using this state transfer equation, we can get an ideal PAS algorithm, PAS_I. Especially, we let PAS_I(P) denote the algorithm of maximizing the sum of packet priorities and PAS_I(S) for sizes.

The maximum operation in Eq. (3) can be executed in parallel, reducing the time complexity to $O(N)$. Even so, it is still overwhelming for hardware implementations when N goes large. Besides, the model is based on the fact that we know the information of all packets in advance. However, an online scheduler could not know the packet arriving in the future, so it fails to achieve the real optimal solution. Therefore, instead of getting the optimal solution by PAS_I, we tend to find a feasible algorithm in practice with the fast greedy strategy, called PAS_G, for the two optimization objectives as mentioned earlier.

For the objective of maximizing the sum of packet priorities, our first algorithm PAS_G(P/S) is to use the greedy strategy to select the head packet with the largest priority density, $p_{i,1}/s_{i,1}$. We define the priority density as the rank and get it when a packet is delivered into the scheduler. However, the priority density ranking needs division operations which are unfriendly to hardware implementation. We then turn to use the packet priority as the rank for simplicity, called PAS_G(P). PAS_G(P) can not only be deployed easily, but also be shared with most of the main scheduling algorithms which dequeue the packet with the highest rank each time. This is because the remaining band is larger that the largest Ethernet frame ahead of the guard band. In other words, the highest-ranked packet ahead of the guard band is indeed the highest-ranked eligible packet.

For the objective of maximizing bandwidth utilization, we select the head packet with the largest eligible size greedily, called PAS_G(S). PAS_G(S) tends to select fewer large packets to fill the remaining band, so the waste on IFG can be reduced. Note that PAS_G(S) cannot be completely shared with the most mainstream scheduling algorithms because it needs to maintain the packet size information separately.

4 Hardware designs

Considering the complexity, we focus on the hardware designs based on PAS_G(P) and PAS_G(S) in this section. For the standard TSN scheduler, we propose two structures to select the highest-ranked eligible item based on BCT and PIFO. However, these two structures both have limits on scalability, resulting in poor performance for a programmable TSN scheduler. Instead, we elaborate a new MPIFO structure based on the algorithm PAS_G(M).

4.1 Hardware designs for the standard TSN scheduler

Whatever we apply the PAS_G(P) or PAS_G(S) algorithms, their essential behavior is the same, i.e., to select the highest-ranked packet with a size constraint: the rank is either priority or size, while the size constraint is that the packet size

should always be smaller than the remaining band. Therefore, we can apply the same structure for both of them methodologically.

For cases without size constraints, the above two algorithms' behavior will become selecting the highest-ranked packet. Under this case, two classes of hardware-based work can be referred. The first class traverses all items to get the highest-ranked one upon each dequeue operation. As the fact that finding the maximum among N items takes at least $N-1$ pairwise comparisons, doing parallel comparisons by a mean of tree structure such as the BCT, should be the most efficient and economic way. The other class is based on the priority queue, which maintains a sorted list of items by their ranks, so the dequeue operation only needs to pop out the head item, though the enqueue operation needs to insert the item to a proper place. The priority queue is an abstract data structure, and many hardware-based designs have been proposed in academia: PIFO (Sivaraman et al. 2016), calendar queues (Brown 1988), shift registers (Moon et al. 2000; Chandra and Sinnen 2010), systolic arrays (Moon et al. (2000)), and binary heaps (Bhagwan and Lin 2000; Ioannou and Katevenis 2007; Huang et al. 2014). Next, the constraint adds a layer of filter fundamentally, which filters out packets that do not meet the constraint and limits the search space to eligible packets only.

In this subsection, we modify the two typical structures, i.e., BCT and PIFO, to realize the PAS_G enabled standard TSN scheduler. Before explaining the detailed hardware design, we first need to recall and clarify the requirements of the standard TSN scheduler as shown in Fig. 2: 1) a small number of (e.g., eight) physical FIFO queues with different priorities, 2) TAS to control the availability of some queues according to the GCL, and 3) SPS to select the queue with the highest priority among all the 'non-empty' queues upon each dequeue operation.

4.1.1 BCT-based structure

A BCT is a tree structure for extracting the maximum or minimum value among N inputs, which consists of $\log(N)$ levels of parallel comparators, assuming N is a power of 2 for simplicity. We use the PAS_G(S)-oriented BCT-based scheduler as an example in Fig. 4 to explain how it works. The scheduler does not need to store all the entire packets but only the extracted scheduling items, each of which has three fields, *queue index*, *priority*, and *size*, to represent its corresponding queue, the priority of this queue, and the size of the queue's head packet, respectively. The scheduling items first go through a filter layer to filter out ineligible ones that do not meet the size constraint. The passed items will update their corresponding input units in the BCT, each of which has two fields, *queue index* and *rank*. For example, the rank field of a unit represents the packet size in Fig. 4. Moreover, we only need to change the ranks of input units and can share the BCT structure, when exchanging the scheduling algorithm between PAS and SPS.

To simplify the hardware complexity, the number of input units should be fixed, e.g., four in the Fig. 4. We set the items that are filtered out by the constraint or corresponding to 'empty' queues with rank zero and others with positive ranks. For the example in Fig. 4, the rank of a packet is equal to its queue indexes plus one. In this way, the ineligible item will not appear at the output of the BCT as long as there exists an eligible packet. This setting is also applied to the PIFO-based and MPIFO structures.

