Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.056925

ARTICLE

An Adaptive Congestion Control Optimization Strategy in SDN-Based Data Centers

Jinlin Xu[1,2], Wansu Pan[1,*], Haibo Tan[1,2], Longle Cheng[1] and Xiaofeng Li[1,2]

[1]Hefei Institutes of Physical Science, Chinese Academy of Sciences, Hefei, 230031, China

[2]University of Science and Technology of China, Hefei, 230026, China

*Corresponding Author: Wansu Pan. Email: wspan@hfcas.ac.cn

Received: 02 August 2024 Accepted: 29 September 2024 Published: 18 November 2024

ABSTRACT

The traffic within data centers exhibits bursts and unpredictable patterns. This rapid growth in network traffic has two consequences: it surpasses the inherent capacity of the network's link bandwidth and creates an imbalanced network load. Consequently, persistent overload situations eventually result in network congestion. The Software Defined Network (SDN) technology is employed in data centers as a network architecture to enhance performance. This paper introduces an adaptive congestion control strategy, named DA-DCTCP, for SDN-based Data Centers. It incorporates Explicit Congestion Notification (ECN) and Round-Trip Time (RTT) to establish congestion awareness and an ECN marking model. To mitigate incorrect congestion caused by abrupt flows, an appropriate ECN marking is selected based on the queue length and its growth slope, and the congestion window (CWND) is adjusted by calculating RTT. Simultaneously, the marking threshold for queue length is continuously adapted using the current queue length of the switch as a parameter to accommodate changes in data centers. The evaluation conducted through Mininet simulations demonstrates that DA-DCTCP yields advantages in terms of throughput, flow completion time (FCT), latency, and resistance against packet loss. These benefits contribute to reducing data center congestion, enhancing the stability of data transmission, and improving throughput.

KEYWORDS

Data centers; SDN; TCP congestion control; RTT; ECN

1 Introduction

The majority of switches in data centers are shallow buffered Ethernet switches with limited resources, whereas switches with big buffers and numerous ports are more expensive. The all-to-one communication mode in data centers is prone to switch buffer overflow, resulting in throughput crashes and Transmission Control Protocol (TCP) Incast [1]. Additionally, the mixed flow of elephant and mouse flows in the data center makes it easy for mouse flows on one port to be affected by elephant flow activity on other ports. Elephant flows use a lot of shared memory pools and create queues on their ports, which makes the switch less able to handle burst flows and causes issues like packet loss and queue latency.

Scholars have proposed numerous research studies on congestion in data centers, including the active queue management mechanism [2], improved TCP [3,4], and Explicit Congestion Notification (ECN) [5]. The shallow caching nature of data centers may limit the efficacy of optimization schemes in responding to congestion and adapting to burst flows promptly. This can result in issues such as queue accumulation and switch cache overflow, which can significantly reduce network utilization and transmission performance [6–8]. Other researchers have designed protocols that differ from existing TCP [9–11]. However, TCP is currently the most widely used transmission protocol on the Internet, and the cost of replacing it is considerable, making it challenging to implement these schemes [12]. In conclusion, novel research concepts and technical solutions must be developed to address the data center network congestion issue.

A unique solution to data center congestion is provided by the introduction of Software Defined Network (SDN) technology, which has programmability and centralized control [13]. Firstly, the SDN controller can accurately comprehend flow information in the network, thereby facilitating a more expedient and precise determination of the extent of network congestion. Secondly, an SDN system may help spread congestion feedback quickly, which can shorten transmission times and allow the terminal to quickly modify the sending rate.

It is a very significant solution to use SDN controllers to assist and lessen the impact of congestion in the data center. Hence, we have designed a dynamic adaptive congestion control strategy called DA-DCTCP, which can be deployed to the controller through OpenFlow to collect network metrics. DA-DCTCP employs the network attributes acquired by the SDN controller to compute and adjust congestion control parameters that are optimal for the current network state. DA-DCTCP integrates Round-Trip Time (RTT) and ECN mechanisms, utilizing RTT to characterize network conditions and determining ECN marking strategies based on queue length and queue growth slope to adjust congestion window (CWND). Concurrently, to accommodate the evolving state of the data center, a queue length marking threshold adaptive dynamic adjustment optimization model is established with the current queue length of the switch as the perception parameter and the sum of all queue buffer usage rates as the target variable. This model enables precise control of data flow and meets different service-level requirements.

The rest of the paper is structured as follows. Section 2 summarizes the related work. Section 3 presents the design of an adaptive CWND regulation optimization model based on RTT and ECN. In Section 4, experimental results are presented. We conclude the paper in Section 5.

2 Related Work

With the advantage SDN has in the data center, many congestion control schemes have been proposed by researchers. The OpenTCP [14] is the first suggested SDN-based congestion control architecture, and it allows for dynamic adjustment of TCP parameters based on network status. Jouet et al. proposed Omniscient TCP (OTCP) [15], which makes use of the OpenFlow protocol to gather switch status data and distribute it to the host. The eSDN [16] framework introduces the concept of lightweight end-host controllers to supplement the centralized SDN controller. Scalable Congestion Control Protocol (SCCP) [17] is a scheme that exploits SDN to measure the fair share of all TCP flows traversing the output ports of SDN switches by extending the OpenFlow specifications. SDN-based Incast Congestion Control (SICC) [18] mainly relies on the SDN control plane to oversee the influx of SYN/FIN packets and regularly monitor the OpenFlow switch queues occupancy to infer the start of incast epochs before flows start sending data into the network. The TCP Congestion Control based on SDN (STCC) [19] monitors network performance through centralized control and a global network

view of SDN and employs a routing algorithm based on the minimum path bandwidth utilization rate to forward packets. These solutions lack fairness and require high accuracy in traffic detection, which can potentially result in suboptimal bandwidth utilization.

Other solutions involve utilizing various parameters and characteristics of TCP congestion mechanisms to adjust the CWND of the sender or receiver as a means to proactively reduce the transfer rate before congestion occurs. The common TCP in the data centers uses RTT to reflect the degree of network congestion and to adjust the sending window [20]. The initial TCP designed for data centers is Data Center TCP (DCTCP [3]), which employs the ECN congestion control mechanism to feedback on the level of network congestion, maintain the switch queue length at a low value, and maximize network throughput while evenly allocating bandwidth between flows. Deadline-Aware Datacenter TCP (D^2TCP) [4] is a mechanism that adjusts the sending window based on the deadline and CWND. Zou et al. [21] proposed a general supporting scheme Adaptive Pacing (AP), which dynamically adjusts burstiness according to the flow concurrency. Zhang et al. [10] designed a Fast and Friendly Converging (FFC) protocol, which makes independent decisions and adjustments by retrieving the two-dimensional congestion notification from both RTT and ECN. Chen et al. [22] presented Time-Division TCP (TDTCP), which multiplexes each connection across multiple independent congestion states. Bai et al. [23] designed MQ-ECN for congestion control based on forwarding notifications from switches. Majidi et al. [24] proposed ECN threshold for marking-aware optimization (ECN+), which theoretically analyzes the ratio of marked packets in multiple queues and subtracts the ratio from the output port buffer. The Explicit Centralized Congestion Avoidance Mechanism for TCP(ECTCP) [25] used centralized control to calculate the fair bandwidth of each flow and uses a receiving window to allocate bandwidth, changing the traditional TCP's preemption of bandwidth. Aina et al. [26] proposed Fair Data Center TCP (F-DCTCP), which monitors the possibility of congestion in all switches in the controller domain and actively reduces the sending rate of participating flows to avoid congestion. Reinforcement learning and SDN-aided Congestion Avoidance Tool (RSCAT) [27] uses data classification to determine if the network is congested and actor-critic reinforcement learning to find better TCP parameters.

