2017 IEEE 37th International Conference on Distributed Computing Systems

Flow-Aware Adaptive Pacing to Mitigate TCP Incast in Data Center Networks

Shaojun Zou[§], Jiawei Huang[§], Yutao Zhou[§], Jianxin Wang[§], Tian He[‡]

[§]School of Information Science and Engineering, Central South University, ChangSha, China 410083
[‡]Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, USA 55455
Email: zoushj@csu.edu.cn, jiaweihuang@csu.edu.cn, zhouyutao999@gmail.com, jxwang@csu.edu.cn, tianhe@umn.edu

***Abstract*—In data center networks, many network-intensive applications leverage large fan-in and many-to-one communication to achieve high performance. However, the special traffic patterns, such as micro-burst and high concurrency, easily cause TCP Incast problem and seriously degrade the application performance. To address the TCP Incast problem, we first reveal theoretically and empirically that alleviating packet burstiness is much more effective in reducing the Incast probability than controlling the congestion window. Inspired by the findings and insights from our experimental observations, we further propose a general supporting scheme Adaptive Pacing (AP), which dynamically adjusts burstiness according to the flow concurrency without any change on switch. Another feature of AP is its broad applicability. We integrate AP transparently into different TCP protocols (i.e., DCTCP, L^2DCT and D^2TCP). Through a series of large-scale NS2 simulations, we show that AP significantly reduces the Incast probability across different TCP protocols and the network goodput can be increased consistently by on average 7x under severe congestion.**

***Index Terms*—Data center; Pacing; TCP; Incast; Congestion control**

I. Introduction

As a prosperous industry, data centers have become the critical infrastructure to host a large amount of applications and provide diverse user services, such as recommendation systems, web search and MapReduce [1]. In modern data center networks (DCNs), however, the micro-burst is a common traffic pattern, which has adverse impact on the network performance, especially in highly concurrent applications [2]. The micro-burst generally indicates that bursty traffic is generated by the multiple servers in a small time-scale, potentially making some flows unstable and decreasing the link utilization [3], [4]. The main reasons for micro-burst can be attributed to the following special traffic characteristics of DCNs.

First, the many-to-one and high fan-in communication patterns widely exist in data center network. The high fan-in traffic patterns have been reported by lots of measurements on productive data centers [5]. In the data centers of Google and Microsoft, the web search application generally involves 10-1000 servers [6]. For instance, a single HTTP request in Facebook often incurs 130 internal requests on average [7]. When all servers inject their packets into the network in a micro-burst, these in-flight packets will constitute a large burst. Once the bursty traffic reaches the switch with shallow buffer, it causes rapid fluctuations in queue length, potentially resulting in packet loss and even TCP timeout.

Second, since server request unit (SRU) in DCNs is usually less than 100KB, a TCP flow generally finishes its data transmission in the slow-start phase [8], [9]. During slow-start, the sender transmits at least two back-to-back packets in response to each ACK. Moreover, if the receiver adopts cumulative ACKs to indicate successful receipt of multiple packets, the sender will transmit more packets in a micro-burst [10].

Data center networks have several special features, such as shallow-buffer switches and high-bandwidth links [11]. More especially, the link capacity in today's data centers is quite large, typically on the order of dozens of Gbps (e.g., 10Gbps and 40Gbps). Although the round trip time is only a few hundreds of microseconds [12], the high link capacity results in a large bandwidth-delay product (BDP). For example, the commodity switches generally have about 100KB buffer per port [13], while the value of BDP is 250KB in 10Gbps network with 200us RTT. Therefore, it is reasonable to use the links with large bandwidth to help the switch accommodate more packets.

Admittedly the concept of TCP pacing has been proposed in wide area networks, under which the sender evenly transmits a whole window of packets over a round-trip time [14]. Since pacing can make full use of high-bandwidth links in data centers to accommodate packets, it is natural choice to leverage pacing to eliminate the traffic burst, especially in high fan-in applications. However, this simple approach does not work well in all data center environments, because it failed to consider the rich diversity of applications. The network applications involving small number of concurrent flows are still common in data centers. For instance, a few hundreds KB file is cut into several data blocks, which are stored on a small amount of servers in the distributed file storage application. When the switch has enough buffer space to accommodate the burst, pacing will waste bandwidth resource and degrade the network throughput if the senders still transmit an entire window of packets over a round-trip time.

