Merger: Inversion-Free and Practical Programmable Packet Scheduler

Jinbin Hu[1], Zhao Zhang[1] and Jin Wang[2]

[1]Changsha University of Science and Technology, Changsha 410114, China
[2]Hunan University of Science and Technology, Xiangtan 411201, China
jinbinhu@csust.edu.cn,
zhangzhao746@stu.csust.edu.cn,jinwang@hnust.edu.cn

Abstract. The packet schedulers are crucial for determining the transmission order, a factor that significantly influences key network performance metrics. However, conventional hardware-based schedulers lack flexibility. Recent programmable data-plane technologies enable flexible schedulers without compromising throughput and latency. We propose Merger, a novel scheduler combining multi-merge and dynamic queue management with limited queue resources. Merger achieves comparable performance to PIFO (Push-In First-Out queue) and SP-PIFO (Approximating Push-In First-Out Behaviors using Strict-Priority Queues) but with significantly reduced implementation complexity and resource overhead. The simulation results show that Merger offers performance close to the ideal PIFO while being more practical than SP-PIFO and Sifter.

Keywords: Packet Scheduler, Inversion-free, Programmable

1 Introduction

Packet schedulers are crucial for determining transmission order, which greatly affects key network performance metrics like latency and throughput [4]. Their algorithms directly impact resource allocation among competing flows, influencing overall network efficiency and service quality. However, conventional hardware-based schedulers have fixed functionalities set during design and manufacturing, making it hard to update scheduling policies without replacing hardware [1-4]. Thus, there's a rising need for more adaptable scheduling mechanisms to meet the dynamic demands of today's network environments.

In recent years, with the development of programmable data-plane technology, researchers have focused on creating high-performance, flexible packet schedulers without compromising throughput and latency. PIFO [1], a new hardware primitive offering a priority-queue abstraction, enables line-rate operation and programmability. But its hardware implementation faces scalability challenges [6], especially in scenarios with large traffic volumes and high data rates, limiting its use in current network devices.

To address PIFO's hardware deployment challenges, SP-PIFO was introduced [2]. By dynamically adjusting the mapping between packet ranks and FIFO queues, it minimizes scheduling errors and approximates PIFO's behavior. However, the SP-PIFO introduces packet inversion [3]. Hence, Sifter was devised to enhance the scalability of packet schedulers with[1]out causing packet inversion. Although Sifter resolves PIFO's scalability issues and eliminates packet inversion, its introduction of two new

[2] Jin Wang is the corresponding author.

structures—the RCQ and Mini-PIFO [3], which are absent in existing switches, makes its direct application to current switches challenging.

In this paper, we propose Merger, a novel scheduler combining multi-merge and dynamic queue management within limited queue resources. Merger achieves performance comparable to SP-PIFO and Sifter while reducing implementation complexity. It ensures accurate and efficient packet scheduling by being packet inversion-free. Simulation results show that Merger closely approaches the ideal PIFO's performance and outperforms SP-PIFO and Sifter. In summary, our key contributions are:

(1) We analyze the limitations of conventional hardware-based schedulers and existing programmable schedulers like SP-PIFO and Sifter.

(2) We propose Merger, a new programmable packet scheduler that combines multi-merge operations and dynamic queue management to maintain precise scheduling without packet inversion.

(3) We conduct extensive simulations to evaluate Merger's performance. The results demonstrate that Merger reduces normalized FCT by approximately 30%-40% and End-to-End delay by about 20%-30% compared to SP-PIFO and Sifter. Additionally, Merger can be deployed on existing hardware without costly modifications.

2 Motivation

PIFO ensures precise packet ordering through priority-queue abstraction but has high hardware demands that grow with traffic volume. SP-PIFO improves scalability by dynamically mapping packet ranks to FIFO queues but introduces packet inversion [2], where lower-rank packets are transmitted after higher-rank ones, increasing FCT and End-to-End delay for latency-sensitive flows. To demonstrate packet inversion's adverse effects, we ran the SFQ [5] algorithm on SP-PIFO and PIFO within an 8×8 leaf-spine topology using the NS-3 simulator [8]. The results shown in Figure 1, indicate that SP-PIFO's performance metrics were significantly higher than PIFO's.