The major difference between our work and previous methods is that our goal is to improve the accuracy of CWND regulation by designing a new TCP congestion avoidance mechanism. DA-DCTCP does not require modifying the TCP stack and is easy to implement in SDN-based data centers.

3 Scheme Architecture

TCP congestion control based on ECN mechanism can only reflect whether the network is congested, without knowing the delay and cannot effectively reduce the FCT. The RTT-based TCP congestion control cannot reflect whether the flow encounters high latency and other issues. When the delay increases, it cannot provide precise control over the sending window [28,29]. ECN and RTT each have limitations, but they can provide useful information about network status. RTT acts as the congestion control signal to correct possible misjudgments caused by ECN, while ECN helps RTT improve the precision of the CWND control.

The DA-DCTCP congestion control strategy is designed based on DCTCP. This strategy leverages SDN's global network view and centralized control features, along with integrating the RTT change gradient and ECN marking mechanism, to reduce congestion. Considering data center traffic's high dynamics and low latency, DA-DCTCP grades the network congestion state by RTT and ECN, and modifies the CWND in time to limit the transmission rate that consumes switch buffer resources.

We adopt the standardized OpenFlow protocol and SDN architecture which typically contains three components: the networking controller, OpenFlow switches, and end hosts. Its architecture is displayed in Fig. 1. The global information caching, congestion control, routing policy, and topology management modules are among the modules in the SDN control layer. The global information caching module periodically sends query requests to switches to obtain link information; the congestion control module gets network state information and modifies the CWND following the DA-DCTCP; the routing policy module calculates the cost of each port on nodes to direct flow forwarding and management; and the topology management module is in charge of overseeing all routing nodes and global topology.

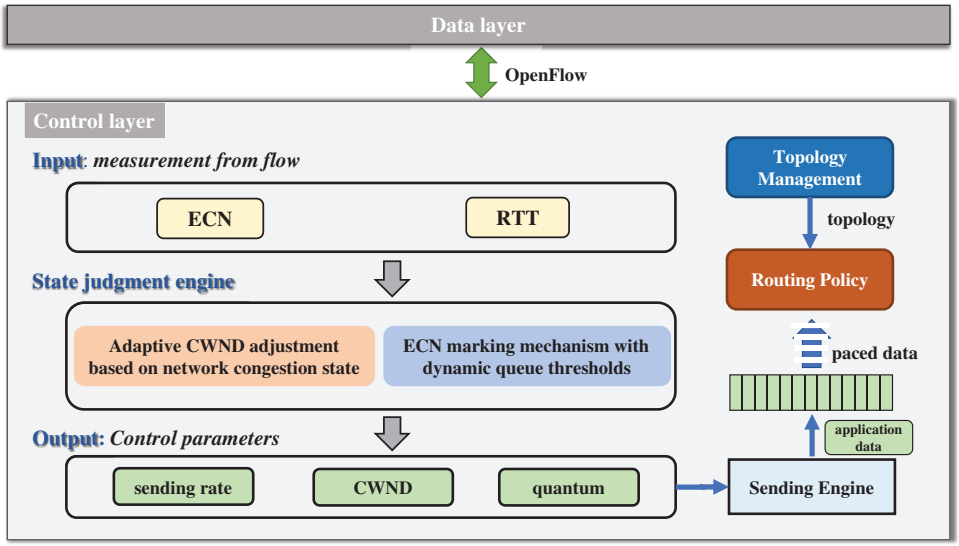


Figure 1: The overall architecture of DA-DCTCP congestion control

3.1 Adaptive CWND Adjustment Mechanism with Flow-Aware

The standard ECN mechanism significantly relies on the marking method and threshold employed. Queue marking methods maintain queue isolation and can provide more precise control over the length of individual queues. Typically, standard ECN utilizes lower fixed thresholds to achieve low latency. However, improper threshold settings can lead to marking errors that cause switch buffer overflows and severe throughput loss. To reduce the occurrence of ECN marking errors, a queue growth slope-based ECN marking and a dynamic queue length threshold are adopted. Moreover, incorporating RTT parameters to categorize the network status into different levels of congestion prevents the unnecessary reduction of CWND consumption. By using both RTT and ECN feedback, CWND control is accomplished more precisely, improving congestion control efficacy and precision.

The ECN marking rate (α_{ecn}) is calculated as:

$$\alpha_{ecn} = (1 - g) \times \alpha_{ecn} + g \times F \tag{1}$$

where F is the score of the group marked in the last window, and g is the weight factor, which should satisfy the following:

$$g < 1.386/\sqrt{2\,(C \times RTT + K)} \tag{2}$$

where C is the link capacity, K is the queue length, and in DCTCP, the value of g is 1/16.

In the ECN mechanism, K plays a role in controlling queue length. To ensure sufficient space to absorb the impact of sudden traffic without causing additional queuing delays, and to ensure 100% throughput, the queue cannot be empty. Therefore, K should satisfy the following constraints, where K_1 is the minimum queue length and K_2 is the maximum queue length.

$$0 \leq K_1^j(t) \leq K_j \leq K_2^j(t) \tag{3}$$

However, due to the delay in network information transmission, the accumulated length of switch queues often leads to rate mismatch in dynamic networks. In a brief amount of time, the burst flows can exceed the K threshold, producing a significant volume of false congestion information. A queue growth slope-based ECN marking judgment is presented to prevent incorrect ECN marking. Assuming the queue length threshold K is K_1 and $K_2 = 2 * K_1$. K_2 is used to prevent the current queue state's burst flow from causing inaccurate congestion information to arise. When the packet is out of the queue, calculate its queue growth slope K_{slope}:

$$K_{slope} = \frac{q_j(t) - q_j\left(t_{pre}\right)}{t - t_{pre}} \tag{4}$$

where t_{pre} is the previous sampling time. Then using Exponential Weighted Moving Average (EWMA) to calculate the average slope avg_K_{slope}:

$$avg\_K_{slope} = (1-\delta)\, avg\_K_{slope}^{pre} + \delta K_{slope} \tag{5}$$

where δ is the weighted coefficient and $0 < \delta < 1$, taken as 0.5 in the paper. And $avg\_K_{slope}^{pre}$ is the previous avg_K_{slope}. Combining the transmission rate for ECN marking judgment, the flow will mark the probability based on the K and avg_K_{slope}, as shown in Eq. (6):

$$p_i(t) = \begin{cases} 1; & K_1 \leq q(t) < K_2 \,\&\&\, K_{slope} \geq 0 \\ 1; & q(t) > K_2 \\ 0; & \text{else} \end{cases} \tag{6}$$

When the current queue length is greater than K_1 and less than K_2, and K_{slope} is greater than 0, the data packet is marked with ECN. When the current queue length is greater than K_2, the data packet is marked with ECN. All other states are considered as non-congested or temporarily incorrectly congested.

The CWND can be adjusted based on the ECN marking and combined with RTT to ascertain the network status. RTT can be used to quantify the link utilization at the bottleneck, and it shows a degree of congestion, as shown in Fig. 2.

