QCluster: Clustering Packets for Flow Scheduling

Tong Yang[†‡]
Peking University

Jizhou Li[†]
Peking University

Yikai Zhao[†]
Peking University

Kaicheng Yang[†]
Peking University

Hao Wang[§]
Hong Kong University of Science and Technology

Jie Jiang[†]
Peking University

Yinda Zhang[†]
Peking University

Nicholas Zhang[¶]
HUAWEI

ABSTRACT

Flow scheduling is crucial in data centers, as it directly influences user experience of applications. According to different assumptions and design goals, there are four typical flow scheduling problems/solutions: SRPT, LAS, Fair Queueing, and Deadline-Aware scheduling. When implementing these solutions in commodity switches with limited number of queues, they need to set static parameters by measuring traffic in advance, while optimal parameters vary across time and space. This paper proposes a generic framework, namely QCluster, to adapt all scheduling problems for limited number of queues. The key idea of QCluster is to cluster packets with similar weights/properties into the same queue. QCluster is implemented in Tofino switches, and can cluster packets at a speed of 3.2 Tbps. To the best of our knowledge, QCluster is the fastest clustering algorithm. Experimental results in testbed with programmable switches and ns-2 show that QCluster reduces the average flow completion time (FCT) for short flows up to 56.6%, and reduces the overall average FCT up to 21.7% over state-of-the-art. All the source code in ns-2 is available in Github [45].

CCS CONCEPTS

• **Networks → Programmable networks**.

KEYWORDS

Datacenter Networks, Flow Scheduling, Queue Clustering

ACM Reference Format:
Tong Yang, Jizhou Li, Yikai Zhao, Kaicheng Yang, Hao Wang, Jie Jiang, Yinda Zhang, and Nicholas Zhang. 2022. QCluster: Clustering Packets for Flow Scheduling. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3485447.3511980

[†]School of Computer Science, and National Engineering Laboratory for Big Data Analysis Technology and Application, Peking University, China.
[‡]Peng Cheng Laboratory, Shenzhen, China.
[§]Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong.
[¶]Huawei Technologies Co., Ltd. Theory Lab, China.
This work is supported by Key-Area Research and Development Program of Guangdong Province 2020B0101390001, National Natural Science Foundation of China (NSFC) (No. U20A20179).

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France.

ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3511980

1 INTRODUCTION

1.1 Background and Motivation

Given some flows/packets in a node, flow scheduling is to decide the forwarding sequences of packets for some optimization goals, such as flow completion time, fairness, or meeting deadlines. Flow scheduling has been a hot topic in data centers ([8, 10, 11, 16, 30, 32, 33, 37–39, 42, 55, 62]), because it directly determines bandwidth usage, latency, and Quality of Service of applications.

According to different optimization goals and assumptions, there are typically four kinds of scheduling problems/solutions: SRPT (Shortest Remaining Processing Time first, *e.g.*, pFabric [8]), LAS (Least Attained Service, *e.g.*, PIAS [10], Auto [33]), Fair Queueing (*e.g.*, Nagle [41], BR[18], AFQ [48]), and Deadline-Aware Scheduling (*e.g.*, pFabric [8] with Earliest-Deadline-First (pFabric-EDF), D^3 [61], PDQ [26] and Karuna [13]). These works have made great contributions, and can achieve near-optimal or excellent performance when there are a great number of queues in each egress port. For example, pFabric [8] achieving near optimal performance, assumes there are infinite number of queues. Another example is Fair Queueing. The earlier work, Nagle [41], assigns each flow one queue to achieve the theoretical Fair Queueing, requiring a great many queues. Instead, Bit by bit round robin [18] uses one preemptive queue to conduct the scheduling. However, BR is still far from practice, and the preemptive queue can only be approximately implemented with multiple queues, which is done by approximate Fair Queueing (AFQ) [48]. However, AFQ needs to rotate the queue priorities, which has not been achieved in current switches.

The number of queues in commodity switches is very limited, *e.g.*, $k = 8$ queues, and therefore the above works must be adapted to a limited queue version for practical use. To adapt for k queues, one commonly used approach is to measure traffic in advance. Specifically, one first builds a measurement system to collect traffic from switches or end hosts, analyzes the statistics, and makes many tests to find appropriate parameters. However, when traffic changes and mismatches the parameters, the performance could degrade a lot. For example, the authors of PIAS show that when thresholds mismatch traffic, the flow completion time (FCT) could be degraded by 38% [10, 33]; For another example, when implementing pFabric in $k = 8$ queues, the FCT of pFabric could be degraded by 30% [8, 33]. However, the optimal parameters often vary across time and space, and the traffic distribution in network changes frequently and quickly. Measuring traffic in advance cannot adapt to the quick

change of traffic. Therefore, it is desirable but challenging to adapt existing solutions for limited number of queues without measuring in advance. The design goal is to devise a framework to address this challenge for all existing scheduling problems.

1.2 Our Solution

Our insight is that adapting existing solutions for k queues is actually a clustering problem: clustering packets into queues. The k queues in current commodity switches are first-in-first-out (FIFO). If packets with drastically different weights or properties are placed into one queue, the performance will be poor because of FIFO. For example, if a small flow and a large flow are placed into the same queue, the small flow will be blocked by the large flow and the overall FCT will be large. For another example, if a common flow and a flow with a deadline are placed in one queue, the deadline may be missed. Therefore, our insight is that packets in the same queue should have similar weights/properties, and this is actually a clustering problem, and we name it the ***Queue Clustering*** problem.

Queue Clustering has the following two requirements that existing clustering problems often do not have. 1) The clustering speed needs to catch up with the line rate, *e.g.*, 3.2 Tbps, and no existing clustering algorithm can achieve this speed; 2) Packet disorder should be avoided. Due to the above special requirements, existing clustering algorithms [35, 36, 46] cannot be directly used, and this paper proposes the ***QCluster*** to cluster packets with similar weights/priorities into the same queue.

QCluster uses packet weight and property as features. For flows with special properties, such as deadline or time sensitiveness, they should be in different clusters/queues from other flows. For flows with the same property, we cluster them according to the packet weight. Inspired by k-means [35], we maintain the average packet weight for each queue, and call it *queue weight*. Given an incoming packet, we compare the packet weight with the k queue weights to choose a queue. The packet weight is recorded and updated in the Scheduling Count-min sketch (see details in Section 3.2), which also records timestamp and last queue ID. For different scheduling problems, packet weight has different definitions, and different dequeuing policy should be chosen. For LAS, we define packet weight as the number of bytes sent, and use strict priority to dequeue packets. We also propose an adaptive method (see detail in §3.3) to make high-priority queue have fewer packets so that the probability that small flows are blocked by large flows will be reduced. For Fair Queueing, we also define packet weight as the number of bytes sent, but use round robin to dequeue packets. For SRPT and Deadline-Aware, we implement them in ns-2, but not in our testbed because these two policies need to know the remaining flow size that the current TCP protocol does not support.

During the clustering process, the latter packets of a flow could be placed into a higher-priority queue, and thus packet disorder could happen. Existing solutions (*e.g.*, pFabric) also have this problem. To avoid packet disorder, we propose the PDA algorithm. Our key idea is that given an incoming packet a_{now} of flow a, only if all previous packets of a are not in any queue of this switch, we can place a_{now} to any queue and packet disorder will not happen; Otherwise, we need to schedule this packet according to the state of the previous packet and the scheduling policy.

Key Contributions:
1) We propose the QCluster to adapt all existing scheduling algorithms for limited number of queues (§3). In QCluster, we propose the Scheduling Count-Min sketch to record all necessary information, and propose the PDA algorithm to avoid packet disorder.
2) We apply QCluster to four typical scheduling policies (SRPT, LAS, Fair Queueing, and Deadline-Aware Scheduling) as case studies.
3) We implement QCluster in Tofino switch and build a testbed (§5). We also conduct large-scale simulations using ns-2.
4) Extensive experimental results on a testbed and simulators show that QCluster can well adapt existing scheduling solutions to limited number of queues, achieving similar or better performance (§6).

