2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)

FastTune: Timely and Precise Congestion Control in Data Center Network

Renjie Zhou, Dezun Dong*, Shan Huang, Yang Bai
National University of Defense Technology, Changsha, China
{renjiezhou, dong, huangshan12, baiyang14}@nudt.edu.cn

***Abstract*—Modern data center networks (DCNs) exhibit high dynamics in both time and space dimensions, which poses challenges for congestion control protocols to achieve low latency, fast convergence, and high throughput. Existing methods have leveraged fine-grained link load information to achieve precise congestion control, but it still suffers from untimely control in highly dynamic DCNs. In this paper, we propose a timely and precise congestion control method called FastTune. FastTune employs fine-grained network status to achieve accurate feedback, uses switch feedback to control the first RTT, and leverages ACK-padding to shorten the feedback path and regulate congestion in time. FastTune develops a multiplicative increase/decrease (MI/MD) algorithm to achieve fast convergence based on timely and precise feedback. Large-scale evaluations show that, compared with state-of-the-art work, FastTune significantly reduces the feedback delay by up to 87%, reduces the average flow completion time by 40%, and the 99th percentile flow completion time by 51%. Besides, FastTune maintains near-zero queueing and reasonable throughput.**

***Index Terms*—Data Center, Congestion Control, Feedback Accuracy, Feedback Delay**

I. Introduction

A large body of applications in modern data centers, such as machine learning and high-performance storage, require extremely low latency and high throughput [1]–[3]. With the development of data centers, the scale and link rate of data center networks (DCNs) are constantly increasing [4], leading to complex and rapidly-changing network status. Recent data center traffic research shows that traffic frequently fluctuates in two dimensions of space and time [5]–[8], which places higher demands on congestion control to promise optimal user experiences [9].

Mainstream congestion control protocols use coarse-grained feedback signals, e.g., explicit congestion notification (ECN) [1], [10] or packet delay [11], [12], which are inefficient for regulating congestion. As these signals cannot accurately reflect the network status, the protocols are highly dependent on the control algorithm to manage congestion. Moreover, by using inaccurate information, these protocols require multiple iterations to converge the flow transmission to the optimal level, which is insufficient for highly dynamic traffic.

In recent years, the in-band network telemetry (INT) [13] is drawing much attention and gradually supported by modern switches. INT can obtain fine-grained network status, which can be used for performing precise congestion control, such as the state-of-the-art method HPCC [14]. However, HPCC still suffers from untimely control in highly dynamic DCNs. In HPCC, fine-grained feedback requires at least one round-trip time (RTT) to take effect, causing slow response to the network congestion and free link capacity.

We have studied many works to observe that the performance of congestion control is limited by the granularity and timeliness of the network feedback. In the spatial dimension, congestion control protocol requires fine-grained network feedback that describes the complex network status to make a more precise reaction. In the time dimension, rapidly changing network status requires more timely control. Therefore, congestion feedback needs to be both timely and fine-grained.

HPCC already uses the INT technique to obtain fine-grained network information to make precise response, but how to achieve timely feedback is still challenging. In HPCC, Fine-grained load information needs to go through the long feedback path of switch-receiver-switch-sender. The non-negligible feedback delay will affect the effectiveness of feedback and leads to two significant problems in highly dynamic DCNs. First, The sender cannot react to the fine-grained feedback in the first RTT of flow transmission, which causes uncontrollable congestion and affects network performance in high-speed network. Second, the feedback path is too long for the sender to precisely regulate congestion in the non-first RTT of flow transmission, causing slow response to the network congestion and free bandwidth.

Based on the discussed problems, we propose a method called FastTune, which can achieve timely and precise congestion control for DCNs. In the spatial dimension, FastTune uses fine-grained network information to accurately feed the network status back, while in the time dimension, it also realizes the timely delivery of fine-grained information. Since the switch has a closer topological location to the sender than the receiver, the switch can inform the sender of the link status earlier. Fortunately, the latest commercial switches [15] already have the function of actively sending back control messages. FastTune uses active feedback from the switch to control the first RTT of the flow. Besides, FastTune uses ACK-Padding to shorten the feedback path for the non-first RTT of the flow, making control more timely. FastTune uses a multiplicative method to adjust the sending window to efficiently react to timely and precise feedback. Thanks to the above mechanisms, FastTune can quickly and accurately

*corresponding author.

978-0-7381-2646-3/21/$31.00 ©2021 IEEE
DOI 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00043

adjust the data transmitting rate to control congestion earlier and occupy the idle link capacity faster.

In summary, the main contributions of this paper are as follows:

- This paper proposes a timely and precise congestion control method called FastTune, which uses fine-grained feedback to achieve earlier congestion control.
- We propose a method of active feedback from the switch for FastTune to make up for the lack of control in the first RTT of flow transmission.
- We develop ACK-Padding for FastTune to reduce feedback delay in the non-first RTT of flow transmission.
- We propose a calculation method for feedback utility to prove the efficiency of FastTune.