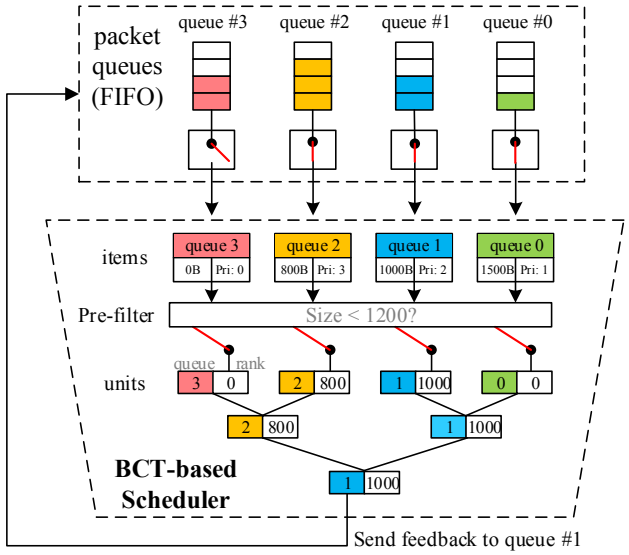


Fig. 4: An example of a BCT-based scheduler for the PAS_G(S) algorithm.

Finally, once getting an output unit from the BCT, we will operate its corresponding packet queue to dequeue its head, and deliver the information of the new head to the scheduler if the queue is not 'empty'. Besides, whenever an 'empty' packet queue becomes 'non-empty', we will also deliver the information of its head to the scheduler.

4.1.2 PIFO-based structure

The core of PIFO is a sorted list which inserts an item to an appropriate position according to its rank and removes the head item accordingly. The sorted list is based on the shift registers (Moon et al. 2000). Each enqueue operation need three steps: 1) compare the new item with all items in the list in parallel; 2) get the new item's position by encoding the comparation result; 3) according to the new item's position, shift the items in the list forward, backward or stay in place, and insert the new item in parallel. The dequeue operation is much more simple, which dequeues the head item and shift the other items forward in parallel. In contrast, BCT takes $\log(N)$ steps of parallel comparison for each dequeue operation, which takes more time as N grows.

The BCT-based structure needs a pre-filter to filer out ineligible items. On the contrary, PIFO-based structure needs a post-filter and a priority encoder to select the highest-ranked eligible item, as shown in the SPS_S(P)-oriented PIFO-based structure in Fig. 5. The PIFO queue is a list sorted by packet priorities for PAS_S(P). The post-filter consists of a layer of parallel comparators. The items whose sizes are smaller than the remaining band will be encoded to 1, and 0 otherwise. After this filter layer, we get a bitmap, where '1' denotes the corresponding item is eligible. Next, the priority encoder encodes the bitmap to an index of the item whose '1' is nearest to the head. For the example in Fig. 5, the bitmap 0111

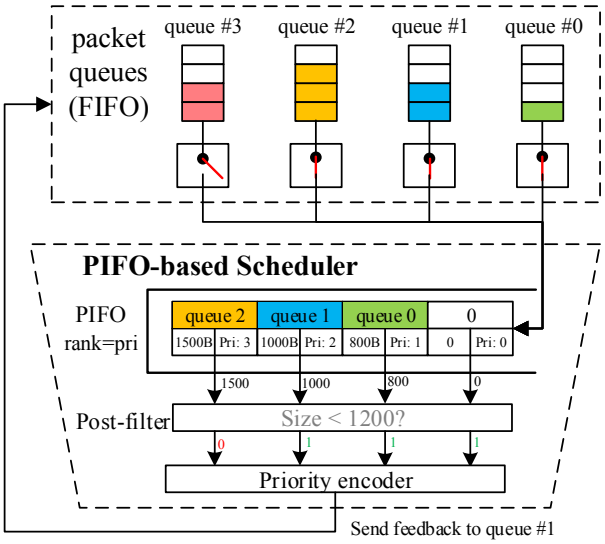


Fig. 5: An example of a PIFO-based scheduler for PAS_G(P) algorithm.

is encoded to 1. Then we can figure out which PIFO queue needs to dequeue its head packet from the queue index field.

The PAS_G(P)-oriented PIFO-based structure can be used throughout if the main scheduling algorithms always dequeue the highest-priority packet, such as SPS. However, PAS_G(S)-oriented one needs an isolated PIFO queue to maintain the packet size order, which is not cost-effective. In view of this, we will only evaluate the performance of PAS_G(P)-oriented PIFO-based scheduler in Section 5.

4.2 Hardware design for the programmable TSN scheduler

With the development of TSN, a more diverse and harsh application requirements have emerged, so the design of a packet scheduler needs to consider the following potential features for the programmable TSN scheduler: 1) the scheduling should be programmable to flexibly support a variety of scheduling algorithms such as WFQ, with no change to the existing hardware structure; 2) the scheduling should be fine-grained to satisfy the demand of different flows, e.g., storing packets in per-flow queues instead of per-class queues; 3) the scheduling should be scalable with the increasing number of scheduling items.

Sivaraman et al. (2016) show that a large number of scheduling algorithms can be expressed by the PIFO primitive. In other words, both the BCT-based and the PIFO-based structures are programmable if we adopt a specified rank distributor. Furthermore, replacing the eight per-class queues with hundreds of per-flow queues and distributing rank per flow, they can also provide fine-grained scheduling. However, the biggest bastion for them is the scalability. For BCT-based structure, each dequeue operation costs the time of $\log(N)$ levels of comparisons, which takes more time as the number of scheduling items increases. The bottleneck