2 BACKGROUND AND RELATED WORK

Due to the significance of flow scheduling, there are a great many works in the literature, and we introduce them briefly.
1) SRPT: SRPT (shortest remaining processing time first) assumes that the remaining size of each flow is known, and lets the flow with the smallest remaining size go first to minimize FCT. Typical solutions include pFabric [8], Homa [11], and more [21]. In pFabric [8], the smaller the remaining flow size is, the higher priority a packet gets. And for dequeuing, the switch should find the earliest packet from the flow with the highest priority. However, this is hard to be deployed in commodity switches. A recent work, Homa [11], also belongs to this kind.
2) LAS: LAS (least attained service first) lets the flow with the smallest bytes sent go first. Typical solutions include PIAS [10] and AuTO [33]. PIAS [10] can achieve small FCT by carefully setting the thresholds for each queue according to the flow size distribution and network load. Using the handcrafted thresholds for fixed distribution, PIAS achieves very small FCT. However, if the distribution changes, the FCT will drop significantly [33].
3) Fair Queueing: Fair Queueing was first introduced by Nagle [41], which owns some good characteristics compared to FCFS (Fisrt Come First Serve). To achieve per-flow fairness, all active flows in the switch should have the same priority to use the bandwidth. Typical solutions include Nagle [41], RFQ [12], BR(bit-by-bit round robin) [18], Gearbox [3], and AFQ [48].
4) Deadline-Aware Scheduling: In data centers, some flows could have deadlines, and are called *deadline flows*. The goal of Deadline-Aware scheduling is to meet the deadlines of deadline flows first, and then minimize the FCT of non-deadline flows. Typical solutions include pFabric-EDF [8], D^3 [61], PDQ [26], D2TCP [58], MCP [14] and Karuna [13]. pFabric [8] with Earliest-Deadline-First (pFabric-EDF) assigns priorities of packets for the deadline flows to be the flow's deadline. And it assigns priorities of packets for the non-deadline flows based on remaining flow size. D^3 [61] proposes a deadline-aware control protocol, which uses explicit rate control to apportion bandwidth to meet the deadlines of flows.
5) Recent Hardware Solutions: Recent works also leverage the emerging new hardware in networking. [5, 15, 27, 31, 43, 44, 49, 51–54, 56]. PIFO [53] designs a priority queue. In this design, each incoming packet can be enqueued into an arbitrary position of the queues, while PIFO dequeues packets from the head. PIEO [49] is a generalization of the PIFO primitive allowing dequeue from arbitrary positions. SP-PIFO [5] uses strict-priority queues to achieve similar behavior of an ideal PIFO. PIFO is indeed very flexible and

generic to a great many traffic optimization problems. However, PIFO cannot implement pFabric when dealing with starvation.

3 THE QCLUSTER FRAMEWORK

In this section, we first propose a generic framework, ***QCluster***, to address the *queue clustering* problem. Second, we show how to use the SCM sketch to record and update flow information. Third, we show how to control cluster sizes. Last, we propose an algorithm to avoid packet disorder.

3.1 The QCluster Framework

Queue Clustering: Given a switch with k queues per port, there are incoming packets belonging to different flows. The problem is how to cluster packets with similar weights/properties into the same queue without measuring traffic in advance. Note that the queue clustering problem has four requirements that most existing clustering problems do not have (see Section 1.2).

The QCluster Framework: Our framework is inspired by k-means, and can be applied to all flow scheduling problems. QCluster works as follows (see Figure 1). 1) For packets with the same special property, *e.g.*, with a deadline, or time-sensitive, QCluster clusters them into queues different from other flows; for flows with the same property, QCluster clusters them according to the packet weight. 2) Packet weight has different definitions for different scheduling problems. For example, the packet weight in LAS is the number of bytes sent. Packet weight is recorded and updated by the SCM sketch which is detailed in Section 3.2. QCluster maintains the average packet weight for each queue, namely *queue weight*. 3) When choosing the queue, we consider three factors: distance, cluster size, and packet disorder. Given an incoming packet, we compare the packet weight with the k queue weights to choose two adjacent queues, and then choose one of them according to our requirement for cluster size. For example, for Fair Queueing, we need to let all clusters have the same size; for LAS, we need to let higher priority queues have fewer packets. The reason behind is shown in Section 3.3. Section 3.4 shows how to avoid packet disorder. 4) For different scheduling problems, we need to choose the corresponding dequeuing policy. For example, we should choose strict priority [10] for minimizing FCT, and weighted round-robin [60] for fairness.

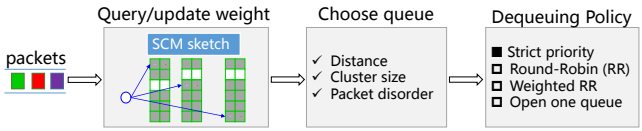


Figure 1: The QCluster framework.

3.2 The Scheduling Count-Min Sketch

In QCluster, given an incoming packet a_{now} with flow ID a, we need to know three kinds of information: 1) The number of bytes sent; 2) the arriving time of the last packet of the incoming flow; 3) the queue that the last packet was sent into. To record these kinds of information with extremely limited memory in switches, we propose an enhanced CM sketch [17], namely the Scheduling Count-Min sketch (SCM). Compared to CM, SCM has three additional functions: 1) SCM can automatically delete the information of aged flows; 2) SCM records the queue ID of the previous packet of each flow goes; 3) SCM can distinguish messages[1] and Flowlets.

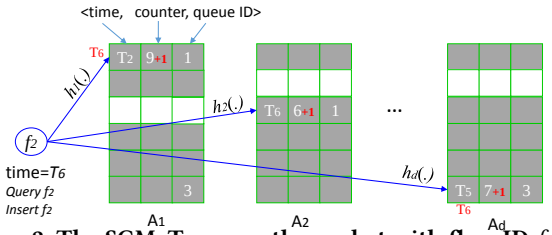


Figure 2: The SCM. To query the packet with flow ID f_2, we get d hashed buckets, and then report the oldest time T_2, and the smallest counter 6. To insert a packet of flow f_2, for each hashed buckets, we update its timestamp to the current time T_6 and increment the counter by 1.

Data Structure (Figure 2): The SCM sketch consists of d arrays, $A_1, \cdots, A_d$. Each array has l buckets. Each bucket has three fields: a *timestamp* recording the last access time, a *counter* recording the number of bytes (or packets), and a *queue ID* recording the queue that the last packet hashed into this bucket was sent into. Each array is associated with a hash function $h_i(.)$.

Insert: To insert a packet a_{now} with flow ID a, we calculate d hash functions, and get d *hashed buckets*. We first define a threshold $\Delta T_{message}$. Suppose that the time now is t_{now}. For each of the d hashed buckets, if its timestamp t_{bucket} is older than $t_{now} - \Delta T_{message}$, we consider the incoming packet as the first packet of a new message. We clear this bucket, set the counter field to the number of bytes in the packet, and set the timestamp field to the current time. Otherwise, we add the number of bytes in the packet to the counter, and update timestamp to the current time.

Query: To query a packet a_{now} with flow ID a, similarly, we get the d hashed buckets. We choose the smallest size $\hat{w}(a_{now})$ as the weight of packet a and the oldest timestamp $\hat{t}(a)$ as its timestamp. If $\hat{t}(a)$ is older than $t_{now} - \Delta T_{message}$, it means that no packets arrived in recent $\Delta T_{message}$ time.

Distinguishing flowlets is very similar to distinguishing message. The recorded queue ID is only used for packet disorder avoidance. The insertion and query complexity of the SCM sketch are $O(1)$.

3.3 Adjusting Cluster Size

QCluster clusters all the packets into k clusters, and each cluster corresponds to a queue. The mean of each cluster corresponds to the queue weight. Let q_i be the i^{th} queue, and let m_i be the queue weight of q_i. Given an incoming packet with weight $w(a_{now})$, we compare $w(a_{now})$ with $m_1, \cdots, m_k$. Suppose $m_i \leqslant w(a_{now}) < m_{i+1}$, it means we should choose q_i or q_{i+1}. Different from traditional approaches which choose the nearest one, we may choose differently in order to control the cluster size. We have two strategies: Same-Cluster-Size and Proportional-Cluster-Size.

Same-Cluster-Size. For Fair Queueing, we need to let all clusters have the similar size. In practice, when all clusters have the similar size, the queue weight will be inversely proportional to the number of flows in the queue. In dequeuing, we need to perform weighted round robin, and the weight is inversely proportional to the queue

[1] In data centers, applications often establish persistent TCP connections. Communications can reuse the opened connections. Such communications are known as "messages". Packets that belong to different messages will be scheduled individually.

weight. We propose a technique, namely *Adaptive Threshold*. We define the threshold between q_i and q_{i+1} as: $thres_i = m_i * \beta + m_{i+1} * (1 - \beta)$, where $\beta = (\frac{p_i}{p_i + p_{i+1}})^{\alpha}$ and p_i is the number of packets in q_i. When increasing/decreasing α, the number of packets in q_i will increase/decrease. In implementation, α will increase or decrease automatically according to the cluster size.

Proportional-Cluster-Size. For policies of LAS, SRTP, Deadline-Aware, we need to let the size of each cluster be proportional to the queue weight. It is well known that a large number of flows only contain a small number of packets [34, 63]. If we still let all clusters have the same size, small flows will be blocked by large flows in the first queue. Therefore, Proportional-Cluster-Size can reduce FCT for small flows. Similarly, we can also use the above adaptive-threshold technique. The only difference is that we need to change p_i to $\frac{p_i}{m_i}$. We also try three other methods (arithmetic mean, geometric mean, harmonic mean) to achieve similar performance. According to our experimental results (*see* Fgure 10(a)), we recommend using adaptive-threshold or geometric mean.

