Accurate and Scalable Rate Limiter for RDMA NICs

Zilong Wang, Xinchen Wan, Chaoliang Zeng, Kai Chen
Hong Kong University of Science and Technology

ABSTRACT

Rate limiter is required by RDMA NIC (RNIC) to enforce the rate limits calculated by congestion control. RNIC expects the rate limiter to be accurate and scalable: to precisely shape the traffic for numerous flows with minimized resource consumption, thereby mitigating the incasts and congestions and improving the network performance. Previous works, however, fail to meet the performance requirements of RNIC while achieving accuracy and scalability.

In this paper, we present Tassel, an accurate and scalable rate limiter for RNICs, including the algorithm and architecture design. Tassel first extends the classical WF^{2}Q+ algorithm to support rate limiting in the context of the RNIC scenario. Then Tassel designs a high-precision and resource-friendly rate limiter and integrates it into classical RNIC architecture. Preliminary simulation results show that Tassel precisely enforces the rate limits ranging from 100 Kbps to 100 Gbps among 1 K concurrent flows while the resource consumption is limited.

CCS CONCEPTS

• **Networks** → **Network protocols**; • **Hardware**;

KEYWORDS

Hardware Transport, Congestion Control, Rate Limiter

ACM Reference Format:
Zilong Wang, Xinchen Wan, Chaoliang Zeng, Kai Chen. 2023. Accurate and Scalable Rate Limiter for RDMA NICs. In *7th Asia-Pacific Workshop on Networking (APNET 2023), June 29–30, 2023, Hong Kong, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3600061.3600078

1 INTRODUCTION

RDMA is becoming the defacto transport for high-speed networks in modern data centers [1, 4, 7, 8, 10, 22, 23]. RDMA achieves high throughput, low latency, and low CPU overhead by means of architecture innovations including transport offload and kernel bypass. Transport features such as congestion control (CC), packet ordering, and reliability are fully offloaded to the NIC for optimal performance.

The significance of CC to RDMA continues to grow in modern data centers with the trends of high link rate, low fabric latency, and large infrastructure scale [12, 13, 23]. This requires RDMA NICs (RNICs) to leverage an accurate and scalable rate limiter to precisely enforce the calculated CC rate for numerous flows and shapes the traffic accordingly [15]. Consequently, RNIC is capable of transmitting smoother traffic, thereby reducing network queueing and losses, and also easier to handle the incast [11].

However, designing an accurate and scalable rate limiter for RNICs is challenging, as RNICs are highly sensitive to performance and resource consumption [21, 22]. The complexity of this problem manifests in three aspects: First, the design of the rate limiting algorithm must consider the characteristics of RDMA protocol, which is message-based and where the size of adjacent packets may vary. Second, when designing the rate limiter architecture, we must consider how to integrate it into the common RNIC architecture while compatible with its scheduler module and DMA engine. Finally, the overall solution must satisfy the performance and resource requirements of RNICs. These challenges demand a comprehensive and thoughtful approach to the design and implementation of the rate limiter.

Previous works, however, are unsuitable for RDMA, as they fail to meet the performance requirements of RNICs. PIEO [17] proposes a general scheduling primitive called Push-In-Extract-Out and designs a programmable packet scheduler that supports rate limiting. But the hardware design of this scheduler is too complicated to achieve the desired high clock frequency of RNICs, resulting in decreased performance. SENIC [14] develops a hybrid hardware/software system to improve the packet scheduling efficiency of the kernel. However, the system's hardware design uses token buckets, which causes the traffic burst and hurts the accuracy, while leaving a critical sorting problem unsolved.

In this paper, we propose Tassel, an accurate and scalable rate limiter for RNICs, which addresses the aforementioned challenges via the algorithm (§3.1) and architecture (§3.2) design. Specifically, Tassel extends the classical time-based weighted sharing algorithm WF^{2}Q+ to accommodate the RDMA data flow, such as considering WQE fetching. Tassel

APNET 2023, June 29–30, 2023, Hong Kong, China

ACM ISBN 979-8-4007-0782-7/23/06…$15.00
https://doi.org/10.1145/3600061.3600078

computes the transmission time for each packet according to the size and the rate limit, and schedules the packet that is eligible to send with the smallest finish time of transmission. This rate limiting algorithm realizes accuracy as it guarantees the bounded scheduling delay for each packet. In terms of the architecture, Tassel integrates the rate limiter into the scheduler module. and designs a high-precision, high-performance, and resource-friendly timeline data structure to accomplish accuracy and scalability.