In this paper, we propose an adaptive scheme named Adaptive Pacing (AP) to mitigate TCP Incast problem. AP dynamically adjusts the time interval according to the flow concurrency. Therefore, AP not only effectively eliminates the

1063-6927/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDCS.2017.170

micro-burst, but also prevents the throughput loss. While the adjustment strategy of congestion window has been extensively studied in previous congestion control research, our design joints the control of congestion window and time interval, solving the TCP Incast problem.

The rest of this paper is organized as follows. In Section II, we discuss our design motivation in detail. In Section III, the design of AP is presented. In section IV, we evaluate the performance of AP through NS2 simulations. In section V, we summarizes the related works. Finally, we conclude the paper in section VI.

II. Motivation

In this section, we first theoretically analyze the benefit and limitation of pacing under different scenarios. Then, we quantitatively show the benefit and drawback of pacing through experiments. Finally, we summarize our observations, which indicates the need for pacing adaptation.

A. The benefit of pacing under highly concurrent flows

In this part, we first theoretically analyze the influence of micro-burst. For ease of description, let n, w and B denote the number of synchronized flows, the congestion window and the size of switch buffer, respectively. Since the link capacity in data center network is typically not less than 1Gbps, the transmission delay can be ignored. For example, it takes 1.2us to transmit a 1500B packet in 10Gbps networks. Hence we assume that the whole window of packets simultaneously arrives at the switch. Since there are n synchronized flows with congestion window size of w, the total number of in-flight packets is $n \times w$. When the number of in-flight packets exceeds the switch buffer, the packet loss rate is $1-B/(n \times w)$.

Some researchers have proved that a full window of packet loss is a principal factor that causes timeout and TCP Incast problem [15]. Thus we can calculate the probability of full window loss as the probability of timeout:

$$p = (1 - \frac{B}{n \times w})^w. \tag{1}$$

Furthermore, it is easy to understand that no matter which flow experiences timeout, it will cause TCP Incast for n concurrent flows. Therefore, the TCP Incast probability P can be calculated as

$$P = 1 - (1 - (1 - \frac{B}{n \times w})^w)^n. \tag{2}$$

For pacing, if the time interval between two adjacent packets is set to t and the link capacity is C (packets/sec), the switch will forward $C \times t$ packets before the next packet of each flow arrives at the switch. Obviously, if none of flows experience packet loss, there are $n \times w - (w-1) \times C \times t$ packets in queue as the whole window of packets reach the switch. Otherwise, the packet loss rate is $1 - (t \times (w-1) \times C + B)/(n \times w)$.

Under pacing, we obtain the probability of timeout $p^{'}$ and the TCP Incast probability $P^{'}$ as

$$p^{'} = (1 - \frac{t \times (w-1) \times C + B}{n \times w})^w, \tag{3}$$

$$P^{'} = 1 - (1 - (1 - \frac{t \times (w-1) \times C + B}{n \times w})^w)^n. \tag{4}$$

When TCP timeout happens, the sender has to wait for excessive idle period of RTO (i.e., 200ms in most operating systems), greatly enlarging the flow completion time (FCT). Here, for ease of description, we define the window completion time as the time during that the sender receives all ACKs for the whole window of packets. To analyze the impact of TCP Incast probability, we first calculate the window completion time T_w without pacing as

$$T_w = (1 - P) \times RTT + P \times RTO_{min}. \tag{5}$$

Under pacing, it takes extra $(w-1) \times t$ for the sender to transmit an entire window of packets due to time interval between adjacent packets. According to Equation (3) and (4), we calculate the window completion time $T_w^{'}$ with pacing as

$$T_w^{'} = (1 - P^{'}) \times RTT + P^{'} \times RTO_{min} + (w-1) \times t. \tag{6}$$