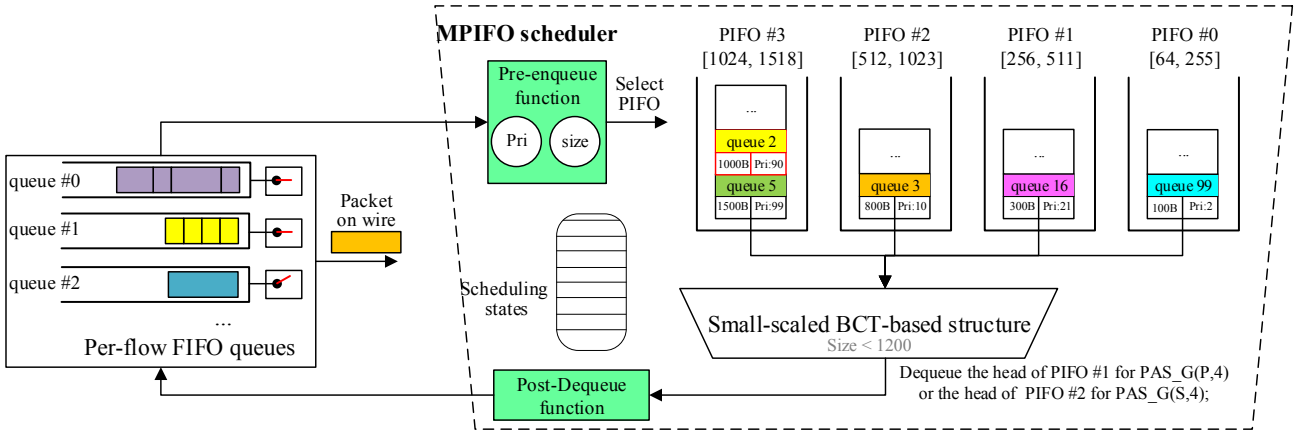


Fig. 6: An example of an MPIFO scheduler for PAS_G(4).

of PIFO-based structure is the priority encoder, which does not scale with the input number increasing and causes large latency in the extreme case. Applying pipeline technology cannot solve this problem, as the remaining size may be changed after several stages. Therefore, we strive to design a structure that can do dequeue operation at a stable and fast speed regardless of the number of per-flow packets queues.

4.2.1 Structure overview

The original intention of PIFO is to make the dequeue operation as fast as possible, but the post-filter layer and the priority encoder in the PIFO-based structure surely increase the dequeue complexity. If we can only dequeue a head packet like PIFO, the dequeue operation will be much simplified. Based on this idea, we propose the PAS_G(M) algorithm. PAS_G(M) divides the items into M sorted lists, e.g., PIFO queues, according to their size fields. When dequeuing, the scheduler selects the highest-ranked eligible item among the M heads of these lists. In this way, the scale of the problem is converted from N (the number of per-flow queues) to M (the number of size-based PIFO queues), reducing the complexity significantly. Note that PAS_G(M) provides a feasible solution, but can not guarantee the same performance of PAS_G, which will be analyzed later. Particularly, we define PAS_G(P, M) to select the highest-priority eligible item and PAS_G(S, M) to select the largest eligible packet.

Then we design the MPIFO structure to realize PAS_G(M), which sets M PIFO queues and a small-scaled BCT-based structure. Fig. 6 shows an MPIFO scheduler with four PIFO queues whose packet sizes are in range of 64 to 255 bytes, 256 to 511 bytes, 512 to 1023 bytes and 1024 to 1518 bytes, as the length of IEEE 802.3 Ethernet frame is from 64 to 1518 bytes. This grouping is merely an example, and the actual grouping should be based on the distribution of packet sizes in the network to balance the item number in each PIFO queue. Besides, to support the programmability for various scheduling algorithms, each enqueued and dequeued item should go through the pre-enqueue function and post-dequeue

function, respectively. The pre-enqueue and the post-dequeue functions work together to provide a rank according to the configured scheduling algorithms and maintain the scheduling states, referring to (Shrivastav 2019) for details. Next, we will detail the dequeue and enqueue operation of the MPIFO scheduler as follows.

Dequeue: whenever the output link is idle, the heads of the four PIFO queues will be pushed into the internal BCT-based structure. The rest operations are the same as the BCT-based structure, except that the post-dequeue function needs the information of the dequeued item to update the scheduling states. As the number of PIFO queues is small, the tree depth can be much shallow and even we can do $M(M-1)$ comparisons in parallel for a small scale, e.g., four items. Hence, the dequeue latency is greatly reduced to realize a high scheduling rate as shown in Section 5.

Enqueue: when a 'non-empty' per-flow queue receives a call from the post-dequeue function or an 'empty' per-flow queue becomes 'non-empty', the information of its flow id and head packet size will be sent to the MPIFO scheduler. The pre-enqueue function will first extract these information and create a new scheduling item with an appropriate rank, Then, it pushes the item to a PIFO queue according to its size, which can be realized by a layer of parallel comparators and an encoder. Note that this encoder is not a priority encoder and thus can work fast.

4.2.2 Mistake analysis

We define a mistake as a dismatched scheduling decision of PAS_G(M) to that of PAS_G , i.e., the dequeued packet may not be the eligible one with the largest size or highest priority. Mistakes are caused by the head blocking problem. We take the example in Fig. 6 to explain it, assuming the remaining size is 1200 bytes. For selecting the highest-priority eligible item, the head item in the PIFO #3 is oversized, so it blocks the second item which is assumed to be the highest-priority eligible one. As a result, PAS_G(S,M) will select the head packet of PIFO #1, and thus cause a mistake. On the other hand, for the mistake of selecting the largest eligible item, the head blocking problem causes that the second item of the PIFO #3 cannot be selected as well. However, the mistake ratio compared with PAS_G(P) is much smaller than that compared with PAS_G(S), as the PIFO queues are sorted by the priority as its rank.

We should keep in mind that a mistake just shows that the dequeued packet does not obey PAS_G but cannot assert it is even a worse selection, as PAS_G is also a local optimal solution. Nevertheless, we still want to figure out the expected mistake ratio and do some analysis as follows.