Our preliminary simulation results demonstrate Tassel's accuracy and scalability. Tassel precisely enforces the rate limits ranging from 100 Kbps to 100 Gbps among 1 K concurrent flows, and falls back to weighted sharing when the link is oversubscribed.

2 BACKGROUND AND MOTIVATION

2.1 Need for Rate Limiters in RDMA NICs

RDMA NICs (RNICs) constitute the essential components in high-speed data center networks, as they offload the RDMA transport and provide high-throughput, low-latency network services [4].

Rate limiter is as crucial to RNIC as congestion control is to the high-speed networks. In modern data centers with high link rate, low fabric latency, and large infrastructure scale, the importance of congestion control continues to grow [11, 12]. This is because high speed enables the flows to grab available network capacity more aggressively, easily causing severe congestion and high queueing delay, especially when the large-scale incast happens. RNIC implements the CC mechanism by means of the hardware rate limiter, which enforces the calculated CC rate for each flow (*a.k.a.*, connection, or queue pair (QP))[1] and shapes the traffic accordingly. Consequently, RNIC is capable of limiting the traffic volume and transmitting smoother traffic, thereby mitigating the incasts as well as congestions and improving the network performance [3, 23].

RNICs expect the rate limiter to be accurate and scalable:

- **Accuracy.** Inaccurate rate limiting brings up bursts and exacerbates the network congestion. The rate limiter should accurately compute and inject the inter-packet gaps to enforce rate limiting on each flow given the rate value, which ranges from 100 Kbps to 100 Gbps[2], while as well allowing for fine-grained adjustments instead of restricted fixed values. In addition, the rate limiter should support weighted bandwidth sharing in cases where the aggregate rate exceeds the link rate [14].

[1]We use *flows* and *QPs* interchangeably in the paper.

[2]Supporting small rate limit helps CC to handle large-scale incast in fine granularity. We set the minimum rate limit to 100 Kbps referring to the design of swift [11], and the maximum rate limit equal to the link rate, i.e. 100 Gbps.

```
flow.start_time  = max ( flow.finish_time, system_time )
                                  # if enqueue in empty queue
                 = flow.finish_time                 # if dequeue
flow.finish_time = flow.start_time + flow.l / flow.r

# eligibility evaluation
amongst all flow s.t. ( system_time >= flow.start_time ):
   transmit the packet with smallest finish_time # rank sorting
```

Figure 1: WF^{2}Q+ rate limiting algorithm. l is the length of the packet at the head of the flow queue. r is the configured rate for this flow.

- **Scalability.** In RNIC, the number of active flows may reach tens of thousands scale [7], while the computing and storage resources are restricted and scarce [22]. Thus, RNIC desires a scalable rate limiter that supports pacing tens of thousands of concurrent flows accurately while meeting the hardware resource constraints.

Current commercial RNICs support around 1 K rate limiters with a deviation of 10% from the target rate limit [2]. The limited capacity of the RNICs causes multiple flows to share the rate limiters, thereby decreasing overall performance.

2.2 Rate Limiter Model

2.2.1 Time-based Rate Limiting Algorithm. Time-based rate limiting algorithms such as Worst-case Fair Weighted Fair Queueing (WF^{2}Q+ [5]) provide bounded delay of the packet scheduling and can be used for accurate rate limiting [14–19].

As illustrated in Figure 1, WF^{2}Q+ (non-work conserving version [14]) computes the start and finish transmission time for each packet in a flow according to the packet size and the rate limit while maintaining the system time globally. To transmit the next packet among flows, WF^{2}Q+ first filters *eligible* packets whose start time are less than or equal to the current system time (*eligibility evaluation*) and then schedules the one with the smallest finish time (*rank sorting*).

We then introduce a rate limiter model to analyze the challenges of deploying the above algorithm into RDMA scenario while meeting the performance and resource requirements of RNIC.

2.2.2 Rate Limiter Model for RNIC. Figure 2 shows a rate limiter model for RNIC. RNIC consists of tens of thousands of queue pairs (QPs). The user posts a work queue element (WQE) into a QP to issue a request. Each WQE corresponds to a message which consists of multiple packets, and messages in the same QP are recognized as a flow. WQEs contain the message size (64 B ∼ 2 GB), and the rate limiter must process the WQE to get the varying packet size and then compute the transmission time.