The RTT can be used to quantify the congestion level, the RTT of the flow at time t is:

$$RTT_i(t) = \frac{Q_i(t)}{C} + RT_{\text{prop}} \tag{7}$$

where $Q_i(t)$ is the instantaneous queue length at time t, and RT_{prop} represents the minimum round-trip delay. To obtain an accurate RTT, we modify the timestamp counter to 100-ns granularity to obtain accurate RTT. The link utilization U is expressed as the percentage of RTT and the maximum link delay $RTT_{\max}$:

$$U = \frac{RTT_{\text{i}}}{RTT_{\max}} \tag{8}$$

To balance bandwidth usage and rate of convergence, the link state threshold should be the intersection of the network in the idle and fully loaded states. In this paper, this threshold is set to 0.8 based on the network utilization and fair convergence time analysis of U threshold in reference [30].

The current RTT is marked as *cur_rtt*, the previous RTT is marked as *pre_rtt*. The gradient of RTT change G is:

$$G = \frac{cur\_rtt - pre\_rtt}{RTT_{\min}} \tag{9}$$

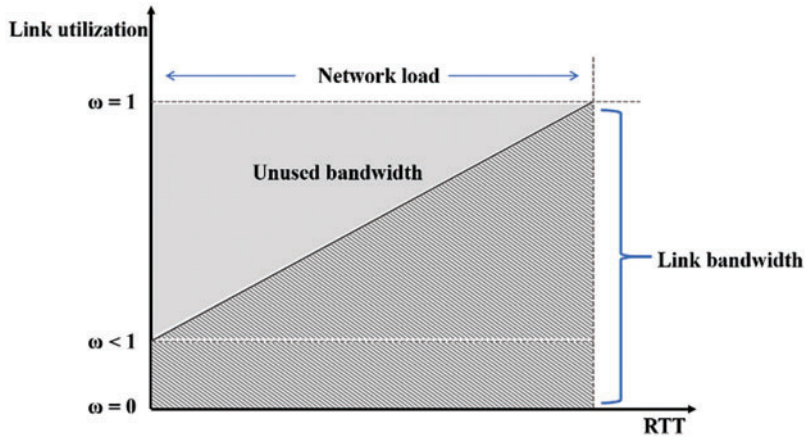


Figure 2: The schematic diagram of the queue growth slope

In the context where $G \leq 0$, it indicates that the network condition has improved, and DA-DCTCP will allow for reducing smaller CWND to maintain stable transmission rates. Conversely, when $G > 0$, it indicates an escalating congestion level within the network. As congestion severity continues to worsen, it becomes imperative to significantly reduce the CWND.

As demonstrated by Algorithm 1, the CWND adjustment mechanism is optimized based on ECN and RTT parameters. RTT is used to grade the congestion level, making long flows more vulnerable to CWND limiting while fully safeguarding the transmission speed of short flows. The queue growth slope-based ECN marking detects the queue information and selectively performs ECN marking. This strategy avoids blind window shrinkage at the sender, while alleviating the convergence issues between flows at different times, improving the network state adjustment capability.

Algorithm 1: The adjustment strategy of CWND

Input: F, *cur_rtt* // Every ACK update
Output: *CWND*
1: **for** every ACK **do**
2: $U = \frac{RTT_{\mathrm{i}}}{RTT_{\max}}$
3: $G = \frac{cur\_rtt - pre\_rtt}{RTT_{\min}}$ // *Initialize distance*
4: $\alpha_{ecn} = (1 - g) \times \alpha_{ecn} + g \times F$
5: **if** $\alpha_{ecn} >$ 0.125 of delivered packets for 2 consecutive RTTs **then**

(Continued)

Algorithm 1 (continued)

```
    if U > 0.8 && G > 0 then     // Determine congestion level
        CWND = CWND/2   // Half the current CWND
    end if
    if U < 0.8 and G > 0 then
        CWND = CWND × (1 − α_ecn/2);
    end if
    else
        CWND = CWND × (1 − α_ecn/4);   //Increase congestion window
    end if
else
    if G > 0 then     // Determine congestion level
        CWND = CWND × (1 − α_ecn/4);   // Half the current CWND
    end if
    else
        CWND = CWND +1;
    end if
end for
```

3.2 Dynamic Queue Length Threshold Adjustment Model

The load on each queue varies due to the constant dynamic nature of the data center, and choosing the right queue length threshold is critical. If the threshold is too low, the sender may abruptly drop the rate, underutilizing the bottleneck link; if the threshold is too high, packets accumulate in the buffer, increasing latency. To address this issue, a dynamic queue length threshold adjustment method is proposed to control the occupied buffer size by controlling the queue length threshold.

In switches, the instantaneous queue length varies in response to changes in the network state. A long queue length indicates a high queue load, while a low queue length suggests an idle queue. The instantaneous queue length can be used to calculate the buffer idle rate:

$$\eta = \frac{Q_j}{K_j} \tag{10}$$

The goal of the dynamic queue length adjustment model is to maximize the η, which indicates how the switch queue is currently using the buffer. The queue length threshold must meet the following restrictions:

$$s.t \begin{cases} K_{port} = \sum K_j \text{ (a)} \\ K_j > q_j > 0 \text{ (b)} \end{cases} \tag{11}$$

where K_{port} is the threshold value calculated in port as the marker unit, representing the sum of the theoretical queue outgoing rate. For restriction (a), it is necessary to ensure that the queue in rate and queue out rate of the switch queue are equal after each dynamic adjustment, to avoid congestion or resource waste due to rate mismatch. The restriction (b) is to ensure that the queue length is always greater than 0, and it is ensured that before and after the threshold adjustment, packets that are not marked with congestion marks will not be erroneously marked with congestion marks because the threshold decreases. Based on the above restrictions, the model is further solved. Defined K_j^{rest} as the

buffer rest of queue j, and calculated as follows:

$$\begin{cases} K_j^{rest} = 0, & q_j \geq K_j \\ K_j^{rest} = K_j - q_j, & q_j < K_j \end{cases} \tag{12}$$

Defined K_j^{over} as the buffer overflow of queue j, and calculated as follows:

$$\begin{cases} K_j^{over} = q_j - K_j, & q_j \geq K_j \\ K_j^{over} = 0, & q_j < K_j \end{cases} \tag{13}$$

The dynamic threshold K is calculated by a heuristic algorithm:

$$K_j^{new} = \begin{cases} K_j^{baseline} + \dfrac{q_j}{\sum q_j} \times \sum K_j^{rest}, & q_j > K_j \,\&\&\, \sum q_j \geq K_{port} \\ q_j + \dfrac{q_j}{\sum q_j} \times \sum K_j^{rest}, & q_j \leq K_j \,\&\&\, \sum q_j \geq K_{port} \\ q_j - \dfrac{q_j}{\sum q_j} \times \sum K_j^{rest}, & q_j > K_j \,\&\&\, \sum q_j < K_{port} \\ K_j^{baseline} - \dfrac{q_j}{\sum q_j} \times \sum K_j^{rest}, & q_j \leq K_j \,\&\&\, \sum q_j < K_{port} \end{cases} \tag{14}$$

Algorithm 2 demonstrates how to adjust the dynamic queue length threshold to fully utilize switch resources and reduce congestion in high-load queues. Idle queue buffer resources are reassigned to busy queues, and the switch buffer is reallocated by dynamically adjusting the switch queue threshold.