We conduct plenty of experiments to verify the superior performance of FastTune from multiple aspects. Compared with HPCC, FastTune can reduce the feedback delay by 87%, the average FCT by 40%, and the 99th FCT by 51% in the Cache Follower [7] traffic pattern of 0.8 link load. Besides, FastTune can maintain near-zero queueing and reasonable throughput.

II. State-of-the-art and Motivation

The performance of congestion control is limited by the granularity and timeliness of the feedback information. In the spatial dimension, fine-grained feedback can provide precise network description, which helps the sender make more precise adjustments to the flow. In the time dimension, timeliness of feedback is also crucial to congestion control because the network status in the data center changes rapidly and requires more timely control. HPCC leverage fine-grained feedback to perform accurate congestion control. However, how to achieve timely feedback is still challenging.

A. Fine-grained feedback provides a more accurate network description

Traditional protocols that use coarse-grained feedback signals cannot precisely control network congestion in highly dynamic DCNs. DCTCP [10] and DCQCN [1] use ECN to feedback network status. When congestion occurs, the switch puts a one-bit ECN mark on the passing data packet, and then the receiver sends an ACK or CNP packet to notify the sender to reduce the sending rate. DX [12] and TIMELY [11] use the delay as the congestion signal. They measure the delay of the data packet, and then the sender adjusts the sending rate based on the packet delay. These methods can handle network congestion well, but they still suffer from slow convergence, inevitable packet queuing, and complex parameter settings, especially when the network status changes rapidly.

To deal with the problems above, HPCC uses the fine-grained link load information from INT (including timestamp, queue length, transmitted bytes, and the link bandwidth capacity) to achieve precise congestion control. As shown in Figure 1, in HPCC, the switches pad the fine-grained load information of the local link to each passing data packet. Then, the receiver generates a corresponding ACK and copy the load information

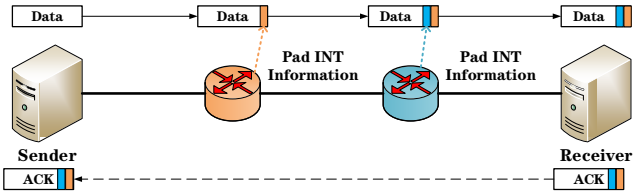
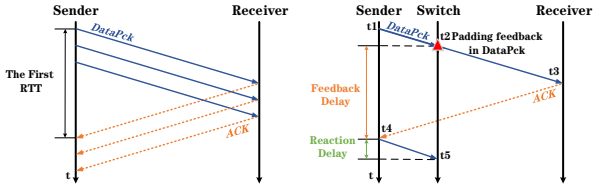


Fig. 1. The overview of HPCC framework.



(a) Lack of feedback in the first RTT.

(b) Feedback Delay and Reaction Delay.

Fig. 2. The lack of timeliness of HPCC is manifested in two aspects: (a) shows that HPCC cannot receive feedback within the first RTT; (b) shows that HPCC feedback requires a long time to work.

to it after receiving the data packet. Next, the ACK is sent back to the sender. Finally, the sender adjusts the sending window according to the feedback information in the received ACK.

B. Timely feedback is necessary and challenging

Highly dynamic DCNs require timely control to avoid congestion and improve link utilization. Due to untimely network feedback, the current mainstream methods cannot control the congestion in time. There are two main difficulties to achieve timely feedback: First, the sender cannot react to the fine-grained network information in the first RTT of flow transmission. Second, the feedback path is too long for the sender to precisely regulate congestion in the non-first RTT of flow transmission.

*1) **Lack of feedback in the first RTT**:* HPCC cannot receive network feedback in the first RTT, which leads to uncontrollable congestion. As shown in Figure 2 (a), When a new flow starts, its ACK requires at least one RTT to reach the sender, which means that the flow is out of control in the first RTT. As the link speed of DCNs has grown steadily from 1 Gbps to 10 Gbps and now 100 Gbps, the data flow will be completed in a much shorter time, so the first RTT becomes more critical in the flow life circle. To complete the data transmission faster, HPCC uncontrollably sends data at the line rate in the first RTT. As the link rate grows, the number of flows that can be completed within the first RTT also increases. We analyzed the proportion of these flows in three realistic traffic patterns. As Table I shows, the amount of out-of-control data in the first RTT increases significantly as the link rate increases, which may cause network congestion and affect the fairness between different flows. When the congestion occurs, the flows in the first RTT will not reduce the sending rate because they have not received feedback information, which is unfair to other flows that have sent more than one RTT.

TABLE I
PROPORTION OF FLOWS THAT HAS BEEN SENT BY ONE RTT IN REALISTIC TRAFFIC PATTERNS

	Cache Follower [7]	Web Search [10]	Web Server [7]
1 Gbps/10 μs	0.4	0.02	0.33
100 Gbps/10 μs	0.54	0.56	0.81

Given that, we develop a switch feedback mechanism to notify the sender of network congestion in advance.

*2) **Long feedback delay in the non-first RTT**:* In high-speed data centers, the network status changes rapidly, so the timeliness of feedback information is vitally important. We divide the time required for the feedback to take effect into two parts: feedback delay and reaction delay. The feedback delay is the time for the feedback information to be transmitted in the path. The reaction delay is the time for the sender to respond. Reaction delay is unavoidable, but the feedback delay is related to the length of the feedback path. Therefore, the key to achieving timely feedback is to shorten feedback delay rather than reaction delay.