3.4 Packet Disorder Avoidance

When deployed in switches, QCluster will automatically adjust queue thresholds across time and space. On the one hand, dynamic thresholds can achieve better performance; on the other hand, latter packets could be sent to higher-priority queues while the former packets of the same flow are still in a congested low-priority queue, thus packet disorder may happen. While existing solutions [6, 22, 23, 25, 29, 30] can significantly reduce the probability of packet disorder, we aim to avoid disorder.

We propose an algorithm named Packet Disorder Avoidance (PDA). In the PDA algorithm, we need to know whether the previous packet of the current flow is in the switch, which currently cannot be implemented in commodity switches, and thus we use the SCM sketch and *Flowlet* for approximate implementation. Flowlet is first proposed by Erico Vanini *et al.* [59], and we change the definition a little: given an incoming packet a_{now} with flow ID a, if all the packets in all queues do not belong to flow a, we consider a_{now} as the beginning of a Flowlet. In this case, the last packet a_{last} of flow a has already been sent to the next-hop switches, and a_{now} can go to any queue. In other words, different Flowlets can be scheduled individually and packet disorder will not happen.

The SCM sketch reports the last arriving time of flow a. If the last packet of flow a was sent to a queue more than $\Delta T_{Flowlet}$ ago, we consider a_{now} the first packet of a new Flowlet. If a_{now} is not the first packet of a new Flowlet, we query the SCM sketch to get which queue the previous packet stays. PDA works slightly differently for LAS/SRPT/Deadline-Aware and Fair Queueing. For LAS/SRPT/Deadline-Aware, given an incoming packet a_{now} of flow a, if a_{now} is not the first packet of a Flowlet, and the previous packet of flow a is in queue q_i, we do not allow a_{now} to go to *any higher-priority queue* than q_i; Otherwise, we can choose the queue according to the clustering algorithm, and packet disorder will not happen. For Fair Queueing, if the previous packet of flow a is still in one queue (q_i), we do not allow a_{now} to change the queue, *i.e.*, a_{now} can only go to q_i; Otherwise, we choose the queue according to the clustering algorithm.

4 APPLICATIONS

We have applied QCluster to four scheduling problems: SRPT, LAS, Fair Queueing, and Deadline-Aware scheduling. This section also shows how to apply QCluster to other 6 flow scheduling problems. We list all the scheduling problems and applications in Table 1. When applying QCluster to different scheduling problems, the differences lie in the following aspects:

1) Strategy: The strategies for scheduling flows. If there is no special property, most scheduling problems let the packet with the smallest weight go first.

2) Special property: Special requirements that the algorithms must meet. For example, for Deadline-Aware Scheduling, deadline flows must complete before their deadlines; for TSN flows [2], they have the highest priority when they arrive at a switch. Flows with special property are clustered into high-priority queues, and the other flows are clustered into low-priority queues. When all high-priority queues have no packet, we can dequeue packets from low-priority queues using strict priority or round robin.

3) Packet weight: The packet weight for scheduling. Packet weight can be the number of bytes already sent, total flow size, or the remaining flow size. Hybrid means for flows without special property, the weight is different for SRPT, LAS, Fair Queueing, *etc.*

4) PDA (in a flowlet): The requirements for packet disorder avoidance. For Fair Queueing, we do not allow the incoming packet to change its egress queue in a flowlet. For other scheduling policies, we do not allow the incoming packet to go to a queue with a higher priority than the previous packets in a flowlet.

5) Dequeuing policy: Most scheduling problems use strict priority. Fair Queueing uses weighted round robin. Weighted sharing is proposed by Aalo [16].

In practice, applications may need hybrid policies, as shown in the end of Table 1. As QCluster can implement all basic scheduling policies, and thus can also be adapted for hybrid scheduling scenarios. For example, there are many deadline flows, and users may also want to maximize fairness for deadline flows. In this case, we can cluster deadline flows into the first several queues, and dequeue with weighted round robin.

5 TESTBED AND IMPLEMENTATIONS

We build a testbed to evaluate QCluster, and deploy it in a Tofino switch. In our testbed, we focus on LAS and Fair Queueing. For other two policies, we show the large-scale simulations in ns-2.

5.1 Testbed Setup

Our testbed consists of 7 servers and an Edgecore Wedge 100BF-32X switch (with Tofino ASIC). We use another 1000 Mbps switch to manage the servers and the Tofino switch. Each server runs Ubuntu 16.04-64bit with Linux kernel 4.13, and is connected to the Tofino switch via a Mellanox ConnectX-3 40GbE NIC. We are able to achieve about 37Gbit/s throughput between each pair of servers. To improve the FCT, we set the RTO-min in Linux kernel to 10ms. In the switch, we use the per-port ECN marking, and set the marking threshold to 300KBytes (about 200 packets).

5.2 Implementation in P4

We have fully implemented a P4 version of QCluster with 500 lines of P4 code, including all the *registers* and *metadata* for QCluster in the data plane, and compiled it to the Tofino switch [57].

Table 1: Applying QCluster to Scheduling Algorithms. We have applied QCluster to the first four scheduling problem. According to pFabric [8], Deadline-aware scheduling means Deadline-first-then-SRPT in default: after meeting the dealines, we should use SRPT for other non-deadline flows. Similarly, we have Deadline-first-then-LAS, Deadline-first-then-SJF.

Scheduling problem	Strategy	Special property	Packet weight	PDA (in a flowlet)	Dequeuing policy
SRPT [8, 11, 21]	Smallest weight first	–	# bytes remained	Priority cannot ascend	Strict priorty
LAS [10, 33, 40]	Smallest weight first	–	# bytes sent	Priority cannot ascend	Strict priorty
Fair Queueing [18, 41, 48]	Fair scheduling	–	# bytes sent	Queue cannot change	Weighted round robin
Deadline-aware-SRPT [8, 26, 61]	Deadline flow first, then ...	Deadline flows	Hybrid	Priority cannot ascend	Strict priorty
Shortest Job First (SJF)	Smallest weight first	–	Total flow size	Priority cannot ascend	Strict priorty
Deadline-first-then-LAS/SJF/Fairness	Deadline flow first, then ...	Deadline flows	Hybrid	–	Hybrid
Coflow scheduling [4, 16, 64]	Weighted fair scheduling	–	# bytes sent	Priority cannot ascend	Weighted sharing
Minimum rate guarantees [20]	Flow below its minimum rate first	Flows below their minimum rate	Hybrid	–	Hybrid
TSN flow scheduling [1, 2]	TSN flow first	TSN flows	Hybrid	–	Hybrid
Hybrid scheduling scenarios	Hybrid	–	Hybrid	–	Hybrid

Using Registers and SALUs. In the Tofino switch, we use *registers* to implement the SCM sketch, where registers are a kind of stateful objects. We leverage the SALU (Stateful ALU) in each stage to look up and update the entries in the registers. In the current Tofino switches, a SALU can at most update a pair of up to 32-bit registers, while one bucket of our SCM sketch has three fields. To address this problem, we divided our SCM sketch into two sketches. The two sketches have the same number of buckets and the same k hash functions. The difference is that each bucket of the first sketch includes the timestamp and counter, while that of the second sketch includes the timestamp and Queue ID. The timestamps of two sketches are the same, and thus redundant, but inevitable.

Update of Queue Weights. For each queue, we maintain two variables: *weight sum* (the number of total weight) and *packet number*. The queue weight is the weight sum divided by packet number. After an incoming packet is sent to the chosen queue, we add its weight to the weight sum and increment the packet number by 1. When one packet dequeues, we do not update the queue weights because the ingress/egress pipelines in Tofino switches do not share memory, so it is tricky to implement the update.

Table 2: H/W resources used in P4 by QCluster.

Resource	Usage	Percentage
SRAM	61	6.35%
TCAM	3	1.04%
Hash Bits	94	1.88%
Stateful ALU	5	10.42%

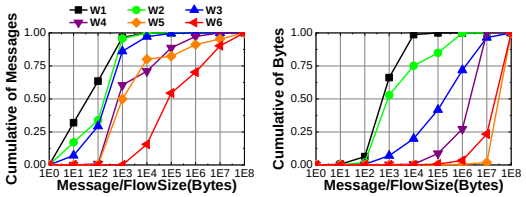
The Problem of Division. The current Tofino switch does not support the division operation in the data plane, and thus we cannot directly calculate the queue weight. However, we noticed that these queue weights can tolerate errors, which allows some delay before updating the average values. Our key idea is to use the control plane to calculate and update the queue weights periodically. Every packet goes through a *range match-action table* to determine which queue to send, increments the *packet number*, and accumulates the *weight sum* for this queue. Using range match-action tables is because comparing one weight with the k queue weights one by one needs too many stages. The thresholds of the range match entries are inserted and updated by the control plane periodically.