A QP is either active when it contains WQEs or inactive otherwise. Rate limiter filters the active QPs from tens of thousands of QPs in host memory and computes the

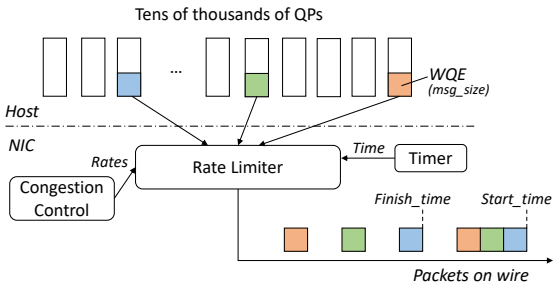


Figure 2: Rate limiter model for RNIC. RNIC queries the activity of each QP, evaluates the eligibility, sorts the eligible packets by their finish time, and transmits the smallest one.

send/finish time based on their WQE sizes and rate limits. Rate limiter then evaluates the eligibility of active QPs using the system time maintained in the timer and transmits packets with the smallest finish time.

We conclude three challenges of designing an accurate and scalable rate limiter as follows:

- **Challenge #1**: In the algorithm aspect, the conditions to judge whether to send a QP's packet are too complicated, as they involve activity, eligibility, and time-consuming factor, *i.e.*, finding the smallest finish time. Once a QP is scheduled but fails to satisfy all conditions, the scheduling does not take effect but wastes time, thereby degrading the performance. Moreover, identifying the minimal value is quite challenging because it requires an ordered list and entails balancing the fundamental trade-off between time complexity and hardware resource consumption.
- **Challenge #2**: In the architecture aspect, the PCIe round-trip latency between RNIC and host memory is high (greater than 1 µs). However, rate limiter needs to read the host WQE per scheduling iteration to calculate the inter-packet interval, as the WQE and adjacent packet sizes may vary. As a result, without careful design, the high latency will decrease the scheduling frequency and hurt the performance. Additionally, common batch operations used to hide PCIe latency can cause burst and reduce the accuracy of the rate limiter.
- **Challenge #3**: In terms of practicality, implementing rate limiter in RNIC poses challenges both in resource and performance constraints. Due to the large number of QPs to support but limited computing and storage resources in RNIC, common methods like allocating buffers to store WQEs/packets for each QP or using dedicated token bucket are not feasible, which, however, are widely used in Ethernet NICs [22]. In addition, RNIC operates at a high clock frequency (250 MHz in FPGA-based RNIC [22]) to achieve high performance. However, if the hardware design of the rate limiter is too complicated, it may be impractical, decrease the clock rate, and hurt the performance of RNIC.

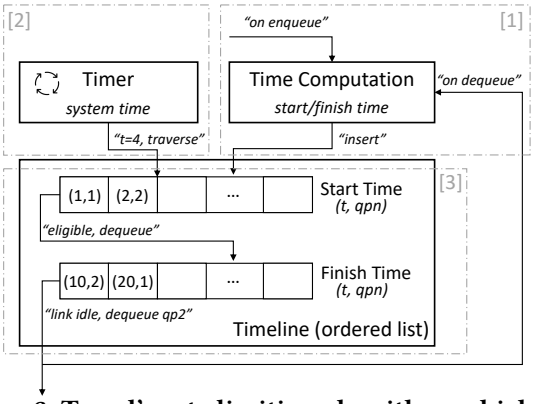


Figure 3: Tassel's rate limiting algorithm, which consists of three functionalities: (1) compute start/finish time and insert into ordered timeline; (2) update system time and traverse the timeline; and (3) transmit the eligible packet with the smallest finish time.

3 TASSEL DESIGN

We propose Tassel, an accurate and scalable rate limiter for RNICs, which addresses the aforementioned challenges via the algorithm and architecture design.

3.1 Tassel's Rate Limiting Algorithm

Tassel aims to precisely enforce a wide range of rate limits for numerous QPs and support fine-grained rate adjustments while consuming limited resources. In addition, when the aggregated rate limits exceed the link capacity and oversubscribes the link, Tassel should adjust the rate proportionally following the weighted sharing manner.

As shown in Figure 3, Tassel's rate limiting algorithm consists of three parts: (1) computation and insertion; (2) timing and monitoring; and (3) sorting and transmission. In brief, Tassel computes the start/finish time of the first few packets for each QP and inserts them into a time-basis sorted array, *i.e.*, timeline. Meanwhile, Tassel records the system time in a global timer and monitors the timeline to trigger the eligibility evaluation. Then Tassel transmits the packet which is eligible and has the smallest finish time in the timeline. Next, we discuss the three parts in detail.