Algorithm 2: Dynamic queue length threshold adjustment

Input: *The instantaneous queue length* $\boldsymbol{q[N]}$*; queue baseline* $\boldsymbol{K^{baseline}[N]}$*;the old threshold* $\boldsymbol{K^{old}[N]}$

Output: *N queues' new threshold baseline* $\boldsymbol{K^{new}[N]}$

Scheduler(*queue*[N]);
for each $q_j \in q[N]$, $K_j^{old} \in K^{old}$ **do**
  $total\_q \ += q_j$
  $total\_k \ += K_j^{old}$
  if $q_j > K_j^{old}$ **then**
    $K_{sum}^{over} += q_j - K_j^{old}$
  else
    $K_{sum}^{rest} += K_j^{old} - q_j$
  end if
  if $total_q \geq total\_k$ **then**
    for each $q_j \in q[N]$ **do**
      if $q_j > K_j^{old}$ **then**
        $K_j^{new} = K_j^{baseline} + \dfrac{q_j}{total\_q} \times K_{sum}^{rest}$
      else
        $K_j^{new} = q_j + \dfrac{q_j}{total\_q} \times K_{sum}^{rest}$
      end if

(Continued)

Algorithm 2 (continued)

```
      end for
    else
      for each q_j ∈ q[N] do
        if q_j > K_j^old then
          K_j^new = q_j − (q_j / total_q) × K_sum^over
        else
          K_j^new = K_j^baseline − (q_j / total_q) × K_sum^over
        end if
      end for
    end if
end for
```

4 Evaluations

4.1 Experiment Setup

We implemented the DA-DCTCP mechanism by using Mininet. The network controller uses Ryu v4.3, the OpenFlow switch uses Openswitch v2.0.2, and the OpenFlow protocol version is 1.4. We build our experiment Fat-Tree topology (Fig. 3) with 4 core switches, 8 aggregate switches and 8 edge switches, where each edge switch connects 2 hosts.

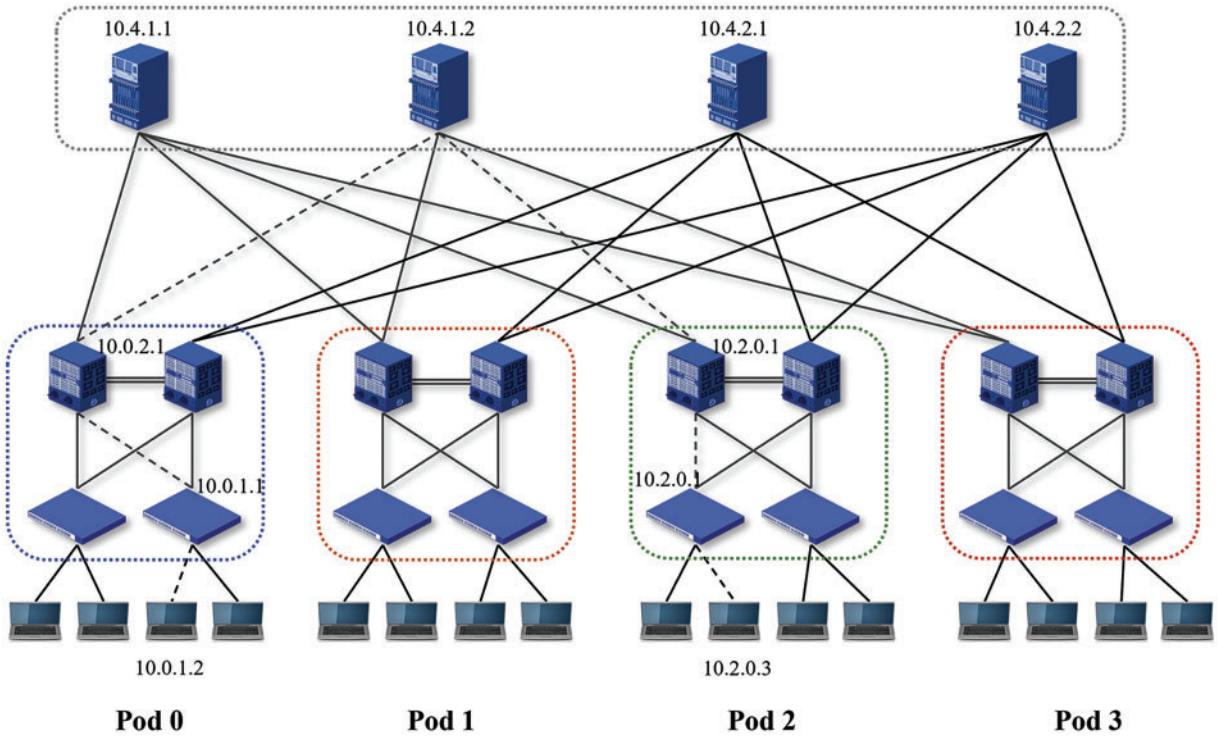


Figure 3: The fat-tree topology (n = 4)

For DCTCP implementation, we use public code from [31] and add ECN capability to SYN packets. In DCTCP, the switch queue length threshold is set to 20 packets (30 KB), and g is set to 1/16. The size of each packet is about 1500 bytes, and the link bandwidth is set to 1 Gbps. The static buffer of the switch port is set to 200 packets (300 KB). The flows arrive according to a Poisson process and are sent from a random source to a random destination server. For our default scenario, 50% of the flows are between (1, 10 KB], 40% are between (10, 100 KB], and 10% are between (100 KB, 10 MB]. We have chosen TCP BBR, DCTCP, and F-DCTCP as the benchmark algorithms. We will utilize these algorithms in various simulation scenarios to conduct a performance comparison and analysis of the optimized algorithms.

4.2 Throughput and Link Utilization

The optimized algorithms should be able to overcome the decline of the whole network throughput performance. The experiment set up 16 hosts to transmit more than 2 GB of data to the same host. The throughput performance of protocols is compared by monitoring and sampling the data throughput of Ethernet from host15 to its edge switch. Fig. 4 depicts the variation in throughput over the first 200 s of the entire transmission process, in which the throughput of DCTCP and DA-DCTCP is significantly higher than that of TCP BBR. The throughput of DCTCP and F-DCTCP will fluctuate sharply and regularly. This indicates that when congestion occurs, the bandwidth resources are not fully occupied, and network congestion cannot be effectively eliminated. The throughput of DA-DCTCP stabilized at roughly 826 Mbps after a minor fall. Compared with DCTCP and F-DCTCP, its throughput has increased by 9.84% and 6.58%, respectively, which means that even in the event of network congestion, the performance of DA-DCTCP throughput can remain stable and of high quality.

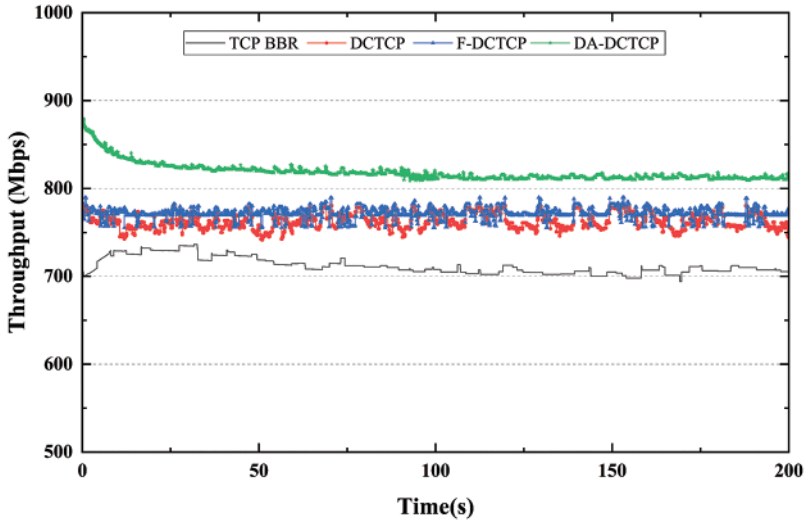


Figure 4: Total throughput in the first 200 s

Then, we conduct experiments to compare the throughput of the four algorithms with various queue and flow numbers. As shown in Fig. 5, the throughput of DA-DCTCP and DCTCP is higher than that of TCP BBR. Fig. 5a shows the throughput under different queue numbers. When the number of queues is less than 4, the throughput of DA-DCTCP is lower than that of F-DCTCP. This is because the latency in DA-DCTCP leads to a decrease in throughput. When there are more than five queues, DA-DCTCP performs much better than DCTCP and F-DCTCP, with a