Resource Usage. In this way, we only need 6 stages: one stage to get the timestamp, two stages for lookup and insertion of SCM sketch, one stage for calculating $\hat{w}$, one stage for range matching, and one stage for the increment of weight sum and packet number. Table 2 shows the resource usage. As a result, we can fit the QCluster into the switch ASIC for packet processing at line-rate.

6 EXPERIMENTAL RESULTS

In this section, we conduct extensive experiments in a testbed and ns-2, and compare our QCluster with the state-of-the-art solutions. In all experimental figures below, QCluster for different problems is abbreviated as follows. **QC-SRPT:** QCluster for SRPT. **QC-LAS:** QCluster for LAS. **QC-FQ:** QCluster for Fair Queueing. **QC-DDL:** QCluster for Deadline-Aware Scheduling.

6.1 Experimental Setup



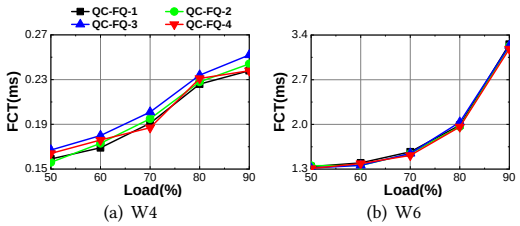
(a) Cumulative of Messages (b) Cumulative of Bytes

Figure 3: The workloads used to evaluate. The distributions are based on measurements from real production data centers. [7, 9, 24, 47, 50]

Workloads (Figure 3): We use six workloads as previous works did [8, 10, 11]. Their distributions are shown in Figure 3. We use W1-W4 which were used to evaluate Homa. Besides, we also use the Data Mining workload(W5) and the Web Search workload(W6) which were used to evaluate DCTCP[7], pFabric[8] and PIAS[10]. We consider flows smaller than 1KB as small flows, and flows larger than 10KB as large flows. Because there is no flow smaller than 1KB in W6, we consider flows less than 10KB and flows larger than 100KB as small flows and large flows in W6. In the following experiments, we mainly use W4 and W6 for comparisons because these two workloads are frequently used by existing solutions. We show the performances on the remaining workloads in supplementary material due to space limitation.

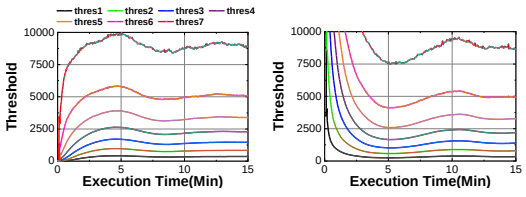
Comparison with state-of-the-art:
1) *For SRPT and LAS*, we compare QC-LAS and QC-SRPT with simulations of pFabric[8], PIAS[10], and DCTCP[7]. We do not compare with Homa [11] because of two reasons: (1) Homa does not provide simulation codes in ns-2. (2) Homa assumes congestion often happens in the downlinks of edge switches, while we do not.
2) For Fair Queueing, we compare QC-FQ with ideal fair queueing, ideal fair queueing with ECN, and AFQ. We use the BR[18] algorithm as the ideal fair queueing. AFQ is chosen because it is approximately the practical version of BR.
3) For Deadline-Aware Scheduling, we compare our QC-DDL with pFabric-EDF[8] and DCTCP[7]. Like pFabric-EDF, QC-DDL aims to minimize the FCT of non-deadline flows and maximize the throughput of deadline flows. However, other algorithms, like MCP[14] and Karuna[13], only address one of these situations in their evaluation.
Metrics:
Flow completion times (FCT): FCT is generally used in measuring the performance of scheduling algorithms. We measure the average FCT across all flows, and separately for different flow sizes. We also consider the 99th percentile flow completion time.
Jain's fairness index [28]: As AFQ [48] does, we use Jain's Fairness index to measure the fairness. It is defined as $J(x_1, \cdots, x_n) = (\sum_{i=1}^{n} x_i)^2/(n \cdot \sum_{i=1}^{n} x_i^2)$, where x_i is the average throughput of flows with the same order of magnitude.
Application throughput: For deadline traffic, we measure the application throughput which is defined as the fraction of flows that meet their deadlines.

6.2 Experiments in Testbed



(a) W4 (b) W6

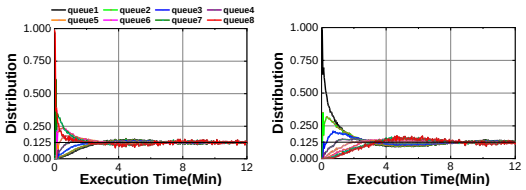
Figure 4: Overall average FCT of different initial thresholds for QC-FQ on different workloads.



(a) QC-FQ-1 (b) QC-FQ-3

Figure 5: Change of thresholds over time for QC-FQ on W6.

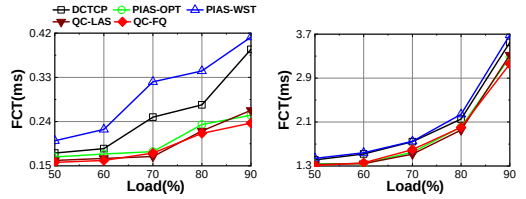
In this section, we show the experimental results in our testbed. We first take QC-FQ as the example to show the influence of different initial thresholds, and then compare QC-LAS and QC-FQ with DCTCP and PIAS. We use four different initial threshold settings for QC-FQ (QC-FQ-1 to QC-FQ-4). In the first three settings, all flows will go to the queue with the lowest, middle or highest priority at the beginning. In QC-FQ-4, all flows will go to queues with either the highest or the lowest priority. The performance of PIAS heavily depends on the thresholds. We implement PIAS in P4 switch, and use two sets of thresholds for PIAS. In our testbed, the



(a) QC-FQ-1 (b) QC-FQ-3

Figure 6: Change of packet distributions over time for QC-FQ on W6. The packet distribution is the proportion of packets in one queue among all packets.

flow size distribution is known, and thus we can obtain the optimal static thresholds, "PIAS-OPT". We can also find static thresholds that result in poor performance, "PIAS-WST". A simple method producing "PIAS-WST" is to let 60% packets go to the first queue, 30% packets go to the last.
QC-FQ performance on different initial thresholds: (Figure 4-6): The overall FCT using different initial thresholds are close to each other in both W4 and W6. For different initial thresholds in 90% workload of W6, the thresholds of same queues converge to the similar values respectively, and the proportion of packets in each queue becomes close to each other. It is because that QCluster will update and optimize the thresholds in a short period of time, and the poor performance at the beginning has little impact on performance in the long term.



(a) W4 (b) W6

Figure 7: Overall average FCT on different workloads.

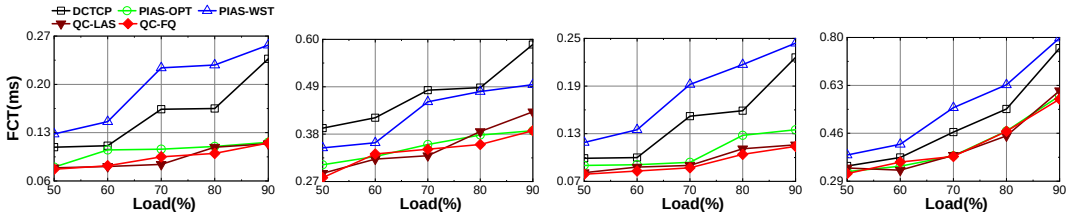
Overall performance for different scheduling policies: (Figure 7): The overall FCT of our QC-FQ is about 42.4% and 14.3% lower than that of PIAS-WST, and about 38.8% and 11.0% lower than that of DCTCP on W4 and W6. Compared to PIAS-OPT, the overall FCT of QC-FQ is about 6.3% and 3.7% lower on W4 and W6.
Performance on W4 (Figure 8): The FCT of QC-FQ is about 55.3% lower than that of PIAS-WST and about 51.5% lower than that of DCTCP for small flows in (0, 1KB). For the 99th percentile flow of small flows, the FCT of QC-FQ is about 21.6% lower than that of PIAS-WST. For middle flows in (1KB, 10KB) and large flows in (10KB, ∞), QC-FQ reduces the FCT by about 53.3% and 27.2% compared to PIAS-WST, respectively.

6.3 Experiments in ns-2

ns-2 settings: We use the leaf-spine topology, which consists of 4 spine switches and 9 leaf (ToR) switches. Each leaf switch is connected to 16 hosts via 10Gbps links, and connected to each spine switch via a 40Gbps link. The round-trip time between hosts under different leaf switches is 40.8μs. We use the packet spraying [19] for load balancing and disable dupACKs. QCluster is deployed in all switches. For each output port, We use a SCM sketch with 83KB memory.