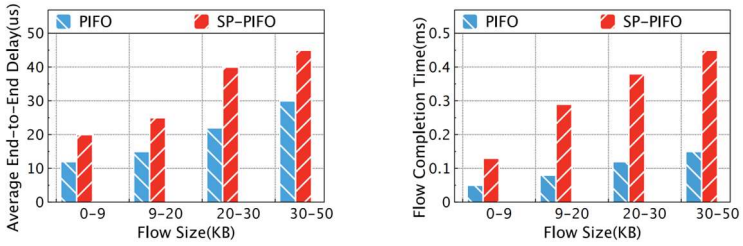

Fig. 1. Inversion Impact on End-to-End Delay and FCT

Besides, Sifter eliminates packet inversion and enhances scalability through RCQ and Mini-PIFO [3]. The RCQ enforces packet order constraints, while the Mini-PIFO efficiently manages smaller packets. However, current switches lack these structures [9], hindering Sifter's direct application without hardware modifications.

Thus, there is an urgent need for a high-performance, programmable, inversion-free packet scheduler on existing hardware. It should maintain PIFO's performance benefits, avoid packet inversion, and enhance scalability and resource efficiency. Since switch queues are FIFO [11], if each queue maintains a strict ascending order of packet ranks, we can select and transmit the packet with the smallest rank from all queues' front.

To ensure packets in each queue are in ascending rank order, we can track the maximum rank of each queue [2]. As shown in Fig.2, Only packets with a rank greater than or equal to this recorded maximum can be pushed into the queue. Consequently, the maximum rank aligns with the rank of the packet at the queue's end.

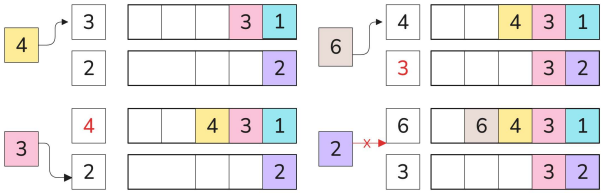

Fig. 2. Push packets in the queue

Clearly, this method may encounter a situation where, after long-term operation, no queue can accept new packets, as shown in the last scenario of Fig. 2. When all queues are full or their maximum rank limits prevent new packets with smaller ranks from being enqueued, incoming packets cannot be placed into any queue.

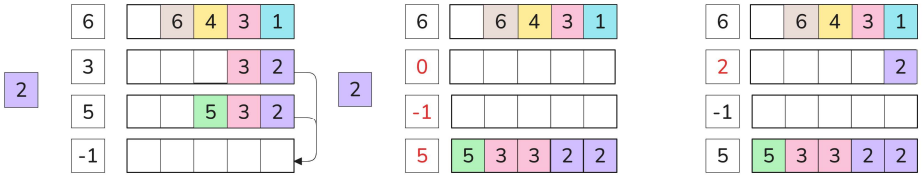

Fig. 3. Merge two queues to free up space

To resolve this issue, a queue-merging strategy is employed. Specifically, when a new packet arrives and cannot be enqueued into any existing queue, the system selects two appropriate queues for merging. As illustrated in Fig. 3, the merging process involves combining the packets from these two queues into a single queue, thereby freeing up space. This approach ensures that the incoming packet can be accommodated, maintaining the continuity and efficiency of the packet scheduling process.

3 Design

3.1 Design overview

In this section, we describe the detailed design of Merger, our proposed programmable packet scheduler. It aims to address current schedulers' limitations in scalability, scheduling accuracy, and hardware resource utilization [1-6]. Merger combines multi-merge sorting and dynamic queue management to achieve high-performance, inversion-free scheduling with limited queue resources.

Specifically, Merger maintains the maximum rank value for each queue, ensuring packets are queued in ascending rank order. When a new packet's rank is lower than ranks of all packets at back of queues [2], it triggers a merge operation. This combines two queues' packets into one, freeing space for the high-priority packet. This design ensures precise scheduling and efficient queue resource utilization, enabling Merger to run efficiently on existing hardware. The architecture of Merger is illustrated at Fig. 4.