throughput improvement of roughly 28% and 8.75%, respectively. Fig. 5b shows the throughput of the four algorithms under different flow numbers. In scenarios where less than five flows, DA-DCTCP throughput is observed to be relatively low due to buffer resource reallocation implementation and higher busy queue thresholds. The throughput of each algorithm approaches an equilibrium condition as the flow number rises. When the flow number is more than 20, DA-DCTCP has the maximum throughput; in comparison to DCTCP and F-DCTCP, the average throughput rises by roughly 11.8% and 17.6%, respectively. This is because DA-DCTCP accurately reduces the CWND of the background flows, which mitigates the increase in queue length and effectively boosts the throughput of flows. This implies that under high flow numbers, DA-DCTCP performs better in terms of throughput and improves network performance.

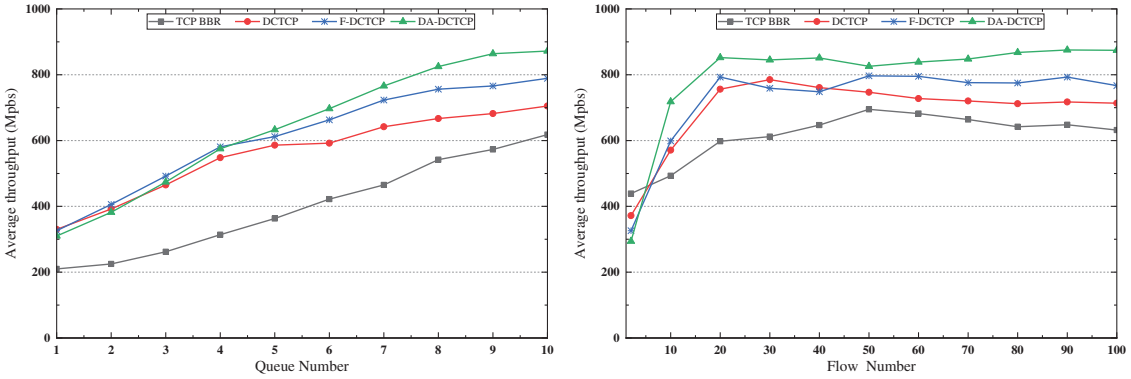


Figure 5: The average throughput comparison in different scenarios

4.3 Flow Completion Time

The FCT is a crucial indication that has a direct impact on the performance and quality of applications. The experimental design client sends 100 flows of different sizes to the server concurrently, including small flows (<100 KB), medium flows (100 KB∼10 MB), and large flows (>10 MB). The average FCT of TCP BBR, DCTCP, F-DCTCP, and DA-DCTCP under different concurrent traffic sizes is measured.

The experimental results are displayed in Fig. 6, and it can be observed that DA-DCTCP performs the best of these algorithms. For small flow sizes, no TCP timeout happens, and DCTCP, F-DCTCP, and DA-DCTCP have little difference in FCT performance, while TCP BBR suffers some packet retransmission. The reason for this behavior can be attributed to BBR's aggressive bandwidth probing mode, which causes queuing queues close to network bottlenecks by overprovisioning the channel with packets. Each of the four methods experiences a high number of TCP timeouts as the flow size grows, which raises the FCT. Notably, DA-DCTCP's FCT is generally lower than the FCTs of the other three algorithms. The CWND of DA-DCTCP can be adaptively adjusted based on the RTT feedback of different flows. This makes it possible to reduce the amount of buffer that long flows occupy and to avoid any negative consequences from CWND constraints on the transmission rate.

Further, test the FCT of network traffic of different scales under different network loads. Fig. 7a shows that compared to TCP BBR, the overall FCT of DCTCP, F-DCTCP, and DA-DCTCP decreased by about 30.7%, 34%, and 37.2%. DA-DCTCP performs best among these algorithms, especially when flow sizes are 1–100 KB. In Fig. 7c,d, when the network load is low (less than 30%),

the normalized FCT of DA-DCTCP is lower than that of DCTCP and F-DCTCP. This is caused by the delay introduced by communication between the central controller and the host/switch. Compared with DCTCP, the overall FCT of DA-DCTCP has increased by about 9.2%. Specifically, FCT for low traffic increased by about 8%, medium traffic increased by about 5.4%, and high traffic increased by about 4.4%. When the load exceeds 30%, some queues become busy, and the DA-DCTCP mechanism reallocates the idle buffer resources of idle queues to busy queues, thus achieving better performance under medium to high loads.

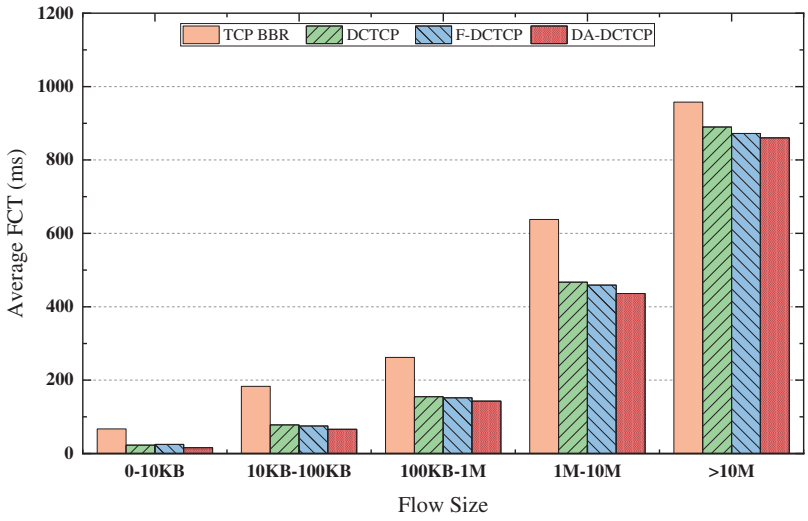


Figure 6: The FCT with different flow sizes

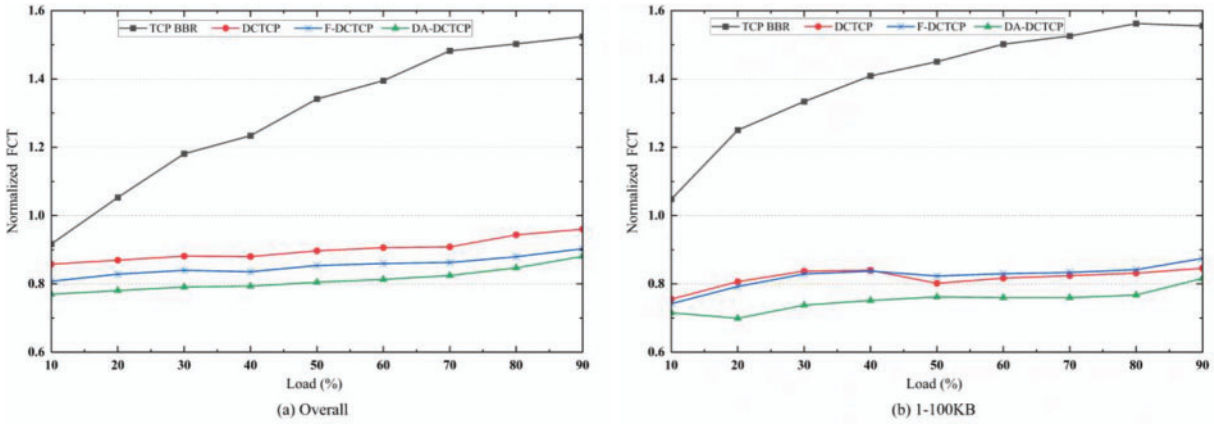


Figure 7: (Continued)