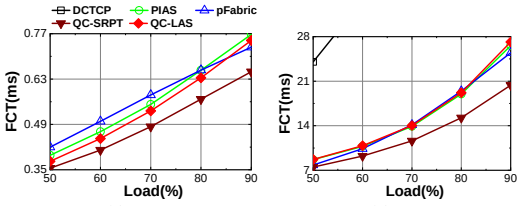
6.3.1 *Evaluation on SRPT and LAS.*

We show the performances on W4 and W6 in Figure 9.



(a) (0,1KB): Avg (b) (0,1KB): 99th Percentile (c) (1KB,10KB): Avg (d) (10KB,∞): Avg

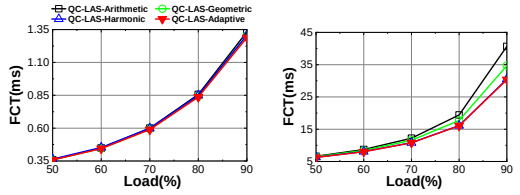
Figure 8: FCT across different flow sizes on W4.



(a) W4 (b) W6

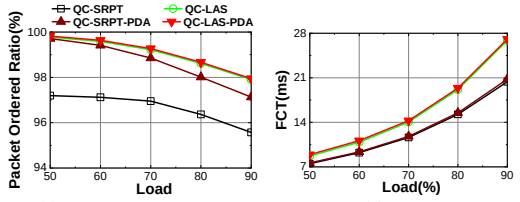
Figure 9: Overall average FCT on different workloads for SRPT and LAS.

Overall performance (Figure 9): Compared to pFabric, the average FCT of QC-SPRT is about 4.3% lower. Especially on W4 and W6, the average FCT of QC-SPRT is about 11.3% and 21.7% lower. Besides, for LAS, the average FCT of QC-LAS is about 4.73% lower than the average FCT of PIAS. For SRPT, with ECN for congestion control, QC-SPRT can perform better on most workloads. Next, we show the detailed analysis on W4.



(a) W4 (b) W6

Figure 10: FCT comparisons using different methods to achieve Proportional-Cluster-Size for LAS.



(a) Packet Ordered Ratio (b) FCT

Figure 11: Impact of Packet Disorder Avoidance on W6 for SRPT and LAS.

Performance on W4 (Figure 12): Experimental results show that for small flows in (0,1KB), the average FCT of QC-LAS is about 3% lower than that of PIAS. As shown in Figure 9(a), the average FCT of QC-SRPT is about 11.3% lower than the average FCT of pFabric in W4. It is because that QC-SRPT decreases the FCT of large flows in (10KB, ∞) by about 20%. Because the FCT of DCTCP is beyond the range that pictures can represent, we do not show it.

Impact of distinguishing the messages (Figure 13): In the above experiments, QC-LAS uses SCM sketch to distinguish the messages, while PIAS in ns-2 distinguishes the messages totally accurately, which cannot be achieved in practice. Therefore, we show how the methods of distinguishing messages influence the FCT. We use W6 to show the impact of distinguishing the messages. We add the suffix "-Ideal" and "-Real" to the algorithms which can accurately distinguish messages and the algorithms which use time interval to distinguish messages, respectively. As shown in Figure 13, the FCT of QC-LAS-Real is about 56.6% lower than that of PIAS-Real for flows in (0, 10KB). Besides, for flows in (0, 10KB) at 90% load, PIAS-Real increases the FCT by about 80%, while QC-LAS-Real increases the FCT by about 20%. The method of distinguishing messages has a great influence on FCT. Besides, the higher the load, the greater the impact of the method of distinguishing messages.

Different methods for Proportional-Cluster-Size: (Figure 10): To achieve Proportional-Cluster-Size, we evaluate the performance of QC-LAS using four methods: adaptive-threshold, geometric mean, harmonic mean, and arithmetic mean. We observe that the average FCT using adaptive-threshold is about 12.5%, 1.2%, and 25.3% lower than that using geometric mean, harmonic mean, and arithmetic mean, respectively. Therefore, we recommend using adaptive-threshold.

Impact of Packet Disorder Avoidance (PDA) (Figure 11): We observe that using PDA on QC-LAS can reduce the packet disorder by about 2.2%, and the cost is to increase the average FCT by about 0.9%. This means that most of packet disorder in QC-LAS is caused by packet loss, which can not be avoided by PDA. Using PDA on QC-SRPT can reduce the disorder by about 35.2%, and the cost is to increase the average FCT by about 2%. This is because in SRPT, late-arriving packets have higher priority, which is more likely to cause packet disorder. This kind of disorder can be avoided by PDA.

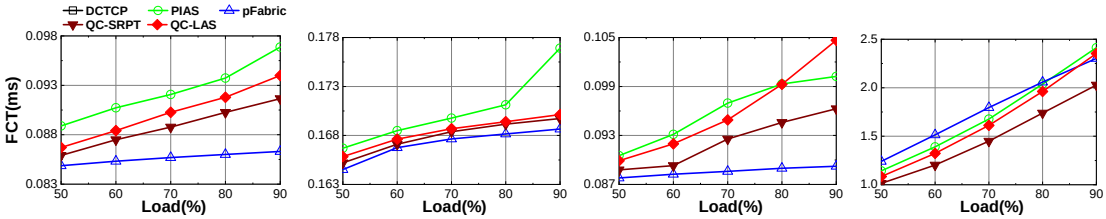
Summary: 1) QC-LAS outperforms PIAS in general, and improve the average FCT of PIAS by about 4.73%. Taking the impact of distinguishing the messages into consideration, QC-LAS significantly improves the FCT for short flows by about 56.6%. 2) QC-SPRT performs better in most workloads. The average FCT of QC-SPRT is about 4.3% lower than that of pFabric.

6.3.2 *Evaluation on Fair Queueing.*

Because QC-FQ uses the number of packets as a unit, we use W6 in the evaluation of Fair Queueing.

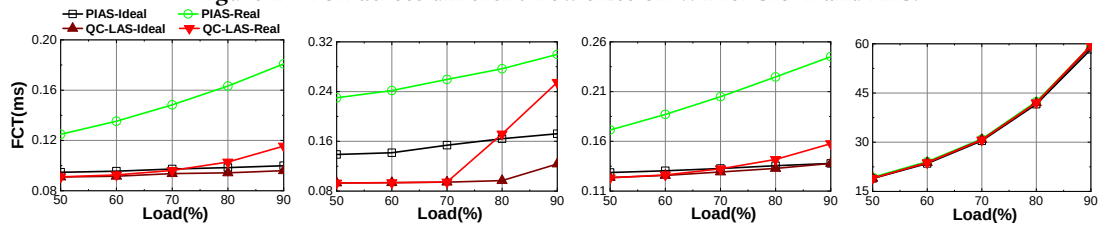
Jain's Fairness Index (Figure 14): As shown in Figure 14, The Jain's Fairness Index of QC-FQ is about 5.5% lower than that of ideal algorithm with ECN and about 11.7% higher than that of AFQ. This result shows that QC-FQ can **halve the gap with the optimal value** compared to the state-of-the-art.

FCT (Figure 15): The average FCT of QC-FQ is about 13.2% lower than that of ideal fair queueing with ECN, and about 8.4% lower than that of AFQ. We measure the FCT with the Average FCT - Flow Size diagram. An ideal fair queueing should be a direct proportion function. As we can see from Figure 15, though the performance of



(a) (0,1KB): Avg (b) (0,1KB): 99th Percentile (c) (1KB,10KB): Avg (d) (10KB,∞): Avg

Figure 12: FCT across different flow sizes on W4 for SRPT and LAS.



(a) (0KB, 10KB): Avg (b) (0, 10KB): 95th Percentile (c) (10KB,100KB): Avg (d) (100KB,∞): Avg

Figure 13: Impact of distinguishing the messages on W6 for LAS.

our algorithm is not exactly a straight line compared to that of ideal fair queueing, it nearly satisfies the Fair Queueing requirement. Meanwhile, it improves the average FCT.

Summary: 1) Though QC-FQ does not achieve as high fairness as the ideal fair queueing, the overall FCT of QC-FQ is about 13.2% lower. 2) Compared with AFQ, QC-FQ achieves both higher fairness and lower latency. The Jain's Fairness Index of QC-FQ is about 11.7% higher and the overall FCT of QC-FQ is about 8.4% lower.

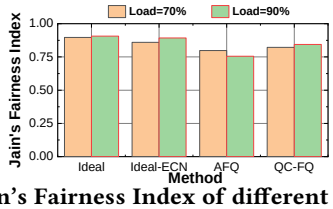
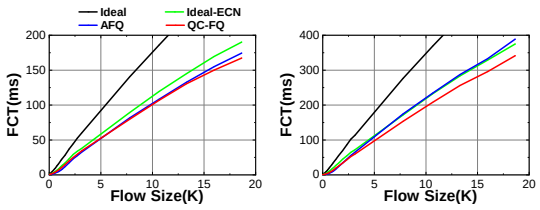


Figure 14: Jain's Fairness Index of different Fair Queueing Algorithms on W6.