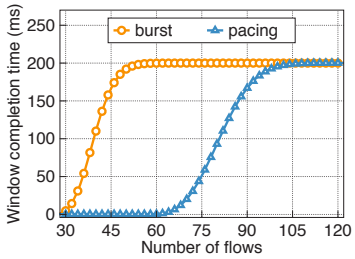


Fig. 1: Window completion time with varying number of flows

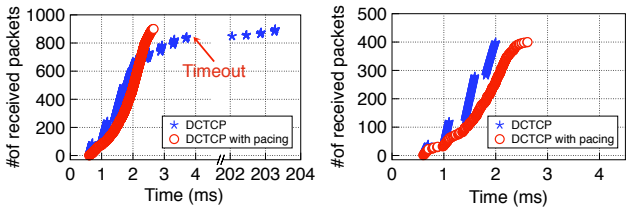
According to Equation (5) and (6), we plot the window completion time with different the number of flows in Fig.1, where B, C and w are 100 packets, 10Gbps and 4, respectively. From Fig.1, we observe that the window completion time increases since the probability of full window loss becomes large with the increasing of the number of flows. Compared with the case without pacing, however, pacing significantly reduces the probability of timeout and thus obtains lower window completion time.

B. The limitation of pacing under lowly concurrent flows

For highly concurrent applications, pacing is a simple yet effective solution to dispel the micro-burst. In order to illustrate intuitively the bursty behavior of flows and verify the effectiveness of pacing, we conduct a simple experiment using NS2 simulator to record the moments when packets are successfully received.

In this simulation, 45 senders and one receiver are connected to the same switch with 10 Gbps links. The switch takes the drop-tail queue management policy and its buffer can accommodate 100 packets. The volume of data sent by each

sender is 30KB and the packet size is 1500B. The round trip of time is set to 400us. The RTO_{min} is set to 200 milliseconds [2], [8]. Considering that DCTCP has been integrated into common operating systems and extensively applied in many data centers, such as Google [16] and Morgan Stanley [17], we utilize DCTCP as the TCP protocol, and the marking threshold of switch buffer is set to 65. The experimental result is illustrated in Fig.2 (a).



(a) $n = 45$ (b) $n = 20$

Fig. 2: The distribution of moments when packets are received.

From Fig.2 (a), we observe that if pacing is disabled, the receiver intermittently receives the clustering of packets from its senders, showing the burstiness of traffic. Unfortunately, some unlucky flows experience timeout due to lack of sufficient duplicate ACKs. As a result, it takes around 200ms for DCTCP to finish the data transmission. Interestingly, compared with DCTCP, pacing only take about 2.6ms to finish data transmission. That is to say, pacing spends much less time finishing data transmission than DCTCP, significantly improving the network efficiency.

However, pacing is not appropriate for all network scenarios. When the number of concurrent flows is not large, pacing wastes bandwidth resource and causes the throughput loss. In order to illustrate this problem, we change the the number of senders to 20. Fig.2 (b) shows that for DCTCP, all flows have finished data transmission at around 2ms. However, although pacing eliminates the micro-burst, it takes about 2.6ms to finish the data transmission. The main reason is that interval time between adjacent packets is too large so that pacing can not make full use of bandwidth resource. As a result, pacing reduces about 30% of throughput compared with DCTCP.

C. The need for pacing adaptation

As stated above, we draw the conclusion that (i) pacing is an effective way to dispel micro-burst and significant reduces TCP Incast probability. In highly concurrent scenarios, pacing avoids severe goodput degradation, and (ii) in low concurrent scenarios, pacing causes a waste of bandwidth sources due to excessive time interval, resulting in throughput loss.

These conclusions naturally motivate us to design an adaptive scheme called Adaptive Pacing (AP). According to the number of concurrent flows, AP dynamically adjusts the time interval to deal with TCP Incast problem and achieve high bandwidth utilization. In the following sections, we will present the design of adaptive scheme.

III. Protocol Design

In this section, we concentrate on the design of our proposed scheme AP. We first build a model of TCP Incast problem. Then we present how to obtain the optimal time interval by the number of concurrent flows.

A. Model analysis

To obtain optimal time interval, we derives the relation between the network goodput G and the time interval t. First, we obtain the Incast probability by calculating the probability of timeout. Then, we take the normalized method to express the relation between G and t.