Figure 2 (b) shows the feedback delay of HPCC, which makes it trivial to provide efficient congestion control in highly dynamic DCNs. From $t2$ to $t5$, at least one RTT is required for the feedback to take effect. A sender with a bandwidth of B can send data of $B * RTT$ at most within one RTT. In the incast scenario, assuming n senders send data to one receiver simultaneously, and the bottleneck port will accumulate $(n-1) * B * RTT$ data within one RTT. As the link rate increases, the uncontrolled packets in the first RTT will quickly lead to link suspension or packet loss. Similarly, when the link is under-utilized, HPCC also needs one RTT to re-seize the idle link resources, causing the waste of bandwidth. Motivated by the above problems, we develop ACK-padding to shorten the feedback delay in non-first RTT flow transmission.

C. Summary

As one of the most advanced congestion control methods in modern data centers, HPCC uses fine-grained link load information to achieve high-precision congestion control, showing us the potential of fine-grained feedback information. However, we find that HPCC cannot provide timely control in highly dynamic DCNs. First, HPCC needs one RTT to feed the network status back and has no control for the packets sent within this period. Second, HPCC cannot receive timely feedback due to the long feedback path in the non-first RTT flow transmission. Therefore, we proposed a new protocol called FastTune, which can achieve timely and accurate control over the entire life cycle of the data flow.

III. DESIGN

In this section, we present an overview of FastTune, which shows the design overview of FastTune. Then we introduce the design details of FastTune, including the mechanism of switches and hosts.

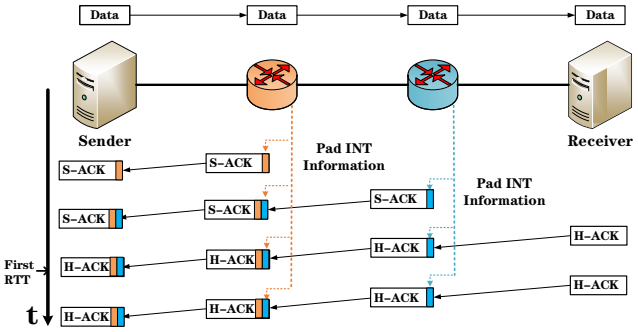


Fig. 3. The overview of FastTune framework.

A. Design overview

FastTune is a congestion control framework driven by the sender. In the spatial dimension, FastTune employs the fine-grained feedback from INT (including timestamp, queue length, transmitted bytes, and the link bandwidth capacity) to precisely adjust the sending window. In the time dimension, FastTune leverages two key techniques to achieve timely congestion control in the flow's entire life, i.e., 1) switch feedback to make up for the lack of control in the first RTT, 2) ACK-Padding shorten the feedback path and reduce the feedback delay.

As Figure 3 shows, the data packet will be acknowledged by the switch (S-ACK) and receiver (H-ACK), and the ACKs and data packets are transmitted through a symmetrical path. FastTune switch uses the fine-grained load information from INT as the feedback. FastTune uses S-ACK to feedback network status to achieve early control in the first RTT of flow transmission. It pads the INT information in the ACK instead of the data packet can reduce the feedback delay. In addition, Senders multiplicatively adjust the sending window according to the timely and fine-grained feedback for fast convergence.

B. Switch design

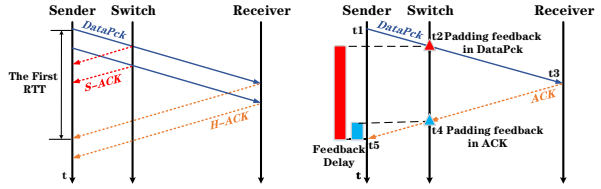
In this section, we describe the design details of switch, including switch feedback and ACK-padding.

*1) **Switch feedback**:* **FastTune uses switch feedback to control the first RTT**. As shown in Figure 4 (a), H-ACK requires at least one RTT to reach the sender. Since the switch is closer to the sender than the receiver, S-ACK can reach the sender earlier. FastTune uses the S-ACK to transmit network status back to the sender within the first RTT to achieve full life cycle control of the flow.

FastTune switch conditionally generates the ACK (S-ACK) for the flow (lines 3 to 5 of Algorithm 1). It records an $NewACK$ boolean value for each active flow, which indicates whether to generate ACK for the flow. The initial value of $NewACK$ is set to $true$. If $NewACK$ is $true$, the switch will generate an ACK with INT information for the received packet. When the switch receives the ACK (including H-ACK and S-ACK) from the link, it sets $NewACK$ to $false$, and no longer generates ACK for this flow (lines 7 of Algorithm 1).

Algorithm 1 Switch algorithm. H-ACK and S-ACK are the acknowledgement packets generated by the host and switch respectively. The initial value of $NewACK$ is true. $Info$ is the information obtained by TNT function, it contains timestamp (ts), queue length ($qLen$), transmitted bytes ($txBytes$), and the link bandwidth capacity (B) of port.

