


Article

A Buffer Management Algorithm Based on Dynamic Marking Threshold to Restrain MicroBurst in Data Center Network

Yan Yu ¹, Xianliang Jiang ^{1,*} , Guang Jin ¹, Zihang Gao ² and Penghui Li ¹

¹ Faculty of Electrical Engineering and Computer, Ningbo University, 818 Fenghua Road, Ningbo 315211, China; ylsbylsb123red@gmail.com (Y.Y.); jinguang@nbu.edu.cn (G.J.); 1911082059@nbu.edu.cn (P.L.)

² Department of Information and Technology, Wenzhou Vocational College of Science and Technology, Wenzhou 325006, China; gaozihang@wzvcst.edu.cn

* Correspondence: jiangxianliang@nbu.edu.cn

Abstract: The data center has become the infrastructure of most Internet services, and its network carries different types of business flow, such as query, data backup, control information, etc. At the same time, the throughput-sensitive large flows occupy a lot of bandwidth, resulting in the small flow's longer completion time, finally affecting the performance of the applications. Recent proposals consider only dynamically adjusting the ECN threshold or reversing the ECN packet priority. This paper combines these two improvements and presents the HDCQ method for coordinating data center queuing, separating large and small flows, and scheduling in order to ensure flow completion time. It uses the ECN mechanism to design load-adaptive marking threshold update algorithms for small flows to prevent micro-bursts from occurring. At the same time, packets marked with ECN or ACK are raised in priority, prompting these packets to be fed back to the sender as soon as possible, effectively reducing the TCP control loop delay. Extensive experimental analysis on the network simulator (NS-2) shows that the HDCQ algorithm has better performance in the face of micro-burst traffic, reducing the average flow completion time by up to 24% compared with the PIAS.

Keywords: data center network; flow scheduling; active queue management; Incast; explicit congestion notification



Citation: Yu, Y.; Jiang, X.; Jin, G.; Gao, Z.; Li, P. A Buffer Management Algorithm Based on Dynamic Marking Threshold to Restrain MicroBurst in Data Center Network. *Information* **2021**, *12*, 369. <https://doi.org/10.3390/info12090369>

Academic Editor: Carlos Filipe Da Silva Portela

Received: 26 July 2021

Accepted: 10 September 2021

Published: 12 September 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

With the rapid development of network applications, high-performance data centers have been established worldwide to carry most of the internet traffic [1]. Traffic can be classified into large and small or long and short flows according to their size. Since long and short flows originate from different applications, they correspond to additional requirements. For example, applications such as regular updates of massive data, data reorganization, backup, and replication belong to large flows and need to ensure stable and high throughput [2]. By contrast, applications with little traffic, such as Web search, require fast response from users [3]. Therefore, a well-designed data center network must provide low latency for short flows and high throughput for long flows.

In the data center network environment, simply improving the network transmission speed does not meet the differentiated application needs, with limited resources requiring us to target traffic characteristics. In recent years, flow priority scheduling schemes have addressed the requirement by arranging the priority of flows. The existing methods can be classified according to flow sizes [4,5] and deadline information [6–8], etc. The typical scheduling design [5] progressively marks the packets of this flow with a higher to lower priority based on the cumulative number of bytes sent by the end host, enabling short flows to be complete in the first few high-priority queues and improving the completion time. This method judges traffic on the end host side. However, the network state is changing. The researcher should design a scheme that is adaptive to the specific network conditions.

Hamed et al. [9] developed an algorithm based on the network state. Specifically, they increased the priority of packets marked with the Explicit Congestion Notification (ECN) at the switch side. Since ECNs provide congestion signals, raising the priority of these packets reduces the completion time of the tail flows. Theoretically, increasing the priority of ACK packets or setting them directly to the highest priority also works. It speeds up the TCP control loop by receiving packets containing signals fast.

On the other hand, micro-burst flows [10] cannot be ignored as data center networks continue to scale. Its emergence causes standard ECN to suffer from performance degradation because the ECN threshold is generally set to a small and fixed value. When two micro-burst flows arrive at the switch simultaneously, a micro-burst flow can reach up to 64 KB instantaneously; thus, the fixed ECN marking threshold in the switch cannot mitigate the microburst phenomenon [11]. We need to adjust the threshold dynamically. Although Hamed et al. [9] used ECN signals, they did not consider dynamic adjustment of ECN marking thresholds. In this paper, relevant experiments (in Section 3) were conducted further in order to illustrate the effect of different thresholds on the network, and the experimental results confirmed the phenomenon. Therefore, we expect to compensate for the drawback of marking packets on the end host side only, which lacks real-time, by increasing the priority of ECN and ACK packets on the network side. At the same time, the ECN threshold is dynamically adjusted to prevent micro-burst flow. The basic idea of changing the ECN threshold can be analogous to setting the red light time at different times of the day. When fewer vehicles are present, the red light is set for an extended period (with a larger ECN marking threshold). By contrast, when the number of vehicles increases or even when there are long queues, we need to reduce the red light time so that vehicles pass as quickly as possible.

Inspired by the above explanation, We proposed a buffer management algorithm for the Harmony Data Center Queue (HDCQ) in this paper. The main design idea of the algorithm is first to create three types of priorities based on the characteristics of data center network traffic and mark them on the sender side. We set packets containing ACK information to the highest priority, thus ensuring that the packets carrying signals reach the sender side quickly and reducing the TCP control loop delay. Marking second and third priorities by the cumulative number of packets ensures that small flows can be completed first. On the switch side, we increased the priority of small or large flow packets marked with ECN. Since ECN is readily supported and widely deployed in today's data centers, relying on ECN makes our design much more implementation-friendly. In order to prevent network microbursts, the HDCQ algorithm dynamically adjusts the threshold based on the number of packets marked by the ECN in the queue to avoid network congestion from occurring and to maintain network robustness. Compared with other flow scheduling and ECN-based congestion control schemes, the HDCQ algorithm can more effectively reduce the completion time of short flows and maintain the throughput of long flows.

In summary, HDCQ's contributions include the following:

- Unlike previous solutions, the HDCQ algorithm is deployed on both the end-host side and network side, making full use of network resources and requiring no additional hardware support, with the exception of only simple modifications. Moreover, dynamically adjusting the small-flow ECN marking threshold in the switch prevents network microbursts, avoids network congestion, and increases the robustness of the data center.
- The HDCQ algorithm speeds up the TCP control loops by setting the ACK and ECN packets to the highest priority, reducing short flows' completion time, especially the tail flow completion time, and maintaining the throughput of large flows.
- We conducted experiments in Incast topology and showed that the HDCQ algorithm outperforms both Practical Information-agnostic Flow Scheduling (PIAS) and Random Early Detection (RED) algorithms in terms of Average Flow Completion Time (AFCT) of small flows and the throughput of large flows. Under severe network congestion, it has up to 24% reduction in AFCT of small flows compared to the PIAS algorithm.

- We evaluated PIAS, Data Center TCP (DCTCP) [12], Low Latency Data Center Transport (LLDCT) [13], and HDCQ algorithms under various web searching and data mining workloads. The results show that the HDCQ algorithm performs well in AFCT under varying traffic. Under small flows, it reduces 2.11~4.70% compared to the PIAS algorithm and 21.74~44.25% on average compared to the DCTCP algorithm (data mining workload). It outperforms other algorithms under both medium and large flows.

2. Relate Work

To motivate our design, we first introduce the schemes related to flow scheduling and classify them into whether the traffic distribution is aware of it or not. Moreover, because ECNs are easily supported and widely deployed in today's data centers, there are already algorithms designed based on ECNs, such as the DCTCP [12] algorithm, etc. We further discuss the relevant schemes for using ECNs to prevent or mitigate microbursts.

2.1. Flow Scheduling Schemes

We present the flow scheduling schemes categorized according to whether the traffic distribution is aware or not.

2.1.1. Information-Aware Scheduling

Various algorithms for flow scheduling assume that the size distribution of the flow is aware, and these schemes require switches to know essential information about the flows, such as the traffic size and deadline, etc. This information is usually given to switches either by a central controller or by end-host applications. In general, we can divide these methods into two different groups.