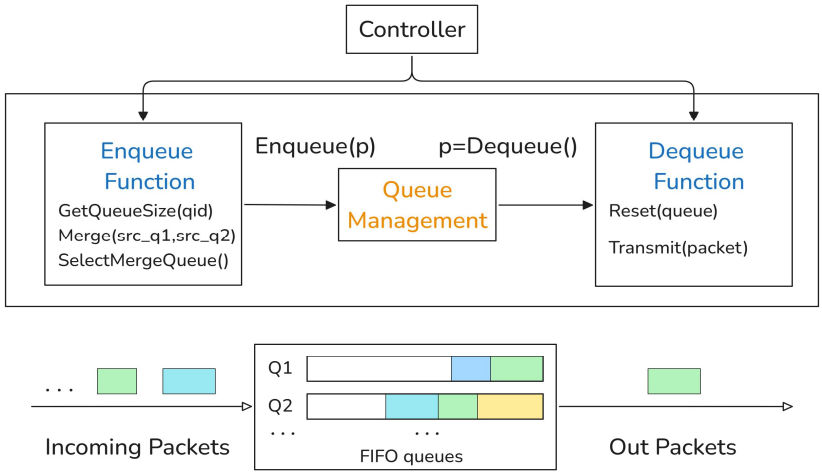

Fig. 4. Merger architecture

3.2 Design Details

The architecture of Merger centers on enqueue and dequeue processes. The enqueue functions manage incoming packets, placing them in corresponding FIFO queues based on priority [9]. If no suitable FIFO queue is available to accept the packet, the Merge operation will be triggered. The dequeue functions selects appropriate packets for transmission to the next hop [11-12].

Algorithm 1: Merge Function

```
Function Merge(src_q1, src_q2):
    while src_q1 and src_q2 are not empty:
        if src_q1.front() < src_q2.front():
            Pop src_q1 and push it in dst_q
        else:
            Pop src_q2 and push it in dst_q
    //Take the remain to dst_q
    while src_q1 is not empty:
        Pop src_q1 and push it in dst_q
    while src_q2 is not empty:
        Pop src_q2 and push it in dst_q
    Set src_q1 to dst_q
    Return src_q2
```

The Merge Function, as outlined in Algorithm 1, is responsible for merging two queues when the system encounters a situation where a new packet cannot be enqueued into any existing queue due to rank constraints or full queues. This function compares the front packets of the two source queues and transfers them to a destination queue based on their ranks [3]. Specifically, it pops the smaller rank packet from the front of one queue and pushes it into the destination queue, repeating this until both source queues are empty.

This merging mechanism effectively combines the packets from two queues into one, thereby freeing up space for the incoming packet. By doing so, it ensures that the system can continue to accommodate new packets without significant disruptions, maintaining the continuity and efficiency of the packet scheduling process. This is particularly important in environments where network conditions can change rapidly, and the ability to adapt quickly is essential for performance.

Algorithm 2: Enqueue Function

```
Function Enqueue(packet):
    //Find valid queue to push in
    for queue in range(Queue_Count):
        if is_valid(queue, packet):
            PushIn(queue, packet)
            Return
    //If can't find, select two queue to merge
    src_q1, src_q2 = SelectMergeQueue()
    target = Merge(src_q1, src_q2)
    PushIn(target, packet)
```

As shown in Algorithm 2, The Enqueue Function, as depicted in Algorithm 2, manages the placement of incoming packets into suitable FIFO queues based on their priority. Upon the arrival of a new packet, it checks if the packet's rank meets the criteria of any queue. If a valid queue is found, the packet is enqueued. otherwise, the Merge operation is triggered to free up space by combining two queues' packets, as detailed in Algorithm 1. This dynamic adjustment through merging ensures that high-priority packets can be accommodated even when queues are full or rank limits prevent direct enqueuing, which is crucial for maintaining packet scheduling efficiency under heavy loads or complex traffic conditions.

Algorithm 3: SelectMergeQueue Function

```
Function SelectMergeQueue():
    //Get two queues with minimun length
    q_1 = queues[0],q_2 = queues[1]
    if length(q_1) > length(q_2):
        swap(q_1,q_2)
    for idx in range(2,Queue_Count):
        current_length = length(queue[idx])
        if current_length < length(q_1):
            q_2 = q_1, q_1 = queues[idx]
        elif current_length < length(q_2):
            q_2=queues[idx]
    return (q_1,q_2)
```

The SelectMergeQueue Function, depicted in Algorithm 3, is responsible for identifying two queues that are most suitable for merging when the system encounters a situation where a new packet cannot be enqueued due to rank constraints. This function

plays a critical role in maintaining scheduling efficiency by dynamically adjusting the queue structure to accommodate high-priority packets.

The function iterates through all available queues and selects the two queues with the smallest lengths. By choosing the shortest queues for merging, it minimizes resource usage and ensures efficient queue management. This mechanism is particularly beneficial under heavy network loads or complex traffic conditions, as it allows the system to adapt quickly and maintain performance.