3.1.1 Computation and Insertion. Tassel computes the packet transmission time and QP rescheduling time for the QP which is either enqueued or dequeued. An enqueued QP means that it is newly activated and has new WQEs to process. And a dequeued QP means that it has just been scheduled to transmit a packet but still remains active and should rejoin the scheduling

The packet transmission time is the start and finish time for a QP's first few packets (assuming n packets) within at most n WQEs. Tassel first fetches n WQEs of that QP from the host to calculate the length of each packet in the n WQEs, queries the rate limit configured for that QP, and processes

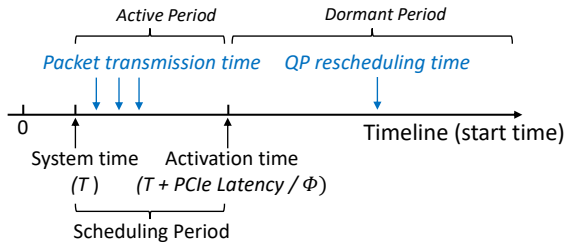


Figure 4: Timeline of start time consists of two periods: *active period* that contains packet transmission time and *dormant period* that contains QP rescheduling time.

the time computation following Algorithm 1. Then Tassel filters the *urgent* packets (assuming m packets) that can be transmitted within the scheduling period, which is equal to the PCIe latency (1 µs). Tassel only stores the timestamps and WQE info for these *urgent* packets, as the information of the others can be retrieved in the next iteration after fetching the WQEs again and thus can be discarded for now to save memory resources.

The QP rescheduling time is the time to reactivate the QPs which have finished transmitting their *urgent* packets and wait to rejoin the scheduling, fetch WQEs, and compute the packet transmission time for the new round. As Figure 4 shows, the QP rescheduling time is actually the smallest packet transmission time that falls in the dormant period:

$$reschedule\_t = max(pkt\_i\_start\_t, activation\_t)$$
$$i = min(m+1, n), n \text{ packets in total}$$

After the computation, Tassel inserts the packet transmission time for m urgent packets and the QP rescheduling into the timeline of start time, as shown in Figure 3.

3.1.2 Timing and Monitoring. Tassel records the system time inside a global timer and updates it per cycle, *i.e.*, 4 ns for 250 MHz clock frequency. It realizes strict rate limiting with a fallback to the weighted sharing by means of adjusting the timing rate. When there is no oversubscription, Tassel increases the system time by one granularity (4 ns) each time. When the link is oversubscribed, Tassel slows down the system time proportionally. This allows RNIC to transmit more packets per actual unit time, thus preventing packet accumulation and queuing caused by oversubscription.

Tassel adjusts the timing rate according to the rate oversubscription factor Φ, which is defined as the sum of the active QPs' rate limits divided by the link capacity. After the time period δ, Tassel updates the system time as follows:

$$T(t+\delta) = T(t) + \delta \times \max(1, \Phi)$$

Meanwhile, Tassel keeps monitoring the timeline of start time. Tassel recognizes the eligible packets whose timestamps are smaller than the system time T in the active period. It then sorts the eligible packets, schedules the one with the smallest finish time, fetches data via PCIe, and transmits the packets (§3.1.3). Tassel also recognizes the QPs whose QP rescheduling time is less than the activation time $T + PCIe\_Latency/\Phi$ in the dormant period, and reschedules them to fetch WQEs for the next round's time computation.

3.1.3 Sorting and Transmission. As shown in the bottom half of Figure 3, Tassel leverages the ordered timeline to support eligibility evaluation and rank sorting in order to transmit the eligible packet with the smallest finish time.

Specifically, Tassel's timeline consists of two ordered lists to sort the start time and finish time separately. The eligible packets recognized by the timer (eligibility evaluation) are dequeued from the timeline of start time and enqueued into the timeline of finish time (rank sorting). Whenever the link is idle, Tassel dequeues and transmits the head packet from the timeline of finish time.