In data center networks, it is unrealistic that all flows start transmissions at the same time. Therefore, we first set up a more realistic model in which the starting times of all flows are asynchronous. To simplify the presentation, we assume that the starting time of n flows are uniformly distribution in [0, t_0]. We divide the t_0 into time slots, and the length of a time slot equals the transmission delay t_d of a packet. These flows whose starting time lie in the same time slot are synchronous. Then we obtain the average number of concurrent flows n_{avg} in a time slot as

$$n_{avg} = \frac{n \times t_d}{t_0}. \tag{7}$$

Furthermore, we use d ($0 \leq d \leq 1$) to denote the paced ratio, which means that the entire window of packets is transmitted during $d\times$RTT if AP is enabled. Besides, $d\times$RTT can be divided into $d\times\text{RTT}/t_d$ time slots. Therefore, we obtain the average number of packets m_{avg} transmitted by each flow in these time slots as

$$m_{avg} = \frac{w \times t_d}{d \times RTT} = \frac{t_d}{t}. \tag{8}$$

We suppose that the starting time of flows in the ith time slot is t_i. Since the whole window of packets are transmitted across $d\times$RTT, packets from these flows with their starting times in [$t_i - d\times$RTT, t_i] will be transmitted in the ith time slot. That is, there are $d\times\text{RTT}/t_d$ (denoted by s) time slots and flows in these time slots will transmits m_{avg} packet in the ith time slot. Consequently, the total number of packets m transmitted in a time slot can be calculated as

$$m = n_{avg} \times m_{avg} \times s = \frac{n \times t_d \times w}{t_0}. \tag{9}$$

We simply denote $t_d \times w/t_o$ as k, and then $k \times n$ packets are injected into the network in a time slot. For the purposes of discussion and analysis, we suppose that the packets are uniformly distributed among n flows when the number of packets in switch buffer reaches buffer size B. Therefore, the congestion window w of each flow is B/n when the switch buffer becomes full. Without loss of generality, each flow increases itself congestion window with additive increasing and the congestion window becomes $w + 1$ in the next

round of RTT. Similarly to Equation (4), we obtain the Incast probability $P^{'}$ as

$$P^{'} = 1 - (1 - (1 - \frac{t \times w \times C + B}{n \times (w+1)})^{w+1})^{n \times k}. \quad (10)$$

Inspired by literature [18], we leverage the normalization network throughput T to study the impact of TCP Incast. We assume that the maximum value of the throughput T is 1 when there is no packet loss. Furthermore, T can be calculated as $1-P^{'}$ when TCP Incast probability is $P^{'}$. Then the normalized throughput T can be expressed as

$$T = \begin{cases} 1 - P^{'} & n \times k + w \times (n \times k - C \times t) > B \\ 1 & n \times k + w \times (n \times k - C \times t) \leq B \end{cases} \quad (11)$$

Nevertheless, the time interval t between adjacent packets has negative effects on network goodput. For micro-burst, since the whole window of packets are injected into the network back-to-back (i.e., time interval t is 0), it takes one RTT to finish the transmission of this window. For Adaptive Pacing, the completion time of this window is RTT+$w \times t$ when the congestion window is $w+1$. Consequently, the ratio of the completion time without and with Adaptive Pacing is RTT/(RTT+$w \times t$). In short, taking the overhead of time interval into consideration, the normalized network goodput G can be calculated as

$$G = \begin{cases} \frac{(1-P^{'}) \times RTT}{RTT + w \times t} & n \times k + w \times (n \times k - C \times t) > B \\ \frac{RTT}{RTT + w \times t} & n \times k + w \times (n \times k - C \times t) \leq B \end{cases} \quad (12)$$

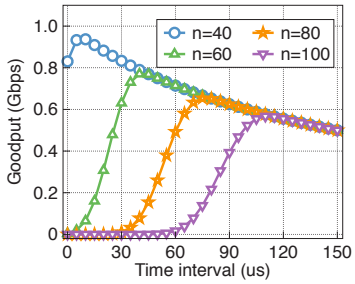


Fig. 3: Goodput with varying time interval t and number of flows n

According to Equation (12), we plot the normalized network goodput with different time interval in Fig.3, where B, C, w and k are 100 packets, 10Gbps, 3 and 1, respectively. From Fig.3, we observe that, when the time interval t is small, the network goodput G increases since the Incast probability becomes small with the increasing of t. However, if the time interval is too large, G is reduced due to the unnecessary waste of bandwidth resource.