Algorithm 4: Dequeue Function

```
Function Dequeue():
    //Get the queue with the smallest front
    target = 0
    for queue in range(Queue_Count):
        if queue.front()<target.front():
            target = queue
    packet = target.pop()
    if target is empty:
        reset(target)
    transmit(packet)
```

The Dequeue Function, as shown in Algorithm 4, selects and transmits the highest-priority packet across all queues to the next-hop. It identifies the queue with the smallest front packet rank, ensuring correct transmission order [12]. If transmitting the last packet empties a queue, the queue is reset. This mechanism guarantees low latency and efficient resource utilization, making it ideal for delay-sensitive applications and diverse traffic environments by prioritizing critical packets first.

4 Evaluation

This section aims to evaluate the performance of the proposed Merger programmable packet scheduler, with a focus on latency and flow completion time. By comparing it with existing schedulers, we verify Merger's performance under different network load conditions.

4.1 Experimental Setup

Platform. The experimental platform integrates the NS-3 network simulator. NS-3 precisely emulates packet transmission, latency, and packet loss in networks, while Mininet facilitates the rapid setup of customizable network environments.

Network topologies. The topology is a common leaf-spine topology in data center clusters. It consists of 8 × 8 leaf-spine switches and 128 servers, with each leaf switch connected to 16 servers. All links have a bandwidth of 100 Gbps [10].

Evaluation Metrics. We use normalized FCT and End-to-End delay as the evaluation metric. Normalized FCT scales observed FCT against theoretical minimal completion time to eliminate biases from flow size and network load, enabling cross-scenario performance analysis. The expression for normalized FCT is $\frac{ObservedFct * \text{Bandwitdth}}{\text{Flow\_size}}$. And End-to-End Delay measures the latency of individual packets from source to destination.

4.2 Experimental Results

We deployed the Start - time Fair Queueing (SFQ) algorithm in all packet schedulers due to its simplicity and accuracy. We evaluated the normalized FCT under random and In-cast traffic patterns. As shown in Figure 5, Merger achieved the most idealized normalized FCT among the three schedulers in both random and In-cast traffic patterns. Specifically, Merger reduces normalized FCT by approximately 30%-40% compared to SP-PIFO and Sifter. This significant improvement demonstrates Merger's efficiency in completing data flows, especially in environments with diverse traffic demands [13].

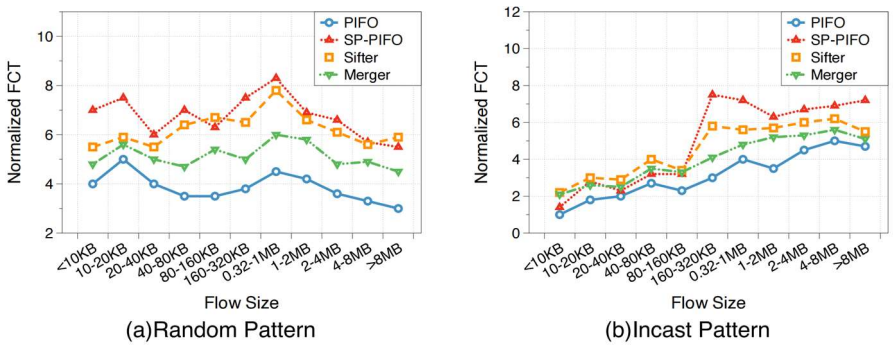

Fig. 5. Normalized FCT with SFQ

Besides, we assessed the End-to-End delay to evaluate the data transmission performance of different packet schedulers. As shown in Fig. 6, in both random and In-cast traffic patterns, Merger achieves a reduction of about 20%-30% in End-to-End delay compared to SP-PIFO and 15%-25% compared to Sifter. This indicates that Merger maintains low latency, making it suitable for delay-sensitive applications [7].