Let a random variable S denote packet size which is an integer in the range of $[S_{min}, S_{max}]$; a random variable R, the remaining band which is in the range of $[0, S_{max}]$; and a constant M, the number of PIFO queues indexed from 0 to $(M-1)$. For the i-th PIFO queue, its size coverage is denoted by $[L_n, U_n]$. The mapping from a remaining size r to the queue index i is denoted by $i = f(r) \in \{0, 1, ..., M-1\}$. Assume the size of each item in i-th queue obeys independent and identical distribution denoted by S_i, where $Pr\{S_i = s\} = \frac{\Pr\{S=s\}}{\Pr\{L_n \leq S \leq U_n\}}$. Besides, we assume all the PIFO queues have more than one packet.

Let M_p denote the mistake ratio of the PAS_G(P, M) algorithm. The head of each PIFO queue must store the highest priority in its own queue and we assume all the heads have the highest priority with equal probability. Hence, PAS_G(G, M) will cause a mistake when 1) the highest-priority eligible packet appears in the $f(r)$-th PIFO queue, whose probability is $\frac{\Pr\{L_{f(r)} \leq S \leq r\}}{\Pr\{S \leq r\}}$; and 2) the head packet of the $f(r)$-th PIFO is larger than r, whose size is larger than r. Hence, we can get the expectation of M_p as the following equation,

$$\boldsymbol{E}\{M_p\} = \sum_{r=S_{min}}^{S_{max}} \Pr\{R = r\} \frac{\Pr\{L_{f(r)} \leq S \leq r\}}{\Pr\{S \leq r\}} \Pr\{S_{f(r)} > r\}. \tag{4}$$

If S and R obey the uniform distribution, we can obtain,

$$\boldsymbol{E}\{M_p\} = \frac{1}{S_{max}} \sum_{r=S_{min}}^{S_{max}} \frac{r - L_{f(r)} + 1}{r - S_{min} + 1} \frac{U_{f(r)} - r}{U_{f(r)} - L_{f(r)} + 1}. \tag{5}$$

Then, assume the range of S_{min} to S_{max} can be divided into M groups with interval T evenly, an approximation of Eq. (5) will be

$$\boldsymbol{E}\{M_p\} \approx \frac{1}{S_{max}} \sum_{i=0}^{M-1} \sum_{t=1}^{T-1} \frac{t(T-t)}{T(iT+t)}, \tag{6}$$

which decreases as T decreases. Therefore, more groups can help reduce the mistake ratio.

Let M_s denote the mistake ratio of PAS_G(S, M). For a given remaining size r, PAS_G(S, M) can obtain the correct item only if the head of the $f(r)$-th PIFO queue is the largest eligible one in its queue, or all items in queue $f(r)$ are not eligible but the head of the queue $f(r) - 1$ is the largest one in its queue. Assume the priority and size are independent mutually and the item number in queue i is c_n, so if there is at least an eligible item in the $f(r)$-th queue, the largest eligible item appears at the head with the probability of $\frac{1}{c_i}$. On the other hand, if all items are not eligible in the $f(r)$-th queue, the largest eligible item will be in the queue $f(r) - 1$ and appears at the head with the probability of $\frac{1}{c_{f(r)-1}}$. Hence, we can get the expectation of M_s as follows,

$$\boldsymbol{E}\{M_s\} = 1 - \sum_{r=S_{min}}^{S_{max}} \Pr\{R = r\} \left[\frac{1-q}{c_{f(r)}} + \frac{q}{c_{f(r)-1}} \right], \tag{7}$$

where $q = (\Pr\{S_{f(r)} > r\})^{c_{f(r)}}$. If each queue has the same item number c, we have an approximation $\boldsymbol{E}\{M_s\} \approx 1 - \frac{1}{c}$. Even though PAS_G($M$) has low probability to get the largest eligible item, an average large eligible packet is still encouraging to reduce the waste of the guard band. Furthermore, for a given r, we can get the size expectation of the output item S_o as follows,

$$\boldsymbol{E}\{S_o|r\} = \sum_{s=L_{f(r)}}^{r} \Pr\{S_{f(r)} = s\}s + \Pr\{S_{f(r)} > r\} \sum_{s=L_{f(r)-1}}^{U_{f(r)-1}} \Pr\{S_{f(r)-1} = s\}s. \tag{8}$$

Table 1: Frequency distribution of packet sizes

Size interval	[84, 138)	[138, 1438)	[1438, 1538)	1538
Probability	0.45	0.15	0.2	0.2

If S obeys the uniform distribution, we can obtain,

$$\begin{aligned} \boldsymbol{E}\{S_o|r\} &= \sum_{s=l}^{r} \frac{s}{u-l+1} + \frac{u-r}{u-l+1} \sum_{s=l'}^{u'} \frac{s}{u'-l'+1} \\ &= \frac{(r+l)(r-l+1)+(u'+l')(u-r)}{2(u-l+1)}, \end{aligned} \tag{9}$$

where $u = U_{f(r)}, l = L_{f(r)}, u' = U_{f(r)-1}$, and $l = L_{f(r)-1}$. Eq. (9) shows the expectation is a quadratic function of r and it increases in the interval $[l, u]$. Particularly, if $r = u$, i.e., the items in the queue $f(u)$ are all eligible, the expectation will get the maximum $(u + l)/2$.