(1) Prior to being aware of flow deadlines: Deadline-Driven Delivery (D3) [6] is a deadline-based priority scheduling for achieving differential service for various types of flow-based deadlines. Still, it cannot optimize performance when flows of the same priority use network resources, while D3 requires changes to the switch hardware and is not compatible with TCP. The Deadline-Aware Datacenter TCP (D2TCP) [7] algorithm combines the advantages of DCTCP and D3, taking into account congestion avoidance and flow deadline, hardware, and TCP compatibility. However, D2TCP's sender decision only addresses the congestion level of the previous RTT and its flow demand. It cannot control the flow rate of each flow, resulting in it being impossible to render the most reasonable flow rate planning for the entire data center network and limiting the overall performance improvement of the data center network. The Preemptive Distributed Quick (PDQ) protocol [8] achieves global Earliest Deadline First (EDF) and Shortest Job First (SJF) by collaborating among switches distributed across the data center network. However, taking into account the rate of each flow requires the switch to provide enough information. It involves the coordination of all switches to participate, which is difficult to achieve in the dynamically changing data center network. (2) Prior to being aware of flow sizes: The pFabric [4] protocol provides a priority-based and balanced solution for short and long flows. Since data center network applications require low latency for short flows while long flows do not, the key to reducing latency is to reduce short flow wait times. The pFabric considered only conceptual models. It is challenging to realize this idea in the real world.

There are serious practical problems that plague the above Information-aware scheduling protocols. First, we must modify the hardware or the upper-layer protocol to provide some flow information. The extensive complexities are added to end-hosts and switches, which is impossible. Second, such information is challenging to obtain for most applications and may even be unavailable.

2.1.2. Information-Agnostic Scheduling

Bai et al. [5] proposed the PIAS algorithm in 2017, which is designed based on the premise that the distribution of flows in the network is agnostic and reduces FCT by SJF.

The algorithm takes advantage of the capability of existing switches. It sets up a Multi-Level Feedback Queue (MLFQ) at the sender side to downgrade packets from high-priority to low-priority queues based on the number of bytes sent. Short flows can be completed in the first few high-priority queues, thus reducing FCT. The PIAS is the first algorithm that does not apply a priori knowledge, increasing pervasiveness compared to previous schemes. However, PIAS is deployed at the end host and relies heavily on selecting multi-level feedback queue thresholds. The heuristic algorithm used consumes an extended time to compute the thresholds.

The most significant point of information-agnostic flow scheduling schemes is they do not rely on a priori information about the flow and are, therefore, easy to implement in data center networks. However, they still have several drawbacks, such as algorithms mainly deployed on the end-host side. In the data center, the network side is more authoritative in relieving network congestion by obtaining information about the entire network in order to understand the traffic distribution. It is not enough to consider only limited flow information for scheduling packets. Hamed et al. [3] mentioned that relying on the scheduling algorithm alone, regardless of the packet history experience, results in a packet of the flow experience queuing delay at multiple hops, resulting in an increase in flow completion time which is not tolerable in datacenter networks. Moreover, some machine learning solutions [14,15] are gradually applied to data centers.

2.2. Protocols Using ECN Signals

ECN is an Active Queue Management (AQM) [16] solution widely used in data centers. For example, DCTCP mainly converts the ECN sequence into the multi-bit signal to indicate congestion and adjust the congestion window size in order to achieve congestion avoidance. High-bandwidth Ultra-Low Latency Protocols (HULL) [17] centers on reducing FCT for small flows by emulating a low-rate switch port with advanced ECN marking. When the queue length exceeds a certain threshold, the switch marks the packet and notifies the sender to adjust the sending rate by marking the packet with ECN to avoid network congestion. Zhang et al. [18] propose the Adaptive Marking Threshold (AMT), which proactively tunes the marking threshold to eliminate unnecessary queuing delay and maintains high link utilization. Yan et al. [19] designed an automatic run-time optimization scheme that leverages the multi-agent reinforcement learning technique to adjust the marking threshold at each switch dynamically. The switches are all capable of ECN marking, so this simple signal is frequently used in algorithms that prevent micro-burst flows. In 2017, Shan et al. [20] proposed the Combined Enqueue and Dequeue ECN Marking (CEDM) strategy based on queue rise slope and double threshold. If the instantaneous queue length is greater than the ECN marking threshold and the queue length does not decrease, the packet is marked; when the packet leaves the queue, the queue length is less than the ECN threshold, or the queue length decreases, the ECN marking is revoked. In 2018, Shan et al. [21] proposed the S-ECN scheme by further studying the generation of microbursts, when the slope s is equal to or greater than the send rate R , all packets are marked ECN, and the traffic is immediately slowed down; when the slope s is lower than the send rate R , then packets are marked ECN with the probability of s/R . Finally, when the slope is lower than or equal to zero, no packets are marked. LossPass was proposed by Kim et al. [22] to break the trade-off of ECN by absorbing microbursts without headroom. Kang et al. [11] proposed a MicroBurst ECN (MBECN) algorithm for the case of false congestion signals generated by microbursts and triggering ECN mismarking.

However, most of the studies on ECN marking strategies cannot adapt to dynamic data center network environments and make too many changes to network hardware devices. They mainly target network measurement, load balancing, less congestion control, and are designed for only one of the network's metrics.

As we saw, numerous flows scheduling schemes have problems realizing them in the real world and cannot be applied to data center networks. Marking packets with priority at the sender side causes some packets to experience multiple queuing delays because the

network side's current link state is unknown. Therefore, we need to design algorithms to use signaling packets such as ECN within the switch in order to understand the link-state and to avoid network congestion. On the other hand, fixed ECN thresholds cannot mitigate microbursts in data center networks, and we need to adjust ECN thresholds dynamically. Datacenter switches can mark ECNs on packets without additional hardware support. Our biggest motivation is to provide a scheme that uses both end-host-side and network-side information. It improves the priority of packets containing signals (ECNs and ACKs), notifying the sender as soon as possible, and dynamically changing the ECN marking threshold to adapt to the network link-state. For this purpose, we propose the HDCQ algorithm, which improves network performance and robustness. In the next section, we specify the design ideas of the HDCQ algorithm.

3. The HDCQ Design

This section introduces the general idea of the HDCQ design and then describes the deployment of algorithms on the end-host side (Section 3.2) and network side (Section 3.3), respectively.

We propose the HDCQ algorithm based on network traffic characteristics and network micro-burst phenomena according to the current stage of data centers. Figure 1 shows the overall framework of the HDCQ algorithm. We only need to configure at the sender and switch sides and require no NIC modification or a customized TCP.

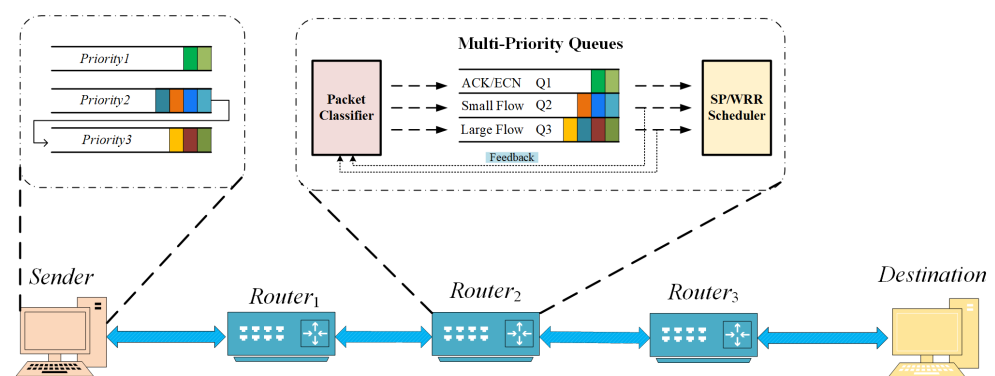


Figure 1. The overall design framework of HDCQ algorithm.

3.1. Overall Scheme

Sender: We used the PIAS [5] algorithm's MLFQ idea, which marks packet priorities from highest to lowest according to the cumulative number of bytes sent by the flow in the face of agnostic flows. The PIAS sets eight priority levels, and the difficulty lies in selecting the threshold value for each priority level. Inaccurate selection of the threshold value or failure to match between priorities can result in loss of switch performance. The HDCQ algorithm establishes only three types of priorities at the end host according to network traffic characteristics: signal, small flow, and large flow packets. On the one hand, such a design approach avoids complex threshold calculations. On the other hand, short flows complete transmission earlier by their high priority [23].

Switch: According to the priority setting of the end host, we create the corresponding three queues in the switch. The high priority queue receives ACKs or packets marked with ECNs, the medium priority queue receives small flows packets, and the low priority queue receives large flows packets. On this basis, in order to effectively deal with data center network congestion, the HDCQ algorithm switches on the switch ECN marking threshold and notifies the sender to reduce the sending rate in time by adjusting the packet priority [9] (Packet Classifier model). To further prevent microbursts effectively, we dynamically change the ECN marking threshold. To additionally avoid starving the long flow for a long time, which affects its throughput, we consider the strict priority out queue or the Weighted Round-Robin (WRR) [24] approach (SP/WRR Scheduler module).