1: **function** HANDLEPCKFROMLINK(pck)
2: **if** pck is $DataPck$ and $NewACK$ is $True$ **then**
3: new $S\text{-}ACK$;
4: $S\text{-}ACK.Infos.insert(port.Info)$
5: SendBack($S\text{-}ACK$)
6: **else if** pck is $H\text{-}ACK$ or pck is $S\text{-}ACK$ **then**
7: $NewACK = False$
8: $pck.Infos.insert(port.Info)$
9: Forward(pck)



(a) Switch feedback and Host feedback. (b) ACK-Padding and Data Packet-Padding.

Fig. 4. Design Rationale. (a) illustrates that switch feedback can make up for the lack of feedback in the first RTT; (b) illustrates that ACK-Padding can significantly reduce the feedback delay compared to Packet-Padding.

In this way, each switch only needs to generate a few ACKs to control the first RTT of the data flow.

*2) **ACK-padding**:* **ACK-Padding can shorten the feedback path**. FastTune uses symmetric routing to ensure that the ACK packet must follow the reverse path of the corresponding data packet. Since the paths of ACK and data packets are symmetrical, they will pass through the same switches, so ACK can perceive the status of the data packet path. As Figure 4 (b) shows, data packet and ACK pass through the switch at $t2$ and $t4$. If the switch pads the feedback into the data packet, the delay of the feedback information is $Delay_{pck} = t5 - t2$. However, if the switch pads the feedback information in the ACK, the feedback delay is only $Delay_{ack} = t5 - t4$. Padding the feedback in the ACK can significantly reduce the delay of the feedback information. It allows the sender to perceive a more real-time network status for more accurate control.

FastTune switch pads the INT information into the received ACK (lines 6 to 8 of Algorithm 1). Since the path of the data packet and the path of the ACK are symmetrical, the input port of the ACK is the output port of the data packet. When the switch receives the ACK (including H-ACK and S-ACK) from the link, it pads the fine-grained status of the data packet output port into the ACK through the INT function. Since S-ACK is generated by the switch, it only passed a partial path, in addition to the S-ACK generated by the last switch, the other S-ACKs can only feed back the status of the partial path.

C. *Host design*

In this section, we describe the design details of the host, including calculation of link utilization and adjustment of sending window.

Algorithm 2 Host algorithm. η is the expected link utilization, the initial values of W and W^c are both set to a bandwidth-delay product (BDP), the initial value of the $SACKUpdate$ is True.

1: **function** HANDLEACK(ack)
2: u = 0
3: **for** each link i on the ACK path **do**
4: $txRate = \frac{ack.Infos[i].txBytes - Last[i].txBytes}{ack.Infos[i].ts - Last[i].ts}$
5: $qLen = min(ack.Infos[i].qLen, Last[i].qLen)$
6: $u' = \frac{qLen + txRate * T}{ack.Infos[i].B * T}$
7: **if** $u' > u$ **then**
8: $u = u'; \tau = ack.Infos[i].ts - Last[i].ts$
9: $\tau = min(\tau, T)$
10: $U = (1 - \frac{\tau}{T}) * U + u$
11: $W = \frac{W^c}{U/\eta} + W_{AI}$
12: **if** ack is $S\text{-}ACK$ and $SACKUpdate$ is $True$ **then**
13: $W^c = W$
14: $SACKUpdate = False$
15: **else**
16: **if** $ack.seq > lastUpdateSeq$ **then**
17: $W^c = W$
18: $lastUpdateSeq = snd\_nxt$
19: $Last = ack.Infos$
20: $R = \frac{W}{T}$

Algorithm 2 illustrates the congestion control process of a single flow at the sender. The sender controls the flow based on the sending window W and stops sending when the window runs out. The sending rate R of the flow is paced according to the sending window. The sender calculates the link utilization U according to the INT information padded in the received ACK, U represents the most congested link in the path, and the sender uses multiplicative increase/decrease (MI/MD) to adjust the sending window.

*1) **Link utilization calculation**:*

The sender calculates the max link utilization U according to the INT information padded in the received ACK. For a link, its utilization is the ratio of the amount of data D that it needs to transmit and the amount of data D_0 it can transmit within an RTT T. The amount of data that the path needs to transmit within one RTT is equal to the sending rate multiplied by the RTT plus the queue length of the port. The amount of data that a path can transmit in an RTT is a BDP. Therefore, the link utilization is calculated according to the following equation.

$$u = \frac{txRate * T + qLen}{B * T} \tag{1}$$

As shown in lines 4 of the algorithm 2, the sender calculates the port's sending rate $txRate$ according to the amount of data $txBytes$ and timestamp ts recorded in the current ACK and the previous ACK, and then calculates the Utilization

of each link according to Equation 1 (line 6). Because the most congested link among all links represents the degree of congestion of the entire path, the algorithm needs to find the highest link utilization in the path (line 7-8). In order to avoid the influence of the instantaneous queue, the algorithm uses the smaller queue length of the last two acks to participate in the calculation , and the algorithm uses the exponential weighted moving average (EWMA) to filter the noise (line 10).