Therefore, an appropriate time interval should be adopted to avoid the TCP Incast and achieve high goodput. At the same time, the complicated calculation of solving the optimal value of time interval at the sender should be avoided. In the following, an approximation approach is presented for calculating the optimal time interval.

B. Optimal time interval

According to Equation (12), when $n + w \times (n - C \times k \times \text{RTT}/(w+1)) \leq B$, the network goodput becomes smaller with the increasing of t. However, it is analytically hard to verify whether the derivative of goodput G has a monotonic trend. In order to simplify derivation process, we utilize symbol $\bar{w}$ to denote the average congestion window of all flows in Equation (12). Then the numerical approximation of network goodput $\bar{G}$ can be calculated as

$$\bar{G} = \begin{cases} \frac{(1-P^{'}) \times RTT}{RTT + \bar{w} \times t} & n \times k + \bar{w} \times (n \times k - C \times t) > B \\ \frac{RTT}{RTT + \bar{w} \times t} & n \times k + \bar{w} \times (n \times k - C \times t) \leq B \end{cases} \quad (13)$$

We get the derivative of $\bar{G}$ with respect to t as

$$\frac{d\bar{G}}{dt} = \frac{\bar{w} \times (1 - p_l^{\bar{w}+1})^{nk}}{RTT + \bar{w} \times t} \left(\frac{C \times RTT \times p_l^{\bar{w}}}{1 - p_l^{\bar{w}+1}} - \frac{RTT}{RTT + \bar{w} \times t}\right), \quad (14)$$

where $p_l = 1 - (\bar{w} \times t \times C + B)/(n \times k \times (\bar{w}+1))$. According to Equation (14), we find that its value is larger than 0. In other words, when $n \times k + w \times (n \times k - C \times t) > B$, the network goodput becomes larger with the increase of t.

As analyzed above, we can obtain the approximation of optimal time interval t by assigning it with a boundary value as

$$t = \frac{n \times k \times w - B}{C \times (w-1)}. \quad (15)$$

Since the window of packets should be sent out by the sender within one RTT, the time interval is not larger than RTT/w. Besides, no matter whether the network congestion occurs or not, the time interval is not less than 0. In short, we obtain the time interval as

$$t = max\{0, min\{\frac{RTT}{w}, \frac{n \times k \times w - B}{C \times (w-1)}\}\}. \quad (16)$$

To prevent the synchronization of window, it is necessary for Adaptive Pacing to add or subtract a random interval based on the optimal time interval. In our design, the randomizing time interval is set to $t \times (1 + x)$, where x follows the uniform distribution on [-1,1]. This approach can effectively break synchronization among flows. Meanwhile, it ensures that the whole window of packets are injected into the network in one RTT. Besides, some researchers have proposed several approaches, such as Bitmap Algorithms [19] and Packet Labelling [20], to effectively estimate the number of flows on the routers or switches. Like the previous scenarios(i.e., [18], [21]), we obtain the number of flows by counting the number of the TCP SYN/FIN packets.

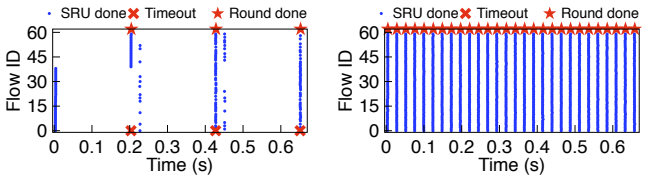
IV. Simulation Evaluation

In this section, we use network simulator NS2 to verify Adaptive Pacing's broad applicability and effectiveness. Considering that TCP NewReno has been integrated into common operating systems, while DCTCP, D^{2}TCP and L^{2}DCT are tailored for data center networks, we evaluate the performance of these protocols with Adaptive Pacing enabled or disabled. For convenience, we use symbols NewReno$_{AP}$, DCTCP$_{AP}$, D^{2}TCP$_{AP}$ and L^{2}DCT$_{AP}$ to denote TCP NewReno, DCTCP, D^{2}TCP and L^{2}DCT with AP enabled, respectively.