The HDCQ algorithm also guarantees the robustness of the network in the face of multi-congestion. As long as each switch performs ECN marking individually, congestion is recognized independently at each hop. This ensures whether there is only one congestion point or multiple points of congestion, and it will be reported by the corresponding switches to end hosts.

Next, we will follow the general idea of the HDCQ algorithm framework shown in Figure 1, from left to right, from end host to switch, and introduce the specific implementation details of the algorithm in turn.

3.2. Sender Modifications

Designing multiple queues in a switch to meet the demand of different types of traffic is an effective measure to improve network performance [22]. Algorithm 1 shows the specific implementation of the HDCQ algorithm for priority marking of packets at the sender side. First, we set the ACKs from the receiver priority *Priority1* (in the switch, the priority of the packets are marked ECN will adjust to *Priority1*, the specific algorithm is in Section 3.3). Packets containing ACK signals are marked as the highest priority because such packets require a fast response and feedback on the network condition. When a new flow is initialized at the sender side, packets are marked as the second (medium) priority *Priority2*. While the cumulative number of packets of the flow *flow_size* exceeds the parameter *flow_type_thresh* that distinguishes between small and large flows, the subsequent packets of those flows are marked as the third (low) priority *Priority3*. The threshold parameter *flow_type_thresh* between *Priority2* and *Priority3* will be set according to the empirical value [25].

Algorithm 1: Packet Priority Marking Algorithm

```

input : pkt; flow_size; flow_type_thresh;
output: pkt.priority
1 if pkt with ACK info then
2   | pkt.priority  $\leftarrow$  priority1;
3 else if flow_size  $\leq$  flow_type_thresh then
4   | pkt.priority  $\leftarrow$  priority2;
5 else
6   | pkt.priority  $\leftarrow$  priority3;
7 end

```

The three types of priorities established by the HDCQ algorithm at the sender side are further summarized as flowing:

- *Priority1*: The highest priority, which include packets containing ACK or ECN signals, is marked as the highest priority.
- *Priority2*: For small flow priority, the packet marks this priority when the threshold less than *flow_type_thresh* is set by the sender.
- *Priority3*: For the priority of large flows, when the accumulated number of packets exceeds *flow_type_thresh*, the packets coming after are marked with this priority.

We design three queues in the switch *Q1*, *Q2*, and *Q3* to receive packets with the corresponding priority, i.e., *Q1* corresponds to *Priority1* and so on. Current data center network switches support priority scheduling and ECN marking features, so we do not need additional hardware configuration. With the priority scheduling feature, the switch queues packets in order of priority, with the highest priority packets being transmitted first. We choose the SP scheduling method instead of WRR because round-robin scheduling breaks the priority order among packets. For SP scheduling, we set three types of priorities at the sender side, and flows are downgraded from high priority to low priority in order to ensure the order among packets. Next, we will describe the buffer management of the switch, which mainly revolves around the ECN marking threshold.

3.3. Switch Buffer Management

The ECN marking threshold affects the performance of each traffic flow in the network. We design a simple Incast experiment in ns-2 [26] (see Section 4 for the specific introduction) in order to verify it. The parameters switch buffer capacity B and bottleneck link bandwidth C are set to 100 packets and gigabytes. The switch buffer port ECN threshold is adjusted to vary from 20 to 100. The experiment first sends one long flow L of infinite bytes during 0.5 s, followed by 300 short flows S randomly sent in parallel according to the 100 KB uniform distribution condition.

From Figure 2a, the AFCT of short flow S increases with the ECN marking threshold, while the value of small flow FCT is at the maximum when the ECN value is equal to the switch buffer because the ECN threshold does not work at this time. On the other hand, it is observed in Figure 2b that the long flow throughput varies with the ECN marking threshold and has the highest throughput and the lowest packet loss rate when the ECN threshold is set to 100. Yet, the small flow has the highest AFCT at this time because the long flow occupies the link bandwidth to transmit packets, resulting in the small flow packets not being delivered, which affects the FCT. Therefore, when we set the ECN threshold, we need to consider both the AFCT of small flows and the throughput of large flows. Through the experiments, we can observe that the setting of the ECN threshold marking has a significant impact on the network performance.

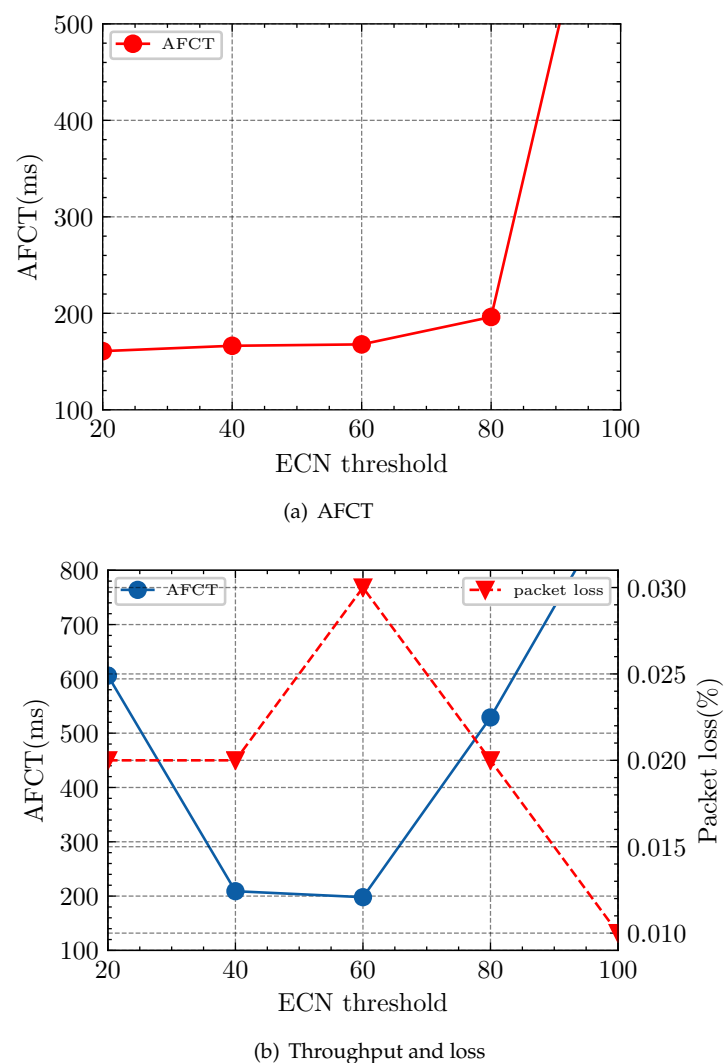


Figure 2. Incast environment, the impact of different ECN thresholds on short-flow AFCT, long-flow throughput, and network packet loss rate.

Next, we will introduce how the HDCQ algorithm modifies the ECN marking threshold at the switch side. We first provide the reasons for enabling ECN marking. Secondly, we present the specific implementation of the HDCQ algorithm for packet priority adjustment. We find that arranging packets marked with ECN to the highest priority and placing them in queue Q1 can effectively prevent microbursts. Then, we describe the algorithm for dynamically adjusting the ECN marking threshold. Finally, we will compare which one of the two queueing methods, SP or WRR, is better at ensuring the throughput of long flows.

3.3.1. ECN Threshold Marking

Related studies [5] have shown that setting the threshold *flow_type_thresh* at the sender side based on network history information or empirical values is efficient and robust to microbursts. However, although existing technologies can collect sufficient network traffic history information, in practice, this information is also unable to predict traffic changes fully, so we need to consider ECN marking within the switch to prevent network microbursts. For this purpose, we introduce two ECN marking threshold parameters, *per-port ECN* and *per-queue ECN*, in the HDCQ algorithm.

- *per-port ECN*. The *per-port ECN* parameter marks the ECN threshold of the switch port, which can provide better burst tolerance and prevent packet overflow when traffic bursts. In addition, since we set high priority packets to dequeue first, there are many low priority packets in the switch. When the arrival of a low-priority packet exceeds the *per-port ECN* threshold, the packet will be marked ECN. Adjusting its priority to the highest priority (*Priority1*) introduces queue Q1, which notifies the sender quickly in order to reduce the sending rate and to prevent the long flows from being starved.
- *per-queue ECN*. We set the ECN marking threshold *per-queue ECN* in queue Q2 because queue Q2 receives packets of small flows with FCT requirements. Establishing a dedicated ECN marking threshold satisfies the network's sensitivity to the FCT of small flows. When the network condition performs well, the *per-queue ECN* can be set to a larger value to make full use of the link bandwidth; conversely, if the current network link condition is terrible, the value of *per-queue ECN* needs to be adjusted to a smaller value in order to reserve space in the switch buffer to prevent packets from overflowing instantaneously, which will result in an increase in the FCT of small flows and a decrease in the throughput of large flows. The above discussion clarifies that the data center network needs to dynamically adjust the *per-queue ECN* in order to reserve enough buffer space.