(a) FairQueue: Load 70% (b) FairQueue: Load 90%

Figure 15: Average FCT comparisons under different flow size on W6 for Fair Queueing.

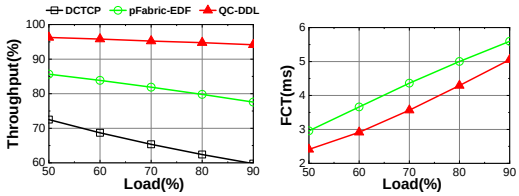
6.3.3 Evaluation on Deadline-Aware Scheduling.

In this experiment, we only assign deadlines for flows that are smaller than 100KB in W4. The deadlines are assumed to be exponentially distributed similar to prior work [8].

Application Throughput (Figure 16(a)): Compared to pFabric-EDF, QC-DDL can increase the application throughput by about 15%. And the application throughput of QC-DDL is about 29% higher than that of DCTCP. Because when switches begin to send the large deadline flow, pFabric-EDF could be too late to catch up with its deadline. Besides, when the deadline is a large number, deadline flows may be blocked by non-deadline flows regardless of the remaining time to deadline in pFabric-EDF.

FCT (Figure 16(b)): the FCT of non-deadline flows in QC-DDL is about 15% lower than that of pFabric-EDF. Because pFabric-EDF does not take the size of flows into consideration, it will send the deadline flows aggressively, which will hurt the FCT of non-deadline flows. The FCT of DCTCP is beyond the range that pictures can represent, so we do not show it in this figure.

Summary: 1) The application throughput of QC-DDL is about 29% higher than that of DCTCP and about 15% higher than that of DCTCP. 2) The FCT of non-deadline flows of QC-DDL is about 15% lower than that of pFabric-EDF.



(a) Application Throughput (b) FCT for non-deadline flows

Figure 16: Deadline-Aware Scheduling on W4.

7 CONCLUSION

This paper proposes a framework, QCluster, to adapt existing flow scheduling solutions (SRPT, LAS, Fair Queueing, and Deadline-Aware Scheduling) for limited number of queues without measuring traffic in advance. The key idea of QCluster is to cluster packets with similar weights/properties into the same queue. We also propose the PDA algorithm to avoid packet disorder incurred by scheduling. We apply QCluster to four typical scheduling problems, and also show how to apply QCluster to other scheduling problems. We implement QCluster with PDA in Tofino switches, achieving a clustering speed of 3.2 Tbps. We also implement QCluster in large-scale ns-2 simulations for four kinds of scheduling problems. Experimental results in testbed and ns-2 show that QCluster achieves better or comparable performance compared to the state-of-the-art algorithms for four typical scheduling policies. All the source codes in ns-2 are available in Github without identity information [45].

REFERENCES

[1] 2016. IEEE 802.1: 802.1Qbv - Enhancements for Scheduled Traffic. http://www.ieee802.org/1/pages/802.1bv.html. (2016).

[2] 2017. IEEE 802.1 Time-Sensitive Networking Task Group. http://www.ieee802.org/1/pages/tsn.html. (2017).

[3] 2022. Gearbox: A Hierarchical Packet Scheduler for Approximate Weighted Fair Queuing. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA. https://www.usenix.org/conference/nsdi22/presentation/gao

[4] Saksham Agarwal, Shijin Rajakrishnan, Akshay Narayan, Rachit Agarwal, David Shmoys, and Amin Vahdat. 2018. Sincronia: near-optimal network design for coflows. In *SIGCOMM*. ACM, New York, NY, USA, 16–29.

[5] Albert Gran Alcoz, Alexander Dietmüller, and Laurent Vanbever. [n. d.]. SPPIFO: Approximating Push-In First-Out Behaviors using Strict-Priority Queues. In *USENIX NSDI*, Vol. 20.

[6] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. 2014. CONGA: Distributed congestion-aware load balancing for datacenters.

[7] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. 2010. Data center TCP (DCTCP). *SIGCOMM* 40, 4 (2010), 63–74.

[8] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick Mckeown, Balaji Prabhakar, and Scott Shenker. 2013. pFabric: Minimal Near-Optimal Datacenter Transport. In *SIGCOMM*. ACM, New York, NY, USA, 435–446.

[9] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Jiang Song, and Mike Paleczny. 2012. Workload analysis of a large-scale key-value store. In *SIGMETRIC*. ACM, New York, NY, USA, 53–64.

[10] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. 2015. Information-agnostic flow scheduling for commodity data centers. In *NSDI*. USENIX Association, Berkeley, CA, USA, 455–468.

[11] Mohammad Alizadeh Behnam Montazeri, Yilong Li and John Ousterhout. 2018. Homa: a receiver-driven low-latency transport protocol using network priorities. In *SIGCOMM*. ACM, New York, NY, USA, 221–235.

[12] Zhiruo Cao, Zheng Wang, and Ellen Zegura. 2000. Rainbow fair queueing: Fair bandwidth sharing without per-flow state. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, Vol. 2. IEEE, 922–931.

[13] Li Chen, Kai Chen, Wei Bai, and Mohammad Alizadeh. 2016. Scheduling mix-flows in commodity datacenters with karuna. In *SIGCOMM*. ACM, New York, NY, USA, 174–187.

[14] Li Chen, Shuihai Hu, Kai Chen, Haitao Wu, and Danny HK Tsang. 2013. Towards minimal-delay deadline-driven data center TCP. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*. ACM, ACM, New York, NY, USA, 21.

[15] Sharad Chole, Andy Fingerhut, Sha Ma, Anirudh Sivaraman, Shay Vargaftik, Alon Berger, Gal Mendelson, Mohammad Alizadeh, Shang-Tse Chuang, Isaac Keslassy, et al. 2017. drmt: Disaggregated programmable switching. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 1–14.

[16] Mosharaf Chowdhury and Ion Stoica. 2015. Efficient Coflow Scheduling Without Prior Knowledge. In *SIGCOMM*. ACM, New York, NY, USA, 393–406.

[17] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[18] Alan Demers, Srinivasan Keshav, and Scott Shenker. 1989. Analysis and simulation of a fair queueing algorithm. In *ACM SIGCOMM Computer Communication Review*, Vol. 19. ACM, ACM, New York, NY, USA, 1–12.

[19] Advait Dixit, Pawan Prakash, Y. Charlie Hu, and Ramana Rao Kompella. 2013. On the impact of packet spraying in data center networks. In *Infocom*. IEEE, Turin, Italy, 2130–2138.

[20] Wu-chang Feng, Dilip D Kandlur, Debanjan Saha, and Kang G Shin. 1999. Understanding and improving TCP performance over networks with minimum rate guarantees. *IEEE/ACM Transactions on Networking* 7, 2 (1999), 173–187.

[21] Peter X Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2015. pHost: Distributed Near-Optimal Datacenter Transport Over Commodity Network Fabric. In *Acm Conference on Emerging Networking Experiments & Technologies*. ACM, New York, NY, USA, 1–12.

[22] Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, and Mohammad Alizadeh. 2016. Juggler: a practical reordering resilient network stack for datacenters. In *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, New York, NY, USA, 20.

[23] Soudeh Ghorbani, Zibin Yang, P Godfrey, Yashar Ganjali, and Amin Firoozshahian. 2017. DRILL: Micro load balancing for low-latency data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 225–238.

[24] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: A Scalable and Flexible Data Center Network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication (SIGCOMM)*. ACM, New York, NY, USA, 51–62.

[25] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. 2015. Presto: Edge-based load balancing for fast datacenter networks. In *ACM SIGCOMM Computer Communication Review*. ACM, New York, NY, USA, 465–478.

[26] Chi-Yao Hong, Matthew Caesar, and P Godfrey. 2012. Finishing flows quickly with preemptive scheduling. In *SIGCOMM*. ACM, ACM, New York, NY, USA, 127–138.

[27] Xin Sunny Huang, Xiaoye Steven Sun, and TS Eugene Ng. 2016. Sunflow: Efficient optical circuit scheduling for coflows. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. 297–311.

[28] Raj Jain, Dah Ming Chiu, and Hawe WR. 1998. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. *CoRR* cs.NI/9809099 (01 1998).

[29] Srikanth Kandula, Dina Katabi, Shantanu Sinha, and Arthur Berger. 2007. Dynamic load balancing without packet reordering. *ACM SIGCOMM Computer Communication Review* 37, 2 (2007), 51–62.

[30] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. 2016. Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*. ACM, ACM, New York, NY, USA, 10.

[31] Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, and Lawrence J Wobker. 2015. In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM*.