In the simulation test, multiple servers are connected to a single ToR switch via 10Gbps link. One of them plays part of a receiver and the others act as senders. The switch buffer can accommodate 100 packets and its marking threshold is set to 20 for ECN-based transports (i.e., DCTCP, D^{2}TCP and L^{2}DCT). The RTT is 300μs. The default RTO_{min} timer value is set to 200ms. The packet size is fixed at 1.5KB and the size of SRU is 45KB. For D^{2}TCP, the deadline imminence factor d is between 0.5 and 2.0. For L^{2}DCT, every flows change its weight from 2.5 to final 0.125 along with data transmission.

A. Basic performance

In this scenario, a receiver sends requests to 60 servers (i.e., senders) to fetch their own data. Then, all senders transmit their SRUs to the receiver. Fig.4 shows the distribution of moments that each SRU are received successfully.



(a) DCTCP w/o AP (b) DCTCP w/ AP

Fig. 4: The distribution of moments when server request unit is completely transmitted

From Fig.4 (a) and (b), we observe that Adaptive Pacing expedites the whole completion time for DCTCP since none of flows experiences timeout. Without Adaptive Pacing, some flows still suffer from timeout, causing goodput collapse. As a result, DCTCP$_{AP}$'s whole transfer time has 7x reduction compared with DCTCP.

B. Performance measures in fat-tree topology

In this scenario, we evaluate the overall performance of AP in different scale fat-tree networks. Here, we select Average Flow Completion Time (AFCT) as our evaluation metrics [22].

To comprehensively understanding the performance of AP, we gradually expand the scale of the network by increasing the pod amount from 4 to 12 at the increasing intervals of 2. This implies that the amount of servers increases from 16 to 432. The link bandwidths between servers and switches is set to 10Gbps. Core switches, aggregation switches and edge switches can accommodate 400, 200 and 100 packets, respectively. The marking threshold of all switches is set to 65. In this experiment, servers in the same pod randomly select a server as their receiver. Note that each server establishes 10 connections with its receiver and a 64KB data block is transmitted by each server to the receiver. The experimental result is illustrated in Fig.5.

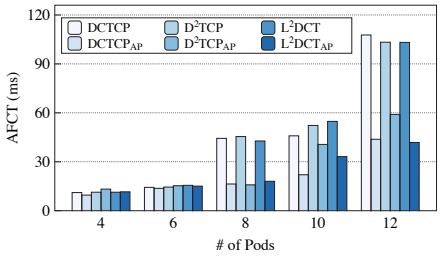
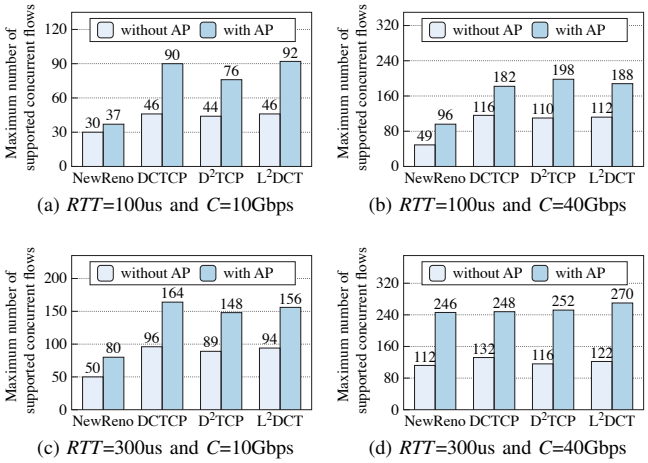


Fig. 5: AFCT with varying number of pods

From Fig.5, we observe that AFCTs of three protocols with AP are basically the same with their original protocols when the number of pods is less than 8. However, the network congestion becomes heavier with the increase of pod amount. As a result, original protocols get large AFCTs due to lots of timeouts. Fortunately, with the aid of AP, the three protocols' AFCTs are remarkably enhanced. This is because AP eliminates the traffic burst so that timeout does not happen frequently.