Noted that whether a small flow packet with *priority2* is marked ECN is decided by *per-queue ECN* and *per-port ECN* together. Switching on these two marking thresholds in the switch ensures that enough packets can be stored in the switch and prevents the switch from having too little free buffer space left to cope with the problem of ECN not being marked in time when network microbursts occur.

3.3.2. Adjusting Packet Priority

Algorithm 2 shows a detailed implementation of adjusting the priority of a packet as it enters the switch. The parameter *Q_size* indicates the number of packets in the switch buffer, and *q2_size* shows the number of packets in the queue Q2.

The meaning of lines 1–3 of the algorithm determines whether an incoming packet will make the buffers exceed the port threshold *per-port ECN* in the switch. If it does, then this means that too many packets are in the link, and they to be marked ECN. Second, we place the packet in the appropriate queue relying on priority (lines 4–21 of Algorithm 2). Queue Q2 is designed for small-flows packets. We need to observe the situation of this queue first. If the packet makes the cumulative number more than the threshold *per-queue ECN*, it will be marked ECN, and the priority will be modified to *Priority1*; otherwise, the ECN of the packet is set to 0 and enqueue Q2. Then, we manage the packet for which its priority is *Priority3*.

If the packet is marked with ECN, its priority is raised to *Priority1*; otherwise, enter the packet into queue Q3. Finally, the packet marked with *Priority1* directly enters into queue Q1.

Algorithm 2: Packet Enqueue Algorithm

```

input : pkt.ecn; pkt.priority; per_port_ecn; q2.per_queue_ecn; Q_size
output: The queue assignment for a packet.
1 if Q_size + pkt.size > per_port_ecn then
2   | pkt.ecn ← 1;
3 end
4 if pkt.proirty == priority2 then
5   | if q2_size > per_queue_ecn then
6     | pkt.ecn ← 1;
7     | pkt.priority ← priority1;
8     | Adjustment(q2.per_queue_ecn);
9   | else
10    | pkt.ecn ← 0;
11    | q2.Enqueue(pkt);
12  | end
13 end
14 if pkt.priority == priority3 then
15   | if pkt.ecn == 1 then
16     | pkt.priority ← priority1;
17   | else
18     | q3.Enqueue(pkt);
19   | end
20 end
21 if pkt.priority == priority1 then
22   | q1.Enqueue(pkt);
23 end
  
```

There are three types of packets in queue Q1: (1) ACK packets; (2) *Priority3* packets that exceed the *per-port ECN* port marking threshold; and (3) *Priority2* packets that exceed the *per-queue ECN* queue marking threshold. Among them, (2) and (3) are packets marked with ECN, and they are placed into queue Q1 to prioritize the sender and to reduce the sending rate and prevent network congestion.

$$F = \begin{cases} 0, & q_2 \leq 0 \\ \frac{q_2(pkt.ecn)}{q_2(pkt)}, & 0 < q_2 \end{cases} \quad (1)$$

The Adjustment function in line eight of Algorithm 2 is for adjusting the *per-queue ECN* dynamically. When the network condition is poor, the queue threshold *per-queue ECN* still maintains the original value, resulting in untimely feedback of marked ECN packets. Hence, we need to reduce the *per-queue ECN* weight in advance. $q_2(pkt)$ means the number of packets in queue Q2 per RTT cycle, and $q_2(pkt.ecn)$ represents the packets marked with ECN, which reflect the congestion of the network. As shown in Equation (1), the current network load is decided by the ratio.

$$\begin{cases} \beta = (1 - k) \times \beta + k \times F \\ \text{per-queue ECN} = \beta \times 100\% \end{cases} \quad (2)$$

$$0 < F < 1, 0 < k < 1, 0 < \beta < 1$$

Since the network is time-varying, the condition in the following period depends on the accumulated network changes in the previous period. As shown in the Equation (2), we perform a smoothing equation, β indicates the packet percentage of the queue, and it is multiplied by 100 is the value of *per-queue ECN*. k is the smoothing parameter that

regulates the weights, and it is taken to prevent the value of β from changing too much each time.

In the actual network, when packets are dequeued, the high sequence number may leave the queue earlier than the lower's, resulting in packet priority reversal, and it affects network performance. If using the SP enqueueing method, although the orderliness between packets is guaranteed [5], it may result in long flow packet starvation. We improve on this basis and propose the SP+WRR dequeuing method. The specific dequeuing scheme is shown in Algorithm 3. Queue Q1 packets are all out of queue first (following the SP method, Algorithm 3, lines 1–3), and then packets in queues Q2 and Q3 are dequeued out following the WRR method. The parameter w controls the dequeuing weights of large and small flows, and after queue Q2 dequeues w packets, queue Q3 dequeues one packet.

Algorithm 3: Packet Dequeue Algorithm

```

input :  $pkt.priority; w; q_1; q_2; q_3$ 
output: Packets that are out of the queue.
1 if  $q_1.length > 0$  then
2   |  $pkt = q_1.Dequeue()$ ;
3 else
4   | if  $q_2.length > 0$  then
5     | if  $w > 0$  then
6       |  $w \leftarrow w - 1$ ;
7       |  $pkt = q_2.Dequeue()$ ;
8       | return  $pkt$ ;
9     | else
10      |  $w \leftarrow 5$ ;
11    | end
12  | end
13  | if  $q_3.length > 0$  then
14    |  $pkt = q_3.Dequeue()$ ;
15  | end
16 end
17 return  $pkt$ ;

```

Keeping the same parameters and environment as is described in Section 4, we compare the SP and SP+WRR dequeuing scheme. The experimental results are shown in Figure 3. Although the AFCT of the SP scheme is slightly smaller than that of SP+WRR, the throughput of the SP+WRR scheme is much better than SP's, which is due to the low latency and high throughput being the two relative performance metrics of data networks. We sacrificed some latency but obtained high throughput in return. The dequeuing method we designed is effective, and when $w = 5$, we obtained the best result.

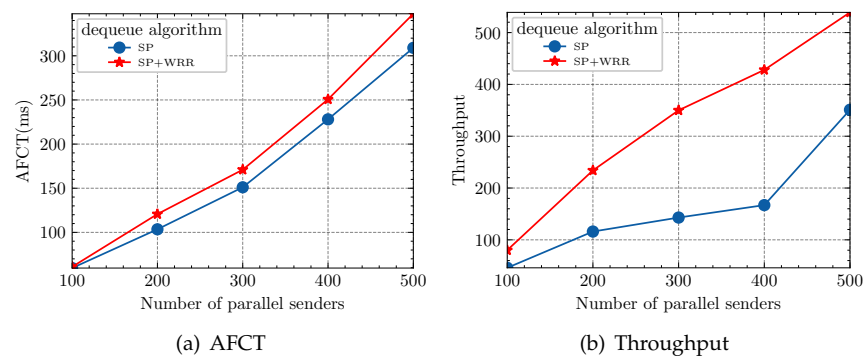


Figure 3. Incast environment: the impact of different dequeue method thresholds on short-flow AFCT and long-flow throughput.

3.3.3. Packet Dequeue Scheduling

In summary, the design idea of the HDCQ algorithm contains the following advantages.

- The HDCQ algorithm sets the signaling packets to the highest priority to prevent congestion in the data center network. The incoming small and large flows are divided by the MLFQ method and marked as *Priority2* and *Priority3* to ensure the AFCT of small flows and the throughput of large flows.
- The HDCQ algorithm creates only three queues in the switch, reducing the space complexity of the algorithm and keeping the switch in a shallow buffer state to ensure low latency.
- The HDCQ algorithm switches on the *per-port ECN* and *per-queue ECN* parameters to adjust the priority of packets according to their ECN marking, which does not bring about priority reversal since only a small number of packets are marked with ECN.
- The HDCQ algorithm uses the SP+WRR queue-out method to prevent the large flows from being starved and by comparing with other experiments prove its effectiveness.

4. The HDCQ's Parameters

Before starting the experiment, we need to determine the values of the parameters of the HDCQ algorithm. We first introduced the setting of the Incast experiment and then analyzed the choice of the HDCQ algorithm's *per-port ECN*, k , and β parameter values.