5 Evaluation

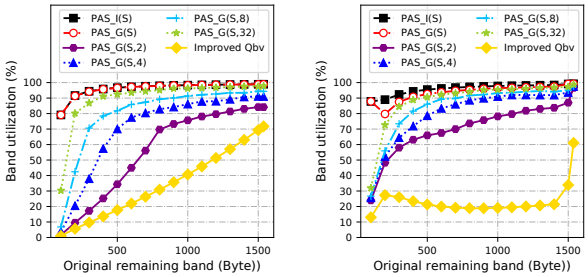
In this section, we first do simulations to compare the band utilization and priority density in the remaining band of the PAS family and the improved Qbv. Then we implement the BCT-based, PIFO-based, and MPIFO designs with System Verilog (Wikipedia 2020) and the synthesis results show that MPIFO has highest performance as the number of flow queues grows.

5.1 Simulation

5.1.1 Setup

Data set: we set 255 per-flow queues indexed for 0 to 254 in our simulations. For simplicity, each queue stores four packets with the same priority equal to their queue indexes plus one, i.e., 1 to 255. The packet size on the wire is in the range of 84 to 1538, which is composed of the 802.3 Ethernet frame size (64 to 1518 bytes), Preamble (1 byte), Start Frame Delimiter (SFD, 7 bytes) and IFG (12 bytes). Next, the packet size obeys two typical distributions: 1) uniform distribution, i.e., $\Pr\{S = s\} = 1/1455$, where $s = 84, 85, ..., 1538$, and 2) frequency distribution (Caida 2019), as shown in Table 1.

Metrics: first, we explicitly define remaining band utilization as the proportion of the remaining band used to send packets (excluding IFG), and priority density as the priority sum divided by the original remaining size. Then, we compare the remaining band utilization of algorithms PAS_I(S), PAS_G(S), PAS_G(S, M), where $M = 2, 4, 8, 32$, and the improved Qbv and the priority density of algorithms PAS_I(P), PAS_G(P/S), PAS_G(P), PAS_G(P, M), where $M = 2, 4, 8, 32$, and the improved Qbv. The values of the improved Qbv is from the theoretical results, and each value of other algorithms is an average of 1000 tests to reduce the impact from random numbers.



(a) Uniform distribution (b) Frequency distribution

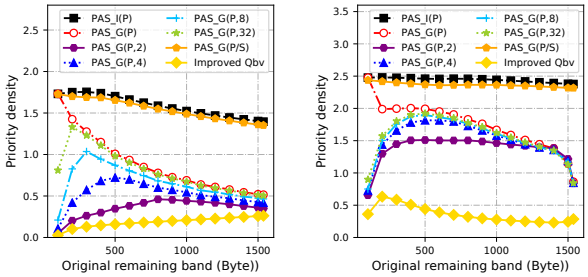
Fig. 7: Remaining band utilization of the algorithms on different packet-size distributions.

5.2 Remaining band utilization

Fig. 7 shows the remaining band utilization on uniform and frequency distribution of packet size. We can see all the algorithms have an upward trend and PAS_I(S) has the highest utilization closely followed by the second highest one PAS_G(S). The larger M is, the smaller gap between PAS_G(S) and PAS_G(S, M) can be achieved, i.e., increasing the PIFO queue number indeed improves the band utilization of MPIFO. However, using M PIFO queues also means M times more logic resources, so the number of M should not be too large. Considering the utilization of PAS_G(S,4) is high enough, we set the default MPIFO with four PIFO queues. Besides, as the original remaining band grows, the gap between PAS_G(S) and PAS_G(S, M) narrows as well, due to more selections out of the M PIFO queues. The improved Qbv has the worst performance, especially when the remaining size is not large. This is because its band utilization heavily relies on whether the first selected highest-priority packet can fit the band. On the other hand, the biggest difference between uniform and frequency distribution is that algorithms on the frequency distribution can achieve higher utilization when the original remaining band is small. This is because nearly half of packets on the wire for the frequency distribution are less than 138 bytes.

5.3 Priority density

Fig. 8 shows the priority density on uniform and frequency distribution. PAS_I(P) still exhibits the highest utilization, closely followed by PAS_G(P/S). The lines of improved Qbv are still at the bottom, much lower than the others. Similarly, as M increase, PAS_G(S, M) is closer to PAS_G(P), but still has a gap to PAS_I(P) and PAS_G(P/S), which means the priority greedy algorithms perform worse than the priority density greedy algorithms. However, as priority greedy algorithms are easier to be implemented in hardware, we still use them to prototype our BCT-based, PIFO-based, and MPIFO schedulers. Specially, PAS_G(S,M) presents a trend of rising first and then falling, which is caused by the following two aspects. 1) When the original remaining band is small, the dequeue selection is limited to



(a) Uniform distribution (b) Frequency distribution

Fig. 8: Priority density of the algorithms on different packet-size distributions.

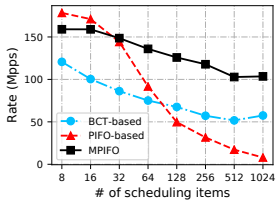
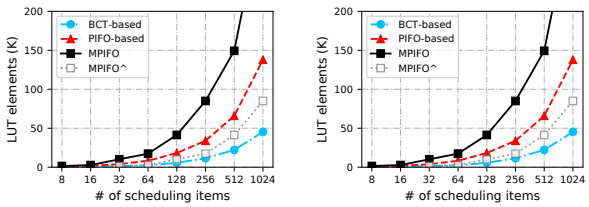


Fig. 9: Scheduling rate of BCT-based, PIFO-based, and MPIFO schedulers as scheduling items increase.

several PIFO queues, so the eligible packets with higher priority are more likely to be blocked. Since a larger M means a larger range for selection under a given remaining band, PAS_G(S,32) grows faster than others. 2) PAS_G(S, M) is more likely to select a large packet with a low priority density, so the curves cannot maintain a linear growth and all drops with the remaining band increasing. The default PAS_G(P,4) can provide a sufficient priority density and is easy to be implemented.