Note that we only dequeue a packet from the timeline of finish time when the link is idle. Otherwise, if we dequeue a packet with the smallest finish time at the time when the link is busy, this packet will get blocked and wait until the link becomes idle. During the waiting period, however, the increasing system time may induce new packets with smaller finish time to become eligible. These newly eligible packets should be transmitted first when the link is idle but are blocked in this case. Therefore, scheduling and dequeuing prematurely may result in wrong scheduling order and hurt the accuracy, and hence should be carefully handled. Besides, Tassel deduces the link state by computing the transmission time of the last packet $last\_packet\_length/link\_rate$.

In summary, Tassel's time-based rate limiting algorithm guarantees accuracy. It bounds the scheduling delay for each packet among QPs [5]. Next, we present Tassel's hardware design that realizes this algorithm and discusses the design complexity, performance, and resource consumption that fit the strict requirements of RNIC.

3.2 Tassel's Rate Limiter Architecture

Tassel designs a cache-free architecture, as shown in Figure 5, that can do fast scheduling and enforce accurate rate limiting among tens of thousands QPs with limited on-chip computing and memory requirements.

3.2.1 Architecture Overview. The *Event MUX* (*EMUX*) module aggregates all scheduling-related events, including (1) host doorbell indicating some QP is active and has new WQEs to process; (2) rates and credits update from the *congestion control* module to control QPs' data transmission; and (3) dequeue event from *timeline* to indicate that the QP is ready to schedule again. These events trigger *EMUX* to change the *scheduling states* stored in *QP Context (QPC)*. QPs that are active, ready, and have credits can enqueue the *schedule queue* to evaluate the eligibility and rank order. *EMUX* helps

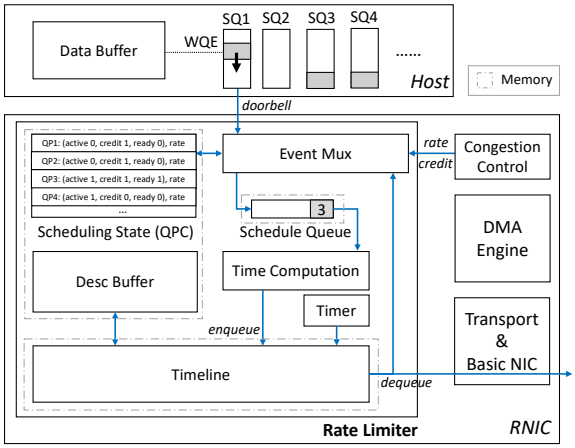


Figure 5: Rate limiter architecture in RNICs.

process complex events while the *schedule queue* filters valid QPs and prevents ineffective scheduling, which effectively addresses Challenge #1.

FIFO-based *schedule queue* maintains a list of QPs ready for scheduling. Tassel polls *schedule queue*, dequeues the head QP, and fetches from that QP a given amount of WQEs via *DMA Engine*. These WQEs are then processed to compute the packet transmission and QP rescheduling times. The urgent packets are then inserted into the global *timeline*. Meanwhile, the descriptors (*i.e.*, WQE info) of urgent packets are stored in *Desc Buffer*, and at most k descriptors for all QPs are stored. The capacity of *Desc Buffer* determines the maximal message rate of RNIC, which will be discussed in §3.2.3.

Meanwhile, Tassel queries the global *timer* to compare current system and activation time with the packet transmission and QP rescheduling time recorded in *timeline*. For packet transmission, Tassel dequeues the packet, consumes its descriptor in Desc Buffer, reads the packet data from the host via *DMA engine*, then appends the RDMA header in the *Transport* module, and transmits it to the network through *Basic NIC*. For QP rescheduling, Tassel dequeues the QP and informs EMUX that the QP can be rescheduled and is ready to fetch WQEs for the next round.

3.2.2 Hardware Design of Timeline. The essence of *timeline* in Tassel is the ordered list. However, it is challenging to implement this data structure in hardware as it presents the tradeoff between time complexity and resource consumption.

In the context of high-performance RNIC, the ordered list is expected to support enqueuing and dequeuing in O(1) time. If this is not achieved, the high operation latency between scheduling iterations will significantly degrade the performance. Meanwhile, the design of ordered list is desired to be scalable and practical. To this end, we combine the high-precision timing wheel [20] and the scalable pipelined heap (P-heap [6]) to achieve accuracy and scalability simultaneously while meeting the performance requirements with the support of constant operation time, as shown in Figure 6.