C. Maximum number of supported flows

In this part, we explore the maximum number of supported flows by varying the number of simultaneous senders. In our experiment, each sender transmits 10 packets and we measure the maximum the number of flows in 10G and 40G network with 100us and 300us RTT. As illustrated in Fig.6, with the help of AP, the maximum number of supported flows can be increased by more than 2x on average.



(a) *RTT*=100us and *C*=10Gbps (b) *RTT*=100us and *C*=40Gbps

(c) *RTT*=300us and *C*=10Gbps (d) *RTT*=300us and *C*=40Gbps

Fig. 6: Maximum number of supported flows

V. Related Works

To date, many researchers have proposed various solutions to solve TCP Incast problem. However, while these solutions can be effective, most of them do not consider the micro-burst and fail to cope with highly concurrent applications. Then the most relevant works are introduced as follows.

Adjusting System Parameters: The literature [12] modifies the default value of RTO_{min} from 200ms to 200us, decreasing the link idle time caused by timeouts. Although this solution can significantly improve the goodput, it does not reduce the number of timeout retransmission. Besides, some researcher try to void TCP Incast problem by changing block size, enlarging the size of switch buffer, and shrinking Maximum Transmission Unit (MTU) [23].

Designing New Transmission Protocols: According to the fraction of marked packets, DCTCP [8] adjusts the congestion window size and achieves excellent performance, such as low queueing delay and high throughput. D^{2}TCP [6] and L^{2}DCT [24] are based on DCTCP, but they can not deal with highly concurrent applications. ICTCP [9] dynamically adjusts the receive window according to the available bandwidth. Similarly, PAC [13] regulates the sending rate of the senders by controlling the ACKs at the receiver. Both ICTCP and PAC are implemented on the receiver side, but they cannot avoid the micro-burst.

Solving TCP Incast at Other Layers: PLATO [10] adopts a packet labeling scheme to prevent labelled packets being dropped. However, it requires modification on the switch. Furthermore, some researchers employ coding-based approaches to prevent TCP Incast [25], [26]. However, these approaches inevitably transmit lots of redundant packets, which reduces bandwidth utilization.

Compared with the enhanced TCP protocols focusing on the congestion window adjustment, Adaptive Pacing deals with the TCP Incast problem by adaptively adjusting time interval according to the flow concurrency. Adaptive Pacing can be directly integrated into the state-of-the-art TCP protocols designed for data centers. Moreover, since Adaptive Pacing only modifies TCP stack at the sender-side, it is easily deployed.

VI. Conclusion

In this paper, a general supporting scheme, called Adaptive Pacing, is proposed for data center networks to mitigates TCP Incast problem. According to the number of concurrent flows, Adaptive Pacing dynamically adjusts the time interval to prevents packet loss in micro-burst. Our design just needs to modify the TCP stack of the sender while it can keep compatibility on existing transport layer protocols without any modifications on switch. Furthermore, the key feature of Adaptive Pacing is its its broad applicability. In other words, AP can be directly integrated into existing transport protocols and obtains great performance improvement.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (61572530, 61502539, 61402541, 61462007 and 61420106009) and the Next Generation Internet Innovation Foundation (Grant No. NGII201601130).