*2) **Adjustment of sending window**:*

The sender controls the flow based on the sending window W and stops sending when the window runs out. So there is the following relationship between the expected new sending window and the current sending window.

$$\frac{W_{new}}{W_{cur}} = \frac{\eta}{U} \tag{2}$$

FastTune sender uses Eqn(3) to quickly update the sending window W. η is the expected link utilization.

$$W = \frac{W}{U/\eta} + W_{AI} \tag{3}$$

W_{AI} is a small increase factor. It prevents the sending window from dropping to zero and then unable to restart. W_{AI} is usually set to $BDP*(1-\eta)/n$, which ensures that concurrent flows will not oversubscribe the link capacity, where BDP and N is the bandwidth-delay product and maximum number of concurrent flows in the link.

Since consecutive ACKs carry overlapping path information, if the sender reacts to all ACKs, it will cause overreaction. FastTune uses a strategy that combines per-RTT and per-ACK to ensure that the flow converges quickly without overreaction. The update of the sending window W is based on the reference window W^c (line 11 of Algorithm 2).

$$W = \frac{W^c}{U/\eta} + W_{AI} \tag{4}$$

The sender updates the sending window every ACK, and the reference window W^c is updated every RTT. The variable $lastUpdateSeq$ records the sequence number of the first data packet sent after the W^c is updated, when the sender receives the ACK corresponding to the $lastUpdateSeq$, then update the W^c to achieve per-RTT update (line 16 to 18 of Algorithm 2). Since only the switch ACK arrives in the first RTT, the switch ACK can update the reference window only once, so we use the variable $SACKupdate$ to ensure this (line 12 to 14 of Algorithm 2).

IV. Analysis and Discussion

This section proposes a calculation method of feedback utility. Then we discussed the hardware support required by FastTune.

A. Utility of feedback

The utility of feedback determines the ability of congestion control, and it is mainly related to the timeliness and the richness of the feedback. For the ACK that carries network status, its utility is determined by the number of switches it passes through and the delay of each switch's feedback. We define the utility of the i-th switch's feedback as follows:

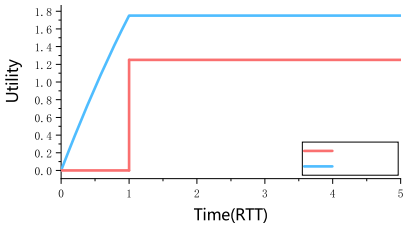


Fig. 5. Comparison of the feedback utility between HPCC and FastTune

$$u_i = \frac{2RTT - delay_i}{T} \tag{5}$$

This is a heuristic definition, $delay_i$ is the switch's feedback delay. It is inversely proportional to utility. Without considering packet queuing, the $delay_i$ will not exceed one RTT. In order to ensure that the utility of one switch's feedback does not exceed the sum of the two switches, we set the constant in Eqn(5) as $2RTT$. So the utility of ACK is as follows, n is the number of switches passed by the ACK.

$$U = \frac{1}{n} * \sum_{i=1}^{n} u_i = 2 - \frac{1}{n*T} * \sum_{i=1}^{n} delay_i \tag{6}$$

There are N switches in the path, and the feedback delays of the i-th switch in HPCC and FastTune are as follows:

$$delay^i_{hpcc} = (1 - \frac{i}{2(N+1)}) * RTT \tag{7}$$

$$delay^i_{FastTune} = \frac{i}{2(N+1)} * RTT \tag{8}$$

$$U_{hpcc} = 2 - \frac{1}{N*T} * \sum_{i=1}^{N} delay^i_{hpcc} = 1.25 \tag{9}$$

$$U_{FastTune} = 2 - \frac{1}{n*T} * \sum_{i=1}^{n} delay^i_{hpcc} = 2 - \frac{n+1}{4(N+1)} \tag{10}$$

For HPCC, each ACK can represent the congestion level of the entire path, so the utility of ACK feedback in HPCC is shown in Eqn (9). For FastTune, the S-ACK may only represent the congestion level a partial path. Therefore, the utility of its ACK is shown in Eqn (10), where n is the number of switches in the partial path. The S-ACK generated by the switch that is farther from the sender can indicate the information of the longer path, but it also takes more time to reach the sender. So n is gradually increasing in the first RTT. As shown in Figure 3, in the first RTT, FastTune can feed partial network status back to make up for the lack of control. After the first RTT, FastTune can also generate more effective feedback to guide the sender to perform precise control.

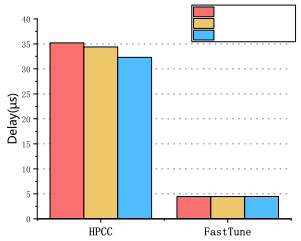


Fig. 6. Cross-rack feedback delay for different realistic traffic paterns in Clos topology.

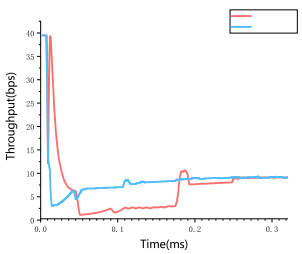


Fig. 7. Average throughput of flow2 - flow4 when congestion occurs in a bottleneck link.