5.4 Implementation

We prototype the BCT-based, PIFO-based, and MPIFO scheduler on a Xilinx Virtex-7 (Xilinx 2020) FPGA comprising 712K Look-Up Table (LUT) elements and 1424K Flip-flops. Our prototypes are written in System Verilog comprising ∼400, ∼400, and ∼700 LOCs for BCT-based, PIFO-based, and MPIFO scheduler, respectively. We evaluate the performance of our prototypes across two metrics: scheduling rate and resource occupation. To serve as the baseline, we synthesize the three implementations atop our FPGA by Xilinx Vivado. We use 8-bit priority and 12-bit size fields, and the MPIFO scheduler is equipped with four PIFO queues.



(a) LUT consumption (b) Flip-fop consumption

Fig. 10: Resource consumption of BCT-based, PIFO-based, MPIFO, and MPIFOˆ schedulers as the number of scheduling items increases.

5.4.1 Scheduling rate

We evaluate the scheduling rate below which the scheduler can do the scheduling decisions normally, i.e., the enqueue and dequeue operations. Scheduling rate is a significant indicator for a scheduler design. Fast scheduling rate means the scheduler can adapt to high line rate. Although pipeline technology can achieve high throughput in many scenarios, it cannot be used in PAS. This is because the remaining band may have changed from the first stage to the last one in the pipeline.

Fig. 9 shows the scheduling rate for the three scheduler designs. We can see that when the item scale is small, the complexities of their dequeue operations are all relatively low, so they all have high scheduling rate. PIFO-based scheduler even is the best due to its fast enqueue and dequeue operation. However, as the item scale expands, though they all show a downward trend, MPIFO scheduler outperforms PIFO-based one and achieves more than 100 M packet per second (Mpps) scheduling rate when the number of items is 1024, which proves it is better design of the high-speed programmable TSN scheduler. The dequeue operation of MPIFO scheduler only involves parallel comparisons among the four heads, so its dequeue rate keeps stable. Its scheduling rate is limited by its enqueue operation, whose encoding step costs more time as the item scale expands. PIFO-based scheduler suffers a lot with the scale increasing because its dequeue operation needs a priority encoder, which is not suitable for large-scaled inputs. aCT-based scheduler sees a slow downwards trend as the delay of its dequeue operation is linear with the number of BCT levels. We find that the scheduling rate of BCT-based scheduler at 1024 items is a little higher than that at 512 items. We cannot figure out the specific reason yet but we speculate it is caused by the synthesis settings.

5.4.2 Resource occupation

The logic resource consumption on LUT and Flip-flop is a key to the feasibility of a design in practice. As shown in Fig. 10, we count the LUT and Flip-flop consumption of different schedulers with the number of scheduling items increases. MPIFOˆ specifies a MPIFO scheduler that sacrifices packet loss in exchange for low resource consumption. For example, a MPIFOˆ maintains 4x256 PIFO queues

for 1024 scheduling items, in contrast to 4x1024 queues in the default MPIFO. Fig. 10a and Fig. 10b show the same trends that the MPIFO scheduler explodes with the scale expanding, while the BCT-based scheduler keeps the smallest. If we allow a low packet loss ratio, adopting MPIFO^ takes the second smallest resources and is relatively salable as the scheduling items increase.

6 Related work

We will mainly focus on two aspects: 1) the general packet scheduling algorithms, and 2) the current TSN standards involved in packet scheduling.

6.1 General packet scheduling algorithms

Many algorithms have been proposed in the long history of packet scheduling algorithms. Broadly speaking, they determine when and in what order for incoming packets to be transmitted, which can be divided into two classes: work-conserving and non-work conserving algorithms. The former will never let the output be link idle as long as there are packets in queue. Some typical work-conserving scheduling algorithms are Deficit Round Robin (DDR) (Shreedhar and Varghese 1996), WFQ (Demers et al. 1989), Worst-case Fair Weighted Fair Queuing (WF^2Q) (Bennett and Zhang 1996), and Stochastic Fairness Queuing (SFQ) (McKenney 1990). On the other hand, non-work conserving scheduling algorithms, whose representative is Token Bucket (Wikipedia 2020), will let the link idle if they think the packets in the queue are not eligible for transmission, so they actually express traffic shaping primitives (Saeed et al. 2017; Radhakrishnan et al. 2014).

Since the advent of SDN, the programmability of data plane has been remarkably developed, e.g., the language P4 (Bosshart et al. 2014) and reprogrammable hardware Barefoot Tofino (Barefoot Networks 2017). Achieving the programmable packet scheduling algorithm is the last mile to the complete data-plane programmability. Although no universal structure can express all the scheduling algorithms, which are proved by Mittal et al. (2016), researchers strive to find a more general structure to cover more scheduling algorithms. Sivaraman et al. (2016) propose a PIFO structure to realize many rank-based scheduling algorithms at line rate, and Shrivastav (2019) further proposes a PIEO structure to realize scheduling algorithms that can be abstracted as dequeuing the highest-ranked eligible packet. Nevertheless, they are both far away to be implemented in commercial switches due to they need a new ASIC for the scheduling module. To utilize the current programmable hardware switches, SP-PIFO (Alcoz et al. 2020) seeks to realize approximate performance of PIFO with multiple FIFO queues and SPS by adaptively controlling packets entering different queues.

6.2 Packet scheduling algorithms in TSN

Back to current TSN standards, there are two typical non-work conserving algorithms, namely TAS and CBS, and one work-conserving algorithm SPS. Most research work focuses on TAS as it is one of the most prominent features in TSN.

