Dynamic Per-Flow Queues for TSN Switches*

Wenxue Wu[1], Zhen Li[1], Tong Zhang[3], Xiaoqin Feng[1], Liwei Zhang[1], Xuelong Qi[1], and Fengyuan Ren✉[1,2]

[1]Lanzhou University, China [2]Tsinghua University, China
[3]Nanjing University of Aeronautics and Astronautics, China

***Abstract*—Dynamic Per-Flow Queues (DFQ) extend queues from per-class to per-flow in Time-Sensitive Networking (TSN) switches that overcome large resource consumption by dynamically mapping a fixed number of physical queues to active flows. It can implement per-flow queuing with much less on-chip resource. Compared to brute-force hardware queues, DFQ prototyped on an FPGA, can effectively manage more per-flow queues, allowing for improved priority scheduling with minimal throughput and latency impact.**

***Index Terms*—Time-Sensitive Networking, Per-flow Queues, Flow Isolation, FPGA**

I. Introduction

Time-Sensitive Networking (TSN) is an Ethernet enhancement mechanism to support real-time services. One of the challenges of TSN traffic scheduling comes from the finite eight-queue structure inherited from traditional Ethernet, which makes flow management and scheduling complicated and inflexible.

To deal with this problem, we introduce Dynamic Per-Flow Queues (DFQ) mechanism. DFQ is tailored for shared buffer switches with limited on-chip storage, efficiently managing flow within a fixed number of buffer-constrained queues. Its central strategy involves maintaining and dynamically adjusting the mapping relationship between queues and flows by Flow Mapping Table (FMT) in response to traffic variations in the switch's buffer. The key contributions of this paper are as follows:

- We propose and design DFQ to support per-flow queues with a relatively small number of physical queues through dynamic mapping of FMT.
- We implement DFQ on FPGA devices, which uses much less on-chip recourse than brute-force hardware queues.
- DFQ can generate additional per-flow queues with negligible impact on switch performance, as evaluated in micro-benchmarks.

II. Background

The shared buffer architecture implements a unified buffer across all ports, dynamically allocating memory to improve utilization. This architecture enables switches to schedule buffer indices of frames in egress queues to ensure quality of service (QoS), with greater flexibility than Output Queue (OQ), Input Queue (IQ), Combined Input and Output Queue (CIOQ), etc. The enhanced efficiency and adaptability offered by this approach are the primary reasons why TSN switches [1], [2] predominantly opt for the shared buffer mode.

Current TSN switches [3] manage queue gates via a predefined Gate Control List (GCL), guaranteeing deterministic delivery. However, this mechanism is heavily dependent on precise global clock synchronization. In practical TSN application scenarios, if the frame arrival time at the switch does not align with the gating time, adjustments cannot be made in a timely and flexible manner. The primary constraint is the finite number of class-specific queues. Per-flow queuing can provide a more adaptable scheduling space, yet the constrained on-chip storage resources preclude the allocation of individual queues for each flow.

In TSN scenarios, the flow count is significantly lower than traditional Internet [2]. Time-sensitive traffic in TSN exhibits cyclical and predictable patterns, often exceeding the expected latency in Ethernet bridges [4]. Furthermore, observations from the switch reveal that the number of active flows in the switch is generally limited by the buffer size [5]. For instance, a link with a 1 Gbps capacity and an average packet size of 256 Bytes, considering a 125 µs time window and a 16 KBytes buffer size per port, can support no more than 64 active flows.

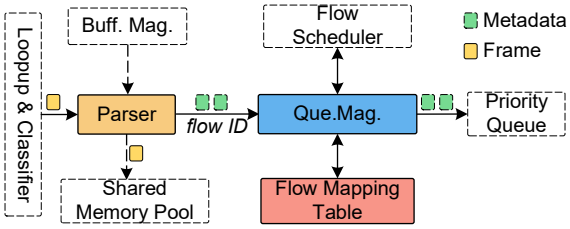


Fig. 1. Overview of DFQ

III. Dynamic Per-Flow Queues

Drawing from the preceding analysis, we propose Dynamic Per-Flow Queues (DFQ) framework deployed in a shared buffer switch. Fig. 1 shows the overview of DFQ. *Parser* is tasked with extracting flow characteristics from the header fields of incoming frames and segmenting these frames into fixed-size cells. *Buff.Mag.* is responsible for allocating addresses to each cell, storing these cells in *Shared Memory Pool* based on these addresses, and generating *Metadata*. Moreover, *Que.Mag.* controls the mapping of *flow IDs* with respective queues in Flow Mapping Table (FMT).

*This work is supported in part by the National Key Research and Development Program of China (No.2022YFB2901404), and by National Natural Science Foundation of China (NSFC) under Grant No. 62132007, 62221003.

✉Corresponding author: Fengyuan Ren, rfy@lzu.edu.cn

979-8-3503-4859-0/DATE24/© 2024 EDAA

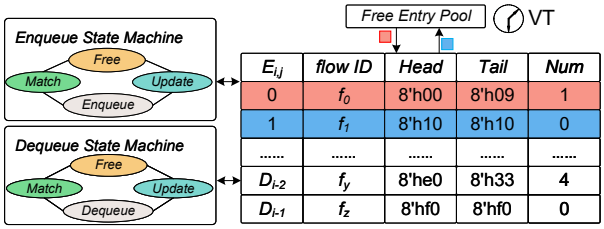


Fig. 2. Que.Mag. controls Flow Mapping Table by *Enqueue* and *Dequeue State Machine* to execute different operations depending on the queue status.