References

[1] B. Palanisamy, A. Singh, and L. Liu, Cost-effective resource provisioning for mapreduce in a cloud, IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 5, pp. 1265-1279, 2015.
[2] W. Bai, L. Chen, K. Chen, and H. Wu, Enabling ECN in Multi-Service Multi-Queue Data Centers, in Proc. USENIX NSDI, 2016.
[3] D. Shan, W. Jiang, and F. Ren, Absorbing Micro-burst Traffic by Enhancing Dynamic Threshold Policy of Data Center Switches, in Proc. IEEE INFOCOM, 2015.
[4] F. Liu, J. Guo, and X. Huang, eBA: Efficient Bandwidth Guarantee Under Traffic Variability in Datacenters, IEEE/ACM Transactions on Networking, vol. pp, no. 99, pp. 1-14, 2016.
[5] T. Benson, A. Akella, D. A. Maltz, Network Traffic Characteristics of Data Centers in the Wild, ACM IMC, 2003.
[6] B. Vamanan, J. Hasan, and T. Vijaykumar, Deadline-Aware Datacenter TCP (D^2 TCP), in Proc. ACM SIGCOMM, 2012.
[7] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, Workload Analysis of a Large-Scale Key-Value Store, in Proc. ACM SIGMETRICS, 2012.
[8] 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.
[9] H. Wu, Z. Feng, C. Guo, and Y. Zhang, ICTCP: Incast Congestion Control for TCP in Data-Center Networks, IEEE/ACM Transactions on Networking, vol. 21, no. 2, pp. 345-358, 2013.
[10] S. Shukla, S. Chan, A. S. W. Tam, A. Gupta, Y. Xu, and H. J. Chao, TCP PLATO: Packet Labelling to Alleviate Time-out, IEEE Journal on Selected Areas in Communications, vol. 32, no.1, pp. 65-76, 2014.
[11] Y. Yu, and C. Qian, Space shuffle: A scalable, flexible, and high-bandwidth data center network, in Proc. IEEE ICNP, 2014.
[12] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, Safe and Effective Finegrained TCP Retransmissions for Datacenter Communication, in Proc. ACM SIGCOMM, 2009.
[13] W. Bai, K. Chen, H. Wu, W. Lan, and Y. Zhao, PAC: Taming Tcp Incast Congestion Using Proactive ACK Control, in Proc. IEEE ICNP, 2014.
[14] L. Zhang, S. Shenker, and D. Clark, Observations on The Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic, in Proc. ACM SIGCOMM, 1991.
[15] W. Chen, F. Ren, and J. Xie, Comprehensive Understanding of TCP Incast Problem, in Proc. IEEE INFOCOM, 2015.
[16] A. Singh, J. Ong, and A. Agarwal, Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Networkin, in Proc. Special Interest Group on Data Communication, 2015.
[17] Judd G. Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter, in Proc. USENIX NSDI, 2015.
[18] J. Huang, Y. Huang, J. Wang, and T. He, Packet Slicing for Highly Concurrent TCPs in Data Center Networks with COTS Switches, in Proc. IEEE ICNP, 2015.
[19] C. Estan, G. Varghese, and M. Fisk, Bitmap Algorithms for Counting Active Flows on High Speed Links, in Proc. ACM IMC, 2003.
[20] J. Zhang, F. Ren, R. Shu, P. Cheng. TFC: token flow control in data center networks, in Proc. ACM EuroSys, 2016.
[21] C. Wilson, H. Ballani, T. Karagiannis, A. Rowstron. Better never than late: Meeting deadlines in datacenter networks, in Proc. ACM SIGCOMM, 2011.
[22] T. Zhang, F. Ren, R. Shu, Backlog-Aware SRPT Flow Scheduling in Data Center Networks, in Proc. IEEE ICDCS, 2016.
[23] P. Zhang, H. Wang, and S. Cheng. Shrinking MTU to Mitigate TCP Incast Throughput Collapse in Data Center Networks. In Third International Conference on Communication and Mobile Computing, 2011
[24] A. Munir, I. Qazi, Z. Uzmi, A. Mushtaq, S. Ismail, M. Iqbal, B. Khan. Minimizing flow completion times in data centers, in Proc. IEEE INFOCOM, 2013.
[25] C. Jiang, D. Li, and M. Xu, LTTP: an LT-code Based Transport Protocol for Many-to-One Communication in Data Centers, IEEE Journal on Selected Areas in Communications, vol. 32, no. 1, pp. 52-64, 2014.
[26] J. Sun, Y. Zhang, D. Tang, S. Zhang, Z. Xu, J. Ge. Improving TCP performance in data center networks with Adaptive Complementary Coding, in Proc. IEEE LCN, 2015.