B. Hardware support

The latest switch can already support switch feedback [15]. Commercial switch hardware only needs a few modifications to support FastTune. The INT function can obtain switch information on the data plane. Since the paths of the data packet and the ACK are symmetrical, the output port of the data packet is the input port of the ACK, and the two are buffered in different queues in the switch, which means that the switch needs to collect cross-port information.

Most commercial switches adopt a shared buffer port queue design. The switch hashes the five-tuple of the flow *(source IP, destination IP, source port, destination port, protocol type)* and then allocates the buffer according to the hash value. When receiving an ACK, the switch only needs to exchange the *(source IP, source port)* and *(destination IP, destination port)* first, and then hash the exchanged five-tuple to find the queue of the data packet and collect information.

V. Evaluation

In this section, we will conduct lots of experiments to evaluate the performance of FastTune based on the OMNeT++ simulator [16] with our extended INET framework [17].

A. Evaluation configuration

Schemes. We make a comprehensive comparison between FastTune and HPCC. FastTune only uses MI/MD to adjust the sending window. HPCC uses a combination of MI/MD and additive increase (AI). AI conservatively increases the sending rate, while MI is more aggressive than AI. For the sake of fairness, we have also implemented a version of HPCC without AI for comparison. We call it HPCC+.

Network topologies. Most of the evaluations in this paper use the Clos topology. In this topology, four spine switches are connected to 24 leaf switches, and each leaf switch is connected to 10 hosts. The link rate between the host and the leaf switch is 10 Gbps, and the link rate between the leaf switch and the spine switch is 100 Gbps. The delay of two adjacent nodes is 2 μs. Besides, we used a simple many-to-one topology to verify the convergence of FastTune.

Traffic patterns. We have used the widely accepted and publicly available data center traffic patterns, Cache Follower [7], Web Search [10] and Web Server [7] in our evaluation. We adjusted the traffic generation rate to set the average link load to 40%, 60%, and 80%, respectively. We also created some simple manual traffic patterns to evaluate FastTune's microbenchmarks.

Parameters. HPCC uses the parameters recommended by the paper [14]: $maxStage$ = 5, η = 0.95, $W_{AI} = BDP * (1 - \eta)/N$, where N is the maximum number of concurrent flows in the link. HPCC+ and FastTune have the same parameters as HPCC except $maxStage$, they do not need this parameter.

Performance metrics. There are four metrics for our evaluation: (1) Feedback delay; (2) Flow completion time (FCT); (3) Throughput of hosts; (4) Switch port queue length.

B. Micro-benchmarks

In this subsection, we evaluate and compare FastTune's convergence and feedback delay.

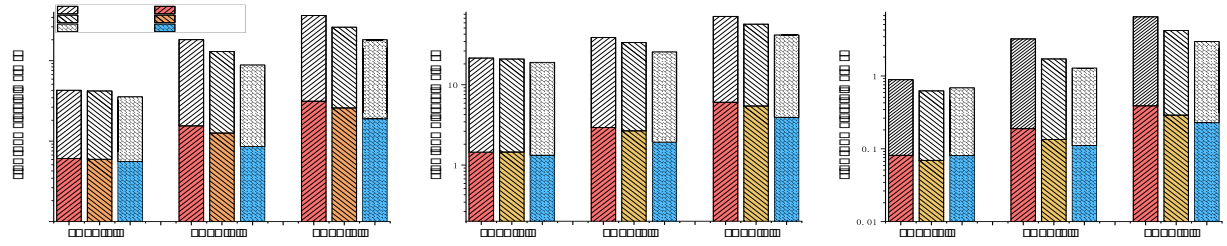
Feedback delay: We ran three realistic traffic patterns in the 240-node Clos topology, and the link load was 0.8. We counted the feedback delay of all transmissions across racks. Each cross-rack path contains three switches. We measure the feedback delay of the three switches in each ACK and then average them to get the ACK's feedback delay.

In the Clos topology, the base RTT is 8 μs. In Figure 6, we can see that the feedback delay of HPCC is several times the basic RTT and varies with the traffic patterns. However, the feedback delay of FastTune is only half of the basic RTT and does not change with traffic patterns. The reason is that FastTune pads the feedback information in the ACK instead of in the data packet and allows the ACK to be transmitted before the data packet. Thanks to timely feedback, the sender can obtain more real-time network status and make more precise feedback.

Fast convergence: We compared the convergence speed of HPCC and FastTune when the link is congested. In the many-to-one topology, flow1 sends data to the receiver first, and then flow2 to flow4 sends data to the receiver simultaneously. Figure 7 shows the average throughput change of flow2-flow4 during congestion. In the beginning, all flows are sent at the line rate, which leads to congestion at the bottleneck link. FastTune started to reduce the sending rate at 6 μs, and the rate is reduced to the lowest point at 14 μs. However, HPCC started to reduce the rate to 13 μs. At 56 μs, FastTune has almost restored a fair sending rate. HPCC is reducing the sending rate to a minimum at 50 μs and reaches the fair rate after 200 μs. FastTune can detect and control congestion earlier because the switch can feedback in the early stage of flow transmission. The timeliness of the feedback also allows FastTune to use a multiplicative increase to restore the rate without causing the link to oversubscribe. These make FastTune have better convergence than HPCC.