4.1. Incast Experimental Topology and Parameter

This paper uses NS-2 simulation software to build the experimental environment and establish the Incast topology. As shown in Figure 4, where *Router1* and *Router2* represent the queue of the switch, $S_i (i = 2, 3, \dots, N)$ represents the parallel senders, and R represents the receiver. Incast is a many-to-one communication mode. A parent server initiates a request to a group of nodes, and all nodes respond almost simultaneously, resulting in many nodes sending TCP data flows to one machine. Incast easily results in the switch buffer overflow. We need to verify whether the HDCQ algorithm can effectively reduce network congestion in the Incast environment.

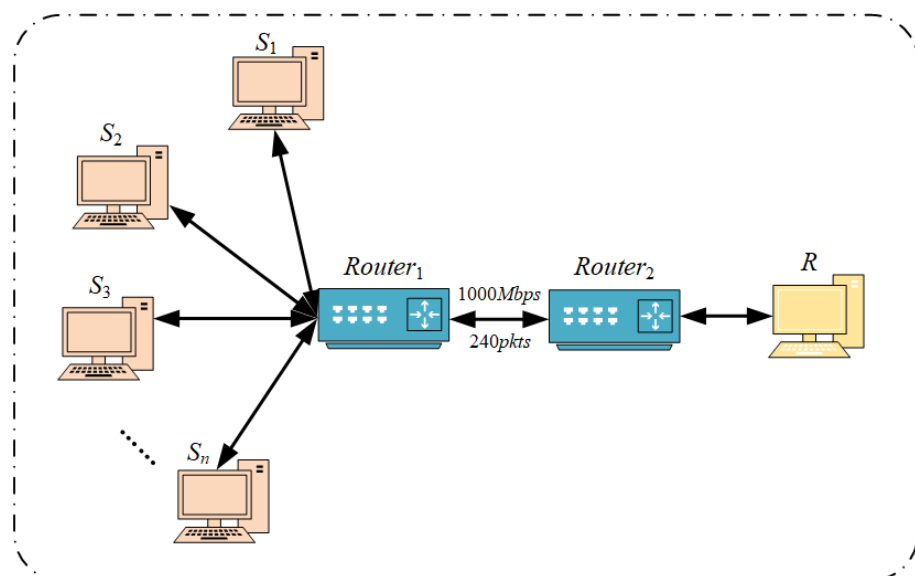


Figure 4. The overall design idea of HDCQ algorithm.

The default parameter values for all experiments in this paper are shown in Table 1. To closely match the actual data center network traffic characteristics, we send short flow packets after the long flow runs for S_{st} seconds.

Table 1. Incast environment: experimental parameters for different ECN marking thresholds.

Parameters	Description	Value
C	Bottleneck link bandwidth	1000 Mbps
B	The size of the buffer that the switch contains	240 pkts
L	Number of long flows	1
S	Number of short flows	100~500
S_{st}	The time when the short flows starts to be sent	0.5 s

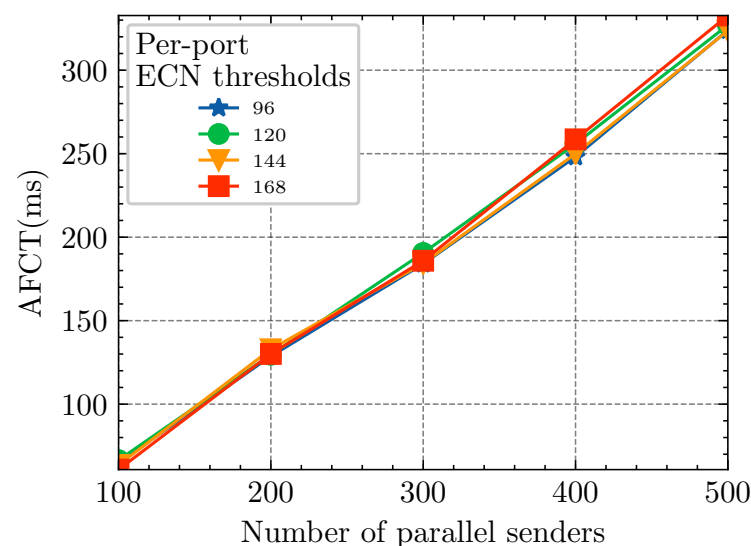
4.2. Parameter Analysis

This section analyzes the effect of *per-port ECN*, β , and k parameters in the HDCQ algorithm. We conduct experiments in the Incast environment. To make the experimental results more distinct, we modify the number of long flows to 3 to build more intense traffic conflicts. The remaining parameter settings are shown in Table 1.

4.2.1. Per-Port ECN

The *per-port ECN* is marked based on the total number of packets in the switch, which controls the rest buffer space. We set the ECN marking threshold to 60% (144 packets) of the switch buffer [12]. To verify its applicability, we set the *per-port ECN* threshold to 40% (96 packets), 50% (120 packets), 60% (144 packets), and 70% (168 packets) of the switch buffer, and the rest experiment parameters are shown in Table 1. Suppose the results show a significant difference between the percentages, the value *per-port ECN* needs to be more fine-grained to find the optimal threshold.

The experimental results are shown in Figure 5. The *per-port ECN* value variation does not affect the network to a great extent because it mainly ensures a certain amount of buffer space within the switch to make full use of the bandwidth. The *per-port ECN* plays a secondary role in the HDCQ algorithm, and whether a packet is marked ECN or not requires further determination by the parameter *per-queue ECN* threshold. For convenience, the subsequent experiments fix the *per-port ECN* as 60%(144 packets) of the switch buffer.

**Figure 5.** AFCT for short flows with different marking thresholds for *per-port ECN*.

4.2.2. β

As is shown in Equation (2), $\text{per-queueECN} = \beta \times 100\%$. We investigate the effect of selecting β primaries on the convergence speed of the HDCQ algorithm by setting 30% (24 packets), 40% (32 packets), 50% (40 packets), 60% (48 packets), and 70% (56 packets)

primaries of a single queue. The results are shown in Figure 6. At first, all initial values of *per-queue ECN* had little effect on the AFCT of small flows, with the number of small flows sent in parallel increasing, and the AFCT is the lowest when the β initial value is set to 40% and 50%. When a number of short flows are sent in parallel $N = 500$ and β is 40%, the AFCT increases suddenly, indicating that this value cannot effectively cope with the gradual increase in network microbursts; thus, we chose $\beta = 50\%$ (40 packets).

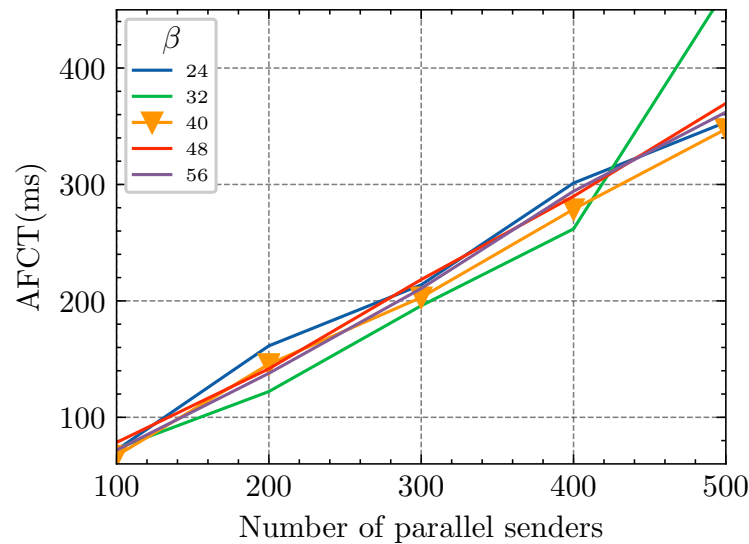


Figure 6. AFCT for short flows with different initial β values.

4.2.3. k

The parameter k is a smoothing parameter for the dynamically adjusted *per-queue ECN* values, and its value should not be large. In this paper, we chose the values of 0.025, 0.05, 0.075, and 0.1, with 0.025 experiments between each value, and fix *per-portECN* = 144 (60% of the switch buffer) and $\beta = 50\%$; the rest of the parameters are shown in Table 1. The results in Figure 7a show that the selection of k value has a significant impact on the experimental results, and the optimal value of k is between [0.05~0.075]. Therefore, we chose 0.055, 0.06, 0.065, 0.07, and 0.075 for further experiments, and the results are shown in Figure 7b. We can find that when $k = 0.06$, the results are all better than the other values.