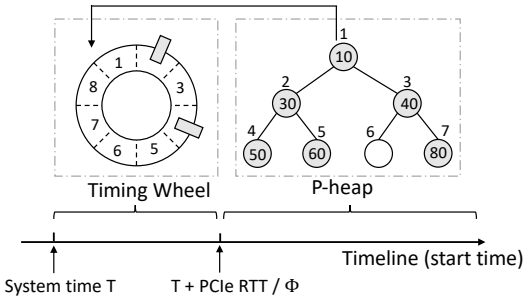


Figure 6: The data structure of timeline in Tassel.

Tassel manages the active period and dormant period using Timing wheel and P-heap, respectively, in order to match each of their needs.

Timing Wheel. Timing wheel is a circular array of slots. It uses time as the basis and each slot represents a time interval. Packets can be inserted into the corresponding slots according to their timestamps in O(1) time, which fits the RNIC performance requirement.

Tassel adopts timing wheel only for performance-sensitive *urgent* packets which desire high precision and extremely high enqueue/dequeue speed (every 4 clocks to transmit a packet) to balance the tradeoff between accuracy and memory consumption. Fine-grained time granularity and large time range contribute to high accuracy of timing wheel, but result in enormous time slots and thus unacceptable memory consumption. Here we leverage the limited time span characteristic of active period for designing our timing wheel. Specifically, Tassel set the time granularity to 4 ns, *i.e.*, 1 clock with 250 MHz frequency, and the time range to 1 µs, *i.e.*, the PCIe latency. In this way, the timing wheel only occupies 250 slots, which is resource-friendly and preserves the highest accuracy for the rate limiting.

In terms of collision [15], thanks to the small number of slots in timing wheel, Tassel can leverage registers to find a suitable location for the conflicting timestamps rapidly.

Pipelined heap (P-heap). We leverage P-heap [6] to manage the QP rescheduling time, which is less performance-sensitive but higher resource-sensitive. P-heap maintains a min heap and enables the pipelining of the enqueue and dequeue operations, thereby allowing these operations to execute in essentially constant time.

P-heap stores and sorts the QP rescheduling time. It can be viewed as a complete binary tree (see Appendix A). P-heap supports pipelined operations of *enqueue*, *dequeue*, and *enqueue-dequeue*. Each operation consumes *height* cycles, while P-heap can start a new operation every two cycles. Each operation only accesses two adjacent layers of the tree and thus operations with a gap are independent and can be pipelined.

In summary, P-heap's constant operation time fits Tassel's performance demand. Tassel can dequeue the rescheduled QP immediately and enqueue QPs in a non-blocking manner,

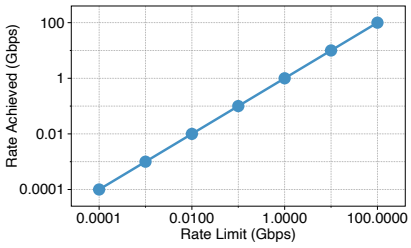
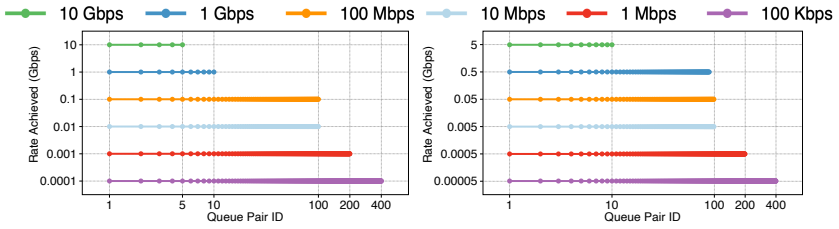


Figure 7: Accuracy. Tassel supports accurate rate limiting for a single QP, ranging from 100 Kbps to 100 Gbps.



(a) No oversubscription. **(b) Oversubscription. Φ=2.**

Figure 8: Scalability. (a) Tassel maintains the accuracy of different rate limits with numerous QPs. (b) Tassel can fall back to weighted fair sharing when the link is oversubscribed.

making the scheduling efficient. In terms of resource requirements, P-heap is scalable as it only stores several bytes of metadata per QP.

3.2.3 Cache-free Scheduler. Tassel adopts a cache-free scheduler via the fetch-and-drop policy. When a QP is scheduled, the DMA engine fetches up to n WQEs from that QP and filters m *urgent* packets. Tassel then stores the timestamps and the descriptors (WQE info) of *urgent* packets in the *Desc Buffer* with capacity k. Other unused WQEs are dropped instead of being cached in RNIC, and will be fetched again next time. This fetch-and-drop policy allows a cache-free architecture and improves resource efficiency.