TABLE I
RESOURSE UTILIZATION COMPARISON

Resources	BRAM	LUT	FF
DFQ_32(per port)[1]	47%	16%	11%
DFQ_128(per port)	47%	25%	18%
DFQ_256(per port)	47%	36%	26%
HQ_32(one port)[2]	76%	-	-
RS[3]	19%	9%	6%

[1] 4 fixed per-flow queues per priority per port.
[2] 4 fixed hardware queues per priority per port.
[3] Reference switch with 8 priority queues.

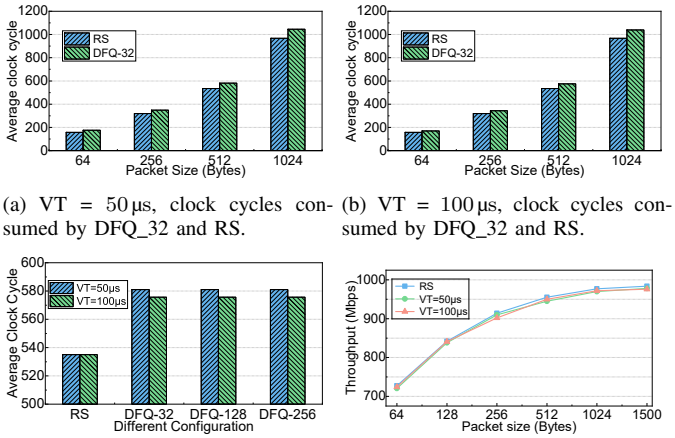
FMT, shown in Fig. 2, records the correspondence between active flows and fixed physical queues. In FMT, each entry represents a physical queue. Upon the arrival of an active flow, FMT allocates a mapping entry (*flow ID* $\rightarrow E_{i,j}$) linking flow ID to queue parameters. These parameters include the head and tail of the linked list queue, as well as the number of frames in the queue, denoted as *Num*.

Specifically, *Que.Mag.* triggers *Enqueue State Machine* to match the appropriate per-flow queue to enqueue. If a corresponding *flow ID* is not located, this indicates the absence of a dedicated queue for the particular flow. Under such circumstances, *Free Entry Pool* generates a new per-flow queue by allocating a new entry. Likewise, if a flow's dedicated queue becomes empty and remains unoccupied for the defined Vacant Time (VT) period after a dequeue operation, *Free Entry Pool* recycles the entry for reuse by subsequent other flows.

IV. Evaluation

We implement a DFQ Verilog prototype and target a Xilinx Kintex-7 series FPGA card. The FPGA boasts a total Block RAM (BRAM) capacity of 2MBytes. Resource utilization, as reported by Xilinx Vivado, is summarized in Table I, showing the difference of different queue configurations. Increasing the number of brute-force hardware queues (HQ) significantly consumes valuable on-chip BRAM and falls short of providing sufficient queues for each port. Conversely, DFQ effectively mitigates resource consumption by using virtual queues, and it can be found that DFQ can only rely on on-chip storage suitable for time-sensitive services to support much more per-flow queues than HQ.

Fig. 3(a) and Fig. 3(b) show hardware clock cycles (avoid the effect of end-device processing delay) for a frame from ingress to egress under various VT. The experiment employed a linear micro-benchmark topology consisting of one switch



(a) VT = 50 µs, clock cycles consumed by DFQ_32 and RS.
(b) VT = 100 µs, clock cycles consumed by DFQ_32 and RS.
(c) Clock cycles consumed by different fixed queues.
(d) Throughput of RS and DFQ_32 with different VT.

Fig. 3. Micro-benchmark evaluation

and two servers, each with a bandwidth of 1 Gbps. The test flow f interval was fixed at 70 µs, utilizing a Round-Robin (RR) scheduling algorithm. For VT=100 µs, DFQ only allocates queue for f once, whereas for VT=50 µs, DFQ allocates queue for f every time because of queue recycle. The results show that the additional clock cycles consumed by DFQ are quite small compared to RS, regardless of VT. Fig. 3(c) demonstrates uniform clock cycle consumption of different numbers of fixed queues in FMT. Furthermore, the efficient pipeline design of DFQ, as evidenced in Fig. 3(d), ensures consistent throughput performance.

V. Conclusion

In this paper, we present DFQ, a system designed to maintain a fixed number of per-flow queues determined by the buffer size and dynamically manage the mapping relationship between queues and flows using FMT. Micro-benchmark experimental results indicate that DFQ markedly reduces resource consumption and can set more per-flow queues in comparison to brute-force HQ while avoiding the impact on the switch itself.

References

[1] Z. Li, H. Wan, Y. Deng, X. Zhao, Y. Gao, X. Song, and M. Gu, "Time-triggered switch-memory-switch architecture for time-sensitive networking switches," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 1, pp. 185–198, 2020.
[2] J. Yan, W. Quan, X. Yang, W. Fu, Y. Jiang, H. Yang, and Z. Sun, "Tsn-builder: enabling rapid customization of resource-efficient switches for time-sensitive networking," in *Proc. 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020.
[3] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, "Ultra-low latency (ull) networks: The ieee tsn and ietf detnet standards and related 5g ull research," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 88–145, 2019.
[4] R. Belliardi, J. Dorr, T. Enzinger, F. Essler, J. Farkas, M. Hantel, M. Riegel, M. Stanica, G. Steindl, R. Wamßer, *et al.*, "Use cases iec/ieee 60802 v1.3," 2018.
[5] A. Kortebi, L. Muscariello, S. Oueslati, and J. Roberts, "Evaluating the number of active flows in a scheduler realizing fair statistical bandwidth sharing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1, pp. 217–228, 2005.