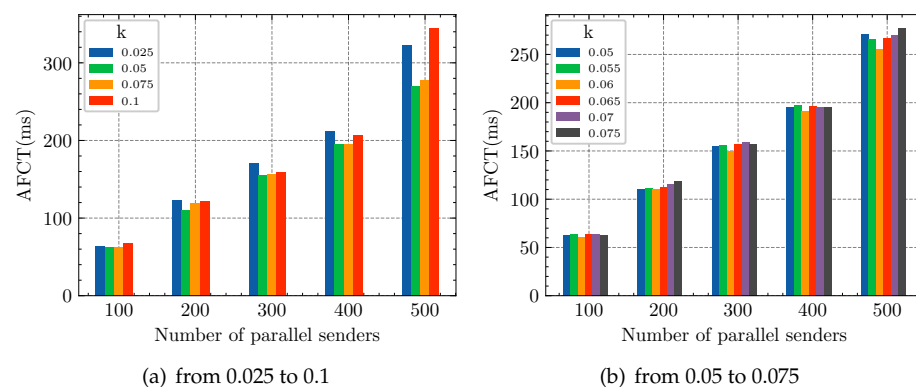


Figure 7. AFCT for short flows with different k values.

5. Experiment Results

In this section, we first conduct two comparison experiments in Incast environment [27]. The first experiment sends concurrent short flows and analyzes the experimental results by comparing HDCQ with RED [28] and PIAS algorithms. After that, we increase the number of small flows and compare the FCT of small flows under various algorithms when the

network state is more deplorable. Finally, we experiment with realistic workloads by using the widely accepted data center traffic distribution, i.e., data mining and web search. We built a client/server model, generated dynamic traffic [24], and measured the FCT.

5.1. Incast Experiment

We control the microburst degree by changing the concurrent number of small flows. The number of long flows is set at $L = 5$, and the rest specific parameters are shown in Table 1. With the parameter analysis in Section 4, we set the variables at $per-portECN = 144$ (60% of the switch buffer), $\beta = 0.5$, and $k = 0.06$.

The experiments first send the packets of long flows to enter the switch. When filling a certain number of buffer, we start sending short flows containing different numbers of bytes randomly in order to recreate the micro-burst phenomenon caused by one node sending requests and multiple nodes correspondingly to the data center.

The results are shown in Figure 8. Figure 8a indicates that the AFCT of short flows in the HDCQ algorithm is better than the RED (sender side is set DCTCP algorithm) and PIAS algorithm. Figure 8b reveals that when the number of short flows is small, the throughput of long flows does not differ much among the schemes. As the number of short flows increases, resulting in the possibility of microbursts in the network, the throughput of long flows of the HDCQ algorithm gradually outperforms the RED and PIAS algorithm. Indicating that the HDCQ algorithm prevents network microbursts effectively.

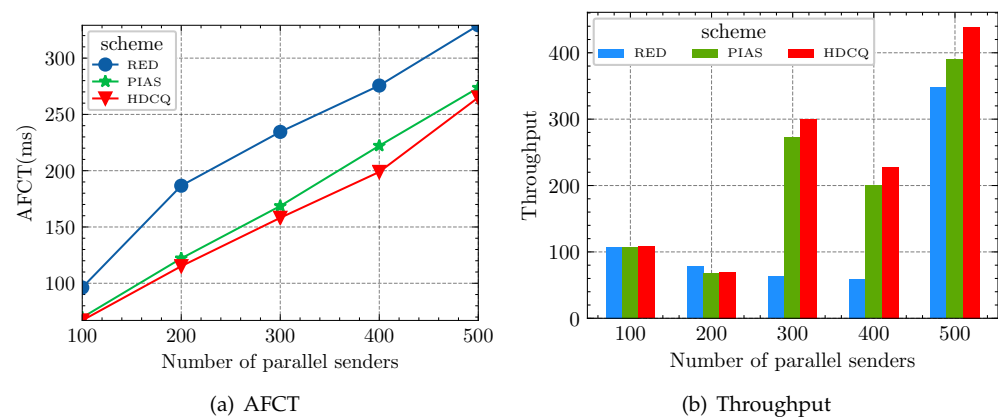


Figure 8. Incast environment: the impact of different scheme on short-flow AFCT and long-flow throughput.

We modify the network parameter $C = 10Gbps$ and $S = 1000 \sim 4000$ flows and observe the performance of RED, PIAS, and HDCQ algorithms in the case of severe network congestion. The results are shown in Figure 9. At $S = 1000$ flows, the AFCT when executing these three schemes is almost the same. However, as the number of concurrent short flows keeps increasing, the AFCT of PIAS and RED algorithms still maintain similar AFCT, while HDCQ is gradually more diminutive than the other two algorithms. At $S = 3000$ flows, the difference between the two reaches the maximum value, i.e., it comes to 24%. Thus, we verify that the HDCQ algorithm outperforms the RED and PIAS algorithms for poor network conditions, indicating that the HDCQ algorithm is robust. The queue management scheme we designed in the switch is reasonable and adequate. The difference in throughput under each algorithm is not significant because of severe network congestion.

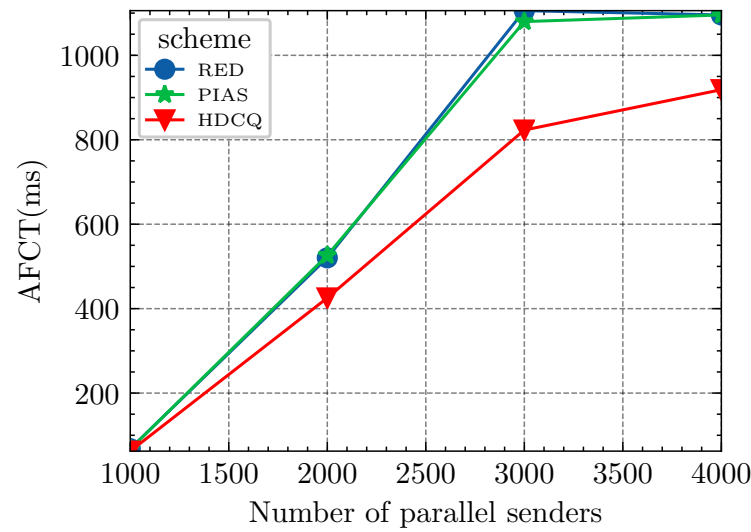


Figure 9. Incast environment, the impact of different scheme on short-flow AFCT.

5.2. Realistic Traffic Model

In the realistic traffic model, the topology is a client application on a machine that connects to the rest of 15 machines by establishing five persistent TCP connections. In the experimental run, the client application periodically generates requests to other machines for data by using available connections, and the application running on the other 15 machines responds with the data. The traffic is sent as Poisson flow, and the sending time of the traffic obeys the negative exponential distribution. The required load size calculates each flow's average sending interval time, which is the negative exponential distribution value. We evaluated the PIAS, DCTCP, LLDCT, and HDCQ algorithms while varying the network loads from 0.1 to 0.9 (0.1 per interval). We ran 10,000 flows for each setting.

Figure 10 provides the overall AFCT for the web search workload and the data mining workload at various loads. Overall, HDCQ performs the best, and the AFCT of HDCQ can be as low as 8.91% in terms of web searching compared to the PIAS algorithm and up to 14.78% compared to the DCTCP algorithm. As for data mining, HDCQ's AFCT can reach as low as 5.46% compared to the PIAS algorithm and up to 11.31% compared to the DCTCP algorithm.

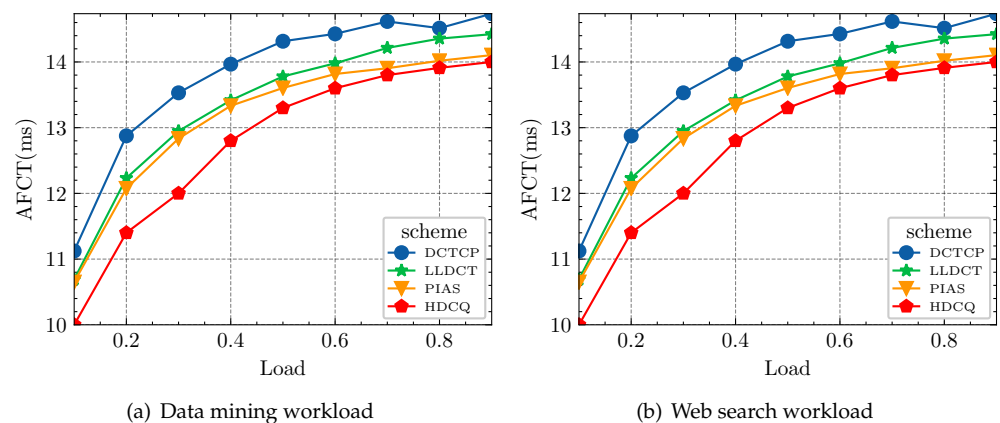


Figure 10. Overall average flow completion times for both workloads.