There are two critical parameters n and k: n is the maximum number of WQEs, and k is the capacity of the desc buffer. A larger n can better hide the PCIe latency and achieve a higher message rate at the cost of more PCIe bandwidth waste, while a smaller n performs inversely. Tassel set n to 8 to balance the tradeoff. With this setting, the maximum message rate of a single QP is 8 Mrps considering 1 µs PCIe latency. k implies the number of packets that can send within this scheduling iteration and corresponds to the maximal aggregate message rate of multiple QPs that RNIC can achieve. Tassel sets k to 60 to achieve the performance of the state-of-the-art RNIC while minimizing the memory consumption.

3.2.4 Scalability Analysis. Tassel is a scalable design as it limits the memory and computing resource consumption as much as possible.

Tassel's memory consumption is negligible compared to 10 Mb total memory size or 375 B QPC [9], making it a scalable design for RNICs. Figure 5 enumerates all memory-consuming modules surrounded by dotted lines. Specifically, Tassel occupies 17 bytes for each QP, *i.e.*, 5 B for the scheduling states, 2 B in the schedule queue, and 10 B in the timeline module. Besides, Tassel consumes 480 bytes to store the packet descriptors, which is a constant value unrelated to the QP number. When supporting 10 K QPs, Tassel consumes 166.5 KB on-chip SRAM in total theoretically. In terms of computing resources, Tassel uses shared instead of dedicated processing logic for QPs. Thus, the consumption of computing resources is limited and unrelated to the QP number.

4 EVALUATION

In this section, we present the preliminary simulation results of Tassel. Our results reveal that:

- Tassel achieves high accuracy: it precisely enforces the rate limits ranging from 100 Kbps to 100 Gbps, and shares bandwidth proportionally when the link is oversubscribed.
- Tassel achieves high scalability: it supports accurate rate limiting for thousands of concurrent QPs while consuming limited on-chip resources.

We evaluate Tassel using a Python-based simulator that models the behavior of the RNIC at a link rate of 100 Gbps. We measure the accuracy by timestamping every packet with a clock resolution of 4ns and retrieving the inter-packet timestamp difference for packets within each QP.

Accuracy. We measure Tassel's accuracy by configuring a QP's rate limit from 100 Kbps to 100 Gbps. As shown in Figure 7, Tassel enforces any given rate limit accurately, including the small rate demanding the timeline to support a large time range and the high rate demanding the timeline to support high precision. Our results demonstrate the efficiency of Tassel's timeline design which combines the timing wheel and P-heap.

Scalability. We initiate 1 K concurrent QPs with different rate limits to evaluate Tassel's scalability. The assigned rate limits range from 100 Kbps to 100 Gbps, similar to Figure 7. When the number of high-rate QP we set is small, and the link is not oversubscribed, as shown in Figure 8a, Tassel preserves high accuracy for different rate limits. When we increase the number of high-rate QPs, making the link oversubscribed (the rate oversubscription factor $\Phi = 2$), Tassel can proportionally adjust the rate limits among QPs according to the original rate limit and Φ, as shown in Figure 8b.

ACKNOWLEDGMENTS

We would like to thank our anonymous reviewers for their valuable comments. This work is supported in part by the Key-Area Research and Development Program of Guangdong Province (2021B0101400001), the Hong Kong RGC TRS T41-603/20-R, and GRF-16213621.

REFERENCES

[1] 2014. Supplement to InfiniBand architecture specification volume 1 release 1.2.2 annex A17: RoCEv2 (IP routable RoCE). https://www.infinibandta.org/specs. (2014).

[2] 2020. NVIDIA MLNX_OFED Documentation Rev 5.1-0.6.6.0. https://download.lenovo.com/servers/mig/2020/09/14/22700/mlnx-lnvgy_dd_nic_cx.ib-5.1-0.6.6.0-0_rhel8_x86-64.pdf. (2020).

[3] Mina Tahmasbi Arashloo, Alexey Lavrov, Manya Ghobadi, Jennifer Rexford, David Walker, and David Wentzlaff. 2020. Enabling programmable transport protocols in high-speed NICs. In *Proc. NSDI*.

[4] Wei Bai, Ankit Agrawal, Ameya Bhagat, Mahmoud Elhaddad, Neetha John, Jitu Padhye, Madhav Pandya, Krishan Kumar Attre, Gowri Bhaskara, Lei Cao, et al. 2023. Empowering Azure Storage with 100× 100 RDMA. In *Proc. NSDI*.