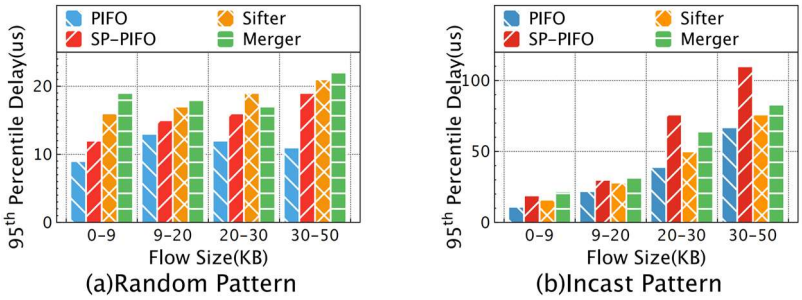

Fig. 6. End-to-End Delay with SFQ

The simulation results show Merger nears ideal PIFO's performance and outperforms SP-PIFO and Sifter in normalized FCT and End-to-End delay. Its practicality, needing no costly hardware changes, makes Merger a better choice. Its design ensures efficient scheduling and resource use, offering a scalable solution for dynamic networks.

5 Conclusion

This paper presented Merger, a programmable packet scheduler that combines merge operation and dynamic queue management. It ensures precise scheduling and efficient queue resource use by maintaining each queue's maximum rank and merging queues

when needed. Experiments show Merger outperforms existing schedulers, achieving significant improvements in normalized FCT and End-to-End delay under various traffic patterns. This makes Merger a superior choice for modern network systems requiring high performance and programmability.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China under Grant 62472050 and Grant 62473146; in part by the Natural Science Foundation of Hunan Province under Grant 2025JJ20070 and Grant 2024JJ3017; and supported by the Science and Technology Project of Hunan Provincial Department of Water Resources under Grant XSKJ2024064-36.

References

1. A. Sivaraman, S. Subramanian, M. Alizadeh, et al. Programmable Packet Scheduling at Line Rate. In Proc. ACM SIGCOMM, 2016.
2. A. G. Alcoz, A. Dietmüller, L. Vanbever. SP-PIFO: Approximating Push-In First-Out Behaviors using Strict-Priority Queues. In Proc. USENIX NSDI, 2020.
3. P. Gao, A. Dalleggio, J. Liu, C. Peng. Sifter: An Inversion-Free and Large-Capacity Programmable Packet Scheduler. In Proc. USENIX NSDI, 2024.
4. B. N. Astuto, M. Mendonça, X. N. Nguyen, et.al. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. IEEE Communications surveys & tutorials 16(3), 1617-1634 (2014).
5. W. Chen, Y. Tian, X. Yu, B. Zheng, X. Zhang. Enhancing Fairness for Approximate Weighted Fair Queueing With a Single Queue. IEEE/ACM Transactions on Networking 32(5), 3901-3915 (2024).
6. V. Shrivastav. Fast, Scalable, and Programmable Packet Scheduler in Hardware. In Proc. ACM SIGCOMM, 2019.
7. J. Hu, S. Rao, M. Zhu, J. Huang, J. Wang, J. Wang*. SRCC: Sub-RTT Congestion Control for Lossless Datacenter Networks. IEEE Transactions on Industrial Informatics, 2024, DOI: 10.1109/TII.2024.3495759
8. Z. Zhang, S. Chen, R. Yao, R. Sun, et al. vPIFO: Virtualized Packet Scheduler for Programmable Hierarchical Scheduling in High-Speed Networks. In Proc. ACM SIGCOMM, 2024.
9. N. K. Sharma, C. Zhao, M. Liu, P. G. Kannan, et al. Programmable Calendar Queues for High-speed Packet Scheduling. In Proc. USENIX NSDI, 2020.
10. J. Hu, Y. He, W. Luo, J. Huang, J Wang. Enhancing Load Balancing with In-network Recirculation to Prevent Packet Reordering in Lossless Data Centers. IEEE/ACM Transactions on Networking, 32(5), 4114-4127 (2024).
11. Z. Yu, C. Hu, J. Wu, X. Sun, V. Braverman, M. Chowdhury, Z. Liu, X. Jin. Programmable Packet Scheduling with a Single Queue. In Proc. ACM SIGCOMM, 2021.
12. C. H. Song, X. Z. Khooi, I. Choi, J. Li, M. C. Chan. Network Load Balancing with In-network Reordering Support for RDMA. In Proc. ACM SIGCOMM, 2023.
13. J. Min, M. Liu, T. Chuph, C. Zhao, A. Wei. Gimbal: Enabling Multi-tenant Storage Disaggregation on SmartNIC JBOFs. In Proc. ACM SIGCOMM, 2021.
14. N. Atre, H. Sadok, J. Sherry. BBQ: A Fast and Scalable Integer Priority Queue for Hardware Packet Scheduling. In Proc. USENIX NSDI, 2024.