First, TAS relies on an efficient GCL to divide the time windows. Generating a GCL is proved to be an NP-hard problem by Steiner (2010). To obtain a feasible solution for a certain scheduled traffic pattern, many algorithms based on SMT are proposed (Steiner 2010; Craciunas et al. 2016; Oliver et al. 2018). SMT are designed to determine the satisfiability of the first-order logical formulas against certain background theories like linear integer arithmetic or bit-vectors. There are many constraints in these algorithms, such as the end-to-end latency and hop-to-hop latency, and the biggest difference lies in the specific constraints. Particularly, the flow isolation constraint is that if a frame of a given flow has entered a queue, no frame of another flow may arrive at this queue until all frames of the previous flow have been fully dispatched, which maintains the determinacy of packet forwarding. From another point of view, as these algorithms are based on limited constraints, it remains to be seen whether they perform well in actual TSNs.

The guard band problem has also attracted attentions from the academia. Dürr and Nayak (2016) notice that if multiple time windows are seamlessly back-to-back, only one guard band is needed, so they aggregate as many time windows as possible. This scheme is effective when the end-to-end latency is not strict so that the time window has much space to move forward and backward. Apart form this, Heilmann and Fohler (2019) try to reduce the guard band size like PAS. They set two FIFO queues for small and large packets, respectively, for each packet class. The GCL controls the gates for the large packets' queues to be closed when the remaining band is small enough. This scheme can improve the bandwidth utilization, but it has the following drawbacks: 1) it needs an extra physical queue for each class; 2) it is not fine-grained except increasing the queue number; 3) the GCL size will double and be more complex. On the contrary, our BCT-based and PIFO based structures for standard TSN scheduler need no extra physical queue, and MPIFO is more resource-efficient.

The critical issue of TAS is that it needs a synchronized clock with nanosecond precision. Although it can be realized by IEEE 802.1AS, it is still much complex and expensive for commercial switches. Hence, some work explores how to ensure the end-to-end latency without a strict synchronized clock. Li et al. (2019) solve this problem by keeping the scheduled flows on two disjoint paths with TAS and best-effort method. Their experimental results show that the latency is much smaller thanks to the assistance of the best-effort path, and thus the strict timing requirement of TAS can be relaxed. An alternative for TAS is to use the Asynchronous Traffic Shaping (ATS), which does not need a strict timing synchronization. The fundamental research of ATS is the Urgency Based Scheduler (UBS) (Specht and Samii 2016), which achieves the Quality of Service (QoS) by asynchronous sub-shapers in each switch. Each switch possesses a number of FIFO queues for packets form different upstream switches and priorities and schedules urgent traffic first. As the scheduling decision is based on the local clock, UBS is easy to be implemented, but the end-to-end jitter may be amplified.

With the introduction of the standards and schemes for TSN, theoretical work is also proceeding steadily. Traditional probability theory focuses on the expectation of latency, but TSN cares more about whether the latency requirement is met in the worst cases. Network Calculus (Le Boudec and Thiran 2001) is a theory for analyzing the bound of latency, queue length, and so on, which is suitable for TSN. Therefore, Zhao et al. (2018) use Network Calculus to calculate the worst-case la-

tency that the scheduled traffic may experience along a path with configured time windows.

7 Conclusion

The guard band is born with the TAS in IEEE std 802.1Qbv, and it brings non-negligible bandwidth waste especially when the window number is large and link speed is not too high. We propose an algorithm family of PAS to utilize the guard band efficiently, which is based on the classic PCKP. Considering the hardware implementation, we focus on designs on PAS_G, and propose BCT-based and PIFO-based structures for a standard TSN scheduler with eight packet queues per port. The programmable TSN scheduler needs hundreds of per-flow packet queues to realize fine-grained custom scheduling algorithms, but these two structures do not scale well as the queue number increases. Therefore, we further propose the MPIFO structure for optimizing scheduling rate. In the future, we will focus on improving the MPIFO structure with less resource consumption and no packet drop.

References

Alcoz AG, Dietmüller A, Vanbever L (2020) SPPIFO: Approximating push-in first-out behaviors using strict-priority queues. In: USENIX Symposium on Networked Systems Design and Implementation (NSDI)

Barefoot Networks (2017) Barefoot Tofino. URL http://barefootnetworks.com/

Bennett JC, Zhang H (1996) WF^2Q: worst-case fair weighted fair queueing. In: Proceedings of IEEE INFOCOM'96. Conference on Computer Communications, IEEE, vol 1, pp 120–128

Bhagwan R, Lin B (2000) Fast and scalable priority queue architecture for high-speed network switches. In: Proceedings of the IEEE International Conference on Computer Communications (INFOCOM), IEEE, vol 2, pp 538–547

Bosshart P, Daly D, Gibb G, Izzard M, McKeown N, Rexford J, Schlesinger C, Talayco D, Vahdat A, Varghese G, et al. (2014) P4: Programming protocol-independent packet processors. ACM Special Interest Group on Data Communication (SIGCOMM) Computer Communication Review 44(3):87–95

Brown R (1988) Calendar queues: a fast 0(1) priority queue implementation for the simulation event set problem. Communications of The ACM 31(10):1220–1227

Caida (2019) Packet size distribution comparison between Internet links in 1998 and 2008. URL https://www.caida.org/research/traffic-analysis/$\mathrm{pkt}_s ize_d istribution/graphs.xml$

Chandra R, Sinnen O (2010) Improving application performance with hardware data structures. In: Proceeding of the IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), IEEE, pp 1–4

Craciunas SS, Oliver RS, Chmelík M, Steiner W (2016) Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks. In: Proceedings of the ACM International Conference on Real-Time Networks and Systems, ACM, pp 183–192

Demers A, Keshav S, Shenker S (1989) Analysis and simulation of a fair queueing algorithm. In: the ACM Special Interest Group on Data Communication (SIGCOMM) Computer Communication Review, ACM, vol 19, pp 1–12