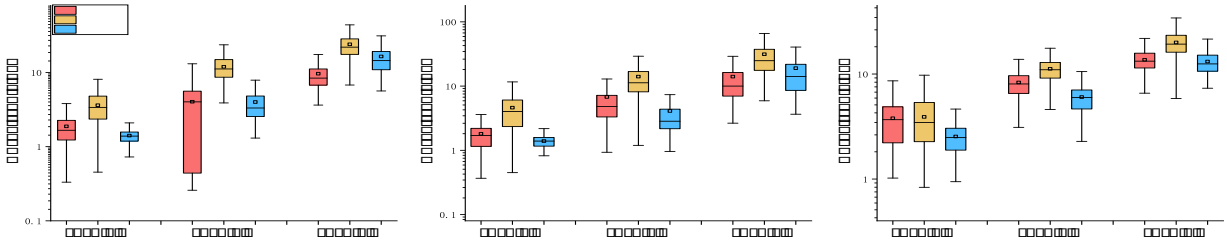
C. Large-scale simulations

In this subsection, we run three realistic traffic patterns in a 240-node large-scale Clos topology to evaluate the performance of FastTune. The delay between adjacent nodes is 2 μs, so the RTT across the rack in the topology is 8 μs, and



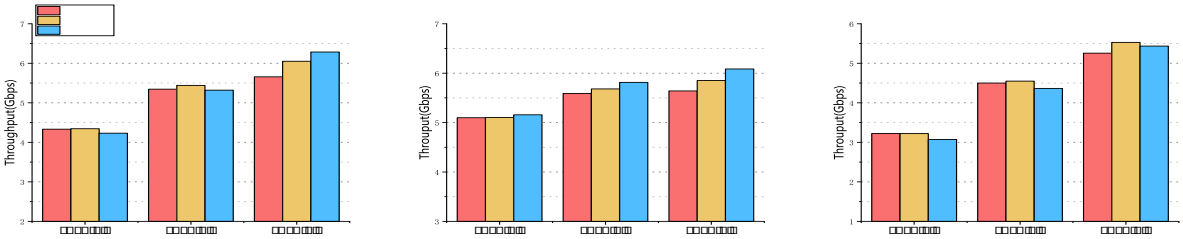
(a) Cache Follower (b) Web Search (c) Web Server

Fig. 8. Three methods of AVG FCT and 99-th FCT in different realistic traffic patterns and workloads.



(a) Cache Follower (b) Web Search (c) Web Server

Fig. 9. Average queue distribution of all downstream ports of leaf switches in Clos topology with different real traffic patterns and workload.



(a) Cache Follower (b) Web Search (c) Web Server

Fig. 10. Average throughput of all host ports in Clos topology with different real traffic patterns and workload.

the RTT within the rack is 4 μs. To pressure test various flow patterns, each pattern has a workload of 0.4, 0.6, 0.8. In the experiment, we compare FastTune with HPCC and HPCC+.

FastTune can significantly reduce the FCT. Figure 8 shows the 99th percentile FCT and average FCT of FastTune and the other two methods under different realistic traffic patterns. In all scenarios, the FCT of FastTune is better than that of HPCC. When the link workload is low, the FCT performance of the three methods is similar. With the increase of workload, the performance difference of FCT becomes more obvious. For example, in the Cache Follower traffic pattern with a workload of 0.8, the AVG-FCT and 99th-FCT of HPCC are 3.13 ms and 33.34 ms, but FastTune has only 1.89 ms and 16.3 ms. FastTune reduces the AVG-FCT by 40% and the 99th-FCT by 51%.

HPCC+ is also better than HPCC. Because HPCC+ only performs multiplicative increase when adjusting the sending window. FastTune's FCT is better than HPCC+ in most scenarios. The reason is HPCC+ does not have timely network feedback, so its response lags behind the network status, which causes HPCC+ to easily lead to oversubscription of links. As Figure 8 (c) shows, in the Web Server traffic pattern, FastTune's FCT is slightly worse than HPCC+ of load 0.4. The reason is that most of the Web Server traffic patterns are short flows, and HPCC and HPCC+ cannot control them well. Due to low workload, these uncontrolled flows are sent out quickly and rarely cause congestion. When the link load increases, these uncontrolled flows will cause congestion. So when the load is 0.6 and 0.8, FastTune is significantly better than HPCC+.

FastTune can achieve near-zero queuing. In FastTune, the sender can obtain more timely and effective network feedback. When the network is congested, the sender can detect the congestion earlier and deal with it to avoid worsening congestion. However, HPCC needs more time to react when congestion occurs, which leads to the accumulation of congested queues. Figure 9 shows the distribution of the average queues of all the downstream ports of the leaf switch in the Clos topology. We can see that FastTune can maintain a lower or similar queue length in different traffic patterns and workload scenarios than

HPCC. However, HPCC+ maintains a very high queue level in all scenarios. The reason is that HPCC+ uses a multiplicative method to increase the sending window. Therefore, due to untimely feedback, it is easy to cause link congestion.