[32] Changhyun Lee, Chunjong Park, Keon Jang, Sue B Moon, and Dongsu Han. 2015. Accurate Latency-based Congestion Feedback for Datacenters.. In *USENIX Annual Technical Conference*. 403–415.

[33] Kai Chen Feng Liu Li Chen, Justinas Lingys. 2018. AuTO: scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *SIGCOMM*. ACM, New York, NY, USA, 191–205.

[34] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM.

[35] J. Macqueen. 1965. Some Methods for Classification and Analysis of MultiVariate Observations. In *Proc of Berkeley Symposium on Mathematical Statistics & Probability*.

[36] Dominik Mautz, Claudia Plant, and Christian Böhm. 2020. DeepECT: The Deep Embedded Cluster Tree. *Data Science and Engineering* 5, 4 (2020), 419–432.

[37] Radhika Mittal, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2016. Universal packet scheduling. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*. 501–521.

[38] Radhika Mittal, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, David Zats, et al. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. In *ACM SIGCOMM Computer Communication Review*. ACM, 537–550.

[39] Radhika Mittal, Justine Sherry, Sylvia Ratnasamy, and Scott Shenker. 2014. Recursively Cautious Congestion Control.. In *NSDI*. 373–385.

[40] Ali Munir, Ihsan A Qazi, Zartash A Uzmi, Aisha Mushtaq, Saad N Ismail, M Safdar Iqbal, and Basma Khan. 2013. Minimizing flow completion times in data centers. In *Proc. IEEE INFOCOM, 2013*. 2157–2165.

[41] John Nagle. 1987. On packet switches with infinite storage. *IEEE transactions on communications* 35, 4 (1987), 435–438.

[42] Akshay Narayan, Frank Cangialosi, Deepti Raghavan, Prateesh Goyal, Srinivas Narayana, Radhika Mittal, Mohammad Alizadeh, and Hari Balakrishnan. 2018. Restructuring endpoint congestion control. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 30–43.

[43] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. 2017. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 85–98.

[44] Srinivas Narayana, Mina Tahmasbi, Jennifer Rexford, and David Walker. 2016. Compiling Path Queries.. In *NSDI*. 207–222.

[45] QCluster [n. d.]. The source codes of our and other related algorithms. https://github.com/qcluster/QCluster. ([n. d.]).

[46] Florian Richter, Yifeng Lu, Daniyal Kazempour, and Thomas Seidl. 2020. "Show Me the Crowds!" Revealing Cluster Structures Through AMTICS. *Data Science and Engineering* 5, 4 (2020), 360–374.

[47] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In *SIGCOMM*. ACM, New York, NY, USA.

[48] Naveen Kr. Sharma, Ming Liu, Kishore Atreya, and Arvind Krishnamurthy. 2018. Approximating Fair Queueing on Reconfigurable Switches. In *NSDI*. 1–16.

[49] Vishal Shrivastav. 2019. Fast, scalable, and programmable packet scheduler in hardware. In *Proceedings of the ACM Special Interest Group on Data Communication*. ACM, 367–379.

[50] R. Sivaram. 2008. Some Measured Google Flow Sizes. *Google internal memo* (2008).

[51] Anirudh Sivaraman, Alvin Cheung, Mihai Budiu, Changhoon Kim, Mohammad Alizadeh, Hari Balakrishnan, George Varghese, Nick McKeown, and Steve Licking. 2016. Packet transactions: High-level programming for line-rate switches. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 15–28.

[52] Anirudh Sivaraman, Changhoon Kim, Ramkumar Krishnamoorthy, Advait Dixit, and Mihai Budiu. 2015. Dc. p4: Programming the forwarding plane of a datacenter switch. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. ACM, 2.

[53] Anirudh Sivaraman, Nick Mckeown, Suvinay Subramanian, Mohammad Alizadeh, and Sachin Katti. 2016. Programmable Packet Scheduling at Line Rate. In *SIGCOMM*. ACM, New York, NY, USA.

[54] Anirudh Sivaraman, Suvinay Subramanian, Anurag Agrawal, Sharad Chole, Shang-Tse Chuang, Tom Edsall, Mohammad Alizadeh, Sachin Katti, Nick McKeown, and Hari Balakrishnan. 2015. Towards programmable packet scheduling. In *Proceedings of the 14th ACM workshop on hot topics in networks*. ACM, 23.

[55] Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. 2014. An experimental study of the learnability of congestion control. In *ACM SIGCOMM Computer Communication Review*. ACM, 479–490.

[56] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S Muthukrishnan, and Jennifer Rexford. 2017. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*. ACM.

[57] Tofini [n. d.]. Barefoot Tofino: World's fastest P4-programmable Ethernet switch ASICs. https://barefootnetworks.com/products/brief-tofino/. ([n. d.]).

[58] Balajee Vamanan, Jahangir Hasan, and TN Vijaykumar. 2012. Deadline-aware datacenter tcp (d2tcp). *ACM SIGCOMM Computer Communication Review* 42, 4 (2012), 115–126.

[59] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. 2017. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*. 407–420.

[60] W. Wang and G. Casale. 2015. Evaluating weighted round robin load balancing for cloud web services. In *International Symposium on Symbolic & Numeric Algorithms for Scientific Computing*.

[61] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. 2011. Better never than late: Meeting deadlines in datacenter networks. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 50–61.

[62] Keith Winstein and Hari Balakrishnan. 2013. TCP ex machina: computer-generated congestion control. In *ACM SIGCOMM Computer Communication Review*. ACM, 123–134.

[63] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic sketch: Adaptive and fast network-wide measurements. In *Proc. ACM SIGCOMM*. 561–575.

[64] Hong Zhang, Li Chen, Bairen Yi, Kai Chen, Mosharaf Chowdhury, and Yanhui Geng. 2016. CODA: Toward automatically identifying and scheduling coflows in the dark. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 160–173.

Appendices

A PACKET DISORDER AVOIDANCE

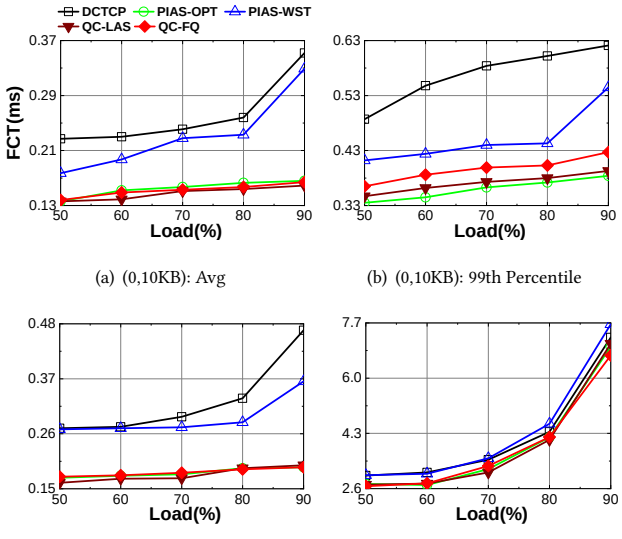
For an incoming packet a_{now}, we use the SCM sketch to get the queue where the previous packet stays. The update and query of queue ID are different between LAS and Fair Queueing.

Update of queue ID: We check the timestamp of each mapped bucket in the SCM sketch. Suppose that the time now is t_{now}, and the chosen queue for a_{now} is the i^{th} queue. If the timestamp t_{bucket} is smaller/older than $t_{now} - \Delta T_{Flowlet}$, we directly set the queue ID of this bucket to i. Otherwise, for LAS, we update the queue ID to i only if the i^{th} queue has lower-priority than the queue recorded in the bucket, and for fair queueing, we do not update the queue ID.

Query of queue ID: We only query the queue ID if a_{now} is not the first packet of a new Flowlet. For LAS, we choose the queue with the highest priority in all mapped buckets. For Fair Queueing, we choose the queue with the smallest/oldest timestamp in all mapped buckets. Both methods can minimize the minus effect of hash collisions.

The hash collisions can still cause the SCM sketch giving the wrong queue IDs. For LAS, this mistake will slightly downgrade the performance, but it will not incur packet disorder, because the SCM sketch only gives the overestimation on queue IDs. In other words, the SCM sketch may give a lower-priority queue than the queue of the previous packet, and it will never give a higher-priority queue, so the incoming packet will not go to the higher-priority queue and packet disorder will not happen. For Fair Queueing, all queues have the same priority but different weights, so there are still chances that packet disorder happens.