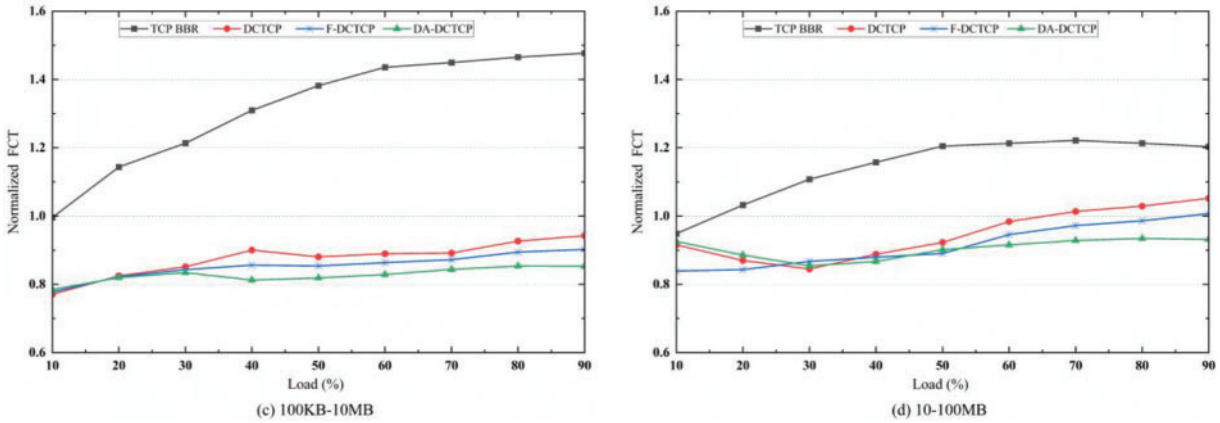


Figure 7: The FCT under different load scenarios

4.4 Latency

The RTT can be calculated at the sender end by measuring the time elapsed between the send packet and the arrival of the acknowledgment (ACK). As illustrated in Fig. 8, RTT tends to increase with increasing concurrent traffic and network congestion. The average RTT of DA-DCTCP is approximately 25 ms, which includes the basic transmission delay and the controller processing time. When the flow number reaches 100, the RTT of DA-DCTCP decreases by 46.4%, 35.2%, and 12.9%, compared to TCP BBR, DCTCP, and F-DCTCP, respectively. The DA-DCTCP can effectively improve the forwarding rate by reducing the threshold (i.e., reducing the queue length) to reduce waiting time. Despite the requirement for iterative calculations in dynamically adjusting the threshold, the dynamic adjustment algorithm and marking strategy impose a negligible computational burden, taking only a few CPU cycles to complete the entire calculation. The additional time required for dynamic threshold calculation and packet processing is on the order of nanoseconds.

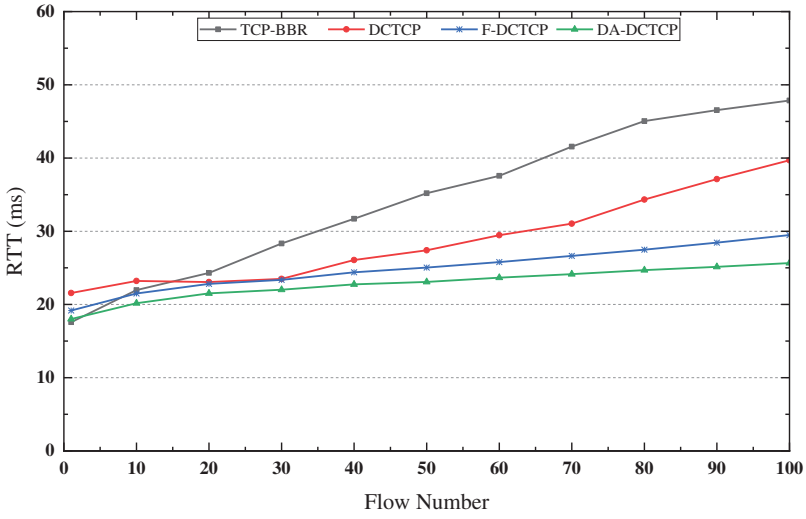


Figure 8: RTTs with different flow numbers

4.5 Loss Rate

In this experiment, burst flows are introduced into the steady background flows to produce network congestion, which is used to assess the algorithm's anti-packet loss performance. Using iperf, background flows were created in the network starting in the 10∼60 s, with each host transmitting 10 flows to a designated destination host.

Fig. 9 illustrates that in the scenario when there are less than two burst flows, the loss rates of the three protocols are all 0, indicating no congestion under this condition. When there are four burst flows, the loss rate of DA-DCTCP is about 1.2% and 0.7% higher than that of DCTCP and F-DCTCP, respectively. These results indicate that when the number of burst flows is minimal, the impact of DA-DCTCP on flow dispersion-based network bandwidth balancing is not substantial. The difference between the loss rates of TCP BBR and DA-DCTCP narrows as the number of burst flows rises, while DA-DCTCP's superiority becomes more apparent. In the case of 50 burst flows, DA-DCTCP was observed to reduce packet loss by 19.4%, 16.7%, and 5.1% when compared to TCP BBR, F-DCTCP, and DCTCP, respectively. This is a result of DA-DCTCP's adaptive queue marking threshold, which allows for a more optimal resource allocation balance. In conclusion, DA-DCTCP can lower the loss rate, demonstrating stability in the face of burst flow conditions and improving network performance.