We further break down the FCT across different flow sizes. Figures 11 and 12 show the FCT across small flows (a,b) 0~100 KB, (c) medium flows 100 KB~10 MB, and (d) large flows 10 MB~∞, respectively. Through conducting experiments, we observed the following: First, in both the average and 99th percentile FCTs of small flows, The HDCQ

algorithm performs slightly better than the PIAS algorithm and far better than the other two algorithms (LLDCT and DCTCP). Figure 11a shows that the AFCT values reduced from 2.11% to 4.70% compared to the PIAS algorithm and 21.74% to 44.25% compared to the DCTCP algorithm. The HDCQ algorithm is similar to the PIAS algorithm and much smaller than the DCTCP algorithm, with a range of 70.20% to 80.10%, as shown in Figure 11b. Similarly, under web search, as shown in Figure 12a, it is 42.19~84.85% smaller than DCTCP. Under 99th percentile FCT of short flows (Figure 12b), it is 80~95% smaller. Second, the HDCQ algorithm also performs well in medium flows, effectively reducing FCT by up to 6.61~24.26% in data mining compared to the LLDCT algorithm and reduces 28.72% compared to the LLDCT in the web search. Third, the HDCQ algorithm performs slightly better than the other three algorithms under FCT of large flows. When designing the HDCQ algorithm, we mainly considered maintaining the throughput of long flows. We effectively prevent microbursts and improved the network's overall performance through methods such as our design of the SP+WRR queuing mechanism and raising the priority of packets marked by the ECN.

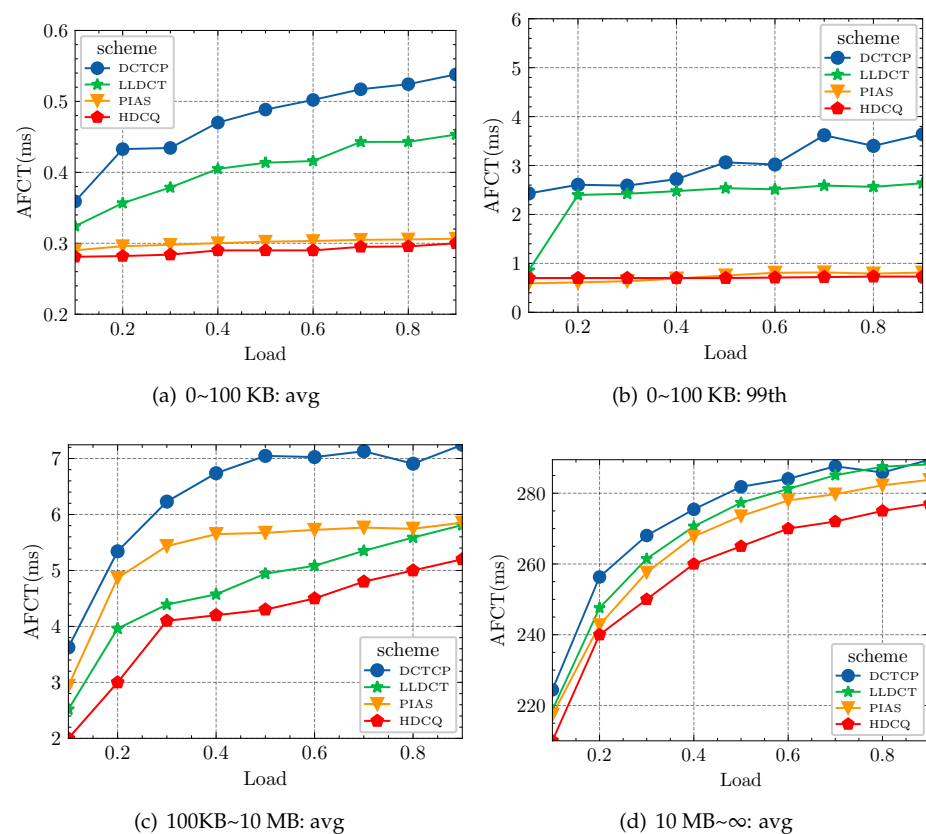


Figure 11. Data mining workload: FCT across different flow sizes.

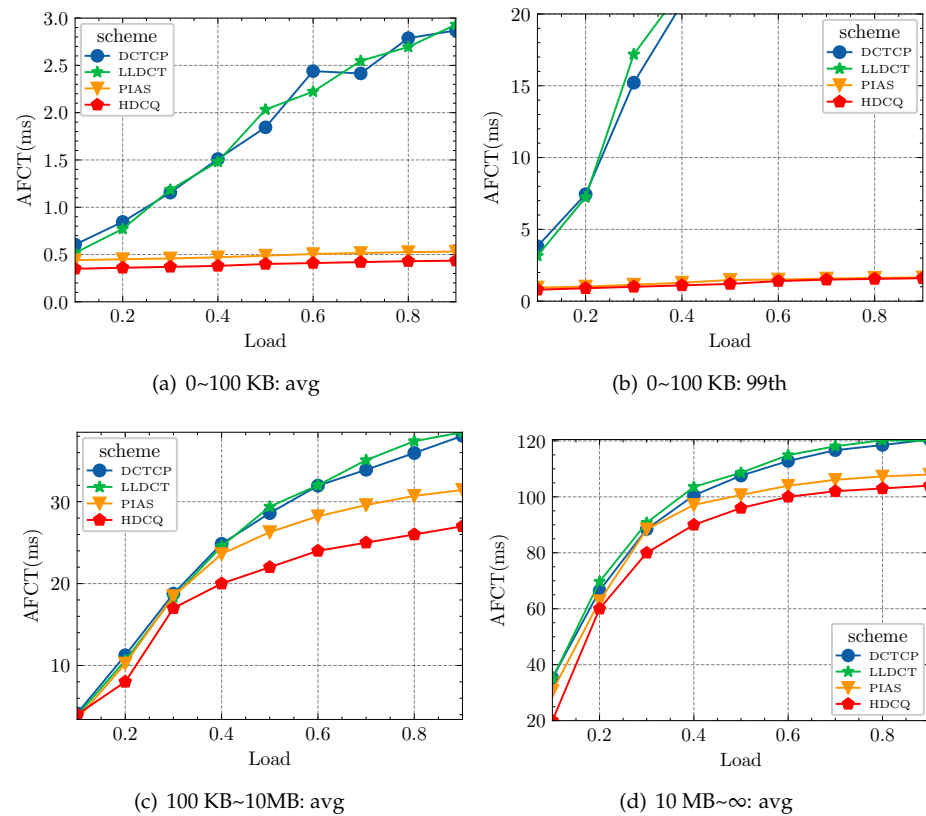


Figure 12. Web search workload: FCT across different flow sizes. The performance of DCTCP and LLDCT is outside the plotted range of (b).

Using large-scale NS-2 simulations, we showed that HDCQ scales to multi-hop topologies and performs the best among all schemes. The HDCQ algorithm speeds up the network TCP control loop in the switch by setting the ACK to the highest priority (*Priority1*) on the sender side and raising its priority within the switch for packets marked with an ECN to maintain the data center network. On the other hand, the PIAS algorithm only deploys the algorithm at the end host and lacks feedback to the network, thus performing worse than the HDCQ algorithm. While helping to speed up the FCT of small flows, it also indirectly improves large flows' FCT. Meanwhile, combined with our SP+WRR out queueing algorithm, the HDCQ algorithm still performs the best in the AFCT for medium and large flows.

6. Conclusions

This paper presents the HDCQ algorithm based on the data center network micro-burst phenomenon, which establishes multi-priority queues for flows based on the different characteristics of ACK/ECN packets, short-flows, and long-flows packets. In the switch, the HDCQ algorithm dynamically adjusts the ECN marking threshold according to the current buffer space and adjusts the packet priority according to the packet features. Experimental results demonstrate that, compared with RED and PIAS algorithms, the HDCQ algorithm proposed in this paper effectively reduces the AFCT of short flows by up to 24% while maintaining the throughput of long flows. We plan to investigate the impact on FCT and throughput when dynamically marking the ECN threshold through extensive experiments in the future, to quantify the results through theoretical analysis, and to develop the HDCQ algorithm in order to enhance the network performance of the data center with few changes to the network.

7. Patents

The authors of this work have applied for a patent and that patent is undergoing substantive examination.