FastTune can maintain a more reasonable throughput. It is unreasonable to blindly increase throughput, which will only lead to link congestion. The Web Server traffic pattern is mostly short flow, and HPCC and HPCC+ are insufficient to control it. As Figure 10 (c) and Figure 8 (c) shows, although the throughput is high, it is easy to cause congestion and hurt the FCT. The Web Search flow pattern has many long flows. Figure 10 (b) shows that FastTune can better avoid congestion and occupy the idle link capacity faster, so it has higher throughput than others. In the Cache follower traffic pattern, the long and short flows are relatively uniform. As Figure 10 (a) Shows, when the link load is low, the throughput of FastTune is slightly lower than the other two schemes. When the link load increases, FastTune can provide higher throughput.

VI. Related works

In recent years, to achieve both low latency and high throughput, various transmission protocols have been proposed for DCNs.

DCTCP [10] uses instantaneous ECN marking to detect the network congestion, and the sender adjusts the sending window according to the ECN mark. Many works have been proposed based on DCTCP. D2TCP [18] adjusts the sending window for the deadline. DCQCN [1] uses both ECN and congestion quantification feedback to control congestion. HPCC [14] relies on INT to achieve precise control of congestion, but it still suffers from untimely control.

DX [12], TIMELY [11] and Swift [3] are delay-based congestion control schemes. DX judges and controls network congestion based on queuing, Timely adjust the sending rate according to the gradient of RTT. Swift uses a simple target end-to-end delay instead of RTT gradient. However, delay-based solutions are easily affected, and all require high RTT measurement accuracy.

pHost [19], NDP [20], Homa [21] and ExpressPass [22] are proactive congestion control schemes, which use credits to allocate the link bandwidth for flows and actively prevent network congestion. However, they usually require unpractical changes to the hardware.

VII. conclusion

This paper proposes a timely and precise congestion control method called FastTune. In the spatial dimension, FastTune leverages fine-grained feedback to respond to congestion precisely; In time dimension, It uses the switch feedback to make up for the lack of control in the first RTT. It also uses ACK-padding to shortens the feedback path and reduces the feedback delay. Compared with the state-of-the-art work, FastTune significantly reduces FCT and achieves faster convergence, higher throughput, and near-zero queueing at the same time. Besides, this paper propose a calculation method for feedback utility to prove the efficiency of FastTune's feedback.

Acknowledgement

We would like to thank the anonymous reviewers for their insightful comments. We gratefully acknowledge members of HiNA group at NUDT for many inspiring conversations. The work was supported by the National Key R&D Program of China under Grant No. 2018YFB0204300, Excellent Youth Foundation of Hunan Province (Dezun Dong) and National Postdoctoral Program for Innovative Talents Grant No. BX20190091.

References

[1] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," in *Proc. ACM SIGCOMM*, 2015.

[2] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, "Network requirements for resource disaggregation," in *Proc. USENIX OSDI*, 2016.

[3] G. Kumar, N. Dukkipati, K. Jang, H. M. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan *et al.*, "Swift: Delay is simple and effective for congestion control in the datacenter," in *Proc. ACM SIGCOMM*, 2020.

[4] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," in *Proc. ACM SIGCOMM*, 2015.

[5] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM SIGCOMM*, 2010.

[6] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc. ACM SIGCOMM*, 2009.

[7] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proc. ACM SIGCOMM*, 2015.

[8] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic sdn controller assignment in data center networks: Stable matching with transfers," in *Proc. ACM INFOCOM*, 2016.

[9] L. Jose, L. Yan, M. Alizadeh, G. Varghese, N. McKeown, and S. Katti, "High speed networks need proactive congestion control," in *Proc. ACM HotNets*, 2015.

[10] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proc. ACM SIGCOMM*, 2010.

[11] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," in *Proc. ACM SIGCOMM*, 2015.

[12] C. Lee, C. Park, K. Jang, S. Moon, and D. Han, "Accurate latency-based congestion feedback for datacenters," in *Proc. USENIX ATC*, 2015.

[13] (2021) In-band network telemetry in barefoot tofino. [Online]. Available: https://www.opencompute.org/files/INT-In-Band-Network-Telemetry-A-Powerful-Analytics-Framework-for-your-Data-Center-OCP-Final3.pdf

[14] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh *et al.*, "Hpcc: high precision congestion control," in *Proc. ACM SIGCOMM*, 2019.

[15] (2021) Huawei 6866 datacenter switch. [Online]. Available: https://e.huawei.com/mediares/MarketingMaterial_MCD/EBG/PUBLIC/zh/2020/12/5e68b1a5-7c03-4724-a640-926462874f30.pdf

[16] (2021) Omnet++. [Online]. Available: https://omnetpp.org

[17] (2021) Inet. [Online]. Available: https://inet.omnetpp.org/

[18] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," in *Proc. ACM SIGCOMM*, 2012.

[19] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, "phost: Distributed near-optimal datacenter transport over commodity network fabric," in *Proc. ACM CoNEXT*, 2015.

[20] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik, "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proc. ACM SIGCOMM*, 2017.

[21] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *Proc. ACM SIGCOMM*, 2018.

[22] I. Cho, K. Jang, and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *Proc. ACM SIGCOMM*, 2018.