[5] Jon CR Bennett and Hui Zhang. 1997. Hierarchical packet fair queueing algorithms. *IEEE/ACM Transactions on networking* (1997).

[6] Ranjita Bhagwan and Bill Lin. 2000. Fast and scalable priority queue architecture for high-speed network switches. In *Proc. INFOCOM*.

[7] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, et al. 2021. When Cloud Storage Meets RDMA. In *Proc. NSDI*.

[8] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over commodity ethernet at scale. In *Proc. SIGCOMM*.

[9] Anuj Kalia, Michael Kaminsky, and David Andersen. 2019. Datacenter RPCs can be general and fast. In *Proc. NSDI*.

[10] Xinhao Kong, Yibo Zhu, Huaping Zhou, Zhuo Jiang, Jianxi Ye, Chuanxiong Guo, and Danyang Zhuo. 2022. Collie: Finding Performance Anomalies in RDMA Subsystems. In *Proc. NSDI*.

[11] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan MG Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, et al. 2020. Swift: Delay is simple and effective for congestion control in the datacenter. In *Proc. SIGCOMM*.

[12] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. 2019. HPCC: High precision congestion control. In *Proc. SIGCOMM*.

[13] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. 2018. Revisiting network support for RDMA. In *Proc. SIGCOMM*.

[14] Sivasankar Radhakrishnan, Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, George Porter, and Amin Vahdat. 2014. SENIC: Scalable NIC for end-host rate limiting. In *Proc. NSDI*.

[15] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. 2017. Carousel: Scalable traffic shaping at end hosts. In *Proc. SIGCOMM*.

[16] Ahmed Saeed, Yimeng Zhao, Nandita Dukkipati, Ellen Zegura, Mostafa Ammar, Khaled Harras, and Amin Vahdat. 2019. Eiffel: Efficient and flexible software packet scheduling. In *Proc. NSDI*.

[17] Vishal Shrivastav. 2019. Fast, scalable, and programmable packet scheduler in hardware. In *Proc. SIGCOMM*.

[18] Anirudh Sivaraman, Suvinay Subramanian, Mohammad Alizadeh, Sharad Chole, Shang-Tse Chuang, Anurag Agrawal, Hari Balakrishnan, Tom Edsall, Sachin Katti, and Nick McKeown. 2016. Programmable packet scheduling at line rate. In *Proc. SIGCOMM*.

[19] Brent E Stephens, Aditya Akella, and Michael M Swift. 2019. Loom: Flexible and Efficient NIC Packet Scheduling.. In *Proc. NSDI*.

[20] George Varghese and Tony Lauck. 1987. Hashed and hierarchical timing wheels: Data structures for the efficient implementation of a timer facility. In *Proc. SOSP*.

[21] Xizheng Wang, Guo Chen, Xijin Yin, Huichen Dai, Bojie Li, Binzhang Fu, and Kun Tan. 2021. StaR: Breaking the Scalability Limit for RDMA. In *Proc. ICNP*.

[22] Zilong Wang, Layong Luo, Qingsong Ning, Chaoliang Zeng, Wenxue Li, Xinchen Wan, Peng Xie, Tao Feng, Ke Cheng, Xiongfei Geng, et al. 2023. SRNIC: A scalable architecture for RDMA NICs. In *Proc. NSDI*.

[23] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion control for large-scale RDMA deployments. In *Proc. SIGCOMM*.

APPENDIX A P-HEAP

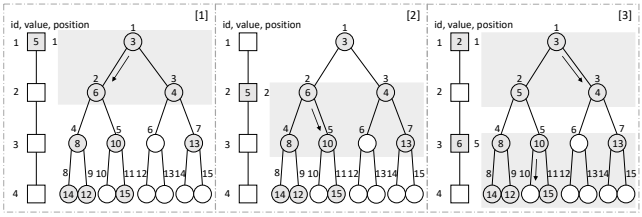


Figure 9: An example of pipelined enqueue operation in P-heap [6].

Figure 9 illustrates an example of pipelined enqueue operations of a 4-level P-heap. The size of P-heap is fixed and equals to the QP number (*e.g.*, 10 K). This is different from the conventional binary heap whose size may vary as long as it stays an almost complete binary tree. The height of P-heap is the logarithm of its size.