Dürr F, Nayak NG (2016) No-wait packet scheduling for IEEE time-sensitive networks (TSN). In: Proceedings of the ACM International Conference on Real-Time Networks and Systems, ACM, pp 203–212

Heilmann F, Fohler G (2019) Size-based queuing: an approach to improve bandwidth utilization in TSN networks. ACM Special Interest Group on Embedded Systems Review 16(1):9–14

Huang M, Lim K, Cong J (2014) A scalable, high-performance customized priority queue. In: Proceedings of the IEEE International Conference on Field Programmable Logic and Applications (FPL), IEEE, pp 1–4

International Organization for Standardization (2003) ISO 11898: Road vehicles – Controller area network (CAN). ISO

International Organization for Standardization (2013) ISO 17458: Road vehicles - FlexRay communications system (1st Edition). ISO

Ioannou A, Katevenis MG (2007) Pipelined heap (priority queue) management for advanced scheduling in high-speed networks. IEEE/ACM Transactions on Networking (ToN) 15(2):450–461

Jansen D, Buttner H (2004) Real-time Ethernet: the EtherCAT solution. Computing and Control Engineering 15(1):16–21

Kopetz H, Bauer G (2003) The time-triggered architecture. Proceedings of the IEEE 91(1):112–126

Kopetz H, Ademaj A, Grillinger P, Steinhammer K (2005) The time-triggered Ethernet (TTE) design. In: Proceedings of the IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC), IEEE, pp 22–33

LAN/MAN Standards Committee (2009) IEEE Standard for Local and metropolitan area networks—Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams. IEEE Std pp 1–71

LAN/MAN Standards Committee (2011) Ieee standard for local and metropolitan area networks—timing and synchronization for time-sensitive applications in bridged local area networks. IEEE Std pp 1–274

LAN/MAN Standards Committee (2016a) IEEE Standard for Ethernet Amendment 5: Specification and Management Parameters for Interspersing Express Traffic. IEEE Std pp 1–58

LAN/MAN Standards Committee (2016b) IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption. IEEE Std pp 1–52

LAN/MAN Standards Committee (2016c) IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic. IEEE Std pp 1–57

LAN/MAN Standards Committee (2017) IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing. IEEE Std pp 1–65

Le Boudec JY, Thiran P (2001) Network calculus: a theory of deterministic queuing systems for the internet, vol 2050. Springer Science & Business Media

Li Z, Wan H, Zhao B, Deng Y, Gu M (2019) Dynamically optimizing end-to-end latency for time-triggered networks. In: Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM) Workshop on Networking for Emerging Applications and Technologies, ACM, pp 36–42

McKenney PE (1990) Stochastic fairness queueing. In: Proceedings. IEEE INFOCOM'90: Ninth Annual Joint Conference of the IEEE Computer and Communications Societies@ m_The Multiple Facets of Integration, IEEE, pp 733–740

Meyer P, Steinbach T, Korf F, Schmidt TC (2013) Extending IEEE 802.1 AVB with time-triggered scheduling: A simulation study of the coexistence of synchronous and asynchronous traffic. In: Proceedings of the IEEE Vehicular Networking Conference, IEEE, pp 47–54

Mittal R, Agarwal R, Ratnasamy S, Shenker S (2016) Universal packet scheduling. In: USENIX Symposium on Networked Systems Design and Implementation (NSDI), pp 501–521

Moon S, Rexford J, Shin KG (2000) Scalable hardware priority queue architectures for high-speed packet switches. IEEE Transactions on Computers (TOC) 49(11):1215–1227

Oliver RS, Craciunas SS, Steiner W (2018) IEEE 802.1 Qbv gate control list synthesis using array theory encoding. In: Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), IEEE, pp 13–24

Pedreiras P, Gai P, Almeida L, Buttazzo GC (2005) FTT-Ethernet: A flexible real-time communication protocol that supports dynamic QoS management on Ethernet-based systems. IEEE Transactions on industrial informatics 1(3):162–172

Radhakrishnan S, Geng Y, Jeyakumar V, Kabbani A, Porter G, Vahdat A (2014) SENIC: Scalable NIC for end-host rate limiting. In: USENIX Symposium on Networked Systems Design and Implementation (NSDI), pp 475–488

Saeed A, Dukkipati N, Valancius V, Contavalli C, Vahdat A, et al. (2017) Carousel: Scalable traffic shaping at end hosts. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, ACM, pp 404–417

Shreedhar M, Varghese G (1996) Efficient fair queuing using deficit round-robin. IEEE/ACM Transactions on networking (3):375–385

Shrivastav V (2019) Fast, scalable, and programmable packet scheduler in hardware. In: Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM), ACM, pp 367–379

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 Special Interest Group on Data Communication (SIGCOMM), ACM, pp 44–57

Specht J, Samii S (2016) Urgency-based scheduler for time-sensitive switched Ethernet networks. In: Proceedings of the IEEE Euromicro Conference on Real-Time Systems (ECRTS), IEEE, pp 75–85

Steiner W (2010) An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks. In: Proceedings of the Real-Time Systems Symposium, IEEE, pp 375–384

Wikipedia (2020) System Verilog. URL https://en.wikipedia.org/wiki/SystemVerilog

Wikipedia (2020) Wikipedia. Token Bucket. URL https://en.wikipedia.org/wiki/Token_bucket

Xilinx (2020) Virtex-7. URL https://www.xilinx.com/products/silicon-devices/fpga/virtex-7.html

Zhao L, Pop P, Craciunas SS (2018) Worst-case latency analysis for IEEE 802.1 Qbv time sensitive networks using network calculus. IEEE Access 6:41803–41815