Author Contributions: Conceptualization and methodology, Y.Y. and X.J.; software and validation, Y.Y. and X.J.; formal analysis, Y.Y.; investigation and resources, Y.Y.; data curation, Y.Y. and X.J.; writing—original draft preparation, Y.Y.; writing—review and editing, Y.Y., P.L. and G.J.; visualization, G.J.; supervision, G.J. and Z.G.; project administration, G.J. and X.J.; funding acquisition, G.J. and Z.G. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (61601252), the Natural Science Foundation of Zhejiang Province (LY20F020008, LY21F020006), the Ningbo Natural Science Foundation (202003N4085, 2019A610088), the Ningbo Public Welfare Project (202002n3109), the Basic Scientific Research Project of Wenzhou (G2020021), the Key Laboratory (Engineering Center) Construction Project of Wenzhou (ZD202003), the Ningbo Key Science and Technology Plan (2025) Project (2018B10075, 2019B10125, and 2019B10028), the Special Research Funding from the Marine Biotechnology and Marine Engineering Discipline Group in Ningbo University (422004582), the Quzhou Science and Technology Planning Project (2020k06), Key Project of National Natural Science Foundation of China (u20a20121).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript or in the decision to publish the results.

References

1. Zhang, J.; Yu, F.R.; Wang, S. Load balancing in data center networks: A survey. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 2324–2352. [[CrossRef](#)]
2. Zeng, G.X.; Hu, S.H.; Zhang, J.X.; Chen, K. Transport Protocols for Data Center Networks: A Survey. *J. Comput. Res. Dev.* **2020**, *57*, 74.
3. Rezaei, H.; Vamanan, B. ResQueue: A Smarter Datacenter Flow Scheduler. In Proceedings of the Web Conference 2020 (WWW'20), Taiwan, China, 20–24 April 2020; pp. 2599–2605.
4. Alizadeh, M.; Yang, S.; Sharif, M.; Katti, S.; McKeown, N.; Prabhakar, B.; Shenker, S. pFabric: Minimal near-optimal datacenter transport. In Proceedings of the Special Interest Group on Data Communication (SIGCOMM), Hong Kong, China, 12–16 August 2013; pp. 435–446.
5. Bai, W.; Chen, L.; Chen, K.; Han, D.S.; Tian, C.; Wang, H. PIAS: Practical information-agnostic flow scheduling for commodity data centers. *IEEE/ACM Trans. Netw.* **2017**, *25*, 1954–1967. [[CrossRef](#)]
6. Wilson, C.; Ballani, H.; Karagiannis, T.; Rowtron, A. Better never than late: Meeting deadlines in datacenter networks. In Proceedings of the Special Interest Group on Data Communication (SIGCOMM), Toronto, ON, Canada, 15–19 August 2011; pp. 50–56.
7. Vamanan, B.; Hasan, J.; Vijaykumar, T.N. Deadline-aware datacenter tcp (d2tcp). In Proceedings of the Special Interest Group on Data Communication (SIGCOMM), Helsinki, Finland, 13–17 August 2012; pp. 115–126.
8. Hong, C.Y.; Caesar, M.; Godfrey, P.B. Finishing flows quickly with preemptive scheduling. In Proceedings of the Special Interest Group on Data Communication (SIGCOMM), Helsinki, Finland, 13–17 August 2012; pp. 127–138.
9. Rezaei, H.; Malekpourshahraki, M.; Vamanan, B. Slytherin: Dynamic, network-assisted prioritization of tail packets in datacenter networks. In Proceedings of the 2018 27th International Conference on Computer Communication and Networks (ICCCN), Hang Zhou, China, 30 July–2 August 2018; pp. 1–9.
10. Shan, D.; Ren, F.; Cheng, P.; Shu, R.; Guo, C.X. Micro-burst in data centers: Observations, analysis, and mitigations. In Proceedings of the 2018 IEEE 26th International Conference on Network Protocols (ICNP), Cambridge, UK, 24–27 September 2018; pp. 88–98.
11. Kang, K.; Zhang, J.H.; Jin, J.H.; Shen, D.; Luo, J.Z.; Li, W.X.; Wu, Z.A. MBECN: Enabling ECN with micro-burst traffic in multi-queue data center. In Proceedings of the IEEE International Conference on Cluster Computing, Albuquerque, NM, USA, 23–26 September 2019; pp. 1–12.
12. Alizadeh, M.; Greenberg, A.; Maltz, D.A.; Padhye, J.; Patel, P.; Prabhakar, B.; Sengupta, S.; Sridharan, M. Data center tcp (dctcp). In Proceedings of the ACM SIGCOMM 2010 Conference, New Delhi, India, 30 August–3 September 2010; pp. 63–74.
13. Munir, A.; Qazi, I.A.; Uzmi, Z.A.; Mushtaq, A.; Ismail, S.N.; Iqbal, M.S.; Khan, B. Minimizing flow completion times in data centers. In Proceedings of the 32nd IEEE International Conference on Computer Communications, Turin, Spain, 14–19 April 2013; pp. 2157–2165.

14. Giannakas, F.; Troussas, C.; Voyiatzis, I.; Sgouropoulou, C. A deep learning classification framework for early prediction of team-based academic performance. *Appl. Soft Comput.* **2021**, *106*, 107355. [[CrossRef](#)]
15. Giannakas, F.; Troussas, C.; Krouska, A.; Sgouropoulou, C.; Voyiatzis, I. XGBoost and Deep Neural Network Comparison: The Case of Teams' Performance. In Proceedings of the 2021 International Conference on Intelligent Tutoring Systems, Cambridge, UK, 7–11 June 2021; pp. 343–349.
16. Liu, Z.F.; Sun, J.S.; Hu, S.Q.; Hu, X.L. An Adaptive AQM Algorithm Based on a Novel Information Compression Model. *IEEE Access* **2018**, *6*, 31180–31190. [[CrossRef](#)]
17. Alizadeh, M.; Kabbani, A.; Edsall, T.; Prabhakar, B.; Vahdat, A.; Yasuda, M. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In Proceedings of the Usenix Conference on Networked Systems Design and Implementation (NSDI), San Jose, CA, USA, 25–27 April 2012; pp. 253–266.
18. Zhang, T.; Wang, J.X.; Huang, J.W.; Huang, Y.; Chen, J.; Pan, Y. Adaptive marking threshold method for delay-sensitive TCP in data center network. *J. Netw. Comput. Appl.* **2016**, *61*, 222–234. [[CrossRef](#)]
19. Yan, S.Y.; Wang, X.L.; Zheng, X.L.; Xia, Y.B.; Liu, D.R.; Deng, W.S. ACC: Automatic ECN tuning for high-speed datacenter networks. In Proceedings of the 2021 ACM SIGCOMM 2021 Conference, Online, 23–27 August 2021; pp. 384–397.
20. Shan, D.; Ren, F. Improving ECN marking scheme with micro-burst traffic in data center networks. In Proceedings of the IEEE Conference on Computer Communications (INFOCOM), Atlanta, GA, USA, 1–4 May 2017; pp. 1–9.
21. Shan, D.; Ren, F. ECN Marking With Micro-Burst Traffic: Problem, Analysis, and Improvement. *IEEE/ACM Trans. Netw.* **2018**, *26*, 1533–1546. [[CrossRef](#)]
22. Kim, G.; Lee, W. Absorbing microbursts without headroom for data center networks. *IEEE Commun. Lett.* **2019**, *23*, 806–809. [[CrossRef](#)]
23. Peng, Y.; Chen, K.; Wang, G.H.; Bai, W.; Ma, Z.Q.; Gu, L. Hadoopwatch: A first step towards comprehensive traffic forecasting in cloud computing. In Proceedings of the IEEE Conference on Computer Communications (INFOCOM), Toronto, ON, Canada, 27 April–2 May 2014; pp. 19–27.
24. Bai, W.; Chen, K.; Chen, L.; Kim, C.H.; Wu, H.T. Enabling ECN over generic packet scheduling. In Proceedings of the 12th International Conference on emerging Networking Experiments and Technologies, Irvine, CA, USA, 12–15 December 2016; pp. 191–204.
25. Noormohammadpour, M.; Raghavendra, C. Comparison of flow scheduling policies for mix of regular and deadline traffic in datacenter environments. *arXiv* **2017**, arXiv:1707.02024.
26. The Network Simulator NS-2. Available online: <http://www.isi.edu/nsnam/ns/> (accessed on 4 November 2011).
27. Almasi, H.; Rezaei, H.; Chaudhry, M.U.; Vamanan, B. Pulser: Fast Congestion Response using Explicit Incast Notifications for Datacenter Networks. In Proceedings of the 2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), Paris, France, 1–3 July 2019; pp. 1–6.
28. Floyd, S.; Jacobson, V. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.* **1993**, *1*, 397–413. [[CrossRef](#)]