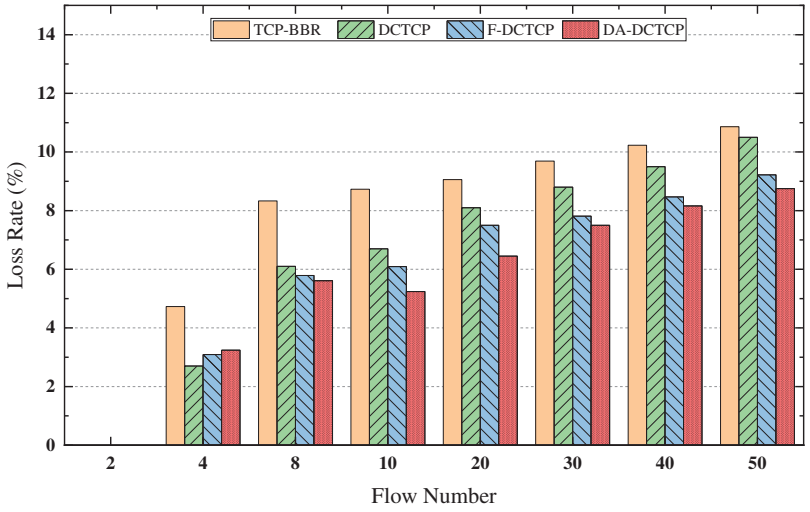


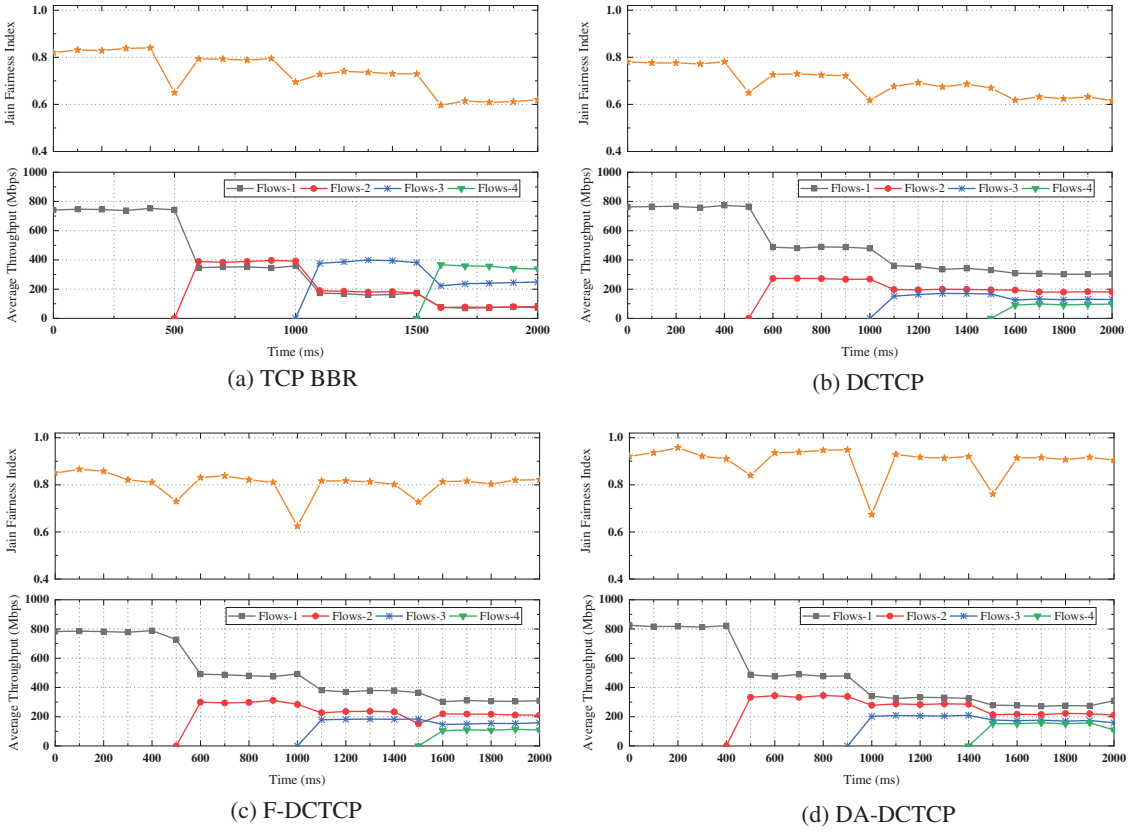
Figure 9: The comparison of packet loss rate

4.6 Fairness

The algorithms' fairness is assessed in various parallel flows while taking flow size into account. With 10 initial concurrent flows of 1.5 KB each, the number of parallel flows was set between 10 and 40. Then burst flows were introduced at 500, 1000, and 1500 ms, respectively, with RTTs of 1.5, 10, and 30 KB. This paper introduces the Jain fairness index [32] as a way to measure the fairness of various flows under various algorithms. The Jain fairness index is a means of measuring fairness in the competition for resources. It is calculated as follows:

$$J = \left(\frac{\left(\sum_{i=1}^{n} x_i \right)^2}{n \sum_{i=1}^{n} x_i^2} \right) \tag{15}$$

where n is the number of links and x_i is the throughput of the i-th link. The Jain fairness index can well reflect the difference in throughput and takes a value in the range of [0, 1]. The closer this fairness index is to 1, the better the fairness of bandwidth allocation is.



(a) TCP BBR (b) DCTCP

(c) F-DCTCP (d) DA-DCTCP

Figure 10: The Jain fairness of different congestion control algorithms

As illustrated in Fig. 10, the BBR algorithm (Fig. 10a) demonstrates a throughput of approximately 764 Mbps before congestion, and its fairness index of 0.83 is somewhat higher than that of DCTCP (Fig. 10b). When additional parallel traffic is added, DCTCP shows a significant bandwidth imbalance and a corresponding drop in the fairness index. Once the flow number reaches 30, it is evident that the initial flows of DCTCP, F-DCTCP, and DA-DCTCP continue to exhibit superior throughput, while the large flows of BBR show a bandwidth advantage. This results in a queue forming at the bottleneck as a result of BBR flows entering the link and rapidly raising its sending rate in an attempt to find more bandwidth. The transmission of larger flows will occupy more buffers, which will make it easier to allocate a greater bandwidth, according to the rules of queuing theory. As seen in Fig. 10c, DA-DCTCP achieves a steady throughput between flows with different sizes after congestion starts. When 40 flows are reached, DCTCP's performance drastically drops and its fairness index stays at roughly 0.61, which is comparable to TCP BBR. While F-DCTCP and DA-DCTCP can maintain good fairness with a fairness index of 0.82 and 0.9, respectively. It can be concluded that DA-DCTCP is well adapted to different traffic requirements, including those involving more flow numbers and different flow sizes. Consistently, DA-DCTCP uses more bandwidth than DCTCP, as demonstrated by preceding experiments.

5 Conclusion and Future Work

To address the challenge of ineffective congestion perception in SDN-based data centers, a DA-DCTCP congestion control strategy is proposed. This strategy optimizes the standard ECN marking mechanism by taking into account the transmission rate and queue growth slope. Additionally, it incorporates RTT information to accurately assess network congestion status, which reduces the potential of incorrect marks. Moreover, DA-DCTCP introduces an adaptive optimization model for the queue length threshold based on the current queue length of the switch. This approach alleviates the burden on queue buffers and enables fine-grained adjustment of CWND. Experimental results demonstrate that DA-DCTCP enhances network throughput, reduces average FCT, and outperforms alternative approaches in terms of latency and packet loss rate relative to TCP, DCTCP, and F-DCTCP. For future work, we intend to adaptively tune the queue thresholds using reinforcement learning for enhanced performance.

Acknowledgement: Not applicable.

Funding Statement: This present research work was supported by the National Key R&D Program of China (No. 2021YFB2700800) and the GHfund B (No. 202302024490).

Author Contributions: The authors confirm their contribution to the paper as follows: proposed the idea, conducted the experiments, and wrote the paper: Jinlin Xu, Wansu Pan; analysis and interpretation of results: Longle Cheng, Wansu Pan; project administration: Haibo Tan, Xiaofeng Li. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available from the corresponding author, upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] Y. P. Chen, R. Griffith, J. D. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *Proc. ACM Workshop Res. Enterp. Netw.*, Barcelona, Spain, Aug. 21, 2009. doi: 10.1145/1592681.1592693.

[2] M. M. Hamdi, H. F. Mahdi, M. S. Abood, R. Q. Mohammed, A. D. Abbas and A. H. Mohammed, "A review on queue management algorithms in large networks," *IOP Conf. Series: Mat. Sci. Eng.*, vol. 1076, no. 1, 2021, Art. no. 012034, vol. 1076. doi: 10.1088/1757-899X/1076/1/012034.

[3] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 63–74, 2010. doi: 10.1145/1851182.1851192.

[4] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-Aware Datacenter TCP ((DTCP)-T-2)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 115–126, 2000. doi: 10.1145/2377677.2377709.