B OTHER EXPERIMENT RESULTS IN TESTBED



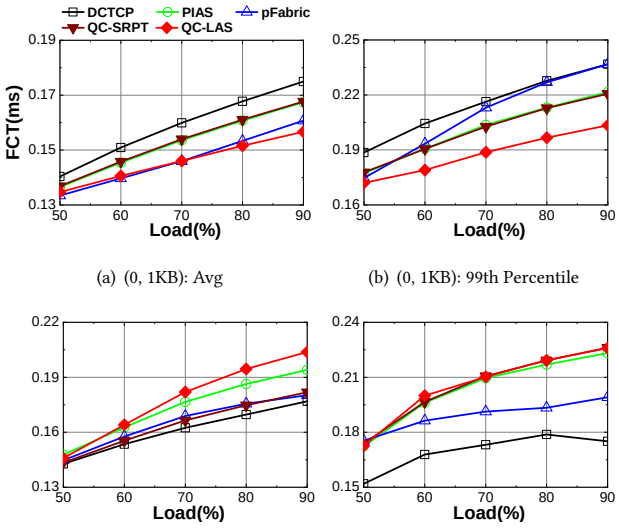
(a) (0,10KB): Avg (b) (0,10KB): 99th Percentile

(c) (10KB,100KB): Avg (d) (100KB,∞): Avg

Figure 17: FCT across different flow sizes on W6 in testbed.

Performance on W6 (Figure 17): For small flows in (0, 10KB), the FCT of QC-FQ is about 50.2% lower than that of PIAS-WST and about 53.4% lower than that of DCTCP. For the 99th percentile flow of small flows, the FCT of QC-FQ is about 21.7% lower than that of PIAS-WST. For middle flows in (10KB, 100KB), QC-FQ reduces the FCT by about 47.1% compared to PIAS-WST. For large flows in (100KB, ∞), our QC-FQ reduces the FCT by about 12.5% compared to PIAS-WST.

C OTHER EXPERIMENT RESULTS ON LAS AND SRPT IN NS2



(a) (0, 1KB): Avg (b) (0, 1KB): 99th Percentile

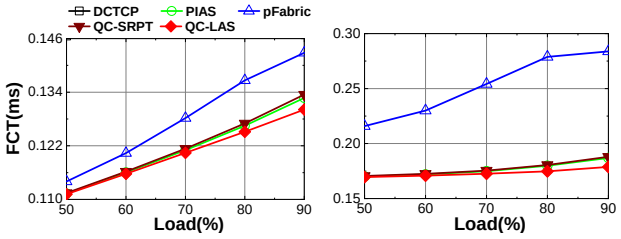
(c) (1KB,10KB): Avg (d) (10KB,∞): Avg

Figure 18: FCT across different flow sizes on W1 for SRPT and LAS.

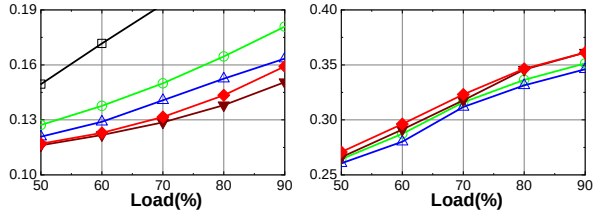
Performance on W1 (Figure 18): In W1, more than 95% of flows are smaller than 1KB. That is to say, more than 95% of flows can be transmitted in one TCP packet. In this situation, the advantage of SRPT over LAS is not obvious. Therefore, our QC-LAS may outperform pFabric on W1 in some loads. For short flows in (0, 1KB), the FCT of our QC-LAS is about 6.9% lower than that of PIAS. Without ECN for congestion control, congestion is more likely to occur in switches. Therefore, the 99th percentile FCT for small flows of pFabric is even higher than that of DCTCP at 90% load.

Performance on W2 (Figure 19): For small flows in (0,1KB), the average FCT of the QC-SRPT is about 7% lower than that of the pFabric. Moreover, QC-SRPT also reduces the 99th percentile FCT by about 50.1% at 90% load. Similar to W1, pFabric does not perform so well when data centers are dominated by very small flows. The FCT of QC-LAS is about 1.2% lower than the FCT of PIAS for small flows in (0, 1KB). And QC-LAS reduces the FCT for middle flows in (1KB, 10KB) by about 13%.

Performance on W3 (Figure 20): For small flows in (0,1KB), our QC-LAS, QC-SRPT, PIAS and pFabric achieve similar FCT. However, the 99th percentile FCT for small flows of pFabric is about 8% higher than that of QC-SRPT. For middle flows in (1KB,10KB), compared

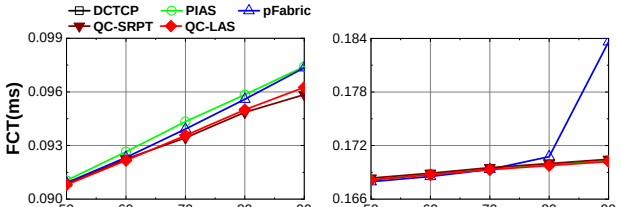


(a) (0,1KB): Avg (b) (0,1KB): 99th Percentile

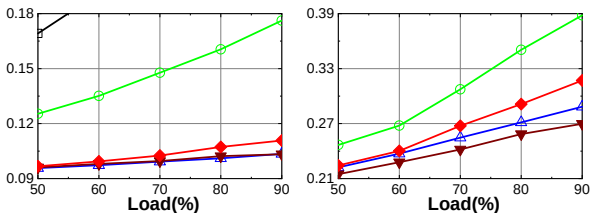


(c) (1KB,10KB): Avg (d) (10KB,∞): Avg

Figure 19: FCT across different flow sizes on W2 for SRPT and LAS.



(a) (0, 1KB): Avg (b) (0, 1KB): 99th Percentile



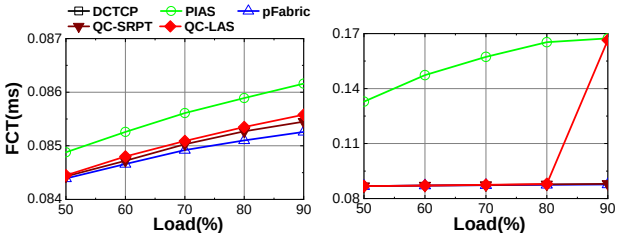
(c) (1KB,10KB): Avg (d) (10KB,∞): Avg

Figure 20: FCT across different flow sizes on W3 for SRPT and LAS.

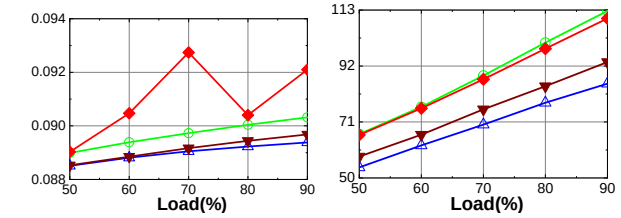
to PIAS, our QC-LAS reduces the FCT by about 60.8%. Besides, our QC-LAS reduces the FCT of large flows by about 22.4% compared to PIAS.

Performance on W5 (Figure 21): Our QC-SRPT and pFabric achieve similar FCT for small flows. And the FCT for small flows of PIAS is about 1% higher than that of QC-LAS. The 99th percentile FCT for small flows of PIAS is about 87.9% higher than that of QC-LAS.

Performance on W6 (Figure 22): For small flows in (0,10KB), the average FCT of the QC-SRPT is nearly the same as that of pFabric.

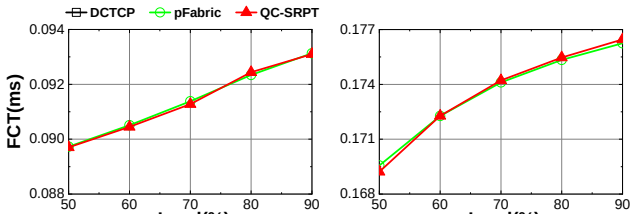


(a) (0, 1KB): Avg (b) (0, 1KB): 99th Percentile

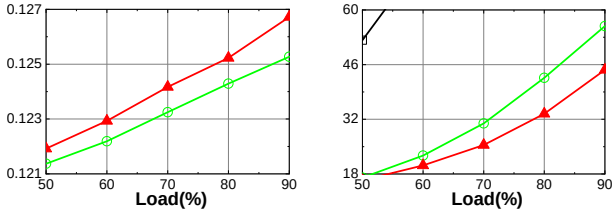


(c) (1KB,10KB): Avg (d) (10KB,∞): Avg

Figure 21: FCT across different flow sizes on W5 for SRPT and LAS.



(a) (0KB, 10KB): Avg (b) (0, 10KB): 99th Percentile



(c) (10KB,100KB): Avg (d) (100KB,∞): Avg

Figure 22: FCT across different flow sizes on W6 for SRPT.

For flows in (10KB, 100KB), the FCT of QC-SRPT is higher than that of pFabric. However, similar to W4, the average FCT of QC-SRPT is about 21.7% lower than the average FCT of pFabric in W6. It is because that QC-SRPT decreases the FCT of large flows in (10KB, ∞) by about 25%.