[5] E. Gilliard, K. Sharif, A. Raza, and M. M. Karim, "Explicit congestion notification-based congestion control algorithm for high-performing data centers," in *Proc. AFRICON*, Nairobi, Kenya, Sep. 20–22, 2023. doi: 10.1109/AFRICON55910.2023.10293272.

[6] S. Huang, D. Dong, and W. Bai, "Congestion control in high-speed lossless data center networks: A survey," *Future Gener. Comput. Syst.*, vol. 289, no. 17, pp. 360–374, 2018. doi: 10.1016/j.future.2018.06.036.

[7] H. Ghalwash and C. Huang, "A congestion control mechanism for SDN-based fat-tree networks," in *Proc. HPEC*, Waltham, MA, USA, Sep. 22–24, 2020. doi: 10.1109/HPEC43674.2020.9286156.
[8] G. Zeng *et al.*, "Congestion control for cross-datacenter networks," *IEEE/ACM Trans. Netw.*, vol. 30, no. 5, pp. 2074–2089, 2022. doi: 10.1109/TNET.2022.3161580.
[9] M. Alizadeh *et al.*, "pFabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 435–446, 2013. doi: 10.1145/2534169.2486031.
[10] T. Zhang *et al.*, "Rethinking fast and friendly transport in data center networks," *IEEE/ACM Trans. Netw.*, vol. 28, no. 5, pp. 2364–2377, 2020. doi: 10.1109/TNET.2020.3012556.
[11] H. Lim, W. Bai, Y. Zhu, Y. Jung, and D. Han, "Towards timeout-less transport in commodity datacenter networks," in *Proc. Eur. Conf. Comput. Syst.*, UK, Apr. 26–28, 2021. doi: 10.1145/3447786.3456227.
[12] D. Ongaro, S. M. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum, "Fast crash recovery in RAMCloud," in *Proc. ACM Symp. Oper. Syst. Princ.*, Cascais, Portugal, Oct. 23–26, 2011. doi: 10.1145/2043556.2043560.
[13] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun. Surv. Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014. doi: 10.1109/SURV.2014.012214.00180.
[14] M. Ghobadi, S. H. Yeganeh, and Y. Ganjali, "Rethinking end-to-end congestion control in software-defined networks," in *Proc. HotNets*, Redmond, WA, USA, Oct. 29–30, 2012. doi: 10.1145/2390231.2390242.
[15] S. Jouet, C. Perkins, and D. Pezaros, "OTCP: SDN-managed congestion control for data center networks," in *Proc. NOMS*, Istanbul, Turkey, Apr. 25–29, 2016. doi: 10.1109/NOMS.2016.7502810.
[16] H. Pirzada, M. Mahboob, and I. Qazi, "eSDN: Rethinking datacenter transports using end-host SDN controllers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 605–606, 2015. doi: 10.1145/2829988.2790022.
[17] J. Hwang, J. Yoo, S. Lee, and H. Jin, "Scalable congestion control protocol based on SDN in data center networks," in *Proc. GLOBECOM*, San Diego, CA, USA, Dec. 06–10, 2015. doi: 10.1109/GLOCOM.2015.7417067.
[18] A. Abdelmoniem, B. Bensaou, and A. Abu, "SICC: SDN-based incast congestion control for data centers," in *2017 IEEE Int. Conf. Commun. (ICC)*, Paris, France, May 21–25, 2017. doi: 10.1109/ICC.2017.7996826.
[19] T. Yu and X. Qiu, "STCC: A SDN-oriented TCP congestion control mechanism for datacenter network," *IET Netw.*, vol. 10, no. 1, pp. 13–23, 2021. doi: 10.1049/ntw2.12005.
[20] A. Mushtaq, R. Mittal, J. McCauley, M. Alizadeh, S. Ratnasamy and S. Shenker, "Datacenter congestion control: Identifying what is essential and making it practical," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 49, pp. 32–38, 2019. doi: 10.1145/3371927.3371932.
[21] S. Zou, J. Huang, J. Wang, and T. He, "Flow-aware adaptive pacing to mitigate TCP incast in data center networks," *IEEE/ACM Trans. Netw.*, vol. 29, no. 1, pp. 134–147, 2020. doi: 10.1109/TNET.2020.3027749.
[22] S. S. Chen, W. Wang, C. Canel, S. Seshan, A. C. Snoeren and P. Steenkiste, "Time-division TCP for reconfigurable data center networks," in *Proc. ACM SIGCOMM, 2022 Conf.*, Amsterdam, Netherlands, Aug. 22–26, 2022. doi: 10.1145/3544216.3544254.
[23] W. Bai, L. Chen, K. Chen, and H. T. Wu, "Enabling ECN in multi-service multi-queue data centers," in *Proc. 13th Usenix Conf. Netw. Syst. Design Implement.*, Santa Clara, CA, USA, Mar. 16–18, 2016. doi: 10.5555/2930611.2930646.
[24] A. Majidi, N. Jahanbakhsh, X. Gao, J. Zheng, and G. Chen, "ECN+: A marking-aware optimization for ECN threshold via per-port in data center networks," *J. Netw. Comput. Appl.*, vol. 152, 2020, Art. no. 102504. doi: 10.1016/j.jnca.2019.102504.
[25] J. Bao, J. Wang, Q. Qi, and J. Liao, "ECTCP: An explicit centralized congestion avoidance for TCP in SDN-based data center," in *2018 IEEE Symp. Comput. Commun. (ISCC)*, Natal, Brazil, Jun. 25–28, 2018. doi: 10.1109/ISCC.2018.8538608.
[26] J. Aina, L. Mhamdi, and H. Hamdi, "F-DCTCP: Fair congestion control for SDN-based data center networks," in *2019 Int. Symp. Netw. Comput. Commun. (ISNCC)*, Istanbul, Turkey, Jun. 18–20, 2019. doi: 10.1109/ISNCC.2019.8909171.

[27] G. Diel, C. C. Miers, M. A. Pillon, and G. P. Koslovski, "Data classification and reinforcement learning to avoid congestion on SDN-based data centers," in *Proc. GLOBECOM*, Rio de Janeiro, Brazil, Dec. 04–08, 2022. doi: 10.1109/GLOBECOM48099.2022.10000708.

[28] R. Mittal *et al.*, "TIMELY: RTT-based congestion control for the datacenter," in *Proc. ACM Conf. Special Interest Group Data Commun.*, London, UK, Aug. 17–21, 2015. doi: 10.1145/2829988.2787510.

[29] G. Kumar *et al.*, "Delay is simple and effective for congestion control in the datacenter," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, USA, Aug. 10–14, 2020. doi: 10.1145/3387514.3406591.

[30] W. S. Pan, H. B. Tan, X. F. Li, J. L. Xu, and X. R. Li, "Improvement of BBRv2 congestion control algorithm based on flow-aware ECN," *Secur. Commun. Netw.*, vol. 2022, pp. 1–17, 2022. doi: 10.1155/2022/1218245.

[31] The Linux Kernel, "DCTCP (DataCenter TCP)," Accessed: Mar. 15, 2023. [Online]. Available: https://www.kernel.org/doc/html/latest/networking/dctcp.html

[32] R. Ware, M. K. Mukerjee, S. Seshan, and J. Sherry, "Beyond Jain's fairness index: Setting the bar for the deployment of congestion control algorithms," in *Proc. ACM Workshop Hot Topics Netw.*, Princeton, NJ, USA, Nov. 13–15, 2019. doi: 10.1145/3365609.3365855.